Fall 1998

Iceberg database system for the graduate advisors of Computer and Information Science Department of New Jersey Institute of Technology

Tao Lin
New Jersey Institute of Technology

ABSTRACT

ICEBERG DATABASE SYSTEM FOR THE GRADUATE ADVISORS OF COMPUTER AND INFORMATION SCIENCE DEPARTMENT OF NEW JERSEY INSTITUTE OF TECHNOLOGY

by
Tao Lin

Iceberg system is a departmental database system. It is built for the graduate advisors of the CIS department. It stores the graduate student's information, such as background, bridge requirement and transcript. The graduate advisors can process the student's records using the graphic user interface of Iceberg system.

Iceberg system is an example of the powerful Java language. We use the latest Java technologies to build a flexible system, which is easily extended. The system consists of Iceberg client, Iceberg server and Oracle data source. The Iceberg client is a web-based applet, which can be easily accessed using a browser. The Iceberg server runs on a fast UNIX machine, providing service to the Iceberg client through RMI.

The most interesting feature of Iceberg system is the component architecture of the Iceberg client. The Iceberg client is consisted of visual components that have no knowledge of each other at compile time. They are assembled together at run time, following the instruction of a script file. Since the container component can hold any components, the Iceberg system is readily extendable.

ICEBERG DATABASE SYSTEM FOR THE GRADUATE ADVISORS OF COMPUTER AND INFORMATION SCIENCE DEPARTMENT OF NEW JERSEY INSTITUTE OF TECHNOLOGY

by
Tao Lin

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

January 1999

APPROVAL PAGE

ICEBERG DATABASE SYSTEM FOR THE GRADUATE ADVISORS OF COMPUTER AND INFORMATION SCIENCE DEPARTMENT OF NEW JERSEY INSTITUTE OF TECHNOLOGY

Tao Lin

Dr. James McHugh, Thesis Advisor Acting Chairperson Professor of Computer and Information Science, NJIT	Date
Dr. Franz Kurfess, Committee Member Assistant Professor of Computer and Information Science, NJIT	Date
Mr. Leon Jololian, Committee Member Director of Computing, NJIT	Date

BIOGRAPHICAL SKETCH

Author:	Tao Lin
Degree:	Master of Science in Computer Science
Date:	January 1999

Undergraduate and Graduate Education:

- Master of Science in Computer Science
  New Jersey Institute of Technology, Newark, NJ, 1999
- Bachelor of Chemistry
  Peking University, Beijing, China, 1996

Major:	Computer Science

To my parents
for their long time expectation

ACKNOWLEDGMENT

I particularly want to thank Dr. McHugh. As my thesis advisor, he not only initiates the project, but also contribute a lot of time and valuable opinions to it. He helped me to get the necessary resources and stay with me to test the program.

I would like to thank Dr. Kurfess for his support and kindness. His insight let me avoid many errors in the early design phase.

I would like to thank Mr. Jololian for his support. He provides me the resources to make this project possible.

I also thank Dr. Shih for testing the program and offering good advice.

Thanks for all graduate advisors of CIS department for this thesis would have been impossible without their help and encouragement.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

In this thesis, we will explain our design and implementation of a departmental database system for graduate advisors of CIS department. The database system is named 'Iceberg', so we will call it Iceberg system in the whole paper. Iceberg system is coded in Java 1.1, and takes fully advantage of Java's superior network and database manipulation capability. Iceberg system uses an Oracle 8.0.3 enterprise server as the storage device for the student records. The idea of Iceberg system comes from the graduate advisors of the CIS departme nt, especially Dr. McHugh. The CIS department is one of the biggest departments in NJIT. It has more that 500 hundred Master students and 80 doctoral students . During the daily operation, the graduate advisors of CIS department strongly feel the need of sharing the student's records among themselves. They also want other advisors to know their decision for students. Sometimes, students got di fferent instructions from different advisors. They are free to choose the most favorable instruction to follow. To end this lack of communication among adviso rs, a centralized database system is needed to make one decision for a student immediately observable by other advisors. Maintaining student's records also be come easy using a centralized database. Iceberg system is the name of the proje ct to build such a database.

1.1 Objective

During the design of Iceberg system, we have the following goals in mind:

1.1.1 Distributed Access

The graduate advisors can run Iceberg system from any machine. To accomplish this, we decided to make Iceberg system web based applet. To run the Iceberg system, the graduate advisors need only type the URL of the Iceberg system in a browser, such as Netscape Navigator or Internet Explorer.

1.1.2 Concurrently Process

The graduate advisors may log into the Iceberg System at the same time, processing the student's records. Iceberg system must make sure that no two users are processing the same student's record, so they can not corrupt the student's record by writing to it simultaneously.

1.1.3 Crash Recovery

If one client crash in the middle of a session, it must not affect the server and other graduate advisors who are logging in. Iceberg system must detect the failure and take action on it.

1.1.4 Easy to Use

The Iceberg system shall provide the graduate advisors an easy-to-use graphic user interface. The interface displays the student's information in tables, check boxes and lists. The graduate advisors do not need to learn SQL (Structured Query Language) to process the data in the Oracle database. We are inspired by the web browser and provide a color navigating capability. For example, If multiply student's names are shown as a list, the names of the visited student change the color to pink, while the names that have not been visited remain blue. This feature is the same that a visited link in a web page will change color.

1.1.5 Future Extendibility

New functionality shall be able to be incorporated into the Iceberg system easily. Later when needed, new service for CIS students and faculties may be added into the Iceberg system. That means we must take a modular approach, separate the functionality into different modules. New modules shall use the currently available service easily. We have taken this design a step further to embrace the Software Component infrastructure. By coding each module as a self-contained component, complicated system can be build faster and less frustrating than put everything into one huge executable (a monolithic approach).

1.1.6 User Customizable

Different users shall be able to tailor to Iceberg system to their own need. Not only changing some parameters like the window size, table header width, the graduate advisor shall also be able to choose the functionality they want to use. For example, one graduate advisor may be named as the super user of the Iceberg system. His Iceberg user interface may contain some menu items, tools and panels not seen in most other advisor's window. This change shall not require different executable files to be compiled. Moreover new functionality may be added into the current interface without modifying current code. This goal also requires us to use component architecture to build the system.

1.2 Background Information

Iceberg system is a showcase of the powerful Java language. We use the new JFC (Java Foundation Class) package to build the user interface. We also extensively use Java RMI (Remote Method Invocation), JDBC and multitasking. We also use features like Java Plug-in, Java digital signature.

1.2.1 Java and JDK

Java is an object oriented programming language invented by a group of Engineers from Sun Microsystems, Inc. From its birthday, Java has been aimed at "Write once, run everywhere". The source code is compiled into byte code, a machine independent format. The byte code can be download across the network to the user machine. A Java Virtual Machine then executes the code. If available, the virtual machine can also run a Just-in-time compiler through the byte codes, translate them into the machines specific machine code. The machine code may run at full speed like ordinary c program.

JDK (Java Development Kit) is the Java development environment distributed by Sun Microsystems, Inc. JDK includes a set of tools for create and run Java programs. It also include codes published by Sun which called Java core API (application programming interface). Those codes provide standard access routines to the service of Java Virtual Machine as well as a lot of reusable components that can be used to build our own programs. The current JDK version is 1.1.6. JDK 1.2beta4 is available freely from Sun's Web site. They can be download from Sun's web site freely. Iceberg system is built using JDK 1.1 class package.

1.2.2 JFC and Swing

JFC (Java Foundation Class) is the new GUI (graphic user interface) package aimed at replace the AWT (Abstract Window Toolkit) package in JDK. JFC includes Swing package [?], Accessibility package and Java 2D package. Among them, we use Swing package extensively to build a powerful GUI for Iceberg. Like AWT, Swing is able to run on different platform, but the underlying architecture and implementation are totally different. AWT components, as the letter 'A' (Abstract) implies, do not provide implementation by themselves. They simply create peer components of the underlying operating system and show them on the screen. Therefore on a machine running Windows 95, a AWT button looks same as a Windows 95 button, while on a machine running X Windows, a AWT button is the same as a X Windows button. Swing components actually draw themselves on the screen. They borrow a portion of the screen real estate and draw themselves from scratch. From the viewpoint of the underlying operating system, a Swing button is nothing more than a small picture. Swing package is much more powerful that the AWT package. There is no limit on Swing. You can create strange looking buttons or lists as you like. On the contrary, the AWT components are limited to the components provided by existing operating system. Worse yet, AWT has to satisfy with the components available on all platforms. That means you can not find an AWT component functions like the toolbar of the Windows 95, because other operation systems do not support toolbar. Iceberg System uses Swing package to create a powerful user interface. There is a list component that the currently selected student's names are displayed in it. Each entry has a small icon on the left. When not selected, the icon shows a quiet face. When this entry has been selected, the icon shows a smile face. This feature can remind the graduate advisors which student's record they have not processed.

1.2.3 Java Bean Architecture and InfoBus

Java Bean [?] Architecture is the Java's software component infrastructure. It specifies how a piece of software may export its service to the outsider, such as a compound document framework. To build a Java bean program, programmers may buy existing Java bean components from the component venders, assemble them together and fill in the necessary flow control code. InfoBus is a standard extension of the Java bean architecture. It provides standard protocols to facilitate data exchange between components. The concept of the InfoBus is like hardware buses on PCs. CPU, memory and hard disks are plugged into the bus. They listen to the activity of the bus, receive data from the bus and send data to the bus. In a similar way, components can be plugged into the InfoBus. Whether they have data or need data, they all talk to the InfoBus. Each component is oblivious of other components on the InfoBus.

1.2.4 Infobus

InfoBus [?] is jointly developed by Sun Microsystems and Lotus Development Corporation, aiming at providing an inter-component communication mechanism for the Java bean architecture. Strictly speaking, it is not an object bus since it is not capable to pass object reference around. However it is a very powerful tool for pure data exchange.

1.2.5 RMI and Java's Superior Network Capability

RMI (Remote Method Invocation) is the Java version of RPC (Remote Procedure Call). By using RMI, objects on one machine are able to invoke methods of objects resided on other remote machines. Considering the difficulty that the address space are discontinued for the two machine, RMI successfully disguises the dirty works of establishing the connection, packing up the parameters, calling the remote method, handing back the result if any and finally tearing down the connection. It is greatly simplify the Iceberg system by using RMI as the methods to connect the client and server. RMI is more powerful than RPC in that RMI can pass not only primary data types (like RPC) but also complex objects across the network. This feature is valuable in an object world because objects can now walking around the network on their wish. No other language has such powerful network abilities.

1.2.6 JDBC and Java's Database Access Capability

JDBC (Java DataBase Connectivity) is Java's standard API for manipulating Relational Databases. Each database venders who want to support JDBC should implement the API. This piece of code is called JDBC driver. Java programs talk to any JDBC drivers using the JDBC API. From their viewpoint, all database are the same data sources, no matter it is an Oracle or Sybase database. In a network with heterogeneous database coexist, JDBC program are able to talk to any of them without any consideration in the coding phase.

1.2.7 Java applet Security Model and Digital Signature

Java Virtual Machine has an elaborated security model to handle applet. An applet is a piece of code that is able to run in a browser. Applets are stored on the web server. When meeting the embedded applet tags in html pages, a web browser downloads the code from web server, passes them to a Java Virtual Machine to run. Since it do not know who write the applet and what the applet do, the Virtual machine scrutinize the applet, prohibit them from performing certain action, such as open a file on the local machine or connect to another machine. In Java's term, the applets run in a sandbox. While being cautious won't hurt, sometimes we need more freedom to have the work done. Java provides tools to digitally sign an applet. By verifying the signature, the Virtual Machine will let the signed applet do whatever it can do. Java's digital signature framework is based on the public-private key pair algorithm. The public key and private key are generated pair wise. Programmers use the private key to sign the applet. The users hold certificates, which contain public keys. Before running a signed applet, the Virtual Machine verifies the private key with the public key. If the two keys match, then it knows who create the applet and trust the applet will behave well.

1.2.8 Java Plug-in

Java plug-in [?] is the solution of browser incompatible problem from Sun Microsystems, the Java's birthplace. Sun has been advocating the motto 'Write once, run everywhere' from the first Java day. Unfortunately, that miracle has not happened plainly. Every major browsers or even different versions of the same browser have Java Virtual Machines with some distinct characters. Those difference cause an applet runs well in a Netscape Navigator crashes in the Internet Explorer, or the other way around. The annoyed applet programmers finally agree the motto from Sun with minor change 'Write once, debug everywhere'. Java Plug-in can provide a consistent Java runtime environment across all browsers. It come with the Java Virtual Machine from Sun and the standard core Java API also available in the JDK. A modified HTML page can specify to invoke the Virtual Machine of the Java Plug-in, not the default Virtual Machine of the browser if the browser has installed Java Plug-in. Another advantage of Using Java Plug-in is that its Virtual Machine is always newer than those browsers' . Therefore it is possible to deploy the latest Java technologies just when they come out and hot. The front end of the Iceberg system is web based Java applets. It uses the latest Java technologies like JFC, InfoBus. Currently no browsers are capable of handling it. Therefore Iceberg system is designed to work with Java Plug-in.

CHAPTER 2

SYSTEM ANALYSIS

Iceberg system is consisting of three tires. The first tire is the Java applet that interacts with the graduate advisors. It plays the role of the client in the client/server model. We will call it Iceberg client. The second tire is the Java application server that interacts both with Iceberg client and the Oracle database. We will call it Iceberg server. The third tire is an Oracle database instance acting as the data source. Basically it is a passive part of the database. We will refer it as the Oracle data source.

The Iceberg system class hierarchy is composed of four Java packages. Basically, iceberg.advisor package contains the classes of the Iceberg client and iceberg.server package contains the classes of the Iceberg server. In the iceberg.util package are the data structures used by Iceberg client and Iceberg server. They will be passed across the network between the Iceberg client and Iceberg Server and are implemented as class with only public fields and no methods. The last package, iceberg.guiUtility is only used by the Iceberg client. Later the Iceberg system may have other client packages like iceberg.student and iceberg.faculty. Those client packages can share the iceberg.guiUtility package. This is the reason to separate it from the iceberg.advisor package. In this thesis, we will use the full package name when we refer to a class or interface.

2.1 Implementation of the Iceberg Client

The Iceberg client is a Web based Java applet. It provides an easy-to-use graphic user interface for the user to view the student's records and make decisions for students. The applet can be run from any browsers which have the Java Plug-in installed

The Java Plug-in is required because Iceberg client applet utilize some latest Java technologies which are not supported by existing browsers.

The Iceberg client consists of Java bean components and Java bean container frameworks. It takes fully advantage of the Compound Document Architecture, the most popular Software Component Infrastructure. Each Java bean component, such as advisor.BackgroundInfoPane[1] can be used to view part of the student's records. They are compiled separately with no knowledge of the existence of other components. The Java bean components can be placed in the Java bean container frameworks. The container framework give out a portion of its own screen space to each component, instruct the component to draw itself. After the components have shown, they will process their own mouse and keyboard event and exchange information with other components. The container framework also displays menu items of the components in its menu bar.

2.1.1 Class Initiating Sequence

Iceberg client takes a hierarchical approach to load its components, so it may achieve maximum flexible and is very user customizable. It first initiate the container frameworks, the frameworks in turn Using a property editor, users can instruct Iceberg client to run any combination of container frameworks and any components. That feature shall satisfy the users with diversified need. For example, some graduate advisors are super users of the Iceberg system. Their user interface will include components not used by most other advisors. Iceberg client needs not to be recompiled to suit each advisor's requirement.

The first class loaded in the browser is the iceberg.Iceberg applet. When initiate itself, it connect to Iceberg server, get a reference of the server.SessionServerX[2] interface. This applet then asks user for user ID and password and try to establish

[1]The full package name of advisor package is iceberg.advisor.
[2]The full package name of server package is iceberg.server.

a session with the server.Sessionserver on the Iceberg server. If the advisor logs in successfully, a session number will return to the iceberg.Iceberg applet from Session-Server. Session number is a random generated long value. The server.SessionServer guarantees this number to be unique so following requests to the Iceberg server will use this session number as the identification.

The iceberg.Iceberg then begins to load the user interface. First it requests the server.SessionServer for the properties table of this advisor. The properties are stored in a file. Each line has the following format: NAME=VALUE. The lines begin with '#' are commands. Following is part of the currently used property file[3]:

```
OUTER_FRAMEWORK=iceberg.advisor.ControlWindow
INNER_FRAMEWORK=iceberg.advisor.DataPane

PERSON_LIST_WIDTH=150
STUDENT_TAB_PANE_HEADER=Bridge Requirement|Student Info

Bridge Requirement_HEADER=BridgeRequirementPane|BackgroundInfoPane|LogInfoPane
Bridge Requirement_MENU_LABEL=BridgeTabPane

BackgroundInfoPane_ACTION=iceberg.advisor.BackgroundInfoPane
BackgroundInfoPane_HEIGHT=4
BackgroundInfoPane_WEIGHT=3
BACKGROUND_INFO_TITLE=Education Background
BACKGROUND_TABLE_HEADER=Degree|Major|College|GDate|G.P.A.
BACKGROUND_TABLE_HEIGHT=6
Degree_LENGTH=44
Major_LENGTH=61
College_LENGTH=101
GDate_LENGTH=67
G.P.A._LENGTH=20

BridgeRequirementPane_ACTION=iceberg.advisor.BridgeRequirementPane
BridgeRequirementPane_HEIGHT=3
BridgeRequirementPane_WEIGHT=1

LogInfoPane_ACTION=iceberg.advisor.LogInfoPane
LogInfoPane_HEIGHT=12
```

[3]See APPENDIX B for the complete property file

```
LogInfoPane_WEIGHT=10

Student Info_HEADER=PersonalInfoPane|TranscriptInfoPane

PersonalInfoPane_ACTION=iceberg.advisor.PersonalInfoPane
PersonalInfoPane_HEIGHT=6
PersonalInfoPane_WEIGHT=2
PERSONAL_INFO_TYPE=id|name|bDate|gender|address|homePhone|workPhone|eAddress
id_LABEL=SSN\#/SID\#:
name_LABEL=Student Name:
bDate_LABEL=Birthday:
gender_LABEL=Gender:
address_LABEL=Home Address:
homePhone_LABEL=  Home Phone:
workPhone_LABEL=  Work Phone:
eAddress_LABEL= Email Address:

TranscriptInfoPane_ACTION=iceberg.advisor.TranscriptInfoPane
TranscriptInfoPane_HEIGHT=10
TranscriptInfoPane_WEIGHT=10
TRANSCRIPT_TABLE_HEADER=CNO|Cname|Semester|Grade
CNO_LENGTH=34
Cname_LENGTH=34
Semester_LENGTH=34
Grade_LENGTH=9
```

The iceberg.Iceberg looks into the 'OUTER_FRAMEWORK' property to find the class to load. According to the example property file, it is the advisor.ControlWindow.

The advisor.ControlWindow is a Java bean framework. Other Java bean components can be added to it. It loads the class specified in the 'INNER_FRAMEWORK' property. It is the advisor.DataPane.

The advisor.DataPane is another Java bean framework. It loads the classes specified in the 'STUDENT_TAB_PANE'.

By looking up the values in the property table, Each Java bean container knows which components to load. This information is only available at run time. We can change the look and functionality of an Iceberg client but just editing the property file. No recompile is needed.

2.1.2 Data Communication between Components

Iceberg components communicate with each other via InfoBus mechanism. Since the Iceberg client is assembled in runtime, components can not call methods of another component directly.

The javax.infobus package defines a small set of standard interfaces, and some standard service. Components implement the InfoBusMember[4] interface are able to join the InfoBus. They may optionally implement the InfoBusDataConsumer interface, InfoBusDataProducer interface or both. A component implements the InfoBusDataProducer interface is able to tell InfoBus it has data available. On the other hand, if a component implements the InfoBusDataConsumer interface, InfoBus is able to inform it when new data is available. InfoBus does not interested in the data flowing through it. All those data are objects that implement the DataItem interface. That object may optionally implement the DataItemChangeManager interface. Thus a InfoBusMember may implement the DataItemChangeListener interface so that it can register itself to the DataItem.

There are three basic scenarios for the InfoBus data flow. In all three scenarios, the InfoBusDataProducer and InfoBusDataConsumer do not interact with each other directly. They exchange data through the InfoBus or the DataItems.

- Scenario one, an InfoBusDataProducer informs the InfoBus that it has a new DataItem object available. The InfoBus notifies all the registered InfoBusDataConsumers that a DataItem is now available. One or more InfoBusDataConsumers may then ask the InfoBus for this DataItem. The InfoBus goes back to the InfoBusDataProducer to fetch the DataItem, pass them to the requested InfoBusDataConsumers.

[4]The class name specified in this section are all come from the javax.infobus package if a package name is not prefixed.

- Scenario two, an InfoBusDataConsumer request a DataItem B from the InfoBus. The InfoBus ask all the registered InfoBusDataProducers who has the DataItem B. If one InfoBusDataProducer replies that it has the DataItem, the InfoBus fetches the DataItem and send it to the InfoBusDataConsumer. If none InfoBusDataProducer replies, the InfoBus simply return null back. If more that one InfoBusDataProducers reply, the InfoBus get the DataItem from the first replied InfoBusDataProducer.

- Scenario three, an InfoBusDataConsumer gets a DataItem C. It register itself as DataItemChangeListener to the DataItemChangeManager of the DataItem. When the owner of the DataItem, a InfoBusDataProducer changes the value of the DataItem, all registered DataItemChangeListener are informed the change. The InfoBus is not involved in this scenario.

The components of the Iceberg client use InfoBus as their communication mechanism. Iceberg system has two GUI components, guiUtility.IcebergJFrame[5] and guiUtility.IcebergJPanel and one non-GUI component, util.IcebergInfoBusDataMember

2.1.3 Extendibility of the Iceberg Client

Java bean component architecture and InfoBus inter-component communication mechanism make Iceberg client extremely flexible. Graduate advisors may change many settings of the system and tailor the system to their needs. Moreover, Iceberg client is open and easy to extend to greater functionality. Now components can be added into the Iceberg client and currently used components may be replaced by new component with lease effort. Iceberg system takes a modular design approach. Only iceberg.Iceberg has the knowledge of server.SessionServer and advisor.DataStore handles the request to server.AdvisorServer. Other components only concern about

[5]The full package name of guiUtilitiy package is iceberg.guiutility.

the DataItem they can send out and get in. Therefore, replacing an old component with a new one is very easy.

2.1.4 Digital Signature

Digital Signature is used to sign the applet so the browser may give the signed applet more freedom to fulfil its task. The Virtual Machines in the browsers has a set of sophisticated mechanism to restrict the applets from doing something harmful. For example, applets are not allowed to read and write local files, open network connections to other machines except the home machine where it come from. This is a valuable feature that protects the majority Internet users. However, if we knows who write the code and trust the author, we would allow the applet to run without restriction. This often occurs in an company's Intranet environment where user can always trust the code coming from their own company's web site.

Digital signature provides the mechanism to explicitly establish the trusted relationship between the Intranet users and the applet authors. It is based on the public-private key pair algorithm. Both keys are long byte sequences generated together. The magic in it is that only the public key can decrypt the applet having been encrypted by the corresponding private key. We will show how Java use the digital signature.

JDK 1.1 includes a tool, javakey, to perform the key generation and management tasks. The javakey utility provides a command line interface with quite a few options. In the Java security infrastructure, there are identities, signers and certificates. An identity is associated with a public key and used by the web user's Java virtual machine. An signer is associated with a public-private key pair and stored on the applet author's machine. The javakey manages a database that can store identities and signers. This database is actually a serialized Java Object stored in the identitydb.obj file in the javakey user's home directory. In a Windows 95 environment, the identitydb.obj file is stored in the Windows directory. An certificate holds a public key so it can be passed from the key owner to the users. The signing process begins at the applet author side.

- First, the applet author creates a signer entity. Following command create a trusted signer taolin.

  ```
  javakey -cs taolin true
  ```

- Second, the author generates the public-private key pair. Following command instructs the javakey to generate an 512 bytes long key pair use DSA algorithm, and associate the key pair with the named signer taolin.

  ```
  javakey -gk taolin DSA 512
  ```

- Third, the author generates a certificate. He need create a directive file first. Th instructions are stored in the directive file. A example file, certificate.dir, is as following:

  ```
  issuer.name=taolin

  subject.name=taolin
  subject.real.name=Frank T. Lin
  subject.org.unit=Research Division
  subject.org=Network Research Lab.
  subject.country=USA

  start.date=11 Jun 1998
  end.date=10 Aug 1998
  serial.number=1001

  out.file=taolin.x509
  ```

  Following command generate a certificate stored in the file taolin.x509. Now the applet author can send the certificate to the applet users.

```
javakey -gc certificate.dir
```

- Forth and the last, the author sign his applet with the private key. The applet must be stored in a jar file. Actually, the javakey encrypted the jar file use the private key. Another directive file is needed to instruct javakey. A sample file, signer.dir, is as following:

```
signer=taolin
cert=1
chain=0
signature.file=iceberg
out.file=Iceberg.jar
```

  Assuming our applet is stored in the Iceberg.jar file, following command will sign the jar file and store the signed file in the Iceberg.jar.sig file. We need to rename the Iceberg.jar.sig to Iceberg.jar. The signer.dir is another directive file, its content is as following:

```
javakey -gs signer.dir Iceberg.jar
```

Now the applet is ready to use. However we still need configure the user account. we assume the users trust the applet author and know how to play with the javakey utility.

- First, a user shall create a identity for the applet author. Following command generates a named identity taolin and marks it as trusted. It is very important to mark the identity as trusted. Otherwise although signer taolin has signed the applet, the virtual machine on the user machine still consider the signed applet as entrusted.

```
javakey -c taolin true
```

- Second, the user get a certificate from the applet author. The certificate shall contain the public key of that author. Following command imports the certificate into the security database, associate it with the identity taolin.

```
javakey -ic taolin taolin.x509
```

Now the user has finished his work. Once the signed jar file is downloaded to the user machine, the virtual machine goes to the security database in the users home directory and find the public key of the applet author. It then tries to decrypt the jar file. If the description is successful, the jar file is considered trusted by the virtual machine and runs without any restriction enforced on other applets. As we can see, this scheme involves five steps and is cumbersome to use.

2.2 Implementation of the Iceberg Server

The Iceberg server handles the request from the applets, query the oracle database using JDBC and send the result back. It runs in the background on a UNIX machine in CIS department. . The application server are also act as the session server which controls the user log in and log out process.

Iceberg server is composed of several RMI interfaces and their implements. When iceberg.server.IcebergServer runs, it first read in a property file, find the Oracle database login ID and password, and try to log into the Oracle database. If the log in is successful, it initiates all the servers include iceberg.server.SessionServer and iceberg.server.AdvisorServer and registers these service with the rmiregistry. The rmiregistry is the Naming service provided by the JDK. The servers then sit there passively and wait for requests from the Iceberg clients.

2.2.1 Session Management in Iceberg System

The iceberg.server.SessionServer takes care of the session management, acts as the guard of the whole Iceberg server. It provide public methods through a RMI interface iceberg.server.SessionServerX. Iceberg clients can obtain a reference of this interface and call its methods to create a session.

The iceberg.server.SessionServerX provide following public methods:

```
public Long login( String userID, String passwd )
public void logout( Long sessionID )
```

When a Iceberg client call the login method, passing the userID and password. Iceberg will check the ACCOUNT table in the Oracle data source for a matching account. If it finds one, it gathers this user's information and create a server.Session Object. If it can not find a match, it throws a server.IncorrectLoginException. The server.Session class is defined as:

```
public class Session
{
    public Long    sessionID;
    public String  currentLock;
    public int       timer;
    public Person  user;
}
```

The user's SSN, name and other information are stored in user Session.user field as an util.Person object. The server.SessionServer then generates a random long number as the sessionID and check whether any other active session already use this number. If so, it will generate another number and verify it's unique again. by this way, the server.SessionServer can make sure this sessionID is unique across the system. It then put the server.Session object into its internal tables, return the long value back to the Iceberg client. Iceberg client can use this number to identify itself. The server.SessionServer has a methods to verify a sessionID.

```
Session matchSession( Long sessionID )
```

Any RMI request from a Iceberg client brings the sessionID with it. The server.AdvisorServer provides service for the graduate advisor. Its public methods first call SessionServer.matchSession to verify the sessionID of the client. If the sessionID field of a currently active session matches the sessionID of the caller, this session will be returned. Otherwise null is returned. If the methods of the AdvisorSever gets a null value, it will throw a server.SessionNotExistException and stop. Iceberg client will report error to the graduate advisor. This mechanism prevent the unauthorized access to the server.

The server.Session object has a Session.currentLocks field. This field can hold a java.lang.String object as a student ID. By recording which student the graduate advisor is currently processing, we prevent two graduate advisors from working on the same student's record at any time. The server.SessionServer has a method:

```
synchronized void setLock( Session session, String id  )
throws DataProcessException
```

Any session wanted to read a student record must first call setLock, trying to set a lock on this student's record. If succeeded, nothing happen and the request can go on. However, if server.SessionServer detects a lock already set on this student's record, it will throw an util.DataProcessException. The exception will terminate the current request and Iceberg client will report error to the unlucky graduate advisor.

When the server.Session object is initiated, the Session.timer field is set to zero. This field is used by the server.SessionServer to guard against client crashing. The Session.currentLock field keeps the state of a session, so Iceberg server is a stateful server. The old student's record lock is released when the graduate advisor shifts to a new student's record or he logs out. However if the client process is stopped abruptly as a power failure or the advisor presses Ctrl-C, no clean up routine will be done and the lock in the server side will not be released. Just imagine the student's record lock can sit in the server.SessionServer forever, prevent anyone from touching the record. This situation is too bad to live with. Thus as soon as the server.SessionsServer initiate, it create another thread. This thread sleeps in the background and periodically wake up. When it is up, it checks each session currently active in the server.SessionServer. The thread increase the timer by one. If the value of the timer is bigger than a predefined threshold now, the thread concludes that the Iceberg client creates this session has died. Then it will release the Session.currentLock if there is one and clean up the session object from the internal tables of the server.SessionServer. The current setting is that the thread wakes up every 60000 ms and the session whose value of the timer bigger than 5 will be purged. This setting is set in the IcebergServerInfo.properties file and read into the system when server.SessionServer initiated. Using a text editor can change these settings.

