
# NAVAL POSTGRADUATE SCHOOL

**MONTEREY, CALIFORNIA**

# THESIS

**EFFECTIVE USE OF JAVA DATA OBJECTS IN DEVELOPING DATABASE APPLICATIONS. ADVANTAGES AND DISADVANTAGES**

Paschalis Zilidis

June 2004

| | |
|---|---|
| Thesis Advisor: | Thomas Otani |
| Second Reader: | Arijit Das |

**Approved for public release; distribution is unlimited**

THIS PAGE INTENTIONALLY LEFT BLANK

| REPORT DOCUMENTATION PAGE | | | Form Approved OMB No. 0704-0188 |
|---|---|---|---|
| Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503. | | | |
| **1. AGENCY USE ONLY** *(Leave blank)* | **2. REPORT DATE** June 2004 | **3. REPORT TYPE AND DATES COVERED** Master's Thesis | |
| **4. TITLE AND SUBTITLE**: Effective Use of Java Data Objects in Developing Database Applications. Advantages and Disadvantages | | | **5. FUNDING NUMBERS** |
| **6. AUTHOR(S) Paschalis ZILIDIS** | | | |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Naval Postgraduate School Monterey, CA 93943-5000 | | | **8. PERFORMING ORGANIZATION REPORT NUMBER** |
| **9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)** N/A | | | **10. SPONSORING/MONITORING AGENCY REPORT NUMBER** |
| **11. SUPPLEMENTARY NOTES** The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. | | | |
| **12a. DISTRIBUTION / AVAILABILITY STATEMENT** Approved for public release; distribution is unlimited | | | **12b. DISTRIBUTION CODE** |


**13. ABSTRACT (maximum 200 words)**

Currently, the most common approach in developing database applications is to use an object-oriented language for the frontend module and a relational database for the backend datastore. The major disadvantage of this approach is the well-known "impedance mismatch" in which some form of mapping is required to connect the objects in the frontend and the relational tuples in the backend.

Java Data Objects (JDO) technology is recently proposed Java API that eliminates the impedance mismatch. By using JDO API, the programmers deal strictly with objects. JDO hides the details of the backend datastore by providing the object-oriented view of the datastore. JDO automatically handles the mapping between the objects and the underlying data in the relational database, which is hidden from the programmer.

This thesis investigates the effectiveness of JDO. Part of the analysis will develop a database application using JDO. Although JDO provides the benefits of object-orientation in design and implementation of the databases, it is not immune from problems and limitations. The thesis will also analyze the advantages and disadvantages of using JDO and discuss the areas requiring improvements in future releases.


| **14. SUBJECT TERMS** Datastore, Java Data Objects, JDO, API, Java | | | **15. NUMBER OF PAGES** 285 |
|---|---|---|---|
| | | | **16. PRICE CODE** |
| **17. SECURITY CLASSIFICATION OF REPORT** Unclassified | **18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified | **19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified | **20. LIMITATION OF ABSTRACT** UL |


NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

**Approved for public release; distribution is unlimited**

# EFFECTIVE USE OF JAVA DATA OBJECTS IN DEVELOPING DATABASE APPLICATIONS. ADVANTAGES AND DISADVANTAGES

Paschalis Zilidis
Major, Hellenic Air Force
B.S., Hellenic Air Force Academy, 1988

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN COMPUTER SCIENCE**

from the

**NAVAL POSTGRADUATE SCHOOL**
**June 2004**

Author: Paschalis Zilidis

Approved by: Thomas Otani
Thesis Advisor

Arijit Das
Second Reader

Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

# ABSTRACT

Currently, the most common approach in developing database applications is to use an object-oriented language for the frontend module and a relational database for the backend datastore. The major disadvantage of this approach is the well-known "impedance mismatch" in which some form of mapping is required to connect the objects in the frontend and the relational tuples in the backend.

Java Data Objects (JDO) technology is recently proposed Java API that eliminates the impedance mismatch. By using JDO API, the programmers deal strictly with objects. JDO hides the details of the backend datastore by providing the object-oriented view of the datastore. JDO automatically handles the mapping between the objects and the underlying data in the relational database, which is hidden from the programmer.

This thesis investigates the effectiveness of JDO. Part of the analysis will develop a database application using JDO. Although JDO provides the benefits of object-orientation in design and implementation of the databases, it is not immune from problems and limitations. The thesis will also analyze the advantages and disadvantages of using JDO and discuss the areas requiring improvements in future releases.

THIS PAGE INTENTIONALLY LEFT BLANK

# TABLE OF CONTENTS


I. INTRODUCTION....1

II. DEVELOPMENT TOOLS AND TECHNIQUES....5
- A. LIDO....5
- B. MYSQL....7
- C. JDBC CONNECTOR....8
- D. THE JFREECHART....8
- E. THE PROPERTIES FILE....9

III. PROJECT DESCRIPTION....11
- A. THE TRAINING-SQUADRON APPLICATION....11
- B. UML....12
- C. THE INTERFACE MYINTERFACE....13
  - 1. The Method: getFieldLengths()....13
  - 2. The Method: getFieldByName ()....14
  - 3. The Method: getField....15
  - 4. The Method: setFieldByName ()....15
  - 5. The Method: setFieldValue ()....16
  - 6. The Method: getTheTypeOfTheCollection ()....18
- D. THE FORM InstructorsForm....20
  - 1. The JDOImplHelper....20
  - 2. Collection Fields....23
- E. GETTING THE VALUES OF THE RECORD....24
- F. SETTING VALUES TO THE RECORD....26
  - 1. Creating Subforms....27

IV. JDO DISADVANTAGES....29
- A. MANY TO MANY RELATIONSHIPS....29
  - 1. The Redundancy of Tables....29
  - 2. Inverse Relationship....31
- B. FIELD REDUNDANCY....35
- C. JDO GROUP BY QUERIES....40
  - 1. Performance Imitating a GROUP BY Query....40
  - 2. Performance for Simple SELECT Queries....42
- D. NO SUPPORT OF ALL THE DATA STRUCTURE TYPES....49
- E. NO FREE SOURCE CODE....52
- F. WHY XML?....52
- G. METADATA DEFICIENCY....54

V. JDO ADVANTAGES....57
- A. OBJECT ORIENTED VIEW OF THE DATABASE....57
- B. AUTOMATE PERSISTENT....60
- C. EASE OF IMPLEMENTATION....61

D. JDOQL ..... 64
VI. CONCLUSION ..... 69
A. ADVANTAGES ..... 69
B. DISADVANTAGES ..... 70
APPENDIX. SOURCE CODE ..... 75
A. PACKAGE COMPANY ..... 75
1. The Class: Aircraft.java ..... 75
2. The Class : AircraftType.java ..... 78
3. The Class: BaseCategory.java ..... 79
4. The Class: BaseStadio.java ..... 82
5. The Class: Categories.java ..... 85
6. The Class: Exercise.java ..... 90
7. The Class: Flights.java ..... 94
8. The Class: GroundCourse.java ..... 99
9. The Class: Instructors.java ..... 103
10. The Interface: MyInterface.java ..... 110
11. The Class: Rank.java ..... 111
12. The Class: Schedule.java ..... 114
13. The Class: Series.java ..... 118
14. The Class: SpecialType.java ..... 123
15. The Class: Squadron.java ..... 126
16. The Class: Stadio.java ..... 130
17. The Class: Students.java ..... 135
B. PACKAGE TEST ..... 142
1. The Class: InstructorsForm.java ..... 142
2. The Class: MyBarChart.java ..... 189
3. The Class: Populate.java ..... 195
4. The Class: PureSQL.java ..... 198
5. The Class: QueryForm.java ..... 201
6. The Class: ReportsForm.java ..... 215
7. The Class: SquadronsForm.java ..... 243
C. METADATA JDO FILE ..... 264
LIST OF REFERENCES ..... 267
INITIAL DISTRIBUTION LIST ..... 269

# LIST OF FIGURES


Figure 1. The Properties File for the Training-Squadron Application. ....9
Figure 2. UML for the Training-Squadron Application. ....12
Figure 3. JDO Performance Imitating a GROUP BY Query. ....40
Figure 4. JDO Performance Imitating a GROUP BY Query (Second Execution)....41
Figure 5. LIDO Message for 'sql' tag. ....42
Figure 6. JDO Performance Using 'sql' Tag and retrieveAll() Operation. ....43
Figure 7. JDO Performance Using 'sql' Tag without retrieveAll() Operation. ....44
Figure 8. JDO Performance Using 'sql' Tag without retrieveAll() but with Iteration through the Result Set. ....44
Figure 9. JDO Performance without 'sql' Tag with retrieveAll() Operation and with Iteration through the Result Set. ....45
Figure 10. JDO Performance without 'sql' Tag with retrieveAll() Operation but without Iteration through the Result Set. ....46
Figure 11. JDO Performance with 'sql' Tag with retrieveAll() operation but without Iteration through the Result Set. ....46
Figure 12. JDO Performance without retrieveAll(), and with Iteration. ....47
Figure 13. JDO Performance without retrieveAll(), and without Iteration. ....48
Figure 14. JDO Performance without retrieveAll(), and without Iteration Using 'sql' Tag. ....48

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF TABLES

Table 1. Features of LiDO's Community Edition.....6

THIS PAGE INTENTIONALLY LEFT BLANK

# ACKNOWLEDGMENTS

This thesis is dedicated to my loving family, my beloved and wonderful wife, Petroula, for her unconditional support and patience during my studies, and my wonderful and lovely daughter Despoina whose birth nine months ago I was not able to attend since I was here at the Naval Postgraduate School.

I would like to also thank my thesis advisor Thomas Otani for his outstanding help support and guidance for the implementation of the thesis, and Arijit Das for his insightful commitment to this work. In addition, I would like to thank Nancy Sharrock for her valuable help in editing and formatting this thesis.

Finally I would like to thank my father Christos and my mother Despoina, who in sacrificing much of their personal life in trying to provide me everything that I needed and taught me through their valuable experiences and undisputable values about life.

THIS PAGE INTENTIONALLY LEFT BLANK

# I. INTRODUCTION

The Java Data Objects (JDO) API is a standard interface-based Java model abstraction of persistence. It was developed in 2000 as Java Specification Request 12 under the auspices of the Java Community Process.

The specification, reference implementation, and technology compatibility kit were approved by the Java Community Process (JCP) in March 2002 and published in April 2002. Alternatives to JDO include direct file I/O, serialization, JDBC, and Enterprise Java Beans (EJB) Bean Managed Persistence (BMP) or Container Managed Persistence (CMP) Entity Beans. Even though the program was not indented to replace the previous solutions as the JDO API, the product appeared to be very promising that resulted in providing valuable solutions to the programmer and the implementation of the program.

According to the specifications, the programmer can write code in the Java programming language that transparently accesses the underlying data store, without using database-specific code. Thus, the Application programmers can use JDO to store their Java object instances directly into the persistent datastore (database).

JDO provides the following benefits:

- ***Portability:*** Applications written with the JDO API can be run on multiple implementations without recompiling or changing source code.
- ***Database independence:*** Applications written with the JDO API are independent of the underlying database.
- ***Ease of use:*** Application programmers can focus on their domain object model and leave the details of persistence (field-by-field storage of objects) to the JDO implementation.
- ***High performance:*** Application programmers delegate the details of persistence to the JDO implementation, which can optimize data access patterns for optimal performance.
- ***Integration with EJB:*** Applications can take advantage of EJB features such as remote message processing, automatic distributed transaction coordination, and security, using the same domain object models throughout the enterprise."[1]

[1] See http://java.sun.com/products/jdo/overview.html, accessed March 2004.

The thesis explores JDO capabilities and deficiencies and evaluates JDO performance against traditional ways of accessing and creating a database.

As part of the study, the thesis develops an application which involves the creation of a database, and a suitable GUI for accessing and navigating the data along with the creation of queries. The entire application will be constructed using only JDO interfaces and pure Java classes. No direct interaction with DBMS is made.

Finally, in an effort to asses the performance of JDO queries, the thesis involves the creation of pure SQL commands to retrieve the same set of data and analyze their retrieval performance.

The organization of the thesis follows.

Chapter I is the introduction which presents the JDO specifications and the purpose of the thesis. Chapter II includes the presentation and the installation of the development tools to use for the implementation of the application.

Chapter III introduces the specification of the database application- form currently on the *Training Squadron* application, with a short description of the tables and the java classes that represent these tables. The Training-Squadron application created is used to automate the data of flights and the training program of student pilots in a Training Squadron. Thus, this chapter describes the relationship found in this specific application. There is also a description of a model that can be created to achieve access to different java classes, which represent entities of the model for this thesis and the queries used to handle the data retrieval.

Chapter IV discusses the weakness of JDO. Thus, this chapter provides detailed disadvantages about using JDO in creating database applications. The chapter uses the data from the previous chapter and identifies when the implementation was difficult or cumbersome.

Chapter V presents the advantages of JDO. In contrast to the previous chapter, it will present the strengths of JDO and the advantages of using it in creating a database application. Specifically, the chapter uses the experience obtained in using JDO when developing the case study application to formulate the general discussion. In addition,

contrary to the previous chapter, there is a description of the good points encountered during the implementation of the application and assesses how they facilitate the implementation of an application.

The final chapter is the conclusion providing the overall evaluation of JDO and some assessment for its future.

THIS PAGE INTENTIONALLY LEFT BLANK

# II. DEVELOPMENT TOOLS AND TECHNIQUES

## A. LIDO

JDO is a specification from Sun Microsystems that helps Java programmers to persist objects. In other words, to read objects from or write objects to a data source. JDO greatly simplifies persistence compared to other solutions such as JDBC, JCA or EJBs.

The Java Specification Request that lead to the creation of JDO can be found at: http://www.jcp.org/jsr/detail/12.jsp.

LiDO is a JDO implementation that support different types of data sources including many relational databases. Libelis created LIDO. It is also an implementation of the official JDO 1.0 specification.

The project uses LIDO version 1.4.0. JDO provides the basic APIs for managing persistent java objects in a transactional manner. LiDO uses the functionality of JDO APIs. There are currently three editions of LiDO:

- Community Edition
- Standard Edition
- Professional Edition

The LiDO Community Edition is free of charge. However, it requires a license that must be obtained from Libelis. The thesis application uses the Community Edition, and available for downloading from http://www.libelis.com/.

The features of LiDO's community edition appear in the next table:

Table 1. Features of LiDO's Community Edition.

| | Community Edition |
|---|---|
| Open Source RDBMS | **Yes** |
| Mapping of existing database model (Application Identity) | **Yes** |
| Connection pooling of client/server connections (at least 2 connections) | **Yes** |
| LIBeLIS FileDB (File system database) | No |
| Commercial RDBMS | No |
| Versant ODBMS | No |
| JCA Compliance for J2EE integration | No |
| JSP tag library | No |
| GUI mapping tool (LiDO Project Manager) | No |
| NAViLIS (e-Business Browser) | No |

In order to use LiDO effectively, the following are also required:

- Java SDK 1.2.x or higher http://java.sun.com/j2se/
- If Java SDK is version 1.2: JCE http://java.sun.com/products/jce/

The following are used for the development of a database:

- RDBMS: The database and JDBC drivers
- ODBMS: Versant JVI 2.4.x or enJin 2.x
- LIBeLIS FileDB: a standard or professional LiDO license

RDBMS, and specifically, MySQL is used for the development of the application of this thesis.

The installation file, called LiDO_<Edition>_x.y.z.jar, is used to install LiDO where the <Edition> is either "Community", "Standard or "Professional" and "x.y.z" is the release number. The installation file is an "executable jar".[2]

The following script sets the LiDO environment:

- <LIDO_HOME>/bin/lidoEnv.bat (Windows)

[2] For information about LIDO's installation, see the LIDO user's manual.

This script adds or modifies the following environment variables:

- **LIDO_HOME**: set to the root installation directory of LiDO
- **PATH:** <LIDO_HOME>/bin is pretended.
- **CLASSPATH:** <LIDO_HOME>/bin is pretended and all jar files in <LIDO_HOME>/lib are appended.

LiDO currently supports a wide area of RDBMS. As states previously, the thesis uses MySQL, which the Community edition supports. Even though LiDO relational uses JDBC, each RDBMS requires a custom dictionary to support it since implementation details are abstracted by JDBC. The dictionary tells LiDO how to interact with a given database.

The file <LIDO_HOME>/bin/dictionary.properties maps JDBC driver names to database engines. When creating a schema or opening a connection with PersistenceManagerFactory, this file is read to discover which database engine to use.[3]

The <LIDO_HOME>/bin/dictionary.properties file must be in the Java CLASSPATH. This can be done by running the <LIDO_HOME>/bin/lidoEnv.bat script.

Before using a JDBC driver, verify that it is in <LIDO_HOME>/bin/dictionary.properties. If not present, it must be added by associating it with the appropriate dictionary.

## B. MYSQL

MySQL edition 4.0 is used for the development of the Training-Squadron application possessing the following specifications

| | | |
|---|---|---|
| Windows 95/98/NT/2000/XP/2003 (x86) | 4.0.18 | 23.1M |

It is available for download from http://www.mysql.com/downloads/mysql-4.0.html, and is installed in c:\mysql.

The *mysql* program is used for the creation of the database as follows:

[3] For database support, see the LiDO user's manual.

**c:\ mysql\bin> mysql**

**mysql> create database <database_name>**

For example,

**mysql> create database *pilotdb***

***Pilotdb* will be the name of the database for holding the data of our *Training-Squadron* application**

Additionally, it is necessary to change the *javax.jdo.option.connectionURL* property as follows:

**javax.jdo.option.connectionURL=jdbc:mysql://localhost/pilotdb**

Thus, it is now possible to use the database pilotdb and let LIDO's JDO implementation create the schema.

## C. JDBC CONNECTOR

MySQL Connector/J 3.0, the production release, was used. It is also available for download from http://www.mysql.com/downloads/mysql-4.0.html. It is also necessary to to change the Lido_mysql.properties file and assign the property javax.jdo.option.connectionDriverName as follows:

**javax.jdo.option.connectionDriverName=com.mysql.jdbc.Driver**

The driver comes with a jar file.

In order for the program to be able to find the com.mysql.jdbc. driver, it is necessary to install the jar file of *mysql* JDBC connector (*mysql-connector-java-3.0.9-stable-bin.jar*) to the <*LIDO_HOME*>*lib* directory. Otherwise, it can be added to its parent directory location at CLASSPATH

## D. THE JFREECHART

JFreeChart was used to create the graphs that show the performance of JDO and this of JDBC in similar queries. JFreeChart is available for download from http://www.jfree.org/jfreechart/index.html. JFreeChart contains two jar files: the *jfreechart-0.9.16.jar* and the *jcommon-0.9.1.jar*, which is included in the lib directory of JFreeChart.

In order for the program to be able to find the classes for the graph, it is necessary to install the ***jfreechart-0.9.16.jar*** **and the** ***jcommon-0.9.1.jar*** to the *<LIDO_HOME>\lib* directory. Otherwise, it can be added to its parent directory location at CLASSPATH

## E. THE PROPERTIES FILE

The final properties file for *mysql* driver is as follows:

```
# lido.properties file

# jdo standard properties

javax.jdo.option.connectionURL=jdbc:mysql://localhost/pilotdb
javax.jdo.option.connectionDriverName=com.mysql.jdbc.Driver
#javax.jdo.option.connectionUserName=lido
#javax.jdo.option.connectionPassword=lido

javax.jdo.option.msWait=5
javax.jdo.option.multithreaded=false
javax.jdo.option.optimistic=false
javax.jdo.option.retainValues=false
javax.jdo.option.restoreValues=true
javax.jdo.option.nontransactionalRead=true
javax.jdo.option.nontransactionalWrite=false
javax.jdo.option.ignoreCache=false

# set to PM, CACHE, or SQL to have some traces
# ex:
#lido.trace=SQL,DUMP,CACHE

#  set the Statement pool size
lido.sql.poolsize=10
lido.cache.entry-type=weak

# set the max batched statement
# 0: no batch
# default is 20


lido.sql.maxbatch=30
lido.objectpool=90

# set for PersistenceManagerFactory pool limits
lido.minPool=1
lido.maxPool=10

jdo.metadata=metadata.jdo
```

Figure 1. The Properties File for the Training-Squadron Application.

THIS PAGE INTENTIONALLY LEFT BLANK

# III. PROJECT DESCRIPTION

## A. THE TRAINING-SQUADRON APPLICATION

The application developed in this project is a *Training-Squadron* application. The basic purpose of a training squadron is to provide a flight education to the air cadets. The following are the requirements of the system:

- A Training squadron should have a set of Aircraft that it will use for the implementation of the daily flights
- A Training squadron should have a number of Instructors pilots that will teach the flight techniques to the Students
- A Training squadron should have a number of Students that will undergo the education process
- Every student follows an educated flight program.
- The educated flight program is specific for a class of students. However, it is possible to change it for a new class as the aviation evolutes or new weapon products become known.
- The flight program consists of flight courses, which have some certain limitations and characteristics.
- The flight courses are part of the Categories of flights, which share certain characteristics.
- The categories also are part of a bigger subdivision, the *Stadio* which also shares certain characteristics and limitations. A student moves from one *Stadio* to another as the student's experience increases and the limitations and demands also change.
- Above all is the Series, which represents the total program that a class (a series) of students will follow.
- Series except the students and the flight program have a simulation flight program that must be implemented by the students.
- Additionally, a ground course program that students must attend for their education plan also exists.
- Students could have a set of Instructors that help in the implementation of the flight program. Student can fly only with these instructors during their training.
- The same relation works in reverse for the Instructors.
- The relational schema for the above application demands all types of relationships from one to one, one to many, and many to many.

The project will implement the application using JDO and pure java classes completely in order to access the ability of JDO to handle databases.

**B. UML**

The UML of the application appears below.

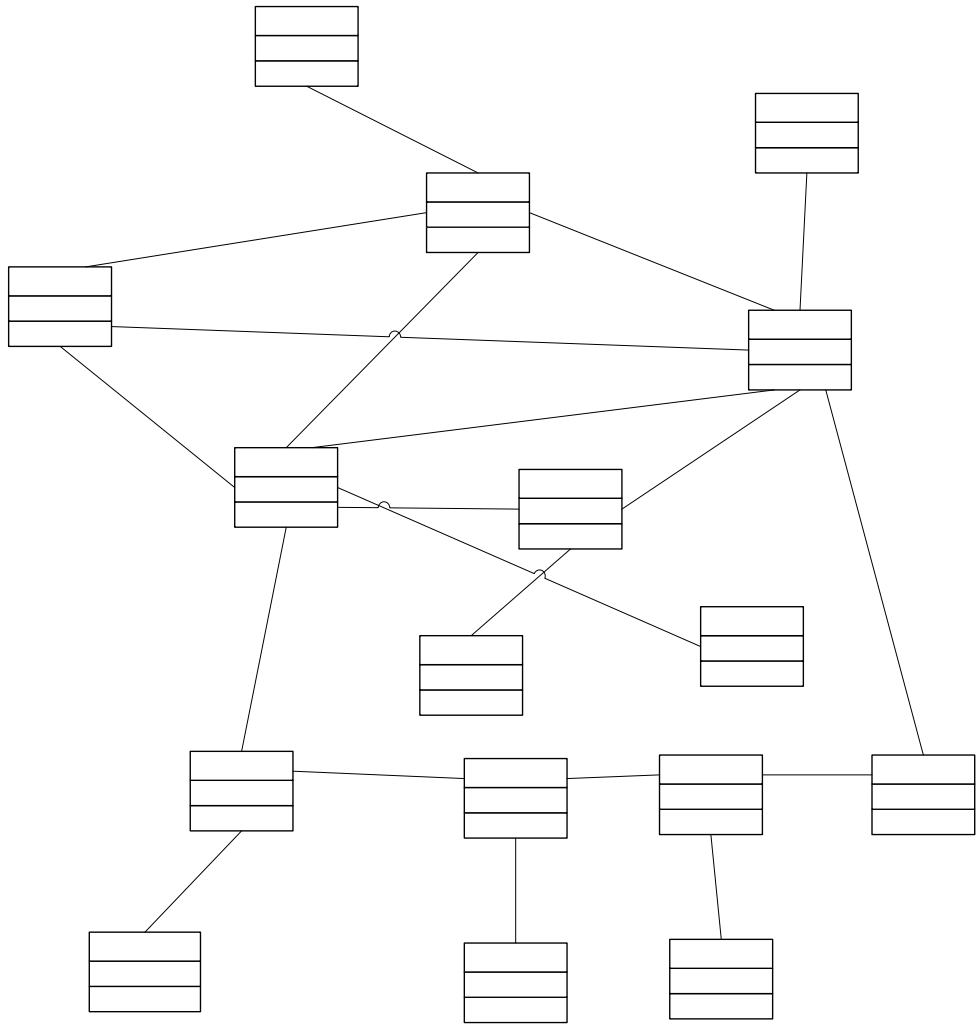


Figure 2. UML for the Training-Squadron Application.

LiDO creates the schema for the application with the following command:

```
java -cp %CLASSPATH% com.libelis.lido.DefineSchema -properties lido_mysql
```

A part of the project includes an effort to develop a form that will show the data of all the tables.

## C. THE INTERFACE MYINTERFACE

The following interface was created to achieve the aforementioned purpose.

```
public interface  MyInterface  {
public int[] getFieldLengths();
public Object getFieldByName(String fieldName);
public Object getField(int i);
//to handle the collections
     public void setFieldByName(String name,Object
     value,boolean flag);
public void setFieldValue(int i,Object value);
public Class getTheTypeOfTheCollection(String
     collectionName);
}
```

Every class handled must implement the previously mentioned interface.

### 1. The Method: getFieldLengths()

*public int[] getFieldLengths*

This returns an integer array containing the length of the fields in number of characters.

For example, the *BaseCategory* class is:

```
public class BaseCategory  {
          private String id;
          private String description;
          ……..
           public int[] getFieldLengths(){
```

```
                    int[] ar={10,20};
                    return ar;
            }
            …….
      }
```

The method returns the length of the two fields of the class.

The application verifies if the number of fields of a particular class is not equal to the number of the elements of the array that the *getFieldLengths()* returns and raises a warning message. In case a contradiction occurs, the application uses a default length for the number of fields.

**2. The Method: getFieldByName ()**

| *public Object getFieldByName(String fieldName)* |
|---|

This returns the value of the field in which the name is equal to the String fieldname.

For example, the BaseStadio class is:

```
public class BaseStadio  {
          private String id;
          private String description;
          …….
public Object getFieldByName(String fieldName){
          if (fieldName.equals("id"))
             return id;
          else if (fieldName.equals( "description"))
             return description;
          return null;
     }
     ……
}
```

### 3. The Method: getField

| *public Object getField(int i);* |
|---|

This returns the value of the field according to the field index ,

For the same example, the method in *BaseStadio* class will provide the following:

```
public class BaseStadio  {
    private String id;
    private String description;
     ……..
     public Object getField(int i){
          switch (i) {
                    case 0:
                         return id;
                    case 1:
                         return description;
               }
               return "";
     }
```

### 4. The Method: setFieldByName ()

| *public void setFieldByName(String name,Object value,boolean flag);* |
|---|

This assigns the value to the field with name "name" and the value "value". The Boolean flag is used for one to many relationships. In this case. the field will be a collection. When the flag is true. the method will add the *Object value* to the collection. If the flag is false, the method will remove the *Object value* from the collection if it is there. For example, in the *Series* class:

```
public class Series  {
     private String id;
     private String description;
     private Collection stadio;
     private LinkedList students;
     private Collection groundcourse;
               ……..
     public void setFieldByName(String name,Object
                                   value,boolean flag){
          if(name.equals("students") ){
               if(flag) addStudents((Students) value);
               else deleteStudents((Students) value);
          }
          if(name.equals("stadio") ){
               if(flag) addStadio((Stadio) value);
               else deleteStadio((Stadio) value);
          }
          if(name.equals("groundcourse") ){
               if(flag)
               addGroundCourse((GroundCourse) value);
               else
               deleteGroundCourse((GroundCourse) value);
          }

     }
```

**5. The Method: setFieldValue ()**

| *public void setFieldValue(int i,Object value);* |
|---|

This assigns a value to a specific field. The method assigns the *Object value* to the field with the index i in the declaration of the class' fields. For example, in the *Categories* class:

```
public class Categories  {
     private Series series;
     private Stadio stadio;
     private BaseCategory catid;
     private String description;
     private LinkedList exercise;
     private String seriesid;
     private String stadioid;
     private String id;
     ……..
     public void setFieldValue(int i,Object value){
          switch (i) {
               case 0:
                    setSeries((Series) value);
                    break;
               case 1:
                    setStadio((Stadio) value);
                    break;
               case 2:
                    setCatId((BaseCategory) value);
                    break;
               case 3:
                    setDescription(value.toString());
                    break;
               case 4:
                     //
                    break;
               }
          }
```

Note: Here the fields *seriesid, stadioid* are not informed by the method.

The reason is that these fields are redundant and were used to make the mapping and assigning the key values. These fields hold the same values as *series* and *stadio*, and are informed inside the methods that inform the *series* and *stadio* fields.

Note: The fields of collection-type as in the *exercise* in this example are not handled by this method, which instead is used the previous value:

*public void setFieldByName(String name,Object value,boolean flag)*

**6. The Method: getTheTypeOfTheCollection ()**

*public Class getTheTypeOfTheCollection(String collectionName);*

This is used for returning the type of objects that a collection-type field contains. Actually, it returns the class of the object contained in the collection field. For example, in the *Series* class:

```
public class Series  {
     private String id;
     private String description;
     private Collection stadio;
     private LinkedList students;
     private Collection groundcourse;
     public Series(){
     }
     ……..
     public Class getTheTypeOfTheCollection(String
                                        collectionName){
          if (collectionName.equals(“students”))
               return Students.class;
          else if (collectionName.equals(“stadio”))
               return Stadio.class;
          else if (collectionName.equals(“groundcourse”))
               return GroundCourse.class;
```

```
            return null;
        }
    }
```

In this manner, it is possible, for example, to know the collection field students contains objects of *Students* class. This is very important for creating a form that could provides the data depiction for every class.

Actually, JDO provides such information when creating the metadata JDO file. For example, for the *Series* class, the metadata is as follows:

```
<class name="Series" identity-type="application">
      <field name="id" primary-key="true"/>
      <field name="stadio" >
          <collection element-type="Stadio">
            <extension vendor-name="libelis"
            key="sql-reverse"
            value="javaField:series"/>
          </collection>
      </field>
      <field name="groundcourse" >
          <collection element-type="GroundCourse"/>
      </field>
      <field name="students" >
          <collection element-type="Students"/>
      </field>
</class>
```

Unfortunately, it was not possible to find a method or an API that returns the metadata to the programmer. Thus, it was necessary to provide the above method along with the others in a separate interface that the classes had to implement.

## D. THE FORM InstructorsForm

```
class InstructorsForm extends JPanel implements ActionListener,
        MouseListener,  ItemListener{
……
}
```

Having the classes implement the above methods included in interface *MyInterface*, it is possible to then use them effectively and along with the advantages that JDO provides, to create an efficient form that will be able to present the data for all of them.

### 1. The JDOImplHelper

An important class for use in the implementation is the JDOImplHelper. JDOImplHelper provides the following methods:

**getFieldFlags**`(java.lang.Class pcClass)`

Get the field flags for a `PersistenceCapable` class.

**getFieldNames**`(java.lang.Class pcClass)`

Get the field names for a `PersistenceCapable` class.

**getFieldTypes**`(java.lang.Class pcClass)`

Get the field types for a `PersistenceCapable` class.

An instance of JDOImplHelper can be taken as follows:

**getInstance**`()`

Get an instance of `JDOImplHelper`.

JDO API states: This class is a helper class for JDO implementations. It contains methods to register metadata for persistence-capable classes and to perform common operations needed by implementations, not by end users.

However, it resulted in being very important for the implementation

Thus, in the *Training –Squadron* application could have an instance:

```
myJDOImplHelper=JDOImplHelper.getInstance();
```

One of the parameters of *InstructorsForm* class is an Object that represents the instance of the class that will display its data from the underlying database.

> The name *InstructorsForm* was given because it was tried first to display the data of the Instructors class. Later, improving the class abilities make it work (display the data) for all the classes, but the initial name is retained.

One of the constructors is as follows:

```
public InstructorsForm (
     PersistenceManager pms,
     LinkedList result,

     Object mainFormObject //represents the object of
                    //the database that we will handle
               ) {
     ……
     }
```

It is then possible to obtain the name of the fields of the class that will its data:

```
String  mainFieldNames=
  myJDOImplHelper.getFieldNames(mainFormObject.getClass());
```

It is also possible to obtain the type of the fields of the class that will display its data:

```
Class[] fieldtypes=
  myJDOImplHelper.getFieldTypes(mainFormObject.getClass());
```

The possible values for fieldtypes are:

- primitive types
- a kind of collection (LinkedList , HashMap etc.)
- a date field
- a class of my classes. (In *Flights* class, for example, the field *student* is a type of *Students* class)

It is then possible to verify the type and introduce the manner in which it will be displayed:

```
if(fieldtypes[i].getName().equals("boolean"))
                mainFelds[i] = new JCheckBox();
```

`mainFields` is an array of components.
`private JComponent[]    mainFelds;`
It will hold all the component objects that will display the data for every field.
Its initial length is assigned using:
`mainFelds = new JComponent [fieldNames.length];`

For the *date- type* fields, the following are used:

```
if(fieldtypes[i].getName().equals("java.util.Date"))
  mainFelds[i] =new JSpinner(new SpinnerDateModel()) ;
```

It is also possible to ascertain if the field represents a class being used. The name of the package for this is:

```
String mypackage=
   mainFormObject.getClass().getPackage().getName() ;
if( fieldtypes[i].getName().indexOf(mypackage+".")>-1)
     mainFelds[i] = new JComboBox(
          getTheDataForComboBoxes(fieldtypes[i]));
     //fieldtypes hold the class of the field
```

Since it is known that the field represents a class of our classes, then it is possible to execute a query using the class of that field as a parameter. The method *getTheDataForComboBoxes(fieldtypes[i])* actually performs exactly this function:

```
public static Collection getTheDataForComboBoxes(
                                        Class dataclass){
                Collection sresult;
                //tx is the transaction
                if(!tx.isActive())  tx.begin();
                     q = pm.newQuery(dataclass);
                     sresult = (Collection) q.execute();
                     tx.commit();
                     return sresult;
     }
```

In this manner, it is feasible to represent, for example, the student field in Flights class as a *comboBox* that will contain the students objects from which it is also possible to choose for editing or inserting new values.

String Fields

For String field, the following is used:

```
mainFelds[i] = new JTextField( fieldLength[i]);
```

where *fieldLength*[i] represents the length of the field as taken using the method int[] *getFieldLengths*() of *MyInterface.*

**2. Collection Fields**

For recognizing and handling the Collection fields, the following are used:

```
byte[] fieldFlags=
  myJDOImplHelper.getFieldFlags(mainFormObject.getClass());
```

The above method returns the flag for every field.

With the following *if* statement, it is possible to ascertain whether a field represents a collection:

```
if (myJDOImplHelper.getFieldFlags(
                    mainFormObject.getClass())[i]==10){
        //the panel that will hold data for the subform
        //we will have one subform for every collection field
        JPanel subpane=createSubFormPanel( i, fieldNames[i]);
        //
}
```

The flag for a collection field is equal to 10, and much experimentation is required before finding the flag values and what they represent. Again, JDO specifications do not provide this information. As the API describes, the JDOImplHelper is for implementations and not for the end user. It is not possible to understand this mystic behavior since this kind of information provides the advantages that an object oriented programming can exploit for the development of an application.

## E. GETTING THE VALUES OF THE RECORD

The *InstructorsForm* has a variable *currentRecord*. The variable is an object and represents the instance of the class currently displayed in the form.

The instances of a class are possible using a query:

```
public static Collection getTheData (Object dataclass){
                Collection sresult;
                //tx is the transaction
                if(!tx.isActive())  tx.begin();
                     q = pm.newQuery(dataclass.getClass());
                     sresult = (Collection) q.execute();
                     tx.commit();
                     return sresult;
}
```

where *dataclass* will be the object passed as parameter to the form and for which the intent is to display the data from the underlying database.

The result of the query can iterate through the result and take every instance. Every time an iteration occurs, the value of the current object is obtained:

```
if(iterator.hasNext()) {
                currentRecord =  iterator.next();
            ……
```

Then, it is possible to obtain the values of the current instance and display them in the form.

```
for (int i = 0; i < mainFelds.length; i++) {
     Class[] rc={int.class};
     Object[] obi={ (new Integer(i)) };
try{
       java.lang.reflect.Method meth=
       currentRecord.getClass().getMethod(
        "getField", rc);
         Object returnValue=
               meth.invoke(currentRecord,obi);
          …….
     }
```

Thus, it is possible to have the value of the field (for every field of the record) by simply passing the *currentRecord* instance as a parameter even without exactly knowing what the current record represents.

It is then possible to inform the form:

```
if(mainFelds[i] instanceof JTextField){
   ((JTextField) mainFelds[i]).setText(
                    returnValue.toString()   );
}
```

```
if(mainFelds[i] instanceof JSpinner){
   ((JSpinner) mainFelds[i]).setValue(  returnValue   );
   ……..
```

As stated previously, the *mainFelds* is a *JComponent*[]that is indented for displaying the data and the same is done for the *comboBoxes* and *checkboxes*.

**F. SETTING VALUES TO THE RECORD**

A similar tactic is followed for setting values to records. Once again, the *currentObject* represents the instance of the class that the program is going to change the values of its field or to create a new instance.

```
for (int i = 0; i < mainFelds.length; i++) {
     Class[] classParameters={
               int.class,(new Object()).getClass()};
     Object[] ob={};
     if(  mainFelds[i] instanceof JTextField  ){
       Object[] obi={(new Integer(i)),
                    ((JTextField)    mainFelds[i]).getText()
                    };
       ob=obi;
     }
```

Next, the values of other *JComponent*s are obtained such as :

*checkboxes*, *ComboBoxes* etc.

Then the values are assigned as:

```
java.lang.reflect.Method meth=
     record.getClass().getMethod("setFieldValue",
                                  classParameters);
meth.invoke(record,ob);
```

Thus, the record is informed without even knowing the class of the record. Note that with JDO implementation and with the java abilities, it is possible to write code that covers many of the cases. Therefore, it is possible to minimize the code and check it more efficiently.

### 1. Creating Subforms

The following function creates the subforms:

```
public void setSubformForTheField(String nameSubformField,
                                  Object subformObject){
for(int i=0;i<mainFieldNames.length;i++)
  if(myJDOImplHelper.getFieldFlags(
     mainFormObject.getClass())[i]==10 )
  if(mainFieldNames[i].equals(nameSubformField)){
          InstructorsForm subform=new InstructorsForm (
                                   pm, subformObject);
          subform.setParentForm(this);

          subform.setTheLinkedFieldName(mainFieldNames[i]);
          mainFelds[i] = subform;

     }
}
```

The *myJDOImplHelper.getFieldFlags(mainFormObject.getClass())[i]==10* verifies if the field represents a collection. Otherwise, it is not necessary to create the subform since the field will not represent a link to another table.

If the above is functioning correctly, then a new form is created using:

*InstructorsForm subform=new InstructorsForm ( pm, subformObject);*

Where:

*pm*: is the Persistent Manager

*subformObject* :declares what kind of object the subform holds.

For example, to create a subform in *Students* form representing the *Instructors* with whom the *student* fly, the subformObject will be an instance *Instructors* class. The *Student* class has a collection field named *instructors* for that purpose. It is possible to create a subform representing the instructors of the *Student* instance using the following:

```
setSubformForTheField("instructors", new Instructors())
```

The link is also set using:

```
subform.setParentForm(this);
```

In order to have the correct data, when iterating through the main form, the dataset of the subform is changed to that of the main form using the following:

```
if(mainFelds[i] instanceof InstructorsForm){// Means that
            //this will be a subform
  LinkedList datalist=new LinkedList();
  if(returnValue!=null)
     datalist=new LinkedList((Collection) returnValue );
    ((InstructorsForm) mainFelds[i]).setMainData(datalist);
      //Get the data from the main form
     //this first and then nextRecord() because nextRecord() goes recursively
     ((InstructorsForm)
          mainFelds[i]).updateNumberOfRecords()
     ((InstructorsForm) mainFelds[i]).nextRecord();
```

# IV. JDO DISADVANTAGES

This chapter explores the disadvantages of JDO.

## A. MANY TO MANY RELATIONSHIPS

### 1. The Redundancy of Tables

The JDO is able to create a one to many relationship by using the <collection> Tag in the JDO metadata file. For example, in the *Training-Squadron* application, the class Categories exists, which have a one to many relationship with the class Exercise (An Instance of Categories can have many Exercise instances).

A field in class Categories (named exercises) is used to create the relationship. The field is a list (LinkedList) and it is necessary to provide the content of the list in order to help LiDO create the tables in the database.

The relative commands to the JDO metadata file were:

```
<class name="Categories" identity-type="application">
     <field name="seriesid" primary-key="true"/>
     <field name="stadioid" primary-key="true"/>
     <field name="id" primary-key="true"/>
     <field name="exercise" >
          <collection element-type="Exercise"/>
     </field>
 </class>
```

As seen, it is possible to declare explicitly that the exercise field will contain instances of Exercises class. The metadata for the Exercise class are as follows:

```
<class name="Exercise" identity-type="application">
     <field name="seriesid" primary-key="true"/>
     <field name="stadioid" primary-key="true"/>
     <field name="categoriesid" primary-key="true"/>
     <field name="id" primary-key="true"/>
                    </class>
```

When the batch file for the creation of the schema is run (using inside the command):

*java -cp %CLASSPATH% com.libelis.lido.DefineSchema -properties lido_mysql).*

LiDO created three tables for the above relationship:

One Table represents the Categories named: **c**_categories.

(**c**_ :comes from the initial of the package containing the class- **c**ompany in our case)

Another Table represents the Exercise named **c**_exercise.

The last table represents the relationship named: c_categories_exercises.

The relationship contains the primary keys of both tables.

The primary key of the Table from the **one** side becomes a foreign key in that table named as follows:

*LIDOFK_ seriesid*
*LIDOFK_ stadioid*
*LIDOFK_ id*
*(for Link to Categories class)*

The primary key of the Table from the **many** side becomes a value key in that table named as follows:

*LIDOVALUE_ seriesid*
*LIDOVALUE _ stadioid*
*LIDOVALUE _ categoryid*
*LIDOVALUE_id*
*(for Link to Exercise class)*

Thus, it is possible to have a clear representation for a one to many relationship. The problem is that it is not possible to navigate from *Exercise* to *Categories* while it is easy to navigate from *Categories* to *Exercises*.

An inverse relationship is necessary to navigate in both sides.

### 2. Inverse Relationship

The implementation of a **many-**to-**many** relationships is very difficult in JDO, which demands the use of a *key- inverse* tag, also called a managed relationship. The main problem is that managed relationships are not supported by JDO but is an implementation a specific feature. In other words, some vendors support it while others do not, which is not, of course, a portable solution.

A relational DBMS could solve the problem by using another table that could hold the reference of both tables (their relationship). An attempt at representing the **many**-to-**many** relationship in JDO is discussed as follows. The Training Squadron application has a **many-**to-**many** relationship between Instructors and Students. Students can have many Instructors and Instructors can teach many students. The JDO creates four tables when using collections on both sides of the tables, and it is necessary to inform the relation in both sides programmatically. Trying to use a managed relationship with a *reverse*-key in both tables does not work. To resolve the problem, it was first necessary to define two *lists* in both classes: One in *Instructors* class is named *students* and the other in *Students* class is named *instructors*. Then, a managed relationship in both sides was attempted as follows.

```
<class name="Instructors" >
     <field name="id" primary-key="true"/>
<field name="students" >
     <collection element-type="Students">
          <extension vendor-name="libelis"
          key="sql-reverse"
          value="JavaField:Students.instructors"/>
     <collection/>

</field>


</class>

<class name="Students" >
```

```
     <field name="id" primary-key="true"/>
<field name="instructors" >
<collection element-type="Instructors">
     <extension vendor-name="libelis"
     key="sql-reverse"
     value="JavaField:Instructors.students"/>
<collection/>

</field>
</class>
```

However, this approach does not work since JDO was unable to do the mapping using this syntax. The next attempt tried to change the references from a Java field to a Table field (JavaField- dbfield) as follows but was also unsuccessful:

```
 <class name="Students" >
     <field name="id" primary-key="true"/>
<field name="instructors" >
    <collection element-type="Instructors">
<extension vendor-name="libelis"
        key="sql-reverse"
        value="dbfield:id=LIDOVALUE_id"/>
     <collection/>

 </field>

</class>
```

The only way to solve the problem is to create an additional class to represent the relationship (An Intsructors_Students class) and then use a managed relationship with collections in both of the other two tables These represent the Students and the Instructors classes. Such an implementation would increase the complexity of the program and the queries. The most important, however, is that such an implementation requires a managed relationship (the use of a *key-inverse* element) which is not supported by JDO, but is implementation specific (vendor specific). LiDO fortunately supports managed

relationships but implementing the program using that LiDO-specific feature might case a lack of portability among other JDO implementations.

It is possible to write the metadata as follows by using the features that JDO supports:

```
<class name="Instructors" identity-type="application">
     <field name="id" primary-key="true"/>
     <field name="lname" >
          <extension vendor-name="libelis" key="sql-index"
          value="unique"/>
      </field>
<field name="students" >
     <collection element-type=" Students ">
     </collection>
</field>

      <field name="aircrafts" >
          <collection element-type="Aircraft"/>
      </field>
</class>


<class name="Students" identity-type="application">
     <field name="id" primary-key="true"/>
<field name="instructors" >
     <collection element-type="Instructors">
     </collection>
</field>

</class>
```

This implementation demonstrates that JDO creates the following tables:

c_instructors

c_instructors_students

c_students

c_students_instructors

Thus, there are four tables instead of three, and it is necessary to inform both relationships explicitly when a new reference is added in either the Instructors or Students classes.

Fortunately, this is easily done using JDO capabilities, but in any case, it is not a good implementation. This implementation could render false data if a break occurs in the sequence of informing the tables. Thus, it is possible to inform the Instructors table about the students and a loss of power could prevent the informing of the Student class about the instructors, leading to possible corrupted data in the application. Additionally, since it is necessary to inform both reference tables instead of one, the time for such an operation will almost double. Thus, a decrease in performance occurs.

In this case, the following commands in the implementation are used in Instructors class:

```
public void addStudents(Students student) {
      if(students==null)
            students=new LinkedList();
            students.add(student);
if(!student.getInstructors().contains(this))
       student.addInstructors(this);
}
```

In Students class:

```
public void addInstructors(Instructors instructor) {
      if(instructors==null)
            instructors=new LinkedList();
            instructors.add(instructor);
if(!instructor.getStudents().contains(this))
              instructor.addStudents(this);
}
```

The same must be done for the delete procedures. In Instructors class:

```
public void deleteStudents(Students student) {
      if(students!=null{
          students.remove(student);
if(student.getInstructors().contains(this))
      student.deleteInstructors(this);
      }
}
```

In Students class:

```
public void deleteInstructors(Instructors instructor) {
      if(instructors!=null){
           instructors.remove(instructor);
if(instructor.getStudents().contains(this))
      instructor.deleteStudents(this);
      }
}
```

In summary, an additional table is needed in order to implement a many to many relationship. Thus, the normal JDO implementation requires more space on a hard disk, more code to implement the read, write, and delete procedures, and a profoundly longer time operation for also informing the additional table.

**B. FIELD REDUNDANCY**

Also, a big problem is the use of additional redundant fields. JDO and the LiDO implementation cannot create a compound key when the relative field is part of a reference.

For example, the application contains the *Stadio* class, *Stadio* represents a super category where many categories belongs to it. The *Stadio* class has a **one** to **many** relationship with *Series* class and represents the part of **many** in this relationship.

In the definition of the *Stadio* class, there is a field called *series* which represents the *Series* class of which the *Stadio* is a part. Actually, the *series* field in the database only keeps the primary key of the possessing class, which in this case is the *id* field of the *Series* class. If the following is attempted, an error message occurs stating that JDO is unable to find a getter method for type *company.Series*:

```
<class name="Stadio" identity-type="application">
     <field name="id" primary-key="true"/>
     <field name="series " primary-key="true"/>
     <field name="categories" >
          <collection element-type="Categories"/>
     </field>
</class>
```

Another possible solution could include an effort to make a compound primary key for the *Stadio* class consisting of the *id* field of the *Stadio* class and the *id* field of the *series* field, which is an instance of the *Series* class.

The following shows the metadata code for the above purpose.

```
<class name="Stadio" identity-type="application">
     <field name="id" primary-key="true"/>
     <field name="series.id" primary-key="true"/>
     <field name="categories" >
          <collection element-type="Categories"/>
     </field>
</class>
```

However, this does not work. When it was tried, LiDO created the c_stadio table and used the field name *id* as a primary key, while the *series* field, and specifically, the *series.id* was not part of the table's primary key. Not only did this happen, but there was no error message or any other message stating that JDO completely ignored the following statement:

```
<field name="series.id" primary-key="true"/>
```

The problem is that JDO uses only primitive types for the key values.

However, in this case, some problems need discussion. It was not possible to not reuse the same *id* and include it to another relation with a *series.id* by having the table c_stadio with the *id* as the only member of the primary-key, The limitation results because the *id* will be a primary key, and therefore, duplicate values are not allowed. Thus, the desirable one to many relationship between the c_series and c_stadio tables could not be implemented using the *id* field as the only member of the key. This was impossible since a managed relationship was used for the two tables. The metadata for the *Series* class is as follows:

```
<class name="Series" identity-type="application">
     <field name="id" primary-key="true"/>
<field name="stadio" >
     <collection element-type="Stadio">
           <extension vendor-name="libelis"
            key="sql-reverse"
            value="javaField:series"/>
     </collection>
</field>

     <field name="groundcourse" >
          <collection element-type="GroundCourse"/>
     </field>
     <field name="students" >
          <collection element-type="Students"/>
     </field>
</class>
```

Using the key="*sql-reverse*", LiDO creates only two tables for the relationship: the c_series and the c_stadio. The c_stadio table includes a field *series_id* which declares the relationship between the two tables. Since the primary key for c_stadio is only the *id*

field of the *Stadio* class, it was not able to have a *stadio* instance with a same *id* to be referenced by another *Series* instance. Actually, this means that it was not possible to have a **one** to **many** relationship.

A possible solution was to let LIDO create the primary key. In this case, the identity type of the class would be *datastore* as follows:

```
<class name="Stadio" identity-type="datastore">
```

In this case, as JDO determines the vendor, it will create the key. The primary key will be one field of a long unique value. However, this implementation again entails the danger that it is possible to install the same *stadio* instance (with the same *id* value) more than once in the same *series* instance without receiving an error message, something that makes the database very risky and error prone.

A solution could be to write a query to check whether the *stadio* instance is already referenced by a specific *series* instance before the addition of a new reference. However, this is also time consuming and it could not prevent the insertion of wrong data from another application.

Thus, the only solution for the Training-Squadron application was to introduce a new field to keep the series *id* in the c_stadio table, which will provide the reference for the *series* instance of which the *stadio* is a part. The final metadata was:

```
<class name="Stadio" identity-type="application">
     <field name="seriesId" primary-key="true"/>
     <field name="id" primary-key="true"/>
     <field name="categories" >
          <collection element-type="Categories">
          </collection>
     </field>
</class>
```

where *seriesId* is the new field t introduced to the database. The definition of the class became:

```
public class Stadio  {
      private Series series;
      private BaseStadio sid;
      private String description;
      private Collection categories;
      private String seriesId;
      private String id;.
      ……..
```

In order to avoid new entries, the *seriesId* field is informed automatically along with the *series* class.

```
public void setSeries(Series ser) {
        this.series= ser;
        this.seriesId=ser.getId();
}
```

Thus, there is a redundant field. The same situation appears to *Categories* and *Exercise* classes where it was necessary to introduce two and three more fields in order to have a suitable primary key.

Although the information of these fields is not difficult, more code is needed when creating a form to display the data either for the display of these fields or for their abstract of the set fields that appears on the form. Actually, these additional fields must be hidden since the user should not be able to change their values. Instead, their values must be set automatically and according to their referenced class.

However, more importantly, there are redundant fields consuming more space and time for the implementation. The author believes that JDO could be able to understand the values. Although JDO needs to use primitive types, there must be a solution either to have the primary key of the class when referred to a class or to be able to set the value of the field to the metadata JDO file such as

```
<field name="series.id" primary-key="true"/>
```

instead of

```
<field name="series" primary-key="true"/>
```

which does not work, or when in the second case, the JDO could automatically provide transformation to the key field or fields of the series class.

## C. JDO GROUP BY QUERIES

JDO does not support GROUP BY queries, which is a great disadvantage.

### 1. Performance Imitating a GROUP BY Query

The performance of JDO queries in a set of 2000 records is provided that have been stored in *Flights* class in the underlying c_flights table.

The following chart shows the time required for a JDBC GROUP BY query to take group data. The first bar is the time of receiving the result set while the second includes the time for iteration through the result. The third bar represents the time it takes the JDOQL to execute a query and then iterates through the result set in order to achieve the same group data as the JDBC GROUP BY query.

JDO takes almost 4 seconds when it executes for the first time while the group by query is significantly faster.

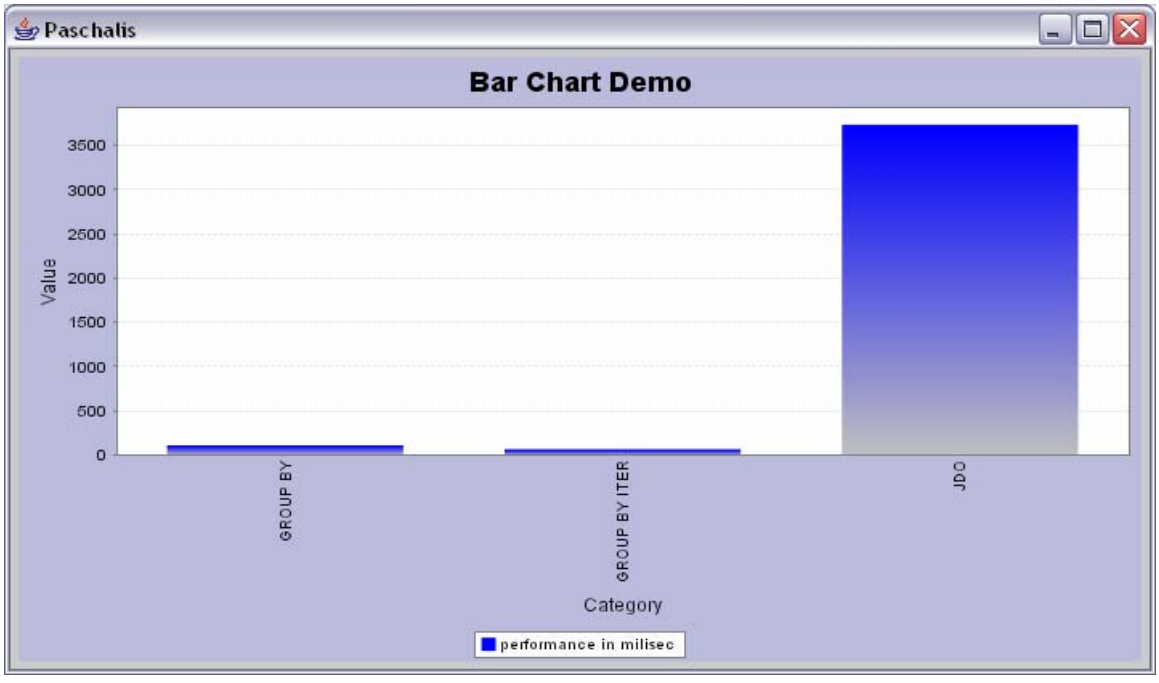



Figure 3. JDO Performance Imitating a GROUP BY Query.

The following chart shows the results after executing the queries for a second time with more operations already in memory. Note that the time is minimized to less than one second but the same happens to the GROUP BY query and the difference is again the same.

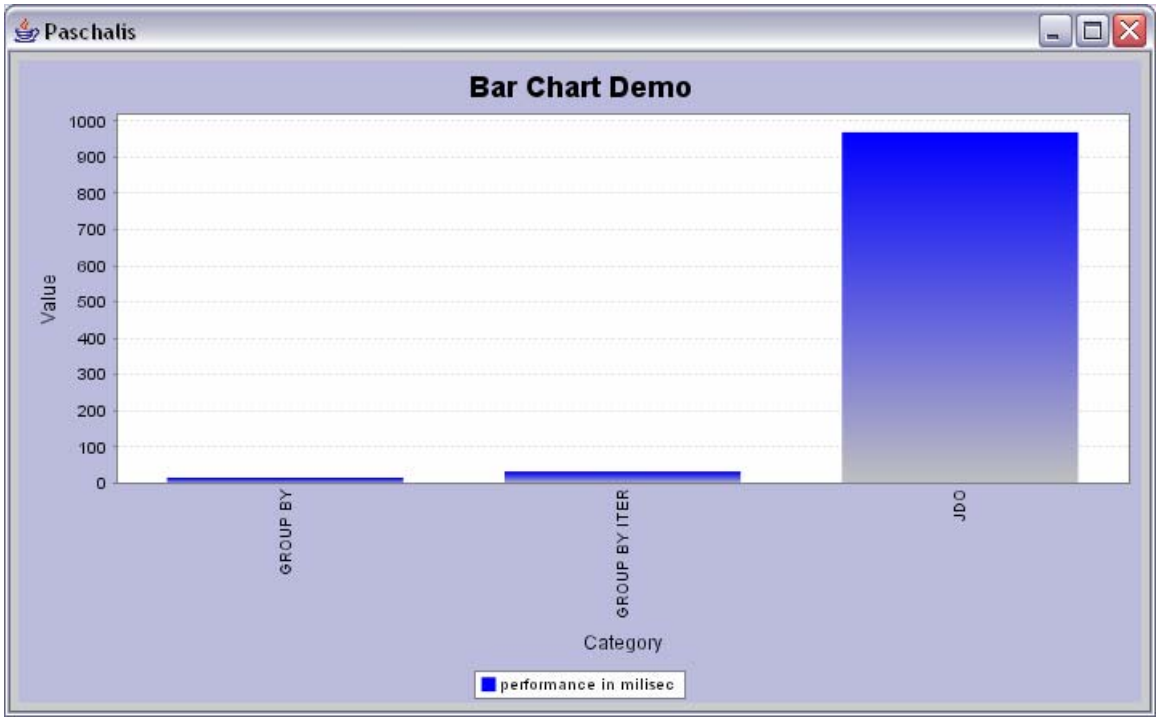



Figure 4. JDO Performance Imitating a GROUP BY Query (Second Execution).

LIDO can pass pure SQL commands using the "sql" tag.

In attempting to pass a GROUP BY query as follows:

```
String sql="SELECT a.instructor_id, SUM(a.endurance) "+
           "FROM c_flights a "+
           "GROUP BY a.instructor_id ";
Query query=pm.newQuery("sql",sql);
query.setClass(Flights.class);
Collection result=(Collection) query.execute();
```

the following message is received.

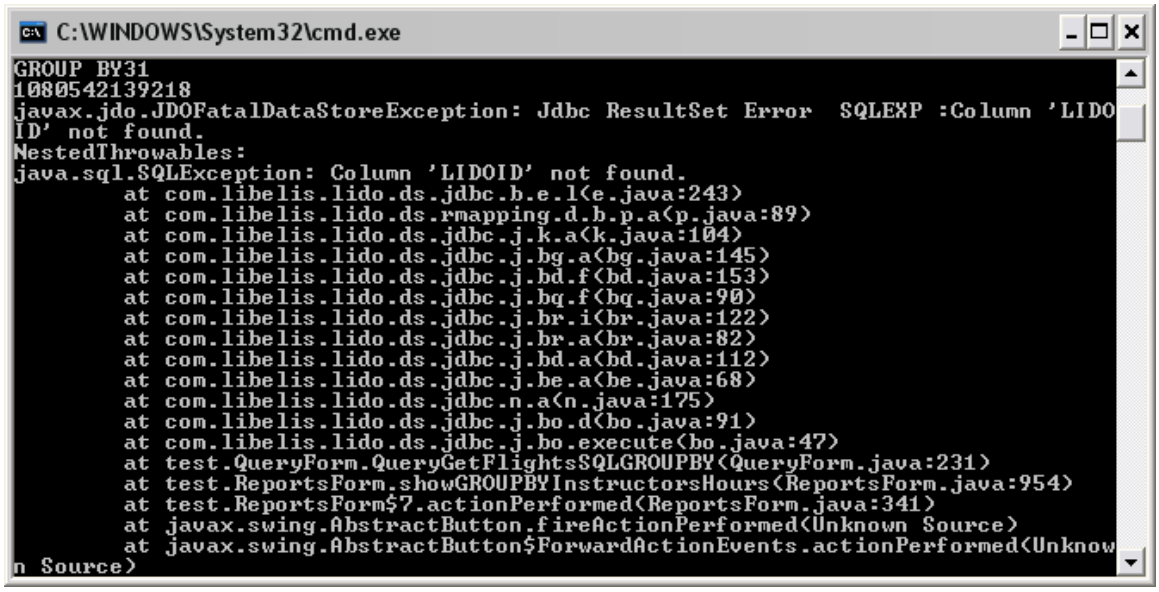



Figure 5. LIDO Message for 'sql' tag.

It appears that the "sql" tag does not support GROUP BY queries.

**2. Performance for Simple SELECT Queries**

JDO's performance, however, is slow even for simple SELECT statement queries. The following chart is created by executing a select query using the "sql" tag and retrieveAll(). The first bar is again from JDBC execution of the same query (SELECT query here). The time for JDO is 1672 milisec.

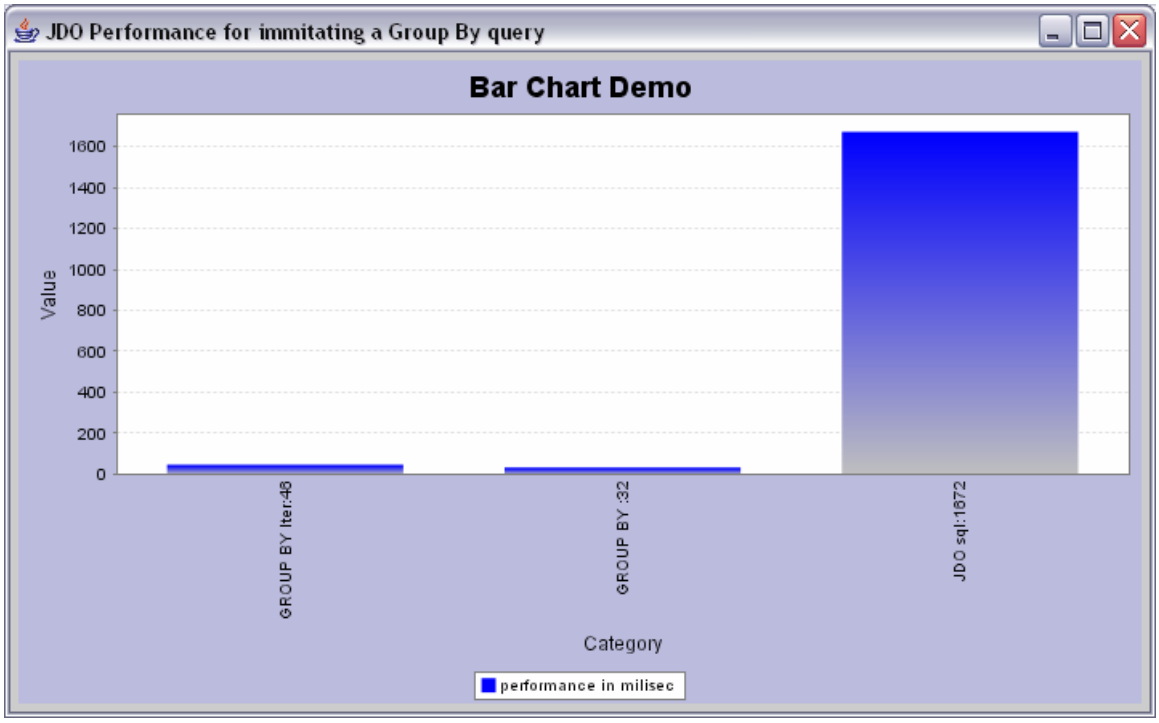



Figure 6. JDO Performance Using 'sql' Tag and retrieveAll() Operation.

If the same query is executed without the retrieveAll() operation, the following result occurs. The time for JDO falls to 297 milisec. This huge difference is caused by JDO because the result set does not contain the values of each record but contains only hollow instances, indicating that the time desired to learn the *student_id* of a specific flight instance JDO will go and read it from the datastore after the request. Data information are not loaded in memory.

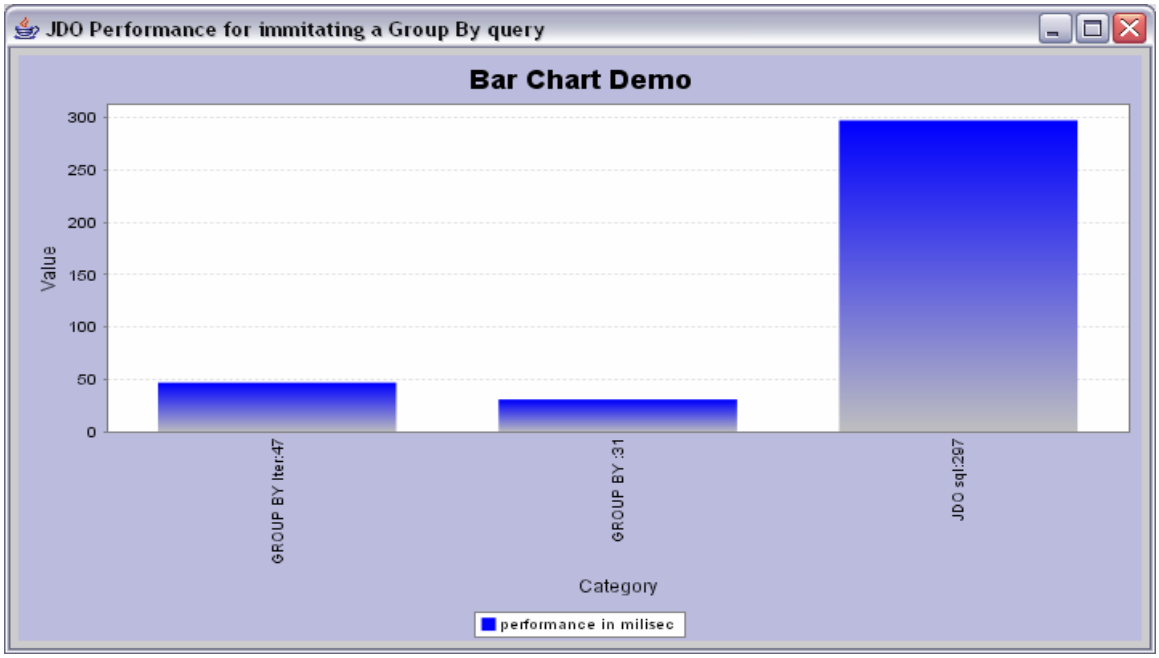



Figure 7. JDO Performance Using 'sql' Tag without retrieveAll() Operation.

For example, if the same query is executed without retrieveAll() and iterated through the result set to get values of the instances, the time will increase significantly as the following chart shows.

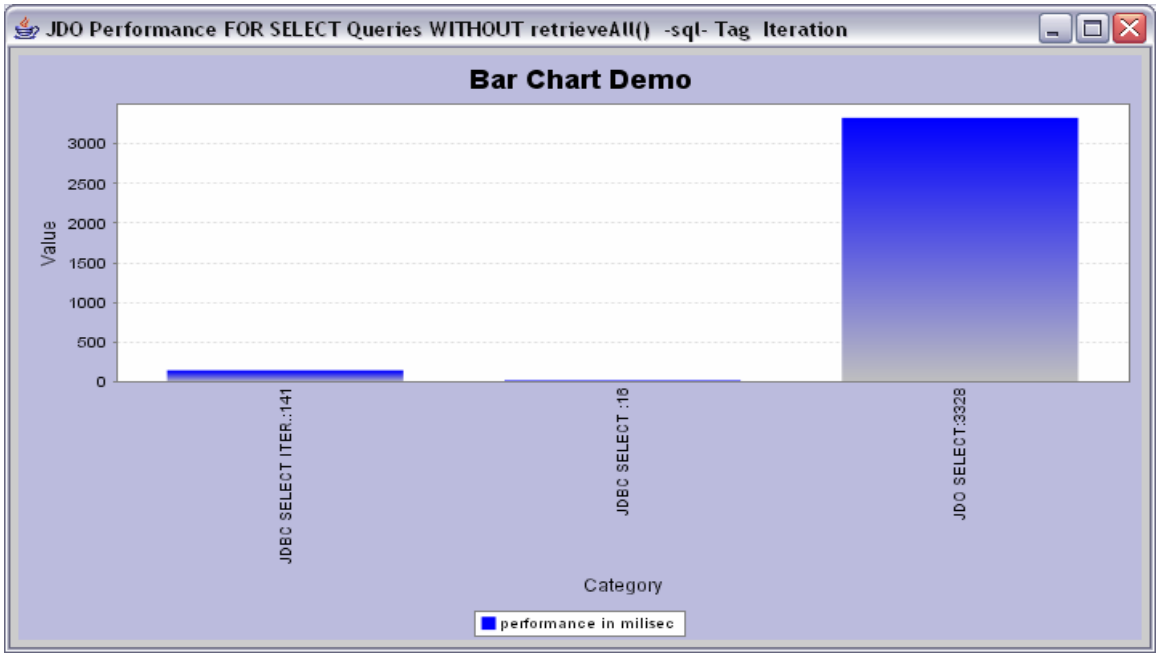



Figure 8. JDO Performance Using 'sql' Tag without retrieveAll() but with Iteration through the Result Set.

The situation is not better in a simple SELECT clause query. Below are the results of executing a select query using JDBC and JDO without the "sql" tag. The results are taken using RetrieveAll() and iteration through the data of the JDO result set.

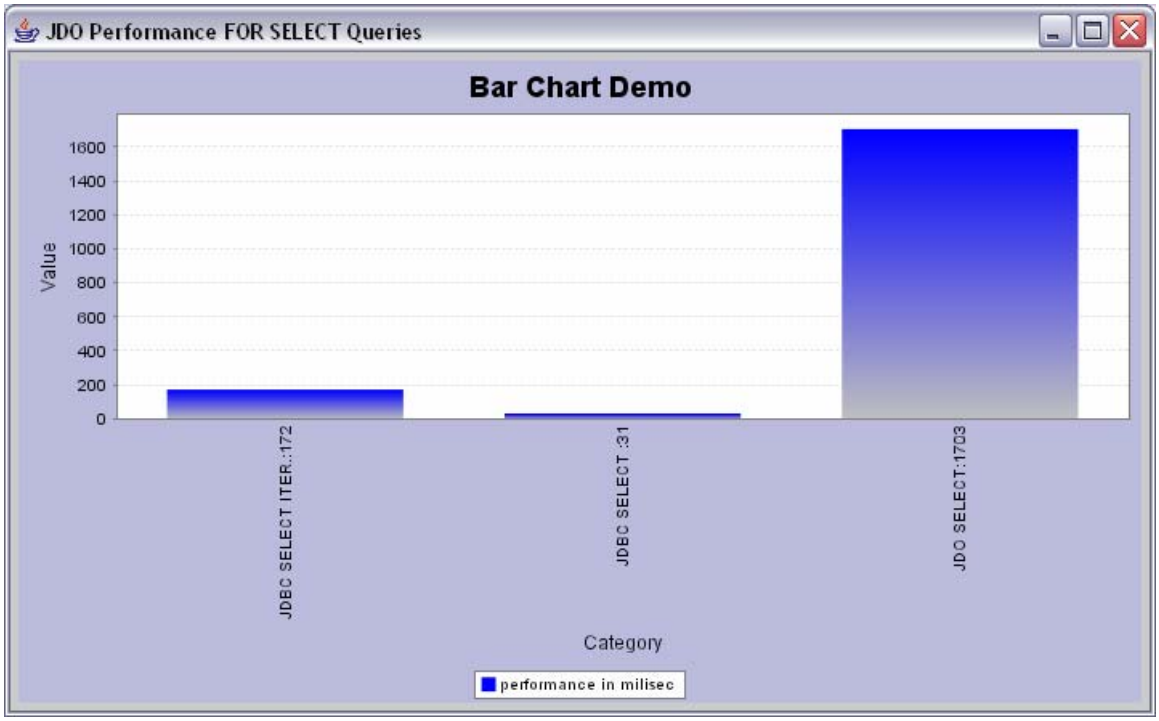



Figure 9. JDO Performance without 'sql' Tag with retrieveAll() Operation and with Iteration through the Result Set.

The same query without iterating through the data of JDO result test gives the following chart.

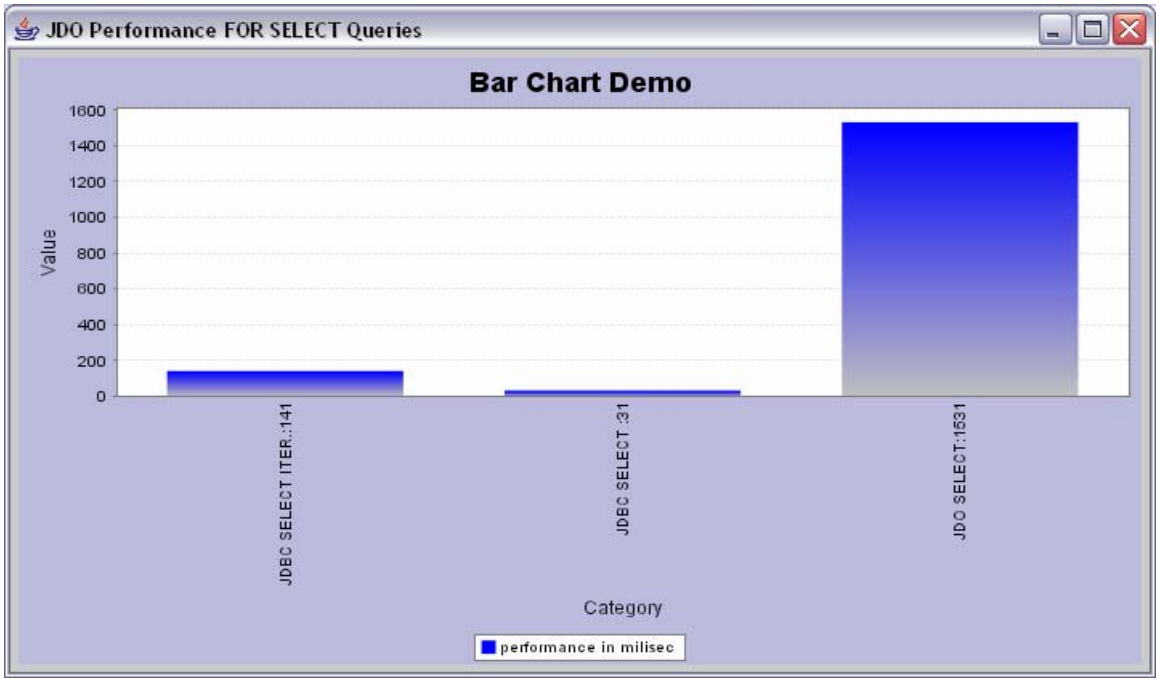



Figure 10. JDO Performance without 'sql' Tag with retrieveAll() Operation but without Iteration through the Result Set.

The results are the same using the " sql" tag.

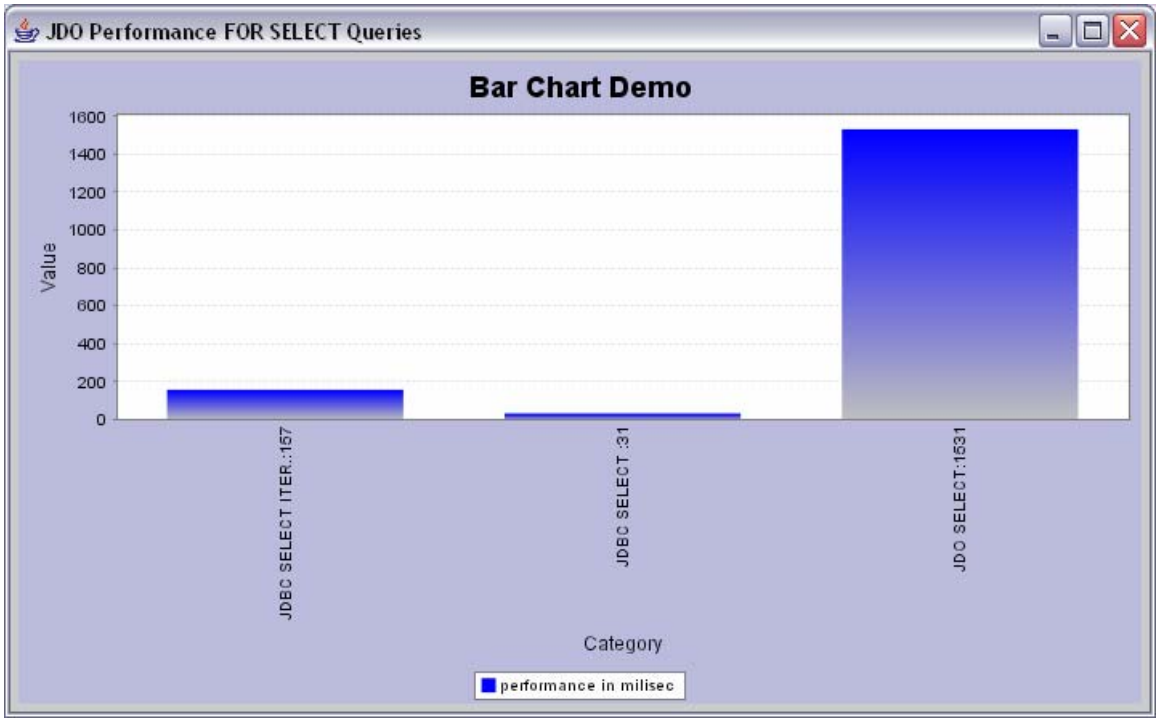



Figure 11. JDO Performance with 'sql' Tag with retrieveAll() operation but without Iteration through the Result Set.