Iceberg client has two ways to tell the server.SessionServer it is alive. First, we have said each RMI call from a Iceberg client to the server.AdvisorServer calls SessionServer.matchSession(). This method also refreshes the timer of the valid session to zero. In the second case, we have to consider that five minutes is the longest time the server.SessionServer can tolerant a idle client. It is possible for a Iceberg client to idle more that 5 minutes because the graduate advisor may works on something else. Therefore the iceberg.Iceberg class, which is the only class knows server.SessionServer, must automatically touch the server from time to time. There is a public method in the server.SessionServerX interface:

```
public void isAlive( Long sessionID )
throws SessionNotExistException, DataProcessException
```

As soon as the iceberg.Iceberg successfully verified the uesrID and password., it will lunch a thread sleeping in the background. The thread wakes up periodically and call SessionServerX.isAlive method, then sleeps again. This methods just call SessionServer.matchSession to refresh the timer. The setting is for the thread to wake up every 60000ms. It is set in the HTML page and can be changed. By this way, the session will stay in the SessionServer as long as the iceber.Iceberg is alive.

2.2.2 The iceberg.server.AdvisorServer Server

The server.AdvisorServer provides service for graduate advisors. It defines methods to query and update student's records from the Oracle data source. Each public method begin with the routine of checking the validity of the sessionID. All methods that updates the student's records also check if the student is the one locked by this session. If in the future, other servers like StudentServer or Faculty-Server are developed, these codes shall be preserved in the front of each public methods. The server.AdvisorServer do not include any code querying or updating the database. Querying means reading data from the database, while updating means changing data in the database. Those functionality is delegated to two utility class, server.QueryFactory and server.UpdateFactory. Utility class typically only contain public static methods which are call user ClassName.methodName. We consider that in the future, new servers may be added into the Iceberg server. It is best for them to share the utility classes.

2.3 Oracle Data Source

Student's records are stored in the tables maintained in the Oracle database. The user name and password of the Iceberg users are also stored in the database. The application server contacts the Oracle database by a set of Oracle thin JDBC drivers freely distributed by Oracle Corporation. APPENDIX A provides the complete SQL DDLs for create these tables.

CHAPTER 3

USER MANU

This chapter provides information on how to use the Iceberg system. Step by Step, we will show how to process student's record and create new records.

3.1 Log Into Iceberg System

The Iceberg client is a web-based applet that can be accessed from any browser. Make sure your browser has installed the Java Plug-in 1.1. The Iceberg client can only run in the Java Plug-in. The URL of Iceberg client is http://www.cis.njit.edu/ taolin/Iceberg.html.

After the HTML page is displayed in the browser, the browser loads the Java Plug-in into the memory. The status bar shows a line "Loading plug-in". This loading may take 30 seconds. Then the virtual machine in the Java Plug-in loads the Iceberg.jar archive and run the iceberg.Iceberg applet. This may take another 30 seconds and there is a line "Loading applet..." shown in the browser.

The iceberg applet pops up a small login window (Figure 3.1), asking the user for the userID and password. The graduate advisor keys in their userID and password and press 'Connect button'. If the information is correct, the login window will disappear and main window will appear. Otherwise, the login window stays on the screen and a dialog window appears, telling the graduate advisor that the userID and password are not correct. The graduate advisor can try to type in the correct value again. At any time, the graduate advisor may get out of the Iceberg applet by press the "Exit" button or the dispose box on the right upper corner of the login window.

The main window (Figure 3.2) will appear if the graduate advisor loges in successfully. Due to the size of the program, this may take 15 seconds. The main window has menu bar and tool bar. Left side of the window is a list where student names will be displayed. Right side is a tabbed panel. The first panel displays the student's background, bridge requirement and the logs written by the graduate advisors. The second panel displays the student's personal information and transcript.

Iceberg system has two basic functions. One is create new student records in the database. The other is processing a student's record.

3.2 Process the Student's Records

The main function of Iceberg system is processing the student's records. The main window can display one student's record at one time.

3.2.1 Open Student's Records

To open one or more student's records, the graduate advisor may go to the 'Record' menu and press 'Open', or he can press the 'Open' button on the tool bar. A window pops up. The window has three fields, for the student's ID, name and email address. Each field has different search criterions to choose. Iceberg system has vague search ability. Each field has a 'Find similar' option. The graduate advisor may input as much as he can remember and select the 'Find similar' in the combo box. Iceberg system will try to mach student's records that close to the search string. For example, search for ID '135026823' using 'Find similar' option, the Iceberg server will return student whose ID is '135026822'. This option may return multiple records. In addition, the name field has a 'Sound like' option. If 'Sound like' is selected, Iceberg system tries to find students whose name pronounces like the inputted name.

The search result shows in the list in the left part of the main window. The student's name is displayed as an entry in the list in blue letters. At the left of each entry is an icon with quiet face. If an entry has been selected, the color of its name turns into magenta and the icon changes to a smile face. This mechanism helps the graduate advisor to differentiate the student's records they have processed with those they have not. This design is inspired by the web browser design that the color of hyperlink changes after it has been visited.

When an entry is selected, the student's record is displayed in the tabbed panel. There are two tab panels. The 'Bridge Requirement' panel contains background information, bridge requirement and system log information. The 'Student Information' panel contains student's personal information and transcript.

Iceberg system has a concurrency control mechanism to guard against two advisors modifying the same student's record. When the graduate advisor selects a student's record to view, sometimes a dialog window will appear, showing that another graduate advisor is processing the student's record. There is no way to access this student's record now; this graduate advisor has to try later.

3.2.2 Change the Background Information

The background information is displayed in a table. Each row is an entry of the student's background. The width of each row is changeable and the relative position can be changed also. You can drag a column header to change its place with adjacent columns.

The graduate advisor can edit the content of the background table. Click any place in a row selects this row. The color of the row changes into blue. Double click a field selects the row and a cursor appears in the field rectangle. The graduate advisor can edit this field now. When finished, he can click any other place or press the 'Enter' key. The cursor disappears and the content is fixed.

The graduate advisor can add and delete rows. Go to the menu bar and open the TabA menu, there are two menu items 'Add a background entry' and 'Remove a background entry'. Press the 'Add a background entry', a new row with blank content appears in the bottom of the background table. The graduate advisor may fill in the fields of the new row. To delete a row, the graduate advisor needs to select the row first. Make sure the background color of this row turns into blue, then press the 'Remove a background entry' menu item. That row is deleted.

3.2.3 Assign the Bridge Requirement

The student's bridge requirement is shown in a panel labeled 'Bridge Requirement'. The panel displays all possible bridge courses required by the student's program. The checked checkbox indicates that the student needs to take this course. The graduate advisor can click any checkbox to check or uncheck it.

3.2.4 Write Logs

A graduate advisor can write logs in the 'System Log' panel. The logs are stored in the database so that other advisors can see them. The logs displayed are associated with the current student and the graduate advisor shall write something relevant to the current selected student. The logs are displayed with the newest one on the top and the oldest one at the bottom. The header of each log entry displays the creating time and the creator's name. The log content follows the header. The log content can be modified and the whole text area will expend if more space is needed. The modified log entry will have a new time stamp indicates the last modification time. Be ware that a graduate advisor can only modify the logs he created.

The graduate advisor can add log entries. Click the TabA menu and press 'Add a log entry', a new log entry appears. A line 'New entry" appears in the timestamp position. In the writer position is the current advisor's name. The graduate advisor may type letters in the empty text area. It is impossible to delete a log entry since this contradicts to the purpose of having logs.

The log panel has color navigating ability. The log entry headers are shown in blue color. If an entry is modified, the color of the header changes to magenta. This feature reminds the graduate advisor which log entries he has modified.

3.2.5 Change Personal Information

The student's personal information shows in the 'Personal Information' panel. There are eight fields in the panel. They are 'ID', 'Name', 'Birthday', 'Gender', 'Address', 'Home phone', 'Work phone' and 'Email address'. the graduate advisor may change all fields except 'ID' and 'Name'.

3.2.6 View the Student's Transcript

The student's transcript is showing in the 'Transcript' panel in a table form. Each column width and the relative sequence are changeable. However the content can not be changed.

3.2.7 Save the Change

After the graduate advisor has made the necessary change, he may save the change to the database. Go to the 'Record' menu and press 'Save', or click the 'Save' button on the tool bar, Iceberg system begins to save the changed data. Iceberg system first checks the validity of those data. If any field is wrong, a dialog window will pop up, indicating the incorrect field. The graduate advisor needs to change the incorrect data and save the data again.

3.3 Create a New Record

To create a new student's record, the graduate advisor may click the 'Record' menu and press 'New', or he can press the 'New' button on the tool bar (All buttons have ToolTips). A window pops up. The graduate advisor then input the student's ID, name and an optional email address. The ID, first name and last name field must be filled out. The graduate advisor must choose the student's program too. After filling out necessary information, the graduate advisor press 'Start' buttons. Iceberg system will check all inputted data. If some data is not correct, a dialog window will appear, telling the graduate advisor what is wrong.

If the student's information is correct, a student's record is created in the database. The small window disappears and the new student is the current student selected in the main window. The graduate advisor may change the student's record as same as processing other student's records.

CHAPTER 4

CONCLUSION

4.1 The Work Done

Iceberg system is a useful tool for the graduate advisors. It allows multiple graduate advisors to log into the system, view the student's records and make decisions. The system basically fulfils the goals we have set at the beginning. Iceberg system is a good example of the powerful Java language. It utilizes many new Java technologies such as RMI, JDBC and Java bean architecture. The distinctive character of the Iceberg system is that the system is an extensible framework. We have this goal in mind and take fully advantage of the software component architecture. Iceberg client consists of components that form a hierarchical structure. When it is assembled, the parent components follow the instructions of a resource file to load and initiate its children components. By editing the resource file, We can add new components to the Iceberg client easily without modifying the existing code. Iceberg system is more like an open framework than a program. Such architecture is superior to the traditional software architecture because it extends the reusable software concept to a new height.

Software reuse concept has been proposed for more than thirty years. The first form of reusable software came as function library. They have to be linked with the program to generate the executable. The second form is the class library where data and functions are encapsulated in classes. These class libraries are easier and safer to use because the classes know what data they can handle. the Java core API is an example of carefully designed class libraries. However, classes are too small building blocks to build a program. They are more like pebbles than blocks. Software component architecture addresses this problem. Components are modules that provide services as well as a way to let the outsider to detect their service. They can not only use by inspecting the source code, but also can be used in visual frameworks. This is the key point behind the Microsoft Visual Basic and Borland Delphi. Although a component may contain only one class, it is abstracted from a higher level of usage than a class is.

4.2 Future Work

Iceberg system provides a basis for later work. On the server side, it is a full-fledged session management and concurrency control system. The design of the Oracle relational schema has also considered the future extendibility . Therefore the Iceberg server is ready to serve other functionality. On the client side, it utilizes the software component architecture. New components can be add to the system with lest effort. New modules for students and faculties are considered. The student module may include an iceberg.server.StudentServer and a set of visual components. It will allow the student to see his record and input data his or her data into the database. The faculty module is useful for faculty members to post their available project topics.

Java is a fast evolving technology. The current implementation uses mostly the features of Java 1.1. Later it can be enhanced with the Java 1.2 technology. The JDK 1.2 is now under the beta test and will be available soon. Java 1.2 provides several advantages and addresses several shortcoming of the Java 1.1.

The enhancement may include following features:

- The swing user interface. The swing package is part of the core API in the JDK 1.2 release. It is turned up for faster speed and better performance. Base on the source from Javasoft, it screen rendering speed may be doubled in the new release. The current swing 1.0.3 is good enough for daily operating. However there are still several bugs in the JTable class. The swing 1.1 package comes with JDK 1.2 corrects those bugs, so we will not manually fix those bugs ourselves and would rather wait for the new release.

- The RMI package. Java 1.2 add an activation mechanism to the RMI package. A server object need not to be instantiated when it is registered with the rmiregistry. The rmiregistry can create an instance of the server object whenever there is a request to use its service.

- The Java bean architecture. The Java bean architecture is improved in the Java 1.2 as the shortcoming of the Java bean 1.0 specification has become apparent. The Java bean 1.0 does not provide methods to describe the hiarchy of logical structure of the Java beans. It does not specify how a bean can import services from outside environment or export its service to the outside. For example, in the Iceberg client, any component should response to the certain action command such as 'Save', 'Clear' and 'New'. Therefore if a menu item in the parent container is clicked, all of its children shall be ask to perform the action. However the containers do not know what their children can do and the Java bean 1.0 specification does not specify how a container can obtain that information. The coming Java 1.2 include the Java bean 1.2 specification that addresses those fields. Although we can hand coin our own import and export protocols, we felt it is better to wait the Java bean 1.2 specification to come. By deploying Java bean 1.2 specification, we can create more robust bean components.

- Java collection framework. The Java collection package is a new package in the Java 1.2. it compose of a set of primary datatypes such as list, set and map. Using collection object may simplify the API design. Unlike Vector and Hashtable, the collection objects are unsynchronized. Using collection package may speed up the program

Java technology will provide greater and greater functionality as it evolves rapidly. Iceberg system shall keep evolving with the Java so that it may provide greater functionality to the users.

APPENDIX A

DATABASE SCHEMA

This appendix provides the internal structure of the Oracle data source. The database is created using SQLplus software come with the Oracle 8 database.

Following is the SQL DDL commands that create the database

```
Create table PERSON (
ID              char(9) not null,
LNAME           varchar2(30) not null,
MNAME           varchar2(30),
FNAME           varchar2(30) not null,
BDATE           date,
GENDER          char,
ADDRESS         varchar(150),
HOMEPHONE       char(10),
WORKPHONE       char(10),
EADDRESS        varchar2(40),
primary key (ID) );

create table ACCOUNT (
ID              char(9) not null,
USERID          varchar(30) not null,
PASSWD          varchar(30),
USAGE           char,
primary key (ID, USERID),
foreign key (ID) references PERSON(ID) );

create table PROGRAM (
PNO             number(2) not null,
PNAME           varchar(80) not null,
DEGREE          varchar(6) not null,
primary key (PNO) );

create table TRACK (
TRNO            number(2) not null,
PNO             number(2) not null,
TRNAME          varchar(80) not null,
primary key (TRNO, PNO),
foreign key (PNO) references PROGRAM(PNO) );

create table FACULTY (
FID             char(9) not null,
primary key (FID),
foreign key (FID) references PERSON(ID) );

create table STUDENT (
SID             char(9) not null,
PNO             number(2) not null,
TRNO            number(2),
STATUS          char,
primary key (SID),
foreign key (SID) references PERSON(ID),
foreign key (PNO) references PROGRAM(PNO),
foreign key (TRNO,PNO) references TRACK(TRNO,PNO) );

create table ADVISOR (
ID             char(9) not null,
PNO             number(2) not null,
ISADM           char,
primary key (ID, PNO),
foreign key (ID) references PERSON(ID),
foreign key (PNO) references PROGRAM(PNO) );

create table COURSE (
CNO             varchar2(7) not null,
CNAME           varchar2(80) not null,
CREDIT          number(2,1) not null,
primary key (CNO) );

create table PROGRAM_OFFER_BRIDGE (
PNO             number(2) not null,
CNO             varchar2(7) not null,
primary key (PNO, CNO),
foreign key (PNO) references PROGRAM (PNO),
foreign key (CNO) references COURSE (CNO) );


create table PROGRAM_OFFER_CORE (
PNO             number(2) not null,
CNO             varchar2(7) not null,
primary key (PNO, CNO),
foreign key (PNO) references PROGRAM (PNO),
foreign key (CNO) references COURSE (CNO) );


create table TRACK_OFFER_ELECTIVE (
PNO             number(2) not null,
CNO             varchar2(7) not null,
TRNO            number(2) not null,
primary key (PNO, CNO, TRNO),
foreign key (PNO) references PROGRAM (PNO),
foreign key (TRNO, PNO) references TRACK(TRNO, PNO),
foreign key (CNO) references COURSE (CNO) );


create table PROGRAM_OFFER_REQUIRED (
PNO            number(2) not null,
CNO            varchar2(7) not null,
primary key (PNO, CNO),
foreign key (PNO) references PROGRAM (PNO),
foreign key (CNO) references COURSE (CNO) );


create table EDU_BACKGROUND (
ID             char(9) not null,
COLLEGE        varchar2(80),
LOCATION       varchar2(80),
MAJOR          varchar2(30),
DEGREE         varchar2(30),
GDATE          varchar2(15),
GPA            number(3,2),
foreign key (ID) references PERSON(ID) );


create table STUDENT_TAKEN_COURSE (
SID            char(9) not null,
CNO            varchar(7) not null,
SEMESTER       char(5) not null,
GRADE          number(2,1) not null,
primary key (SID, CNO),
foreign key (SID) references PERSON(ID),
foreign key (CNO) references COURSE(CNO) );

create table STUDENT_TAKING_COURSE (
SID            char(9) not null,
CNO            varchar2(7) not null,
primary key (SID, CNO),
foreign key (SID) references STUDENT(SID),
foreign key (CNO) references COURSE (CNO) );

create table BRIDGE_REQUIRE_STUDENT (
SID            char(9) not null,
CNO            varchar2(7) not null,
primary key (SID, CNO),
foreign key (SID) references STUDENT(SID),
foreign key (CNO) references COURSE(CNO) );
```

```
create table STUDENT_GRADUATE (
SID           char(9) not null,
GDATE         date not null,
ADVISOR       char(9),
GTOPIC        long,
STATUS        char,
primary key (SID),
foreign key (SID) references STUDENT(SID),
foreign key (ADVISOR) references FACULTY(FID) );

create table GRADUATE_REQUIRE_STUDENT (
SID           char(9) not null,
CNO           varchar2(7) not null,
primary key (SID, CNO),
foreign key (SID) references STUDENT(SID),
foreign key (CNO) references COURSE(CNO) );

create table EMESSAGE (
OWNER         char(9) not null,
SUBJECT       varchar(60) not null,
CONTENT       long,
primary key (OWNER, SUBJECT),
foreign key (OWNER) references FACULTY(FID) );

create table SYSLOG (
FROM_ID       char(9) not null,
TIME          date    not null,
TYPE          char,
TO_ID         char(9),
CONTENT       varchar2(2000),
foreign key (FROM_ID) references PERSON(ID),
foreign key (TO_ID) references PERSON(ID) );
```

APPENDIX B

PROPERTY FILES FOR CLIENT AND SERVER

This appendix provide the property files used to configure the Iceberg client and Iceberg server

IcebergInfo.properties is the property file used by the Iceberg client to initialize the user interface. All the Iceberg components get information from this file.

```
#***********************
# IcebergInfo.properties
#***********************

# Iceberg System Parameter
OUTER_FRAMEWORK=iceberg.advisor.ControlWindow
INNER_FRAMEWORK=iceberg.advisor.DataPane

#ImageSuffix=_IMAGE
#LabelSuffix=_LABEL
#ToolTipSuffix=_TOOLTIP
#LengthSuffix=_LENGTH
#HeaderSuffix=_HEADER
#WeightSuffix=_WEIGHT
#HeigthSuffix=_HEIGHT
#ActionSuffix=_ACTION

#*************************
# ControlWindow parameters
#*************************
CONTROL_WINDOW_TITLE=Iceberg System for Advisor
CONTROL_WINDOW_WIDTH=680
CONTROL_WINDOW_HEIGHT=669
CONTROL_WINDOW_XPOSITION=151
CONTROL_WINDOW_YPOSITION=50

## Menubar definition
MENUBAR=Record|Tool

### Record menu definition
Record=New|Open|Save|-|Exit
New_ACTION=iceberg.advisor.RecordGenerator
New_IMAGE=images/new.gif
Open_ACTION=iceberg.advisor.Searcher
Open_IMAGE=images/open.gif
Save_IMAGE=images/save.gif

### Tool menu difinition
Tool=StoredProcedure|Mail
StoredProcedure_ACTION=iceberg.advisor.Browser
StoredProcedure_IMAGE=images/browse.gif
#Mail_ACTION=iceberg.advisor.Mailer
Mail_IMAGE=images/mail.gif
```

```
## Tool Bar definition
TOOLBAR=Open|StoredProcedure|-|Mail
Open_TOOLTIP=Open student's records
StoredProcedure_TOOLTIP=Execute a stored procedure
Mail_TOOLTIP=Send student a mail

# ControlWindow parameter end

#*******************
# DataPane parameter
#*******************
PERSON_LIST_WIDTH=150
STUDENT_TAB_PANE_HEADER=Bridge Requirement|Student Info
#STUDENT_TAB_PANE_HEADER=Bridge Requirement

##******************************
## Bridge Requirement paramenter
##******************************
Bridge Requirement_HEADER=BridgeRequirementPane|BackgroundInfoPane|LogInfoPane
#Bridge Requirement_HEADER=BackgroundInfoPane
#Bridge Requirement_HEADER=BridgeRequirementPane
#Bridge Requirement_HEADER=LogInfoPane
Bridge Requirement_MENU_LABEL=BridgeTabPane

###*****************************
### BackgroundInfoPane parameter
###*****************************
BackgroundInfoPane_ACTION=iceberg.advisor.BackgroundInfoPane
BackgroundInfoPane_HEIGHT=4
BackgroundInfoPane_WEIGHT=3
BACKGROUND_INFO_TITLE=Education Background
BACKGROUND_TABLE_HEADER=Degree|Major|College|GDate|G.P.A.
BACKGROUND_TABLE_HEIGHT=6
Degree_LENGTH=44
Major_LENGTH=61
College_LENGTH=101
GDate_LENGTH=67
G.P.A._LENGTH=20
### BackgroundInfoPane parameter end

###*********************************
### BridgeRequirementPane parameter
###*********************************
BridgeRequirementPane_ACTION=iceberg.advisor.BridgeRequirementPane
BridgeRequirementPane_HEIGHT=3
BridgeRequirementPane_WEIGHT=1
```

```
### BridgeInfoPane parameter end

###**********************
### LogInfoPane parameter
###**********************
LogInfoPane_ACTION=iceberg.advisor.LogInfoPane
LogInfoPane_HEIGHT=12
LogInfoPane_WEIGHT=10
### LogInfoPane parameter end

## Bridge Requirement parameter end


##***************************
## Student Info parameter
##***************************
Student Info_HEADER=PersonalInfoPane|TranscriptInfoPane

###******************
### PersonalInfoPane
###******************
PersonalInfoPane_ACTION=iceberg.advisor.PersonalInfoPane
PersonalInfoPane_HEIGHT=6
PersonalInfoPane_WEIGHT=2
PERSONAL_INFO_TYPE=id|name|bDate|gender|address|homePhone|workPhone|eAddress
id_LABEL=SSN#/SID#:
name_LABEL=Student Name:
bDate_LABEL=Birthday:
gender_LABEL=Gender:
address_LABEL=Home Address:
homePhone_LABEL=  Home Phone:
workPhone_LABEL=  Work Phone:
eAddress_LABEL= Email Address:
### PersonalInfoPane parameter end

###******************
### TranscriptInfoPane
###******************
TranscriptInfoPane_ACTION=iceberg.advisor.TranscriptInfoPane
TranscriptInfoPane_HEIGHT=10
TranscriptInfoPane_WEIGHT=10
TRANSCRIPT_TABLE_HEADER=CNO|Cname|Semester|Grade
CNO_LENGTH=34
Cname_LENGTH=34
Semester_LENGTH=34
Grade_LENGTH=9
### TranscriptInfoPane parameter end
```

```
## Student Info parameter end

##***************************
## Graduate parameter
##***************************
Graduation_HEADER=GraduateInfoPane
GraduateInfoPane_ACTION=iceberg.advisor.GraduateInfoPane
## Graduate parameter end

#******************
# Pin Button parameter
#******************
Pinup_IMAGE=images/pinup.gif
Pindown_IMAGE=images/pindown.gif
Pin_TOOLTIP=Stay on/off Desktop
# Pin Button parameter end

##*******************
## Searcher parameter
##*******************
SEARCHER_TITLE=Search Student
SEARCHER_WIDTH=448
SEARCHER_HEIGHT=346
SEARCHER_XPOSITION=569
SEARCHER_YPOSITION=533
## Searcher parameter end

##******************
## Browser parameter
##******************
BROWSER_TITLE=Browse Students
BROWSER_WIDTH=454
BROWSER_HEIGHT=319
BROWSER_XPOSITION=100
BROWSER_YPOSITION=99
PROGRAM_LIST_WIDTH=300
## Browser parameter end

##******************
## Mailer parameter
##******************
MAILER_TITLE=Send Student E-mail
MAILER_WIDTH=478
MAILER_HEIGHT=477
MAILER_XPOSITION=4
MAILER_YPOSITION=228
```

```
## Mailer parameter end

##************************
## RecordGenerator parameter
##***************************
RECORD_GENERATOR_WIDTH=450
RECORD_GENERATOR_HEIGHT=320
RECORD_GENERATOR_XPOSITION=50
RECORD_GENERATOR_YPOSITION=100
## RecordGenerator parameter end
# IcebergInfo.properties end
```

The iceberg.Iceberg applet gets some permeters from the html page that lunch it.

```
<HTML>
<TITLE>Iceberg</TITLE>
<BODY>
<!--"CONVERTED_APPLET"-->
<!-- CONVERTER VERSION 1.0 -->
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
WIDTH = 200 HEIGHT = 100  codebase="http://java.sun.com/products/plugin/1.1/jinstall-1
<PARAM NAME = CODE VALUE = iceberg.Iceberg.class >
<PARAM NAME = ARCHIVE VALUE = "iceberg.jar, infobus.jar" >

<PARAM NAME="type" VALUE="application/x-java-applet;version=1.1">
<PARAM NAME = "SERVER_URL" VALUE ="rmi://unity.njit.edu:3000">
<PARAM NAME = "SCAN_INTERVAL" VALUE ="60000">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.1" java_CODE = iceberg.Iceberg.class

</NOEMBED></EMBED>
</OBJECT>

<!--
<APPLET  CODE = iceberg.Iceberg.class ARCHIVE = "iceberg.jar, infobus.jar" WIDTH = 200
<PARAM NAME = "SERVER_URL" VALUE ="rmi://unity.njit.edu:3000">
<PARAM NAME = "SCAN_INTERVAL" VALUE ="60000">


</APPLET>
-->
<!--"END_CONVERTED_APPLET"-->

</BODY>
</HTML>
```

The IcebergServerInfo.properties property file is used when the server is launced. It provide the Oracle log in user ID and password. For security reason, the value fields of these line are ommitted.

```
#***********************
# IcebergInfo.properties
#***********************

#**********************
# Database Connectivity
#**********************
Database=jdbc:oracle:thin:@logic.njit.edu:1521:logic40
Username=
Password=

#***********************
# rmiregistry parameters
#***********************
Port=3000

#**************
# Serve Inteval
#**************
SCAN_INTERVAL=60000
WAIT_THRESHOLD=5
```

APPENDIX C

PACKAGE iceberg.advisor

The iceberg.advisor package contains all the visual components used by the Iceberg client of the graduate advisors. These components are separately compiled and are assembled together in the run time to form the Iceberg client.