The remaining charts shows the performance of JDO without RetrieveAll() (With Iteration)

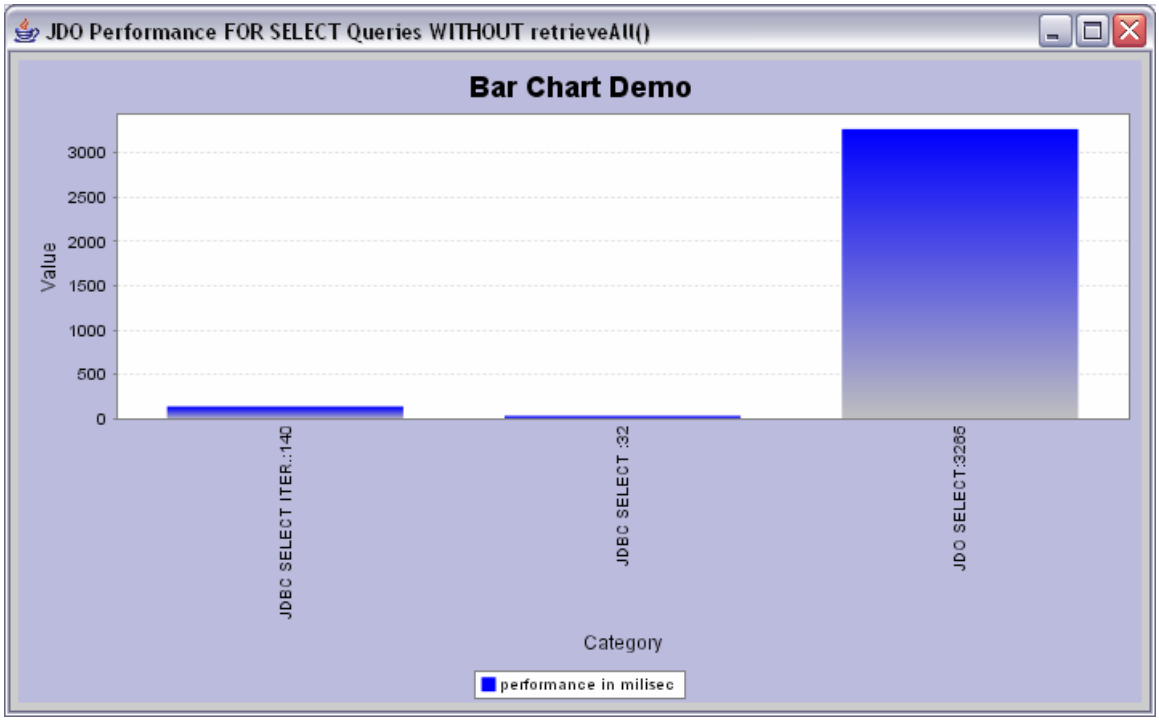



Figure 12. JDO Performance without retrieveAll(), and with Iteration.

Without retrieveAll() (and without iteration), the result was very high because the result set contains hollow instances.

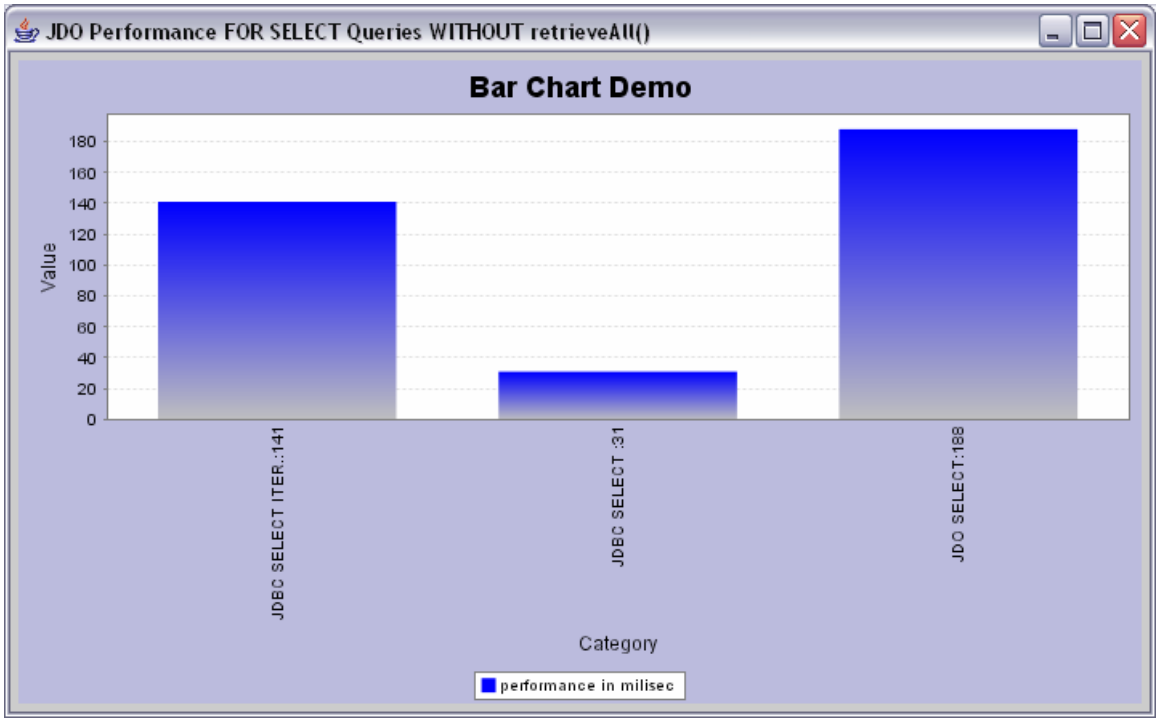



Figure 13. JDO Performance without retrieveAll(), and without Iteration.

Without RetrieveAll() (Without iteration) and using SQL TAG results in:

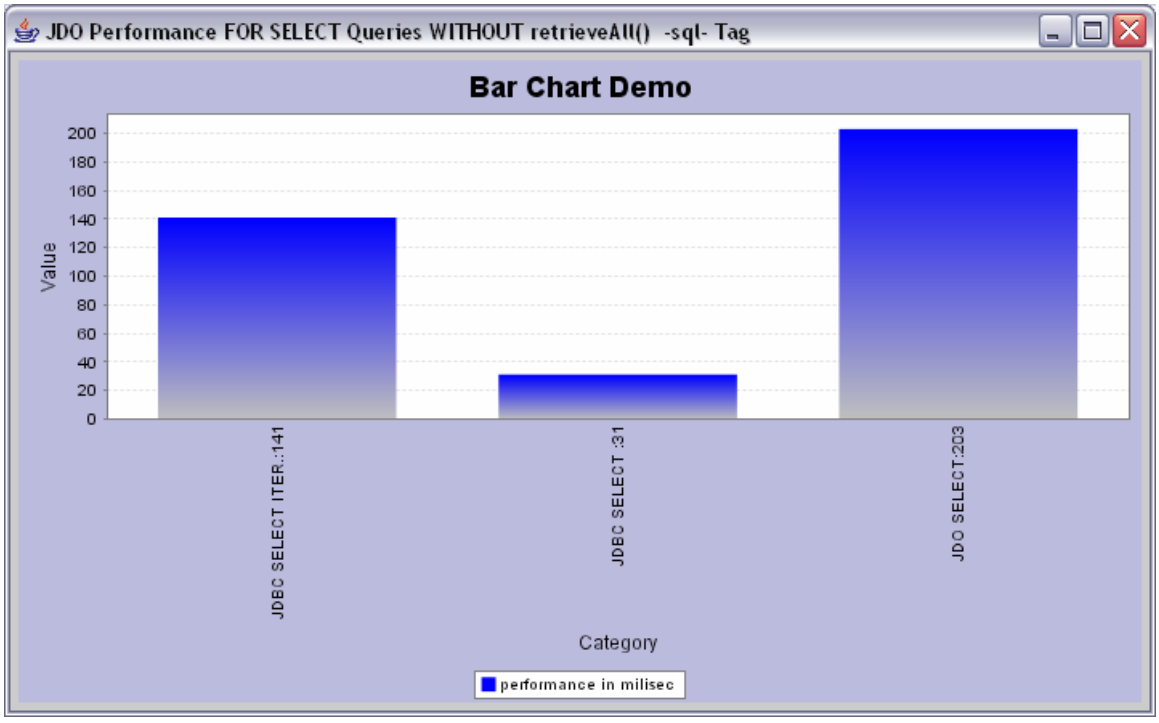



Figure 14. JDO Performance without retrieveAll(), and without Iteration Using 'sql' Tag.

Note that the performance of the JDO is far worse than that of JDBC and pure SQL. The problem is more significant in GROUP BY queries where it is necessary to iterate through the data in order to achieve a group result.

Another interesting conclusion is that even when using the "sql" tag and passing pure SQL to the query, the performance of JDO remained far behind the performance of JDBC using the same SQL commands. The retrieveAll() command almost doubles the performance of JDO queries when iterating through the data.

**D. NO SUPPORT OF ALL THE DATA STRUCTURE TYPES**

Another disadvantage is that JDO does not currently support all types of data structures for all the operations.

For example, the following definition for the *Series* class was used:

```
public class Series  {
    private String id;
    private String description;
    private LinkedList  stadio;
    private LinkedList students;
…….
}
```

The *Series* class as described previously has a **one** to **many** relationship with the *Stadio* class. (The *Series* is part of the **one** while the *Stadio* is part of the **many**)

Initially the (I THINK THAT YOU LEFT SOMETHIG OUT HERE) was used to represent the above relationship.

```
private LinkedList  stadio;
```

Attempts at mapping the underlying database were unsuccessful.

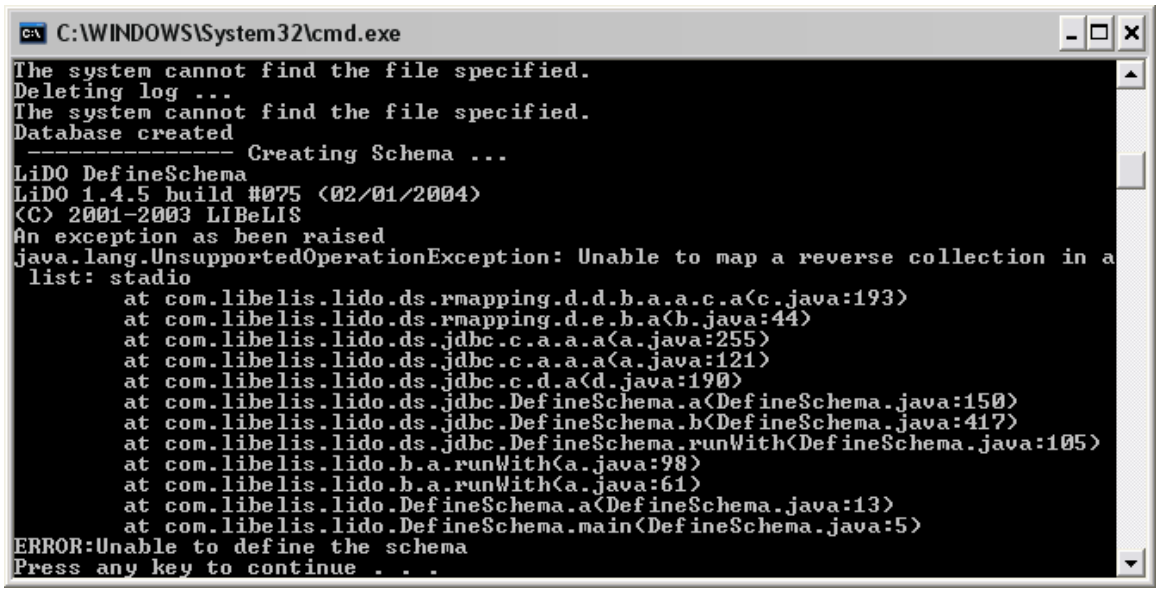
Efforts were made to keep the data in two tables without the intermediate representing the relationship. Thus, the "sql-reverse" key was used to keep the data in only two tables, with the table representing the Series class and the table representing the Stadio class, which will have a foreign key declaring the ownership reference in the series table.

The metadata file appears as:

```
<class name="Series" identity-type="application">
     <field name="id" primary-key="true"/>
     <field name="stadio" >
          <collection element-type="Stadio">
               <extension vendor-name="libelis"
               key="sql-reverse"
               value="javaField:series"/>
          </collection>
     </field>
     <field name="students" >
          <collection element-type="Students"/>
     </field>

</class>
```

The code seemed valid but the following error message was received:




The problem occurred when using the LinkedList for the *stadio* data. The definition of the *Series* class was changed as follows:

```
public class Series  {

            private String id;
            private String description;
            private Collection stadio;
            private LinkedList students;
            private Collection groundcourse;
           ……..
```

The problem was solved without changing the metadata JDO file, which was extremely interesting since the implementation of a LinkedList is not a problem for representing the **one** to **many** relationships. The problem only appeared when attempts were made to use an inverse relationship. In this case, only a Collection is suitable while LinkedList is prohibited or unsupported as the message states.

It was not possible to find a good reason for this diversity since LinkedList is used for relationships as previously stated. Moreover, this kind of diversity creates inquiries concerning the reason for the error. Additionally, the use of metadata file, which is written in XML, makes discovering the errors more difficult. It will not be easy to accept that a collection will be supported for normal and inverse-key relationships while the LinkedList is only supported for a normal relationship. Therefore, the use of LinkedList is all the more effective. It provides the *listIterator*, which is able to use the *previous*() along with *next*(). Thus, a far more better navigation ability exists than using the simple *iterator* of a Collection. In this application, the *listIterator* is used in order to be able to move efficiently through the data in the forms. In order to do so, it was necessary to convert every Collection data type to LinkedList using the following constructor:

new LinkedList( Collection c);

Thus, support of all kinds of data structures could eliminate the need to write much more code and use of memory.

## E. NO FREE SOURCE CODE

One of the main problems impeding the evolution of JDO is its vendor specific implementation. No open source implementation exists so becoming familiar with the concept is not very easy. The only open source implementation is Sun's reference implementation but this is very limited in abilities.

One of the very few free implementations is the community edition of LiDO by libelis. However, even this needs a license which expires after six months. Most vendors provide their product only by payment. This is expensive for a product when no clear depiction of how it works exists and how efficient it could be. This depiction is all the more difficult since familiarization with JDO is not easy considering the absence of open source implementations.

This image has, of course, negative repercussions for the evolution of JDO. Although its investors support that the implementation of an application will be much faster, many problems exists.

Actually, quite a bit of time is required for installing and the executing the classes of the program. The enhancement of the classes requires additional preparation which makes the compilation time little trouble. Batch files could be used for the execution of the code and much time was spent on debugging and error detection and correction with this procedure. An integrated environment covering the JDO specification would be very useful.

The vendors might have their integrated environments. However, these are not open source and so familiarization with the product is difficult. Moreover, the use of the metadata file, which is written in XML, makes the situation more cumbersome.

## F. WHY XML?

The metadata file could also be considered one of the negative aspects of JDO. First, it is written in XML, which is another language with which the user needs to be familiar.

However, JDO came as a solution that will work completely with java and will free the user from the necessity of learning and using another complex language, such as SQL. Then, why does the user need to become familiar with XML? At least SQL can

provide a more efficient approach for retrieving and manipulating data and is much faster as the previous chart and figures have shown. Its use could be justified since the performance of JDO lags behind, so it is a risk worth taking. However, using XML just for mapping or describing the method for creating the underlying database is not necessarily a very good idea.

The detection of errors is very difficult inside the XML file, which could result in a bad experience. For example, using the following metadata in Series class renders:

```
<class name="Stadio" identity-type="application">
     <field name="id" primary-key="true"/>
     <field name="series.id" primary-key="true"/>
     <field name="categories" >
          <collection element-type="Categories"/>
     </field>
</class>
```

It can be expected that the JDO would create the primary key using the primitive type of *series.id*. However, this does not happen because a message for the above code fragment stating that something is going to be ignored will not be received. It is possible to discover that JDO ignores the command only when looking at the underlying database.

Writing the metadata file also, in a simple text editor, is not very simple, which is especially true when it is necessary to create a large application with many tables taking care of the xml file. This will now be all the more difficult. Using different environments for writing the xml file and the java files is also a disturbing issue. In conclusion, this metadata file will be preferable to be included as an API interface in JDO. No other important reason exists why it could not be done.

Classes that need to be persistent could be done by implementing the particular *Persistent interface*, for example. Mapping with a database could be done also using the particular methods of that interface. (i.e. Persistent.setKey(Class, field[]) ) Thus, it is possible to achieve more scalable applications.

It is necessary to now provide the vendor's name and the vendor's specific keys with the xml file. Additionally, it is essential to learn what every vendor does and how it uses its commands, something that could be hidden behind the implementation of the *Persistent interface*. With the current profile, it is at least necessary to change the metadata file when changing vendors, which could be very burdensome if the application is large enough.

Also, what if one vendor supports a managed relationship while others do not. Then, the application will only be vendor specific, and one of the key strengths of JDO disappears. In addition, what if a managed relationship is desired but referential integrity is required during the deletion of a record or the receiving of a message for applying or not applying referential integrity or not is also preferred. It is possible to create such issues easier using an interface instead of the xml file.

## G. METADATA DEFICIENCY

Finally, the problem with the metadata exists. For example, while the length of the fields is specified, it was not possible to find a good way to retrieve this metadata information.

The only usable method for the programmer found is JDOImplHelper CLASS. However, as the API states:

> This class is a helper class for JDO implementations. It contains methods to register metadata for persistence-capable classes and to perform common operations needed by implementations, not by end users.

The following can be taken from JDOImplHelper:

Field names

Field types

Field flags.

It was not even possible to find a description of the flags. JDO is pragmatically very limited in what it provides to the programmer.

It was necessary to experiment with different cases to conclude finally that

A flag of 10 is a list

A flag of 8 is a primary key …

In addition, this information is very important for the programmer. Attempts were made to find classes that provided the field length but they were unsuccessful. Methods that return the values of the field parametrically were also tried, such as

```
getField(int fieldIndex)
```

or

```
getFieldByname(String fieldname)
```

These types of methods are very useful for creating strong and short applications. Additional methods must be found for setting values to a field. Certainly, JDO knows these kinds of methods and uses them for the implementation but it does not provide them. It might provide them, but the author did not find them, although even the JDOImplHelper comes with the comment that it is not indented for use by the final user.

Since the idea was to create a very fast application and exploit the advantages of java and since no JDO metadata information was found, an interface was provided to provide some metadata and the parameterized access of the fields for reading and writing. While the interface does not use more than six methods, it was possible to create a class that provides the navigation and handling of all the classes that implement this interface.

The description of this class is provided in Chapter III where the description of the application appears. However, it is important that using the metadata can result in a very good and very fast job with JDO. It will be greatly advantageous to application development if JDO provided all this metadata information. It was, however, disturbing to give the metadata information to the xml file and write them again using an interface simply because no other way exists to retrieve them.

THIS PAGE INTENTIONALLY LEFT BLANK

# V. JDO ADVANTAGES

This chapter describes the advantages of JDO for the development of an application in comparison with other approaches.

JDO is an important innovation for the Java platform. Before JDO, java developers had to use the JDBC for database access. For enterprise applications, the solution was Enterprise Java Beans APIs which provided a container-managed persistence. JDO was an effort to provide a simpler way for creating persistence in a Java platform. Thus, development could be easier for java developers. JDO came as a solution for providing database persistence minimizing the amount of time the Java programmer must spend developing.

## A. OBJECT ORIENTED VIEW OF THE DATABASE

JDO provides transparent persistence of Java object models in transactional databases. The user can map the Java classes to the data of the underlying datastore. With this mapping, the user does not need to be extremely knowledgeable of the database, and only needs to know how to handle the Java classes. Thus, the user works with an object-oriented model even if the application manages data from tables in a relational data model.

The development of a database application comes with the advantages of the object oriented programming. During the development of the Training–Squadron application, many of these advantages were apparent. First, the access of field values and the read write commitment to the data happened inside the class in a clear and efficient way. For the Instructor class, for example, it was possible to take the value of a field by simply calling a method of the class:

```
Public String .getLastName(){
     return lastName;
 }
```

This is a very simple way to access the data of a database. Similarly, it is easy to assign the values to a particular field:

```
Public void setLastName(String name){
```

```
        lastName=name;
    }
```

This is easier, of course, than executing a SQL query of type:

```
INSERT INTO Instructors
            ……..
VALUES […….]

Or better :

UDATE Instructors
SET lastName ……..
VALUES ……
WHERE id=……..
```

What is more impressive, however, is the method used to access fields in a relationship: Instructor class (actually the underlying instructors table) has a **one** to **many** relationship with Students class (actually the students table).

The retrieval or modification of data from one table to the referenced one can be done in a completely object oriented way.

The instructors class have a list representing the students

```
public class Instructors  {

        private String id;
                ……
                private LinkedList students;
                ……..
}
```

Some simple java methods can easily manage the relation.

```
public void addStudents(Students student) {
     if(students==null)
          students=new LinkedList();
           students.add(student);
……..
}
```

Either a Map can be used to keep uniformity of the data or verification if the student is already present in the Instructors list by simply using

```
If (!students.contains(stiudent))//
          students.add(student);
```

This is easier than using a query of type

```
INSERT INTO Instructor_student
……..
VALUES    xx……yy……..
```

before maybe executing a SELECT query for obtaining the values of xx or yy in both the Instructors and Students tables. Equally iImpressive is the method used to access values of fields of a referenced table.

Series.getStadio().get(1).getCategory().get(1).getExercise().get(1).getId()

This is much more understandable and easier than a query of type:

```
SELECT a.id
FROM exercise a INNER JOIN (categories b INNER JOIN
        (Stadio c INNER JOIN (series d on d.id=c.series_id)
         On c.id=b.stadio_id)
         on b.id=exercise.category_id
WHERE Series.id=xx and stadio id=yy and categories.id =zz..
```

## B. AUTOMATE PERSISTENT

Also important is the method for automate or transitive persistent. Attempting to add some records to the *Flights* class could create an instance of *Flights* class using the constructor:

```
public Flights(Aircraft aircraft,Instructors instructor,
                    Students student, Date date) {
        ……..
    }
```

The way to store the new records in JDO could be:

```
Students student= new Student(….);
Instructors instructor= new Instructors(…);
Aircaft aircraft =new Aircraft(…);
```

Finally:

```
Flights flight = new Flights(
          aircraft, instructor, student, today())
```

if :

```
PersistentManager pm.makePersistent(flight)
```

All instances of the instructor, student, and aircraft will become persistent by reachability and will be stored to the database if not already there. This is a great advantage when thinking about how many SQL codes it is necessary to written in order to insert all the above instances to the database.

## C. EASE OF IMPLEMENTATION

During the implementation of the Training-Squadron application, it was able to use one class for handling the data of all the classes. When using JDO, the data using only java classes is handled. Thus, all the advantages of object-oriented programming provided are possible.

For the implementation of the *Training- Squadron* application, it was tried to exploit database programming in this manner. Thus, it used the functionality that java classes provides. During the development, it used parameterized methods and it proved possible to provide one class for the data.

It also exploited the JDOImplHelper class and the data that it provides. The following methods were mainly used:

```
getFieldNames(Classs c)
getFieldTypes (Classs c)
getFieldLength(Classs c)
```

Using an interface succeeded in obtaining additional metadata such as the field length. Moreover, there were created functions for retrieving the values of the fields and setting the values of the fields.

For all the above, of course, java functionality and the object representation of data that JDO provides was exploited. Since the tables are provided as java classes or java objects, their manipulation is very efficient.

It proved possible to obtain the metadata from every table simply by using commands such as the following:

```
myJDOImplHelper=JDOImplHelper.getInstance();

//try to get the lengths of the fields
int[]fieldLength=getTheLengthOftheFields();

//get the name of the fields
```

```
mainFieldNames=
     myJDOImplHelper.getFieldNames(
                          mainFormObject.getClass());
```

myJDOImplHelper.getFieldTypes(mainFormObject.getClass())

It was only necessary to pass an object as a parameter and using the *getClass*() method to receive whatever data desired. It was possible to obtain additional metadata by simply invoking the method that provides these metadata. It used the functionality of *reflection* which made the application very short and cohesive

Below is an example of using reflection for taking the type (the kind of class that a list contains) of the field that represents a collection in a class.

```
java.lang.reflect.Method meth=
        mainFormObject.getClass().getMethod(
              "getTheTypeOfTheCollection", rc);
```

Object returnValue= meth.invoke(mainFormObject,obi);

For the Instructors class where the student field represents a LinkedList (which contains students) the following is applied:

```
public class Instructors  {
    private String id;
    ……
    private LinkedList students;
    …….
}
java.lang.reflect.Method meth=
   mainFormObject.getClass().getMethod(
         "getTheTypeOfTheCollection", "students");
```

//(The String value "students" can be taken just using the )

```
JDOImplHelper.getFieldNames(
```

```
            mainFormObject.getClass()))
Object returnValue=  meth.invoke(mainFormObject,obi);
```

From the above statement, *returnValue* will determine the class of the objects contained in students List.

Next, it was possible to pass the *returnValue* to a query and retrieve all the data of the class contained in the List without even knowing the class.

Actually the `getTheTypeOfTheCollection(String fieldname)` was created in order to receive the metadata information for the relationship between the tables. The implementation could be easier if JDO would provide this kind of metadata, which JDO could do since this information is provided when using the metadata file.

For example, for the instructors class, the following is written:

```
<class name="Instructors" identity-type="application">
     <field name="id" primary-key="true"/>
     <field name="lname" >
         <extension vendor-name="libelis"
          key="sql-index"
          value="unique"/>
     </field>

     <field name="students" >
          <collection element-type="Students">
          </collection>
     </field>
     <field name="aircrafts" >
          <collection element-type="Aircraft"/>
     </field>

</class>
```

Note that the field with name students is explicitly declared, which represents a *Collection* (to the specific example a *LinkedList*) that will contain objects of the *Students* class.

However, even without the provision of the above information by JDO, the implementation was easy using an interface that requires the method: *getTheTypeOfTheCollection(String fieldname)* and then reflection for the invocation of this method. This method of programming might be one of the most powerful points that JDO provides. Using the interface, it was able to create methods for retrieving the value of the fields and for setting the value of the fields. Using reflection for the invocation of these methods could create very beautiful programs minimizing the amount of code needed.

Thus, it used one and only one class for the representation of the data of many tables instead of using many different classes for each table. Also, the same parameterize techniques in JDBC could be used, but this is the most efficient and easiest.

**D. JDOQL**

The ability of using the advantages of java language for accessing the database is obvious also to JDOQL (JDO Query Language). JDOQL could be considered one of the advantages of JDO, used to provide access to persistent instances based on specified search criteria. JDOQL has been designed to accomplish the following goals:

- Query Language neutrality. This means that the query that will actually be executed will be either a SQL query for a relational database, or an object database query such as the Object Query Language (OQL). JDOQL will support all these operations with the same syntax, meaning it is independent of the query language of the underlying datastore.
- Optimization to a specific query language. This means that the query must be able to exploit datastore specific query features.
- Accommodation of multi-tier architectures. A query must be executed entirely in application memory, delegated to a query engine running in a back-end datastore server, or executed using a combination of processing in the application and datastore server processes.[4]
- Large set support. A query might return a massive number of instances. The query architecture must be able to process the results within the resource constrains of the execution environment.

[4] David Jordan and Craig Russell, "Java Data Objects," p. 146.

- Compiled query support. Parsing a query may be resource intensive. In many applications, the parsing can be done during application development or deployment prior to execution. The query interface must allow for the compilation of queries and bind values to parameters at runtime for optimal execution.

The query can be implemented in two ways. Either by the Persistent Manager or delegated to the database. In the second case, the query will actually be translated to the query language that is supported from the underlying database. In this case, the query might be optimized to take advantage of the particular query language implementation.

As understood from the specifications, JDOQL is a powerful tool. Only the ability to write queries that can run in different underlying datastores could be a great advantage for JDO.

This feature could make the application platform independent or better datastore independent. Thus, it is possible to change the database without needing to make any changes to the code. Fr example, moving from a relational database to an object-oriented database, and keeping the code and specifically the queries exactly the same is feasible.

However, this ability is not without cost. JDOQL queries are very slow as opposed to those executed by JDBC as seen in the chapter regarding the disadvantages of JDO. Moreover, JDOQL, at least in this first edition, (JDO 1.0) does not support all type of queries. GROUP BY queries, for example, are absent.

Thus, the advantage is a real advantage as far as the current JDOQL abilities are enough for the development of a specific application. If more query features are needed, then using a native language such as a pure SQL, for example, for a relational database, is required, which actually becomes a disadvantage of JDO.

Otherwise, the queries using JDOQL are very easy to implement if their syntax is understood, which is not difficult part,. The development of the queries in the *Training-Squadron* application was one of the easiest parts of the project.

The implementation of the queries is very uncomplicated and easy as is the parameterization of the queries. An example of a query found in the application is provided below:

```
PersistenceManager pm=..
Extent extent=pm.getExtent(Flights.class,true);

Query query=pm.newQuery(extent);

Collection result=(Collection) query.execute();
query.close(result);
```

It is possible to execute the same query without using the Extent interface as follows

```
Query query=pm.newQuery(Flights.class );
```

Thus, it is easy to use a parameter for the query. If an object instance of a class exists, it is possible to execute the following:

```
Object myObject= …

Query query=pm.newQuery(  myObject.getClass() );
```

Thus, the result set of an object is achieved with the data instances stored in a datastore, even without knowing the type of object, such as if it is an Instructors instance or a Students instance for example.

Therefore, a strong and sort code is possible that could be easier to manage and could cover many different cases. All the advantages of an object oriented programming are available even if working in a relational DBMS.

The same techniques can be applied to more complex queries. For example, the following query uses a filter:

```
Extent extent=pm.getExtent(Flights.class,true);
String filter="date==d && squadron==squad";
Query query=pm.newQuery(extent,filter);
```

```
query.declareImports("import java.util.Date");
query.declareParameters("Date d, Squadron squad");
Collection result=(Collection) query.execute(d,squad);
query.close(result);
```

It is possible to have a method and simply pass a parameter as a string declaring the filter and the objects representing the values of that string, such as an array of objects since there can be many parameters to the filter string.

What is impressive, however, is the manner in which the queries can access fields of classes that are fields in the current class:

```
 Extent extent=pm.getExtent(Flights.class,true);
String filter="exercise.categories==cat";
Query query=pm.newQuery(extent,filter);

query.declareParameters("Categories cat");
Collection result=(Collection) query.execute(cat);
```

Here, *exercise* is a field in *Flights* class. However, *Exercise* is a class in itself, meaning that *exercise* instances are represented in a different data table.

The *Exercise* class has a field named *category*. The value of the field can be taken from the *Flight* class using the dot (.) operator although *Flight* and *Category*, which is the class of the *category* field of the *Exercise* class, has no direct relation.

Thus, the:

```
exercise.categories==cat
```

returns the *category* class of the *exercise* which is installed in the *Flight* class. With just a simple line, access to three tables is possible. In the application, every *student* belongs to a series and must fly the program that is included in that *series*. *Series* includes

different *Stadio*. *Stadio* includes different *Categories* which, in turn, include the *Exercises* that the *Student* must fly. For example, it is possible to take the *Series* that the *exercise* is a part of simply by doing the following in JDOQL:

```
exercise.categories.stadio.series
```

where:

*exercise.categories* returns the Categories class

*categories.stadio* returns the Stadio class

*and stadio.series* returns the Series class

This simple line could provide data or filter the data using a command

*exercise.categories.stadio.series==series1// series1* an object of *Series* class

and accessing five tables simultaneously. The *Flights* table is included.

Trying to do a similar action using pure SQL commands, of course, will be far more difficult, and result in longer code lines, which is the great advantage of working with relational databases but using an object-oriented approach.

It must be noted that the above implementation of the queries actually violates the security of the classes. Even if the field *categories* in *Exercise* class is declared private, or the *stadio* field in *Categories* class is declared private, or the *series* field in *Stadio* class is declared private, as they actually are,, the code violates the privacy and accesses the value of those fields.

Additional advantages and derivatives from JDOQL is not having to know a second query language, SQL. Finally, since JDOQL works with many databases, it is thus understandable that it is not necessary to know the specifications of the underlying database.

# VI. CONCLUSION

This thesis concludes by summarizing the advantages and disadvantages of JDO by beginning with the advantages:

## A. ADVANTAGES

JDO provides transparent persistence of Java object models in transactional databases. The user can map the java classes to the data of the underlying datastore. Thus, the user works with the object-oriented model even if the application manages data from tables in a relational data model. The handling of objects during the development of an application for database management is far easier than the usual method used and provides many advantages for the implementation.

The automate persistent or persistent by reachability is also a very flexible method for handling a database. JDO provides a means so that the objects become persistent even without declaring them as persistent, which happens when the objects is referenced by a persistent object. Thus, a code will also make the Students s persistent because it is referenced by a persistent object flight f.

```
Students s = new Students(..);
Flights f=new Flights(s,…..);
PersistentManager  pm.makePersistent(f);
```

Thus, the code is minimized and is more efficient for avoiding error prone procedures.

JDOQL also provides flexibility with the underlying database. This means that the query that actually will be executed will be either a SQL query for a relational database, or an object database query such as the Object Query Language (OQL). JDOQL will support all these operations with the same syntax, even if the underlying datastore can be queried using SQL or OQL. It is independent of the query language of the underlying datastore.

Additionally JDOQL can be optimized to a specific query language and can support large datasets. Moreover, JDOQL can be easily parsed. Using parameters can maintain the same code for the execution of a large set of query options.

As a result, there is no need for knowing a separate query language. JDBC, for example, assumes that the programmer knows SQL and can write queries in order to access the data of a relational DBMS, which according to specifications, is not a necessity in JDO. The user can access a database using only one language: Java. However, many aspects in queries have not yet been implemented in the current JDO edition.

This implies that the programmer does not need to know the peculiarities of the underlying datastore. The programmer writes the codes according to the JDO specification and the code is able to run no matter what the underlying database. It is only necessary to change the drivers to the Factory. The result is less code and less time to implement the code. JDO, however, also has a number of disadvantages, some of which are strong enough to negate its advantages.

**B. DISADVANTAGES**

Creating a **many** to **many** relationship is extremely difficult. Even if JDO provides a solution, it is not good enough since the implementation requires four tables instead of three for this kind of relationship. Moreover, this solution of four tables requires additional code for informing all the tables and for verifying the correctness of the insertion of new records. This creates additional problems also concerning the performance of JDO. Accessing four tables instead of three will require more time and JDO is already behind in terms of quickness of returned results.

Even if some vendors can support a managed relationship, which might enable the better representation of a **many** to **many** relationship, this is vendor specific and not a general JDO characteristic. Thus, the code could not be independent of the vendor in this case, which again is a disadvantage.

Also, another important aspect is that JDO requires primitive types to be part of a relationship. This goes against the philosophy of object-oriented programming.

Additionally, JDOQL does not support all types of queries. GROUP BY queries, for example, are absent from JDOQL specification. Either it is necessary to use direct SQL commands using the JDBC interface for a relational database or accept the low performance of JDO providing group results via iteration through a *select* result set. In both cases, the disadvantages fo JDO are still encountered. In the former, the specification that states there is no need for other language since SQL is used is simply discarded, and in the latter, the very low speed, which can be disturbing in a large dataset, must be accepted.

The speed is not good even for simple SELECT clause queries. The retrieveAll() operation, when used, provides the double speed but lags again far behind JDBC and could be a problem for large datasets since JDOQL supports large datasets.

Furthermore, an important problem is that JDO does not support all types of data structures. There are cases where JDO supports some data structures for specific operations while not supporting other data structures for the same operations. LinkedLists, for example, are not supported for a key-inverse mapping. Additionally, no central specification exists of what will or will not be supported. Thus, a data structure for an operation can be used and the situation encountered in which another vendor does not support this data structure for the same specific operation. Then, the compatibility, even binary compatibility, of the code among vendors disappears, and the implementation would become vendor specific.

Another problem is the use of a XML file for the metadata description and for the mapping with the underlying database. Many errors cannot be detected during the compilation of the code. This increases the developing time. It will be better to use a java interface (s) for the same purpose.

Finally, the current JDO edition does not provide efficient metadata. This is a huge disadvantage because it negates the advantages of object oriented programming. Cases exist in which information is provided for the metadata but it is not possible to this information. When creating the metadata file, information is provided about the length of the field but it is not feasible to retrieve this information. No interface with such functionality exists.

For example, for the following:

```
< class  name ="Flights">

   <field name="date">

         <!-- Specify the format of the date -->
               <extension vendor-name="libelis"
                key="sql-mapping"
                value="date, date-only"/>

   </field>
</ class >
```

There was no method to retrieve the information that the date field is of type date-only. Additionally, for the following, no method was found to retrieve the length of the field. This will be very important for the implementation of a form where the edit box that shows the value of the specific field must be of length 50.

```
<field name="field">

    <extension vendor-name="libelis"
               key="sql-mapping"
               value="string,50"/>


</field>
```

A great problem is the absence of open source implementation. A considerable amount of time was spent to discover which implementation to use. Fortunately, with considerable help from the author's hesis advisor, LiDO was used. However, this was one

of the very few solutions found. Again, its free source edition was under license which also must be renewed every six months. Additionally, LiDO has developed the next edition of the other money products, but this next edition has not been provided to the community, which is free. Also, the community edition might not continue to be updated. Then, how is a programmer to learn about the benefits of a product? Who will pay the considerable amount of money to use the JDO vendor edition when not knowing the benefits? In addition, how could the programmer learn of the benefits without the free source practice that characterize Sun products thus far?

Moreover, Sun's reference implementation is very poor in providing incentives for the use of JDO. Nonetheless, the product is very promising. Do not forget that it is currently the first edition available to the market. Usually, the first edition is not very good. The second edition might succeed in solving many of the current problems.

However, what will be the driving factor for this evolutionary process? With all these vendors, the problem that comes to light is the decentralization of the effort which may cause a common solution to be impossible. Then, the likely outcome could be some very good products from strong vendors but which are vendor-specific and far from the general purpose of JDO. This is an undesirable solution since the very concept of JDO is rather interesting and has provided much thus far.

THIS PAGE INTENTIONALLY LEFT BLANK

# APPENDIX. SOURCE CODE

## A. PACKAGE COMPANY

### 1. The Class: Aircraft.java

```
/**
 * <p>Title: Aircraft.java</p>
 * <p>Description:Handles the setter and getter methods of the Aircraft table </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     cs JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */


package company;

import java.util.List;
import java.io.Serializable;

public class Aircraft  {

   private String num;
   private AircraftType type;
   private double totalHours;


   /**NO ARGUMENT CONSTRUCTOR
    */
   public Aircraft(){
      this.totalHours = 0.0;
   }


   /**MAIN CONSTRUCTOR
    */
   public Aircraft(String num,AircraftType type,  double totalHours) {
      this.num = num;
      this.type = type;
      this.totalHours = totalHours;
   }


   /**Returns the length of the fields. The number of the
```

```
 *elements of the array that holds the length of the fields must be
 *equal with the TOTAL number of the class' fields
 *@return the length of the fields
 */

public int[] getFieldLengths(){
    int[] ar={10,20,20};
    return ar;
}


/**represents a getter method. Returns the value of a field
 *in the class
 *@param i the index of the field
 *@return the value of the field
 */
public Object getField(int i){
    switch (i) {

        case 0:
            return num;
        case 1:
            return type;
        case 2:
            return new Double(totalHours);
    }
    return ““;
}


/**represents a setter method. Sets the value of a field
 *in the class
 *@param value the value that the field will be assigned to
 *@param i the index of the field
 *@return the value of the field
 */

public void setFieldValue(int i,Object value){
    //here to make try catch formatException for the numbers
    switch (i) {

        case 0:
            setNum(value.toString());
            break;
        case 1:
            setType( (AircraftType) value  );
```

```
            break;

         case 3:
            setTotalHours(Double.parseDouble(value.toString()) );
            break;

      }

   }

   public String getNum() {
      return num;
   }

   public void setNum(String num) {
      this.num = num;
   }

   public double getTotalHours() {
      return totalHours;
   }

   public void setTotalHours(double hours) {
      this.totalHours = hours;
   }

   public AircraftType getType() {
      return type;
   }
   public void setType(AircraftType type) {
      this.type = type;
   }









   public String toString() {
      return getNum()+" "  + getType();//"Aircraft (" + getNum()+" "  + getType()+")";
   }


}
```

### 2. The Class : AircraftType.java

```
/**
 * <p>Title: AircraftType.java</p>
 * <p>Description:Handles the setter and getter methods of the AircraftType table </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     cs JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */
package company;

import java.util.List;
import java.io.Serializable;

/**NO ARGUMENT CONSTRUCTOR
 */

public class AircraftType  {

   private String type;
   private String role;




   /**MAIN CONSTRUCTOR
    */

   public AircraftType(String type,String role) {
      this.type = type;
      this.role = role;

   }



   public String getType() {
      return type;
   }

   public void setType(String type) {
      this.type = type;
   }

   public String getRole() {
      return role;
```

```
    }

    public void setRole(String role) {
        this.role = role;
    }









    public String toString() {
        return "AircraftType (" + getType()  + ")";
    }


}
```

### 3. The Class: BaseCategory.java

```
BaseCategory
/**
 * <p>Title: BaseCategory.java</p>
 * <p>Description:Handles the setter and getter methods of the BaseCategory table </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     cs JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */
package company;

//import java.util.List;
//import java.util.LinkedList;
import java.io.Serializable;

public class BaseCategory  {
    private String id;
    private String description;


    /**NO ARGUMENT CONSTRUCTOR
     */

    public BaseCategory() {

    }
```

```
/**MAIN CONSTRUCTOR
 */

public BaseCategory(String num,String type) {
    this.id = num;
    this.description = type;

}


/**Returns the length of the fields. The number of the
 *elements of the array that holds the length of the fields must be
 *equal with the TOTAL number of the class' fields
 *@return the length of the fields
 */

public int[] getFieldLengths(){
    int[] ar={10,20};
    return ar;
}


/**represents a getter method. Returns the value of a field
 *in the class
 *@param i the index of the field
 *@return the value of the field
 */

public Object getField(int i){
    switch (i) {

        case 0:
            return id;
        case 1:
            return description;
    }
    return ““;
}


/**represents a setter method. Sets the value of a field
 *in the class
 *@param value the value that the field will be assigned to
 *@param i the index of the field
 *@return the value of the field
```

```
     */

    public void setFieldValue(int i,Object value){
       switch (i) {

          case 0:
             setId(value.toString());
             break;
          case 1:
             setDescription(value.toString());
             break;
       }
    }


    public String getId() {
       return id;
    }

    public void setId(String num) {
       this.id = num;
    }

    public String getDescription() {
       return description;
    }

    public void setDescription(String desc) {
       this.description = desc;
    }







    public String toString() {
       return getId()+"  "+getDescription();
    }


}
```

### 4. The Class: BaseStadio.java

BaseStadio

```
/**
 * <p>Title: BaseStadio.java</p>
 * <p>Description:Handles the setter and getter methods of the BaseStadio table </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     cs JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */
package company;

//import java.util.List;
//import java.util.LinkedList;
import java.io.Serializable;

public class BaseStadio  {
   private String id;
   private String description;


   /**NO ARGUMENT CONSTRUCTOR
    */
   public BaseStadio() {

   }

   /**MAIN CONSTRUCTOR
    */

   public BaseStadio(String num,String type) {
      this.id = num;
      this.description = type;

   }



   /**Returns the length of the fields. The number of the
    *elements of the array that holds the length of the fields must be
    *equal with the TOTAL number of the class' fields
    *@return the length of the fields
    */

   public int[] getFieldLengths(){
```

```
    int[] ar={10,20};
    return ar;
}



/**represents a getter method. Returns the value of a field
 *in the class
 *@param i the index of the field
 *@return the value of the field
 */

public Object getField(int i){
    switch (i) {

        case 0:
            return id;
        case 1:
            return description;
    }
    return ““;
}



/**represents a getter method. Returns the value of a field
 *in the class
 *@param fieldName the name of the field
 *@return the value of the field
 */

public Object getFieldByName(String fieldName){
    if (fieldName.equals(“id”))
        return id;
    else if (fieldName.equals( “description”))
        return description;

    return null;
}



/**represents a setter method. Sets the value of a field
 *in the class
 *@param value the value that the field will be assigned to
 *@param i the index of the field
 *@return the value of the field
```

```
     */

    public void setFieldValue(int i,Object value){
       switch (i) {

          case 0:
             setId(value.toString());
             break;
          case 1:
             setDescription(value.toString());
             break;
       }
    }


    public String getId() {
       return id;
    }

    public void setId(String num) {
       this.id = num;
    }

    public String getDescription() {
       return description;
    }

    public void setDescription(String desc) {
       this.description = desc;
    }







    public String toString() {
       return getId()+"  "+getDescription();
    }


}
```

## 5. The Class: Categories.java

Categories

```
/**
 * <p>Title: Categories.java</p>
 * <p>Description:Handles the setter and getter methods of the Categories table </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     cs JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */
package company;

import javax.swing.*;
import java.util.LinkedList;
import java.io.Serializable;

public class Categories  {
   private Series series;
   private Stadio stadio;

   private BaseCategory catid;
   private String description;
   private LinkedList exercise;


   private String seriesid;
   private String stadioid;
   private String id;



   /**NO ARGUMENT CONSTRUCTOR
    */
   public Categories() {
      seriesid="mal1";
      stadioid="mal2";
   }


   /**MAIN CONSTRUCTOR
    */

   public Categories(Series series, String num,String type) {
      this.series = series;
      this.id = num;
```

```
    this.description = type;

}

//I must to delete this method . JDO must provides the metadata for this
public Class getTheTypeOfTheCollection(String collectionName){
    if (collectionName.equals("exercise"))
        return Exercise.class;

    return null;
}


/**Returns the length of the fields. The number of the
 *elements of the array that holds the length of the fields must be
 *equal with the TOTAL number of the class' fields
 *@return the length of the fields
 */

public int[] getFieldLengths(){
    int[] ar={10,20,20,15,15,10,10,10};
    return ar;
}



/**represents a getter method. Returns the value of a field
 *in the class
 *@param i the index of the field
 *@return the value of the field
 */

public Object getField(int i){
    switch (i) {

        case 0:
            return series;
        case 1:
            return stadio;

        case 2:
            return catid;
        case 3:
            return description;
        case 4:
            return exercise;
```

```
        case 5:
           return seriesid;
        case 6:
           return stadioid;
        case 7:
           return id;

     }
     return ““;
}

//to handle the collections

/**Represents a setter method for the collection fields
 *we can add or delete a value from a collection field
 * according to whether a flag is true of false
 *@param name the name of the collection field
 *@param value the value
 *@param flag the flag when it is true we add otherwise we delete
 */

public void setFieldByName(String name,Object value,boolean flag){
     if(name.equals(“exercise”) ){
        if(flag) addExercise((Exercise) value);
        else deleteExercise((Exercise) value);
     }

}


/**represents a setter method. Sets the value of a field
 *in the class
 *@param value the value that the field will be assigned to
 *@param i the index of the field
 *@return the value of the field
 */

public void setFieldValue(int i,Object value){
     switch (i) {

        case 0:
           setSeries((Series) value);
           break;
        case 1:
           setStadio((Stadio) value);
           break;
```

```
        case 2:
            setCatId((BaseCategory) value);
            break;
        case 3:
            setDescription(value.toString());
            break;
        case 4:
            //
            break;
    }
}

public Series getSeries() {
    return series;
}

public void setSeries(Series ser) {
    if (ser!=null){
        this.seriesid=ser.getId();

        this.series= ser;
    }
}
public Stadio getStadio() {
    return stadio;
}
public void setStadio(Stadio stad) {
    this.stadio=stad;
    if (stad!=null)
        stadioid=stad.getId();


}
public BaseCategory getCatId() {
    return catid;
}

public void setCatId(BaseCategory cat) {
    this.catid = cat;
    if(cat!=null)
        this.id= cat.getId();
}

public String getId() {
    return id;
```

```
    }

    public void setId(String num) {
        this.id = num;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String desc) {
        this.description = desc;
    }

    public LinkedList getExercise() {
        return exercise;
    }

    public void addExercise(Exercise iexercise) {
        if(exercise==null)
            exercise = new LinkedList();
        exercise.add(iexercise);
    }

    public void deleteExercise(Exercise iExercise) {
        if(exercise!=null)
            exercise.remove(iExercise);
    }









    public String toString() {
        if (series!=null && stadio!=null)
            return getId();
        else
            return "(Series null stadio null )"+" "+getId() +" poly malakas-->"+seriesid;
    }


}
```

### 6. The Class: Exercise.java

Exercise

```
/**
 * <p>Title: Exercise.java</p>
 * <p>Description:Handles the setter and getter methods of the Exercise table </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     cs JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */
package company;

//import java.util.List;
import java.io.Serializable;

public class Exercise  {

   private Series series;

   private Stadio stadio;
   private Categories categories;
   private String id;

   private String description;

   private String seriesid;
   private String stadioid;
   private String categoriesid;

   /**NO ARGUMENT CONSTRUCTOR
    */
   public Exercise() {

   }


   /**MAIN CONSTRUCTOR
    */

   public Exercise(Stadio stadio,String num,String type) {

      this.id = num;
      this.description = type;
      this.stadio= stadio;
   }
```

```
/**Returns the length of the fields. The number of the
 *elements of the array that holds the length of the fields must be
 *equal with the TOTAL number of the class' fields
 *@return the length of the fields
 */

public int[] getFieldLengths(){
   int[] ar={10,20,20,15,15,12,15,10};
   return ar;
}



/**represents a getter method. Returns the value of a field
 *in the class
 *@param i the index of the field
 *@return the value of the field
 */

public Object getField(int i){
   switch (i) {

       case 0:
          return series;
       case 1:
          return stadio;
       case 2:
          return categories;

       case 3:
          return id;
       case 4:
          return description;
       case 5:
          return seriesid;

       case 6:
          return stadioid;
       case 7:
          return categoriesid;

   }
   return ““;
```

```
}


/**represents a setter method. Sets the value of a field
 *in the class
 *@param value the value that the field will be assigned to
 *@param i the index of the field
 *@return the value of the field
 */
public void setFieldValue(int i,Object value){
   switch (i) {

       case 0:
          setSeries((Series) value);
          break;
       case 1:
          setStadio((Stadio) value);
          break;
       case 2:
          setCategories((Categories) value);
          break;
       case 3:
          setId(value.toString());
          break;
       case 4:
          setDescription(value.toString());
          break;
   }
}

public Series getSeries() {
   return series;
}

public void setSeries(Series ser) {
   this.series= ser;
   if(ser!=null)
      this.seriesid=ser.getId();
}

public Stadio getStadio() {
   return stadio;
}

public void setStadio(Stadio stad) {
```

```
        this.stadio = stad;
        if(stad!=null)
            this.stadioid=stad.getId();

    }
    public Categories getCategories() {
        return categories;
    }

    public void setCategories(Categories Categorie) {
        this.categories = Categorie;
        if(Categorie!=null)
            categoriesid=Categorie.getId();
        //since there is not referencial integrity
        //Categorie.addExercise(this);
    }

    public String getId() {
        return id;
    }

    public void setId(String num) {
        this.id = num;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String desc) {
        this.description = desc;
    }









    public String toString() {
        return "Exercise (" + getId()  + ")";
    }


}
```

### 7. The Class: Flights.java

Flights

```
/**
 * <p>Title: Flights.java</p>
 * <p>Description:Handles the setter and getter methods of the Flights table </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     cs JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */
package company;

import java.util.List;
import java.util.Date;
import java.io.Serializable;
import javax.swing.*;

public class Flights  {

   private Squadron squadron;
   private Aircraft aircraft;
   private Instructors instructor;
   private Students student;
   private Date date;
   private double endurance;
   private Exercise exercise;
   private SpecialType specialtype;


   /**NO ARGUMENT CONSTRUCTOR
    */
   public Flights(){
      this.endurance=0;
      this.date= new Date();
   }


   /**MAIN CONSTRUCTOR
    */

   public Flights(Aircraft aircraft, Instructors instructor,
   Students student, Date date) {
      this.aircraft = aircraft;
      this.instructor = instructor;
      this.student = student;
```

```
    this.date = date;
    this.endurance=0;

}



/**Returns the length of the fields. The number of the
 *elements of the array that holds the length of the fields must be
 *equal with the TOTAL number of the class' fields
 *@return the length of the fields
 */

public int[] getFieldLengths(){
    int[] ar={10,10,20,20,35,10,10,15};
    return ar;
}

public Object getFieldByName(String fieldName) {//NoSuchFieldException{
    try{
        //Not a beautiful way however, this will have as a result JDO will assign
        //values to the private fields retrieving values from the database
        //otherwise JDO cannot understand the complexity of the following code and will
        //return null values for the fields of this specific object
        String x=""+date;


        return this.getClass().getDeclaredField(fieldName).get(this);
    }
    catch (Exception e){e.printStackTrace();}
    return " ";

}




/**represents a getter method. Returns the value of a field
 *in the class
 *@param i the index of the field
 *@return the value of the field
 */

public Object getField(int i){
    switch (i) {
        case 0:
```

```
            return squadron;

         case 1:
            return aircraft;
         case 2:
            return instructor;
         case 3:
            return student;
         case 4:
            return date;
         case 5:
            return new Double(endurance);
         case 6:
            return exercise;
         case 7:
            return specialtype;


      }
      return ““;
   }
   //to handle the collections
/*       public void setFieldByName(String name,Object value,boolean flag){
                  if(name.equals(“students”) ){
                        if(flag) addStudents((Students) value);
                        else deleteStudents((Students) value);

                  }

      }
 */




   /**represents a setter method. Sets the value of a field
    *in the cass
    *@param value the value that the field will be assigned to
    *@param i the index of the field
    *@return the value of the field
    */

   public void setFieldValue(int i,Object value){
      switch (i) {
         case 0:
```

```
            setSquadron((Squadron) value);
            break;

        case 1:
            setAircraft((Aircraft) value);
            break;
        case 2:
            setInstructor((Instructors) value);
            break;
        case 3:
            setStudent((Students) value);
            break;
        case 4:
            setDate((Date) value );
            break;
        case 5:
            setEndurance(Double.parseDouble(value.toString()) );
            break;
        case 6:
            setExercise((Exercise) value );
            break;
        case 7:
            setSpecialType((SpecialType) value );
            break;


    }

}
public Squadron getSquadron() {
    return squadron;
}

public void setSquadron(Squadron squadron) {
    this.squadron=squadron;
}

public Aircraft getAircraft() {
    return aircraft;
}

public void setAircraft(Aircraft aircraft) {
    this.aircraft=aircraft;
}

public Instructors getInstructor() {
```

```
    return instructor;
}

public void setInstructor(Instructors instructor) {
    this.instructor = instructor;
}

public Students getStudent() {
    return student;
}

public void setStudent(Students student) {
    this.student = student;
}

public double getEndurance() {
    return endurance;
}

public void setEndurance(double endurance) {
    this.endurance = endurance;
}

public Date getDate() {
    return date;
}

public void setDate(Date dates) {
    this.date = dates;
}
public Exercise getExercise() {
    return exercise;
}

public void setExercise(Exercise ex) {
    this.exercise = ex;
}
public SpecialType getSpecialType() {
    return specialtype;
}

public void setSpecialType(SpecialType type) {
    this.specialtype = type;
}
```

```
   public String toString() {
      return “flight (“ +getAircraft()+” “ +  “ “ +
      “  “+getDate()+” “+getEndurance()+”)”;
   }

}
```

### 8. The Class: GroundCourse.java

GroundCourse

```
/**
 * <p>Title: GroundCourse.java</p>
 * <p>Description:Handles the setter and getter methods of the GroundCourse table </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     cs JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */package company;

 //import java.util.*;
 //import java.util.LinkedList;
 import java.io.Serializable;

 public class GroundCourse implements MyInterface {

    private Series series;
    private String id;
    private String description;
    private int hours;
    private boolean done;


    /**NO ARGUMENT CONSTRUCTOR
     */
    public GroundCourse(){

    }


    /**MAIN CONSTRUCTOR
     */

    public GroundCourse(String num,String type) {
       this.id = num;
```

```
   this.description = type;

}

//I must to delete this method . JDO must provides the metadata for this
public Class getTheTypeOfTheCollection(String collectionName){

   return null;
}



/**Returns the length of the fields. The number of the
 *elements of the array that holds the length of the fields must be
 *equal with the TOTAL number of the class' fields
 *@return the length of the fields
 */

public int[] getFieldLengths(){
   int[] ar={10,20,20,15,10};
   return ar;
}



/**represents a getter method. Returns the value of a field
 *in the class
 *@param i the index of the field
 *@return the value of the field
 */

public Object getField(int i){
   switch (i) {
      case 0:
         return series;
      case 1:
         return id;
      case 2:
         return description;
      case 3:
         return new Integer(hours);
      case 4:
         return new Boolean(done);

   }
   return ““;
```

```
}


/**represents a getter method. Returns the value of a field
 *in the class
 *@param fieldName the name of the field
 *@return the value of the field
 */

public Object getFieldByName(String fieldName){
   return null;
}


//to handle the collections
/**Represents a setter method for the collection fields
 *we can add or delete a value from a collection field
 * according to whether a flag is true of false
 *@param name the name of the collection field
 *@param value the value
 *@param flag the flag when it is true we add otherwise we delete
 */
public void setFieldByName(String name,Object value,boolean flag){

}



/**represents a setter method. Sets the value of a field
 *in the cass
 *@param value the value that the field will be assigned to
 *@param i the index of the field
 *@return the value of the field
 */

public void setFieldValue(int i,Object value){
   switch (i) {
      case 0:
         setSeries((Series) value);
         break;
      case 1:
         setId(value.toString());
         break;
      case 2:
         setDescription(value.toString());
```

```
            break;
        case 3:
            setHours(Integer.parseInt(value.toString()));
            break;
        case 4:
            setDone((new Boolean(value.toString())).booleanValue()  );
            break;
    }
}
public Series getSeries() {
    return series;
}

public void setSeries(Series ser) {
    this.series = ser;
}


public String getId() {
    return id;
}

public void setId(String num) {
    this.id = num;
}

public String getDescription() {
    return description;
}

public void setDescription(String desc) {
    this.description = desc;
}

public int getHours() {
    return hours;
}

public void setHours(int desc) {
    this.hours = desc;
}

public boolean getDone() {
    return done;
}
public void setDone(boolean b) {
```

```
        this.done=b;
    }




    public String toString() {
        return “Course (“ + getId()  + “)”;
    }


}
```

### 9. The Class: Instructors.java

Instructors

```
/**
 * <p>Title: Instructors.java</p>
 * <p>Description:Handles the setter and getter methods of the Instructors table </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     cs JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */

package company;

import java.util.List;
import java.util.LinkedList;
import java.io.Serializable;

public class Instructors  {

    private String id;
    private String lname;
    private String fname;
    private Rank rank;
    private boolean absent;

    private double totalHours;


    private int status; //permanent or temporary
    private LinkedList students;
    private Squadron squadron;
```

```
private LinkedList aircrafts;

/**NO ARGUMENT CONSTRUCTOR
 */
public Instructors(){
   this.id="defaultid";
   this.lname ="defaullname";
   this.fname = "defaullfname";
   this.totalHours = 0;
   this.status=10;
   // this.absent=false;
   this.students=new LinkedList();

}

/**MAIN CONSTRUCTOR
 */

public Instructors(String id,String lname,String fname,  double totalHours) {
   this.id=id;
   this.lname = lname;
   this.fname = fname;
   this.totalHours = totalHours;

}


/** Returns the class fo the objects that a collection
 *field holds in it.
 *@param collectionName the name of the field that represents
 * a collection
 *@return the class of the object that the collection holds
 */

public Class getTheTypeOfTheCollection(String collectionName){
   if (collectionName.equals("students"))
      return Students.class;

   return null;
}

/**Returns the length of the fields. The number of the
 *elements of the array that holds the length of the fields must be
 *equal with the TOTAL number of the class' fields
 *@return the length of the fields
 */
```

```
public int[] getFieldLengths(){
   int[] ar={10,20,20,15,10,7,17,10,10,15};
   return ar;
}


/**return the field names of the class
 */
public String[] getMetaData(){
   String[] ar={“id”,”fname”,”lname”,”status”,”totalHours”};
   return ar;
}

/**represents a getter method. Returns the value of a field
 *in the class
 *@param i the index of the field
 *@return the value of the field
 */
public Object getField(int i){
   switch (i) {

       case 0:
          return id;
       case 1:
          return lname;
       case 2:
          return fname;
       case 3:
          return rank;
       case 4:
          return new Boolean(absent);

       case 5:
          return new Double(totalHours);

       case 6:
          return new Integer(  status);
       case 7:
          return students;
       case 8:
          return squadron;
       case 9:
          return aircrafts;


   }
```

```
   return ““;
}
//to handle the collections

/**Represents a setter method for the collection fields
 *we can add or delete a value from a collection field
 * according to whether a flag is true of false
 *@param name the name of the collection field
 *@param value the value
 *@param flag the flag when it is true we add otherwise we delete
 */
public void setFieldByName(String name,Object value,boolean flag){
   if(name.equals(“students”) ){
      if(flag) addStudents((Students) value);
      else deleteStudents((Students) value);
   }
   if(name.equals(“aircrafts”) ){
      if(flag) addAircrafts((Aircraft) value);
      else deleteAircrafts((Aircraft) value);
   }

}


/**represents a setter method. Sets the value of a field
 *in the class
 *@param value the value that the field will be assigned to
 *@param i the index of the field
 *@return the value of the field
 */

public void setFieldValue(int i,Object value){
   switch (i) {

      case 0:
         setId(value.toString());
         break;
      case 1:
         setLastName(value.toString());
         break;
      case 2:
         setFirstName(value.toString());
         break;
      case 3:
         setRank((Rank) value );
         break;
```

```
        case 4:  //this will become false in case of null value
           setAbsent( (new Boolean(value.toString())).booleanValue()  );
           break;

        case 5:
           setTotalHours(Double.parseDouble(value.toString()) );
           break;
        case 6:
           setStatus( Integer.parseInt(value.toString()) );
           break;
        case 7:
           break;
           //setStudents((LinkedList) value) ;
        case 8:
           setSquadron( (Squadron) value  );
           break;

     }

}

public String getId() {
   return id;
}

public void setId(String id) {
   this.id = id;
}

public String getLastName() {
   return lname;
}

public void setLastName(String name) {
   this.lname = name;
}
public String getFirstName() {
   return fname;
}

public void setFirstName(String name) {
   this.fname = name;
}

public Rank getRank() {
   return rank;
```

```
}

public void setRank(Rank r) {
   this.rank = r;
}

public double getTotalHours() {
   return totalHours;
}
public void setTotalHours(double hours) {
   this.totalHours = hours;
}

public boolean getAbsent() {
   return absent;

}

public void setAbsent(boolean absent) {
   this.absent = absent;
}


public int getStatus() {
   return status;
}

public void setStatus(int status) {
   this.status = status;
}

public LinkedList getStudents() {
   return students;
}
public void setStudents(LinkedList listStudents) {
   this.students=listStudents;
}

public void addStudents(Students student) {
   if(students==null)
      students=new LinkedList();

   students.add(student);

   if( student.getInstructors()==null)
      student.addInstructors(this);
```

```
        else if(!student.getInstructors().contains(this))
            student.addInstructors(this);
    }

    public void deleteStudents(Students student) {
        if(students!=null){
            students.remove(student);

            if( student.getInstructors()!=null)
                if(!student.getInstructors().contains(this))
                    student.deleteInstructors(this);

        }
    }


    public Squadron getSquadron() {
        return squadron;
    }
    public void setSquadron(Squadron sq) {
        this.squadron=sq;

    }

    public void addAircrafts(Aircraft student) {
        aircrafts.add(student);

    }

    public void deleteAircrafts(Aircraft student) {
        aircrafts.remove(student);
    }




    public String toString() {
        return getId()+"  "+getLastName();//"Instructor (" + getLastName()  + ")";
    }


}
```

## 10. The Interface: MyInterface.java

MyInterface

```
/**
 * <p>Title: MyInterface.java</p>
 * <p>Description:Dtermines an interface with methods that we need
 * to use for the implementation of the forms that display data for the tables<p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     cs JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */
package company;


public interface  MyInterface  {


   /**Returns the length of the fields. The number of the
    *elements of the array that holds the length of the fields must be
    *equal with the TOTAL number of the class' fields
    *@return the length of the fields
    */
   public int[] getFieldLengths();


   /**represents a getter method. Returns the value of a field
    *in the class
    *@param fieldName the name of the field
    *@return the value of the field
    */
   public Object getFieldByName(String fieldName);


   /**represents a getter method. Returns the value of a field
    *in the class
    *@param i the index of the field
    *@return the value of the field
    */
   public Object getField(int i);


   //to handle the collections
   /**Represents a setter method for the collection fields
    *we can add or delete a value from a collection field
    * according to whether a flag is true of false
```

```
     *@param name the name of the collection field
     *@param value the value
     *@param flag the flag when it is true we add otherwise we delete
     */
    public void setFieldByName(String name,Object value,boolean flag);


    /**represents a setter method. Sets the value of a field
     *in the class
     *@param value the value that the field will be assigned to
     *@param i the index of the field
     *@return the value of the field
     */
    public void setFieldValue(int i,Object value);


    /** Returns the class of the objects that a collection
     *field holds in it.
     *@param collectionName the name of the field that represents
     * a collection
     *@return the class of the object that the collection holds
     */
    public Class getTheTypeOfTheCollection(String collectionName);

}
```

### 11. The Class: Rank.java

Rank

```
/**
 * <p>Title: Rank.java</p>
 * <p>Description:Handles the setter and getter methods of the Rank table </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     cs JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */

package company;

//import java.util.List;
import java.io.Serializable;

public class Rank  {
```

```
private String id;
private String description;
// private Stadio stadio;
//private List exercise;



/**MAIN CONSTRUCTOR
 */

public Rank(String num,String type) {

    this.id = num;
    this.description = type;

}

/**Returns the length of the fields. The number of the
 *elements of the array that holds the length of the fields must be
 *equal with the TOTAL number of the class' fields
 *@return the length of the fields
 */
public int[] getFieldLengths(){
    int[] ar={10,20};
    return ar;
}

/**represents a getter method. Returns the value of a field
 *in the class
 *@param i the index of the field
 *@return the value of the field
 */
public Object getField(int i){
    switch (i) {

        case 0:
            return id;
        case 1:
            return description;

    }
    return ““;
}


/**represents a setter method. Sets the value of a field
```

```
     *in the cass
     *@param value the value that the field will be assigned to
     *@param i the index of the field
     *@return the value of the field
     */
    public void setFieldValue(int i,Object value){
        switch (i) {

            case 0:
                setId(value.toString());
                break;
            case 1:
                setDescription(value.toString());
                break;
        }

    }


    public String getId() {
        return id;
    }

    public void setId(String num) {
        this.id = num;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String desc) {
        this.description = desc;
    }







    public String toString() {
        return "Rank (" + getDescription()  + ")";
    }


}
```

### 12. The Class: Schedule.java

Schedule

```
/**
 * <p>Title: Schedule.java</p>
 * <p>Description:Handles the setter and getter methods of the Schedule table </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     cs JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */

package company;

import javax.swing.*;
import java.util.LinkedList;
import java.io.Serializable;

public class Schedule implements MyInterface {
   private Squadron squadron;
   private int month;
   private int year;

   private double hours;
   private int totalflights;

   private String squadronid;

   /**NO ARGUMENT CONSTRUCTOR
    */

   public Schedule() {
      month=0;
      year=0;
      hours=0;
      totalflights=0;
   }



   //I must to delete this method. JDO must provides the metadata for this
   /** Returns the class of the objects that a collection
    *field holds in it.
    *@param collectionName the name of the field that represents
    * a collection
    *@return the class of the object that the collection holds
```

```
 */
public Class getTheTypeOfTheCollection(String collectionName){
   return null;
}


/**Returns the length of the fields. The number of the
 *elements of the array that holds the length of the fields must be
 *equal with the TOTAL number of the class' fields
 *@return the length of the fields
 */

public int[] getFieldLengths(){
   int[] ar={10,20,20,15,15,10};
   return ar;
}



/**represents a getter method. Returns the value of a field
 *in the class
 *@param fieldName the name of the field
 *@return the value of the field
 */
public Object getFieldByName(String i){
   return null;
}


/**represents a getter method. Returns the value of a field
 *in the class
 *@param i the index of the field
 *@return the value of the field
 */
public Object getField(int i){
   switch (i) {

       case 0:
          return squadron;
       case 1:
          return new Integer(month);

       case 2:
          return new Integer(year);
       case 3:
          return new Double(hours);
```

```
        case 4:
           return new Integer(totalflights);
        case 5:
           return squadronid;

     }
     return "";
}

//to handle the collections
/**Represents a setter method for the collection fields
 *we can add or delete a value from a collection field
 * according to whether a flag is true of false
 *@param name the name of the collection field
 *@param value the value
 *@param flag the flag when it is true we add otherwise we delete
 */
public void setFieldByName(String name,Object value,boolean flag){

}


/**represents a setter method. Sets the value of a field
 *in the class
 *@param value the value that the field will be assigned to
 *@param i the index of the field
 *@return the value of the field
 */
public void setFieldValue(int i,Object value){
   switch (i) {

        case 0:
           setSquadron((Squadron) value);
           break;
        case 1:
           setMonth(Integer.parseInt(value.toString()) );
           break;

        case 2:
           setYear(Integer.parseInt(value.toString()) );
           break;
        case 3:
           setHours(Double.parseDouble(value.toString()) );
           break;
        case 4:
           setTotalFlights(Integer.parseInt(value.toString()) );
```

```
            break;
    }
}

public Squadron getSquadron() {
    return squadron;
}

public void setSquadron(Squadron sq) {
    if (sq!=null){
        this.squadronid=sq.getSquadron();

        this.squadron= sq;
    }
}
public int getMonth() {
    return month;
}
public void setMonth(int m) {
    this.month=m;

}

public int  getYear() {
    return year;
}

public void setYear(int num) {
    this.year= num;
}

public double getHours() {
    return hours;
}

public void setHours(double hour) {
    this.hours = hour;
}

public int  getTotalFlights() {
    return totalflights;
}

public void setTotalFlights(int num) {
    this.totalflights= num;
}
```

```
  public String toString() {
     return getSquadron() +" "+getMonth()+" / "+getYear();
  }


}
```

### 13. The Class: Series.java

Series

```
/**
 * <p>Title: Series.java</p>
 * <p>Description:Handles the setter and getter methods of the Series table </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     cs JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */

package company;

import java.util.*;
import java.util.LinkedList;
import java.io.Serializable;

public class Series  {

    private String id;
    private String description;
    private Collection stadio;
    private LinkedList students;
    private Collection groundcourse;


    /**NO ARGUMENT CONSTRUCTOR
     */

    public Series(){

    }
```

```
/**MAIN CONSTRUCTOR
 */

public Series(String num,String type) {
    this.id = num;
    this.description = type;

}



//I must to delete this method . JDO must provides the metadata for this
/** Returns the class of the objects that a collection
 *field holds in it.
 *@param collectionName the name of the field that represents
 * a collection
 *@return the class of the object that the collection holds
 */
public Class getTheTypeOfTheCollection(String collectionName){
    if (collectionName.equals("students"))
        return Students.class;
    else if (collectionName.equals("stadio"))
        return Stadio.class;
    else if (collectionName.equals("groundcourse"))
        return GroundCourse.class;

    return null;
}


/**Returns the length of the fields. The number of the
 *elements of the array that holds the length of the fields must be
 *equal with the TOTAL number of the class' fields
 *@return the length of the fields
 */

public int[] getFieldLengths(){
    int[] ar={10,20,20,15,10};
    return ar;
}


/**represents a getter method. Returns the value of a field
 *in the class
 *@param i the index of the field
```

```
 *@return the value of the field
 */
public Object getField(int i){
   switch (i) {

      case 0:
         return id;
      case 1:
         return description;
      case 2:
         return stadio;
      case 3:
         return students;
      case 4:
         return groundcourse;

   }
   return ““;
}

//to handle the collections
/**Represents a setter method for the collection fields
 *we can add or delete a value from a collection field
 * according to whether a flag is true of false
 *@param name the name of the collection field
 *@param value the value
 *@param flag the flag when it is true we add otherwise we delete
 */
public void setFieldByName(String name,Object value,boolean flag){
   if(name.equals(“students”) ){
      if(flag) addStudents((Students) value);
      else deleteStudents((Students) value);
   }
   if(name.equals(“stadio”) ){
      if(flag) addStadio((Stadio) value);
      else deleteStadio((Stadio) value);
   }
   if(name.equals(“groundcourse”) ){
      if(flag) addGroundCourse((GroundCourse) value);
      else deleteGroundCourse((GroundCourse) value);
   }

}
```

```
/**represents a setter method. Sets the value of a field
 *in the class
 *@param value the value that the field will be assigned to
 *@param i the index of the field
 *@return the value of the field
 */
public void setFieldValue(int i,Object value){
   switch (i) {

       case 0:
          setId(value.toString());
          break;
       case 1:
          setDescription(value.toString());
          break;
       case 2:
          //
          break;
       case 3:
          //
          break;
   }
}

public String getId() {
   return id;
}

public void setId(String num) {
   this.id = num;
}

public String getDescription() {
   return description;
}

public void setDescription(String desc) {
   this.description = desc;
}

public Collection getStadio() {
   return stadio;
}
public void addStadio(Stadio stadion) {
   if(stadio==null)
      stadio = new LinkedList();
```

```
    stadio.add(stadion);
}

public void deleteStadio(Stadio stadion) {
    if(stadio!=null)
        stadio.remove(stadion);
}


public LinkedList getStudents() {
    return students;
}
public void setStudents(LinkedList listStudents) {
    this.students=listStudents;
}

public void addStudents(Students student) {
    if(students==null)
        students=new LinkedList();

    students.add(student);
}

public void deleteStudents(Students student) {
    students.remove(student);
}

public Collection getGroundCourse() {
    return groundcourse;
}
public void addGroundCourse(GroundCourse groundcours) {
    if(groundcourse==null)
        groundcourse = new LinkedList();
    groundcourse.add(groundcours);
}

public void deleteGroundCourse(GroundCourse groundcours) {
    if(groundcourse!=null)
        groundcourse.remove(groundcours);
}




public String toString() {
    return “Series (“ + getId()  + “)”;
```

```
    }


}
```

### 14. The Class: SpecialType.java

SpecialType

```
/**
 * <p>Title: SpecialType.java</p>
 * <p>Description:Handles the setter and getter methods of the SpecialType table </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     cs JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */

package company;

//import java.util.List;
import java.io.Serializable;

public class SpecialType {

   private String id;

   private String description;

   private String color;


   /**NO ARGUMENT CONSTRUCTOR
    */

   public SpecialType() {

   }



   /**MAIN CONSTRUCTOR
    */

   public SpecialType(String id) {
```

```
    this.id = id;
}



/**Returns the length of the fields. The number of the
 *elements of the array that holds the length of the fields must be
 *equal with the TOTAL number of the class' fields
 *@return the length of the fields
 */
public int[] getFieldLengths(){
    int[] ar={10,20,10};
    return ar;
}


/**represents a getter method. Returns the value of a field
 *in the class
 *@param i the index of the field
 *@return the value of the field
 */
public Object getField(int i){
    switch (i) {

        case 0:
            return id;
        case 1:
            return description;
        case 2:
            return color;

    }
    return ““;
}



/**represents a setter method. Sets the value of a field
 *in the class
 *@param value the value that the field will be assigned to
 *@param i the index of the field
 *@return the value of the field
 */
public void setFieldValue(int i,Object value){
    switch (i) {
```

```
        case 0:
           setId(value.toString());
           break;
        case 2:
           setDescription(value.toString());
           break;
        case 3:
           setColor(value.toString());
           break;

     }
}


public String getId() {
   return id;
}

public void setId(String num) {
   this.id = num;
}

public String getDescription() {
   return description;
}

public void setDescription(String desc) {
   this.description = desc;
}
public String getColor() {
   return color;
}

public void setColor(String color) {
   this.color = color;
}







public String toString() {
   return  getId() ;
}
```

}

### 15. The Class: Squadron.java

Squadron

```
/**
 * <p>Title: Squadron.java</p>
 * <p>Description:Handles the setter and getter methods of the Squadron table </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     cs JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */

package company;

import java.util.*;
import java.io.Serializable;

public class Squadron implements MyInterface {

   private String squadron;
   private String description;
   private Collection schedule;
   //private List exercise;


   /**NO ARGUMENT CONSTRUCTOR
    */

   public Squadron() {
   }


   /**MAIN CONSTRUCTOR
    */

   public Squadron(String num,String type) {

      this.squadron = num;
      this.description = type;

   }
```

```
/**Returns the length of the fields. The number of the
 *elements of the array that holds the length of the fields must be
 *equal with the TOTAL number of the class' fields
 *@return the length of the fields
 */

public int[] getFieldLengths(){
   int[] ar={10,20,30};
   return ar;
}


/**represents a getter method. Returns the value of a field
 *in the class
 *@param i the index of the field
 *@return the value of the field
 */
public Object getField(int i){
   switch (i) {

      case 0:
         return squadron;
      case 1:
         return description;
      case 2:
         return schedule;

   }
   return ““;
}


/**represents a getter method. Returns the value of a field
 *in the class
 *@param fieldName the name of the field
 *@return the value of the field
 */
public Object getFieldByName(String fieldName) {//NoSuchFieldException{
   try{
      //Not a beautiful way however, this will have as a result JDO will assign
      //values to the private fields retrieving values from the database
      //otherwise JDO cannot understand the complexity of the following code and will
      //return null values for the fields of this specific object
      String x=““+squadron;
```

```
        return this.getClass().getDeclaredField(fieldName).get(this);
    }
    catch (Exception e){e.printStackTrace();}
    return " ";

}



//I must to delete this method . JDO must provides the metadata for this
/** Returns the class of the objects that a collection
 *field holds in it.
 *@param collectionName the name of the field that represents
 * a collection
 *@return the class of the object that the collection holds
 */
public Class getTheTypeOfTheCollection(String collectionName){
    if (collectionName.equals("schedule"))
        return Schedule.class;

    return null;
}



//to handle the collections
/**Represents a setter method for the collection fields
 *we can add or delete a value from a collection field
 * according to whether a flag is true of false
 *@param name the name of the collection field
 *@param value the value
 *@param flag the flag when it is true we add otherwise we delete
 */
public void setFieldByName(String name,Object value,boolean flag){
    if(name.equals("schedule") ){
        if(flag) addSchedule((Schedule ) value);
        else deleteSchedule((Schedule) value);
    }

}



/**represents a setter method. Sets the value of a field
```

```
 *in the class
 *@param value the value that the field will be assigned to
 *@param i the index of the field
 *@return the value of the field
 */
public void setFieldValue(int i,Object value){
   switch (i) {

       case 0:
          setSquadron(value.toString());
          break;
       case 1:
          setDescription(value.toString());
          break;

   }
}

public String getSquadron() {
   return squadron;
}

public void setSquadron(String num) {
   this.squadron = num;
}

public String getDescription() {
   return description;
}

public void setDescription(String desc) {
   this.description = desc;
}


public Collection getSchedule() {
   return schedule;
}

public void addSchedule(Schedule schedules) {
   if(schedule==null)
      schedule=new LinkedList();
   schedule.add(schedules);

   //instructor.addStudents(this);
}
```

```
   public void deleteSchedule(Schedule schedules) {
      if(schedule!=null)
         schedule.remove(schedules);

   }



   public String toString() {
      return “Squadron (“ + getSquadron()  + “)”;
   }


}
```