This package contains following classes:

- BackgrondInfoPane.java
- BridgeRequirementPane.java
- Browser.java
- ControlWindow.java
- DataPane.java
- DataStore.java
- LogInfoCellRender.java
- LogInfoPane.java
- Mailer.java
- PersonalInfoPane.java
- PerosnListPane.java
- RecordGenerator.java
- Searcher.java
- SimpleTableDataModel.java
- TranscriptInfoPane.java

```
/*********** BackgroundInfoPane.java *************/
package iceberg.advisor;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.beans.*;
import com.sun.java.swing.*;
import com.sun.java.swing.table.*;
import com.sun.java.swing.border.*;
import iceberg.guiUtility.*;
import iceberg.util.*;
import javax.infobus.*;

public class BackgroundInfoPane extends IcebergJPanel
implements ActionListener
{
    private JTable                  backgroundTable;
    private SimpleTableDataModel    backgroundData;
    private IcebergInfo             params;
    private boolean                 isStudentShowing = false;

    public BackgroundInfoPane()
    {
        super();
        setLayout( new BorderLayout() );
        setBorder
        (
            BorderFactory.createTitledBorder
            (
                BorderFactory.createLineBorder(Color.black),
                "Background"
            )
        );

        params =
        (IcebergInfo)getDataValue("PARAMETERS",null,false);
        if ( params == null )
        {
System.err.println ( "Can not find system parameters" );
            params = new IcebergInfo ( new Hashtable ( 2 ) );
        }

        String[] headers =
        params.getStrings( "BACKGROUND_TABLE_HEADER" );

        backgroundData =
        new SimpleTableDataModel( headers, null );
        backgroundData.setTableEditable ( true );
        backgroundTable = new JTable( backgroundData );

        TableColumn tc;
        int width, totalWidth=0;
        for( int i=0; i<headers.length; i++ )
        {
            tc = backgroundTable.getColumn( headers[i] );
            width = params.getInt( headers[i] + "_LENGTH" );
            tc.setMinWidth( width );
            totalWidth += width;
        }
        backgroundTable.setPreferredScrollableViewportSize(
        new Dimension( 100, 50 ) );

        JScrollPane backgroundScroll =
        new JScrollPane ( backgroundTable );
        backgroundScroll.setBackground ( Color.lightGray );

        add( backgroundScroll, BorderLayout.CENTER );

      // export the menu items it can react.
        String[] menuLabel =
        { "Add Background", "Delete Background" };
        Vector menuItemList = new Vector ( menuLabel.length );
        for ( int i=0; i<menuLabel.length; i++ )
        {
            JMenuItem item = new JMenuItem ( menuLabel[i] );
            item.addActionListener ( this );
            menuItemList.addElement ( item );
        }
        fireDataItemAvailable("MENU_ITEM_LIST",menuItemList,null);
    }


    public void actionPerformed ( ActionEvent event )
    {
        String arg = event.getActionCommand();
        if ( arg.equals ( "Add Background" ) )
        {
            if ( isStudentShowing )
                backgroundData.addRow();
        }
        else if ( arg.equals ( "Delete Background" ) )
        {
            if ( isStudentShowing )
```

```
          {
              int row = backgroundTable.getSelectedRow();
              if ( row != -1 )
                  backgroundData.removeRow ( row );
              backgroundTable.clearSelection();
          }
      }
  }

  public void cleanup() {
  /*
  int headerSize = backgroundTable.getColumnCount();
  TableColumn tc;
  String columnHeader;
  StringBuffer headerString = new StringBuffer( 80 );
  for( int i=0; i<headerSize; i++ ) {
    columnHeader = backgroundTable.getColumnName( i );
    headerString.append( columnHeader );
    if( i != headerSize -1 )
      headerString.append( "|" );
    tc = backgroundTable.getColumn( columnHeader );
    Iceberg.setInt( columnHeader + Iceberg.lengthSuffix,
    tc.getWidth()/2-5 );
  }
  Iceberg.setString(
  "BACKGROUND_TABLE" + Iceberg.headerSuffix,
   headerString.toString() );
  */
}


/*----------- Methods of bean proerties -------------*/

  public String[] getHeaders()
  {
      return backgroundData.getHeaders();
  }


  public void setHeaders ( String[] h )
  {
      backgroundData.setHeaders ( h );
  }


  public Vector getContent()
  {
        return backgroundData.getContent();
    }


    public void setContent ( Vector c )
    {
        backgroundTable.clearSelection();
        backgroundData.setContent ( c );
    }

  /*---------- Methods of IcebergPanel -------------*/

   /**
    * Remove itself from this InfoBus Object
    */
    protected void removeInfoBusRoles ( InfoBus oldBus )
    {
        oldBus.removeDataProducer ( this );
        oldBus.removeDataConsumer ( this );
    }


   /**
    * Add itself to this InfoBus Object
    */
    protected void addInfoBusRoles ( InfoBus newBus )
    {
        newBus.addDataProducer ( this );
        newBus.addDataConsumer ( this );
    }

  /*--------- Methods of InfoBusDataProducer -----------*/

  // Use default implement


  /*--------- Methods of InfoBusDataConsumer -----------*/

   /**
    * Being noticed that a dataItem is available
    * @param event a InfoBusItemAvailableEvent Object
    */
public void dataItemAvailable(InfoBusItemAvailableEvent event)
    {
        String itemName = event.getDataItemName();
        if ( itemName.equals ( "STUDENT" ) )
        {
            Student student =
            (Student)getDataValue ( event, true );
            if ( student == null )
            {
                isStudentShowing = false;
                setContent ( null );
            }
            else
            {
                isStudentShowing = true;
                setContent ( student.background );
            }
        }
        else if (itemName.equals("BACKGROUND_ENTRY_LIST"))
       {
System.err.println ( "BackgroundInfoPane, get background ");
            setContent((Vector)getDataValue (event,true));
       }
        else if ( itemName.equals ( "SAVE" ) )
        {
            Vector change = backgroundData.getChangedData();
            if ( change != null )
    fireDataItemAvailable("NEW_BACKGROUND_LIST", change, null );
        }
    }



   /**
    * Being noticed that a dataItem is revoked
    * @param event a InfoBusItemRevokedEvent Object
    */
    public void dataItemRevoked(InfoBusItemRevokedEvent event)
    {
        String itemName = event.getDataItemName();
        if ( itemName.equals ( "STUDENT" ) )
        {
            setContent ( null );
            dataItemTable.remove ( "NEW_BACKGROUND_LIST" );
   getInfoBus().fireItemRevoked ( "NEW_BACKGROUND_LIST", this );
        }
    }


   /*---------- Methods of DataItemChangeListener -----------*/

   /**
```

```
    * Being noticed that a dataItem has changed
    * @param event contain the changed information
    */
public void dataItemValueChanged ( DataItemValueChangedEvent event )
    {
        String propertyName =
        (String)event.getProperty ( "NAME" );
        if ( propertyName.equals ( "STUDENT" ) )
        {
            Student student = (Student)getDataValue ( event );
            if ( student == null )
            {
                isStudentShowing = false;
                setContent ( null );
            }
            else
            {
                isStudentShowing = true;
                setContent ( student.background );
            }
        }
        else if ( propertyName.equals ( "BACKGROUND_ENTRY_LIST" ) )
            setContent ( (Vector)getDataValue ( event ) );
    }

}
```

```
/************ BridgeRequirementPane.java **************/
package iceberg.advisor;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.beans.*;
import com.sun.java.swing.*;
import com.sun.java.swing.event.*;
import com.sun.java.swing.border.*;
import javax.infobus.*;
import iceberg.util.*;
import iceberg.guiUtility.*;

public class BridgeRequirementPane extends IcebergJPanel
{

    private CardLayout  requireManager;
    private Integer     currentPno = null;
    private Hashtable   checkBoxTable;
    private Vector      programList = null;
    private Vector      bridgeRequirement = null;
    private IcebergInfo params;

    public BridgeRequirementPane()
    {
        super();
        requireManager = new CardLayout ( 10, 5 );
        setLayout ( requireManager );
        setBorder
        (
            BorderFactory.createTitledBorder
            (
            BorderFactory.createEtchedBorder(),
            "Bridge Requirements"
        )
    );

        params =
        (IcebergInfo)getDataValue ( "PARAMETERS", null, false );
        if ( params == null )
        {
System.err.println ( "Can not find system parameters" );
            params = new IcebergInfo ( new Hashtable ( 2 ) );
        }

    setProgramList((Vector)getDataValue("PROGRAM_LIST",null,false));
```

```
}



public Vector getProgramList()
{
    return programList;
}


public void setProgramList ( Vector p )
{
    programList = p;
    removeAll();
    add( new JPanel(), "EMPTY" );
    if ( programList == null )
        return;

    checkBoxTable = new Hashtable( programList.size() * 2, 1.0f);
    Program prog = null;
    for ( int i=0; i<programList.size(); i++ )
    {
        prog = ( Program ) programList.elementAt(i);
        if ( prog.bridgeCourses == null )
            continue;

        JPanel card =
        new JPanel( new FlowLayout ( FlowLayout.LEFT, 5, 5 ) );
        JCheckBox[] c =
        new JCheckBox [ prog.bridgeCourses.size() ];
        String cno;
        for ( int j=0; j<c.length; j++ )
        {
            cno = ( String ) ( prog.bridgeCourses.elementAt(j) );
            c[j] = new JCheckBox( cno );
            card.add( c[j] );
        }
        checkBoxTable.put( prog.pno, c );
        add( card, prog.pno.toString() );
    }
}


protected void setBridgeRequirement ( Vector bridge )
{
    if ( currentPno == null )
```

```
            return;
        JCheckBox[] c =
        (JCheckBox[])checkBoxTable.get ( currentPno );
        int i, j;
        for ( i=0; i<c.length; i++ )
            c[i].setSelected ( false );
        bridgeRequirement = bridge;
        if ( bridgeRequirement == null )
            return;
        Course course;
        for ( i=0; i<c.length; i++ )
        {
            String cno = c[i].getText();
            for ( j=0; j<bridgeRequirement.size(); j++ )
            {
                course =
                (Course)bridgeRequirement.elementAt(j);
                if( cno.equals ( course.cno ) )
                {
                    c[i].setSelected( true );
                    break;
                }
            }
        }
    }


    protected void setCard ( Student s )
    {
        if ( s==null || !checkBoxTable.containsKey(s.pno))
        {
            requireManager.show( this, "EMPTY" );
            bridgeRequirement = null;
            currentPno = null;
            return;
        }
        currentPno = s.pno;
        setBridgeRequirement ( s.bridgeRequirement );
        requireManager.show ( this, currentPno.toString() );
    }

    protected Vector getBridgeRequirement()
    {
        if ( currentPno == null )
            return null;
        JCheckBox[] c =
        ( JCheckBox[] ) checkBoxTable.get ( currentPno );
        Vector newChoices = new Vector( c.length );
        for ( int i=0; i<c.length; i++ )
        {
            if( c[i].isSelected() )
                newChoices.addElement ( c[i].getText() );
        }

        return newChoices;
    }



    protected void saveChange()
    {
        if ( currentPno == null )
            return;
        JCheckBox[] c = ( JCheckBox[] ) checkBoxTable.get ( currentPno );
        Vector currentRequirement = new Vector( c.length );

        for ( int i=0; i<c.length; i++ )
        {
            if( c[i].isSelected() )
                currentRequirement.addElement ( c[i].getText() );
        }

        bridgeRequirement = currentRequirement;
        fireDataItemAvailable ( "NEW_BRIDGE_REQUIREMENT", currentRequirement, null );
    }


  /*-------------- Methods of IcebergJFrame -----------------*/

    /**
     * Remove itself from this InfoBus Object
     */
    protected void removeInfoBusRoles ( InfoBus oldBus )
    {
        oldBus.removeDataProducer ( this );
        oldBus.removeDataConsumer ( this );
    }


    /**
     * Add itself to this InfoBus Object
     */
  protected void addInfoBusRoles ( InfoBus newBus )
  {
      newBus.addDataProducer ( this );
      newBus.addDataConsumer ( this );
  }

/*-------------- Methods of InfoBusDataProducer -----------------*/

// Use default Value


/*-------------- Methods of InfoBusDataConsumer -----------------*/

 /**
  * Being noticed that a dataItem is available
  * @param event a InfoBusItemAvailableEvent Object
  */
  public void dataItemAvailable ( InfoBusItemAvailableEvent event )
  {
      String itemName = event.getDataItemName();
      if ( itemName.equals ( "PROGRAM_LIST" ) )
      {
          setProgramList((Vector)getDataValue(event, false ) );
      }
      else if ( itemName.equals ( "STUDENT" ) )
      {
          setCard ( (Student)getDataValue ( event, true ) );
      }
      else if ( itemName.equals ( "SAVE" ) )
      {
          saveChange();
      }
      else if ( itemName.equals ( "UNDO" ) )
      {
          setBridgeRequirement ( bridgeRequirement );
      }
      else if ( itemName.equals ( "CLEAR" ) )
      {
          setCard ( null );
      }
  }


 /**
  * Being noticed that a dataItem is revoked
  * @param event a InfoBusItemRevokedEvent Object
```

```
   */
   public void dataItemRevoked ( InfoBusItemRevokedEvent event )
   {
       String itemName = event.getDataItemName();
       if ( itemName.equals ( "STUDENT" ) )
       {
           requireManager.show ( this, "EMPTY" );
           currentPno = null;
           bridgeRequirement = null;

           fireDataItemRevoked ( "NEW_BRIDGE_REQUIREMENT" );
       }

   }


  /*------------- Methods of DataItemChangeListener -------------*/

   /**
    * Being noticed that a dataItem has changed
    * @param event contain the changed information
    */
   public void dataItemValueChanged(DataItemValueChangedEvent event )
    {
        String propertyName = (String)event.getProperty ( "NAME" );
        if ( propertyName.equals ( "STUDENT" ) )
        {
            setCard ( (Student)getDataValue ( event ) );
        }
    }


}
```

```
/************* Browser.java *****************/
package iceberg.advisor;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.beans.*;
import com.sun.java.swing.*;
import com.sun.java.swing.event.*;
import com.sun.java.swing.border.*;
import iceberg.util.*;
import iceberg.guiUtility.*;
import javax.infobus.*;
import iceberg.Iceberg;

public class Browser extends IcebergJFrame
implements ActionListener
{

    private JComboBox       programView;
    private JList           procedureView;
    private Vector          programList;
    private Vector          procedureList;
    private Pin             pinBox;
    private IcebergInfo     params;


    public Browser()
    {
        super();

        params =
        (IcebergInfo)getDataValue ( "PARAMETERS", null, false );
        if ( params == null )
        {
System.err.println ( "Can not find system parameters" );
            params = new IcebergInfo ( new Hashtable ( 2 ) );
        }

        setContentPane( createMainPane() );

        pack();

        setSize
        (
            params.getInt( "BROWSER_WIDTH" ),
            params.getInt( "BROWSER_HEIGHT" )
    );

    setLocation
    (
        params.getInt( "BROWSER_XPOSITION" ),
        params.getInt( "BROWSER_YPOSITION" )
    );

    setProgramList(
    (Vector)getDataValue("PROGRAM_LIST", null, true ) );

    setProcedureList(
    (Vector)getDataValue ( "PROCEDURE_LIST", null, true ) );

}


protected JPanel createMainPane()
{
    JPanel mainPane = new JPanel( new BorderLayout(5, 5), true );
    mainPane.setBorder( BorderFactory.createEtchedBorder() );

    mainPane.add( createToolBar(), BorderLayout.NORTH );

    mainPane.add( createContent(), BorderLayout.CENTER );

        String[] cmd = { "Run", "Cancel" };
        JButtonPanel buttonBox = new JButtonPanel ( cmd );
        buttonBox.addActionListener ( this );

    mainPane.add ( buttonBox, BorderLayout.SOUTH );

    return mainPane;
}


protected JToolBar createToolBar()
{
    JToolBar toolBar = new JToolBar();

        pinBox = new Pin();

    toolBar.add( pinBox );
    return toolBar;
}
```

```
protected Container createContent( )
{
    JPanel panel = new JPanel();
    panel.setLayout( new BoxLayout( panel, BoxLayout.Y_AXIS ) );
    panel.setBorder(
    BorderFactory.createEmptyBorder( 0, 15, 5, 15 ) );

    panel.add( new JLabel( "Choose a Stored Procedure to Run" ) );

    panel.add( Box.createVerticalStrut ( 5 ) );

        programView = new JComboBox();
        programView.setEditable( false );

    panel.add ( programView );

    panel.add( Box.createVerticalStrut ( 5 ) );

        procedureView = new JList();
        JScrollPane scroll = new JScrollPane( procedureView );

    panel.add( scroll );
    return panel;
}


public Vector getProgramList()
{
    return programList;
}


public void setProgramList ( Vector p )
{
    programList = p;
    programView.removeAllItems();
    if ( programList != null )
    {
        Program prog;
        for ( int i=0; i<programList.size(); i++ )
        {
            prog = ( Program ) programList.elementAt( i );
            programView.addItem( prog.pName );
        }
        programView.setSelectedIndex ( 0 );
    }
}
```

```
    public Vector getProcedureList()
    {
        return procedureList;
    }


    public void setProcedureList ( Vector p )
    {
        procedureList = p;
        procedureView.clearSelection();
        if ( procedureList != null )
        {
            String[] description = new String [ procedureList.size() ];
            StoredProcedure sp;
            for ( int i=0; i< description.length; i++ )
            {
                sp = (StoredProcedure)procedureList.elementAt(i);
                description [ i ] = sp.description;
            }
            procedureView.setListData ( description );
        }
        else
            procedureView.setListData ( new String[] {} );
    }


    public void actionPerformed ( ActionEvent event )
    {
        String arg = event.getActionCommand();
        if ( arg.equals( "Run" ) )
        {
            int programIndex = programView.getSelectedIndex();
            int procedureIndex = procedureView.getSelectedIndex();

            if( programIndex == -1 || procedureIndex == -1 )
            {
                JOptionPane.showMessageDialog
                (
                    this,
                    "Not enough information",
                    "Error",
    JOptionPane.ERROR_MESSAGE
);
return;
            }
            BrowseInfo bi = new BrowseInfo();
            bi.pno =
            ((Program )programList.elementAt ( programIndex ) ).pno;
            b.spno =
  ((StoredProcedure)procedureList.elementAt(procedureIndex ) ).spno;

            fireDataItemAvailable ( "BROWSE_INFO", bi, null );

            if( pinBox.getState() == Pin.UP )
processWindowEvent(new WindowEvent(this, WindowEvent.WINDOW_CLOSING));
        }
        else if ( arg.equals( "Cancel" ) )
processWindowEvent(new WindowEvent(this,WindowEvent.WINDOW_CLOSING ));
        else
System.err.println( "Searcher unknown event: " + event.toString() );
    }

  public void cleanup() {
  /*
    Dimension size = getSize();
    if( width != size.width )
      Iceberg.setInt( "BROWSER_WIDTH", size.width );
    if( height != size.height )
      Iceberg.setInt( "BROWSER_HEIGHT", size.height );
    if( isShowing() ) {
      Point origion = getLocationOnScreen();
      if( xPos != origion.x )
        Iceberg.setInt( "BROWSER_XPOSITION", origion.x );
      if( yPos != origion.y )
        Iceberg.setInt( "BROWSER_YPOSITION", origion.y );
    }
   */
  }


  /*-------------- Methods of IcebergJFrame -----------------*/

   /**
    * Remove itself from this InfoBus Object
    */
    protected void removeInfoBusRoles ( InfoBus oldBus )
    {
        oldBus.removeDataProducer ( this );
        oldBus.removeDataConsumer ( this );
    }
  /**
   * Add itself to this InfoBus Object
   */
   protected void addInfoBusRoles ( InfoBus newBus )
   {
       newBus.addDataProducer ( this );
       newBus.addDataConsumer ( this );
   }

 /*-------------- Methods of InfoBusDataProducer -----------------*/

// Use default implementation


 /*-------------- Methods of InfoBusDataConsumer -----------------*/

  /**
   * Being noticed that a dataItem is available
   * @param event a InfoBusItemAvailableEvent Object
   */
   public void dataItemAvailable ( InfoBusItemAvailableEvent event )
   {
       String itemName = event.getDataItemName();
       if ( itemName.equals ( "PROGRAM_LIST" ) )
       {
           setProgramList ( (Vector)getDataValue ( event, true ) );
       }
       else if ( itemName.equals ( "PROCEDURE_LIST" ) )
       {
        setProcedureList ( (Vector)getDataValue ( event, true ) );
       }

   }



  /**
   * Being noticed that a dataItem is revoked
   * @param event a InfoBusItemRevokedEvent Object
   */
   public void dataItemRevoked ( InfoBusItemRevokedEvent event )
   {
       String itemName = event.getDataItemName();
       if ( itemName.equals ( "PROGRAM_LIST" ) )
       {
           setProgramList ( null );
        }
        else if ( itemName.equals ( "PROCEDURE_LIST" ) )
        {
            setProcedureList( null );
        }
    }


  /*------------- Methods of DataItemChangeListener --------------*/

   /**
    * Being noticed that a dataItem has changed
    * @param event contain the changed information
    */
public void dataItemValueChanged ( DataItemValueChangedEvent event )
    {
        String propertyName =( String ) event.getProperty ( "NAME" );
        if ( propertyName.equals ( "PROGRAM_LIST" ) )
        {
            setProgramList ( (Vector)getDataValue ( event ) );
        }
        else if ( propertyName.equals ( "PROCEDURE_LIST" ) )
        {
            setProcedureList ( (Vector)getDataValue ( event ) );
        }
    }

}
```

```
package iceberg.advisor;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.rmi.*;
import com.sun.java.swing.*;
import javax.infobus.*;
import iceberg.util.*;
import iceberg.guiUtility.*;
import iceberg.server.*;
import iceberg.Iceberg;

/**
 * Main application window
 *
 * @author  Frank T. Lin
 */
public class ControlWindow extends IcebergJFrame
implements  ActionListener
{
    private Hashtable   toolTable;
    private JMenuBar    menuBar;
    private IcebergInfo params;

    public ControlWindow( Long sessionID )
    {
        super();

        params =
        (IcebergInfo)getDataValue ( "PARAMETERS", null, false );
        if ( params == null )
        {
System.err.println ( "Can not find system parameters" );
            params = new IcebergInfo ( new Hashtable ( 2 ) );
        }

        new DataStore ( sessionID );

        toolTable = new Hashtable ( 10 );
        menuBar = createMenuBar();
        setJMenuBar( menuBar );

        JPanel mainPane = new JPanel ( new BorderLayout(), true );
//       mainPane.setBorder ( BorderFactory.createEtchedBorder() ) ;

        mainPane.add ( createToolbar(), BorderLayout.NORTH );
        try
        {
            String innerPaneClass =
            params.getString ( "INNER_FRAMEWORK" );
            JPanel innerPane =
            (JPanel)Class.forName ( innerPaneClass ).newInstance();
            mainPane.add( innerPane, BorderLayout.CENTER );
        }
        catch ( Exception e )
        {
            e.printStackTrace();
        }

        setContentPane( mainPane );

        setSize
        (
            params.getInt( "CONTROL_WINDOW_WIDTH" ),
            params.getInt( "CONTROL_WINDOW_HEIGHT" )
        );
        setLocation
        (
            params.getInt( "CONTROL_WINDOW_XPOSITION" ),
            params.getInt( "CONTROL_WINDOW_YPOSITION" )
        );

        show();
    }


    protected JMenuBar createMenuBar()
    {
        JMenuBar mb = new JMenuBar();
        String[] menuKeys = params.getStrings ( "MENUBAR" );
        for ( int i = 0; i < menuKeys.length; i++ )
        {
            JMenu m = createMenu ( menuKeys[i] );
            if ( m != null )
                mb.add ( m );
        }
        return mb;
    }


    protected JMenu createMenu ( String key )
    {
        String[] itemKeys = params.getStrings ( key );
        JMenu menu = new JMenu ( key );
        for ( int i = 0; i < itemKeys.length; i++ )
        {
            if ( itemKeys[i].equals("-") )
                menu.addSeparator();
            else
            {
                JMenuItem mi = createMenuItem ( itemKeys[i] );
                menu.add(mi);
            }
        }
        return menu;
    }


    protected JMenuItem createMenuItem ( String cmd )
    {
        JMenuItem mi = new JMenuItem( cmd );

        String imageFile = params.getString ( cmd + "_IMAGE" );
        if ( imageFile != null)
        {
            mi.setHorizontalTextPosition ( JButton.RIGHT );
            mi.setIcon( ImageIconFactory.getImageIcon (imageFile ) );
        }

        if ( toolTable.get ( cmd ) == null )
        {
            String actionClass =
            params.getString ( cmd + "_ACTION" );
            if ( actionClass != null )
            {
                try
                {
      toolTable.put(cmd,Class.forName(actionClass).newInstance());
                }
                catch ( Exception e )
                {
                    e.printStackTrace();
                }
            }
        }

        mi.addActionListener ( this );
        return mi;
    }
```

```
protected JToolBar createToolbar()
{
    JToolBar toolBar = new JToolBar();
    String[] toolKeys = params.getStrings ( "TOOLBAR" );
    for ( int i = 0; i < toolKeys.length; i++ )
    {
        if ( toolKeys[i].equals ( "-" ) )
            toolBar.add ( Box.createHorizontalStrut(5) );
        else
            toolBar.add ( createToolBarButton ( toolKeys[i] ) );
    }
    toolBar.add(Box.createHorizontalGlue());
    return toolBar;
}


protected JButton createToolBarButton ( String cmd )
{
    String imageFile = params.getString ( cmd + "_IMAGE" );
    JButton b =
    new JButton( ImageIconFactory.getImageIcon ( imageFile ) )
    {
        public float getAlignmentY() { return 0.5f; }
    };

    b.setRequestFocusEnabled(false);
    b.setMargin ( new Insets( 1, 1, 1, 1 ) );

    b.setActionCommand( cmd );
    String tip = params.getString ( cmd + "_TOOLTIP" );
    if (tip != null)
        b.setToolTipText ( tip );

    if ( toolTable.get ( cmd ) == null )
    {
String actionClass =
params.getString ( cmd + "_ACTION" );
        if ( actionClass != null )
        {
            try
            {
toolTable.put ( cmd, Class.forName ( actionClass ).newInstance() );
            }
            catch ( Exception e )
            {
                    e.printStackTrace();
                }
            }
        }

        b.addActionListener ( this );
        return b;
    }


    public void actionPerformed ( ActionEvent event )
    {
        String arg = event.getActionCommand();
        if ( arg.equals("Exit") )
        {
processWindowEvent(new WindowEvent(this,WindowEvent.WINDOW_CLOSING));
        }
        else if ( toolTable.get ( arg ) != null )
        {
            Frame tool = ( Frame )toolTable.get ( arg );
            if ( tool.isShowing() )
                tool.toFront();
            else
                tool.setVisible ( true );
        }
        else
        {
getInfoBus().fireItemAvailable ( arg.toUpperCase(), null, this );
        }
    }


    protected void cleanup()
    {
        Enumeration enum = toolTable.elements();
        Window tool;
        while ( enum.hasMoreElements() )
        {
            tool = (Window)enum.nextElement();
            tool.dispose();
        }
    }


  /*--------------- Methods of IcebergJFrame --------------------*/
   /**
    * Remove itself from this InfoBus Object
    */
    protected void removeInfoBusRoles ( InfoBus oldBus )
    {
        oldBus.removeDataConsumer ( this );
        oldBus.removeDataProducer ( this );
    }


   /**
    * Add itself to this InfoBus Object
    */
    protected void addInfoBusRoles ( InfoBus newBus )
    {
        newBus.addDataConsumer ( this );
        newBus.addDataProducer ( this );
    }


  /*------------- Methods of InfoBusDataConsumer -----------------*/
   /**
    * Being noticed that a dataItem is available
    * @param event a InfoBusItemAvailableEvent Object
    */
    public void dataItemAvailable( InfoBusItemAvailableEvent event )
    {
        String itemName = event.getDataItemName();
        if ( itemName.equals ( "MENU" ) )
        {
            menuBar.add ( (JMenu)getDataValue ( event, false ) );
        }
        else if ( itemName.equals ( "EXIT_REQUEST" ) )
processWindowEvent(new WindowEvent(this,WindowEvent.WINDOW_CLOSING));
        else if ( itemName.equals ( "EXIT" ) )
            cleanup();
    }
}
```

```
/********** DataPane.java ************/
package iceberg.advisor;

import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.io.*;
import java.util.*;
import com.sun.java.swing.*;
import com.sun.java.swing.event.*;
import com.sun.java.swing.border.*;
import javax.infobus.*;
import iceberg.util.*;
import iceberg.guiUtility.*;
import iceberg.Iceberg;

class DataPane extends IcebergJPanel
{

    private JMenu     currentMenu = null;
    private IcebergInfo params;

    DataPane()
    {
        super();
        setLayout( new BorderLayout() );

        params =
        (IcebergInfo)getDataValue ( "PARAMETERS", null, false );
        if ( params == null )
        {
System.err.println ( "Can not find system parameters" );
            params = new IcebergInfo ( new Hashtable ( 2 ) );
        }

      // Add a student list
        add ( new PersonListPane(), BorderLayout.WEST );

      // Add tab pane
        add ( createStudentTabPane(), BorderLayout.CENTER );
    }


    protected JComponent createStudentTabPane()
    {
        String tabs[] =
        params.getStrings ( "STUDENT_TAB_PANE" + "_HEADER" );
        JTabbedPane studentTabPane = new JTabbedPane();
        studentTabPane.setBorder( BorderFactory.createEtchedBorder() );

        for ( int i=0;i<tabs.length;i++)
       studentTabPane.addTab( tabs[i], createTabContent( tabs[i] ) );
        studentTabPane.setSelectedIndex(0);

        return studentTabPane;
    }


    protected JComponent createTabContent( String header )
    {
        String menuLabel =
        params.getString ( header + "_MENU_LABEL" );
        if ( menuLabel != null )
            currentMenu = new JMenu ( menuLabel );

        String[] components =
        params.getStrings ( header + "_HEADER" );
        JPanel pane = new JPanel();
        pane.setLayout( new GridBagLayout() );

        GridBagConstraints constraints = new GridBagConstraints();
        constraints.gridx = 0;
        constraints.gridy = GridBagConstraints.RELATIVE;
        constraints.gridwidth = 1;
        constraints.weightx = 100;
        constraints.fill = GridBagConstraints.BOTH;

        JComponent comp;
        try
        {
            String actionClass;
            for ( int i=0; i<components.length; i++ )
            {
                actionClass =
                params.getString ( components[i] + "_ACTION" );
System.out.println( actionClass );
                comp =
               ( Component)Class.forName(actionClass ).newInstance();
                constraints.gridheight =
                params.getInt( components[i] + "_HEIGHT" );
            constraints.weighty =
           (double)params.getInt(components[i] + "_WEIGHT" );
        constraints.weighty = (new Integer( weightY )).doubleValue();
```

```
                pane.add( comp, constraints );
            }
        }
        catch( Exception e )
        {
        // We can do nothing.
            e.printStackTrace();
        }

        if ( currentMenu != null )
        {
            fireDataItemAvailable ( "MENU", currentMenu, null );
            currentMenu = null;
        }
        return pane;
    }


public void cleanup() {
/*
  IcebergCallback callback;
  for( int i=0; i<componentList.size(); i++ ) {
    callback = (IcebergCallback)componentList.elementAt( i );
    callback.cleanup();
  }
*/
}

  protected void addMenuItems ( Vector menuItemList )
  {
      if(currentMenu == null ||
      menuItemList == null || menuItemList.size() == 0 )
         return;
      if ( currentMenu.getItemCount() != 0 )
          currentMenu.addSeparator();

      for ( int i=0; i<menuItemList.size(); i++ )
      {
          currentMenu.add ( (JMenuItem)menuItemList.elementAt(i) );
      }
  }


/*--------------- Methods of IcebergJFrame -------------------*/
 /**
  * Remove itself from this InfoBus Object
  */
  protected void removeInfoBusRoles ( InfoBus oldBus )
  {
      oldBus.removeDataConsumer ( this );
      oldBus.removeDataProducer ( this );
  }


 /**
  * Add itself to this InfoBus Object
  */
  protected void addInfoBusRoles ( InfoBus newBus )
  {
      newBus.addDataConsumer ( this );
      newBus.addDataProducer ( this );
  }


/*-------------- Methods of InfoBusDataProducer -----------------*/

// Use default implement


/*-------------- Methods of InfoBusDataConsumer -----------------*/

 /**
  * Being noticed that a dataItem is available
  * @param event a InfoBusItemAvailableEvent Object
  */
  public void dataItemAvailable ( InfoBusItemAvailableEvent event )
  {

      String itemName = event.getDataItemName();
 if ( itemName.equals ( "MENU_ITEM_LIST" ) && currentMenu != null )
      {
          addMenuItems ( (Vector)getDataValue ( event, false ) );

      }
  }

}
```

```
/************* DataStore.java *****************/
package iceberg.advisor;

import java.rmi.*;
import java.util.*;
import java.beans.*;
import java.awt.event.*;
import java.awt.*;
import com.sun.java.swing.*;
import com.sun.java.swing.event.*;
import iceberg.util.*;
import iceberg.server.*;
import javax.infobus.*;
import iceberg.Iceberg;

class DataStore extends IcebergInfoBusMember
{
    private AdvisorServerX          advisorServer;
    private Long                    sessionID;
    private IcebergInfo             params;

    DataStore( Long id )
    {
        super();
        sessionID = id;
        params =
        (IcebergInfo)getDataValue ( "PARAMETERS", null, false );
        if ( params == null )
        {
System.err.println ( "Can not find system parameters" );
            params = new IcebergInfo ( new Hashtable ( 2 ) );
        }
        init();
    }


    protected void init()
    {
        try
        {
            advisorServer = ( AdvisorServerX ) Naming.lookup (
            Iceberg.serverURL + "/AdvisorServer" );

            Advisor advisor = advisorServer.getAdvisor ( sessionID );

            fireDataItemAvailable ( "CURRENT_USER", advisor, null  );
```

```
fireDataItemAvailable("PROGRAM_LIST" advisor.getProgramList() null );

            Vector procedureList =
            advisorServer.getStoredProcedureList ( sessionID );

       fireDataItemAvailable("PROCEDURE_LIST", procedureList, null );

            Vector eMessageList =
            advisorServer.getEMessageList ( sessionID );
       fireDataItemAvailable ( "EMESSAGE_LIST", eMessageList, null );
        }
        catch( Exception e )
        {
            e.printStackTrace();

            JOptionPane.showMessageDialog
            (
                null,
                "Initialization failed!",
                "Iceberg will closing",
                JOptionPane.ERROR_MESSAGE
            );

       getInfoBus().fireItemAvailable ( "EXIT_REQUEST", null, this );
        }
    }



    protected void processData ( String itemName, Object item )
    {
        try
        {
            if ( itemName.equals ( "SEARCH_INFO" ) )
            {
                Vector personList =
     advisorServer.getStudentList ( sessionID, ( SearchInfo ) item );
           fireDataItemAvailable ( "PERSON_LIST", personList, null );

                if ( personList == null || personList.size() == 0 )
                {
                    JOptionPane.showMessageDialog
                    (
                        null,
                        "Did not match any student",
                        "Search Result",
                    JOptionPane.INFORMATION_MESSAGE
                .   );
                }
            }
            else if ( itemName.equals ( "BROWSE_INFO" ) )
            {
                Vector personList =
        advisorServer.getStudentList ( sessionID, ( BrowseInfo ) item );
              fireDataItemAvailable ( "PERSON_LIST", personList, null );