### 16. The Class: Stadio.java

Stadio

```
/**
 * <p>Title: Stadio.java</p>
 * <p>Description:Handles the setter and getter methods of the Stadio table </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     cs JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */

package company;

import java.util.*;
import java.util.LinkedList;
import java.io.Serializable;

public class Stadio  {
   private Series series;
   private BaseStadio sid;
   private String description;
   private Collection categories;

   private String seriesId;
   private String id;

   /**NO ARGUMENT CONSTRUCTOR
    */
```

```
public Stadio() {

}


/**MAIN CONSTRUCTOR
 */

public Stadio(Series series, BaseStadio num,String type) {
    this.series = series;
    this.sid = num;
    this.description = type;

}



//I must to delete this method . JDO must provides the metadata for this
/** Returns the class of the objects that a collection
 *field holds in it.
 *@param collectionName the name of the field that represents
 * a collection
 *@return the class of the object that the collection holds
 */
public Class getTheTypeOfTheCollection(String collectionName){
    if (collectionName.equals("categories"))
        return Categories.class;

    return null;
}



/**Returns the length of the fields. The number of the
 *elements of the array that holds the length of the fields must be
 *equal with the TOTAL number of the class' fields
 *@return the length of the fields
 */

public int[] getFieldLengths(){
    int[] ar={10,20,20,15,10,12};
    return ar;
}
```

```
/**represents a getter method. Returns the value of a field
 *in the class
 *@param i the index of the field
 *@return the value of the field
 */
public Object getField(int i){
   switch (i) {

       case 0:
          return series;
       case 1:
          return sid;
       case 2:
          return description;
       case 3:
          return categories;
       case 4:
          return seriesId;
       case 5:
          return id;



   }
   return ““;
}

//to handle the collections
/**Represents a setter method for the collection fields
 *we can add or delete a value from a collection field
 * according to whether a flag is true of false
 *@param name the name of the collection field
 *@param value the value
 *@param flag the flag when it is true we add otherwise we delete
 */
public void setFieldByName(String name,Object value,boolean flag){
   if(name.equals(“categories”) ){
       if(flag) addCategories((Categories ) value);
       else deleteCategories((Categories ) value);
   }

}



/**represents a setter method. Sets the value of a field
```

```
 *in the class
 *@param value the value that the field will be assigned to
 *@param i the index of the field
 *@return the value of the field
 */
public void setFieldValue(int i,Object value){
   switch (i) {

       case 0:
          setSeries((Series) value);
          break;
       case 1:
          setSId((BaseStadio) value);
          break;
       case 2:
          setDescription(value.toString());
          break;
       case 3:
          //
          break;
   }
}

public Series getSeries() {
   return series;
}

public void setSeries(Series ser) {
   this.series= ser;
   this.seriesId=ser.getId();
}

public String getId() {
   return id;
}
public String getSeriesId() {
   return seriesId;
}


public void setId(String  num) {

   this.id = num;

}
```

```
    public BaseStadio getSId() {
        return sid;
    }

    public void setSId(BaseStadio num) {
        this.sid = num;
        if(num!=null)
            setId(num.getId());

    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String desc) {
        this.description = desc;
    }

    public LinkedList getCategories() {
        return new LinkedList(categories);
    }

    public void addCategories(Categories icategories) {
        if(categories==null)
            categories = new LinkedList();
        categories.add(icategories);
    }

    public void deleteCategories(Categories icategories) {
        if(categories!=null)
            categories.remove(icategories);
    }









    public String toString() {
        return getDescription()+getId();

    }


}
```

### 17. The Class: Students.java

Students

```
/**
 * <p>Title: Students.java</p>
 * <p>Description: Handles the setter and getter methods of the Students table </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     cs JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */

package company;

import java.util.List;
import java.util.LinkedList;
import java.io.Serializable;

public class Students  {

   private String id;
   private String lname;
   private String fname;
   private Rank rank;
   private boolean absent=false;
   private double totalHours;

   private int status;

   private LinkedList instructors;

   private Squadron squadron;

   private Series series;


   //private boolean isManager;

   /**NO ARGUMENT CONSTRUCTOR
    */

   public Students(){
      totalHours=0.0;
   }
```

```
   /**MAIN CONSTRUCTOR
    */

   public Students(String id, String lname,String fname,  double totalHours) {
      this.id=id;
      this.lname = lname;
      this.fname = fname;
      this.totalHours = totalHours;

   }




   /**Returns the length of the fields. The number of the
    *elements of the array that holds the length of the fields must be
    *equal with the TOTAL number of the class' fields
    *@return the length of the fields
    */
   public int[] getFieldLengths(){
      int[] ar={10,20,20,15,7,17,7,10,10,10};
      return ar;
   }

   //poly shmantiko . An den zhthsv th timh ths linkedlist to pedio ua exei null otan pav
na to
   //diabasv tote kanei prosbash. An exv quoted to //LinkedList l=getInstructors(); pernv
null gia
   //tous instructors
   /**represents a getter method. Returns the value of a field
    *in the class
    *@param fieldName the name of the field
    *@return the value of the field
    */

   public Object getFieldByName(String fieldName) {//NoSuchFieldException{
      try{
         //Not a beautiful way however, this will have as a result: JDO will assign
         //values to the private fields retrieving values from the  database
         //otherwhise JDO cannot understand the complexity of the following code and
will
         //return null values for the fields of this specific object
         String x=""+totalHours;
         //LinkedList l=getInstructors();
         return this.getClass().getDeclaredField(fieldName).get(this);
      }
```

```
    catch (Exception e){e.printStackTrace();}
    return “malaka”;

}


/** Returns the class of the objects that a collection
 *field holds in it.
 *@param collectionName the name of the field that represents
 * a collection
 *@return the class of the object that the collection holds
 */

public Class getTheTypeOfTheCollection(String collectionName){
    if (collectionName.equals(“instructors”))
        return Instructors.class;

    return null;
}



//to handle the collections
/**Represents a setter method for the collection fields
 *we can add or delete a value from a collection field
 * according to whether a flag is true of false
 *@param name the name of the collection field
 *@param value the value
 *@param flag the flag when it is true we add otherwise we delete
 */
public void setFieldByName(String name,Object value,boolean flag){
    if(name.equals(“instructors”) ){
        if(flag) addInstructors((Instructors) value);
        else deleteInstructors((Instructors) value);
    }

}


/**represents a setter method. Sets the value of a field
 *in the cass
 *@param value the value that the field will be assigned to
 *@param i the index of the field
 *@return the value of the field
 */
public void setFieldValue(int i,Object value){
```

```
    switch (i) {

        case 0:
            setId(value.toString());
            break;
        case 1:
            setLastName(value.toString());
            break;
        case 2:
            setFirstName(value.toString());
            break;
        case 3:
            setRank((Rank) value );
            break;
        case 4:  //this will become false in case of null value
            setAbsent( (new Boolean(value.toString())).booleanValue()  );
            break;

        case 5:
            setTotalHours(Double.parseDouble(value.toString()) );
            break;
        case 6:
            setStatus( Integer.parseInt(value.toString()) );
            break;
        case 7:
            break;
            //setStudents((LinkedList) value) ;
        case 8:
            setSquadron( (Squadron) value  );
            break;
        case 9:
            setSeries( (Series) value  );
            break;

    }
}


/**represents a getter method. Returns the value of a field
 *in the class
 *@param i the index of the field
 *@return the value of the field
 */
public Object getField(int i){
    switch (i) {
```

```
        case 0:
           return id;
        case 1:
           return lname;
        case 2:
           return fname;
        case 3:
           return rank;
        case 4:
           return new Boolean(absent);
        case 5:
           return new Double(totalHours);
        case 6:
           return new Integer(  status);
        case 7:
           return instructors;

        case 8:
           return squadron;
        case 9:
           return series;

     }
     return ““;
}

public String getId() {
     return id;
}

public void setId(String id) {
     this.id = id;
}

public String getLastName() {
     return lname;
}

public void setLastName(String name) {
     this.lname = name;
}
public String getFirstName() {
     return fname;
}

public void setFirstName(String name) {
```

```
    this.fname = name;
}
public Rank getRank() {
    return rank;
}

public void setRank(Rank r) {
    this.rank = r;
}

public double getTotalHours() {
    return totalHours;
}

public void setTotalHours(double hours) {
    this.totalHours = hours;
}

public boolean getAbsent() {
    return absent;

}

public void setAbsent(boolean absent) {
    this.absent = absent;
}

public int getStatus() {
    return status;
}

public void setStatus(int status) {
    this.status = status;
}

public Squadron getSquadron() {
    return squadron;

}

public void setSquadron(Squadron squadron) {
    this.squadron = squadron;
}

public LinkedList getInstructors() {
    return instructors;
```

```
    }

    public void addInstructors(Instructors instructor) {
        if(instructors==null)
            instructors=new LinkedList();

        instructors.add(instructor);

        if(instructor.getStudents()==null)
            instructor.addStudents(this);
        else if( !instructor.getStudents().contains(this))
            instructor.addStudents(this);

    }

    public void deleteInstructors(Instructors instructor) {
        if(instructors!=null){
            instructors.remove(instructor);

            if(instructor.getStudents()!=null)
                if(instructor.getStudents().contains(this))
                    instructor.deleteStudents(this);
        }
    }

    public Series getSeries() {
        return series;

    }

    public void setSeries(Series serie) {
        this.series = serie;
    }




    public String toString() {
        return getId()+"       "+getLastName();//"Student ("  +  getLastName()  +"
"+getFirstName()+ ")";
    }
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////

}
```

## B. PACKAGE TEST

### 1. The Class: InstructorsForm.java

InstructorsForm

```
/**
 * <p>Title: InstructorsForm.java</p>
 * <p>Description: Displays the data of the underlying datastore</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     CS JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */

package test;

import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManagerFactory;
import javax.jdo.PersistenceManager;
import javax.jdo.Transaction;
import javax.jdo.Query;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import javax.jdo.spi.JDOImplHelper;
import javax.jdo.Extent;

import company.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.JComboBox;

import java.util.*;
import java.text.DecimalFormat;


class InstructorsForm extends JPanel implements ActionListener, MouseListener,
ItemListener{
      /* The persistent manager Factory
       */
   private static PersistenceManagerFactory pmf;

      /* The persistent manager
       */
   private static PersistenceManager pm;
```

```
   /* The Transaction
    */
private static Transaction tx;

   /* The query variable
    */
private static Query q;

   /* The source of the data for the form
    */
private   Collection result;

   /* The iterator that we will use to iterate through the data
    */
ListIterator iterator ;

   /* Variable that holds every time the current record when we
    * iterate through the instances of the result Collection.
    */
Object currentRecord;

   /*represent the instance of the object that we are going to display
    * its data to the form. If for example we are going to display records
    *from the students table this will be an instance of the Students class
    */
Object mainFormObject;

   /* Use for entering data to a subform
    */
private static JList subFormList;

   /* Holds the subform data
    */
JPanel[]    listPanel;

   /* For selecting data for various field
    */
JList[]    popUpListBoxes;

   /* The name of the fields of the underlying table
    */
String[]    mainFieldNames;

   /* The toolbar
    */
```

```
JToolBar    mainToolbar;

   /* Hold the main data
    */
JPanel  mainFormPanelData;

   /* The JDOImplHelper
    */
JDOImplHelper myJDOImplHelper;

   /* Hold the name of the fields for which we want to
    * use default values
    */
String[] defaultValueName;

   /* Holds the default values for the fields
    */
Object[] defaultValueValue;

   /* Holds the linked field name
    */
String Name;

   /* Constant for the PREVIOUS button
    */
final int  PREVIOUS=0;

   /* Constant for the NEXT button
    */
final int  NEXT    =1;

   /* Constant for the DELETE button
    */
final int  DELETE=2;

   /* Constant for the NEW button
    */
final int  NEW=3;

   /* Constant for the SAVE button
    */
final int  SAVE=4;

   /* Constant for the UPDATE button
    */
final int  UPDATE=5;
```

```
   /* Constant for the FIRST button
    */
final int  FIRST=6;

   /* Constant for the LAST button
    */
final int  LAST=7;


   /* Holds the buttons
    */

JButton[]mainToolBarButton= new JButton[8];

   /* In case of a subform point to the parent form
    */
InstructorsForm parentForm;

   /* Used to indicate the number of records
    */
JLabel numberOfRecordLabel=new JLabel("0");

/**
 * JComponent array holds the data for every field of a table
 * in the datastore
 */
private JComponent[]    mainFelds;

/**
 * JTextField array holds the data for every field of a linked table
 * (subform) in the datastore
 */
private JTextField[][]    subFields;


/**
 * The top level panel used in showing a dialog using JOptionPane's
 * class method showOptionDialog.
 */
private JPanel         topPanel;

/**
 * Default frame width
 */
private static final int FRAME_WIDTH    = 800;
```

```
/**
 * Default frame height
 */
private static final int FRAME_HEIGHT   = 480;

/**
 * X coordinate of the frame default origin point
 */
private static final int FRAME_X_ORIGIN = 150;

/**
 * Y coordinate of the frame default origin point
 */
private static final int FRAME_Y_ORIGIN = 250;


/**
 * constructs a form and displays the data
 * @param pms the Persistent Manager
 * @param result the LinkedList that contains the data
 * we are going to display
 * @param mainObj an instance of the object that
 * represents the data we are going to display
 */

public InstructorsForm(

PersistenceManager pms,

LinkedList result,

Object mainObj //represents the object of the database that
//we will handle

) {

   this.pm = pms;
   tx = pm.currentTransaction();

   this.mainFormObject=mainObj;
   this.result=result;
   if (result!=null)
      iterator= result.listIterator();
   else
      JOptionPane.showMessageDialog( null,"form is empty of records");
```

```
    myJDOImplHelper=JDOImplHelper.getInstance();


    //try to get the lengths of the fields
    int[]fieldLength=getTheLengthOftheFields();


    //get the name of the fields
    mainFieldNames=myJDOImplHelper.getFieldNames(mainFormObject.getClass());


    listPanel=new JPanel[fieldLength.length];
    for(int i=0;i<fieldLength.length;i++)
       listPanel[i]=new JPanel();
    popUpListBoxes=new JList[fieldLength.length];

    //finaly create the form to handle the data
    topPanel=adjustThePanel();


    mainFormPanelData=new JPanel(new GridLayout( 0,1,5,5 ));


    mainFormPanelData.add(createMainFormPanel(
    mainFieldNames,          //the name of the fields (metadata)

    fieldLength,             //the length of the fields

    myJDOImplHelper.getFieldTypes(mainObj.getClass())          //type of the fields
    )
    );

    JScrollPane jsp=new JScrollPane(mainFormPanelData);

    topPanel.add(jsp,BorderLayout.CENTER);
    this.add(topPanel);



}

/**
 * constructs an initial form. Then we can modify
 * the way the form will display the data
 * @param pms the Persistent Manager
```

```
 * @param result the LinkedList that contains the data
 * we are going to display
 * @param mainObj an instance of the object that
 * represents the data we are going to display
 *@param i integer to declare the difference with the previous
 *constructor
 */


public InstructorsForm(

PersistenceManager pms,

LinkedList result,

Object mainObj ,              //represents the object of the database that
//we will handle
int i            //do nothing

) {

   this.pm = pms;
   tx = pm.currentTransaction();

   this.mainFormObject=mainObj;
   this.result=result;
   if (result!=null)
      iterator= result.listIterator();
   else
      JOptionPane.showMessageDialog( null,"form is empty of records");

   myJDOImplHelper=JDOImplHelper.getInstance();


   //get the name of the fields
   mainFieldNames=myJDOImplHelper.getFieldNames(mainFormObject.getClass());



   mainFelds = new JComponent [mainFieldNames.length];

   JComboBox FormInComboBoxView;
   if (result!=null)
      FormInComboBoxView=new JComboBox(result.toArray());
   else
      FormInComboBoxView=new JComboBox();
```

```
    FormInComboBoxView.addItemListener(this) ;



    this.add(FormInComboBoxView);

    //show the data
    //refreshRecord();


}
/**
 * constructs a form without data
 * @param pms the Persistent Manager
 * @param mainObj an instance of the object that
 * represents the data we are going to display
 *
 */
public InstructorsForm(

PersistenceManager pms,

Object mainObj //represents the object of the database that
//we will handle

) {

    this.pm = pms;
    tx = pm.currentTransaction();

    this.mainFormObject=mainObj;

    myJDOImplHelper=JDOImplHelper.getInstance();

    //try to get the lengths of the fields
    int[]fieldLength=getTheLengthOftheFields();

    //get the name of the fields
    mainFieldNames=myJDOImplHelper.getFieldNames(mainFormObject.getClass());

    listPanel=new JPanel[fieldLength.length];
    for(int i=0;i<fieldLength.length;i++)
       listPanel[i]=new JPanel();
    popUpListBoxes=new JList[fieldLength.length];
```

```
    //finally create the form to handle the data
    topPanel=adjustThePanel();


    mainFormPanelData=new JPanel(new GridLayout( 0,1,5,5 ));


    mainFormPanelData.add(createMainFormPanel(
    mainFieldNames,          //the name of the fields (metadata)

    fieldLength,             //the length of the fields

    myJDOImplHelper.getFieldTypes(mainObj.getClass())          //type of the fields
    )
    );


    JScrollPane jsp=new JScrollPane(mainFormPanelData);

    topPanel.add(jsp,BorderLayout.CENTER);
    this.add(topPanel);

}

/**
 * constructs an initial form and a subform for
 * a specific linkedfield of the form
 * @param pms the Persistent Manager
 * @param result the LinkedList that contains the data
 * of the parent form we are going to display
 * @param mainObj an instance of the object that
 * represents the data of the parent form we are going to display
 * @param nameSubformField the name of the field for which we are going
 * to create a subform
 * @param sameSubformField an instance of the object that
 * represents the data of the subform we are going to display
 *constructor
 */

public InstructorsForm(

PersistenceManager pms,

LinkedList result,

Object mainObj ,              //represents the object of the database that
```

```
//we will handle
String nameSubformField ,//the name of the field that will create a same subform
Object sameSubformField// the object that represent the name
) {

    this.pm = pms;
    tx = pm.currentTransaction();

    this.mainFormObject=mainObj;
    this.result=result;
    if (result!=null)
       iterator= result.listIterator();
    else
       JOptionPane.showMessageDialog( null,"form is empty of records");

    myJDOImplHelper=JDOImplHelper.getInstance();


    //try to get the lengths of the fields
    int[]fieldLength=getTheLengthOftheFields();


    //get the name of the fields
    mainFieldNames=myJDOImplHelper.getFieldNames(mainFormObject.getClass());


    listPanel=new JPanel[fieldLength.length];
    for(int i=0;i<fieldLength.length;i++)
       listPanel[i]=new JPanel();
    popUpListBoxes=new JList[fieldLength.length];

    //finaly create the form to handle the data
    this.setLayout(new BorderLayout());
    this.setBackground( Color.white );

    //create a panel to hold the main form fields

    //create and add the command toolbar to handle the data
    mainToolbar= createToolBar( );
    this.add(mainToolbar, BorderLayout.SOUTH);

    mainFormPanelData=new JPanel(new GridLayout( 0,1,5,5 ));


    mainFormPanelData.add(createMainFormPanel(
    mainFieldNames,          //the name of the fields (metadata)
```

```
    fieldLength,                  //the length of the fields

    myJDOImplHelper.getFieldTypes(mainObj.getClass())          //type of the fields
    )
    );



    for(int i=0;i<mainFieldNames.length;i++)
       if(myJDOImplHelper.getFieldFlags(mainFormObject.getClass())[i]==8)//asfaleia
          if(mainFieldNames[i].equals(nameSubformField)){

             mainFelds[i] = new InstructorsForm( pm, sameSubformField);

             mainFormPanelData.add(mainFelds[i] );
          }

    JScrollPane jsp=new JScrollPane(mainFormPanelData);
    this.add(jsp,BorderLayout.CENTER);
}

/**Sets the parent form for a subform
 * @param parentf the parent form
 */
public void setParentForm(InstructorsForm parentf){
    this.parentForm=parentf;
}

/**Returns the parent form of a subform
 * @return  the parent form of a subform
 */
public InstructorsForm getParentForm(){
    return parentForm;
}

/**Returns the current record of a form
 * @return  the current record of a form
 */
public Object getCurrentRecord(){
    return currentRecord;
}
/**Set the name of the field that links the
 * parent form and the subform
 * @param name  the name of the field
```

```
    */
   public void setTheLinkedFieldName(String name){
      Name=name;
   }

   ////////////////////////////////////////////////////////////////////////////////

   /**Creates a subform for a field
    * @param nameSubformField the name of the field
    * @param subformObject the object that the subform field represents
    */
   public void setSubformForTheField(String nameSubformField, Object
subformObject){

      for(int i=0;i<mainFieldNames.length;i++)

if(myJDOImplHelper.getFieldFlags(mainFormObject.getClass())[i]==10)//asfaleia --
>collection
            if(mainFieldNames[i].equals(nameSubformField)){
               InstructorsForm subform=new InstructorsForm( pm, subformObject);
               subform.setParentForm(this);
               subform.setTheLinkedFieldName(mainFieldNames[i]);
               mainFelds[i] = subform;


                              /*InstructorsForm sf=subform;
                              while (sf.getParentForm()!=null)
                                sf=sf.getParentForm();
                               */
            }


   }

   /**Sets the datalayout for the display of the data
    * in a form
    * @param mgr the Layout Manager
    */
   public  void setMainFormPanelDataLayout(LayoutManager mgr){
      mainFormPanelData.setLayout( mgr);
   }
   ////////////////////////////////////////////////////////////////////
   /**Disables the editing of some fields in the form
    * @param ar an array containing the name of the fields for
    * which we want to disable the editing
    */
```

```
public void disableTheData(String[] ar){
   for (int i = 0; i < mainFelds.length; i++)
      for (int j = 0; j < ar.length; j++)
         if(ar[j].equals(mainFieldNames[i])  )
            mainFelds[i].setEnabled(false);


}
//////////////////////////////////////////////////////////////////
/**Return the object that displays the data of a particular
 * field.
 * @param ar the name of the field
 * @return the container of that field
 */
public Object getTheFieldContainerByFieldName(String  ar){
   for (int i = 0; i < mainFelds.length; i++)
      if(ar.equals(mainFieldNames[i])  )
         return mainFelds[i] ;

   return null;
}

//////////////////////////////////////////////////////////////////////////////////
/**Sets the source of the data for a form. i.e.
 * the data that the form will display
 * @param sresult the source of the data
 */
public void setMainData(LinkedList sresult){

   this.result=sresult;
   if (sresult!=null)
      iterator= sresult.listIterator();
   else
      JOptionPane.showMessageDialog( null,"form is empty of records");


}
/**Gets the data for a form according to a filter
 * and the string that holds the variables
 * @param dataclass the class of the object that it will be displayed to
 * the form
 * @param filter the filter for retrieving the data
 * @param queryVariables the parameters for retrieving the data
 * @param theObject an instance of the object of the data that we are going to
 * display
 */
```

```
   public  void getTheMainData(Class dataclass,String filter, String queryVariables,
Object theObject){
      LinkedList sresult;
      tx.begin();
      q = pm.newQuery(dataclass);
      q.declareParameters(queryVariables);
      q.setFilter(filter);

      sresult = new LinkedList( (Collection) q.execute(theObject) );

      tx.commit();

      this.result=sresult;
      if (sresult!=null)
         iterator= sresult.listIterator();
      else
         JOptionPane.showMessageDialog( null,"form is empty of records");


   }
   /**
    *Adjusts the panel
    *@return the panel
    */

   private JPanel adjustThePanel(){
      JPanel top=new JPanel(new BorderLayout());

      //create a panel to hold the main form fields

      //create and add the command toolbar to handle the data
      mainToolbar= createToolBar( );
      top.add(mainToolbar, BorderLayout.SOUTH);

      return top;

   }


/* Creates the form panel that displays the data of the fields for
 * every record
 *@param fieldNames contains the name of the fields
 *@param fieldLength contains the length of the fields
 *@param fieldtypes contains the type of the fields (if its
 * is a collection, a key etc.
 *@return the panel that displays the data of the fields
```

```
 */

   private JPanel createMainFormPanel(String[] fieldNames, int[] fieldLength, Class[]
fieldtypes){
      JPanel        mainFormPanel = new JPanel( );
      mainFormPanel.setLayout( new BoxLayout( mainFormPanel, BoxLayout.X_AXIS )
);

      //panels for the fields and their labels
      JPanel      mainFormLabelPanel= new JPanel(new GridLayout( 0,1,5,5 ));
      JPanel      mainFormfieldPanel= new JPanel(new GridLayout( 0,1,5,5 ));

      JLabel[]    labels;



      //create and add the labels and the textFields
      mainFelds = new JComponent [fieldNames.length];
      labels = new JLabel[fieldNames.length];

      JPanel psubdata=new JPanel(new GridLayout( 0,1,25,25 ));




      for (int i = 0; i < fieldNames.length; i++) {
         labels[i] = new JLabel( fieldNames[i], JLabel.LEFT);



         JPanel p=new JPanel(new FlowLayout(FlowLayout.LEFT));


         //FieldFlag =10 when the field is a collection
         if(myJDOImplHelper.getFieldFlags(mainFormObject.getClass())[i]==10){
            //the panel that will hold data for the subform
            //we will have one subform for every collection field
            JPanel subpane=createSubFormPanel( i, fieldNames[i]);
            subpane.setPreferredSize(new Dimension(80, 150) );
            psubdata.add( subpane );

         }
         else if(fieldtypes[i].getName().equals("boolean"))
            mainFelds[i] = new JCheckBox();

         else if(fieldtypes[i].getName().equals("java.util.Date"))
```

```
            mainFelds[i] =new  JSpinner(new SpinnerDateModel()) ;

         else{
            String mypackage=mainFormObject.getClass().getPackage().getName() ;
            if(fieldtypes[i].getName().indexOf(mypackage+".")>-1)
               //see if we need some restrictions to the query here
               mainFelds[i]                =                new                JComboBox(
getTheDataForComboBoxes(fieldtypes[i]).toArray() );

            else
               mainFelds[i] = new JTextField( fieldLength[i]);
         }

         // only if the field is not a collection add the field and its label
         // to the mainform's panel
         if(myJDOImplHelper.getFieldFlags(mainFormObject.getClass())[i]!=10){
            p.add(mainFelds[i]);
            JPanel l=new JPanel(new FlowLayout(FlowLayout.LEFT));
            l.add(labels[i]);

            //make a button to filter the record using the key of the record
            if(myJDOImplHelper.getFieldFlags(mainFormObject.getClass())[i]==8){
               JButton filterButton=     new JButton("&");
               filterButton.addActionListener(this);
               l.add(filterButton);
            }

            mainFormLabelPanel.add(l );
            mainFormfieldPanel.add(p );
         }
      }




      // add all to the mainForm's panel
      mainFormPanel.setLayout( new BoxLayout( mainFormPanel, BoxLayout.X_AXIS )
);
      mainFormPanel.add( mainFormLabelPanel );
      mainFormPanel.add( mainFormfieldPanel );
      mainFormPanel.add( psubdata );
      return mainFormPanel;
   }
```

```
   /** Creates a subform for a linked field
    * @param i tyhe index of the field
    * @param theNameOfTheField the name of the field
    * @return a subform for the field
    */
   public JPanel createSubFormPanel(int i, String theNameOfTheField){
      //create the panel that will hold the subform data
      JPanel psub=new JPanel(new  BorderLayout());
      //this must be label[i]
      psub.add(        (new     JPanel()).add(new     JLabel(     theNameOfTheField)     )     ,
BorderLayout.NORTH );

      JList jl=new JList();
      jl.setName(""+i);// for use by the Mouse listener
      mainFelds[i] = jl;
      mainFelds[i].addMouseListener(this);

      JScrollPane scrollpane = new JScrollPane(mainFelds[i]);


      psub.add(scrollpane,BorderLayout.CENTER);

      //create and add the subForm's toolbar
      JToolBar    subFormToolbar= createToolBar2( );

      psub.add(subFormToolbar, BorderLayout.SOUTH);

      //the list that will provide data for the subform
      popUpListBoxes[i]=new
JList(getTheDataForListBoxes(theNameOfTheField).toArray());
      listPanel[i].add(popUpListBoxes[i]);

      psub.setPreferredSize(new Dimension(80, 150) );
      return psub;
   }




   /**Get the data that we will use for the subforms
    * @param dataclassName the class of the field
    * that represents a subform
    * @return the data for the subform */
   public  Collection getTheDataForListBoxes(String dataclassName){
      Class[] rc={String.class};
```

```
        Object[] obi={ dataclassName };

        try{
           //I use reflection here to find out the class type of the collection
           //i.e. what kind of objects are included in the collection field.
           //Unfortunately JDO does not provide this information so I must use
           // a method in every class that will return the class of the object
           //stored in the collection.
           java.lang.reflect.Method
meth=mainFormObject.getClass().getMethod("getTheTypeOfTheCollection", rc);
           Object returnValue=meth.invoke(mainFormObject,obi);

           Collection sresult;
           boolean wasActive=true;
           if (returnValue!=null){

              if(!tx.isActive())
                 tx.begin();

              q = pm.newQuery((Class)returnValue);
              sresult = (Collection) q.execute();
              tx.commit();

              return sresult;
           }

        }
        catch( NoSuchMethodException e){e.printStackTrace();
        printMessageNoSuchMethod(dataclassName);}
        catch( IllegalAccessException ed){ed.printStackTrace();}
        catch( java.lang.reflect.InvocationTargetException ex){ex.printStackTrace();}

        return new LinkedList();
     }

     /** Return the data that we store in comboboxes and can select them
      * to assign value to a specific field of a table
      * @param dataclass the class of the field
      * @return the data of the table that represents the field
      */
     public static Collection getTheDataForComboBoxes(Class dataclass){
        Collection sresult;

        if(!tx.isActive())
           tx.begin();
```

```
    q = pm.newQuery(dataclass);
    sresult = (Collection) q.execute();

    tx.commit();

    return sresult;

}
//////////////////////////////////////////////////////////////

/** Creates a subformpanel
 * @param subFormMetadata the name of the fields for the
 * subform
 * @return the subform panel
 */
private JPanel createSubFormPanel(String[] subFormMetadata){

    JPanel      subFormPanel=new  JPanel();
    subFormPanel.setBackground( Color.blue );

    subFields = new JTextField[6][subFormMetadata.length];

    JLabel[] labels = new JLabel[subFormMetadata.length];
    //create and add the labels
    subFormPanel.setLayout( new GridLayout( 0,2,0,5 ) );
    for (int i = 0; i < subFormMetadata.length; i++) {
       labels[i] = new JLabel( subFormMetadata[i], JLabel.LEFT);
       subFormPanel.add( labels[i] );
    }

    //create and add the subForm's fields
    for (int k = 0; k < 6; k++)
       for (int i = 0; i < subFormMetadata.length; i++) {
          subFields[k][i] = new JTextField( 20 +i);
          subFields[k][i].addMouseListener(this);
          subFormPanel.add( subFields[k][i] );
       }
    //create and add the subForm's toolbar
    JToolBar    subFormToolbar= createToolBar2( );

    JPanel      basesubFormPanel=new  JPanel(new BorderLayout());
    basesubFormPanel.add(subFormPanel,BorderLayout.CENTER);
    basesubFormPanel.add(new JPanel(),BorderLayout.WEST);
    basesubFormPanel.add(new JPanel(),BorderLayout.EAST);
    basesubFormPanel.add(new JPanel(),BorderLayout.NORTH);
    basesubFormPanel.add(subFormToolbar,BorderLayout.SOUTH);
```

```
    return basesubFormPanel;

  }
  ////////////////////////////////////////////////////////////////////////
  //---------------------------------
  //     Main method
  //---------------------------------
  public static void main(String[] args) {


    try {
      pmf = getPMF( "lido_mysql.properties");
      pm = pmf.getPersistenceManager();
      tx = pm.currentTransaction();

      Instructors object=new Instructors();
      String[] instructorsMetaData={"id","LastName"};


      InstructorsForm sf=new InstructorsForm( pm,   object );

      JFrame myframe=new JFrame();
      Container contentPane=myframe.getContentPane( );
      contentPane.setLayout(new BorderLayout());
      JPanel head= new JPanel();
      head.add(new
JComboBox(getTheDataForComboBoxes(Squadron.class).toArray()));
      contentPane.add(head, BorderLayout.NORTH );

      contentPane.add(sf, BorderLayout.CENTER );

      myframe.pack();
      myframe.setVisible(true);
      //pm.close();
      // LiDOHelper.close(pmf);

    } catch (Exception e) {
      e.printStackTrace();
    }

  }


  /**Return all the records from the Instructors
   * @param results the total amount of records
   * @return the results
```

```
 */
public static Collection getAllInstructorsResult(Collection results) {
   String answer="";

   tx.begin();
   q = pm.newQuery(Instructors.class);
   results = (Collection) q.execute();
   //System.out.println("Load all Instructors (" + result.size() + ")");


   tx.commit();
   return results;
}


//---------------------------------
//    actionPerformed
//---------------------------------

/**
 * @param event  */
public void actionPerformed(ActionEvent event){
   if (event.getActionCommand().equals("&"))

      findRecordByKey();
}


/**Depreciated chooses an instructor from a list
 * used for entering data to a subform
 * @param i determines the ist that it will be displayed
 * @return the component that holds the Instructors data
 */
public JComponent[] chooseInstructor(int i){

   String[] buttonLabel = { "Enter", "Delete" };

   int result = JOptionPane.showOptionDialog( null, listPanel[i], "Instructors
Information",
   JOptionPane.DEFAULT_OPTION,
   JOptionPane.QUESTION_MESSAGE, null,
   buttonLabel, buttonLabel[0]    );
   Object isd=popUpListBoxes[i].getSelectedValue() ;
   if(isd!=null&& currentRecord!=null)
      if (result == JOptionPane.OK_OPTION) {
         try{
```

```
                tx.begin();
                pm.makePersistent(currentRecord);

                Class[]                                    classParameters={String.class,(new
Object()).getClass(),boolean.class};//class parameters of the method
                //i.e. “students”
                Object[]    ob={mainFieldNames[i],isd,new    Boolean(true)    };    //value
parameters
                java.lang.reflect.Method
meth=currentRecord.getClass().getMethod(“setFieldByName”, classParameters);
                meth.invoke(currentRecord,ob);
                tx.commit();

                showInstructors();
              }
              catch(Exception e){e.printStackTrace();}


              //  return mainFelds;
            } else {
              try{
                tx.begin();
                pm.makePersistent(currentRecord);

                Class[]                                    classParameters={String.class,(new
Object()).getClass(),boolean.class};//class parameters of the method
                Object[]    ob={mainFieldNames[i],isd,new    Boolean(false)    };    //value
parameters
                java.lang.reflect.Method
meth=currentRecord.getClass().getMethod(“setFieldByName”, classParameters);
                meth.invoke(currentRecord,ob);
                tx.commit();

                showInstructors();
              }
              catch(Exception e){e.printStackTrace();}

            }
          refreshRecord();
          return null;
        }


        /**
         * Creates a toolbar with Add, Delete, Modify, and update etc.
         * control buttons
```

```
     *
     */
    private JToolBar createToolBar( ) {

        JButton button;
        JToolBar toolbar = new JToolBar();

        // toolbar.addSeparator();
        //DELETE
        mainToolBarButton[PREVIOUS] = toolbar.add(
        new AbstractAction("previous", new ImageIcon("previous.gif")) {
            public void actionPerformed(ActionEvent e) {
                //deletePet();
                previousRecord();
                showInstructors();
            }
        } );
        mainToolBarButton[PREVIOUS].setToolTipText("Deletes the selected Pet
object");
        //  toolbar.addSeparator();

        //ADD
        mainToolBarButton[NEXT] = toolbar.add(
        new AbstractAction("next", new ImageIcon("next.gif")) {
            public void actionPerformed(ActionEvent e) {
                //newData();
                nextRecord();
                showInstructors();
                //showInstructorsInList();
            }
        } );
        mainToolBarButton[NEXT].setToolTipText("Adds a new Pet object");
        //  toolbar.addSeparator();


        //SEARCH
        mainToolBarButton[DELETE] = toolbar.add(
        new AbstractAction("delete", new ImageIcon("delete.gif")) {
            public void actionPerformed(ActionEvent e) {

                deleteRecord();
            }
        } );
        mainToolBarButton[DELETE].setToolTipText("Deletes the Record");
        //  toolbar.addSeparator();
```

```
        //LIST
        mainToolBarButton[NEW] = toolbar.add(new AbstractAction("new") {
            public void actionPerformed(ActionEvent e) {

                newRecord();
            }
        } );
        mainToolBarButton[NEW].setToolTipText("Lists all Pet objects in the kennel");
        // toolbar.addSeparator();

        //WORKOUT
        mainToolBarButton[SAVE] = toolbar.add(
        new AbstractAction("Save", new ImageIcon("save.gif")) {
            public void actionPerformed(ActionEvent e) {
                saveRecord();
            }
        } );
        mainToolBarButton[SAVE].setToolTipText("Make the pets go through the workout
routines");
        //toolbar.addSeparator();
        mainToolBarButton[UPDATE] = toolbar.add(
        new AbstractAction("Update") {
            public void actionPerformed(ActionEvent e) {
                upDateRecord();
            }
        } );
        mainToolBarButton[UPDATE].setToolTipText("Make the pets go through the
workout routines");
        //  toolbar.addSeparator();

        mainToolBarButton[FIRST]= toolbar.add(
        new AbstractAction("<<First", new ImageIcon("first.gif")) {
            public void actionPerformed(ActionEvent e) {

            }
        } );
        mainToolBarButton[FIRST].setToolTipText("Make the pets go through the workout
routines");

        mainToolBarButton[LAST] = toolbar.add(
        new AbstractAction("last>>",new ImageIcon("last.gif")) {
            public void actionPerformed(ActionEvent e) {

            }
        } );
```

```
        mainToolBarButton[LAST].setToolTipText("Make the pets go through the workout
routines");

        // toolbar.addSeparator();
        // button.setEnabled(false);
        numberOfRecordLabel=new JLabel("num");
        if(result!=null)
           numberOfRecordLabel.setText(""+result.size());
        toolbar.add(numberOfRecordLabel );

        button = toolbar.add(
        new AbstractAction("Tabular") {
           public void actionPerformed(ActionEvent e) {
              showTabularForm();
           }
        } );

        toolbar.setLayout(new GridLayout(1, 0));
        return toolbar;
    }


/*Disables a set of buttons in the toolbar
 *@param ar int[] an array holding the buttons
 * we are going to disable
 */
    private void mainToolBarSetDisable(int[] ar){
        for(int i=0;i<ar.length;i++)
           mainToolBarButton[ar[i]].setEnabled(false);

    }

/*Enables a set of buttons in the toolbar
 *@param ar int[] an array holding the buttons
 * we are going to enable
 */

    private void mainToolBarSetEnable(int[] ar){
        for(int i=0;i<ar.length;i++)
           mainToolBarButton[ar[i]].setEnabled(true);

    }

/*Disables all the buttons in the toolbar
 */
    private void mainToolBarDisableAll(){
```

```
        for(int i=0;i<mainToolBarButton.length;i++)
            mainToolBarButton[i].setEnabled(false);

    }

/*Enables all the buttons in the toolbar
 */

    private void mainToolBarSetEnableAll(){
        for(int i=0;i<mainToolBarButton.length;i++)
            mainToolBarButton[i].setEnabled(true);

    }


/* updates the label that displays the total number of records
 */
    public void updateNumberOfRecords(){
        numberOfRecordLabel.setText("Records "+this.result.size());
    }




    /**
     * Creates a toolbar with Add, Delete, Modify, and List
     * control buttons
     *
     */
    private JToolBar createToolBar2( ) {

        JButton button;
        JToolBar toolbar = new JToolBar();

        String[] ar={"New","Delete"};
        for(int i=0; i<ar.length;i++){
            button=new JButton(ar[i]);
            toolbar.add(button);
        }

        return toolbar;
    }


/* shows the data in a form using
 * a tabular view
```

```
*/
  public void showTabularForm(){

     String[] fields=myJDOImplHelper.getFieldNames(mainFormObject.getClass());
     JTable jt=createJTableForCollection(result,mainFormObject,fields );

     JScrollPane jsp=new JScrollPane(jt);

     JFrame myframe =new JFrame();
     Container cont=myframe.getContentPane();
     cont.add(jsp);

     myframe.pack();
     myframe.setVisible(true);

  }


  ////////////////////////////////////////////////////////////////////////////////////////
  /**Creates a JTable to display a data from a collection
   * @param sresult the collection
   * @param record represents the object instance of which
   * we will display
   *
   * @param fields the fields of the object or the underlying data table
   * @return  the JTable
   */
  public static JTable createJTableForCollection(Collection sresult,Object record,
String[] fields ){
     Vector rowVector=new Vector();
     Vector columnVector=new Vector();
     Iterator iter=sresult.iterator();
     while(iter.hasNext()){
        rowVector.add( putRecordFieldsInVector( iter.next(),  fields ) );
     }

     //String[] fields=;
     for (int i = 0; i < fields.length; i++)
        columnVector.add(fields [i] );

     return new JTable(rowVector, columnVector);

  }
  ////////////////////////////////////////////////////////////////////////////////////////
  /**Creates a vector form records of a data table
   * @param record the record
```

```
     * @param fieldnames the field names of the data table or the record
     * @return  the vector
     */
    public static Vector putRecordFieldsInVector(Object record, String[] fieldnames){
        Vector returnVector=new Vector();
        Class[] rc={int.class};

        if (record !=null){
            for (int i = 0; i < fieldnames.length; i++) {

                try{

                    Object[] obi={ (new Integer(i)) };
                    java.lang.reflect.Method meth=record.getClass().getMethod("getField", rc);
                    Object returnValue=meth.invoke(record,obi);
                    if(returnValue==null) returnValue="null";
                    returnVector.add(returnValue);

                }
                catch( NoSuchMethodException e){e.printStackTrace();}
                catch( IllegalAccessException ed){ed.printStackTrace();}
                catch( java.lang.reflect.InvocationTargetException ex){ex.printStackTrace();}
            }
        }
        return returnVector;
    }
    ////////////////////////////////////////////////////////////////////////////


    /** refreshes the record
     * @return true if everything is ok */
    public boolean refreshRecord(){

        if (currentRecord !=null){
            for (int i = 0; i < mainFelds.length; i++) {
                Class[] rc={int.class};
                Object[] obi={ (new Integer(i)) };

                try{

                    java.lang.reflect.Method
meth=currentRecord.getClass().getMethod("getField", rc);
                    Object returnValue=meth.invoke(currentRecord,obi);

                    if(mainFelds[i] instanceof JTextField){
                        if (returnValue==null) returnValue=new String("");
```

```
                ((JTextField) mainFelds[i]).setText( returnValue.toString()   );
            }
            if(mainFelds[i] instanceof JSpinner){

                ((JSpinner) mainFelds[i]).setValue(  returnValue   );
            }

            if(mainFelds[i] instanceof JComboBox){


refreshTheLinkedDataForComboBoxesAfterUpdateOrDeleteOrNext((JComboBox)
mainFelds[i],i);

                ((JComboBox) mainFelds[i]).setSelectedItem( returnValue );
            }

            if(mainFelds[i] instanceof JCheckBox)
                ((JCheckBox)              mainFelds[i]).setSelected(              ((Boolean)
returnValue).booleanValue() )  ;

            if(mainFelds[i] instanceof JList){
                if (returnValue==null) returnValue=new LinkedList();
                ((JList) mainFelds[i]).setListData(((Collection) returnValue).toArray())   ;
            }
            if(mainFelds[i] instanceof InstructorsForm){
                LinkedList datalist=new LinkedList();
                if(returnValue!=null) datalist=new LinkedList((Collection) returnValue );
                ((InstructorsForm) mainFelds[i]).setMainData(datalist );

                //this first and then nextRecord() because nextRecord() goes recursively
                ((InstructorsForm) mainFelds[i]).updateNumberOfRecords();
                ((InstructorsForm) mainFelds[i]).nextRecord();

            }

        }
        catch( NoSuchMethodException e){e.printStackTrace();
        printNoSuchGetMethod();
        return false;         }
        catch( IllegalAccessException ed){ed.printStackTrace();}
        catch( java.lang.reflect.InvocationTargetException ex){ex.printStackTrace();}
      }

    /* try{// ayto den doyleyei giati einai private to
                JOptionPane.showMessageDialog(null,
```

```
                “                                Lnamefields                                --
>“+currentRecord.getClass().getDeclaredField(“lname”).get(currentRecord)   );
            }
            catch(Exception e){}

        */
       }
       else{
          for (int i = 0; i < mainFelds.length; i++) {
             if(mainFelds[i] instanceof JTextField)
                ((JTextField) mainFelds[i]).setText( “null” );
             if(mainFelds[i] instanceof JComboBox)
                ((JComboBox) mainFelds[i]).setSelectedItem( null );

             if(mainFelds[i] instanceof InstructorsForm){
                LinkedList datalist=new LinkedList();
                // if(returnValue!=null) datalist=new LinkedList((Collection) returnValue );
                ((InstructorsForm) mainFelds[i]).setMainData(datalist );

                //this first and then nextRecord() because nextRecord() goes recursively
                ((InstructorsForm) mainFelds[i]).updateNumberOfRecords();
                ((InstructorsForm) mainFelds[i]).nextRecord();

             }


          }
       }

       return true;

    }
    ////////////////////////////////////////////////////////////////////////////
    /** goes to the next record
     * @return  true if everything is ok
     */
    public boolean nextRecord(){
       mainToolBarSetEnableAll();
       int[] ar2={SAVE};
       mainToolBarSetDisable( ar2);

       if(iterator.hasNext()) {
          currentRecord =  iterator.next();
          refreshRecord();
          return true;
```

```
        }
        else{
            if(result.size()==0)
                currentRecord=null;
            int[] ar1={DELETE};
            mainToolBarSetDisable( ar1);

            refreshRecord();
        }

        return false;


    }


    /** goes to the previous record
     * @return   true if everything is ok
     */
    public boolean previousRecord(){
        mainToolBarSetEnableAll();
        int[] ar2={SAVE};
        mainToolBarSetDisable( ar2);

        if(iterator.hasPrevious()) {
            currentRecord =  iterator.previous();
            refreshRecord();
            return true;
        }


        return false;
    }


    /**Checks if there are linked tables with records n it
     * we use it to avoid delete a record if there are linked records
     * on it.
     * @return   true if there are dependent records
     */
    public boolean thereAreLinkedTablesWithRecords(){

        if (currentRecord !=null){
            for (int i = 0; i < mainFelds.length; i++) {

if(myJDOImplHelper.getFieldFlags(mainFormObject.getClass())[i]==10){//collection
```

```
                Class[] rc={int.class};
                Object[] obi={ (new Integer(i)) };

                try{

                   java.lang.reflect.Method
meth=currentRecord.getClass().getMethod("getField", rc);
                   Object returnValue=meth.invoke(currentRecord,obi);
                   if (returnValue!=null)
                      // if(returnValue instanceof Collection)
                      if(((Collection) returnValue).size()>0)
                         return true;

                }
                catch(Exception e){JOptionPane.showMessageDialog( null,"public boolean
thereAreLinkedTablesWithRecords()\n"+e);}
             }
          }
       }
       return false;
    }
    ////////////////////////////////////////////////////////////////
    /**deletes the records
     * @return  true if everything is ok
     */
    public boolean deleteRecord(){
       JOptionPane.showMessageDialog( null,"delete?");
       try{

          if (thereAreLinkedTablesWithRecords())
             JOptionPane.showMessageDialog( null,"there are linked records in this records
\n"+
             "You must first delete the records to the linked table");
          else{
             //edv na enhmervsv kai to parent
             tx.begin();

             informParentFormToDeleteTheReference();

             pm.deletePersistent(currentRecord);

             tx.commit();
             pm.refreshAll();
             if(getParentForm()==null){//only for the primary form{
                refreshTheSourceDataForTheFormAfterUpdateOrDelete();
```

```
            }
            updateNumberOfRecords();

            if(iterator.hasNext()) currentRecord=iterator.next();
            if( iterator.previousIndex()>-1)currentRecord=iterator.previous();
            else currentRecord=null;
            refreshRecord();
            return true;
          }
       }catch (Exception ed) {
          ed.printStackTrace();
          return false;
       }

       return false;

    }


/*in case of delete inform the parent record to
 *remove (delete) the reference
 */
    public void informParentFormToDeleteTheReference(){
       if(getParentForm()!=null){
          try{
             Object parentCurrentRecord=getParentForm().getCurrentRecord();
             pm.makePersistent(parentCurrentRecord);
             Class[]                                          classParameters={String.class,(new
Object()).getClass(),boolean.class};//class parameters of the method
             //i.e. "students"
             Object[] ob={Name,currentRecord,new Boolean(false) }; //value parameters

             java.lang.reflect.Method
meth=parentCurrentRecord.getClass().getMethod("setFieldByName", classParameters);
             meth.invoke(parentCurrentRecord,ob);
          }
          catch(Exception e){e.printStackTrace();}

       }

    }


    /**updates the record
     * @return   true if everything is ok
     */
```

```
   public boolean upDateRecord(){
      try{
         tx.begin();
         setTheValuesToTheRecord(currentRecord);


         pm.makePersistent(currentRecord);

         tx.commit();
         return true;
      }catch (Exception ed) {
         ed.printStackTrace();
         tx.rollback();
         return false;
      }


   }


/*refreshes the data after an update or insert or delete
 */
   public void refreshTheSourceDataForTheFormAfterUpdateOrDelete(){
      Extent recExtent=pm.getExtent(mainFormObject.getClass(),true);
      Iterator iter= recExtent.iterator();
      LinkedList listc=new LinkedList();
      while (iter.hasNext()){
         listc.add(iter.next());

      }
      setMainData(listc);
   }


   /**refreshes the data of comboboxes after an update or insert or delete or
    * previous, or next command has been executed
    * @param combo the combobox
    * @param i the index of the field
    */
   public                                                                    void
refreshTheLinkedDataForComboBoxesAfterUpdateOrDeleteOrNext(JComboBox
combo,int i ){
      InstructorsForm currentParentForm=getParentForm();

      while (currentParentForm!=null){ //the form is a subform
```

```
            Object parentCurrentRecord=currentParentForm.getCurrentRecord();
            if(parentCurrentRecord!=null)
               if(myJDOImplHelper.getFieldTypes(mainFormObject.getClass())[i]==
               parentCurrentRecord.getClass()   ){  //there are fields based in the parent form
                  Object[] arobj={parentCurrentRecord};
                  combo.removeAllItems();
                  combo.addItem(parentCurrentRecord);
               }
            currentParentForm=currentParentForm.getParentForm() ;
         }

      }

      /**saves the record
       * @return  true if everything is ok
       */
      public boolean saveRecord(){

         Class[] ar={};
         if(parentForm==null){
            try{ //mainFormObject must not be null or must be
               //class and i have to change the sentence
               //as mainFormObject.getConstructor(ar ).newInstance(null)

               currentRecord=mainFormObject.getClass().getConstructor(ar
).newInstance(null);
               setTheValuesToTheRecord(currentRecord);

               tx.begin();
               pm.makePersistent(currentRecord);

               tx.commit();

               refreshTheSourceDataForTheFormAfterUpdateOrDelete();

               pm.evictAll();
               pm.refreshAll();
               pm.refreshAll(result);


            }
            catch(Exception e){e.printStackTrace();

            tx.rollback();
            }
         }
```

```
      else{ //the form is a subform
         try{
            tx.begin();
            Object parentCurrentRecord=parentForm.getCurrentRecord();
            currentRecord=mainFormObject.getClass().getConstructor(ar
).newInstance(null);
            setTheValuesToTheRecord(currentRecord);

            //curreRecord will also be done persistent
            pm.makePersistent(currentRecord);

            pm.makePersistent(parentCurrentRecord);

            Class[]                                    classParameters={String.class,(new
Object()).getClass(),boolean.class};//class parameters of the method
            //i.e. "students"
            Object[] ob={Name,currentRecord,new Boolean(true) }; //value parameters


            java.lang.reflect.Method
meth=parentCurrentRecord.getClass().getMethod("setFieldByName", classParameters);
            meth.invoke(parentCurrentRecord,ob);
            tx.commit();

            pm.evictAll();
            pm.refreshAll();
            pm.refreshAll(result);

            parentForm.refreshRecord();
         }
         catch(Exception e){e.printStackTrace();
         tx.rollback();}
      }

      int[] ar1={SAVE};
      mainToolBarSetDisable( ar1);
      int[] ar2={DELETE};
      mainToolBarSetEnable(ar2 );

      refreshRecord();
      return false;
   }

/*Depriciated
 */
   public void updateTheComboBoxesData(){
```

```
    /*   InstructorsForm currentParentForm=this;

     while(currentParentForm!=null){ //the form is a subform

        for(int i=0;i<currentParentForm.mainFelds.length;i++){
          //see if we need some restrictions to the query here
          if(currentParentForm.mainFelds[i] instanceof InstructorsForm)
            currentParentForm=currentParentForm.InstructorsForm.mainFelds[i]
          if(currentParentForm.mainFelds[i] instanceof JComboBox)

            currentParentForm.mainFelds[i] = new JComboBox(
            (getTheDataForComboBoxes(myJDOImplHelper.getFieldTypes(
             currentParentForm.mainFormObject.getClass())[i]).toArray() ));
        }
     }
     */

  }


  /**Informs a record with the new values that have been
   *entered to the components of the form
   * @param record the record */
  public void setTheValuesToTheRecord(Object record){
     try{
        for (int i = 0; i < mainFelds.length; i++) {
          Class[]  classParameters={int.class,(new  Object()).getClass()};//,  Object};
//parameters of the method
          Object[] ob={};
          if(  mainFelds[i] instanceof JTextField  ){
            Object[] obi={ (new Integer(i)), ((JTextField) mainFelds[i]).getText() };
            ob=obi;
          }
          if(mainFelds[i] instanceof JComboBox){

            Object[]  obj={  (new  Integer(i)),  ((JComboBox)
mainFelds[i]).getSelectedItem() };
            ob=obj;
          }
          if(mainFelds[i] instanceof JCheckBox){
            Object[]  obj={  (new  Integer(i)),  new  Boolean(  ((JCheckBox)
mainFelds[i]).isSelected() ) };
            ob=obj;
          }
          if(mainFelds[i] instanceof JList){
```

```
            Object[] obj={  (new Integer(i)), null};// do nothing ;
            ob=obj;
          }
          if(mainFelds[i] instanceof JSpinner){
            Object[] obj={(new Integer(i)), ((JSpinner) mainFelds[i]).getValue(  ) };
            //JOptionPane.showMessageDialog(null,obj);
            ob=obj;
          }
          if(mainFelds[i] instanceof InstructorsForm){
            Object[] obj={(new Integer(i)), null};// do nothing ;
            //JOptionPane.showMessageDialog(null,obj);
            ob=obj;
          }

          java.lang.reflect.Method     meth=record.getClass().getMethod("setFieldValue",
classParameters);
          meth.invoke(record,ob);
        }
      }
      catch( NoSuchMethodException e){e.printStackTrace();
      printNoSuchSetMethod();}
      catch( IllegalAccessException ed){ed.printStackTrace();}
      catch( java.lang.reflect.InvocationTargetException ex){ex.printStackTrace();}


    }


    /**Creates a new record with default values
     * @return */
    public boolean newRecord(){


      Class[] ar={}; // for a constructor without arguments
      try{
        currentRecord=mainFormObject.getClass().getConstructor(ar
).newInstance(null);
        //assign default values using the Parent Form
        setDefaultValuesFromParentForm();

        ////assign default values that might user had entry using before the
setDefaultValues(..) method
        setDefaultValuesFromUserEntry();

        //but we must add the new record otherwise it is not going to be in the combobox
        refreshRecord();
```

```
        }
        catch(Exception e){e.printStackTrace(); return false;}

        int[] ar1={DELETE};
        mainToolBarSetDisable( ar1);
        int[] ar2={SAVE};
        mainToolBarSetEnable( ar2);

        return true;
    }


    /**Sets the default values that the user
     *has assigned for the record
     */
    public void setDefaultValuesFromUserEntry(){
        Class[] classParameters={int.class,(new Object()).getClass()};   //class parameters of
the method
        try{

            int fieldindex=-1;
            if(defaultValueName!=null)//if there have been assigned default values bye the
setDefaultValues() method
                for (int i=0; i<mainFieldNames.length;i++)
                    for (int j=0; j<defaultValueName.length;j++)
                        if(mainFieldNames[i].equals(defaultValueName[j])){
                            fieldindex=i;
                            java.lang.reflect.Method
meth=currentRecord.getClass().getMethod("setFieldValue", classParameters);
                            Object[] ob={   (new Integer(fieldindex)), defaultValueValue[j]};//the
parameters of the methods
                            meth.invoke(currentRecord,ob);

                        }
        }
        catch(Exception e){e.printStackTrace();}

    }



    /**Sets the default values from
     *the parent form
     */

    public void setDefaultValuesFromParentForm(){
```

```
    Class[] classParameters={int.class,(new Object()).getClass()};   //class parameters of
the method
    InstructorsForm currentParentForm=getParentForm();

    while(currentParentForm!=null){ //the form is a subform

      try{

        Object parentCurrentRecord=currentParentForm.getCurrentRecord();
        if(parentCurrentRecord!=null){
          for(int i=0;i<mainFieldNames.length;i++)
            if(myJDOImplHelper.getFieldTypes(mainFormObject.getClass())[i]==
            parentCurrentRecord.getClass()    ){//there are fields based in the parent
form

              java.lang.reflect.Method
meth=currentRecord.getClass().getMethod("setFieldValue", classParameters);
              Object[] ob={  (new Integer(i )), parentCurrentRecord};//the parameters
of the methods
              meth.invoke(currentRecord,ob);

            }
        }
      }
      catch(Exception e){e.printStackTrace(); }
      //continue to the previous forms to get the default values
      currentParentForm=currentParentForm.getParentForm();
    }
  }
  ////////////////////////////////////////////////////////////////////////////////


  /** Depriciated
   * @return  */
  public boolean showInstructors(){

      /*for (int k = 0; k < subFields.length; k++)
      for (int i = 0; i < subFields[k].length; i++) {
          subFields[k][i].setText("");
      }

          if (currentRecord !=null){
              //iterator=((Students)currentRecord).getInstructors().listIterator()
          LinkedList linst=((Instructors)currentRecord).getStudents();
              for (int i = 0; i < linst.size(); i++) {
```

```
                                    subFields[i][0].setText( ((Students) linst.get(i)).getId()  );
                                    subFields[i][1].setText(                                ((Students)
linst.get(i)).getLastName()  );
                            }

                        return true;
                        }
                */
        return false;
    }


    /**depreciated
     */
    public void showInstructorsInList(){
        LinkedList l1=((Students)currentRecord).getInstructors();
        //*l1.toArray();
        String[] ar1={"M","ksk"};
        if (l1!=null)

            subFormList.setListData(l1.toArray());//l1.toArray());//Vector listData)
        else
            subFormList.removeAll();

    }


    /**gets the persistent manager factory
     * @param path the url path
     * @return  the persistent manager factory*/
    public static PersistenceManagerFactory getPMF(String path) {
        try {
            InputStream propStream =
            new FileInputStream(path);
            Properties props = new Properties();
            props.load(propStream);

            props.put("javax.jdo.PersistenceManagerFactoryClass",
"com.libelis.lido.PersistenceManagerFactory");

            return JDOHelper.getPersistenceManagerFactory(props);
        } catch (IOException e) {
            e.printStackTrace();
            System.exit(-1);
            return null;
        }
```

```
}

/**
 * @param e  */
public void mouseClicked(MouseEvent e) {
   chooseInstructor(Integer.parseInt(e.getComponent().getName()) );

}
/**
 * @param e  */
public void mousePressed(MouseEvent e) {

}

/**
 * @param e  */
public void mouseReleased(MouseEvent e) {

}
/**
 * @param e  */
public void mouseEntered(MouseEvent e) {

}
/**
 * @param e  */
public void mouseExited(MouseEvent e) {

}
/**
 * @param e  */
public void itemStateChanged(ItemEvent e) {
   currentRecord=e.getItem() ;
   refreshRecord();


}

/** used for experiment
 */
public void expirement(){
   ////////////////////////////////////////////////////////////////////////////////////////////////////

   String[] buttonLabel = { “Enter”, “Delete” };
   JPanel JtablePanel=new JPanel();
```

```
        JTable myjtable= new JTable(10,5);//
        Object mycurrentRecord;
        Object[][] objarray= new Object[result.size()][mainFieldNames.length];
        //iterator.first();
        for (int i=0; i<objarray.length;i++){
           currentRecord =  iterator.next();

           //bazontas to currentRecord na typvuei tote kalei kai enhmervnei ta paidia toy
           //String x=""+currentRecord;
           // JOptionPane.showMessageDialog(null,"The array of the length of the=-
"+currentRecord);

           //  objarray[i]=putRecordFieldsInVector( currentRecord ).toArray();

        }
        //currentRecord = new Flights();
        // JOptionPane.showMessageDialog(null,"The array of the length of the=-
"+getRecordInFormOfObjectArray( currentRecord ));

        Object[]
objcolumn=myJDOImplHelper.getFieldTypes(mainFormObject.getClass());
        myjtable= new JTable(objarray,objcolumn);
        JScrollPane scrollpane = new JScrollPane(myjtable);

        JtablePanel.add(scrollpane);
        JOptionPane.showOptionDialog( null, JtablePanel, "Instructors Information",
        JOptionPane.DEFAULT_OPTION,
        JOptionPane.QUESTION_MESSAGE, null,
        buttonLabel, buttonLabel[0]    );

        ////////////////////////////////////////////////////////////////////////////////////////
        ////////////////////////////////////////////////////////////////////////////////////////


        setVisible(true);
    }

    /**experiment for finding
     *record using key values
     */
    public void findRecordByKey(){
        LinkedList sresult;

        q = pm.newQuery(mainFormObject.getClass());
        byte[] arflag=myJDOImplHelper.getFieldFlags(mainFormObject.getClass())  ;
        String[] arname=myJDOImplHelper.getFieldNames(mainFormObject.getClass());
```

```
        String filter="";
        String parameters="";
        //Students.class,"squadron==sq", "Squadron sq", sq
        for (int i=0;i<arflag.length;i++)
           if(arflag[i]==8)  {//8 means primary key
              if(filter.equals("")){
                 filter=arname[i]+"=="+"param"+i;

parameters+=myJDOImplHelper.getFieldTypes(mainFormObject.getClass())[i] ;
              }
              else  {
                 filter=filter+" && "+arname[i]+"=="+"param"+i;
                 parameters+="                                                                    ;
"+myJDOImplHelper.getFieldTypes(mainFormObject.getClass())[i] ;

              }
           }

    /*       JOptionPane.showMessageDialog( null," filter-->"+filter +"\n"+"parameters-->
"+parameters+
                          "\n\n\nTHIS NEEDS TO FIX IT");
        q.declareParameters(queryVariables);
        q.setFilter(filter);
       // q.declareVariables(queryVariables);

        sresult = new LinkedList( (Collection) q.execute(theObject) );


this.result=sresult;
if (sresult!=null)
 iterator= sresult.listIterator();
else
 JOptionPane.showMessageDialog( null,"form is empty of records");

      // return sresult;
     */

    }

    /**Sets the default values from a user
     * @param defaultValueName array containing the name of the fields
     * @param defaultValueValue array containing the default values for the
     * above fields
     */
    public void setDefaultValues(String[] defaultValueName,Object[]
defaultValueValue){
```

```
    this.defaultValueName=defaultValueName;
    this.defaultValueValue=defaultValueValue;

  }



  /**Tries to get the length of the fields. If there is no result raise a message and assigns
   * default values for the length of the fields
   * @return the length of the fields
   */
  public int[] getTheLengthOftheFields(){

    //just a default value
    int[]fieldLength={30,20,20,15,10,7,17,10};
    try{
      Class[] classParameters={};//class parameters of the method
      Object[] ob={ };                  //value parameters of the method
      java.lang.reflect.Method
meth=mainFormObject.getClass().getMethod("getFieldLengths", classParameters);
      fieldLength=( (int[]) meth.invoke(mainFormObject,ob) );


if(fieldLength.length!=myJDOImplHelper.getFieldNames(mainFormObject.getClass()).l
ength){
        JOptionPane.showMessageDialog(null,"The array of the length of the fields
does not correspond to the \n"+
        "number of the fields. You need to correct the returned int array \n"+
        "to the getFieldLengths() method of the Class. The number of the \n"+
        "elements of the array must be equal to the number of the fields \n"+
        "of the class.\n I will assign default values for the length of the\n"+
        "fields (temporary)");
        fieldLength=new
int[myJDOImplHelper.getFieldNames(mainFormObject.getClass()).length];
        for(int i=0;i<fieldLength.length;i++)
          fieldLength[i]=20;

      }

    }
    catch(Exception e)
    {  e.printStackTrace();
        JOptionPane.showMessageDialog(null,"I don't have information about the
length\n"+
        "of the fields. I will use a default length\n"+
```

```
            "for the fields. If you want to have better\n"+
            "representation of the data you should declare\n"+
            "a public method in the " +mainFormObject.getClass().getName()+"\n"+
            "as bellow:\n\n"+
            "public int[] getFieldLengths(){\n"+
            "       return (the length of ALL the fields)\n"+
            "}\n\n\n"+
            "pzilidis@nps.navy.mil");

            fieldLength=new
int[myJDOImplHelper.getFieldNames(mainFormObject.getClass()).length];
            for(int i=0;i<fieldLength.length;i++)
               fieldLength[i]=20;
      }

      return fieldLength;

   }

   /**Checks when a field is a collection then tries to
    *find what kind of objects holds that collection. If its
    * unable to do so raise a message
    * @param dataclassName the class of the collection field
    */
   public void printMessageNoSuchMethod(String dataclassName){
      JOptionPane.showMessageDialog(null,"I dont have information about the class "+
      "of the collection field:"+ dataclassName+" " +"You need to provide\n"+
      "the class of the object that the above collection field will contain. \n"+
      "Just add a public method in the "+
      """ +mainFormObject.getClass().getName()+"\n"+
      "as bellow:\n\n"+
      "  public Class getTheTypeOfTheCollection(String collectionName){\n"+
      "                                if (collectionName.equals(\"students\""+"))"+
      "                                  return Students.class;"+
      "\n"+
      "                                  return null;\n"+
      "  }\n"+

      "\n\n\n"+
      "pzilidis@nps.navy.mil");

   }

   /**Checks if there is a setter method, otherwise
    * raises a message
    */
```

```
   public void printNoSuchSetMethod(){
      JOptionPane.showMessageDialog(null,"I cannot add the values to the specific class "+
      ":" +  "You need to provide\n"+
      "a method in the class that will set the values to the object. \n"+
      "Just add a public method in the "+
      "" +mainFormObject.getClass().getName()+"\n"+
      "exactly with the same name as bellow:\n\n"+
      "  public void setFieldValue(int i,Object value){\n"+
      "                  switch (i) {\n"+
      "                        case 0:\n"+
      "                        setId(value.toString());\n"+
      "                              break;\n"+
      "          .....\n"+
      "          .....\n"+
      "\n"+
      "  }\n\n\n"+
      "Although you can navigate through the data you can not \n"+
      " add or modify (update ) the records without the above\n"+
      " in your class"+

      "\n\n\n"+
      "pzilidis@nps.navy.mil");




   }

   /**Checks if there is a getter method, otherwise
    * raises a message
    */

   public void printNoSuchGetMethod(){
      JOptionPane.showMessageDialog(null,"I cannot retrieve the values to the specific class "+
      ":" +  "You need to provide\n"+
      "a method in the class that will provide the values to the object. \n"+
      "Just add a public method in the "+
      "" +mainFormObject.getClass().getName()+"\n"+
      "exactly with the same name as bellow:\n\n"+
      "  public Object getField(int i){\n"+
      "                  switch (i) {\n"+
      "                        case 0:\n"+
      "                                          return aircraft;\n"+
      "                              break;\n"+
```

```
        "          .....\n"+
        "          .....\n"+
        "\n"+
        "  }\n\n\n"+


        "\n\n\n"+
        "pzilidis@nps.navy.mil");


    }

}
```

### 2. The Class: MyBarChart.java

MyBarChart

```
/**
 * <p>Title: MyBarChart.java</p>
 * <p>Description: Creates a bar chart</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     CS JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */

package test;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.GradientPaint;
import javax.swing.JFrame;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.CategoryAxis;
import org.jfree.chart.axis.CategoryLabelPositions;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.renderer.BarRenderer;
import org.jfree.data.CategoryDataset;
import org.jfree.data.DefaultCategoryDataset;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;
```

```
import javax.swing.*;

public class MyBarChart extends JPanel{


   private ChartPanel chartPanel;
   JComboBox jComboBox1 = new JComboBox();
   /**
    * Constructor
    *
    * @param data the data
    * @param categories the categories
    * @param series1 the series
    */
   public MyBarChart(int[] data, String[] categories,String series1) {

      CategoryDataset dataset = createDataset(data, categories, series1);
      JFreeChart chart = createChart(dataset);

      chartPanel = new ChartPanel(chart);
      chartPanel.setPreferredSize(new Dimension(700, 400));
      add(chartPanel);

   }


   /**
    * Constructor
    *
    * @param data the data
    * @param categories the categories
    * @param series1 the series
    */

   public MyBarChart(long[] data, String[] categories,String series1) {


      CategoryDataset dataset = createDataset(data, categories, series1);
      JFreeChart chart = createChart(dataset);

      chartPanel = new ChartPanel(chart);
      chartPanel.setPreferredSize(new Dimension(700, 400));
      add(chartPanel);

   }
```

```
/**
 * Creates a dataset
 *
 * @param data the data
 * @param categories the categories
 * @param series1 the series
 * return the dataset
 */

private CategoryDataset createDataset(long[] data, String[] category,String series1) {

    // create the dataset...
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    for (int i = 0; i < data.length; i++) {
        dataset.addValue(data[i], series1, category[i]);
    }
    return dataset;
}


/**
 * Creates a dataset
 *
 * @param data the data
 * @param categories the categories
 * @param series1 the series
 * return the dataset
 */

private CategoryDataset createDataset(int[] data, String[] category,String series1) {

    String category1 = “Category 1”;
    String category2 = “Category 2”;
    // create the dataset...
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    for (int i = 0; i < data.length; i++) {
        dataset.addValue(data[i], series1, category[i]);
    }

    return dataset;

}
```

```
    /**
     * Creates a dataset
     *
     * @param data the data
     * @param categories the categories
     * @param series1 the series
     * return the dataset
     */

    private CategoryDataset createDataset(double[] data, String[] category,String series1)
{

        String category1 = “Category 1”;
        String category2 = “Category 2”;
        // create the dataset...
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        for (int i = 0; i < data.length; i++) {
            dataset.addValue(data[i], series1, category[i]);
        }
        return dataset;
    }


    /**
     * Displays the chart
     *
     * @param title the title of the chart
     *
     */

    public void toForm(String title){
        JFrame jf= new JFrame(title);
        jf.getContentPane().add(this);
        jf.pack();
        jf.setVisible(true);
    }

    /**
     * Sets the dataset for the chart
     *
     * @param data the data
     * @param categories the categories
     * @param series1 the series
     *
     */
```

```
public void setDataset(int[] data, String[] category,String series1){
   CategoryDataset newdata=createDataset( data,  category, series1);
   JFreeChart chart = createChart(newdata);
   chartPanel.setChart(chart);
}


/**
 * Sets the dataset for the chart
 *
 * @param data the data
 * @param categories the categories
 * @param series1 the series
 *
 */

public void setDataset(double[] data, String[] category,String series1){
   CategoryDataset newdata=createDataset( data,  category, series1);
   JFreeChart chart = createChart(newdata);
   chartPanel.setChart(chart);
}


/**
 * Sets the dimensions of the chart
 *
 * @param x the x dimension
 * @param y the y dimension
 *
 *
 */

public void setDimension(int x , int y){
   chartPanel.setPreferredSize(new Dimension(x, y));
}

/**
 * Creates a sample chart.
 *
 * @param dataset the dataset.
 *
 * @return The chart.
 */
private JFreeChart createChart(CategoryDataset dataset) {
```

```
// create the chart...
JFreeChart chart = ChartFactory.createBarChart(
“Bar Chart Demo”,        // chart title
“Category”,              // domain axis label
“Value”,                 // range axis label
dataset,                 // data
PlotOrientation.VERTICAL, // orientation
true,                    // include legend
true,                    // tooltips?
false                    // URLs?
);


// set the background color for the chart...
chart.setBackgroundPaint(new Color(0xBBBBDD));

// get a reference to the plot for further customisation...
CategoryPlot plot = chart.getCategoryPlot();

// set the range axis to display integers only...
NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());

// disable bar outlines...
BarRenderer renderer = (BarRenderer) plot.getRenderer();
renderer.setDrawBarOutline(false);

// set up gradient paints for series...
GradientPaint gp0 = new GradientPaint(
0.0f, 0.0f, Color.blue,
0.0f, 0.0f, Color.lightGray
);
GradientPaint gp1 = new GradientPaint(
0.0f, 0.0f, Color.green,
0.0f, 0.0f, Color.lightGray
);
GradientPaint gp2 = new GradientPaint(
0.0f, 0.0f, Color.red,
0.0f, 0.0f, Color.lightGray
);
renderer.setSeriesPaint(0, gp0);
renderer.setSeriesPaint(1, gp1);
renderer.setSeriesPaint(2, gp2);

CategoryAxis domainAxis = plot.getDomainAxis();
domainAxis.setCategoryLabelPositions(CategoryLabelPositions.UP_90);
```

```
        domainAxis.setMaxCategoryLabelWidthRatio(5.0f);


        return chart;

    }

    public MyBarChart() {
        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
    private void jbInit() throws Exception {
        this.add(jComboBox1, null);
    }
}
```

### 3. The Class: Populate.java

Populate

```
/**
 * <p>Title: Populate.java</p>
 * <p>Description: The class is used to add some records to the tables for the
 * implementation and the checking of the project </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     CS JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */

package test;

import javax.jdo.PersistenceManagerFactory;
import javax.jdo.PersistenceManager;
import javax.jdo.Transaction;
import java.util.Date;

import utils.Connection;

import company.*;

public class Populate {
```

```
   private static PersistenceManagerFactory pmf;
   private static PersistenceManager pm;
   private static Transaction tx;

   public static void main(String[] args) {
      try {
         pmf = Connection.getPMF(args);
         pm = pmf.getPersistenceManager();
         tx = pm.currentTransaction();
         int identity = 0;
         Students student;
         Students student1;
         student1=new Students( ""+100,"paspartou"," fname"+100,  0.0);
         Instructors instructor = null;
         Aircraft aircraft;
         AircraftType aircrafttype;
         Flights flight;

         Rank rank;
         Squadron squadron;
         Series series;
         BaseStadio bstadio;

         long debut = 0;
         String[] ins = {"Paschalis", "Nikos", "George", "Tom", "Malakas"};
         String[] stu = {"papa", "kilo", "lima", "charlie", "echo"};
         String[] bst = {"advance", "beginers", "fihter", "initial", "adaptive"};
         String[] ser = {"72", "73", "74", "75", "76"};
         // String[] depts = {"Sales", "Marketing", "RD", "Admin"};

         debut = System.currentTimeMillis();
         System.out.println("Creating instances ");
         String[] ranks = {"SEc Lieutenant", "Lieutenant", "Captain", "Major",
"colonel"};
         int nbComp = 5;
         int nbdpt = 4;
         int nbEmp = 6;
         for (int i = 0; i < 5; i++) {
            tx.begin();

            series= new Series(ser[i],ser[i]+i);
            bstadio=new BaseStadio(bst[i],bst[i]+i);


            student=new Students( ""+i,stu[i]," fname"+i,  0.0);
            instructor= new Instructors(""+i, ins[i],"instr_name"+i,  0.0);
```

```
        aircrafttype=new AircraftType("T2-EBAck","DEception");
        rank= new Rank(""+i,ranks[i]);
        aircraft= new Aircraft("10"+i,aircrafttype,  0.0);

        student1.addInstructors( instructor);

        flight= new Flights( aircraft, instructor, student, new Date());

        squadron=new Squadron("34"+i,"description-"+i);

        pm.makePersistent(student);

        pm.makePersistent(instructor);
        pm.makePersistent(flight);
        pm.makePersistent(aircraft);
        pm.makePersistent(aircrafttype);
        pm.makePersistent(rank);
        pm.makePersistent(squadron);
        pm.makePersistent(bstadio);
        pm.makePersistent(series);
        try{
           tx.commit();
           System.out.println("]");
        }
        catch(javax.jdo.JDOException e){
           System.out.println("Problem:  "+e);
        }

     }



     tx.begin();
     pm.makePersistent(student1);
     try{
        tx.commit();
        System.out.println("]");
     }
     catch(javax.jdo.JDOException e){
        System.out.println("Problem:  "+e);
     }



} catch (Exception e) {
   e.printStackTrace();
```

```
      }

   }
}
```

### 4. The Class: PureSQL.java

PureSQL

```
/**
 * <p>Title: PureSQL.java</p>
 * <p>Description: Creates pure SQL COMMANDS retrieving
 * exactly the same result set as the JDOQL. These are used to asses the
 * performance of JDO in contrast to that of JDBC</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     CS JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */

package test;



import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;

import javax.swing.*;


/**
 *
 */
class PureSQL  {

   /**
    * multi Constructor
    */

   public PureSQL(  ) {


   }
```

```
    //---------------------------------
    //      Main method
    //---------------------------------
    public static void main(String[] args) {

    }
    ////////////////////////////////////////////////////////////////////////////////

/* creates and executes a GROUP BY query.
 * The method iterates or not through the result
 * set when a boolean is true or false accordingly
 *
 * @param iteration The boolean that determine if we will
 * iterate through the result or not
 * @return the time that takes for the retrieval of the result
 */

    public static long showGROUPBYInstructorsHours( boolean iteration){
        String sql="SELECT a.instructor_id, SUM(a.endurance) "+
        "FROM c_flights a "+
        "GROUP BY a.instructor_id ";

        return getQuery( sql,iteration);

    }
    ////////////////////////////////////////////////////



/* creates and executes a SELECT query.
 * The method iterates or not through the result
 * set when a boolean is true or false accordingly
 *
 * @param iteration The boolean that determine if we will
 * iterate through the result or not
 * @return the time that takes for the retrieval of the result
 */

    public static long showSELECTQueryFlights( boolean iteration){
        String sql="SELECT * FROM c_flights  ";

        return getQuery( sql,iteration);

    }

    ////////////////////////////////////////////////////////////////
```

```
/* EXECUTES A QUERY AND RETURN THE TIME HAS BEEN TAKEN
 * FOR THE EXECUTION.
 *
 * @param sql the SQL command
 * @param iteration The boolean that determine if we will
 * iterate through the result or not
 * @return the time that takes for the retrieval of the result
 */

   public static long getQuery(String sql,boolean iteration){
      long timeStart=System.currentTimeMillis() ;
      System.out.println("GROUP BY: "+iteration+"  timeStart:"+ timeStart);

      Connection connection;
      try{
         Class.forName("com.mysql.jdbc.Driver");
         String dburl="jdbc:mysql://localhost/pilotdb";
         String username="";
         String password="";
         connection = DriverManager.getConnection(dburl);
         System.out.println("Eyruthing ok");

         Statement statement= connection.createStatement();
         ResultSet rset=statement.executeQuery(sql);
         int k=0;
         if (iteration)
            while (rset.next()){
               k++;
               String s=rset.getString(1);
               //System.out.println(k+"  "+s);
            }//do nothing just iterate

      }
      catch(ClassNotFoundException e){
         System.out.println("Database driver not found.");

      }

      catch(SQLException e){
         System.out.println("Error opening db connection: "+ e.getMessage());

      }
      long timeStop=System.currentTimeMillis() ;
```

```
        System.out.println("GROUP BY"+ timeStop);

        System.out.println("GROUP BY"+(timeStop-timeStart));

        return timeStop-timeStart;

    }


}
```