            }
            else if ( itemName.equals ( "PERSON_ID" ) )
            {
                if ( item == null )
                {
                    fireDataItemAvailable ( "PERSON", null, null );
                    fireDataItemAvailable ( "STUDENT", null, null );
             fireDataItemAvailable ( "LOG_ENTRY_LIST", null, null );
                    return;
                }

                String id = (String)item;
                Student student =
                advisorServer.getStudent ( sessionID, id );
                fireDataItemAvailable ( "STUDENT", student, null );

                fireDataItemAvailable ( "PERSON", student, null );

                Vector logList =
                advisorServer.getLogEntryList ( sessionID );
              fireDataItemAvailable ( "LOG_ENTRY_LIST", logList, null );

            }
            else if ( itemName.equals ( "NEW_BRIDGE_REQUIREMENT" ) )
            {
       advisorServer.updateBridgeRequirement ( sessionID, (Vector)item );
            }
            else if ( itemName.equals ( "NEW_LOG_ENTRY_LIST" ) )
            {
                Vector entryList = (Vector)item;
                ValuePair vp;
                for ( int i=0; i<entryList.size(); i++ )
                {
                    vp = (ValuePair)entryList.elementAt(i);

                    LogEntry entry = (LogEntry)vp.oldValue;
                    if ( entry != null )
                   advisorServer.removeLogEntry ( sessionID, entry );
```

```
                 entry = (LogEntry)vp.newValue;
                 if ( entry != null )
                   advisorServer.addLogEntry ( sessionID, entry );
             }

             Vector logList =
             advisorServer.getLogEntryList ( sessionID );
        fireDataItemAvailable ( "LOG_ENTRY_LIST", logList, null );
         }
         else if ( itemName.equals ( "NEW_STUDENT" ) )
         {

             advisorServer.addStudent ( sessionID,(Student)item );
             Vector pList = new Vector ( 1 );
             pList.addElement ( item );
             fireDataItemAvailable ( "PERSON_LIST", pList, null );
         }
         else if ( itemName.equals ( "NEW_PERSONAL_INFO" ) )
         {
             advisorServer.updatePerson(sessionID, (Person)item );
         }
         else if ( itemName.equals ( "EMESSAGE" ) )
         {
             ValuePair vp = (ValuePair)item;
             if ( vp.oldValue != null )
advisorServer.removeEMessage ( sessionID, (EMessage)vp.oldValue );
             if ( vp.newValue != null )
   advisorServer.addEMessage ( sessionID, (EMessage)vp.newValue );

             Vector eList =
             advisorServer.getEMessageList ( sessionID );
           fireDataItemAvailable ( "EMESSAGE_LIST", eList, null );
         }
         else if ( itemName.equals ( "NEW_BACKGROUND_LIST" ) )
         {
             Vector backgroundList = (Vector)item;
             ValuePair vp;
             for ( int i=0; i<backgroundList.size(); i++ )
             {
                 vp = (ValuePair)backgroundList.elementAt(i);
                 if ( vp.oldValue != null )
                     advisorServer.removeBackgroundEntry
                     ( sessionID, (Hashtable)vp.oldValue );
                 if ( vp.newValue != null )
                     advisorServer.addBackgroundEntry
                     ( sessionID, (Hashtable)vp.newValue );
```

```
                }
                Vector bList =
                advisorServer.getBackgroundEntryList ( sessionID );
System.err.println ( bList.toString() );
     fireDataItemAvailable ( "BACKGROUND_ENTRY_LIST", bList, null );
            }
        }
        catch ( DataProcessException e )
        {
            JOptionPane.showMessageDialog
            (
                null,
                e.toString(),
                "Alert",
                JOptionPane.ERROR_MESSAGE
            );
        }
        catch ( SessionNotExistException e )
        {
            JOptionPane.showMessageDialog
            (
                null,
                "Session not exist or closed by Iceberg Server",
                "Iceberg will close",
                JOptionPane.ERROR_MESSAGE
            );
       getInfoBus().fireItemAvailable ( "EXIT_REQUEST", null, this );
        }
        catch ( RemoteException e )
        {
       System.err.println ( e.toString() );
            e.printStackTrace();
            JOptionPane.showMessageDialog
            (
                null,
                "Network error occurs",
                "Iceberg will close.",
                JOptionPane.ERROR_MESSAGE
            );
   //  getInfoBus().fireItemAvailable ( "EXIT_REQUEST", null, this );
        }
    }


    protected void processCommand ( String cmd )
    {
    }
```

```
 /*------------- Methods of IcebergJFrame ----------------*/

  /**
   * Remove itself from this InfoBus Object
   */
   protected void removeInfoBusRoles ( InfoBus oldBus )
   {
       oldBus.removeDataProducer ( this );
       oldBus.removeDataConsumer ( this );
   }


  /**
   * Add itself to this InfoBus Object
   */
   protected void addInfoBusRoles ( InfoBus newBus )
   {
       newBus.addDataProducer ( this );
       newBus.addDataConsumer ( this );
   }

 /*------------- Methods of InfoBusDataProducer ----------------*/

// Use default implementation


 /*------------- Methods of InfoBusDataConsumer ----------------*/

  /**
   * Being noticed that a dataItem is available
   * @param event a InfoBusItemAvailableEvent Object
   */
   public void dataItemAvailable ( InfoBusItemAvailableEvent event )
   {
       if ( event.getSourceAsProducer() == this )
           return;
       String itemName = event.getDataItemName();
System.err.println ( "DataStore:available: " + itemName );
       Object item = event.requestDataItem ( this, null );
       if ( item == null )
           processCommand ( itemName );
       else
       {
           processData
```

```
            (
                itemName,
                ( (ImmediateAccess)item ).getValueAsObject()
            );
  ( (DataItemChangeManager)item ).addDataItemChangeListene (this );
        }
    }



   /**
    * Being noticed that a dataItem is revoked
    * @param event a InfoBusItemRevokedEvent Object
    */
    public void dataItemRevoked ( InfoBusItemRevokedEvent event )
    {
        if ( event.getSourceAsProducer() == this )
            return;
        String itemName = event.getDataItemName();
System.err.println ( "DataStore:revoked: " + itemName );
        processData ( itemName, null );
    }


  /*--------------- Methods of DataItemChangeListener -------------*/

   /**
    * Being noticed that a dataItem has changed
    * @param event contain the changed information
    */
 public void dataItemValueChanged ( DataItemValueChangedEvent event )
    {
        String itemName = ( String ) event.getProperty ( "NAME" );
System.err.println ( "DataStore:valueChange: " + itemName );
        processData ( itemName, getDataValue ( event ) );
    }

}
```

```
/************ LogInfoCellRender.java **************/
    class LogInfoCellRender extends JPanel implements ListCellRenderer
    {

        private JTextArea contentArea;

        LogInfoCellRender()
        {
            super();
            setLayout ( new BorderLayout ( 2, 5 ) );
        }

        public Component getListCellRendererComponent ( JList list,
                                                       Object value,
                                                       int index,
                                                       boolean isSelected,
                                                       boolean cellHasFocus )
        {
            ValuePair vp = (ValuePair)value;
            LogEntry entry = (LogEntry)vp.oldValue;

            add
            (
                new JLabel(entry.timeStamp.toString().substring (10 )
                 + "    " + entry.fromName ),
                BorderLayout.NORTH
            );

            contentArea = new JTextArea ( entry.content );
            add ( contentArea, BorderLayout.CENTER );
            sle.hasBeenSelected |= isSelected;
            if( sle.hasBeenSelected )
                setIcon ( smileIcon );
            else
                setIcon ( cryIcon );

            setText ( sle.name );

            if( isSelected )
            {
                setBackground ( SystemColor.textHighlight );
                setForeground ( SystemColor.textHighlightText );
            }
            else
            {
                setBackground ( list.getBackground() );
                setForeground (
                sle.hasBeenSelected? Color.magenta : Color.blue );
            }

            return this;
        }

/*
      public Dimension getPreferredSize() {
        Dimension dim = super.getPreferredSize();
        dim.width += 16;  // widen
        return dim;
      }
*/
    }
```

```
/************* LogInfoPane.java ****************/
package iceberg.advisor;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.beans.*;
import com.sun.java.swing.*;
import com.sun.java.swing.event.*;
import com.sun.java.swing.table.*;
import com.sun.java.swing.border.*;
import javax.infobus.*;
import iceberg.Iceberg;
import iceberg.guiUtility.*;
import iceberg.util.*;


public class LogInfoPane extends IcebergJPanel
implements ActionListener
{
    private Box         logView;
    private Vector      logEntryList;
    private Vector      logEntryViewList = null;
    private Person      currentUser;

    public LogInfoPane()
    {
        super();
        setLayout ( new BorderLayout() );
        setBorder
        (
            BorderFactory.createTitledBorder
            (
                BorderFactory.createLineBorder( Color.black ),
                "System Log"
            )
        );
        logView = new Box ( BoxLayout.Y_AXIS );
        add ( new JScrollPane ( logView ), BorderLayout.CENTER );

        currentUser =
        (Person)getDataValue ( "CURRENT_USER", null, false );

        String[] menuLabel = { "Add Log Entry" };
        Vector menuItemList = new Vector ( menuLabel.length );
        for ( int i=0; i<menuLabel.length; i++ )
        {
            JMenuItem item = new JMenuItem ( menuLabel[i] );
            item.addActionListener ( this );
            menuItemList.addElement ( item );
        }
      fireDataItemAvailable ( "MENU_ITEM_LIST", menuItemList, null );
    }


/*
    public void setBounds ( int x,
                            int y,
                            int width,
                            int height )
    {

        if ( logEntryViewList != null )
        {
            LogEntryView entryView;
            for ( int i=0; i<logEntryViewList.size(); i++ )
            {
                entryView =
                (LogEntryView)logEntryViewList.elementAt(i);
                entryView.reshapeContent ( width );
            }
        }

        super.setBounds ( x, y, width, height );
    }
*/


    public void actionPerformed ( ActionEvent event )
    {
        if ( event.getActionCommand().equals ( "Add Log Entry" ) )
        {
            LogEntryView entryView = new LogEntryView ( null );
            if ( logEntryViewList == null )
                logEntryViewList = new Vector ( 5 );
            logEntryViewList.addElement ( entryView );
            logView.add ( entryView );

            logView.validate();
        }
    }


    public void saveChange()
```

```
  {
      if ( logEntryViewList == null )
          return;
      LogEntryView entryView;
      Vector changeList = new Vector ( logEntryViewList.size() );
      for ( int i=0; i<logEntryViewList.size(); i++ )
      {
          entryView = (LogEntryView)logEntryViewList.elementAt(i);
          if ( entryView.isModified() )
          {
              ValuePair vp = entryView.getLogEntryPair();
              changeList.addElement ( vp );
          }
      }
      if ( changeList.size() != 0 )
  fireDataItemAvailable ( "NEW_LOG_ENTRY_LIST", changeList, null );
  }


/*------------- Methods of Bean Properties ----------------*/

  public Vector getLogEntryList()
  {
      return logEntryList;
  }


  public void setLogEntryList ( Vector l )
  {
      removeAll();
      logView = new Box ( BoxLayout.Y_AXIS );
      add ( new JScrollPane ( logView ), BorderLayout.CENTER );

      logEntryList = l;
      logEntryViewList = null;

      if ( logEntryList != null )
      {
          logEntryViewList = new Vector ( logEntryList.size() );
          LogEntryView entryPane;
          for ( int i=0; i<logEntryList.size(); i++ )
          {
              entryPane =
          new LogEntryView ( (LogEntry)logEntryList.elementAt(i) );
              logEntryViewList.addElement ( entryPane );
              logView.add ( entryPane );
          }
```

```
        }

        validate();
    }

/*-------------- Methods of IcebergJPanel -----------------*/

    /**
     * Remove itself from this InfoBus Object
     */
    protected void removeInfoBusRoles ( InfoBus oldBus )
    {
        oldBus.removeDataProducer ( this );
        oldBus.removeDataConsumer ( this );
    }


    /**
     * Add itself to this InfoBus Object
     */
    protected void addInfoBusRoles ( InfoBus newBus )
    {
        newBus.addDataProducer ( this );
        newBus.addDataConsumer ( this );
    }

/*-------------- Methods of InfoBusDataProducer -----------------*/

// Use default implement


/*-------------- Methods of InfoBusDataConsumer -----------------*/

    /**
     * Being noticed that a dataItem is available
     * @param event a InfoBusItemAvailableEvent Object
     */
    public void dataItemAvailable ( InfoBusItemAvailableEvent event )
    {
        String itemName = event.getDataItemName();
        if ( itemName.equals ( "LOG_ENTRY_LIST" ) )
        {
            setLogEntryList ( (Vector)getDataValue ( event, true ) );
        }
        else if ( itemName.equals ( "SAVE" ) )
            saveChange();
    }
```

```
 /**
  * Being noticed that a dataItem is revoked
  * @param event a InfoBusItemRevokedEvent Object
  */
  public void dataItemRevoked ( InfoBusItemRevokedEvent event )
  {
      String itemName = event.getDataItemName();
      if ( itemName.equals ( "LOG_ENTRY_LIST" ) )
      {
          setLogEntryList ( null );
          fireDataItemRevoked ( "NEW_LOG_ENTRY_LIST" );
      }
  }


/*------------- Methods of DataItemChangeListener -------------*/

 /**
  * Being noticed that a dataItem has changed
  * @param event contain the changed information
  */
public void dataItemValueChanged ( DataItemValueChangedEvent event )
  {
      String propertyName = (String) event.getProperty ( "NAME" );
      if ( propertyName.equals ( "LOG_ENTRY_LIST" ) )
      {
          setLogEntryList ( (Vector)getDataValue ( event ) );
      }
  }


/****************** LogEntryView.java **********************/
  class LogEntryView extends JPanel implements CaretListener
  {
      private JLabel    header;
      private JTextArea contentArea;
      private boolean   modified = false;
      private ValuePair logEntryPair;

      LogEntryView ( LogEntry entry )
      {
          super();
          logEntryPair = new ValuePair ( entry, null );
```

```
        setLayout ( new BorderLayout( ) );

            if ( entry == null )
                header =
    new JLabel ( "New Entry        " + currentUser.getName() );
            else
                header = new JLabel
        ( entry.timeStamp.toString() + "    " + entry.fromName );
        setBorder(BorderFactory.createEmptyBorder( 5, 5, 5, 5 ));
        add ( header, BorderLayout.NORTH );

        Dimension size = logView.getSize();
        reshapeContent ( size.width );
    }


    public void reshapeContent( int width )
    {
        if ( contentArea != null )
            remove ( contentArea );

        LogEntry data;
        if ( logEntryPair.newValue == null )
            data = (LogEntry)logEntryPair.oldValue;
        else
            data = (LogEntry)logEntryPair.newValue;

        if ( data == null || data.content == null )
            contentArea = new JTextArea ( 1, 20 );
        else
        {
//       FontMetrics fm = getToolkit().getFontMetrics ( getFont() );
            FontMetrics fm = getFontMetrics ( getFont() );
            int contentWidth = fm.stringWidth ( data.content );
            int rows = contentWidth / ( width - 20 ) + 1;
            contentArea =
            new JTextArea ( data.content, rows, 20 );
        }
            contentArea.setLineWrap ( true );
  if ( data != null && ! data.fromID.equals ( currentUser.id ) )
                contentArea.setEditable ( false );
            contentArea.addCaretListener ( this );
        add ( contentArea, BorderLayout.CENTER );
    }


    public boolean isModified()
```

```
        {
            return modified;
        }


        public ValuePair getLogEntryPair()
        {
            return logEntryPair;
        }


        public void caretUpdate ( CaretEvent event )
        {
            String newContent = contentArea.getText();
            if ( ! modified )
            {
                LogEntry oldEntry = (LogEntry)logEntryPair.oldValue;
                if ( oldEntry == null || oldEntry.content == null ||
                ! newContent.equals ( oldEntry.content ) )
                {
                    modified = true;
                    header.setForeground ( Color.magenta );

                    LogEntry newEntry = new LogEntry();
                    newEntry.fromID = currentUser.id;
                    newEntry.content = newContent;
                    logEntryPair.newValue = newEntry;
                }
            }
            else
            {
                ( (LogEntry)logEntryPair.newValue ).content =
                newContent;
            }
        }


        public Dimension getMaximumSize()
        {
            Dimension size = super.getMaximumSize();
            size.height = super.getPreferredSize().height;
            return size;
        }
    }
}
```

```
/************* Mailer.java ***************/
package iceberg.advisor;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.beans.*;
import com.sun.java.swing.*;
import com.sun.java.swing.event.*;
import com.sun.java.swing.border.*;
import sun.net.smtp.*;
import javax.infobus.*;
import iceberg.util.*;
import iceberg.guiUtility.*;

public class Mailer extends IcebergJFrame
implements ActionListener, ItemListener, Runnable
{
    private JTextField  toField;
    private JTextField  ccField;
    private JComboBox   subjectView;
    private JTextArea   messageArea;
    private Vector      eMessageList;
    private int         eMessageListSize = 0;
    private String      fromEAddress;
    private Vector      personList;
    private IcebergInfo params;
    private Pin         pinBox;

    public Mailer()
    {
        super();

        params =
        (IcebergInfo)getDataValue ( "PARAMETERS", null, false );
        if ( params == null )
        {
System.err.println ( "Can not find system parameters" );
            params = new IcebergInfo ( new Hashtable ( 2 ) );
        }

        setContentPane( createMainPane() );
        pack();

        setSize
        (
            params.getInt( "MAILER_WIDTH" ),
            params.getInt( "MAILER_HEIGHT" )
        );
        setLocation
        (
            params.getInt( "MAILER_XPOSITION" ),
            params.getInt( "MAILER_YPOSITION" )
        );
    /*
    addWindowListener( new WindowAdapter() {
public void windowClosing( WindowEvent event ) {
  saveOnClose();
}
     });
    */
        Person currentUser =
        (Person)getDataValue ( "CURRENT_USER", null, false );
        if ( currentUser == null || currentUser.eAddress == null )
        {
System.err.println ( "Can not get Current User's E-mail Address" );
            setFromEAddress ( null );
        }
        else
            setFromEAddress ( currentUser.eAddress );

setEMessageList((Vector)getDataValue("EMESSAGE_LIST", null, true ) );
    }


    private JPanel createMainPane()
    {
        JPanel mainPane = new JPanel ( new BorderLayout(), true );
        mainPane.setBorder ( BorderFactory.createEtchedBorder() );

        mainPane.add ( createToolBar(), BorderLayout.NORTH );

            JPanel contentPane =
            new JPanel ( new BorderLayout ( 5, 5 )  );
contentPane.setBorder(BorderFactory.createEmptyBorder(5, 15, 5, 15));

            contentPane.add (createHeaderPane(), BorderLayout.NORTH);

                messageArea = new JTextArea( 15, 50 );
                messageArea.setLineWrap(true);
                JScrollPane textScroller =
                new JScrollPane( messageArea,
                JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED );

            contentPane.add( textScroller, BorderLayout.CENTER );

        mainPane.add ( contentPane, BorderLayout.CENTER );

           String[] cmd = { "Send", "Cancel" };
           JButtonPanel buttonBox = new JButtonPanel ( cmd );
           buttonBox.addActionListener ( this );

        mainPane.add ( buttonBox, BorderLayout.SOUTH );

        return mainPane;
    }


    private JToolBar createToolBar()
    {
        JToolBar toolBar = new JToolBar();
        pinBox = new Pin();
        toolBar.add ( pinBox );
        return toolBar;
    }


    private JPanel createHeaderPane()
    {
        JPanel header = new JPanel( new BorderLayout() );

            JPanel p = new JPanel( new LabeledPairLayout( 5, 10 ) );

            toField = new JTextField(25);
            p.add ( new JLabel ("To: ", JLabel.RIGHT ), "label" );
            p.add ( toField, "field" );

            ccField = new JTextField(25);
            p.add ( new JLabel ("cc: ", JLabel.RIGHT ), "label" );
            p.add ( ccField, "field" );

            subjectView = new JComboBox();
            subjectView.setEditable ( true );
            subjectView.addItemListener ( this );
            p.add (new JLabel ("Subject:", JLabel.RIGHT ), "label" );
            p.add ( subjectView, "field");

        header.add ( p, BorderLayout.CENTER );
```

```
            String[] cmd = { "Compose", "Save", "Delete" };
            JButtonPanel buttonBox = new JButtonPanel ( cmd );
            buttonBox.addActionListener ( this );

        header.add ( buttonBox, BorderLayout.SOUTH );

        return header;
    }


    public void itemStateChanged ( ItemEvent event )
    {
        if( event.getStateChange() == ItemEvent.SELECTED )
        {
            int index = subjectView.getSelectedIndex();
            if ( index >= 0 && index < eMessageListSize )
            {
                EMessage em =
                (EMessage)eMessageList.elementAt( index );
        messageArea.setText( em.content == null ? "" : em.content );
            }
        }
    }


    public void actionPerformed ( ActionEvent event )
    {
        String arg = event.getActionCommand();
        if ( arg.equals( "Send" ) ) {
            SwingUtilities.invokeLater( this );

        if ( pinBox.getState() == Pin.UP )
processWindowEvent(new WindowEvent(this,WindowEvent.WINDOW_CLOSING));

        else if ( arg.equals( "Compose" ) )
        {
            subjectView.setSelectedIndex( eMessageListSize );
            messageArea.setText( "" );
        }
        else if( arg.equals( "Save" ) )
            saveEMessage();
        else if( arg.equals( "Delete" ) )
            deleteEMessage();
        else if( arg.equals( "Cancel" ) )
processWindowEvent(new WindowEvent(this,WindowEvent.WINDOW_CLOSING));
        else
   System.err.println( "Mailer unknown event: " + event.toString() );
```

```
    }


    public void run()
    {
        try
        {
            String receiver = toField.getText();
            if ( receiver.equals( "" ) )
               throw new DataProcessException ( "Receiver Unknown" );

            String subject = (String)subjectView.getSelectedItem();
            String content = (String)messageArea.getText();
if( subject == null || subject.equals( "" ) || content.equals( "" ) )
throw new DataProcessException("Messge subject or content is absent");

  // Connect to mail system use sun.java.smtp package
            SmtpClient sc = new SmtpClient( "homer.njit.edu" );
            sc.from( fromEAddress );
            sc.to( receiver );
            PrintStream ps = sc.startMessage();
            ps.println( "Subject: " + subject );
            ps.println();
            ps.println( content );
            sc.closeServer();

            JOptionPane.showMessageDialog
            (
                this,
                "Email Message has been sent.",
                "Notice",
                JOptionPane.INFORMATION_MESSAGE
            );
        }
        catch( IOException e )
        {
            e.printStackTrace();
            JOptionPane.showMessageDialog
            (
                this,
                e.toString(),
                "Mail Error",
                JOptionPane.ERROR_MESSAGE
            );
        }
        catch ( DataProcessException e )
        {
            JOptionPane.showMessageDialog
            (
                this,
                e.toString(),
                "Error",
                JOptionPane.ERROR_MESSAGE
            );
        }
    }


    protected void saveEMessage()
    {
        ValuePair change = getChangedValue();
        if ( change == null )
        {
            JOptionPane.showMessageDialog
            (
                this,
                "No Email Message to Save",
                "Error",
                JOptionPane.ERROR_MESSAGE
            );
        }
        else
            fireDataItemAvailable ( "EMESSAGE", change, null );
    }


    protected void deleteEMessage()
    {
        int emIndex = subjectView.getSelectedIndex();
        if ( emIndex != -1 && emIndex == eMessageListSize )
        {
            EMessage oem =
            (EMessage)eMessageList.elementAt( emIndex );
fireDataItemAvailable( "EMESSAGE",new ValuePair(oem, null ), null );
        }
    }


    protected ValuePair getChangedValue()
    {
        int emIndex = subjectView.getSelectedIndex();
        ValuePair vp = null;
        if ( emIndex == -1 )
        {
          String subject = (String)subjectView.getSelectedItem();
          String content = messageArea.getText();
          if ( ! subject.equals ( "" ) && ! content.equals ( "" ) )
    vp = new ValuePair ( null, new EMessage ( subject, content ) );
      }
      else if ( emIndex != eMessageListSize )
      {
          EMessage oem =
          (EMessage)eMessageList.elementAt ( emIndex );
          String content = messageArea.getText();
          if( ! content.equals ( oem.content ) )
 vp = new ValuePair ( oem, new EMessage ( oem.subject, content ) );
      }
      return vp;
  }


 public void cleanup() {
 /*
// Save Mailer Window's dimension and position
   Dimension size = getSize();
   if( width != size.width )
     Iceberg.setInt( "MAILER_WIDTH", size.width );
   if( height != size.height )
     Iceberg.setInt( "MAILER_HEIGHT", size.height );
   if( isShowing() ) {
     Point origion = getLocationOnScreen();
     if( xPos != origion.x )
       Iceberg.setInt( "MAILER_XPOSITION", origion.x );
     if( yPos != origion.y )
       Iceberg.setInt( "MAILER_YPOSITION", origion.y );
     saveOnClose();
   }
   */
 }


 protected void saveOnClose() {
 }

 /*------------- Methods of Bean Properties ---------------*/

   public String getToEAddress()
   {
       String e = toField.getText();
       if ( e == null )
           return null;
    else
        return e;
}


public void setToEAddress ( String eAddress )
{
    if ( eAddress == null )
        toField.setText ( "" );
    else
        toField.setText ( eAddress );
}


public String getFromEAddress()
{
    return fromEAddress;
}


public void setFromEAddress ( String eAddress )
{
    fromEAddress = eAddress;
}


public Vector getEMessageList()
{
    return eMessageList;
}


public void setEMessageList ( Vector e )
{
    eMessageList = e;
    messageArea.setText ( "" );
    subjectView.removeAllItems();
    if( eMessageList == null )
    {
        eMessageListSize = 0;
    }
    else
    {
        eMessageListSize = eMessageList.size();
        for ( int i=0; i<eMessageListSize; i++ )
          subjectView.addItem (
           ( (EMessage)eMessageList.elementAt(i) ).subject );
        }

        subjectView.addItem( "" );
        subjectView.setSelectedIndex( eMessageListSize );
    }


/*------------- Methods of IcebergJFrame ----------------*/

 /**
  * Remove itself from this InfoBus Object
  */
  protected void removeInfoBusRoles ( InfoBus oldBus )
  {
      oldBus.removeDataProducer ( this );
      oldBus.removeDataConsumer ( this );
  }


 /**
  * Add itself to this InfoBus Object
  */
  protected void addInfoBusRoles ( InfoBus newBus )
  {
      newBus.addDataProducer ( this );
      newBus.addDataConsumer ( this );
  }

/*------------- Methods of InfoBusDataProducer ----------------*/

// Use default implement


/*------------- Methods of InfoBusDataConsumer ----------------*/

 /**
  * Being noticed that a dataItem is available
  * @param event a InfoBusItemAvailableEvent Object
  */
  public void dataItemAvailable ( InfoBusItemAvailableEvent event )
  {
      String itemName = event.getDataItemName();
      if ( itemName.equals ( "EMESSAGE_LIST" ) )
      {
          setEMessageList ( (Vector)getDataValue ( event, true ) );
      }
      else if ( itemName.equals ( "PERSON" ) )
```

```
        {
            Person p = (Person)getDataValue ( event, true );
            if ( p == null )
                setToEAddress ( null );
            else
                setToEAddress ( p.eAddress );
        }
    }



    /**
     * Being noticed that a dataItem is revoked
     * @param event a InfoBusItemRevokedEvent Object
     */
    public void dataItemRevoked ( InfoBusItemRevokedEvent event )
    {
        String itemName = event.getDataItemName();
        if ( itemName.equals ( "EMESSAGE_LIST" ) )
        {
            setEMessageList ( null );
            fireDataItemRevoked ( "EMESSAGE" );
        }
        else if ( itemName.equals ( "PERSON" ) )
        {
            setToEAddress ( null );
        }
    }


   /*------------- Methods of DataItemChangeListener --------------*/

    /**
     * Being noticed that a dataItem has changed
     * @param event contain the changed information
     */
public void dataItemValueChanged ( DataItemValueChangedEvent event )
    {
        String propertyName =
        ( String ) event.getProperty ( "NAME" );
        if ( propertyName.equals ( "EMESSAGE_LIST" ) )
        {
            setEMessageList ( (Vector)getDataValue ( event ) );
        }
        else if ( propertyName.equals ( "PERSON" ) )
        {
            Person p = (Person)getDataValue ( event );
            if ( p == null )
                setToEAddress ( null );
            else
                setToEAddress ( p.eAddress );
        }
    }

}
```

```
/************* PersonalInfoPane.java ***************/
package iceberg.advisor;

import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.io.*;
import java.util.*;
import com.sun.java.swing.*;
import com.sun.java.swing.table.*;
import com.sun.java.swing.event.*;
import com.sun.java.swing.border.*;
import javax.infobus.*;
import iceberg.util.*;
import iceberg.guiUtility.*;
import iceberg.Iceberg;

class PersonalInfoPane extends IcebergJPanel
{

    private String[]                infoTypes;
    private IcebergTextField[]      infoFields;
    private Person                  personData;
    private Hashtable               personalInfo;
    private IcebergInfo             params;

    PersonalInfoPane()
    {
        super();

        setLayout ( new LabeledPairLayout( 30, 5 ) );
        setBorder
        (
            BorderFactory.createTitledBorder
            (
                BorderFactory.createEtchedBorder (),
                "Personal Information"
            )
        );

        params =
        (IcebergInfo)getDataValue ( "PARAMETERS", null, false );
        if ( params == null )
        {
System.err.println ( "Can not find system parameters" );
            params = new IcebergInfo ( new Hashtable ( 2 ) );
        }
    infoTypes = params.getStrings ( "PERSONAL_INFO_TYPE" );
    infoFields = new IcebergTextField [ infoTypes.length ];
    String label;

    for ( int i=0; i<infoTypes.length; i++ )
    {
        label = params.getString ( infoTypes[i] + "_LABEL" );
     add ( new JLabel ( label, SwingConstants.RIGHT ), "label" );

        infoFields[i] = new IcebergTextField();
        add ( infoFields[i], "field" );
    }
}


public void cleanup()
{
}


/**
 * return a new personalInfo which contain the information
 * that has been modified
 * @return a Hashtable contains the changed field (name, value)
 */
public Hashtable getChange()
{
    Hashtable newPersonalInfo = null;
    for ( int i=2; i<infoTypes.length; i++ )
    {
        if ( infoFields[i].isModified() )
        {
            if ( newPersonalInfo == null )
                newPersonalInfo = new Hashtable ( 10, 1.0f );

   newPersonalInfo.put ( infoTypes[i], infoFields[i].getText() );
        }
    }
    return newPersonalInfo;
}


/*----------- Methods of Bean Properties ----------------*/

public Person getPersonData()
```

```
{
    return personData;
}


public void setPersonData ( Person p )
{
    personData = p;
    if ( personData == null )
    {
        for ( int i=0; i<infoFields.length; i++ )
            infoFields[i].setText ( "" );
    }
    else
    {
        Hashtable personalTable =
        IcebergUtility.getFieldMap ( p );
        personalTable.put ( "name", p.getName() );
        Object value;
        for ( int i=0; i<infoTypes.length; i++ )
        {
            value = personalTable.get ( infoTypes[i] );
            if ( value == null )
                infoFields[i].setText ( "" );
            else if ( value instanceof String )
                infoFields[i].setText ( (String)value );
            else
                infoFields[i].setText ( value.toString() );
        }
    }
}


protected void saveChange()
{
    Hashtable newPersonalInfo = null;
    String value;
    for ( int i=2; i<infoTypes.length; i++ )
    {
        if ( infoFields[i].isModified() )
        {
            if ( newPersonalInfo == null )
                newPersonalInfo = new Hashtable ( 10, 1.0f );

            value = infoFields[i].getText();
            newPersonalInfo.put ( infoTypes[i], value );
        }
        }
        if ( newPersonalInfo != null )
        newPersonalInfo.remove ( "bDate" );
        if ( newPersonalInfo != null )
        {
            Person p = new Person();
            IcebergUtility.setFieldMap ( p, newPersonalInfo );
            p.id = personData.id;
            fireDataItemAvailable ( "NEW_PERSONAL_INFO", p, null );
        }

    }



/*-------------- Methods of IcebergJFrame -----------------*/

 /**
  * Remove itself from this InfoBus Object
  */
  protected void removeInfoBusRoles ( InfoBus oldBus )
  {
      oldBus.removeDataProducer ( this );
      oldBus.removeDataConsumer ( this );
  }


 /**
  * Add itself to this InfoBus Object
  */
  protected void addInfoBusRoles ( InfoBus newBus )
  {
      newBus.addDataProducer ( this );
      newBus.addDataConsumer ( this );
  }


/*-------------- Methods of InfoBusDataConsumer -----------------*/

 /**
  * Being noticed that a dataItem is available
  * @param event a InfoBusItemAvailableEvent Object
  */
  public void dataItemAvailable(InfoBusItemAvailableEvent event )
  {
      String itemName = event.getDataItemName();
      if ( itemName.equals ( "PERSON" ) )
```

```
        {
            setPersonData ( (Person)getDataValue ( event, true ) );
        }
        if ( itemName.equals ( "SAVE" ) )
        {
            saveChange();
        }

    }



   /**
    * Being noticed that a dataItem is revoked
    * @param event a InfoBusItemRevokedEvent Object
    */
    public void dataItemRevoked ( InfoBusItemRevokedEvent event )
    {
        String itemName = event.getDataItemName();
        if ( itemName.equals ( "PERSON" ) )
        {
            setPersonData ( null );
            fireDataItemRevoked ( "NEW_PERSONAL_INFO" );
        }

    }


  /*-------------- Methods of DataItemChangeListener -------------*/

   /**
    * Being noticed that a dataItem has changed
    * @param event contain the changed information
    */
public void dataItemValueChanged ( DataItemValueChangedEvent event )
    {
        String propertyName =
        ( String ) event.getProperty ( "NAME" );
        if ( propertyName.equals ( "PERSON" ) )
        {
            setPersonData ( (Person)getDataValue ( event ) );
        }
    }


  /************** IcebergTextField.java ***************/
```

```
    class IcebergTextField extends JTextField
    implements CaretListener
    {
        private boolean modified = false;

        IcebergTextField()
        {