### 5. The Class: QueryForm.java

QueryForm

```
/**
 * <p>Title: QueryForm.java</p>
 * <p>Description: Contains the JDOQL queries that we will
 *  use in our application </p>
 *
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     CS JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */

package test;
////////////////////////////////////////////////
////////////////////////////////////////////////
import javax.jdo.JDOHelper;                ///////
import javax.jdo.PersistenceManagerFactory; ///////
import javax.jdo.PersistenceManager;       ///////
import javax.jdo.Transaction;              //////
import javax.jdo.Query;                   ///////
import java.io.FileInputStream;            ///////
import java.io.IOException;               ///////
import java.io.InputStream;               ///////
import javax.jdo.spi.JDOImplHelper;        ///////
import javax.jdo.*;/////////              ///////
////////////////////////////////////////////////
////////////////////////////////////////////////

import company.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
```

```
import javax.swing.JComboBox;

import java.util.*;
import java.text.DecimalFormat;

/**
 *Constructor
 */
class QueryForm extends JFrame {

    //.....................................................
    //.....................................................
    private static PersistenceManagerFactory pmf;
    private static PersistenceManager pm;
    private static Transaction tx;
    private static Query q;
    private static Collection result;

    private JPanel         topPanel;
    private JPanel        formPanel;
    private JComboBox      squadronComboBox;
    private JComboBox      seriesComboBox;
    /**
     * Default frame width
     */
    private static final int FRAME_WIDTH    = 800;

    /**
     * Default frame height
     */
    private static final int FRAME_HEIGHT   = 480;

    /**
     * X coordinate of the frame default origin point
     */
    private static final int FRAME_X_ORIGIN = 150;

    /**
     * Y coordinate of the frame default origin point
     */
    private static final int FRAME_Y_ORIGIN = 250;

    /**
     * multi Constructor
     */
    public QueryForm(  ) {
```

```
    try {
        pmf = getPMF( "lido_mysql.properties");
        pm = pmf.getPersistenceManager();
        tx = pm.currentTransaction();
        Container contentPane=getContentPane( );

        contentPane.setLayout(new BorderLayout());
        topPanel= new JPanel();


    }
    catch (Exception e){e.printStackTrace();};

}

//---------------------------------
//     Main method
//---------------------------------
public static void main(String[] args) {
    QueryForm f=new QueryForm(  );
    f.pack();
    f.setVisible(true);
}



/** Query the flight table and search for a specific
 * instructor
 * @param pm the Persistence Manager
 * @param inst the specific Instructor
 * @return  a collection of the result
 */

public static Collection queryTheFlights(PersistenceManager pm, Instructors inst){
    Extent extent=pm.getExtent(Flights.class,true);
    String filter="instructor==inst";
    Query query=pm.newQuery(extent,filter);
    query.declareParameters("Instructors inst");
    Collection result=(Collection) query.execute(inst);

    return result;
}
```

```
   /**gets the persistent manager factory
    * @param path the url path
    * @return  the persistent manager factory
    */

   public static PersistenceManagerFactory getPMF(String path) {
      try {
         InputStream propStream =
         new FileInputStream(path);
         Properties props = new Properties();
         props.load(propStream);

         props.put("javax.jdo.PersistenceManagerFactoryClass",
"com.libelis.lido.PersistenceManagerFactory");

         return JDOHelper.getPersistenceManagerFactory(props);
      } catch (IOException e) {
         e.printStackTrace();
         System.exit(-1);
         return null;
      }

   }



   /** Tests the query
    */

   private void testTheQuery(ActionEvent e){
      JButton jb=(JButton) e.getSource();
      //i have put the button to the Categories form
      InstructorsForm iform=(InstructorsForm) jb.getParent();
      Categories cat=(Categories) iform.getCurrentRecord();

      LinkedList list= new LinkedList( QueryGetFlightsExcerciseReportPerCategory(
cat));

      JList j2=new JList(list.toArray());
      JPanel l=new JPanel();
      l.add(new JButton());
      l.add(j2);

      JScrollPane jsp=new JScrollPane(l);
```

```
    JFrame myframe =new JFrame();
    Container cont=myframe.getContentPane();
    cont.add(jsp);
    myframe.pack();
    myframe.setVisible(true);

}


////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//////////////////////FLIGHTS QUERIES/////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//////FLIGHTS QUERIES/////////////////////////////////////////////////////////////////

/** Query the flights table
 *
 * @param pm the Persistence Manager
 *
 * @return  a collection of the result
 */

public static Collection QueryGetFlights(PersistenceManager pm){


    Extent extent=pm.getExtent(Flights.class,true);
    Query query=pm.newQuery(extent);
    Collection result=(Collection) query.execute();
    query.close(result);
    return result;


}


/** Query the flights table
 * @param pm the Persistence Manager
 *
 * @return a collection of the result
 */

public static Collection QueryGetFlightsSQL(PersistenceManager pm){


    //Extent extent=pm.getExtent(Flights.class,true);
    Query query=pm.newQuery("sql","SELECT * FROM c_flights");
```

```
        query.setClass(Flights.class);
        Collection result=(Collection) query.execute();
        query.close(result);
        return result;


    }

    /** Query the flight table and GROUP BY THE
     * instructors
     * @param pm the Persistence Manager
     *
     * @return  a collection of the result
     */

    public static Collection QueryGetFlightsSQLGROUPBY(PersistenceManager pm){

        String sql="SELECT a.instructor_id, SUM(a.endurance) "+
        "FROM c_flights a "+
        "GROUP BY a.instructor_id ";


        Query query=pm.newQuery("sql",sql);
        query.setClass(Flights.class);
        Collection result=(Collection) query.execute();
        query.close(result);
        return result;


    }


    /** Query the flights table and search for a specific
     * date
     * @param pm the Persistence Manager
     * @param d the date
     * @return  a collection of the result
     */

    public static Collection QueryGetFlightsDailyReport(PersistenceManager pm, Date
d){


        Extent extent=pm.getExtent(Flights.class,true);
        String filter="date==d";
        Query query=pm.newQuery(extent,filter);
```

```
        query.declareImports("import java.util.Date");
        query.declareParameters("Date d");
        Collection result=(Collection) query.execute(d);
        query.close(result);
        return result;


    }

    /** Query the flights table and search the results for a specific
     * date and squadron
     * @param pm the Persistence Manager
     * @param squad the Squadron
     * @param d the d
     * @return a collection of the result
     */

    public static Collection QueryGetFlightsDailyReportPerSquadron(PersistenceManager
pm,Date d, Squadron squad){


        Extent extent=pm.getExtent(Flights.class,true);
        String filter="date==d && squadron==squad";
        Query query=pm.newQuery(extent,filter);
        query.declareImports("import java.util.Date");
        query.declareParameters("Date d, Squadron squad");
        Collection result=(Collection) query.execute(d,squad);
        query.close(result);
        return result;


    }


    //the field might be private however here the JDOQL seem to have access
    // directly to them using the dot(.) operator i.e. flights.exercise
    /** Query the flights table and search the results for a specific
     * category
     * @param cat the category
     *
     * @return a collection of the result
     */
    public Collection QueryGetFlightsExcerciseReportPerCategory(Categories cat){


        Extent extent=pm.getExtent(Flights.class,true);
```

```
    String filter="exercise.categories==cat";
    Query query=pm.newQuery(extent,filter);
    //query.declareImports("import java.util.Date");
    query.declareParameters("Categories cat");
    Collection result=(Collection) query.execute(cat);

    return result;


}


/** Query the flights table and search the results for a specific
 * year and month
 * @param pm the Persistence Manager
 * @param year the year
 * @param month the month
 * @return a collection of the result
 */
public static Collection QueryGetFlightsMonthlyReportAllSquadrons(
PersistenceManager pm, int month, int year){

    Calendar c1= new GregorianCalendar(year,month,1) ;
    if(month==11){
       year+=1;
       month=0;
    }
    else
       month+=1;
    Calendar c2= new GregorianCalendar(year,month,1) ;

    Date beginDate=c1.getTime();
    Date endDate=c2.getTime();

    Extent extent=pm.getExtent(Flights.class,true);
    String filter="date>=beginDate && date<endDate";
    Query query=pm.newQuery(extent,filter);
    query.declareImports("import java.util.Date");
    query.declareParameters("Date beginDate, Date  endDate");
    Collection result=(Collection) query.execute(beginDate,endDate);
    query.close(result);
    return result;


}
```

```
/** Query the flights table and search the results for a specific
 * Squadron for a specific year and month
 * @param pm the Persistence Manager
 * @param year the year
 * @param month the month
 * @param squad the Squadron
 * @return a collection of the result
 */
public static Collection QueryGetFlightsMonthlyReportPerSquadron(
PersistenceManager pm,int month, int year, Squadron squad){

   Calendar c1= new GregorianCalendar(year,month,1) ;
   if(month==11){
      year+=1;
      month=0;
   }
   else
      month+=1;
   Calendar c2= new GregorianCalendar(year,month,1) ;

   Date beginDate=c1.getTime();
   Date endDate=c2.getTime();
   //JOptionPane.showMessageDialog(null,beginDate+"----- "+endDate);

   Extent extent=pm.getExtent(Flights.class,true);
   String filter="date>=beginDate && date<endDate && squadron==squad";
   Query query=pm.newQuery(extent,filter);
   query.declareImports("import java.util.Date");
   query.declareParameters("Date beginDate, Date  endDate, Squadron squad");
   Collection result=(Collection) query.execute(beginDate,endDate, squad);
   query.close(result);
   return result;


}


/** Query the flights table and search the results for a specific
 * student in a Squadron for a specific year and month
 * @param stude the Student
 * @param year the year
 * @param month the month
 * @param squad the Squadron
 * @return  a collection of the result
 */
```

```
public static Collection QueryGetFlightsMonthlyReportPerSquadronPerStudent(
int month, int year, Squadron squad, Students stude){

    Calendar c1= new GregorianCalendar(year,month,1) ;
    if(month==11){
       year+=1;
       month=0;
    }
    else
       month+=1;
    Calendar c2= new GregorianCalendar(year,month,1) ;

    Date beginDate=c1.getTime();
    Date endDate=c2.getTime();
    //JOptionPane.showMessageDialog(null,beginDate+"----- "+endDate);

    Extent extent=pm.getExtent(Flights.class,true);
    String filter="date>=beginDate && date<endDate && squadron==squad &&
student=stude";
    Query query=pm.newQuery(extent,filter);
    query.declareImports("import java.util.Date");
    query.declareParameters("Date beginDate, Date  endDate, Squadron squad, Students
stude");

    Object[] parameters={beginDate,endDate, squad, stude};
    Collection result=(Collection) query.executeWithArray(parameters);
    query.close(result);
    return result;


}


/** Query the flights table and search the results for a specific
 * instructor in a Squadron for a specific year and month
 * @param inst the instructor
 * @param year the year
 * @param month the month
 * @param squad the Squadron
 * @return a collection of the result
 */

public static Collection QueryGetFlightsMonthlyReportPerSquadronPerInstructor(
int month, int year, Squadron squad, Instructors instr){

    Calendar c1= new GregorianCalendar(year,month,1) ;
```

```
        if(month==11){
           year+=1;
           month=0;
        }
        else
           month+=1;
        Calendar c2= new GregorianCalendar(year,month,1) ;

        Date beginDate=c1.getTime();
        Date endDate=c2.getTime();
        //JOptionPane.showMessageDialog(null,beginDate+"----- "+endDate);

        Extent extent=pm.getExtent(Flights.class,true);
        String filter="date>=beginDate && date<endDate && squadron==squad &&
instructor=instr";
        Query query=pm.newQuery(extent,filter);
        query.declareImports("import java.util.Date");
        query.declareParameters("Date beginDate, Date  endDate, Squadron squad,
Instructors instr");

        Object[] parameters={beginDate,endDate, squad,instr};
        Collection result=(Collection) query.executeWithArray(parameters);
        query.close(result);
        return result;


    }


    /** Query the flights table and search the results for a specific
     * series
     * @param pm the  Persistence Manager
     * @param serie the Series
     *
     * @return  a collection of the result
     */
    public static Collection QueryGetFlightsPerSeries(PersistenceManager pm, Series
serie){


        Extent extent=pm.getExtent(Flights.class,true);
        String filter="student.series==serie";
        Query query=pm.newQuery(extent,filter);
        //query.declareImports("import java.util.Date");
        query.declareParameters("Series serie");
        Collection result=(Collection) query.execute(serie);
```

```
    query.close(result);
    return result;


}


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
//////////////////////////INSTRUCTORS QUERY/////////////////////////////////
////////////////////////////////////////////////////////////////////////////
///INSTRUCTORS QUERY//////////////////////////////

/** Query the Instructors table and search the results for a specific
 * squadron
 * @param pm the  Persistence Manager
 * @param squad the Squadron
 *
 * @return  a collection of the result
 */

public static Collection QueryGetInstructorsPerSquadron(PersistenceManager pm,
Squadron squad){


    Extent extent=pm.getExtent(Instructors.class,true);
    String filter="squadron==squad";
    Query query=pm.newQuery(extent,filter);
    //query.declareImports("import java.util.Date");
    query.declareParameters("Squadron squad");
    Collection result=(Collection) query.execute(squad);
    query.close(result);
    return result;


}


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
//////////////////////////STUDENTS QUERY////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
///STUDENTS QUERY//////////////////////////////

/** Query the Students table and search the results for a specific
 * squadron
```

```
     * @param pm the  Persistence Manager
     * @param squad the Squadron
     *
     * @return  a collection of the result
     */
    public static Collection QueryGetStudentsPerSquadron(PersistenceManager pm,
Squadron squad){


        Extent extent=pm.getExtent(Students.class,true);
        String filter="squadron==squad";
        Query query=pm.newQuery(extent,filter);

        query.declareParameters("Squadron squad");
        Collection result=(Collection) query.execute(squad);
        query.close(result);
        return result;


    }


    /** Query the Students table and search the results for a specific
     * series
     * @param pm the Persistence Manager
     * @param serie the Series
     *
     * @return  a collection of the result
     */
    public static Collection QueryGetStudentsPerSeries(PersistenceManager pm, Series
serie){


        Extent extent=pm.getExtent(Students.class,true);
        String filter="series==serie";
        Query query=pm.newQuery(extent,filter);

        query.declareParameters("Series serie");
        Collection result=(Collection) query.execute( serie);
        query.close(result);
        return result;


    }

    ////////////////////////////////////////////////////////////////////////////
```

```
    ////////////////////////////////////////////////////////////////////////////
    ////////////////////////////SQUADRONS QUERY/////////////////////////////////////
    ////////////////////////////////////////////////////////////////////////////
    ///SQUADRONS QUERY//////////////////////////////

    /** Query the Squadron table and search the results
     * @param pm the Persistence Manager
     *
     * @return a collection of the result
     */
    public static Collection QueryGetSquadrons(PersistenceManager pm ){


        Extent extent=pm.getExtent(Squadron.class,true);

        Query query=pm.newQuery(extent);


        Collection result=(Collection) query.execute();
        query.close(result);
        return result;


    }
    ////////////////////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////////////////////////
    ////////////////////////////SERIES QUERY/////////////////////////////////////////
    ////////////////////////////////////////////////////////////////////////////
    ///SERIES QUERY/////////////////////////////////////////////////////////////////

    /** Query the Stadio table and search the results
     * for a specific series
     * @param pm the Persistence Manager
     * @param serie the Series
     *
     * @return a collection of the result
     */
    public static Collection QueryGetStadioPerSeries(PersistenceManager pm, Series serie
){


        Extent extent=pm.getExtent(Stadio.class,true);
        String filter="series==serie";
        Query query=pm.newQuery(extent,filter);

        query.declareParameters("Series serie");
```

```
        Collection result=(Collection) query.execute(serie);
        query.close(result);
        return result;


    }

}
```

### 6. The Class: ReportsForm.java

ReportsForm

```
/**
 * <p>Title: ReportsForm.java</p>
 * <p>Description: Creates THE gui FOR THE REPRESENTATION OF THE
 *        REPORTS OF THE VARIOUS TABLES </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     CS JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */


package test;
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
import javax.jdo.JDOHelper;                          ///////
import javax.jdo.PersistenceManagerFactory; ///////
import javax.jdo.PersistenceManager;           ///////
import javax.jdo.Transaction;                        ///////
import javax.jdo.Query;                              ///////
import java.io.FileInputStream;                      ///////
import java.io.IOException;                         ///////
import java.io.InputStream;                         ///////
import javax.jdo.spi.JDOImplHelper;           ///////
import javax.jdo.*;                                 ////////
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////


import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
```

```
import company.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.JComboBox;

import java.util.*;
import java.text.DecimalFormat;

class ReportsForm extends JFrame implements ActionListener, MouseListener{

   //.....
   //.....
   private static PersistenceManagerFactory pmf;
   private static PersistenceManager pm;
   private static Transaction tx;
   private static Query q;
   private static Collection result;
   //.....
   //.....


   /**
    * The top level panel used in showing a dialog using JOptionPane's
    * class method showOptionDialog.
    */
   private JPanel         topPanel;
   /**
    * The formpanel level panel used in showing the current form
    * that displays data
    */
   private JPanel        formPanel;

   /**
    * The squadronComboBox is a JComboBox that holds all
    *  the squadrons we can choose from for various \
    * queries
    */
   private JComboBox      squadronComboBox;

   /**
    * The seriesComboBox is a JComboBox that holds all
    *  the series we can choose from for various \
    * queries
```

```
 */
private JComboBox      seriesComboBox;

/**
 * The yearComboBox is a JComboBox that holds all
 *  the years we can choose from for various \
 * queries
 */
private JComboBox      yearComboBox;

/**
 * The monthComboBox is a JComboBox that holds all
 *  the months we can choose from for various \
 * queries
 */
private JComboBox      monthComboBox;

/**
 * The dateSpinner is a JSpinner that
 *   we can choose various dates for various \
 *      queries
 */
private JSpinner                dateSpinner;

/**is the JDOImplHelper
 */
JDOImplHelper myJDOImplHelper;
/**
 * Default frame width
 */
private static final int FRAME_WIDTH    = 800;

/**
 * Default frame height
 */
private static final int FRAME_HEIGHT   = 480;

/**
 * X coordinate of the frame default origin point
 */
private static final int FRAME_X_ORIGIN = 150;

/**
 * Y coordinate of the frame default origin point
 */
private static final int FRAME_Y_ORIGIN = 250;
```

```
   /**
    *CONSTRUCTOR
    */
   public ReportsForm(  ) {


      try {
         pmf = getPMF( “lido_mysql.properties”);
         pm = pmf.getPersistenceManager();
         tx = pm.currentTransaction();
         Container contentPane=getContentPane( );
         myJDOImplHelper=JDOImplHelper.getInstance();
         contentPane.setLayout(new BorderLayout());
         topPanel= new JPanel();

         squadronComboBox=new
JComboBox(getTheDataForComboBoxes(Squadron.class).toArray());

         seriesComboBox=new
JComboBox(getTheDataForComboBoxes(Series.class).toArray());

         yearComboBox=new JComboBox();

         Object[] aryear=new Object[40];
         for(int i=0;i<40;i++)
            aryear[i]= new Integer(2004+i);
         yearComboBox=new JComboBox(aryear);

         Object[]
armonth={“Jan”,”Feb”,”March”,”April”,”May”,”June”,”July”,”August”,
         “September”,”Octomber”,”November”,”December”};
         monthComboBox=new JComboBox(armonth);

         topPanel.add(squadronComboBox);
         topPanel.add(seriesComboBox);
         topPanel.add(monthComboBox);
         topPanel.add(yearComboBox);
         dateSpinner=new JSpinner(new SpinnerDateModel() ) ;
         topPanel.add(dateSpinner);
         contentPane.add(topPanel, BorderLayout.NORTH );
         formPanel=new JPanel();
         contentPane.add(formPanel, BorderLayout.CENTER );

         contentPane.add(createToolBar( ),BorderLayout.WEST);
         contentPane.add(createToolBar( ),BorderLayout.EAST);
```

```
    }
    catch (Exception e){e.printStackTrace();};

}
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////


/**gets data form a table that is represented by
 * a class and store them in list boxes
 * @param dataclassName the class
 * @return the result set containing the data
 */
public Collection getTheDataForListBoxes(String dataclassName){

    Collection sresult;
    boolean wasActive=true;

    if(!tx.isActive()){
        tx.begin();
        wasActive=false;
    }
    q = pm.newQuery(dataclassName);
    sresult = (Collection) q.execute();
    tx.commit();
    if(wasActive)
        tx.begin();

    return sresult;


}

/**gets data form a table that is represented by
 * a class and store them in combo boxes
 * @param dataclass the class
 * @return the result set containing the data
 */
public static Collection getTheDataForComboBoxes(Class dataclass){
    Collection sresult;
    boolean wasActive=true;
    if(!tx.isActive()){
        tx.begin();
        wasActive=false;
    }
```

```
    q = pm.newQuery(dataclass);
    sresult = (Collection) q.execute();

    tx.commit();
    if(wasActive)
        tx.begin();

    return sresult;

}
////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
//---------------------------------
//      Main method
//---------------------------------
public static void main(String[] args) {
    SquadronsForm f=new  SquadronsForm(  );
    f.pack();
    f.setVisible(true);
}

////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////


//---------------------------------
//    actionPerformed
//---------------------------------


public void actionPerformed(ActionEvent event){
    if (event.getActionCommand().equals("exerciseSampleQuery")){

        testTheQuery( event);
    }

}


/**
 * Creates a toolbar with button for every form
 * we are going to display
 *
 */
private JToolBar createToolBar( ) {
```

```
JButton button;
JToolBar toolbar = new JToolBar();

toolbar.addSeparator();



button = toolbar.add(
new AbstractAction("Instructors Monthly") {
   public void actionPerformed(ActionEvent e) {
      Object object=squadronComboBox.getSelectedItem();
      if(object!=null){
         Squadron squad=(Squadron) object;
         int year=  ((Integer) yearComboBox.getSelectedItem()).intValue();
         int month=monthComboBox.getSelectedIndex();
         //showInstructors( squad);
         showInstructorsMonthlyHours( squad, month,  year);
      }
   }
} );
button.setToolTipText("Instructors Monthly");
toolbar.addSeparator();



button = toolbar.add(
new AbstractAction("Students Monthly") {
   public void actionPerformed(ActionEvent e) {
      Object object=squadronComboBox.getSelectedItem();
      if(object!=null){
         Squadron squad=(Squadron) object;
         int year=((Integer) yearComboBox.getSelectedItem()).intValue();
         int month=monthComboBox.getSelectedIndex();

         showStudentsMonthlyHours( squad, month,  year);

      }


   }
} );

toolbar.addSeparator();

button = toolbar.add(
new AbstractAction("Arcrafts Daily") {
```

```
   public void actionPerformed(ActionEvent e) {
      Object object=squadronComboBox.getSelectedItem();
      if(object!=null){
         Squadron squad=(Squadron) object;
         Date d =(Date) dateSpinner.getValue()  ;
         showAircraftsDailyHoursPerSquadron( squad, d );
      }


   }
} );

toolbar.addSeparator();

button = toolbar.add(
new AbstractAction("Aicrafts Monthly") {
   public void actionPerformed(ActionEvent e) {
      Object object=squadronComboBox.getSelectedItem();
      if(object!=null){
         Squadron squad=(Squadron) object;
         int year=((Integer) yearComboBox.getSelectedItem()).intValue();
         int month=monthComboBox.getSelectedIndex();

         showAircraftsMonthlyHoursPerSquadron( squad, month,  year);
      }
   }
} );
button.setToolTipText("Aicrafts Monthly");
toolbar.addSeparator();

button = toolbar.add(
new AbstractAction("Squadron's Monthly") {
   public void actionPerformed(ActionEvent e) {
      Object object=squadronComboBox.getSelectedItem();
      if(object!=null){
         Squadron squad=(Squadron) object;
         int year=  ((Integer) yearComboBox.getSelectedItem()).intValue();
         int month=monthComboBox.getSelectedIndex();

         showSquadronsMonthlyHours( squad, month,  year);
      }
   }
} );
button.setToolTipText(" Squadron's Monthly");
toolbar.addSeparator();
```

```
button = toolbar.add(
new AbstractAction("All Squad. Monthly") {
   public void actionPerformed(ActionEvent e) {
      Object object=squadronComboBox.getSelectedItem();
      if(object!=null){
         Squadron squad=(Squadron) object;
         int year=((Integer) yearComboBox.getSelectedItem()).intValue();
         int month=monthComboBox.getSelectedIndex();

         showALLSquadronsMonthlyHours( squad, month,  year);
      }
   }
} );
button.setToolTipText(" Squadron's Monthly");
toolbar.addSeparator();

button = toolbar.add(
new AbstractAction("JDO JDBC test") {
   public void actionPerformed(ActionEvent e) {
      showGROUPBYInstructorsHours( );
   }
} );
button.setToolTipText(" JDO JDBC test");
toolbar.addSeparator();

button = toolbar.add(
new AbstractAction("add 100 flights") {
   public void actionPerformed(ActionEvent e) {
      add100Flight();

   }
} );
button.setToolTipText(" add 100 flights");
toolbar.addSeparator();

button = toolbar.add(
new AbstractAction("Magic ") {
   public void actionPerformed(ActionEvent e) {
      Object object=seriesComboBox.getSelectedItem();
      if(object!=null){
         Series series=(Series) object;

         showALLExcercisesPerSeries( series );
      }
   }
} );
```

```
        button.setToolTipText(“ Magic”);
        toolbar.addSeparator();

        button = toolbar.add(
        new AbstractAction(“Magic 2”) {
           public void actionPerformed(ActionEvent e) {
              Object object=seriesComboBox.getSelectedItem();
              if(object!=null){
                 Series series=(Series) object;

                 showALLExcercisesPerSeriesForEVERYStudent(series );
              }
           }
        } );
        button.setToolTipText(“ Magic”);
        toolbar.addSeparator();




        toolbar.setLayout(new GridLayout(0, 1));


        return toolbar;
     }


     //.....................................................................................
     //.....................................................................................
     /**gets the PersistenceManagerFactory
      * @param pathteh url
      * @return  the PersistenceManagerFactory
      */
     public static PersistenceManagerFactory getPMF(String path) {
        try {
           InputStream propStream =
           new FileInputStream(path);
           Properties props = new Properties();
           props.load(propStream);
           //!!!!!!ti kanv edv!!!!!
           props.put(“javax.jdo.PersistenceManagerFactoryClass”,
“com.libelis.lido.PersistenceManagerFactory”);

           return JDOHelper.getPersistenceManagerFactory(props);
        } catch (IOException e) {
           e.printStackTrace();
```

```
        System.exit(-1);
        return null;
    }

}


/**
 * @param e  */
public void mouseClicked(MouseEvent e) {

}

/**
 * @param e  */
public void mousePressed(MouseEvent e) {

}

/**
 * @param e  */
public void mouseReleased(MouseEvent e) {

}
/**
 * @param e  */
public void mouseEntered(MouseEvent e) {

}
/**
 * @param e  */
public void mouseExited(MouseEvent e) {

}
//////////////////////////////////////////////////////


/**Shows the exercise per series for every student
 * @param series  the series
 */
public void showALLExcercisesPerSeriesForEVERYStudent(Series series ){

    Object[]arStudent = QueryForm.QueryGetStudentsPerSeries(pm,  series).toArray();

    Object[][][][]arStudentExer=new Object[arStudent.length][][][];
```

```
Object[][][] arExer=getALLExcercisesFromSeries( series );

Object[][][] arResult=new Object[arExer.length][][];

//create an equal array with zero values
for(int i=0;i<arResult.length;i++){
   arResult[i]=new Object[arExer[i].length][];
   for(int j=0;j<arResult[i].length;j++){
      arResult[i][j]=new Object[  arExer[i][j].length  ];
      Arrays.fill( arResult[i][j], "“ ) ;
   }
}


for(int i=0;i<arStudentExer.length;i++){
   arStudentExer[i]=arrayCopy( arResult);
}

Collection c=QueryForm.QueryGetFlightsPerSeries( pm,  series);


Iterator iter=c.iterator();

while(iter.hasNext()){
   Flights flight=( (Flights) iter.next() );
   Students student=flight.getStudent();
   Exercise exercise=flight.getExercise();
   if(student!=null && exercise!=null){
      for (int i=0;i<arStudent.length;i++)
         if(( arStudent[i]).equals(student) ){
            //  JOptionPane.showMessageDialog(null,"lama");
            findAndInformTheExercise( arExer, arStudentExer[i],  flight);
         }
   }
}


JTabbedPane tabs=new JTabbedPane();
for(int i=0;i<arExer.length;i++){
   Stadio  stadio=(Stadio ) arExer[i][0][0];
   JTabbedPane tabcat=new JTabbedPane();
   tabs.addTab(stadio.getId(),tabcat);
   for(int j=0;j<arExer[i].length;j++){
      Categories categories= (Categories) arExer[i][j][1];
      if (categories!=null){
         ////////////////////////////////
```

```
                Object[][] aro=new Object[arStudent.length][];
                for(int r=0;r<arStudent.length;r++){
                   aro[r]= arStudentExer[r][i][j];
                   aro[r][0]=arStudent[r];
                   //JOptionPane.showMessageDialog(null,aro[r]);
                }
                Object[] arColumn=arrayCopy( arExer[i][j]);
                arColumn[0]="Student";
                arColumn[1]="Category";
                for(int h=2;h<arColumn.length;h++)
                   arColumn[h]=((Exercise) arColumn[h]).getId();

                JTable table= new JTable(aro,    arColumn );
                tabcat.addTab(categories.getId(),new JScrollPane(table) );
             }
          }
       }
       showTheForm( tabs);

    }
    /**find the exercise type and stores them in a array
     * @param arExer an array containing the exrcise
     * @param arStudentExer an array containing the exercises that
     * the student has fly
     * @param flight the current flight
     */
    public void findAndInformTheExercise(Object[][][] arExer,Object[][][] arStudentExer,
Flights flight){
       if (flight.getExercise()!=null)
          for(int j=0;j<arExer.length;j++)
             for(int k=0;k<arExer[j].length;k++)
                for(int m=0;m<arExer[j][k].length;m++){
                   Object ex= arExer[j][k][m] ;
                   if(ex!=null)
                      if(  ex.equals(flight.getExercise() )  ){

                         arStudentExer[j][k][m] =flight.getSpecialType();

                         break;
                      }
                }

    }
    ////////////////////////////////////////////////////
    ////////////////////////////////////////////////////
    ////////////////////////////////////////////////////
```

```
/**Creates a copy of an dimensional array
 * @param aray the source array
 * @return a copy of an array */
public Object[] arrayCopy(Object[] aray){
   Object[] ar=new Object[aray.length];
   for(int i=0;i<aray.length;i++)
      ar[i]=aray[i];

   return ar;
}
/**Creates a copy of a two-dimensional array
 * @param aray the source array
 * @return a copy of a two-dimensional array
 */
public Object[][] arrayCopy(Object[][] aray){
   Object[][] ar=new Object[aray.length][];
   for(int i=0;i<aray.length;i++){
      ar[i]=new Object[aray[i].length];
      for(int j=0;j<aray[i].length;j++)
         ar[i][j]=aray[i][j];
   }
   return ar;
}
/**Creates a copy of a 3-dimensional array
 * @param aray
 * @return a copy of a 3-dimensional array
 */
public Object[][][] arrayCopy(Object[][][] aray){
   Object[][][] ar=new Object[aray.length][][];
   for(int i=0;i<aray.length;i++){
      ar[i]=new Object[aray[i].length][];
      for(int j=0;j<aray[i].length;j++){
         ar[i][j]=new Object[aray[i][j].length];
         for(int k=0;k<aray[i][j].length;k++)

            ar[i][j][k]=aray[i][j][k];
      }
   }
   return ar;
}
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
/**Shows all the exercises for a specific
 * series
 * @param series the series
```

```
 */
 public void showALLExcercisesPerSeries(Series series ){

    Object[][][] arExer=getALLExcercisesFromSeries( series );

    JTabbedPane tabs=new JTabbedPane();
    for(int i=0;i<arExer.length;i++){
       Stadio  stadio=(Stadio ) arExer[i][0][0];
       JTabbedPane tabcat=new JTabbedPane();
       tabs.addTab(stadio.getId(),tabcat);
       for(int j=0;j<arExer[i].length;j++){
          Categories categories= (Categories) arExer[i][j][1];
          if (categories!=null)
             tabcat.addTab(categories.getId(),new JScrollPane(new JList(arExer[i][j])));
       }
    }
    showTheForm( tabs);

 }

 /**Gets all the exercises for a specific
  * series
  * @param series
  * @return a 3 dimensional array containing the exercises per
series per category*/
 public  Object[][][] getALLExcercisesFromSeries(Series series ){
    Collection c=QueryForm.QueryGetStadioPerSeries(pm,  series );

    Iterator iter=c.iterator();

    Object[][][] arExer=new Object[c.size()][][];
    int is=-1;
    int ic,ie;
    while(iter.hasNext()){
       Stadio stadio=( (Stadio) iter.next() );
       Iterator itCat=stadio.getCategories().iterator();
       is++; ic=-1;
       arExer[is]=new Object[stadio.getCategories().size()+1][2];
       arExer[is][0][0]=stadio;
       while(itCat.hasNext()){
          Categories categories=( (Categories) itCat.next() );
          Iterator itExc=categories.getExercise().iterator();
          ic++; ie=-1;
       //to avoid null
          arExer[is][ic]=new Object[categories.getExercise().size()+2];
```

```
            arExer[is][ic][0]=stadio;
            arExer[is][ic][1]=categories;

            while(itExc.hasNext()){
               Exercise exercise=( (Exercise) itExc.next() );
               ie++;


               arExer[is][ic][ie+2]=exercise;
            }
         }
      }

      return arExer;
   }
   ////////////////////////////////////////////////////
   /**Show the monthly hours that have been flown for all squadrons
    *
    * @param squad the squadron
    * @param month the month
    * @param year the year
    */
   public void showALLSquadronsMonthlyHours(Squadron squad,int month, int year ){
      Collection c=QueryForm.QueryGetSquadrons( pm );

      Object[] arSquadrons= c.toArray();

      double[] arHours=new double[arSquadrons.length];
      Arrays.fill( arHours, 0 ) ;
      int[] arFligths=new int[arSquadrons.length];
      Arrays.fill( arFligths, 0 ) ;

      Object[][] arSquadronsHour=new Object[arSquadrons.length][3];




      Collection c2=QueryForm.QueryGetFlightsMonthlyReportAllSquadrons(pm,month,
year);
      Iterator iter=c2.iterator();

      while(iter.hasNext()){
         Flights fly=( (Flights) iter.next() );
         Squadron sq= fly.getSquadron();
```

```
        if(sq!=null)
           for(int i=0;i<arSquadrons.length; i++)

              if( fly.getSquadron().equals((Squadron) arSquadrons[i]) ){

                 arHours[i]+= fly.getEndurance();
                 arFligths[i]+=1;
              }
     }

     for(int i=0;i<arSquadronsHour.length; i++){
        arSquadronsHour[i][0]= arSquadrons[i];
        arSquadronsHour[i][1]=new Double(arHours[i]);
        arSquadronsHour[i][2]=new Integer(arFligths[i]);


     }

     Object[] ColumnHeader={"Squadron",  "hours","# fligths"};

     JTable instructorsTable=new JTable(arSquadronsHour,ColumnHeader);
     showTheForm( instructorsTable);


  }

  ////////////////////////////////////////////////////

  ////////////////////////////////////////////////////
  /**Show the monthly hours that have been flown for a
   * specific squadrons
   * @param squad
   * @param month
   * @param year  */
  public void showSquadronsMonthlyHours(Squadron squad,int month, int year ){

     Collection  c=  QueryForm.QueryGetFlightsMonthlyReportPerSquadron(pm,month,
year, squad);

     Object[][] arSquadronHour=new Object[1][3];
     arSquadronHour[0][0]=squad;
     arSquadronHour[0][1]=new Integer(0);
     arSquadronHour[0][2]=new Double(0);

     Iterator iter=c.iterator();
```

```
        while(iter.hasNext()){
           Flights fly=( (Flights) iter.next() );

           int countFlights= ((Integer) arSquadronHour[0][1] ).intValue();
           arSquadronHour[0][1]=new Integer(countFlights+1);

           double hours= ((Double) arSquadronHour[0][2] ).doubleValue();
           arSquadronHour[0][2]=new Double( hours + fly.getEndurance() )  ;
        }


        Object[] ColumnHeader={"Squadron","# flights", "hours"};

        JTable instructorsTable=new JTable(arSquadronHour,ColumnHeader);
        showTheForm( instructorsTable);


    }

    ////////////////////////////////////////////////////

    ////////////////////////////////////////////////////
    /**Shows the hours of all the students of a specific squadron
     * @param squad the squadron
     * @param month the month
     * @param year the year */
    public void showStudentsMonthlyHours(Squadron squad,int month, int year){
        Collection c=QueryForm.QueryGetStudentsPerSquadron(pm, squad);

        Object[] arInstructors= c.toArray();

        double[] arHours=new double[arInstructors.length];
        Arrays.fill( arHours, 0 ) ;
        int[] arFligths=new int[arInstructors.length];
        Arrays.fill( arFligths, 0 ) ;

        Object[][] arInstrHour=new Object[arInstructors.length][4];

        Collection   c2=QueryForm.QueryGetFlightsMonthlyReportPerSquadron(pm,month,
year, squad);
        Iterator iter=c2.iterator();

        while(iter.hasNext()){
           Flights fly=( (Flights) iter.next() );

           for(int i=0;i<arInstructors.length; i++)
```

```
            if( fly.getStudent().equals((Students) arInstructors[i]) ){

               arHours[i]+= fly.getEndurance();
               arFligths[i]+=1;
            }
         }

         for(int i=0;i<arInstrHour.length; i++){
            arInstrHour[i][0]=((Students) arInstructors[i]).getId();
            arInstrHour[i][1]=((Students) arInstructors[i]).getLastName();
            arInstrHour[i][2]=new Double(arHours[i]);
            arInstrHour[i][3]=new Integer(arFligths[i]);


         }

         Object[] ColumnHeader={“ID”, “last name”, “hours”,”# fligths”};

         JTable instructorsTable=new JTable(arInstrHour,ColumnHeader);
         showTheForm( instructorsTable);

      }
      ////////////////////////////////////////////////////
      /**Shows the Aircraft's monthly hours for a
       * specific squadron
       * @param squad the squadron
       * @param month the month
       * @param year the year
       */
      public void showAircraftsMonthlyHoursPerSquadron(Squadron squad,int month, int year ){

         Collection c= QueryForm.QueryGetFlightsMonthlyReportPerSquadron(pm,month, year, squad);

         LinkedList aircraftList=new LinkedList();
         LinkedList aircraftHours=new LinkedList();
         LinkedList aircraftFlights=new LinkedList();

         Iterator iter=c.iterator();

         while(iter.hasNext()){
            Flights fly=( (Flights) iter.next() );

            int i=aircraftList.indexOf(fly.getAircraft() ) ;
            if(i<0){
```

```
            aircraftList.addLast(fly.getAircraft());
            aircraftHours.addLast(new Double(fly.getEndurance() ));
            aircraftFlights.addLast(new Integer(1));
        }
        else{
            double hours= ((Double) aircraftHours.get(i) ).doubleValue();
            aircraftHours.set(i, new Double( hours + fly.getEndurance()  ));

            int countFlights= ((Integer) aircraftFlights.get(i) ).intValue();
            aircraftFlights.set(i, new Integer(countFlights+1));

        }
    }

    Object[] arAircraft=aircraftList.toArray();
    Object[][] arAircraftHour=new Object[arAircraft.length][4];
    for(int i=0;i<arAircraftHour.length; i++){
        arAircraftHour[i][0]=((Aircraft) arAircraft[i]).getNum();
        arAircraftHour[i][1]=((Aircraft) arAircraft[i]).getType();
        arAircraftHour[i][2]=aircraftHours.get(i);
        arAircraftHour[i][3]=aircraftFlights.get(i);

    }

    Object[] ColumnHeader={“Aircraft ID”, “Aircraft type”, “hours”,”# flights”};

    JTable instructorsTable=new JTable(arAircraftHour,ColumnHeader);
    showTheForm( instructorsTable);


}

//////////////////////////////////////////////////////
/**Shows the Aircraft's hours for a
 * specific squadron during a specific date
 * @param squad the squadron
 * @param d the date
 */
public void showAircraftsDailyHoursPerSquadron(Squadron squad,Date d ){

    Collection c= QueryForm.QueryGetFlightsDailyReportPerSquadron(pm, d, squad);

    LinkedList aircraftList=new LinkedList();
    LinkedList aircraftHours=new LinkedList();
    LinkedList aircraftFlights=new LinkedList();
```

```
   Iterator iter=c.iterator();

   while(iter.hasNext()){
      Flights fly=( (Flights) iter.next() );

      int i=aircraftList.indexOf(fly.getAircraft() ) ;
      if(i<0){
         aircraftList.addLast(fly.getAircraft());
         aircraftHours.addLast(new Double(fly.getEndurance() ));
         aircraftFlights.addLast(new Integer(1));
      }
      else{
         double hours= ((Double) aircraftHours.get(i) ).doubleValue();
         aircraftHours.set(i, new Double( hours + fly.getEndurance()  ));

         int countFlights= ((Integer) aircraftFlights.get(i) ).intValue();
         aircraftFlights.set(i, new Integer(countFlights+1));

      }
   }

   Object[] arAircraft=aircraftList.toArray();
   Object[][] arAircraftHour=new Object[arAircraft.length][4];
   for(int i=0;i<arAircraftHour.length; i++){
      arAircraftHour[i][0]=((Aircraft) arAircraft[i]).getNum();
      arAircraftHour[i][1]=((Aircraft) arAircraft[i]).getType();
      arAircraftHour[i][2]=aircraftHours.get(i);
      arAircraftHour[i][3]=aircraftFlights.get(i);

   }

   Object[] ColumnHeader={“Aircraft ID”, “Aircraft type”, “hours”,”# flights”};

   JTable instructorsTable=new JTable(arAircraftHour,ColumnHeader);
   showTheForm( instructorsTable);


}
//////////////////////////////////////////////////////
/**Shows the instructors monthly hours
 * of a specific squadron
 * @param squad the squadron
 * @param month the month
 * @param year the year */
public void showInstructorsMonthlyHours(Squadron squad,int month, int year){
   Collection c=QueryForm.QueryGetInstructorsPerSquadron(pm, squad);
```

```
        Object[] arInstructors= c.toArray();

        double[] arHours=new double[arInstructors.length];
        Arrays.fill( arHours, 0 ) ;
        int[] arFligths=new int[arInstructors.length];
        Arrays.fill( arFligths, 0 ) ;

        Object[][] arInstrHour=new Object[arInstructors.length][4];

        Collection   c2=QueryForm.QueryGetFlightsMonthlyReportPerSquadron(pm,month,
year, squad);
        Iterator iter=c2.iterator();

        while(iter.hasNext()){
           Flights fly=( (Flights) iter.next() );

           for(int i=0;i<arInstructors.length; i++)
              if( fly.getInstructor().equals((Instructors) arInstructors[i]) ){

                 arHours[i]+= fly.getEndurance();
                 arFligths[i]+=1;

              }
        }

        for(int i=0;i<arInstrHour.length; i++){
           arInstrHour[i][0]=((Instructors) arInstructors[i]).getId();
           arInstrHour[i][1]=((Instructors) arInstructors[i]).getLastName();
           arInstrHour[i][2]=new Double(arHours[i]);
           arInstrHour[i][3]=new Integer(arFligths[i]);


        }

        Object[] ColumnHeader={“ID”, “last name”, “hours”,”# flights”};

        JTable instructorsTable=new JTable(arInstrHour,ColumnHeader);
        showTheForm( instructorsTable);

     }

     /**pure SQL query used for accesing the performance
      * of JDO in contrast to JDBC
      */
     public void showGROUPBYInstructorsHours1( ){
```

```
long[] arLong=new long[3];


arLong[0]=PureSQL.showGROUPBYInstructorsHours(true);
arLong[1]=PureSQL.showGROUPBYInstructorsHours(false );


LinkedList lInstructor=new LinkedList();
LinkedList lHours=new LinkedList();

long timeStart=System.currentTimeMillis() ;
System.out.println( timeStart);

Collection c2=QueryForm.QueryGetFlights( pm);
Iterator iter=c2.iterator();

while(iter.hasNext()){
   Flights fly=( (Flights) iter.next() );
   Instructors inst=fly.getInstructor();
   //instructor is not there
   if(  !lInstructor.contains(inst )){
      lInstructor.addLast(inst);
      lHours.addLast(new Double(fly.getEndurance()));
   }
   else{
      Object obj=lHours.get(lInstructor.indexOf(inst));
      obj=new Double(
      ((Double) obj).doubleValue()+
      fly.getEndurance()
      );


   }
}

long timeStop=System.currentTimeMillis() ;

System.out.println( timeStop);

System.out.println(timeStop-timeStart);

arLong[2]=timeStop-timeStart;

Collection c3=QueryForm.QueryGetFlightsSQL( pm);
Iterator iterc3=c3.iterator();
```

```
        while(iterc3.hasNext()){
           Flights fly=( (Flights) iterc3.next() );
        }

        String[] categories={"GROUP BY Iter:"+arLong[0], "GROUP BY
:"+arLong[1],"JDO:"+arLong[2]};
        MyBarChart bar=new MyBarChart(arLong, categories,"performance in milisec");
        bar.toForm("JDO Performance for imitating a Group By query");

    }

    /**pure SQL query used for accessing the performance
     * of JDO in contrast to JDBC
     */

    public void showGROUPBYInstructorsHours2( ){

        long[] arLong=new long[3];


        arLong[0]=PureSQL.showGROUPBYInstructorsHours(true);
        arLong[1]=PureSQL.showGROUPBYInstructorsHours(false );

        ////////////////////////////////////////////////////////////////////////////////////////////////

        LinkedList lInstructor=new LinkedList();
        LinkedList lHours=new LinkedList();

        long timeStart=System.currentTimeMillis() ;
        System.out.println( timeStart);

        //Collection c2=QueryForm.QueryGetFlightsSQL( pm);
        Collection c2=QueryForm.QueryGetFlightsSQL(pm);
        pm.retrieveAll(c2);
        System.out.println(" sixe:"+c2.size());
        Iterator iter=c2.iterator();

        while(iter.hasNext()){
           Flights fly=( (Flights) iter.next() );
           double i=fly.getEndurance();
              /*Instructors inst=fly.getInstructor();
         //instructor is not there
              if(  !lInstructor.contains(inst )){
                    lInstructor.addLast(inst);
                    lHours.addLast(new Double(fly.getEndurance()));
              }
```

```
            else{
                Object obj=lHours.get(lInstructor.indexOf(inst));
                obj=new Double(
                    ((Double) obj).doubleValue()+
                    fly.getEndurance()
                     );
            }
             */
        }

        long timeStop=System.currentTimeMillis() ;

        System.out.println( timeStop);

        System.out.println(timeStop-timeStart);

        arLong[2]=timeStop-timeStart;



        String[] categories={"GROUP BY Iter:"+arLong[0], "GROUP BY
:"+arLong[1],"JDO sql:"+arLong[2]};
        MyBarChart bar=new MyBarChart(arLong, categories,"performance in milisec");
        bar.toForm("JDO Performance for imitating a Group By query");

    }

    /**pure SQL query used for accesing the performance
     * of JDO in contrast to JDBC
     */

    public void showGROUPBYInstructorsHours( ){

        long[] arLong=new long[3];


        arLong[0]=PureSQL.showSELECTQueryFlights(true);
        arLong[1]=PureSQL.showSELECTQueryFlights(false );

        ////////////////////////////////////////////////////////////////////////////////////////////////

        LinkedList lInstructor=new LinkedList();
        LinkedList lHours=new LinkedList();

        long timeStart=System.currentTimeMillis() ;
        System.out.println( timeStart);
```

```
    //Collection c2=QueryForm.QueryGetFlights( pm);
    Collection c2=QueryForm.QueryGetFlightsSQL(pm);
    //pm.retrieveAll(c2);
    System.out.println(" sixe:"+c2.size());
    Iterator iter=c2.iterator();

    while(iter.hasNext()){
       Flights fly=( (Flights) iter.next() );
       double i=fly.getEndurance();
          /*Instructors inst=fly.getInstructor();
     //instructor is not there
          if(  !lInstructor.contains(inst )){
                lInstructor.addLast(inst);
                lHours.addLast(new Double(fly.getEndurance()));
          }
          else{
                Object obj=lHours.get(lInstructor.indexOf(inst));
                obj=new Double(
                      ((Double) obj).doubleValue()+
                      fly.getEndurance()
                       );
          }
           */
    }

    long timeStop=System.currentTimeMillis() ;

    System.out.println( timeStop);

    System.out.println(timeStop-timeStart);

    arLong[2]=timeStop-timeStart;



    String[] categories={"JDBC SELECT ITER.:"+arLong[0], "JDBC SELECT
:"+arLong[1],"JDO SELECT:"+arLong[2]};
    MyBarChart bar=new MyBarChart(arLong, categories,"performance in milisec");
    bar.toForm("JDO Performance FOR SELECT Queries WITHOUT retrieveAll() -
sql- Tag  Iteration");

  }


  ////////////////////////////////////////////////////////////////////////////
```

```
/**Shows the instructors of a specific squadron
 * @param squad the squadron
 */
public void showInstructors(Squadron squad){
   Collection c=QueryForm.QueryGetInstructorsPerSquadron(pm, squad);

   Object[] arInstructors= c.toArray();

   int[] arHours=new int[arInstructors.length];
   Arrays.fill( arHours, 0 ) ;

   Object[][] arInstrHour=new Object[arInstructors.length][3];

   for(int i=0;i<arInstrHour.length; i++){
      arInstrHour[i][0]=((Instructors) arInstructors[i]).getId();
      arInstrHour[i][1]=((Instructors) arInstructors[i]).getLastName();
      arInstrHour[i][2]=new Integer(arHours[i]);


   }

   Object[] ColumnHeader={“ID”, “last name”, “hours”};

   JTable instructorsTable=new JTable(arInstrHour,ColumnHeader);
   showTheForm( instructorsTable);

}


/**displays the form inside a container
 *
 * @param container the container */
public void showTheForm(Container container){

   JScrollPane jsp=new JScrollPane(container);

   JFrame myframe =new JFrame();
   Container cont=myframe.getContentPane();
   cont.add(jsp);
   myframe.pack();
   myframe.setVisible(true);

}


/** adds a number of records to the tables
```

```
 * in order to calculate the performance of JDO
 */
public void add100Flight(){
   tx = pm.currentTransaction();
   for(int i=10000;i<12000; i++){
      tx.begin();
      Students    student=new Students( ""+i,"lname"+i," fname"+i,  1.0);
      Instructors  instructor= new Instructors(""+i, "lname"+i,"instr_name"+i,  1.0);
      AircraftType   aircrafttype=new AircraftType("T2-EBAck","DEception");
      Aircraft          aircraft= new Aircraft("0"+i,aircrafttype,  0.0);

      Flights    flight= new Flights( aircraft, instructor, student, new Date());
      //squadron=new Squadron("34"+i,"description-"+i);

      //pm.makePersistent(student);

      //pm.makePersistent(instructor);
      pm.makePersistent(flight);
      // pm.makePersistent(aircraft);
      //pm.makePersistent(aircrafttype);
      // pm.makePersistent(rank);

      try{
         tx.commit();
         //System.out.println("]");
      }
      catch(javax.jdo.JDOException e){
         System.out.println("malaka  "+e);
      }


   }


}

/** tests the query
 */
private void testTheQuery(ActionEvent e){

   JPanel l=new JPanel();
   JScrollPane jsp=new JScrollPane(l);

   JFrame myframe =new JFrame();
   Container cont=myframe.getContentPane();
   cont.add(jsp);
```

```
        myframe.pack();
        myframe.setVisible(true);

    }


}
```