            super();
            addCaretListener ( this );
        }

        public void caretUpdate ( CaretEvent event )
        {
            modified = true;
        }

        public boolean isModified()
        {
            return modified;
        }
    }
}
```

```
/************ PersonListPane.java ***************/
package iceberg.advisor;

import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.util.*;
import com.sun.java.swing.*;
import com.sun.java.swing.event.*;
import javax.infobus.*;
import iceberg.util.*;
import iceberg.guiUtility.*;

public class PersonListPane extends IcebergJPanel
implements ListSelectionListener
{
    private Vector              personList;
    private int                 currentSelection = -1;
    private IcebergInfo         params;
    private JList               personView;
    private Icon                smileIcon, cryIcon;

    public PersonListPane()
    {
        super();
        setLayout( new BorderLayout() );
        setBorder ( BorderFactory.createEtchedBorder() );

        params =
        (IcebergInfo)getDataValue ( "PARAMETERS", null, false );
        if ( params == null )
        {
System.err.println ( "Can not find system parameters" );
            params = new IcebergInfo ( new Hashtable ( 2 ) );
        }

        smileIcon = ImageIconFactory.getImageIcon (
        params.getString ( "SMILE_IMAGE" ) );
        cryIcon = ImageIconFactory.getImageIcon (
        params.getString ( "CRY_IMAGE" ) );

        personView = new JList();
        personView.setModel ( new DefaultListModel() );
        personView.setCellRenderer ( new PersonCellRender() );
        personView.setFixedCellWidth
        (
            params.getInt ( "PERSON_LIST_WIDTH" )
```

```
      );
      personView.addListSelectionListener( this );

      JScrollPane pane = new JScrollPane ();
      pane.setBorder ( BorderFactory.createLoweredBevelBorder() );
      pane.getViewport().add( personView );

      add ( pane, BorderLayout.CENTER );
  }


  public void cleanup() {

}


/*------------- Methods of ListSelection Listener -------------*/

  public void valueChanged ( ListSelectionEvent event )
  {
      if ( ! event.getValueIsAdjusting() )
      {
          int newSelection = event.getFirstIndex();
      if ( newSelection < 0  ||
      newSelection == currentSelection )
          return;
      else
      {
                  Person p =
                  (Person)personList.elementAt ( newSelection );
              fireDataItemAvailable ( "PERSON_ID", p.id , null );
          currentSelection = newSelection;
      }
      }
  }


/*------------- Methods of bean proerties ---------------*/

  public Vector getPersonList()
  {
      return personList;
  }


  public void setPersonList( Vector pList )
  {
        personList = pList;
        currentSelection = -1;
        DefaultListModel personData =
        (DefaultListModel)personView.getModel();
        personData.removeAllElements();

        if ( personList == null || personList.size() == 0 )
        {
            fireDataItemAvailable ( "PERSON", null, null );
        }
        else
        {
            String name;
            for ( int i=0; i<personList.size(); i++ )
            {
                name = ((Person)personList.elementAt(i) ).getName();
                personData.addElement( new PersonListEntry( name ) );
            }

            personView.setSelectedIndex( 0 );
        }
    }


/*-------------- Methods of IcebergJPanel -----------------*/

 /**
  * Remove itself from this InfoBus Object
  */
  protected void removeInfoBusRoles ( InfoBus oldBus )
  {
      oldBus.removeDataProducer ( this );
      oldBus.removeDataConsumer ( this );
  }


 /**
  * Add itself to this InfoBus Object
  */
  protected void addInfoBusRoles ( InfoBus newBus )
  {
      newBus.addDataProducer ( this );
      newBus.addDataConsumer ( this );
  }

/*-------------- Methods of InfoBusDataProducer -----------------*/
```

```
// Use default implement


/*-------------- Methods of InfoBusDataConsumer -----------------*/

 /**
  * Being noticed that a dataItem is available
  * @param event a InfoBusItemAvailableEvent Object
  */
  public void dataItemAvailable ( InfoBusItemAvailableEvent event )
  {
      String itemName = event.getDataItemName();
      if ( itemName.equals ( "PERSON_LIST" ) )
      {
          setPersonList ( (Vector)getDataValue ( event, true ) );
      }
  }



 /**
  * Being noticed that a dataItem is revoked
  * @param event a InfoBusItemRevokedEvent Object
  */
  public void dataItemRevoked ( InfoBusItemRevokedEvent event )
  {
      String itemName = event.getDataItemName();
      if ( itemName.equals ( "PERSON_LIST" ) )
      {
          setPersonList ( null );
          dataItemTable.remove ( "PERSON" );
          getInfoBus().fireItemRevoked ( "PERSON", this );
      }
  }


/*--------------- Methods of DataItemChangeListener -------------*/

 /**
  * Being noticed that a dataItem has changed
  * @param event contain the changed information
  */
public void dataItemValueChanged( DataItemValueChangedEvent event )
  {
      String propertyName =
      ( String ) event.getProperty ( "NAME" );
      if ( propertyName.equals ( "PERSON_LIST" ) )
```

```
    {
        setPersonList ( (Vector)getDataValue ( event ) );
    }
}


/************ PersonListEntry.java **************/

class PersonListEntry
{
    String   name;
    boolean  hasBeenSelected;

    PersonListEntry(String n)
    {
        name = n;
        hasBeenSelected = false;
    }
}


/************ PersonCellRender.java **************/

class PersonCellRender extends JLabel
implements ListCellRenderer
{

    PersonCellRender()
    {
        setOpaque ( true );
    }


    public Component getListCellRendererComponent ( JList list,
                                                    Object value,
                                                    int index,
                                                    boolean isSelected,
                                                    boolean cellHasFocus )
    {
        PersonListEntry sle = ( PersonListEntry ) value;

        sle.hasBeenSelected |= isSelected;
        if( sle.hasBeenSelected )
            setIcon ( smileIcon );
        else
            setIcon ( cryIcon );
            setText ( sle.name );

            if( isSelected )
            {
                setBackground ( SystemColor.textHighlight );
                setForeground ( SystemColor.textHighlightText );
            }
            else
            {
                setBackground ( list.getBackground() );
   setForeground ( sle.hasBeenSelected? Color.magenta : Color.blue );
            }

            return this;
        }

/*
      public Dimension getPreferredSize() {
        Dimension dim = super.getPreferredSize();
        dim.width += 16;  // widen
        return dim;
      }
*/
    }

}
```

```
/********** RecordGenerator.java **************/
package iceberg.advisor;

import java.util.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import javax.infobus.*;
import iceberg.util.*;
import iceberg.guiUtility.*;
import iceberg.Iceberg;

class RecordGenerator extends IcebergJFrame
implements ActionListener
{

// Iceberg Attributes

    private TextField       idField;
    private TextField       fNameField;
    private TextField       mNameField;
    private TextField       lNameField;
    private JComboBox       programView;
    private Vector          programList;
    private IcebergInfo     params;

    RecordGenerator ()
    {
        super();

        params =
        (IcebergInfo)getDataValue ( "PARAMETERS", null, false );
        if ( params == null )
        {
System.err.println ( "Can not find system parameters" );
            params = new IcebergInfo ( new Hashtable ( 2 ) );
        }

        setContentPane( createMainPane() );

        pack();

        setSize
        (
            params.getInt( "RECORD_GENERATOR_WIDTH" ),
            params.getInt( "RECORD_GENERATOR_HEIGHT" )
    );
    setLocation
    (
        params.getInt( "RECORD_GENERATOR_XPOSITION" ),
        params.getInt( "RECORD_GENERATOR_YPOSITION" )
    );

    ImmediateAccess item = (ImmediateAccess)
    ( getInfoBus().findDataItem("PROGRAM_LIST", null, this ) );
    setProgramList ( ( Vector ) item.getValueAsObject() );
}


protected JPanel createMainPane()
{
    JPanel mainPane = new JPanel ( new BorderLayout ( 10, 10 ) );

        JPanel pane =new JPanel(new LabeledPairLayout( 5, 10 ) );
        pane.setBorder ( BorderFactory.createEmptyBorder(15, 15, 5, 15) );

        pane.add ( new JLabel ( "First Name" ), "label" );
        fNameField = new TextField ( 30 );
        pane.add ( fNameField, "field" );

        pane.add ( new JLabel ( "Middle Name." ), "label" );
        mNameField = new TextField ( 1 );
        pane.add ( mNameField, "field" );

        pane.add ( new JLabel( "Last Name" ), "label" );
        lNameField = new TextField ( 30 );
        pane.add ( lNameField, "field" );

        pane.add ( new JLabel ( "SSN#/SID#" ), "label" );
        idField = new TextField ( 30 );
        pane.add ( idField, "field" );

        pane.add ( new JLabel ( "Program" ), "label" );
        programView = new JComboBox ();
        programView.setBackground ( Color.white );
        pane.add ( programView, "field" );

    mainPane.add ( pane, BorderLayout.NORTH );

        String[] cmd = { "Start", "Clear", "Cancel" };
        JButtonPanel buttonBox = new JButtonPanel ( cmd );
        buttonBox.addActionListener ( this );
        mainPane.add ( buttonBox, BorderLayout.SOUTH );

        return mainPane;
    }


    public void actionPerformed ( ActionEvent event )
    {
        String arg = event.getActionCommand();
        if ( arg.equals ( "Start" ) )
        {
            createRecord();
        }
        else if( arg.equals ( "Clear" ) )
        {
            clear();
        }
        else if( arg.equals ( "Cancel" ) )
processWindowEvent(new WindowEvent(this,WindowEvent.WINDOW_CLOSING));
        else
    System.out.println( "actionPerformed:unknown action " + arg );
    }


    protected void clear()
    {
        idField.setText( "" );
        fNameField.setText( "" );
        mNameField.setText( "" );
        lNameField.setText( "" );
        programView.setSelectedIndex ( 0 );
    }


    protected void createRecord()
    {
        Student s = new Student();

        s. id = idField.getText();
        s.fName = fNameField.getText();
        s.mName = mNameField.getText();
        s.lName = lNameField.getText();

        if ( s.id.equals( "" ) || s.fName.equals( "" ) ||
         s.lName.equals( "" ) )
        {
            JOptionPane.showMessageDialog
    (
        this,
        "Missing information",
        "Error",
        JOptionPane.ERROR_MESSAGE
    );
    return;
}

if ( !isID ( s.id ) )
{
    JOptionPane.showMessageDialog
    (
        this,
        "Incorrect student ID",
        "Error",
        JOptionPane.ERROR_MESSAGE
    );
    return;
}

if ( !isName ( s.mName ) )
{
    JOptionPane.showMessageDialog
    (
        this,
        "The middle name has none letter character",
        "Error",
        JOptionPane.ERROR_MESSAGE
    );
    return;
}

if ( !isName ( s.fName ) )
{
    JOptionPane.showMessageDialog
    (
        this,
        "The first name has none letter character",
        "Error",
        JOptionPane.ERROR_MESSAGE
    );
    return;
}

if ( !isName( s.lName ) )
{
```

```
            JOptionPane.showMessageDialog
            (
                this,
                "The last name has none letter character",
                "Error",
                JOptionPane.ERROR_MESSAGE
            );
            return;
        }

        int pno = programView.getSelectedIndex();
        Program p = ( Program ) programList.elementAt ( pno );
        s.pno = p.pno;

        fireDataItemAvailable ( "NEW_STUDENT", s, null );

processWindowEvent(new WindowEvent(this,WindowEvent.WINDOW_CLOSING));
        clear();
    }


    protected boolean isID ( String id )
    {
        if ( id.length() != 9 )
        {
            if ( id.length() == 11 &&
            ( ( id.charAt(3) == '-' && id.charAt(6) == '-' ) ||
        (id.charAt(3) == ' ' && id.charAt(6) == ' ') ) )
id = id.substring(0,3) + id.substring(4,6) + id.substring(7);
            else
                return false;
        }

        for ( int i=0; i<9; i++ )
        {
            if ( !Character.isDigit( id.charAt(i) ) )
                return false;
        }

        return true;
    }


    protected boolean isName ( String name )
    {
        for ( int i=0; i<name.length(); i++ )
        {
            if( !Character.isLetter ( name.charAt(i) ) )
            return false;
        }

        return true;
    }


/*-------------- Methods of bean properties ----------------*/

    protected Vector getProgramList()
    {
        return programList;
    }


    protected void setProgramList ( Vector p )
    {
        programList = p;
        programView.removeAllItems();
        if ( p != null )
        {
            for ( int i=0; i<p.size(); i++ )
            {
          programView.addItem ( ( (Program)p.elementAt(i) ).pName );
            }
        }
    }

/*------------- Methods of IcebergJFrame -----------------*/

   /**
    * Remove itself from this InfoBus Object
    */
    protected void removeInfoBusRoles ( InfoBus oldBus )
    {
        oldBus.removeDataProducer ( this );
        oldBus.removeDataConsumer ( this );
    }


   /**
    * Add itself to this InfoBus Object
    */
    protected void addInfoBusRoles ( InfoBus newBus )
    {
        newBus.addDataProducer ( this );
        newBus.addDataConsumer ( this );
    }

  /*-------------- Methods of InfoBusDataProducer -----------------*/

  // Use default implement


  /*-------------- Methods of InfoBusDataConsumer -----------------*/

    /**
     * Being noticed that a dataItem is available
     * @param event a InfoBusItemAvailableEvent Object
     */
    public void dataItemAvailable ( InfoBusItemAvailableEvent event )
    {
        String itemName = event.getDataItemName();
        if ( itemName.equals ( "PROGRAM_LIST" ) )
        {
            setProgramList ( (Vector)getDataValue ( event, true ) );
        }
    }



    /**
     * Being noticed that a dataItem is revoked
     * @param event a InfoBusItemRevokedEvent Object
     */
    public void dataItemRevoked ( InfoBusItemRevokedEvent event )
    {
        String itemName = event.getDataItemName();
        if ( itemName.equals ( "PROGRAM_LIST" ) )
        {
            setProgramList ( null );
            fireDataItemRevoked ( itemName );
        }
    }


   /*------------- Methods of DataItemChangeListener -------------*/

     /**
      * Being noticed that a dataItem has changed
      * @param event contain the changed information
      */
public void dataItemValueChanged ( DataItemValueChangedEvent event )
```

```
    {
        String propertyName = ( String ) event.getProperty("NAME");
        if ( propertyName.equals ( "LOG_LIST" ) )
        {
            setProgramList ( (Vector)getDataValue ( event ) );
        }
    }

}
```

```
/************ SimpleTableDataModel.java ***********/
package iceberg.advisor;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import com.sun.java.swing.*;
import com.sun.java.swing.border.*;
import iceberg.util.*;
import iceberg.guiUtility.*;
import javax.infobus.*;
import iceberg.Iceberg;

public class Searcher extends IcebergJFrame
implements ActionListener
{
    private JTextField      input[];
    private JComboBox       choiceList[];
    private JSlider         numSlider;
    private Pin             pinBox;
    private IcebergInfo     params;

    public Searcher()
    {
        super();

        params =
        (IcebergInfo)getDataValue ( "PARAMETERS", null, false );
        if ( params == null )
        {
System.err.println ( "Can not find system parameters" );
            params = new IcebergInfo ( new Hashtable ( 2 ) );
        }

        setContentPane( createMainPane() );

        pack();

        setSize
        (
            params.getInt( "SEARCHER_WIDTH" ),
            params.getInt( "SEARCHER_HEIGHT" )
        );
        setLocation
        (
            params.getInt( "SEARCHER_XPOSITION" ),
            params.getInt( "SEARCHER_YPOSITION" )
        );
    }


    protected JPanel createMainPane()
    {
        input = new JTextField[3];
        choiceList = new JComboBox[3];
        JPanel mainPane =
        new JPanel ( new BorderLayout(10, 10), true );
        mainPane.setBorder( BorderFactory.createEtchedBorder() );

        mainPane.add( createToolBar(), BorderLayout.NORTH );

            JPanel content = new JPanel();
       content.setLayout( new BoxLayout(content, BoxLayout.Y_AXIS) );
  content.setBorder( BorderFactory.createEmptyBorder(0, 15, 0, 15) );

         content.add( createContent(0, 18, "Search by SSN#/SID#",
            new String[] { "Match Exactly", "Find Similar"}));

            content.add( createContent(1, 18, "Search by Name",
    new String[] { "Match Exactly", "Find Similar", "Sound Like" }));

     content.add( createContent(2, 18, "Search by Email Address",
            new String[] { "Match Exactly", "Find Similar" }));

        mainPane.add ( content, BorderLayout.CENTER );

            String[] cmd = { "Search", "Reset", "Cancel" };
            JButtonPanel buttonBox = new JButtonPanel ( cmd );
            buttonBox.addActionListener ( this );

        mainPane.add ( buttonBox, BorderLayout.SOUTH );

        return mainPane;
    }


    protected JToolBar createToolBar()
    {
        JToolBar toolBar = new JToolBar();

            pinBox = new Pin();

        toolBar.add( pinBox );
    toolBar.add( Box.createHorizontalStrut(10) );

        numSlider = new JSlider( JSlider.HORIZONTAL, 0, 30, 5 );
        numSlider.setPaintLabels( true );
  numSlider.setLabelTable( numSlider.createStandardLabels( 5 ) );
    numSlider.setToolTipText( "Number of Students returned" );

    toolBar.add( numSlider );

    toolBar.add( Box.createHorizontalStrut(10) );
    return toolBar;
}


protected Container createContent ( int index,
                                    int textLen,
                                    String label,
                                    String[] choice )
{
    JPanel panel =
    new JPanel ( new FlowLayout(FlowLayout.LEFT,20,5) );
    panel.setBorder
    (
        BorderFactory.createTitledBorder
        (
            BorderFactory.createLineBorder ( Color.black ),
            label
        )
    );

        choiceList[index] = new JComboBox( choice )
        {
            public Dimension getPreferredSize()
            {
                Dimension size = super.getPreferredSize();
                size.width += 30;
                return size;
            }
        };
        choiceList[index].setEditable( false );
        choiceList[index].setSelectedIndex( 0 );

     panel.add( choiceList[index] );

        input[index] = new JTextField( textLen );
        input[index].setActionCommand( "Search" );
        input[index].addActionListener( this );
        panel.add( input[index] );
        return panel;
    }


    public void actionPerformed ( ActionEvent event )
    {
        String arg = event.getActionCommand();
        if ( arg.equals( "Search" ) )
            processSearchInfo();
        else if ( arg.equals( "Reset" ) )
            reset();
        else if ( arg.equals( "Cancel" ) )
processWindowEvent(new WindowEvent(this,WindowEvent.WINDOW_CLOSING));
        else
System.err.println( "Searcher unknown event: " + event.toString() );
    }


    protected void processSearchInfo() {
      SearchInfo s = new SearchInfo();
      s.id = input[0].getText();
      if( s.id.equals( "" ) )
        s.idChoice = SearchInfo.NO_SEARCH;
      else if( isID( s.id ) )
        s.idChoice = choiceList[0].getSelectedIndex();
      else
        return;
      s.name = input[1].getText();
      if( s.name.equals( "" ) )
        s.nameChoice = SearchInfo.NO_SEARCH;
      else if( isName( s.name ) )
        s.nameChoice = choiceList[1].getSelectedIndex();
      else
        return;
      s.eAddress = input[2].getText();
      if( s.eAddress.equals( "" ) )
        s.eAddressChoice = SearchInfo.NO_SEARCH;
      else if( isEAddress( s.eAddress ) )
        s.eAddressChoice = choiceList[2].getSelectedIndex();
      else
        return;
      if( s.idChoice == SearchInfo.NO_SEARCH &&
          s.nameChoice == SearchInfo.NO_SEARCH &&
          s.eAddressChoice == SearchInfo.NO_SEARCH ) {
          JOptionPane.showMessageDialog(this,
        "No search information present", "Error",
        JOptionPane.ERROR_MESSAGE );
      return;
    }
    s.resultSize = numSlider.getValue();
    fireDataItemAvailable ( "SEARCH_INFO", s, null );

    if( pinBox.getState() == Pin.UP )
        processWindowEvent(
        new WindowEvent( this, WindowEvent.WINDOW_CLOSING ) );
    reset();
  }


  protected void reset()
  {
      for ( int i=0; i<3; i++ )
      {
          input[i].setText( "" );
          choiceList[i].setSelectedIndex( 0 );
          numSlider.setValue( 5 );
      }
      repaint();
  }


protected boolean isID( String id ) {
  if( id.length() != 9 ) {
    if( id.length() == 11 && (
        ( id.charAt(3) == '-' && id.charAt(6) == '-' ) ||
        ( id.charAt(3) == ' ' && id.charAt(6) == ' ' ) ) ) {
      StringBuffer idBuffer =
      new StringBuffer( id.substring(0,3) );
      idBuffer.append( id.substring(4,6) );
      idBuffer.append( id.substring(7) );
      id = idBuffer.toString();
    }
    else {
      JOptionPane.showMessageDialog(
      this, "Incorrect ID format", "Error",
      JOptionPane.ERROR_MESSAGE );
      return false;
    }
  }
  for( int i=0; i<9; i++ ) {
    if( !Character.isDigit( id.charAt(i) ) ) {
      JOptionPane.showMessageDialog( this,
        "Unknown character in ID field", "Error",
        JOptionPane.ERROR_MESSAGE );
      return false;
    }
  }
  return true;
}


protected boolean isName( String name ) {
  StringTokenizer names = new StringTokenizer( name );
  if( names.countTokens() >2 ) {
    JOptionPane.showMessageDialog( this,
    "Searcher can only handle First Name and Last Name",
    "Error",  JOptionPane.ERROR_MESSAGE );
    return false;
  }
  String token;
  while( names.hasMoreTokens() ) {
    token = names.nextToken();
    for( int i=0; i<token.length(); i++ ) {
      if( !Character.isLetter( token.charAt(i) ) ) {
        JOptionPane.showMessageDialog( this,
         "Unknown character in Name field",
         "Error",  JOptionPane.ERROR_MESSAGE );
        return false;
      }
    }
  }
  return true;
}

protected boolean isEAddress( String eAddress ) {
  if( eAddress.indexOf( '@' ) == -1 ) {
    JOptionPane.showMessageDialog( this,
    "No Email host specified in Email Address field",
    "Error",  JOptionPane.ERROR_MESSAGE );
    return false;
  }
  if( !eAddress.endsWith( ".edu" ) )
    eAddress.concat( ".njit.edu" );
  return true;
}

public void cleanup() {
/*
  Dimension size = getSize();
```

```
    if( width != size.width )
      Iceberg.setInt( "SEARCHER_WIDTH", size.width );
    if( height != size.height )
      Iceberg.setInt( "SEARCHER_HEIGHT", size.height );
    if( isShowing() ) {
      Point origion = getLocationOnScreen();
      if( xPos != origion.x )
        Iceberg.setInt( "SEARCHER_XPOSITION", origion.x );
      if( yPos != origion.y )
        Iceberg.setInt( "SEARCHER_YPOSITION", origion.y );
    }
  */
  }


  /*------------- Methods of IcebergJFrame -----------------*/

   /**
    * Remove itself from this InfoBus Object
    */
    protected void removeInfoBusRoles ( InfoBus oldBus )
    {
        oldBus.removeDataProducer ( this );
    }


   /**
    * Add itself to this InfoBus Object
    */
    protected void addInfoBusRoles ( InfoBus newBus )
    {
        newBus.addDataProducer ( this );
    }

}
```

```
/************* Searcher.java *****************/
package iceberg.advisor;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import com.sun.java.swing.*;
import com.sun.java.swing.border.*;
import iceberg.util.*;
import iceberg.guiUtility.*;
import javax.infobus.*;
import iceberg.Iceberg;

public class Searcher extends IcebergJFrame
implements ActionListener
{
    private JTextField      input[];
    private JComboBox       choiceList[];
    private JSlider         numSlider;
    private Pin             pinBox;
    private IcebergInfo     params;

    public Searcher()
    {
        super();

        params =
        (IcebergInfo)getDataValue ( "PARAMETERS", null, false );
        if ( params == null )
        {
System.err.println ( "Can not find system parameters" );
            params = new IcebergInfo ( new Hashtable ( 2 ) );
        }

        setContentPane( createMainPane() );

        pack();

        setSize
        (
            params.getInt( "SEARCHER_WIDTH" ),
            params.getInt( "SEARCHER_HEIGHT" )
        );
        setLocation
        (
            params.getInt( "SEARCHER_XPOSITION" ),
            params.getInt( "SEARCHER_YPOSITION" )
    );
  }


  protected JPanel createMainPane()
  {
      input = new JTextField[3];
      choiceList = new JComboBox[3];
      JPanel mainPane =
      new JPanel ( new BorderLayout(10, 10), true );
      mainPane.setBorder( BorderFactory.createEtchedBorder() );

      mainPane.add( createToolBar(), BorderLayout.NORTH );

          JPanel content = new JPanel();
    content.setLayout( new BoxLayout(content, BoxLayout.Y_AXIS) );
content.setBorder( BorderFactory.createEmptyBorder(0, 15, 0, 15) );

          content.add( createContent(0, 18, "Search by SSN#/SID#",
          new String[] { "Match Exactly", "Find Similar"}));

          content.add( createContent(1, 18, "Search by Name",
 new String[] { "Match Exactly", "Find Similar", "Sound Like" }));

   content.add( createContent(2, 18, "Search by Email Address",
          new String[] { "Match Exactly", "Find Similar" }));

      mainPane.add ( content, BorderLayout.CENTER );

          String[] cmd = { "Search", "Reset", "Cancel" };
          JButtonPanel buttonBox = new JButtonPanel ( cmd );
          buttonBox.addActionListener ( this );

      mainPane.add ( buttonBox, BorderLayout.SOUTH );

      return mainPane;
  }


  protected JToolBar createToolBar()
  {
      JToolBar toolBar = new JToolBar();

          pinBox = new Pin();

      toolBar.add( pinBox );
    toolBar.add( Box.createHorizontalStrut(10) );

        numSlider = new JSlider( JSlider.HORIZONTAL, 0, 30, 5 );
        numSlider.setPaintLabels( true );
  numSlider.setLabelTable( numSlider.createStandardLabels( 5 ) );
      numSlider.setToolTipText( "Number of Students returned" );

    toolBar.add( numSlider );

    toolBar.add( Box.createHorizontalStrut(10) );
    return toolBar;
}


protected Container createContent ( int index,
                                    int textLen,
                                    String label,
                                    String[] choice )
{
    JPanel panel =
    new JPanel ( new FlowLayout(FlowLayout.LEFT,20,5) );
    panel.setBorder
    (
        BorderFactory.createTitledBorder
        (
            BorderFactory.createLineBorder ( Color.black ),
            label
        )
    );

        choiceList[index] = new JComboBox( choice )
        {
            public Dimension getPreferredSize()
            {
                Dimension size = super.getPreferredSize();
                size.width += 30;
                return size;
            }
        };
        choiceList[index].setEditable( false );
        choiceList[index].setSelectedIndex( 0 );

    panel.add( choiceList[index] );

        input[index] = new JTextField( textLen );
        input[index].setActionCommand( "Search" );
        input[index].addActionListener( this );
        panel.add( input[index] );
        return panel;
    }


    public void actionPerformed ( ActionEvent event )
    {
        String arg = event.getActionCommand();
        if ( arg.equals( "Search" ) )
            processSearchInfo();
        else if ( arg.equals( "Reset" ) )
            reset();
        else if ( arg.equals( "Cancel" ) )
            processWindowEvent(
            new WindowEvent( this, WindowEvent.WINDOW_CLOSING ) );
        else
System.err.println( "Searcher unknown event: " + event.toString() );
    }


    protected void processSearchInfo() {
      SearchInfo s = new SearchInfo();
      s.id = input[0].getText();
      if( s.id.equals( "" ) )
        s.idChoice = SearchInfo.NO_SEARCH;
      else if( isID( s.id ) )
        s.idChoice = choiceList[0].getSelectedIndex();
      else
        return;
      s.name = input[1].getText();
      if( s.name.equals( "" ) )
        s.nameChoice = SearchInfo.NO_SEARCH;
      else if( isName( s.name ) )
        s.nameChoice = choiceList[1].getSelectedIndex();
      else
        return;
      s.eAddress = input[2].getText();
      if( s.eAddress.equals( "" ) )
        s.eAddressChoice = SearchInfo.NO_SEARCH;
      else if( isEAddress( s.eAddress ) )
        s.eAddressChoice = choiceList[2].getSelectedIndex();
      else
        return;
      if( s.idChoice == SearchInfo.NO_SEARCH &&
          s.nameChoice == SearchInfo.NO_SEARCH &&
          s.eAddressChoice == SearchInfo.NO_SEARCH ) {
```

```
      JOptionPane.showMessageDialog(this,
      "No search information present", "Error",
       JOptionPane.ERROR_MESSAGE );
      return;
    }
    s.resultSize = numSlider.getValue();
    fireDataItemAvailable ( "SEARCH_INFO", s, null );

    if( pinBox.getState() == Pin.UP )
      processWindowEvent(
      new WindowEvent( this, WindowEvent.WINDOW_CLOSING ) );
    reset();
  }


  protected void reset()
  {
      for ( int i=0; i<3; i++ )
      {
          input[i].setText( "" );
          choiceList[i].setSelectedIndex( 0 );
          numSlider.setValue( 5 );
      }
      repaint();
  }


protected boolean isID( String id ) {
  if( id.length() != 9 ) {
    if( id.length() == 11 && (
        ( id.charAt(3) == '-' && id.charAt(6) == '-' ) ||
        ( id.charAt(3) == ' ' && id.charAt(6) == ' ' ) ) ) {
      StringBuffer idBuffer =
      new StringBuffer( id.substring(0,3) );
      idBuffer.append( id.substring(4,6) );
      idBuffer.append( id.substring(7) );
      id = idBuffer.toString();
    }
    else {
      JOptionPane.showMessageDialog( this,
       "Incorrect ID format", "Error",
       JOptionPane.ERROR_MESSAGE );
      return false;
    }
  }
  for( int i=0; i<9; i++ ) {
    if( !Character.isDigit( id.charAt(i) ) ) {
        JOptionPane.showMessageDialog( this,
        "Unknown character in ID field", "Error",
        JOptionPane.ERROR_MESSAGE );
        return false;
      }
    }
    return true;
  }


  protected boolean isName( String name ) {
    StringTokenizer names = new StringTokenizer( name );
    if( names.countTokens() >2 ) {
      JOptionPane.showMessageDialog( this,
      "Searcher can only handle First Name and Last Name",
      "Error",  JOptionPane.ERROR_MESSAGE );
      return false;
    }
    String token;
    while( names.hasMoreTokens() ) {
      token = names.nextToken();
      for( int i=0; i<token.length(); i++ ) {
        if( !Character.isLetter( token.charAt(i) ) ) {
          JOptionPane.showMessageDialog( this,
          "Unknown character in Name field",
          "Error",  JOptionPane.ERROR_MESSAGE );
          return false;
        }
      }
    }
    return true;
  }

  protected boolean isEAddress( String eAddress ) {
    if( eAddress.indexOf( '@' ) == -1 ) {
      JOptionPane.showMessageDialog( this,
      "No Email host specified in Email Address field",
      "Error",  JOptionPane.ERROR_MESSAGE );
      return false;
    }
    if( !eAddress.endsWith( ".edu" ) )
      eAddress.concat( ".njit.edu" );
    return true;
  }

  public void cleanup() {
  /*
```

```
    Dimension size = getSize();
    if( width != size.width )
      Iceberg.setInt( "SEARCHER_WIDTH", size.width );
    if( height != size.height )
      Iceberg.setInt( "SEARCHER_HEIGHT", size.height );
    if( isShowing() ) {
      Point origion = getLocationOnScreen();
      if( xPos != origion.x )
        Iceberg.setInt( "SEARCHER_XPOSITION", origion.x );
      if( yPos != origion.y )
        Iceberg.setInt( "SEARCHER_YPOSITION", origion.y );
    }
  */
  }


  /*-------------- Methods of IcebergJFrame -----------------*/

   /**
    * Remove itself from this InfoBus Object
    */
    protected void removeInfoBusRoles ( InfoBus oldBus )
    {
        oldBus.removeDataProducer ( this );
    }


   /**
    * Add itself to this InfoBus Object
    */
    protected void addInfoBusRoles ( InfoBus newBus )
    {
        newBus.addDataProducer ( this );
    }

}
```

```
/************* TranscriptInfoPane.java ***************/
package iceberg.advisor;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.beans.*;
import com.sun.java.swing.*;
import com.sun.java.swing.table.*;
import com.sun.java.swing.border.*;
import javax.infobus.*;
import iceberg.util.*;
import iceberg.guiUtility.*;
import iceberg.Iceberg;

class TranscriptInfoPane extends IcebergJPanel
{
    private JTable                    courseTable;
    private SimpleTableDataModel      courseData;
    private IcebergInfo               params;

    TranscriptInfoPane()
    {
        super();
        setLayout( new BorderLayout() );
        setBorder
        (
            BorderFactory.createTitledBorder
            (
            BorderFactory.createLineBorder ( Color.black ),
            "Transcript"
        )
    );

        params =
        (IcebergInfo)getDataValue ( "PARAMETERS", null, false );
        if ( params == null )
        {
System.err.println ( "Can not find system parameters" );
            params = new IcebergInfo ( new Hashtable ( 2 ) );
        }

        String[] headers =
        params.getStrings( "TRANSCRIPT_TABLE_HEADER" );

        courseData = new SimpleTableDataModel( headers, null );
        courseTable = new JTable( courseData );
```

```
     TableColumn tc;
     int width, totalWidth=0;
     for( int i=0; i<headers.length; i++ )
     {
         tc = courseTable.getColumn( headers[i] );
         width = params.getInt( headers[i] + "_LENGTH" );
         tc.setMinWidth( width );
         totalWidth += width;
     }

     courseTable.setPreferredScrollableViewportSize(
 new Dimension( 300, 100 ) );

     JScrollPane courseScroll = new JScrollPane ( courseTable );
     courseScroll.setBackground ( Color.lightGray );

     add ( courseScroll, BorderLayout.CENTER );
 }


public void cleanup() {
}


/*-------------- Methods of bean proerties ----------------*/
  public String[] getHeaders()
  {
      return courseData.getHeaders();
  }

  public void setHeaders ( String[] h )
  {
      courseData.setHeaders ( h );
  }

  public Vector getcontent()
  {
      return courseData.getContent();
  }

  public void setContent ( Vector c )
  {
      courseData.setContent ( c );
  }

/*-------------- Methods of IcebergJFrame -----------------*/
```

```
/**
 * Remove itself from this InfoBus Object
 */
 protected void removeInfoBusRoles ( InfoBus oldBus )
 {
     oldBus.removeDataProducer ( this );
     oldBus.removeDataConsumer ( this );
 }


/**
 * Add itself to this InfoBus Object
 */
 protected void addInfoBusRoles ( InfoBus newBus )
 {
     newBus.addDataProducer ( this );
     newBus.addDataConsumer ( this );
 }


/*-------------- Methods of InfoBusDataConsumer -----------------*/
/**
 * Being noticed that a dataItem is available
 * @param event a InfoBusItemAvailableEvent Object
 */
public void dataItemAvailable ( InfoBusItemAvailableEvent event )
 {
     String itemName = event.getDataItemName();
     if ( itemName.equals ( "STUDENT" ) )
     {
         Student student = (Student)getDataValue ( event, true );
         if ( student == null )
             setContent ( null );
         else
             setContent ( student.transcript );
     }
 }


/**
 * Being noticed that a dataItem is revoked
 * @param event a InfoBusItemRevokedEvent Object
 */
 public void dataItemRevoked ( InfoBusItemRevokedEvent event )
 {
     String itemName = event.getDataItemName();
     if ( itemName.equals ( "STUDENT" ) )
```

```
        {
            setContent ( null );
        }
    }


  /*-------------- Methods of DataItemChangeListener -------------*/
   /**
    * Being noticed that a dataItem has changed
    * @param event contain the changed information
    */
public void dataItemValueChanged ( DataItemValueChangedEvent event )
    {
        String propertyName =
        ( String ) event.getProperty ( "NAME" );
        if ( propertyName.equals ( "STUDENT" ) );
        {
            Student student = (Student)getDataValue ( event );
            if ( student == null )
                setContent ( null );
            else
                setContent ( student.transcript );
        }
    }
}
```

APPENDIX D

PACKAGE iceberg.server

The iceberg.server package contains the classes of the Iceberg server.

This package contains following classes:

- AdvisorServer.java
- AdvisorServerX.java
- Distance.java
- IcebergServer.java
- IncorrectloginException.java
- InfoStore.java
- PartialTree.java
- PersonSearcher.java
- QueryFactory.java
- Session.java
- SessionNotExistException.java
- SessionServer.java
- SessionServerX.java
- UpdateFactory.java

```
/************ AdvisorServer.java *****************/
package iceberg.server;

import java.rmi.*;
import java.rmi.server.*;
import java.sql.*;
import java.util.*;
import iceberg.util.*;


class AdvisorServer extends UnicastRemoteObject implements AdvisorServerX {

    SessionServer sessionServer;

    AdvisorServer( SessionServer ss ) throws RemoteException {
        sessionServer = ss;
    }


  // Check whether this session is created by a advisor
    protected Session verifyAdvisorSession ( Long sessionID )
    throws SessionNotExistException
    {
        if( sessionID == null )
            throw new SessionNotExistException ( "Session ID is null" );
        Session user = sessionServer.matchSession ( sessionID );
        if( user == null || !( user.user instanceof Advisor ) )
            throw new SessionNotExistException();
        return user;
    }
// Local function call


/*-------------- Methods of AdvisorServerX --------------------*/

   /**.
    * Return the information of the advisor who create this session
    * @param the sessionID, will be checked for validity
    * @return a Advisor Object
    */
    public Advisor getAdvisor ( Long sessionID )
    throws DataProcessException, SessionNotExistException
    {
        Session session = verifyAdvisorSession ( sessionID );
        return ( Advisor ) session.user;
    }
    public Vector getStudentList ( Long sessionID, SearchInfo key )
    throws DataProcessException, SessionNotExistException
    {
        Session session = verifyAdvisorSession ( sessionID );
        try
        {
            PersonSearcher ps = new PersonSearcher( key,
            null, PersonSearcher.STUDENT_ONLY );
            return ps.getResultList();

        }
        catch ( SQLException e )
        {
System.err.println("AdvisorServer:searchStudent: " + e.toString() );
            throw new DataProcessException(
             "Database error, no Student found." );
        }
    }


    public Vector getStudentList ( Long sessionID, BrowseInfo key )
    throws DataProcessException, SessionNotExistException
    {
        Session session = verifyAdvisorSession( sessionID );
        return null;
    }


    public Student getStudent ( Long sessionID, String sid )
    throws DataProcessException, SessionNotExistException
    {
        Session session = verifyAdvisorSession ( sessionID );
        sessionServer.setLock ( session, sid );
        try
        {
            return QueryFactory.queryStudent ( sid );
        }
        catch ( SQLException se )
        {
System.err.println( se.toString() );
            sessionServer.setLock ( session, null  );
            throw new DataProcessException(
            "Database error, can't access database." );
        }
    }
    public void addStudent ( Long sessionID, Student newStudent )
    throws DataProcessException, SessionNotExistException
    {
        Session session = verifyAdvisorSession ( sessionID );
        sessionServer.setLock ( session, newStudent.id );
        try
        {
            UpdateFactory.addStudent ( newStudent );
        }
        catch ( SQLException se )
        {
            se.printStackTrace();
            throw new DataProcessException(
             "Database error, can't access database." );
        }
    }


    public void removeStudent ( Long sessionID, String sid )
    throws DataProcessException, SessionNotExistException
    {
        Session session = verifyAdvisorSession ( sessionID );
        sessionServer.setLock ( session, sid );
        try
        {
            UpdateFactory.removeStudent ( session.user.id );
        }
        catch ( SQLException se )
        {
System.err.println( se.toString() );
            throw new DataProcessException(
            "Database error, can't access database." );
        }
        sessionServer.setLock ( session, null );
    }


    public void releaseLock ( Long sessionID )
    throws DataProcessException, SessionNotExistException
    {
        Session session = verifyAdvisorSession ( sessionID );
        sessionServer.setLock ( session, null );
    }


    public Vector getEMessageList ( Long sessionID )
```

```
   throws SessionNotExistException, DataProcessException
   {
       Session session = verifyAdvisorSession ( sessionID );
       try
       {
           return QueryFactory.getLogEntryList ( session.user.id );
       }
       catch ( SQLException e )
       {
System.err.println("getEMessageList " + e.toString() );
           throw new DataProcessException(
           "Database Error, can't get emessageList");
       }
   }


   public void addEMessage ( Long sessionID, EMessage em )
   throws SessionNotExistException, DataProcessException
   {
       Session session = verifyAdvisorSession ( sessionID );
       try
       {
           UpdateFactory.addEMessage ( session.user.id, em );
       }
       catch( SQLException e )
       {
System.err.println("addEMessage " + e.toString() );
           throw new DataProcessException(
           "Database Error, can't add emessage");
       }
   }



   public void removeEMessage ( Long sessionID, EMessage em )
   throws SessionNotExistException, DataProcessException
   {
       Session session = verifyAdvisorSession ( sessionID );
       try
       {
           UpdateFactory.removeEMessage( session.user.id, em );
       }
       catch ( SQLException e )
       {
System.err.println("removeEMessageEntry " + e.toString() );
           throw new DataProcessException(
           "Database Error, can't remove emessage");
```

```
        }
    }


    /**
     * Get the system lock for the current student locked by the advisor
     * @param sessionID the session number, will be checked for validity
     * @return a list of LogEntry
     */
    public Vector getLogEntryList ( Long sessionID )
    throws SessionNotExistException, DataProcessException
    {
        Session session = verifyAdvisorSession ( sessionID );
        try
        {
            if ( session.currentLock != null )
                return QueryFactory.getLogEntryList( session.currentLock );
            else
                throw new DataProcessException (
                "No student is locked now" );
        }
        catch ( SQLException e )
        {
System.err.println( "getLogEntryList" + e.toString() );
            throw new DataProcessException(
            "DataBase Error, can't get LogEntry list");
        }
    }


    public void addLogEntry ( Long sessionID, LogEntry entry )
    throws SessionNotExistException, DataProcessException
    {
        Session session = verifyAdvisorSession ( sessionID );
        try
        {
            entry.fromID = session.user.id;

            if ( session.currentLock == null )
                throw new DataProcessException ( "Send to whom?" );
            else
                entry.toID = session.currentLock;

            UpdateFactory.addLogEntry ( entry );
        }
        catch ( SQLException e )
        {
```

```
System.err.println( "addLogEntry " + e.toString() );
            throw new DataProcessException(
            "DataBase Error, can't add logEntry");
        }
    }


    /**
     * remove a log entry from the database
     * @param sessionID the session number, will be checked for validity
     * @param the log entry to be removed
     */
    public void removeLogEntry ( Long sessionID, LogEntry entry )
    throws SessionNotExistException, DataProcessException
    {
        Session session = verifyAdvisorSession ( sessionID );
        try
        {
            if ( ! session.user.id.equals ( entry.fromID ) )
                throw new DataProcessException ( "Not log entry creator" );

            if ( session.currentLock == null )
                throw new DataProcessException (
                "No student record is being processed" );
            else
            {
                if ( entry.toID == null )
                    entry.toID = session.currentLock;
                else if ( ! session.currentLock.equals ( entry.toID ) )
                    throw new DataProcessException (
                    "Can not remove, not current student" );
            }

            UpdateFactory.removeLogEntry ( entry );
        }
        catch ( SQLException e )
        {
            e.printStackTrace();
            throw new DataProcessException(
            "DataBase Error, can't remove Log Entry");
        }
    }


    public void updateBridgeRequirement( Long sessionID, Vector cnoList )
    throws SessionNotExistException, DataProcessException
    {
        Session session = verifyAdvisorSession ( sessionID );
        try
        {
            if ( session.currentLock != null )
            {
                UpdateFactory.updateBridgeRequirement (
                session.currentLock, cnoList );
                IcebergServer.getString("UpdateBridgeRequirement");
            }
        }
        catch ( SQLException e )
        {
System.err.println("updateBridgeRequirement " + e.toString() );
            throw new DataProcessException(
            "DataBase Error, can't update BridgeRequirement");
        }
    }


    public Vector getStoredProcedureList ( Long sessionID )
    throws SessionNotExistException, DataProcessException
    {
        Session session = verifyAdvisorSession ( sessionID );
        try
        {
            return QueryFactory.getStoredProcedureList ( false );
        }
        catch ( SQLException e )
        {
System.err.println("getCannedTransaction: " + e.toString() );
            throw new DataProcessException(
            "DataBase Error, can't get Transaction");
        }
    }


    public void updatePerson ( Long sessionID, Person person )
    throws SessionNotExistException, DataProcessException
    {
        Session session = verifyAdvisorSession ( sessionID );
        if ( ! person.id.equals ( session.currentLock ) )
            throw new DataProcessException (
            "Can not update personal information, not current student" );
        try
        {
            UpdateFactory.updatePerson ( person );
    }
    catch ( SQLException e )
    {
        e.printStackTrace();
        throw new DataProcessException(
        "DataBase Error, can not update personal information." );
    }
}



public void addBackgroundEntry ( Long sessionID, Hashtable entryInfo )
throws SessionNotExistException, DataProcessException
{
    Session session = verifyAdvisorSession ( sessionID );
    try
    {
        if ( session.currentLock != null )
            UpdateFactory.addBackgroundEntry (
            session.currentLock, entryInfo );
    }
    catch ( SQLException e )
    {
        e.printStackTrace();
        throw new DataProcessException(
        "DataBase Error, can not add background entry" );
    }
}



public void removeBackgroundEntry(Long sessionID, Hashtable entryInfo )
throws SessionNotExistException, DataProcessException
{
    Session session = verifyAdvisorSession ( sessionID );
    try
    {
        if ( session.currentLock != null )
            UpdateFactory.removeBackgroundEntry (
            session.currentLock, entryInfo );
    }
    catch ( SQLException e )
    {
        e.printStackTrace();
        throw new DataProcessException(
        "DataBase Error, can not remove background entry" );
    }
```

```
}


public Vector getBackgroundEntryList ( Long sessionID )
throws SessionNotExistException, DataProcessException
{
    Session session = verifyAdvisorSession ( sessionID );
    try
    {
        if ( session.currentLock != null )
            return QueryFactory.queryBackground( session.currentLock );
        else
            throw new DataProcessException (
            "No student is locked now" );
    }
    catch ( SQLException e )
    {
        e.printStackTrace();
        throw new DataProcessException
        ("DataBase Error, can't get Background");
    }
}


public Vector getTranscriptList ( Long sessionID )
throws SessionNotExistException, DataProcessException
{
    Session session = verifyAdvisorSession ( sessionID );
    try
    {
        if ( session.currentLock != null )
            return QueryFactory.queryTranscript ( session.currentLock );
        else
            throw new DataProcessException( "No student is locked now" );
    }
    catch ( SQLException e )
    {
        throw new DataProcessException
        ("DataBase Error, can not get Transcript ");
    }
}


public Vector getBridgeRequirement ( Long sessionID )
throws SessionNotExistException, DataProcessException
{
    Session session = verifyAdvisorSession ( sessionID );
```

```
        try
        {
            if ( session.currentLock != null )
                return QueryFactory.queryBridgeRequirement (
                session.currentLock );
            else
                throw new DataProcessException ( "No student is locked now" );
        }
        catch ( SQLException e )
        {
            throw new DataProcessException
            ("DataBase Error, can't get bridge requirement" );
        }
    }


}
```

```
/************ AdvisorServerX.java ****************/
package iceberg.server;

import java.rmi.*;
import java.util.*;
import iceberg.util.*;

public interface AdvisorServerX extends Remote {

   /**.
    * Return the information of the advisor who create this session
    * @param sessionID the session number, will be checked for validity
    * @return a Advisor Object
    */
    public Advisor getAdvisor( Long sessionID )
    throws RemoteException, SessionNotExistException, DataProcessException;

   /**
    * Return a list of student informations satisfied the search key.
    * Note: only Person Objects will be returned and
    * only the student ID, name and Email address are included.
    * advisor may request a student's detailed data later
    * @param sessionID the session number, will be checked for validity
    * @param key how the search will be taken on
    * @return a list of Person Objects
    */
    public Vector getStudentList( Long sessionID, SearchInfo key )
    throws RemoteException, SessionNotExistException, DataProcessException;


   /**
    * Return a list of student informations defined by the canned transaction
    * Note: only Person Objects will be returned and
    * only the student ID, name and Email address are included.
    * advisor may request a student's detailed data later
    * @param sessionID the session number, will be checked for validity
    * @param pno the program number
    * @param transactionNumber the canned transaction number,
    * those transaction are
    * stored in the database
    * @return a list of Person Objects
    */
    public Vector getStudentList ( Long sessionID, BrowseInfo key )
    throws RemoteException, SessionNotExistException, DataProcessException;


   /**
```

```
 * Return a student's detailed information
 * This function will also release the previous lock hold by the advisor,
 * and set a lock on this student's record so that
 * no other advisor can access it.
 * @see iceberg.util.Student
 * @param sessionID the session number, will be checked for validity
 * @return a Student Object,
 */
 public Student getStudent ( Long sessionID, String sid )
 throws RemoteException, SessionNotExistException, DataProcessException;


/**
 * Add a new student record to the database
 * @param sessionID the session number, will be checked for validity
 * @param student the new student record
 */
 public void addStudent ( Long sessionID, Student newStudent )
 throws RemoteException, SessionNotExistException, DataProcessException;


/**
 * Delete a old student record from the database
 * @param sessionID the session number, will be checked for validity
 * @param student the old student record
 */
 public void removeStudent ( Long sessionID, String sid )
 throws RemoteException, SessionNotExistException, DataProcessException;


/**
 * Release the current lock hold by the advisor
 * @param sessionID the session number, will be checked for validity
 */
 public void releaseLock ( Long sessionID )
 throws RemoteException, SessionNotExistException, DataProcessException;


/**
 * Get a list of email message created by this advisor
 * @param sessionID the session number, will be checked for validity
 * @return a Vector Object, contains EMessage Objects
 * @see iceberg.util.EMessage
 */
 public Vector getEMessageList ( Long sessionID )
 throws RemoteException, SessionNotExistException, DataProcessException;
```

```
/**
 * Add a new email message to the database
 * @param sessionID the session number, will be checked for validity
 * @param em an EMessage Object
 * @see iceberg.util.EMessage
 */
public void addEMessage ( Long sessionID, EMessage em )
throws RemoteException, SessionNotExistException, DataProcessException;



/**
 * remove a email message from the database
 * @param sessionID the session number, will be checked for validity
 * @param em the email message to be removed
 * @see iceberg.util.EMessage
 */
public void removeEMessage ( Long sessionID, EMessage em )
throws RemoteException, SessionNotExistException, DataProcessException;


/**
 * Get the system lock for the current student locked by the advisor
 * @param sessionID the session number, will be checked for validity
 * @return a list of LogEntry
 */
public Vector getLogEntryList ( Long sessionID )
throws RemoteException, SessionNotExistException, DataProcessException;


/**
 * Add a log entry to the database
 * @param sessionID the session number, will be checked for validity
 * @param content the log content
 * note: Implicitly, IcebergServer will associate this log entry with
 * the student that is currently locked by this advisor, and save both
 * the log entry and the student ID to the database
 */
public void addLogEntry ( Long sessionID, LogEntry entry )
throws RemoteException, SessionNotExistException, DataProcessException;


/**
 * remove a log entry from the database
 * @param sessionID the session number, will be checked for validity
 * @param the log entry to be removed
```

```
*/
public void removeLogEntry ( Long sessionID, LogEntry entry )
throws RemoteException, SessionNotExistException, DataProcessException;


/**
* Update the student's bridge requirement. The old beidge requirements
* are deleted first, the the new requirements are added.
* @param sessionID the session number, will be checked for validity
* @param cnoList a list of bridge course number
* note: Implicitely, IcebergServer will use the currentLock value as the
* student whose bridge requirement will be changed
*/
public void updateBridgeRequirement ( Long sessionID, Vector cnoList )
throws RemoteException, SessionNotExistException, DataProcessException;


/**
* Get the canned transaction list in the database
* @param sessionID the session number, will be checked for validity
*/
public Vector getStoredProcedureList ( Long sessionID )
throws RemoteException, SessionNotExistException, DataProcessException;



/**
* Update the personal information
* @param sessionID, the session number, will be checked for validity
* @param person the personal information
*/
public void updatePerson ( Long sessionId, Person person )
throws RemoteException, SessionNotExistException, DataProcessException;



public Vector getBackgroundEntryList ( Long SessionID )
throws RemoteException, SessionNotExistException, DataProcessException;



/**
* Add a background entry to the database
* @param sessionID, the session number, will be checked for validity
* @param entryInfo, the background entry information
*/
public void addBackgroundEntry ( Long sessionID, Hashtable entryInfo )
```

```
        throws RemoteException, SessionNotExistException, DataProcessException;



      /**
        * Remove a background entry from the database
        * @param sessionID, the session number, will be checked for validity
        * @param entryInfo the background entry information
        */
        public void removeBackgroundEntry( Long sessionID, Hashtable entryInfo )
        throws RemoteException, SessionNotExistException, DataProcessException;



        public Vector getTranscriptList ( Long sessionID )
        throws RemoteException, SessionNotExistException, DataProcessException;

        public Vector getBridgeRequirement ( Long sessionID )
        throws RemoteException, SessionNotExistException, DataProcessException;

}
```

```
/************ Distance.java **************/
package iceberg.server;

import java.util.*;
import iceberg.util.*;

class Distance {

public static int stringDistance(String a, String b) {
  int lengthA = a.length(), lengthB = b.length(), i, j;
  int[][] dMatrix = new int[lengthA][lengthB];
  dMatrix[0][0] = 0;
  for(i=1; i<lengthA; i++)
    dMatrix[i][0] = dMatrix[i-1][0] + 1;
  for(j=1; j<lengthB; j++)
    dMatrix[0][j] = dMatrix[0][j-1];
  int cDelete, cRelabel, cInsert;
  for(i=1; i<lengthA; i++)
    for(j=1; j<lengthB; j++) {
      cRelabel = dMatrix[i-1][j-1] + ((a.charAt(i) == b.charAt(j))? 0 : 1);
      cDelete  = dMatrix[i-1][j] + 1;
      cInsert  = dMatrix[i][j-1] + 1;
      dMatrix[i][j] = Math.min( cRelabel, Math.min(cDelete, cInsert) );
    }
  return dMatrix[lengthA-1][lengthB-1];
}

public static int wordSoundDistance(String a, String b) {
  int[] aw = computePronounceWeight( a );
  int[] bw = computePronounceWeight( b );
  if( aw == null || bw == null )
    return -1;
  return numberDistance( aw, bw );
}


public static int numberDistance( int[] a, int[] b ) {
  int distance = 0;
  int size = Math.min( a.length, b.length );
  int i;
  for( i=0; i<size; i++ )
    distance += Math.abs( a[i] - b[i] );
  if( size < a.length ) {
    for( ; i<a.length; i++ )
      distance += Math.abs( a[i] );
  }
  else {
    for( ; i<b.length; i++ )
      distance += Math.abs( b[i] );
  }
  return distance;
}

public static int numberDistance( String a, String b ) {
  int[] aw = computeDigitWeight( a );
  int[] bw = computeDigitWeight( b );
  if( aw == null || bw == null )
    return -1;
  else
    return numberDistance( aw, bw );
}


protected static int[] computeDigitWeight( String num ) {
  int[] weight = new int[num.length()];
  char digit;
  for( int i=0; i<weight.length; i++ ) {
    digit = num.charAt( i );
    if( !Character.isDigit( digit ) )
      return null;
    switch( digit ) {
      case '0': weight[i] = 0;
                break;
      case '1': weight[i] = 1;
                break;
      case '2': weight[i] = 2;
                break;
      case '3': weight[i] = 3;
                break;
      case '4': weight[i] = 4;
                break;
      case '5': weight[i] = 5;
                break;
      case '6': weight[i] = 6;
                break;
      case '7': weight[i] = 7;
                break;
      case '8': weight[i] = 8;
                break;
      case '9': weight[i] = 9;
    }
  }
  return weight;
}
```

```
protected static int[] computePronounceWeight( String word ) {
  int[] weight = new int[word.length()];
  char letter;
  int  letterWeight;
  int  count = 0;
  for( int i=0; i<weight.length; i++ ) {
    letter = word.charAt( i );
    if( !Character.isLetter( letter ) )
      return null;
    if( Character.isUpperCase( letter ) )
      letter = Character.toLowerCase( letter );
    letterWeight = 0;
    switch( letter ) {
/*
      case 'a':
      case 'e':
      case 'h':
      case 'i':
      case 'o':
      case 'u':
      case 'w':
      case 'y': letterWeight = 0; break;
*/

      case 'b':
      case 'f':
      case 'p':
      case 'v': letterWeight = 1; break;

      case 'c':
      case 'g':
      case 'j':
      case 'k':
      case 'q':
      case 's':
      case 'x':
      case 'z': letterWeight = 2; break;

      case 'd':
      case 't': letterWeight = 3; break;

      case 'l': letterWeight = 4; break;

      case 'm':
      case 'n': letterWeight = 5; break;
      case 'r': letterWeight = 6;
    }
    if( letterWeight == 0 ||
       ( count != 0 && letterWeight == weight[count-1] ) )
      continue;
    else
      weight[count++] = letterWeight;
  }
  for( ; count<weight.length; count++ )
    weight[count] = 0;
  return weight;
}

}
```

```
/************** IcebergServer.java *******************/
/**
 * The basic service provider. First Object run by the VM
 * initialize and register all servers
 * create a infoStore Object
 * provide the parameter strings to other Objects
 * connect to database and provide Connection Object to other Objects
 */
package iceberg.server;

import java.io.*;
import java.util.*;
import java.sql.*;
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import iceberg.util.*;

public class IcebergServer {

    static ResourceBundle resources;

    static
    {
        try
        {
            resources = ResourceBundle.getBundle
            ( "IcebergServerInfo", Locale.getDefault() );
        }
        catch ( MissingResourceException e )
        {
            System.err.println("IcebergInfo.properties not found");
            System.exit(1);
        }
    }


    static String getString ( String name )
    {
        try
        {
            return resources.getString ( name );
        }
        catch ( MissingResourceException e )
        {
            return null;
        }
    }


    static String[] tokenize(String input)
    {
        StringTokenizer st = new StringTokenizer(input, "|");
        String[] cmd = new String [ st.countTokens() ];
        for ( int i = 0; i < cmd.length; i++)
            cmd[i] = (String)st.nextToken();
        return cmd;
    }



// Database handle
    static  Connection connection;
    static  InfoStore infoStore;

 // Used by main to test IcebergServer
    SessionServer sessionServer;
    AdvisorServer advisorServer;


    public IcebergServer() throws Exception
    {
      // Load the Oracle thin driver
        Class.forName("oracle.jdbc.driver.OracleDriver" );

      // Connect to the database
        connection = DriverManager.getConnection
        (
        IcebergServer.getString ( "Database" ),
        IcebergServer.getString ( "Username" ),
        IcebergServer.getString ( "Password" )
        );

        connection.setAutoCommit ( true );

      // Get the rmiregistry
        Registry registry = LocateRegistry.getRegistry
        (
        Integer.parseInt ( IcebergServer.getString( "Port" ) )
        );

        System.setSecurityManager( new RMISecurityManager() );
      // Initialize and register all servers
        sessionServer  = new SessionServer();
        registry.rebind( "SessionServer", sessionServer );
        advisorServer = new AdvisorServer( sessionServer );
        registry.rebind( "AdvisorServer", advisorServer );
//      registry.rebind( "AdvisorServer", new AdvisorServer(sessionServer));
//      registry.rebind( "AdminServer", new AdminServer( sessionServer ) );
      // Initialize InfoStore
        infoStore = new InfoStore();
System.err.println( "Iceberg Server initialized successfully." );
    }


    public static void main ( String[] args )
    {
        try {
            IcebergServer is = new IcebergServer();
/*
      Long id = is.sessionServer.login( "McHugh", "McHugh" );
      AdvisorServer as = is.advisorServer;
      SearchInfo si = new SearchInfo();
      si.id="135026822";
      si.idChoice = SearchInfo.MATCH_EXACTLY;
//      si.idChoice = SearchInfo.NO_SEARCH;
//      si.idChoice = SearchInfo.FIND_SIMILAR;
      si.name = "ganguli";
      si.nameChoice = SearchInfo.NO_SEARCH;
//      si.nameChoice = SearchInfo.MATCH_EXACTLY;
//      si.nameChoice = SearchInfo.FIND_SIMILAR;
//      si.nameChoice = SearchInfo.SOUND_LIKE;
      si.eAddressChoice = SearchInfo.NO_SEARCH;
      si.resultSize = SearchInfo.DEFAULT_SIZE;
      as.getStudent ( id, "135026822" );

      Vector v = as.getLogEntryList( id );
     for ( int i=0; i< v.size(); i++ )
  { LogEntry e = (LogEntry)v.elementAt(i);
  System.err.println( e.fromID + e.timeStamp.toString() + e.content );
  }
      if( v == null )
System.err.println( "result is null" );
      Vector clist = new Vector(5);
      clist.addElement( "CIS333" );
      clist.addElement( "CIS251" );
      EMessage em = new EMessage();
      em.subject = "Assign Bridge Requirement";
      em.message =
      "Your Bridge Requirement has been assigned, chech Iceberg";
      as.addEMessageEntry( id, em );
      System.exit( 0 );
      else {

      is.sessionServer.logout( id );
      System.exit( 1 );
      Person p = (Person)v.elementAt(0);
      StudentInfo s = as.queryStudentInfo( id, p.id );
   System.out.println( s.info.toString() );
      Hashtable h = is.advisorServer.queryProgramInfo( id );
      ProgramInfo prog = (ProgramInfo)h.get( new Integer(1) );
      Vector b = prog.bridgeCourses;
   System.out.println( b.toString() );
*/

        } catch ( Exception e )
        {
            System.out.println( e.toString() );
            e.printStackTrace();
            System.exit( 1 );
        }
    }


    /**
     * @return a Statement Object
     */
    static Statement getStatement() throws SQLException {
        return connection.createStatement();
    }


    /**
     * @param query a SQL query that use ? as variables
     * @return a PreparedStatement Object
     */