### 7. The Class: SquadronsForm.java

SquadronsForm

```
/**
 * <p>Title: SquadronsForm.java</p>
 * <p>Description: Creates THE gui FOR THE REPRESENTATION OF THE
 *        DATA OF THE VARIOUS FORMS </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * Company:     CS JDO
 * @author Paschalis Zilidis
 * @version 1.0
 */

package test;
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
import javax.jdo.JDOHelper;                          ///////
import javax.jdo.PersistenceManagerFactory; ///////
import javax.jdo.PersistenceManager;           ///////
import javax.jdo.Transaction;                       ///////
import javax.jdo.Query;                              ///////
import java.io.FileInputStream;                   ///////
import java.io.IOException;                        ///////
import java.io.InputStream;                       ///////
import javax.jdo.spi.JDOImplHelper;          ///////
import javax.jdo.*;                                  ///////
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

import company.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.JComboBox;

import java.util.*;
import java.text.DecimalFormat;
```

```
class SquadronsForm extends JFrame implements ActionListener, MouseListener{

  //.....................................................
  //.....................................................
  private static PersistenceManagerFactory pmf;
  private static PersistenceManager pm;
  private static Transaction tx;
  private static Query q;
  private static Collection result;
  //.....................................................
  //.....................................................

  /**
   * The top level panel used in showing a dialog using JOptionPane's
   * class method showOptionDialog.
   */
  private JPanel         topPanel;
  /**
   * The formpanel level panel used in showing the current form
   * that displays data
   */
  private JPanel        formPanel;

  /**
   * The squadronComboBox is a JComboBox that holds all
   *  the squadrons we can choose from for various \
   * queries
   */
  private JComboBox      squadronComboBox;

  /**
   * The seriesComboBox is a JComboBox that holds all
   *  the series we can choose from for various \
   * queries
   */
  private JComboBox      seriesComboBox;
  /**
   * Default frame width
   */
  private static final int FRAME_WIDTH    = 800;

  /**
   * Default frame height
   */
```

```
    private static final int FRAME_HEIGHT   = 480;

    /**
     * X coordinate of the frame default origin point
     */
    private static final int FRAME_X_ORIGIN = 150;

    /**
     * Y coordinate of the frame default origin point
     */
    private static final int FRAME_Y_ORIGIN = 250;

    /**
     *  Constructor
     */
    public SquadronsForm(  ) {


        try {
            pmf = getPMF( "lido_mysql.properties");
            pm = pmf.getPersistenceManager();
            tx = pm.currentTransaction();
            Container contentPane=getContentPane( );

            contentPane.setLayout(new BorderLayout());
            topPanel= new JPanel();

            squadronComboBox=new
JComboBox(getTheDataForComboBoxes(Squadron.class).toArray());

            seriesComboBox=new
JComboBox(getTheDataForComboBoxes(Series.class).toArray());


            topPanel.add(squadronComboBox);
            topPanel.add(seriesComboBox);


            contentPane.add(topPanel, BorderLayout.NORTH );
            formPanel=new JPanel();
            contentPane.add(formPanel, BorderLayout.CENTER );

            contentPane.add(createToolBar( ),BorderLayout.WEST);
            contentPane.add(createToolBar( ),BorderLayout.EAST);

        }
```

```
    catch (Exception e){e.printStackTrace();};

}


/**gets data form a table that is represented by
 * a class and store them in list boxes
 * @param dataclassName the class
 * @return the result set containing the data
 */
public Collection getTheDataForListBoxes(String dataclassName){

    Collection sresult;

    if(!tx.isActive()){
        tx.begin();
    }
    q = pm.newQuery(dataclassName);
    sresult = (Collection) q.execute();
    tx.commit();

    return sresult;


}


/**gets data form a table that is represented by
 * a class and store them in combo boxes
 * @param dataclass the class
 * @return the result set containing the data
 */

public static Collection getTheDataForComboBoxes(Class dataclass){
    Collection sresult;

    if(!tx.isActive()){
        tx.begin();
    }
    q = pm.newQuery(dataclass);
    sresult = (Collection) q.execute();

    tx.commit();

    return sresult;
```

```
}
////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
//---------------------------------
//     Main method
//---------------------------------
public static void main(String[] args) {
   SquadronsForm f=new SquadronsForm(  );
   f.pack();
   f.setVisible(true);
}
////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////


//---------------------------------
//    actionPerformed
//---------------------------------

public void actionPerformed(ActionEvent event){
   if (event.getActionCommand().equals("exerciseSampleQuery")){

      testTheQuery( event);
   }

}


/**
 * Creates a toolbar with BUTTONS FOR THE
 * DISPLAY OF VARIOUS FORMS
 *
 */
private JToolBar createToolBar( ) {

   JButton button;
   JToolBar toolbar = new JToolBar();

   toolbar.addSeparator();
   final JButton jd=new JButton("flight");
   jd.addMouseListener(this);
   //ADD
   button = toolbar.add(
   new AbstractAction("Instructors") {
      public void actionPerformed(ActionEvent e) {
         Instructors object=new Instructors();
```

```
            LinkedList                    listOfObject=                    new
LinkedList(getTheDataForComboBoxes(Instructors.class));
            InstructorsForm sf=new InstructorsForm( pm, listOfObject,  object );

            getContentPane().remove(formPanel);
            formPanel.removeAll();

            sf.add(jd);

            formPanel.add(sf);

            getContentPane().add(formPanel);

            pack();
            repaint();
         }
      } );
      button.setToolTipText("Instructors");
      toolbar.addSeparator();

      button = toolbar.add(
      new AbstractAction("Students") {
         public void actionPerformed(ActionEvent e) {
            Students object=new Students();
            LinkedList                    listOfObject=                    new
LinkedList(getTheDataForComboBoxes(Students.class));

            InstructorsForm sf=new InstructorsForm( pm, listOfObject, object );
            String[] ar={"squadron","instructors"};


            Squadron sq=(Squadron) squadronComboBox.getSelectedItem();

            getContentPane().remove(formPanel);
            formPanel.removeAll();
            formPanel.add(sf);
            getContentPane().add(formPanel);

            pack();

            repaint();
         }
      } );
      button.setToolTipText("Students");
      toolbar.addSeparator();
```

```
      button = toolbar.add(
      new AbstractAction("Schedule") {
         public void actionPerformed(ActionEvent e) {
            Schedule schedule=new Schedule();

            Squadron squadron=new Squadron();
            LinkedList                    listOfObject=                    new
LinkedList(getTheDataForComboBoxes(Squadron.class));
            InstructorsForm sf=new InstructorsForm( pm, listOfObject, squadron,1 );

            //sf.next();

            sf.setSubformForTheField("schedule",schedule);
            InstructorsForm                    sfschedule=(InstructorsForm)
sf.getTheFieldContainerByFieldName("schedule");

            getContentPane().remove(formPanel);
            formPanel.removeAll();
            formPanel.add(sf);
            formPanel.add(sfschedule);

            getContentPane().add(formPanel);

            pack();
            repaint();

         }
      } );
      button.setToolTipText("schedule");
      toolbar.addSeparator();

      button = toolbar.add(
      new AbstractAction("flights") {
         public void actionPerformed(ActionEvent e) {
            Flights object=new Flights();
            LinkedList                    listOfObject=                    new
LinkedList(getTheDataForComboBoxes(Flights.class));
            InstructorsForm sf=new InstructorsForm( pm, listOfObject,  object );

            getContentPane().remove(formPanel);
            formPanel.removeAll();
            formPanel.add(sf);
            getContentPane().add(formPanel);

            pack();
```

```
            repaint();
         }
      } );
      button.setToolTipText("flights");
      toolbar.addSeparator();

      button = toolbar.add(new AbstractAction("Aircraft") {
         public void actionPerformed(ActionEvent e) {
            Aircraft object=new Aircraft();
            LinkedList                    listOfObject=                    new
LinkedList(getTheDataForComboBoxes(Aircraft.class));
            InstructorsForm sf=new InstructorsForm( pm, listOfObject,  object );
            getContentPane().remove(formPanel);
            formPanel.removeAll();
            formPanel.add(sf);
            getContentPane().add(formPanel);

            pack();



            repaint();
         }
      } );
      button.setToolTipText("Aircraft");
      toolbar.addSeparator();

      button = toolbar.add(
      new AbstractAction("Squadron") {
         public void actionPerformed(ActionEvent e) {
            Squadron object=new Squadron();
            LinkedList                    listOfObject=                    new
LinkedList(getTheDataForComboBoxes(Squadron.class));
            InstructorsForm sf=new InstructorsForm( pm, listOfObject,  object );


            getContentPane().remove(formPanel);
            formPanel.removeAll();
            formPanel.add(sf);
            getContentPane().add(formPanel);
            //getContentPane().add(sf, BorderLayout.CENTER );
            pack();
            repaint();
         }
      } );
```

```
        button.setToolTipText("Squadron");
        toolbar.addSeparator();

        button = toolbar.add(
        new AbstractAction("Series") {
           public void actionPerformed(ActionEvent e) {
              Series object=new Series();

              LinkedList listOfObject= new
LinkedList(getTheDataForComboBoxes(Series.class));
              InstructorsForm sf=new InstructorsForm( pm, listOfObject,  object ,1);
              Stadio stadio=new Stadio();
              sf.setSubformForTheField("stadio",stadio);
              InstructorsForm sfstadio=(InstructorsForm)
sf.getTheFieldContainerByFieldName("stadio");

              getContentPane().remove(formPanel);
              formPanel.removeAll();
              formPanel.add(sf);

              formPanel.add(sfstadio);

              getContentPane().add(formPanel);

              pack();
              repaint();
           }
        } );
        button.setToolTipText("Series");
        toolbar.addSeparator();

        button = toolbar.add(
        new AbstractAction("Stadio") {
           public void actionPerformed(ActionEvent e) {
              Stadio stadio=new Stadio();
              Series object=new Series();

              LinkedList listOfObject= new
LinkedList(getTheDataForComboBoxes(Series.class));

              InstructorsForm sf=new InstructorsForm( pm, listOfObject, object
,"stadio",stadio);


              getContentPane().remove(formPanel);
              formPanel.removeAll();
```

```
            formPanel.add(sf);
            getContentPane().add(formPanel);

            pack();
            repaint();
          }
        } );

        toolbar.addSeparator();
        button = toolbar.add(
        new AbstractAction("Test") {
           public void actionPerformed(ActionEvent e) {
            Stadio stadio=new Stadio();
            Series series=new Series();
            Categories categories=new Categories();
            Exercise exercise=new Exercise();

            LinkedList listOfObject= new
LinkedList(getTheDataForComboBoxes(Series.class));

            InstructorsForm sf=new InstructorsForm( pm, listOfObject,  series );


            sf.setSubformForTheField("stadio",stadio);
            InstructorsForm sfstadio=(InstructorsForm)
sf.getTheFieldContainerByFieldName("stadio");

            sfstadio.setSubformForTheField("categories",categories);


            InstructorsForm sfcateg=(InstructorsForm)
sfstadio.getTheFieldContainerByFieldName("categories");
            sfcateg.setSubformForTheField("exercise",exercise);


            getContentPane().remove(formPanel);
            formPanel.removeAll();
            JPanel l=new JPanel(new GridLayout(0,2,5,5));
            l.add(sf);
            l.add(sfstadio);
            l.add(sfcateg);
            l.add((InstructorsForm)
sfcateg.getTheFieldContainerByFieldName("exercise"));

            JScrollPane jsp=new JScrollPane(l);
```

```
            JFrame myframe =new JFrame();
            Container cont=myframe.getContentPane();
            cont.add(jsp);
            myframe.pack();
            myframe.setVisible(true);

            repaint();
         }
      } );

      toolbar.addSeparator();
      final JButton jbcat=new JButton("exerciseSampleQuery");
      jbcat.addActionListener(this);

      button = toolbar.add(
      new AbstractAction("Categories") {
         public void actionPerformed(ActionEvent e) {
            Categories object=new Categories();

            LinkedList                    listOfObject=                    new
LinkedList(getTheDataForComboBoxes(Categories.class));
            InstructorsForm sf=new InstructorsForm( pm, listOfObject,  object );
            getContentPane().remove(formPanel);
            formPanel.removeAll();

            sf.add(jbcat);
            formPanel.add(sf);
            getContentPane().add(formPanel);

            repaint();
            pack();

         }
      } );

      button.setToolTipText("Categories");
      toolbar.addSeparator();

      button = toolbar.add(
      new AbstractAction("Exercise") {
         public void actionPerformed(ActionEvent e) {
            Exercise object=new Exercise();

            LinkedList                    listOfObject=                    new
LinkedList(getTheDataForComboBoxes(Exercise.class));
            InstructorsForm sf=new InstructorsForm( pm, listOfObject,  object );
```

```
            getContentPane().remove(formPanel);
            formPanel.removeAll();
            formPanel.add(sf);
            getContentPane().add(formPanel);
            //getContentPane().add(sf, BorderLayout.CENTER );
            pack();
            repaint();
         }
      } );

      button.setToolTipText("Exercise");
      toolbar.addSeparator();

      button = toolbar.add(
      new AbstractAction("BasicStadio") {
         public void actionPerformed(ActionEvent e) {
            BaseStadio object=new BaseStadio();

            LinkedList                    listOfObject=                    new
LinkedList(getTheDataForComboBoxes(BaseStadio.class));
            InstructorsForm sf=new InstructorsForm( pm, listOfObject,  object );
            getContentPane().remove(formPanel);
            formPanel.removeAll();
            formPanel.add(sf);
            getContentPane().add(formPanel);
            //getContentPane().add(sf, BorderLayout.CENTER );
            pack();
            repaint();
         }
      } );

      button.setToolTipText("BasicStadio");
      toolbar.addSeparator();

      button = toolbar.add(
      new AbstractAction("BasicCategory") {
         public void actionPerformed(ActionEvent e) {
            BaseCategory object=new BaseCategory();

            LinkedList                    listOfObject=                    new
LinkedList(getTheDataForComboBoxes(BaseCategory.class));
            InstructorsForm sf=new InstructorsForm( pm, listOfObject,  object );
            getContentPane().remove(formPanel);
            formPanel.removeAll();
            formPanel.add(sf);
            getContentPane().add(formPanel);
```

```
            //getContentPane().add(sf, BorderLayout.CENTER );
            pack();
            repaint();
        }
    } );

    button.setToolTipText("BasicCategory");
    toolbar.addSeparator();

    button = toolbar.add(
    new AbstractAction("SpecialType") {
        public void actionPerformed(ActionEvent e) {
            SpecialType object=new SpecialType();

            LinkedList listOfObject= new
LinkedList(getTheDataForComboBoxes(SpecialType.class));
            InstructorsForm sf=new InstructorsForm( pm, listOfObject,  object );
            getContentPane().remove(formPanel);
            formPanel.removeAll();
            formPanel.add(sf);
            getContentPane().add(formPanel);
            //getContentPane().add(sf, BorderLayout.CENTER );
            pack();
            repaint();
        }
    } );

    button.setToolTipText("SpecialType");
    toolbar.addSeparator();

    button = toolbar.add(
    new AbstractAction("Ground course") {
        public void actionPerformed(ActionEvent e) {
            GroundCourse object=new GroundCourse();

            LinkedList listOfObject= new
LinkedList(getTheDataForComboBoxes(GroundCourse.class));
            InstructorsForm sf=new InstructorsForm( pm, listOfObject,  object );
            getContentPane().remove(formPanel);
            formPanel.removeAll();
            formPanel.add(sf);
            getContentPane().add(formPanel);
            //getContentPane().add(sf, BorderLayout.CENTER );
            pack();
            repaint();
        }
```

```
    } );

    button.setToolTipText("Ground course");
    toolbar.addSeparator();

    button = toolbar.add(
    new AbstractAction("Reports form") {
       public void actionPerformed(ActionEvent e) {

          ReportsForm rf=new ReportsForm();

          rf.pack();
          rf.setVisible(true);
          repaint();
       }
    } );

    toolbar.setLayout(new GridLayout(0, 1));


    return toolbar;
}



/**Query the flights for a specific instructor
 * @param inst the instructor
 * @return the result of the query
 */
public Collection queryTheFlights(Instructors inst){
    Extent extent=pm.getExtent(Flights.class,true);
    String filter="instructor==inst";
    Query query=pm.newQuery(extent,filter);
    query.declareParameters("Instructors inst");
    Collection result=(Collection) query.execute(inst);

    return result;

}


/**get the PersistenceManagerFactory
 * @param path the url path
 *     @return the PersistenceManagerFactory
 */
```

```
   public static PersistenceManagerFactory getPMF(String path) {
      try {
         InputStream propStream =
         new FileInputStream(path);
         Properties props = new Properties();
         props.load(propStream);

         props.put("javax.jdo.PersistenceManagerFactoryClass",
         "com.libelis.lido.PersistenceManagerFactory");

         return JDOHelper.getPersistenceManagerFactory(props);
      }
      catch (IOException e) {
         e.printStackTrace();
         System.exit(-1);
         return null;
      }

   }

   /**MouseClicked
    */
   public void mouseClicked(MouseEvent e) {

      InstructorsForm iform=(InstructorsForm) e.getComponent().getParent();
      Instructors inst=(Instructors) iform.getCurrentRecord();

      LinkedList listOfObject= new LinkedList( QueryForm.queryTheFlights(pm, inst) );

      InstructorsForm sf=new InstructorsForm( pm, listOfObject,  new Flights() );
      JPanel l=new JPanel();
      l.add(sf);

      LinkedList listMonth= new LinkedList( QueryGetFlightsMonthlyReport(11, 2003));

      JList j2=new JList(listMonth.toArray());

      Squadron squad=(Squadron) squadronComboBox.getSelectedItem();
      JList     j3=new     JList(QueryGetFlightsMonthlyReportPerSquadron(11,     2003,
squad).toArray());

      JList jl=new JList(listOfObject.toArray());


      l.add(jl);
      l.add(j2);
```

```
        l.add(j3);

        JScrollPane jsp=new JScrollPane(l);

        JFrame myframe =new JFrame();
        Container cont=myframe.getContentPane();
        cont.add(jsp);
        myframe.pack();
        myframe.setVisible(true);



    }


    public void mousePressed(MouseEvent e) {

    }

    public void mouseReleased(MouseEvent e) {

    }
    public void mouseEntered(MouseEvent e) {

    }
    public void mouseExited(MouseEvent e) {

    }

    /** tests the query
     */
    private void testTheQuery(ActionEvent e){
        JButton jb=(JButton) e.getSource();
        //i have put the button to the Categories form
        InstructorsForm iform=(InstructorsForm) jb.getParent();
        Categories cat=(Categories) iform.getCurrentRecord();

        LinkedList list= new LinkedList( QueryGetFlightsExcerciseReportPerCategory(
cat));

        JList j2=new JList(list.toArray());
        JPanel l=new JPanel();
        l.add(new JButton());
        l.add(j2);
        JScrollPane jsp=new JScrollPane(l);
```

```
    JFrame myframe =new JFrame();
    Container cont=myframe.getContentPane();
    cont.add(jsp);
    myframe.pack();
    myframe.setVisible(true);

}

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//////////////////////QUERIES///////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

/**Query the flights for a specific date
 * @param d the date
 * @return the result of the query
 */
public Collection QueryGetFlightsDailyReport(Date d){


    Extent extent=pm.getExtent(Flights.class,true);
    String filter="date==d";
    Query query=pm.newQuery(extent,filter);
    query.declareImports("import java.util.Date");
    query.declareParameters("Date d");
    Collection result=(Collection) query.execute(d);

    return result;


}

/**Query the flights for a specific date
 * and a specific squadron
 * @param squad the squadron
 * @param d the date
 * @return the result of the query
 */
public Collection QueryGetFlightsDailyReportPerSquadron(Date d, Squadron squad){


    Extent extent=pm.getExtent(Flights.class,true);
    String filter="date==d && squadron==squad";
    Query query=pm.newQuery(extent,filter);
    query.declareImports("import java.util.Date");
```

```
    query.declareParameters("Date d, Squadron squad");
    Collection result=(Collection) query.execute(d,squad);

    return result;


}


//the field might be private however here the JDOQL seem to have access
// directly to them using the dot(.) operator i.e. flights.exercise
/**Query the flights for a specific category of
 * exercises
 * @param cat the category
 * @return the result of the query
 */
public Collection QueryGetFlightsExcerciseReportPerCategory(Categories cat){


    Extent extent=pm.getExtent(Flights.class,true);
    String filter="exercise.categories==cat";
    Query query=pm.newQuery(extent,filter);

    query.declareParameters("Categories cat");
    Collection result=(Collection) query.execute(cat);

    return result;


}


/**Query the flights for a specific month
 * @param month the month
 * @param year the year
 * @return the result of the query
 */
public Collection QueryGetFlightsMonthlyReport(int month, int year){

    Calendar c1= new GregorianCalendar(year,month,1) ;
    if(month==11){
        year+=1;
        month=0;
    }
    else
        month+=1;
```

```
        Calendar c2= new GregorianCalendar(year,month,1) ;

        Date beginDate=c1.getTime();
        Date endDate=c2.getTime();

        Extent extent=pm.getExtent(Flights.class,true);
        String filter="date>=beginDate && date<endDate";
        Query query=pm.newQuery(extent,filter);
        query.declareImports("import java.util.Date");
        query.declareParameters("Date beginDate, Date  endDate");
        Collection result=(Collection) query.execute(beginDate,endDate);

        return result;


    }


    /**Query the flights for a specific month
     * and a specific squadron
     * @param month the month
     * @param year the year
     * @param squad the squadron
     * @return the result of the query
     */
    public Collection QueryGetFlightsMonthlyReportPerSquadron(int month, int year,
Squadron squad){

        Calendar c1= new GregorianCalendar(year,month,1) ;
        if(month==11){
           year+=1;
           month=0;
        }
        else
           month+=1;
        Calendar c2= new GregorianCalendar(year,month,1) ;

        Date beginDate=c1.getTime();
        Date endDate=c2.getTime();
        //JOptionPane.showMessageDialog(null,beginDate+"----- "+endDate);

        Extent extent=pm.getExtent(Flights.class,true);
        String filter="date>=beginDate && date<endDate && squadron==squad";
        Query query=pm.newQuery(extent,filter);
        query.declareImports("import java.util.Date");
        query.declareParameters("Date beginDate, Date  endDate, Squadron squad");
```

```
        Collection result=(Collection) query.execute(beginDate,endDate, squad);

        return result;


    }


    /**Query the flights for a specific month and a specific
     * student in a specific squadron
     * @param month the month
     * @param year the year
     * @param stude the student
     * @param squad the squadron
     * @return the result of the query
     */
    public Collection QueryGetFlightsMonthlyReportPerSquadronPerStudent(
    int month, int year, Squadron squad, Students stude){

        Calendar c1= new GregorianCalendar(year,month,1) ;
        if(month==11){
            year+=1;
            month=0;
        }
        else
            month+=1;
        Calendar c2= new GregorianCalendar(year,month,1) ;

        Date beginDate=c1.getTime();
        Date endDate=c2.getTime();
        //JOptionPane.showMessageDialog(null,beginDate+"----- "+endDate);

        Extent extent=pm.getExtent(Flights.class,true);
        String filter="date>=beginDate && date<endDate && squadron==squad &&
student=stude";
        Query query=pm.newQuery(extent,filter);
        query.declareImports("import java.util.Date");
        query.declareParameters("Date beginDate, Date  endDate, Squadron squad, Students
stude");

        Object[] parameters={beginDate,endDate, squad, stude};
        Collection result=(Collection) query.executeWithArray(parameters);

        return result;
```

```
    }


    /**Query the flights for a specific month and a specific
     * instructor in a specific squadron
     * @param month the month
     * @param year the year
     * @param instr the instructor
     * @param squad the squadron
     * @return the result of the query
     */
    public Collection QueryGetFlightsMonthlyReportPerSquadronPerInstructor(
    int month, int year, Squadron squad, Instructors instr){

        Calendar c1= new GregorianCalendar(year,month,1) ;
        if(month==11){
            year+=1;
            month=0;
        }
        else
            month+=1;
        Calendar c2= new GregorianCalendar(year,month,1) ;

        Date beginDate=c1.getTime();
        Date endDate=c2.getTime();

        Extent extent=pm.getExtent(Flights.class,true);
        String filter="date>=beginDate && date<endDate && squadron==squad &&
instructor=instr";
        Query query=pm.newQuery(extent,filter);
        query.declareImports("import java.util.Date");
        query.declareParameters("Date beginDate, Date endDate, Squadron squad,
Instructors instr");

        Object[] parameters={beginDate,endDate, squad,instr};
        Collection result=(Collection) query.executeWithArray(parameters);

        return result;


    }

}
```

## C. METADATA JDO FILE

```
<?xml version="1.0"?>
<!DOCTYPE jdo SYSTEM "jdo.dtd">
<jdo>
   <package name="company">


        <class name="Instructors" identity-type="application">
          <field name="id" primary-key="true"/>
          <field name="lname" >
              <extension vendor-name="libelis" key="sql-index" value="unique"/>
          </field>

          <field name="students" >
                  <collection element-type="Students">

                  </collection>
                </field>
          <field name="aircrafts" >
                  <collection element-type="Aircraft"/>
                </field>

        </class>


        <class name="Students" identity-type="application">
          <field name="id" primary-key="true"/>

          <field name="instructors" >
                  <collection element-type="Instructors">

                  </collection>

                </field>
        </class>

        <class name="Series" identity-type="application">
          <field name="id" primary-key="true"/>
          <field name="stadio" >
                  <collection element-type="Stadio">
                      <extension vendor-name="libelis"
                       key="sql-reverse"
                       value="javaField:series"/>
                  </collection>
```

```
        </field>
    <field name="groundcourse" >
            <collection element-type="GroundCourse"/>
        </field>

    <field name="students" >
            <collection element-type="Students"/>
        </field>

</class>


<class name="Stadio" identity-type="application">
    <field name="seriesId" primary-key="true"/>
    <field name="id" primary-key="true"/>

    <field name="categories" >
            <collection element-type="Categories">

            </collection>

        </field>

</class>

<class name="BaseStadio" identity-type="application">
    <field name="id" primary-key="true"/>
</class>

<class name="Categories" identity-type="application">
    <field name="seriesid" primary-key="true"/>
    <field name="stadioid" primary-key="true"/>
    <field name="id" primary-key="true"/>
    <field name="exercise" >
            <collection element-type="Exercise"/>

        </field>
</class>

<class name="BaseCategory" identity-type="application">
    <field name="id" primary-key="true"/>
</class>

<class name="Exercise" identity-type="application">
    <field name="seriesid" primary-key="true"/>
    <field name="stadioid" primary-key="true"/>
```

```
    <field name="categoriesid" primary-key="true"/>
    <field name="id" primary-key="true"/>
  </class>


  <class name="Aircraft" identity-type="application">
    <field name="num" primary-key="true"/>
  </class>

  <class name="AircraftType" />

  <class name="Rank" />

  <class name="Squadron" identity-type="application">
    <field name="squadron" primary-key="true"/>
    <field name="schedule" >
        <collection element-type="Schedule"/>
          </field>

  </class>

  <class name="Flights">
          <field name="date">
        <!-- Specify the format of the date -->
        <extension vendor-name="libelis" key="sql-mapping" value="date, date-
only"/>
          </field>
        </class>

  <class name="SpecialType" identity-type="application">
    <field name="id" primary-key="true"/>
  </class>

  <class name="Schedule" identity-type="application">
    <field name="squadronid" primary-key="true"/>
    <field name="year" primary-key="true"/>
    <field name="month" primary-key="true"/>
  </class>

  <class name="GroundCourse" identity-type="application">
    <field name="id" primary-key="true"/>
  </class>


  </package>
</jdo>
```

# LIST OF REFERENCES

Jordan, David and Russell, C raig, “Java Data Objects.”

LIDO User’s Manual.

http://java.sun.com/products/jdo/overview.html, accessed April 2004.

THIS PAGE INTENTIONALLY LEFT BLANK

# INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Thomas Otani
   Academic Associate
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California

4. Professor Arijit Das
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California

5. Hellenic Air Force General Staff
   Athens, Greece

6. Hellenic Air Force Academy
   Athens, Greece

7. Christos Zilidis
   Neo Monastiri Domokos
   TK 35010,
   Greece

8. MAJ Paschalis Zilidis
   Neo Monastiri Domokos
   TK 35010,
   Greece