    static PreparedStatement getPreparedStatement( String query )
    throws SQLException {
        return connection.prepareStatement( query );
    }

    /**
     * @param query a SQL query that call stored procedure
     * @return a CallableStatement
     */
    static CallableStatement getCallableStatement( String query )
    throws SQLException {
        return connection.prepareCall( query );
    }

   /**
    * Force the database to commit current work
    */
    static void commitTransaction() throws SQLException
    {
        connection.commit();
    }

}
```

```
/************* IncorrectLoginException.java ****************/
package iceberg.server;

public class IncorrectLoginException extends Exception {

    public IncorrectLoginException() {
        super( "Incorrect userid/password." );
    }

    public IncorrectLoginException( String msg ) {
        super( msg );
    }

}
```

```
/************** InfoStore.java *****************/
/**
 * InfoStore store the database information that may be used repeatly,
 * so when the IcebergServer initializing, those information are fetched
 * from database( disk ) and stored in the infoStore Object( memory )
 * Currently, infoStore contains all the progam in the database.
 * The programs are stored as Program Objects.
 * @see iceberg.util.Program
 */
package iceberg.server;

import java.sql.*;
import java.util.*;
import iceberg.util.*;

class InfoStore
{
    Hashtable programTable;

    InfoStore() throws SQLException
    {
        programTable = new Hashtable( 6, 1.0f );
        gatherProgramInfo();
    }


   /**
    * return a Program Object on request
    * @param pno the program number
    * @return a Program Object, or null otherwise
    */
    public Program getProgram ( Integer pno )
    {
        if ( pno != null )
            return ( Program ) programTable.get ( pno );
        else
            return null;
    }

  // Generate a Program Object for each program in the database
    protected void gatherProgramInfo() throws SQLException {
        Statement st = IcebergServer.getStatement();
        try
        {
            ResultSet rs = st.executeQuery(
            "SELECT PNO, PNAME" + " FROM PROGRAM" );
            Program prog;
            while ( rs.next() )
            {
                Integer pno = new Integer( rs.getInt( "PNO" ) );
                prog = queryProgram( pno );
                prog.pName = rs.getString( "PNAME" );
                programTable.put( pno, prog );
            }
        } finally
        {
            st.close();
        }
    }

// return a Program Object of this pno,fill in each fields
    protected Program queryProgram( Integer pno ) throws SQLException
    {
        Statement st = IcebergServer.getStatement();
        try
        {
            Program prog = new Program();
            prog.pno = pno;
            String pnoString = pno.toString();
            String cno;
            ResultSet rs = st.executeQuery(
            "SELECT COURSE.CNO, CNAME, CREDIT"
            + " FROM COURSE, PROGRAM_OFFER_BRIDGE"
            + " WHERE PNO='" + pnoString
            + "' AND COURSE.CNO=PROGRAM_OFFER_BRIDGE.CNO"
            + " ORDER BY CNO" );
            while ( rs.next() )
            {
                if( prog.bridgeCourses == null )
                    prog.bridgeCourses = new Vector( 10 );
                cno = rs.getString( "CNO" );
                prog.bridgeCourses.addElement( cno );
                prog.putCourseDescription ( cno, rs.getString ( "CNAME" ),
                rs.getFloat ( "CREDIT" ) );
            }
            rs = st.executeQuery(
            "SELECT COURSE.CNO, CNAME, CREDIT"
            + " FROM COURSE, PROGRAM_OFFER_CORE"
            + " WHERE PNO='" + pnoString
            + "' AND COURSE.CNO=PROGRAM_OFFER_CORE.CNO"
            + " ORDER BY CNO" );
            while ( rs.next() )
            {
                if( prog.coreCourses == null )
                prog.coreCourses = new Vector( 6 );
            cno = rs.getString( "CNO" );
            prog.coreCourses.addElement( cno );
            prog.putCourseDescription ( cno, rs.getString ( "CNAME" ),
            rs.getFloat ( "CREDIT" ) );
        }
        rs = st.executeQuery(
        "SELECT COURSE.CNO, CNAME, CREDIT"
        + " FROM COURSE, PROGRAM_OFFER_REQUIRED"
        + " WHERE PNO='" + pnoString
        + "' AND COURSE.CNO=PROGRAM_OFFER_REQUIRED.CNO"
        + " ORDER BY CNO" );
        while ( rs.next() )
        {
            if( prog.requiredCourses == null )
            prog.requiredCourses = new Vector( 6 );
            cno = rs.getString( "CNO" );
            prog.requiredCourses.addElement( cno );
            prog.putCourseDescription ( cno, rs.getString ( "CNAME" ),
            rs.getFloat ( "CREDIT" ) );
        }
        rs = st.executeQuery(
        "SELECT COURSE.CNO, CNAME, CREDIT"
        + " FROM COURSE, TRACK_OFFER_ELECTIVE"
        + " WHERE PNO='" + pnoString
        + "' AND COURSE.CNO=TRACK_OFFER_ELECTIVE.CNO"
        + " ORDER BY CNO" );
        while ( rs.next() )
        {
            if( prog.electiveCourses == null )
            prog.electiveCourses = new Vector( 30 );
            cno = rs.getString( "CNO" );
            prog.electiveCourses.addElement( cno );
            prog.putCourseDescription ( cno, rs.getString ( "CNAME" ),
            rs.getFloat ( "CREDIT" ) );
        }
        return prog;
    } finally
    {
        st.close();
    }
  }
}
```

```
/************** PartialTree.java ***************/
package iceberg.server;

import java.util.*;
import iceberg.util.*;

public class PartialTree implements Enumeration{
  Vector pTree;
  int    size;
  boolean preferMin;
  class Entry {
    Object item;
    int    weight;
    Entry( Object i, int w ) {
      item = i;
      weight = w;
    }
  }


  PartialTree() {
    this( 50, true );
  }

  PartialTree( boolean preferMin ) {
    this( 50, preferMin );
  }

  PartialTree( int capacity ) {
    this( capacity, true );
  }

  PartialTree( int capacity, boolean preferMin ) {
    pTree = new Vector( capacity );
    size = 0;
    this.preferMin = preferMin;
  }

  public Enumeration elements() {
    return (Enumeration)this;
  }

  public boolean hasMoreElements() {
    return size > 0;
  }

  public Object nextElement() {
```

```
  return deleteRoot();
}

public void insert( int weight, Object item ) {
  Entry parent, child = new Entry( item, weight );
  int p, c = size++;
  if( size > pTree.capacity() ) pTree.setSize( size*2 );
  pTree.addElement( child );
  while( c != 0 ) {
    p = (c - 1) / 2;
    parent = (Entry)pTree.elementAt( p );
    if( (preferMin && parent.weight<=child.weight) ||
        (!preferMin && parent.weight>=child.weight) )
      break;
    pTree.setElementAt( parent, c );
//    pTree.setElementAt( child,  p );
    c = p;
  }
  pTree.setElementAt( child, c );
}

Object  deleteRoot() {
  if( size > 0 ) {
    Entry result = (Entry)pTree.elementAt(0);
    size--;
    if( size > 1 )
      reformTree();
    else if( size == 1 )
      pTree.setElementAt( pTree.elementAt(1), 0 );
    return result.item;
  }
  else
    return null;
}

private void reformTree() {
  Entry lChild, rChild, parent = (Entry)pTree.elementAt( size );
  int p = 0, lc, rc, limit = size / 2;
  while( p < limit ) {
    lc = p*2 + 1;
    lChild = (Entry)pTree.elementAt( lc );
    rc = lc + 1;
    if( rc >= size ) {
      if( (preferMin && parent.weight > lChild.weight) ||
          (!preferMin && parent.weight < lChild.weight) )
      {
        pTree.setElementAt( lChild, p );
```

```
          p = lc;
        }
        break;
      }
      rChild = (Entry)pTree.elementAt( rc );
      if( (preferMin && lChild.weight < rChild.weight) ||
          (!preferMin && lChild.weight > rChild.weight) ) {
        if( (preferMin && parent.weight <= lChild.weight) ||
            (!preferMin && parent.weight >= lChild.weight) )
          break;
        else {
          pTree.setElementAt( lChild, p );
          p = lc;
        }
      }
      else {
        if( (preferMin && parent.weight <= rChild.weight) ||
            (!preferMin && parent.weight >= rChild.weight) )
          break;
        else {
          pTree.setElementAt( rChild, p );
          p = rc;
        }
      }
    }
    pTree.setElementAt( parent, p );
  }

}
```

```
/************** PersonSearcher.java **************/
package iceberg.server;

import java.util.*;
import java.sql.*;
import iceberg.util.*;

class PersonSearcher
{
    static int ALL_PERSON    = 0;
    static int STUDENT_ONLY = 1;
    static int ADVISOR_ONLY = 2;
    static int FACULTY_ONLY = 3;
    static int SCALE        = 10000;

    Vector      resultList = null;
    SearchInfo info;
    SearchFlavor flavor;
    int         personType;
    boolean     singleName;
    String      name1;
    String      name2;


    PersonSearcher ( SearchInfo si, SearchFlavor f, int pType )
    throws SQLException
    {
        info = si;
        if( f != null )
            flavor = f;
        else
            flavor = new SearchFlavor( 6, 3, 8, 6, 3, 6, 2 );
        personType = pType;
        if( info.nameChoice != SearchInfo.NO_SEARCH )
            processName();
        search();
    }


    protected void processName()
    {
        StringTokenizer st = new StringTokenizer( info.name );
        int tokenNum = st.countTokens();
        if ( tokenNum == 1 )
            singleName = true;
        else if( tokenNum == 2 )
        {
            name1 = st.nextToken();
            name2 = st.nextToken();
            singleName = false;
        }
        else
System.err.println( "PersonSearcher:processName: Incorrect name " +
info.name );

    }


    protected void search() throws SQLException
    {
        Vector pList;
        if ( personType == ALL_PERSON )
            pList = QueryFactory.getAllPerson();

        else if ( personType == STUDENT_ONLY )
            pList = QueryFactory.getAllStudent();

        else if ( personType == ADVISOR_ONLY )
            pList = QueryFactory.getAllAdvisor();

        else if ( personType == FACULTY_ONLY )
            pList = QueryFactory.getAllFaculty();

        else
        {
System.err.println( "Unknown choice: " + personType );
            return;
        }

        PartialTree pTree =  new PartialTree ( pList.size() + 1 );

        float idDistance = 0;
        float nameDistance = 0;
        float eAddressDistance = 0;
        Person p;
        for ( int i=0; i<pList.size(); i++ )
        {
            p = ( Person ) pList.elementAt( i );

          // Compute id distance
            if ( info.idChoice != SearchInfo.NO_SEARCH )
            {
                idDistance = computeIDDistance( p );
                if ( idDistance == -1 )
                continue;
            else
                idDistance *= flavor.idWeight;
        }

      // Compute eAddress distance
        if( info.eAddressChoice != SearchInfo.NO_SEARCH )
        {
            eAddressDistance = computeEAddressDistance( p );
            if ( eAddressDistance == -1 )
                continue;
            else
                eAddressDistance *= flavor.eAddressWeight;
        }

      // Compute name distance
        if ( info.nameChoice != SearchInfo.NO_SEARCH )
        {
            if( singleName )
                nameDistance = computeSingleNameDistance( p );
            else
                nameDistance = computeDoubleNameDistance( p );
            if( nameDistance == -1 )
                continue;
            else
                nameDistance *= flavor.nameWeight;
        }

      // Compute the total weight
        int sum = (int)( ( idDistance*idDistance +
           nameDistance*nameDistance +
           eAddressDistance*eAddressDistance ) * SCALE );
        pTree.insert( sum, p );

    }

    Enumeration enum = pTree.elements();
    resultList = new Vector ( info.resultSize );
    for ( int i=0; i<info.resultSize && enum.hasMoreElements(); i++ )
        resultList.addElement( enum.nextElement() );

}


protected float computeIDDistance ( Person p )
{
    if ( info.idChoice == SearchInfo.MATCH_EXACTLY )
```

```
        {
            if( info.id.equals ( p.id ) )
                return 0;
            else
                return -1;
        }
        else if ( info.idChoice == SearchInfo.FIND_SIMILAR )
        {
            int distance = Distance.numberDistance ( info.id, p.id );
            if ( distance == -1 || distance > flavor.idRadius )
                return -1;
            else
                return  (float)distance / (float)flavor.idRadius ;
        }
        else
        {
System.err.println("PersonSearcher:computeIDDistance: unknown choice " +
                   info.idChoice );
        return -1;
        }
    }


    protected float computeEAddressDistance ( Person p )
    {
        if ( info.eAddressChoice == SearchInfo.MATCH_EXACTLY )
        {
            if ( info.eAddress.equals ( p.eAddress ) )
                return 0;
            else
                return -1;
        }
        else if ( info.eAddressChoice == SearchInfo.FIND_SIMILAR )
        {
            int p1 = info.eAddress.indexOf( '@' );
            int p2 = p.eAddress.indexOf( '@' );

          // whether the two email addresses are from same host,
            if
            (
                p1 == -1 ||
                p2 == -1 ||
                ! info.eAddress.substring ( p1 + 1 ).equals (
                p.eAddress.substring ( p2 + 1 ) )
            )
                return -1;
            int distance = Distance.stringDistance
            (
                info.eAddress.substring( 0, p1 ),
                p.eAddress.substring( 0, p2 )
            );
            if ( distance == -1 || distance > flavor.eAddressRadius )
                return -1;
            else
                return (float)distance / (float)flavor.eAddressRadius;
        }
        else
        {
System.err.println("PersonSearcher:computeEAddressDistance: unknown choice "
                    + info.eAddressChoice );
            return -1;
        }

    }


    protected float computeSingleNameDistance ( Person p )
    {
        int fnameDistance, lnameDistance;

        if ( info.nameChoice == SearchInfo.MATCH_EXACTLY )
        {
            if ( info.name.equals ( p.fName ) ||
                 info.name.equals ( p.lName ) )
                return 0;
            else
                return -1;
        }
        else if ( info.nameChoice == SearchInfo.FIND_SIMILAR )
        {
            fnameDistance = Distance.stringDistance ( info.name, p.fName );
            lnameDistance = Distance.stringDistance ( info.name, p.lName );
            if ( fnameDistance <= lnameDistance )
            {
                if ( fnameDistance < flavor.nameSpellRadius )
                    return (float)fnameDistance /
                           (float)flavor.nameSpellRadius;
                else
                    return -1;
            }
            else
            {
                if ( lnameDistance < flavor.nameSpellRadius )
                return (float)lnameDistance /
                       (float)flavor.nameSpellRadius;
            else
                return -1;
        }

    }
    else if ( info.nameChoice == SearchInfo.SOUND_LIKE )
    {
        fnameDistance = Distance.wordSoundDistance (
                            info.name, p.fName );
        lnameDistance = Distance.wordSoundDistance (
                            info.name, p.lName );

        if ( fnameDistance <= lnameDistance )
        {
            if ( fnameDistance == -1 ||
                 fnameDistance > flavor.nameSoundRadius )
                return -1;
            else
                return (float)fnameDistance /
                       (float)flavor.nameSoundRadius;
        }
        else
        {
            if ( lnameDistance == -1 ||
                 lnameDistance > flavor.nameSoundRadius )
                return -1;
            else
                return (float)lnameDistance /
                (float)flavor.nameSoundRadius;
        }

    }
    else
    {
System.err.println("PersonSearcher:processVagueSearch: unknown search " +
                   info.nameChoice );
        return -1;
    }
}


protected float computeDoubleNameDistance ( Person p )
{
    if( info.nameChoice == SearchInfo.MATCH_EXACTLY )
    {
            if
            (
                ( name1.equals( p.fName ) && name2.equals( p.lName ) ) ||
                ( name1.equals( p.lName ) && name2.equals( p.fName ) )
            )
                return 0;
            else
                return -1;
        }
        else if ( info.nameChoice == SearchInfo.FIND_SIMILAR )
        {
            float fw, lw, w1, w2;

      // Try name1 as fname, name2 as lname
            fw = 1 - (float)Distance.stringDistance( name1, p.fName ) /
            (float)flavor.nameSpellRadius;
            lw = 1 - (float)Distance.stringDistance( name2, p.lName ) /
            (float)flavor.nameSpellRadius;

            if( fw < 0 || lw < 0 )
                return -1;
            else
                w1 = 1 - ( fw + lw - fw*lw );

      // Try name1 as lname, name2 as fname
            lw = 1 - (float)Distance.stringDistance( name1, p.lName ) /
            (float)flavor.nameSpellRadius;
            fw = 1 - (float)Distance.stringDistance( name2, p.fName ) /
            (float)flavor.nameSpellRadius;
            if( fw < 0 || lw < 0 )
                return -1;
            else
                w2 = 1 - ( fw + lw - fw*lw );

      // compare two result get the smaller
            if( w1 < w2 )
                return w1;
            else
                return w2;
        }
        else if ( info.nameChoice == SearchInfo.SOUND_LIKE )
        {
            float fw, lw, w1, w2;

// Try name1 as fname, name2 as lnamep
            fw = 1 - (float)Distance.wordSoundDistance( name1, p.fName ) /
            (float)flavor.nameSoundRadius;
            lw = 1 - (float)Distance.wordSoundDistance( name2, p.lName ) /
            (float)flavor.nameSoundRadius;
            if( fw < 0 || lw < 0 || fw > 1 || lw > 1 )
                return -1;
            else
                w1 = 1 - ( fw + lw - fw*lw );
// Try name1 as lname, name2 as fname

            lw = 1 - (float)Distance.wordSoundDistance( name1, p.lName ) /
            (float)flavor.nameSoundRadius;
            fw = 1 - (float)Distance.wordSoundDistance( name2, p.fName ) /
            (float)flavor.nameSoundRadius;
            if( fw < 0 || lw < 0 || fw > 1 || lw > 1 )
                return -1;
            else
                w2 = 1 - ( fw + lw - fw*lw );

// compare two result get the smaller
            if( w1 < w2 )
                return w1;
            else
                return w2;
        }
        else
        {
System.err.println("PersonSearcher:computeDoubleNameDistance: unknown choice"
                  + info.nameChoice );
            return -1;
        }

    }


    public Vector getResultList()
    {
        return resultList;
    }

}
```

```
/************** QueryFactory.java ****************/
/**
 * QueryFactory is a utility class which only contain static methods.
 * It provide static methods to query the database
 * Grouping all the JDBC methods together is easier to manage.
 * @see UpdateFactory
 */
package iceberg.server;

import java.sql.*;
import java.util.*;
import iceberg.util.*;


class QueryFactory {

   /**
    * check the if the ACCOUNT table  has an entry for this user
    * @param userID the user login ID
    * @param passwd the user password
    * @return a Session Object if successful, otherwise null
    */
    static Person checkLogin( String userID, String passwd )
    throws SQLException
    {
        Statement st = IcebergServer.getStatement();
        try
        {
            ResultSet rs = st.executeQuery ("SELECT ID, USAGE FROM ACCOUNT"
            + " WHERE USERID='" + userID + "' AND"
            + " PASSWD='" + passwd + "'" );
            if ( !rs.next() )
                return null;
            String id = rs.getString ( "ID" );
            String usage = rs.getString ( "USAGE" );

            Person p = null;
            if( usage != null && usage.equals( "A" ) )
                p = ( Person ) queryAdvisor( id );
            else if( usage != null && usage.equals( "S" ) )
                p = ( Person ) queryStudent( id );
            else {
System.err.println("QueryFactory:verifyUser: unknown usage " + usage);
                return null;
            }
            return p;
        }finally
        {
            st.close();
        }
    }

    /**
     * verify if the ADVISOR table has entries of this advisor
     * @param id the SSN or SID
     * @return a Advisor Object if successful, otherwise null
     */
    static Advisor queryAdvisor( String id ) throws SQLException
    {
        Statement st = IcebergServer.getStatement();
        try
        {
            ResultSet rs = st.executeQuery(
            "SELECT PNO, ISADM FROM ADVISOR"
            + " WHERE ID='" + id + "'");
            Advisor adv = null;
            while ( rs.next() )
            {
                if ( adv == null )
                {
                    adv = new Advisor();
                    adv.id = id;
                    queryPerson ( adv );
                }
                Integer pno = new Integer( rs.getInt( "PNO" ) );
                String a = rs.getString ( "ISADM" );
                if ( a != null && a.equals ( "Y" ) )
                    adv.putProgram ( IcebergServer.infoStore.getProgram(
                    pno ), true );
                else
                    adv.putProgram ( IcebergServer.infoStore.getProgram(
                    pno ), false );
            }
            return adv;
        }finally
        {
            st.close();
        }
    }


    /**
     * verify if the STUDENT table has entries for this student
     * @param id the SSN or SID
```

```
* @return a Student Object, or null otherwise
*/
static Student queryStudent( String id ) throws SQLException
{
    Statement st = IcebergServer.getStatement();
    try
    {
        ResultSet rs = st.executeQuery( "SELECT PNO, TRNO FROM STUDENT"
        + " WHERE SID='" + id + "'");
        Student student = null;
        if( rs.next() ) {
            student = new Student();
            student.id = id;
            queryPerson( student );
            student.pno = new Integer ( rs.getInt( "PNO" ) );
            student.trno = new Integer ( rs.getInt( "TRNO" ) );
            student.background = queryBackground ( id );
            student.transcript = queryTranscript ( id );
            student.bridgeRequirement = queryBridgeRequirement ( id );
        }
        return student;
    }finally
    {
        st.close();
    }
}


/**
 * get the person infomation from PERSON table
 * @param id the SSN or SID
 * @return a Person object, or null otherwise
 */
 static Person queryPerson ( String id ) throws SQLException
 {
     Statement st = IcebergServer.getStatement();
     try
     {
         ResultSet rs = st.executeQuery ( "SELECT * FROM PERSON"
         + " WHERE ID='" + id + "'");

         Person p =  null;
         if ( rs.next() )
         {
             p = new Person();
             p.id = id;
             p.fName = rs.getString ( "FNAME" );
```

```
                p.mName = rs.getString ( "MNAME" );
                p.lName = rs.getString ( "LNAME" );
                p.bDate = rs.getDate ( "BDATE" );
                p.gender = rs.getString ( "GENDER" );
                p.address = rs.getString ( "ADDRESS" );
                p.homePhone = rs.getString ( "HOMEPHONE" );
                p.workPhone = rs.getString ( "WORKPHONE" );
                p.eAddress = rs.getString( "EADDRESS" );
            }
            return p;
        } finally
        {
            st.close();
        }
    }


    /**
     * get the person information from PERSON table
     * @param person a Person Object whose id field is set
     */
    static void queryPerson ( Person p ) throws SQLException
    {
        Statement st = IcebergServer.getStatement();
        try
        {
            ResultSet rs = st.executeQuery ( "SELECT * FROM PERSON"
            + " WHERE ID='" + p.id + "'");

            if( rs.next() )
            {
                p.fName = rs.getString ( "FNAME" );
                p.mName = rs.getString ( "MNAME" );
                p.lName = rs.getString ( "LNAME" );
                p.bDate = rs.getDate ( "BDATE" );
                p.gender =rs.getString ( "GENDER" );
                p.address = rs.getString ( "ADDRESS" );
                p.homePhone = rs.getString ( "HOMEPHONE" );
                p.workPhone = rs.getString ( "WORKPHONE" );
                p.eAddress = rs.getString( "EADDRESS" );
            }
        } finally
        {
            st.close();
        }
    }
```

```
static Vector browseStudent( BrowseInfo key ) {
  return null;
}



 /**
  * return a Hashtable that contain following student information:
  * personal information,
  * "BRIDGEREQUIREMENT_LIST"
  * "TRANSCRIPT_TABLE"
  * "BACKGROUND_TABLE"
  * "LOG_TABLE"
  * @param sid the student SSN or SID
  * @return a Hashtable Object contains above information
  */
  static Hashtable queryStudentInfo ( String sid ) throws SQLException
  {
      Hashtable info = new Hashtable ( 6 );
      IcebergUtility.addToTable ( info, "BRIDGEREQUIREMENT_LIST",
       queryBridgeRequirement( sid ) );
      IcebergUtility.addToTable ( info, "TRANSCRIPT_TABLE",
      queryTranscript( sid ) );
      IcebergUtility.addToTable ( info, "BACKGROUND_TABLE",
      queryBackground( sid ) );

      return info;
  }


/*
 /**
  * return a Hashtable that contain student's personal information
  * such as ID, name, address, gender, birthday, workphone, homephone
  * and email address
  * @param id the SSN or SID
  * @return a Hashtable Object contain the information

  static Hashtable queryPersonalInfo( String id ) throws SQLException
  {
      Statement st = IcebergServer.getStatement();
      try
      {
          Hashtable info = new Hashtable ( 15, 1.0f );
          info.put ( "ID", id );
          ResultSet rs = st.executeQuery (
          "SELECT fName, mName, lName, BDATE,"
```

```
            + " GENDER, ADDRESS, HOMEPHONE, WORKPHONE, EADDRESS"
            + " FROM PERSON WHERE ID='" + id + "'");
            if ( rs.next() )
            {
                String m = rs.getString( "mName" ) ;
                StringBuffer sb = new StringBuffer( 80 );
                if ( m == null || m.equals( " " ) )
                {
                    sb.append( rs.getString( "fName" ) );
                    sb.append( " " );
                    sb.append( rs.getString( "lName" ) );
                }
                else
                {
                    sb.append( rs.getString( "fName" ) );
                    sb.append( " " );
                    sb.append( m );
                    sb.append( " " );
                    sb.append( rs.getString( "lName" ) );
                }
                info.put( "NAME", sb.toString() );
                String bdate = rs.getString( "BDATE" );
                if( bdate != null )
                    info.put( "birthday", bdate.substring( 0, 10 ) );
                IcebergUtility.addToTable( info, "GENDER",
                rs.getString( "GENDER" ) );
                IcebergUtility.addToTable( info, "ADDRESS",
                rs.getString( "ADDRESS" ) );
                IcebergUtility.addToTable( info, "HOMEPHONE",
                rs.getString( "HOMEPHONE" ) );
                IcebergUtility.addToTable( info, "WORKPHONE",
                rs.getString( "WORKPHONE" ) );
                IcebergUtility.addToTable( info, "EADDRESS",
                rs.getString( "EADDRESS" ) );
            }
            return info;
        } finally
        {
            st.close();
        }

    }

*/
  /**
   * return the transcript that the student has taken
   * @param sid the student SSN or SID
* @return a Vector Object contain the courses the student has taken
*/
static Vector queryTranscript( String sid ) throws SQLException
{
    Statement st = IcebergServer.getStatement();
    try
    {
        ResultSet rs = st.executeQuery(
        "SELECT COURSE.CNO, CNAME, CREDIT, SEMESTER, GRADE"
        + " FROM COURSE, STUDENT_TAKEN_COURSE"
        + " WHERE COURSE.CNO=STUDENT_TAKEN_COURSE.CNO"
        + " AND SID='" + sid + "'"
        + " ORDER BY SEMESTER, COURSE.CNO" );
        Vector cList = null;
        Hashtable c;
        while( rs.next() )
        {
            if( cList == null )
                cList = new Vector( 15 );
            c = new Hashtable( 6, 1.0f );
            IcebergUtility.addToTable( c, "CNO",
            rs.getString( "CNO" ) );
            IcebergUtility.addToTable( c, "Cname",
            rs.getString( "CNAME" ) );
            IcebergUtility.addToTable( c, "Credit",
            new Float( rs.getFloat( "CREDIT" ) ) );
            IcebergUtility.addToTable( c, "Semester",
            rs.getString( "SEMESTER" ) );
            IcebergUtility.addToTable( c, "Grade",
            new Float( rs.getFloat( "GRADE" ) ) );
            cList.addElement( c );
        }

        rs = st.executeQuery("SELECT COURSE.CNO, CNAME, CREDIT"
                    + " FROM COURSE, STUDENT_TAKING_COURSE"
                    + " WHERE COURSE.CNO=STUDENT_TAKING_COURSE.CNO"
                    + " AND SID='" + sid + "'"
                    + " ORDER BY COURSE.CNO" );
        while( rs.next() )
        {
            if( cList == null )
                cList = new Vector( 7 );
            c = new Hashtable( 4, 1.0f );
            IcebergUtility.addToTable( c, "CNO",
            rs.getString( "CNO" ) );
            IcebergUtility.addToTable( c, "Cname",
            rs.getString( "CNAME" ) );
```

```
                IcebergUtility.addToTable( c, "Credit",
                new Float( rs.getFloat( "CREDIT" ) ) );
                cList.addElement( c );
            }
            return cList;
        } finally
        {
            st.close();
        }
    }


   /**
    * return the student background
    * @param sid the student SSN/SID
    * @return a Vector Object contain the student's background
    */
    static Vector queryBackground( String sid ) throws SQLException
    {
        Statement st = IcebergServer.getStatement();
        try
        {
            ResultSet rs = st.executeQuery
            ( "SELECT * FROM EDU_BACKGROUND"
              + " WHERE ID='" + sid + "'" );
            Vector bList = null;
            Hashtable be;
            while( rs.next() )
            {
                if( bList == null )
                    bList = new Vector( 5 );
                be = new Hashtable( 7, 1.0f );
                IcebergUtility.addToTable( be, "College",
                rs.getString( "COLLEGE" ) );
                IcebergUtility.addToTable( be, "Location",
                rs.getString( "LOCATION" ) );
                IcebergUtility.addToTable( be, "Major",
                rs.getString( "MAJOR" ) );
                IcebergUtility.addToTable( be, "Degree",
                rs.getString( "DEGREE" ) );
                IcebergUtility.addToTable( be, "GDate",
                rs.getString( "GDATE" ) );
                float gpa = rs.getFloat( "GPA" );
                if( gpa != 0 )
                    be.put( "G.P.A.", new Float( gpa ) );
                bList.addElement( be );
            }
        return bList;
    } finally
    {
        st.close();
    }
}


/**
 * return the student's bridge requirement
 * @param sid the studnet SSN/SID
 * @return a Vector Object contains the student's bridge requirement
 */
static Vector queryBridgeRequirement( String sid ) throws SQLException
{
    Statement st = IcebergServer.getStatement();
    try
    {
        ResultSet rs = st.executeQuery(
        "SELECT COURSE.CNO, CNAME, CREDIT"
        + " FROM COURSE, BRIDGE_REQUIRE_STUDENT"
        + " WHERE COURSE.CNO=BRIDGE_REQUIRE_STUDENT.CNO"
        + " AND SID='" + sid + "'"
        + " ORDER BY COURSE.CNO" );
        Vector cList = null;
        Course c;
        while( rs.next() )
        {
            if( cList == null )
                cList = new Vector( 10 );
            c = new Course();
            c.cno = rs.getString ( "CNO" );
            c.cname = rs.getString ( "CNAME" );
            c.credit = rs.getFloat ( "CREDIT" );
            cList.addElement( c );
        }
        return cList;
    } finally
    {
        st.close();
    }
}


/**
 * return a course information
* @param cno the course number
* return a Course Object
* @see iceberg.util.Course
*/
static Course queryCourseInfo( String cno ) throws SQLException
{
    Statement st = IcebergServer.getStatement();
    try
    {
        ResultSet rs = st.executeQuery(
        "SELECT CNAME, CREDIT FROM COURSE"
        + " WHERE CNO='" + cno + "'" );
        Course c = null;
        if ( rs.next() )
        {
            c = new Course();
            c.cno = cno;
            c.cname = rs.getString( "CNAME" );
            c.credit = rs.getFloat( "CREDIT" );
        }
        return c;
    } finally
    {
        st.close();
    }
}


/**
 * return a list of EMessage created by this advisor
 * @param id the advisor's SSN
 * @return a Vector Object contains EMessage Object
 * @see iceberg.util.EMessage
 */
 static Vector getEMessageList( String id ) throws SQLException
 {
     Statement st = IcebergServer.getStatement();
     try
     {
         ResultSet rs = st.executeQuery("SELECT SUBJECT, CONTENT"
                                   + " FROM EMESSAGE"
                                   + " WHERE OWNER='" + id + "'");
         Vector eList = null;
         EMessage msg;
         while ( rs.next() )
         {
             if( eList == null )
```

```
                eList = new Vector(10);
            msg = new EMessage();
            msg.subject = rs.getString( "SUBJECT" );
            msg.content = rs.getString( "CONTENT" );
            eList.addElement( msg );
        }
        return eList;
    } finally
    {
        st.close();
    }
}


/**
 * return the system log about this person
 * @param id the SSN or SID
 * @return a Vector Object that contains log information
 */
static Vector getLogEntryList( String id ) throws SQLException
{
    Statement st = IcebergServer.getStatement();
    try
    {
        ResultSet rs = st.executeQuery(
        "SELECT FROM_ID, TIME, CONTENT"
        + " FROM SYSLOG"
        + " WHERE TO_ID='" + id + "'"
        + " ORDER BY TIME");
        Vector logList = null;
        LogEntry entry;
        while( rs.next () )
        {
            if( logList == null )
                logList = new Vector( 20 );

            entry = new LogEntry();
            entry.content =  rs.getString( "CONTENT" );
            entry.fromID = rs.getString( "FROM_ID" );
            entry.fromName = queryPerson ( entry.fromID ).getName();
            entry.timeStamp = rs.getDate( "TIME" );

            logList.addElement( entry );
        }
        return logList;
    } finally
    {
        st.close();
    }
}


static Vector getStoredProcedureList( boolean isAdm )
throws SQLException
{
    return null;
}


static Person queryPersonByEAddress ( String eAddress )
throws SQLException
{
    Statement st = IcebergServer.getStatement();
    try
    {
        ResultSet rs = st.executeQuery (
        "SELECT ID, LNAME, MNAME, FNAME FROM PERSON"
        + " WHERE EADDRESS='" + eAddress + "'" );

        Person p = null;
        if ( rs.next() )
        {
            p = new Person();
            p.id = rs.getString ( "ID" );
            p.lName = rs.getString ( "LNAME" );
            p.mName = rs.getString ( "MNAME" );
            p.fName = rs.getString ( "FNAME" );
            p.eAddress = eAddress;
        }

        return p;
    }
    finally
    {
        st.close();
    }
}


static Vector queryPersonByFName ( String fName )
throws SQLException
{
    Statement st = IcebergServer.getStatement();
    try
    {
        ResultSet rs = st.executeQuery (
        "SELECT ID, lName, mName, EADDRESS FROM PERSON"
        + " WHERE fName='" + fName + "'" );

        Person p = null;
        Vector pList = null;
        while ( rs.next() )
        {
            if( pList == null )
                pList = new Vector( 20 );
            p = new Person();
            p.id = rs.getString ( "ID" );
            p.lName = rs.getString ( "LNAME" );
            p.mName = rs.getString ( "MNAME" );
            p.fName = fName;
            p.eAddress = rs.getString ( "EADDRESS" );
        }

        return pList;
    }
    finally
    {
        st.close();
    }
}


static Vector queryPersonByLName ( String lName ) throws SQLException
{
    Statement st = IcebergServer.getStatement();
    try
    {
        ResultSet rs = st.executeQuery (
        "SELECT ID, MNAME, FNAME, EADDRESS FROM PERSON"
        + " WHERE lName='" + lName + "'" );

        Person p = null;
        Vector pList = null;
        if ( rs.next() )
        {
            if( pList == null )
                pList = new Vector( 20 );
            p = new Person();
            p.id = rs.getString ( "ID" );
            p.lName = lName;
            p.mName = rs.getString ( "MNAME" );
```

```
                p.fName = rs.getString ( "FNAME" );
                p.eAddress = rs.getString ( "EADDRESS" );
            }

            return pList;
        }
        finally
        {
            st.close();
        }
    }


static boolean isStudent( String id ) throws SQLException {
  Statement st = IcebergServer.getStatement();
  try {
  ResultSet rs = st.executeQuery("SELECT SID FROM STUDENT WHERE SID='" +
                               id + "'" );
  if( rs.next() )
    return true;
  else
    return false;
  }finally { st.close(); }
}


static boolean isAdvisor( String id ) throws SQLException {
  Statement st = IcebergServer.getStatement();
  try {
  ResultSet rs = st.executeQuery("SELECT ID FROM ADVISOR WHERE ID='" +
                               id + "'" );
  if( rs.next() )
    return true;
  else
    return false;
  }finally { st.close(); }
}


static boolean isFaculty( String id ) throws SQLException {
  Statement st = IcebergServer.getStatement();
  try {
  ResultSet rs = st.executeQuery("SELECT FID FROM FACULTY WHERE FID='" +
                               id + "'" );
  if( rs.next() )
    return true;
  else
```

```
    return false;
  }finally { st.close(); }
}


  static Vector getAllStudent() throws SQLException
  {
      Statement st = IcebergServer.getStatement();
      try
      {
          ResultSet rs = st.executeQuery (
          "SELECT ID, FNAME, MNAME, LNAME, EADDRESS" +
          " FROM PERSON, STUDENT WHERE PERSON.ID=STUDENT.SID" );

          Vector rList = new Vector( 300 );
          Person p;
          while ( rs.next() )
          {
              p = new Person();
              p.id = rs.getString( "ID" );
              p.fName = rs.getString( "FNAME" );
              p.mName = rs.getString( "MNAME" );
              p.lName = rs.getString( "LNAME" );
              p.eAddress = rs.getString( "EADDRESS" );

              rList.addElement( p );
          }

          return rList;
      }
      finally
      {
          st.close();
      }
  }


  static Vector getAllAdvisor() throws SQLException
  {
      Statement st = IcebergServer.getStatement();
      try
      {
          ResultSet rs = st.executeQuery (
          "SELECT PERSON.ID, FNAME, MNAME, LNAME, EADDRESS" +
          " FROM PERSON, ADVISOR WHERE PERSON.ID=ADVISOR.ID" );

          Vector rList = new Vector( 10 );
```

```
        Person p;
        while ( rs.next() )
        {
            p = new Person();
            p.id = rs.getString( "ID" );
            p.fName = rs.getString( "FNAME" );
            p.mName = rs.getString( "MNAME" );
            p.lName = rs.getString( "LNAME" );
            p.eAddress = rs.getString( "EADDRESS" );

            rList.addElement( p );
        }

        return rList;
    }
    finally
    {
        st.close();
    }
}


static Vector getAllFaculty() throws SQLException
{
    Statement st = IcebergServer.getStatement();
    try
    {
        ResultSet rs = st.executeQuery(
        "SELECT ID, FNAME, MNAME, LNAME, EADDRESS" +
        " FROM PERSON, FACULTY WHERE PERSON.ID=FACULTY.FID" );

        Vector rList = new Vector( 50 );
        Person p;
        while ( rs.next() )
        {
            p = new Person();
            p.id = rs.getString( "ID" );
            p.fName = rs.getString( "FNAME" );
            p.mName = rs.getString( "MNAME" );
            p.lName = rs.getString( "LNAME" );
            p.eAddress = rs.getString( "EADDRESS" );

            rList.addElement( p );
        }

        return rList;
    }
```

```
        finally
        {
            st.close();
        }
    }


    static Vector getAllPerson() throws SQLException
    {
        Statement st = IcebergServer.getStatement();
        try
        {
            ResultSet rs = st.executeQuery(
            "SELECT ID, FNAME, MNAME, LNAME, EADDRESS FROM PERSON");

            Vector rList = new Vector( 500 );
            Person p;
            while ( rs.next() )
            {
                p = new Person();
                p.id = rs.getString( "ID" );
                p.fName = rs.getString( "FNAME" );
                p.mName = rs.getString( "MNAME" );
                p.lName = rs.getString( "LNAME" );
                p.eAddress = rs.getString( "EADDRESS" );

                rList.addElement( p );
            }

            return rList;
        }
        finally
        {
            st.close();
        }
    }


}
```

```
/************** Session.java *****************/
/**
 * Session define the minimum information of a session created for a user
 * The user can lock an record in the database, the currentLock should
 * be a Person's id or null
 * The client need to refresh its session by set the time to zero periodically,
 * otherwise an IcebergServer thread wakes up periodically,
 * sweeps out those sessions
 * whose timer exceeds a certain threshold.
 */
package iceberg.server;

import iceberg.util.Person;

public class Session {

    public Long    sessionID;
    public String  currentLock;
    public int     timer;
    public Person  user;

}
```

```
/************** SessionNotExistException.java ****************/
package iceberg.server;

public class SessionNotExistException extends Exception {

    public SessionNotExistException() {
        super("Session not exist or closed by Iceberg Server");
    }

    public SessionNotExistException( String msg ) {
        super( msg );
    }

}
```

```
/*************** SessionServer.java ****************/
/**
 * Control the session session information, provide functions
 * to login, logout and access a session session.
 * It also control the activity of a session
 * The client need to refresh its session from time to time,
 * baecause periodically, a thread of SessionServer is waken
 * and remove those sessions that is not active for a certain time.
 */
package iceberg.server;

import java.util.*;
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.sql.*;
import java.io.*;
import iceberg.util.*;


class SessionServer extends UnicastRemoteObject
implements SessionServerX, Runnable {

    private Hashtable sessionTable;
    private Hashtable lockTable;
    private Vector    sessionList;
    private Random    rand;
    private Session   superUser = null;
    private long      scanInterval;
    private int       waitThreshold;



    SessionServer() throws RemoteException {
        sessionTable = new Hashtable(40);
        lockTable    = new Hashtable(40);
        sessionList     = new Vector(40);
        rand = new Random();
        Thread t = new Thread( this );
        t.start();
    }



    public synchronized Long login( String userID, String passwd )
    throws DataProcessException, IncorrectLoginException
    {
        if( superUser != null )
            throw new DataProcessException(
          "Iceberg is being used by SuperUser, Please exit and try later" );
        try
        {
            Person person = QueryFactory.checkLogin ( userID, passwd );
            if( person == null )
                throw new IncorrectLoginException();
            Session session = new Session();
            session.user = person;
            do {
                session.sessionID = new Long( Math.abs( rand.nextLong() ) );
            } while( matchSession( session.sessionID ) != null );
            sessionList.addElement( session );
            sessionTable.put( session.sessionID, session );
            session.timer = 0;
System.err.println( "Advisor log in successfully." );
            return session.sessionID;
        }
        catch ( SQLException se )
        {
System.out.println( se.toString() );
            throw new DataProcessException(
            "Database error! Can't verify session." );
        }
    }

    public synchronized void logout ( Long sessionID )
    {
        Session session = ( Session ) sessionTable.remove ( sessionID );
        if( session != null )
        {
            if( session.currentLock != null )
                lockTable.remove( session.currentLock );
            sessionList.removeElement( session );
        }
    }

    public void isAlive( Long sessionID )
    throws SessionNotExistException, DataProcessException
    {
        Session session = matchSession( sessionID );
        if( session == null )
            throw new SessionNotExistException();
        if( superUser != null && session != superUser )
            throw new DataProcessException(
            "Iceberg is being used by SuperUser," +
            " Please save your work and exit Iceberg" );
    }


    Session matchSession( Long sessionID ) {
        Session session = ( Session ) sessionTable.get ( sessionID );
        if( session != null &&
        ( superUser == null || session == superUser ) )
            session.timer = 0;
        return session;
    }

    synchronized void setLock( Session session, String id  )
    throws DataProcessException {
        if( session.currentLock != null ) {
            if ( session.currentLock.equals ( id ) )
                return;
            else
            {
                lockTable.remove( session.currentLock );
                session.currentLock = null;
            }
        }
        if( id != null ) {
            Session owner = ( Session )lockTable.get( id );
            if( owner != null )
                throw new DataProcessException(
                "Session " + owner.user.getName() +
                " has locked this record.");
            lockTable.put( id, session );
            session.currentLock = id;
        }
    }

/*
    synchronized void setSuperUserMode( Advisor adv, boolean mode )
    throws DataProcessException {
        if( mode ) {
            if( superUser != null )
                throw new DataProcessException(
                "Error: Can not begin Super Session Mode," +
                " Administrator " + superUser.session.getName() +
                " has been the current Super Session" );
            superUser = adv;
        }
        else {
            if( superUser == null )
```

```
                throw new DataProcessException(
                "Error: Iceberg Server is not in SuperUser mode" );
            if( superUser != adv )
                throw new DataProcessException(
                "Error: not current Super Session");
            superUser = null;
        }
    }
*/
    public synchronized Hashtable getSessionProfile( Long sessionID )
    throws SessionNotExistException, DataProcessException {
        matchSession( sessionID );
        try {
            BufferedReader bw = new BufferedReader( new FileReader(
                    "../IcebergInfo.properties" ) );
            Hashtable profileTable = new Hashtable( 100 );
            String line;
            int delimiter, count = 0;
            while( ( line = bw.readLine() ) != null ) {
                if( line.length() > 0 && line.charAt( 0 ) != '#' ) {
                    delimiter = line.indexOf( '=' );
                    if( delimiter != -1 && delimiter != line.length() - 1 )
                        profileTable.put( line.substring( 0, delimiter ),
                                line.substring( delimiter + 1 ) );
                }
            }
            bw.close();
            return profileTable;
        }catch( IOException e ) {
            throw new DataProcessException( "Can not read profile." );
        }
    }

    public void run() {
        String intervalString = IcebergServer.getString( "SCAN_INTERVAL" );
        if( intervalString == null )
            scanInterval = 60000;
        else
            scanInterval = Long.parseLong( intervalString );
        String thresholdString = IcebergServer.getString( "WAIT_THRESHOLD" );
        if( thresholdString == null )
            waitThreshold = 5;
        else
            waitThreshold = Integer.parseInt( thresholdString );
        for( ; ; ) {
            try {
                Thread.sleep( scanInterval );
            }catch( InterruptedException e ) {}
            cleanup();
        }
    }

    protected synchronized void cleanup() {
        for( int i=sessionList.size()-1; i>=0; i-- ) {
            Session currentSession = ( Session ) sessionList.elementAt( i );
            currentSession.timer++;
            if( currentSession.timer > waitThreshold ) {
System.out.println( "Exceed threshold: " + currentSession.user.getName() );
                sessionTable.remove( currentSession.sessionID );
                if( currentSession.currentLock != null )
                    lockTable.remove( currentSession.currentLock );
                if( currentSession == superUser )
                    superUser = null;
                sessionList.removeElementAt( i );
            }
        }
    }

}
```

```
/********** SessionServerX.java ***************/
/**
 * the RMI interface for SessionServer
 * @see SessionServer
 */
package iceberg.server;

import java.rmi.*;
import java.util.*;
import iceberg.util.*;

public interface SessionServerX extends Remote {

   /**
    * Log into the Iceberg System, SessionServer will create a
    * session and other data structure.
    * @param userID the user login ID
    * @param passwd the user password
    * @return the session number which identities the session
    */
    public Long login( String userID, String passwd )
    throws RemoteException, DataProcessException, IncorrectLoginException;


   /**
    * Log out of the Iceberg System, SessionServer will release all
    * resourses used by this session
    * @param sessionID the session number, will be checked for validity
    */
    public void logout( Long sessionID )
    throws RemoteException;


   /**
    * This function is periodically called by the client to refresh the timer
    * in the session data structure
    * @see Session
    * Other remote call to, say, AdvisorServer will refresh the timer too.
    * this function is useful when the client is not active for a while
    * @param sessionID the session number, will be checked for validity
    */
    public void isAlive( Long sessionID )
    throws RemoteException, SessionNotExistException, DataProcessException;


   /**
    * Get the profile need by the client applet to initialize itself
    * @param sessionID the session number, will be checked for validity
    */
    public Hashtable getSessionProfile( Long sessionID )
    throws RemoteException, SessionNotExistException, DataProcessException;

}
```

```
/************** UpdateFactory.java ****************/
package iceberg.server;

import java.sql.*;
import java.util.*;
import iceberg.util.*;

public class UpdateFactory {

    static void updateTranscript(Statement st, String sid, Vector t) {}

    static void updateBridgeRequirement ( String sid, Vector cnoList )
    throws SQLException
    {
        Statement st = IcebergServer.getStatement();
        PreparedStatement pst = IcebergServer.getPreparedStatement(
        "INSERT INTO BRIDGE_REQUIRE_STUDENT VALUES ('" + sid + "', ?)" );
        try
        {
            st.executeUpdate (
            "DELETE FROM BRIDGE_REQUIRE_STUDENT WHERE SID='"
            + sid + "'" );
            for( int i=0; i<cnoList.size(); i++ )
            {
                pst.setString ( 1, ( String ) cnoList.elementAt( i ) );
                pst.executeUpdate();
            }
        }
        finally
        {
            st.close();
            pst.close();
        }
    }


    static void addEMessage( String id, EMessage em ) throws SQLException
    {
        Statement st = IcebergServer.getStatement();
        try
        {
            st.executeUpdate("INSERT INTO EMESSAGE VALUES ('" + id
            + "', '" + em.subject + "', '" + em.content + "')" );
        }
        finally
        {
            st.close();
    }
}


static void removeEMessage( String id, EMessage em )
throws SQLException
{
    Statement st = IcebergServer.getStatement();
    try
    {
        st.executeUpdate("DELETE FROM EMESSAGE WHERE OWNER='" + id
        + "' AND SUBJECT='" + em.subject + "'" );
    }
    finally
    {
        st.close();
    }
}

static void addLogEntry ( LogEntry entry ) throws SQLException
{
    Statement st = IcebergServer.getStatement();
    try
    {
        st.executeUpdate(
        "INSERT INTO SYSLOG(FROM_ID, TO_ID, TIME, CONTENT) VALUES ('"
        + entry.fromID + "', '" + entry.toID + "', SYSDATE, '"
        + entry.content + "')" );
    }
    finally
    {
        st.close();
    }
}


static void removeLogEntry ( LogEntry entry ) throws SQLException
{
    PreparedStatement pst = IcebergServer.getPreparedStatement (
    "DELETE FROM SYSLOG WHERE FROM_ID=? AND TO_ID=? AND CONTENT=?" );
    try
    {
        pst.setString ( 1, entry.fromID );
        pst.setString ( 2, entry.toID );
        pst.setString ( 3, entry.content );
        pst.executeUpdate();
    }
    finally
    {
        pst.close();
    }
}


static void addStudent ( Student s ) throws SQLException
{
    Statement st = IcebergServer.getStatement();
    try
    {
        StringBuffer header = new StringBuffer (
        "INSERT INTO PERSON (" );
        StringBuffer value = new StringBuffer ( " VALUES (" );

        header.append ( "ID" ).append ( "," );
        value.append ( "'" ).append ( s.id ).append ( "'," );

        if ( s.bDate != null )
        {
            header.append ( "BDATE" ).append ( "," );
    value.append ( "'" ).append ( s.bDate.toString() ).append ( "'," );
        }
        if ( s.gender != null )
        {
            header.append ( "GENDER" ).append ( "," );
            value.append ( "'" ).append ( s.gender ).append ( "'," );
        }
        if ( s.address != null )
        {
            header.append ( "ADDRESS" ).append ( "," );
            value.append ( "'" ).append ( s.address ).append ( "'," );
        }
        if ( s.homePhone != null )
        {
            header.append ( "HOMEPHONE" ).append ( "," );
            value.append ( "'" ).append ( s.homePhone ).append ( "'," );
        }
        if ( s.workPhone != null )
        {
            header.append ( "WORKPHONE" ).append ( "," );
            value.append ( "'" ).append ( s.workPhone ).append ( "'," );
        }
        if ( s.eAddress != null )
        {
            header.append ( "EADDRESS" ).append ( ")" );
            value.append ( "'" ).append ( s.eAddress ).append ( "')" );
        }
        header.append ( "LNAME" ).append ( "," );
        value.append ( "'" ).append ( s.lName ).append ( "'," );
        if ( s.mName != null )
        {
            header.append ( "MNAME" ).append ( "," );
            value.append ( "'" ).append ( s.mName ).append ( "'," );
        }
        header.append ( "FNAME" ).append ( ")" );
        value.append ( "'" ).append ( s.fName ).append ( "')" );



        st.executeUpdate ( header.toString() + value.toString() );

        header = new StringBuffer ( "INSERT INTO STUDENT (" );
        value = new StringBuffer ( " VALUES (" );
        header.append ( "SID" ).append ( "," );
        value.append ( "'" ).append ( s.id ).append ( "'," );
        if ( s.trno != null )
        {
            header.append ( "TRNO" ).append ( "," );
     value.append ( "'" ).append ( s.trno.toString() ).append ( "'," );
        }
        header.append ( "PNO" ).append ( ")" );
     value.append ( "'" ).append ( s.pno.toString() ).append ( "')" );

        st.executeUpdate ( header.toString() + value.toString() );

    }
    finally
    {
        st.close();
    }
}


static void removeStudent( String sid ) throws SQLException {
}


static void updatePerson ( Person person ) throws SQLException
{
    Statement st = IcebergServer.getStatement();

    StringBuffer query = new StringBuffer ( "UPDATE PERSON SET" );
```

```
        if ( person.bDate != null )
            query.append ( " BDATE='" + person.bDate.toString() + "'," );
        if ( person.gender != null )
            query.append ( " GENDER='" + person.gender + "'," );
        if ( person.address != null )
            query.append ( " ADDRESS='" + person.address + "'," );
        if ( person.homePhone != null )
            query.append ( " HOMEPHONE='" + person.homePhone + "'," );
        if ( person.workPhone != null )
            query.append ( " WORKPHOME='" + person.workPhone + "'," );
        if ( person.eAddress != null )
            query.append ( " EADDRESS='" + person.eAddress + "'," );
        String queryString =
        query.toString().substring ( 0, query.length() - 1 );

        try
        {
            st.executeUpdate (
            queryString + " WHERE ID='" + person.id + "'" );
        }
        finally
        {
            st.close();
        }
    }


    public static void addBackgroundEntry ( String id, Hashtable entry )
    throws SQLException
    {
/*
        PreparedStatement pst= IcebergServer.getPreparedStatement
     ( "INSERT INTO EDU_BACKGROUND ( ID, COLLEGE, LOCATION, MAJOR, DEGREE,"
          + " GDATE, GPA) VALUES ( ?, ?, ?, ?, ?, ?, ? )" );

        try
        {
            pst.setString ( 1, id );
            pst.setString ( 2, (String)entry.get ( "College" ) );
            pst.setString ( 3, (String)entry.get ( "Location" ) );
            pst.setString ( 4, (String)entry.get ( "Major" ) );
            pst.setString ( 5, (String)entry.get ( "Degree" ) );
            pst.setString ( 6, (String)entry.get ( "GDate" ) );
            Float gpa = (Float)entry.get ( "G.P.A." );
            if ( gpa != null )
            pst.setFloat ( 7, gpa.floatValue() );
            pst.executeUpdate();
```

```
*/
        Statement st = IcebergServer.getStatement();
        StringBuffer header = new StringBuffer(
        "INSERT INTO EDU_BACKGROUND (");
        StringBuffer value = new StringBuffer ( " VALUES (" );

        Object content;
        content = entry.get ( "College" );
        if ( content != null )
        {
            header.append ( " COLLEGE," );
            value.append ( " '" + (String)content + "'," );
        }
        content = entry.get ( "Location" );
        if ( content != null )
        {
            header.append ( " LOCATION," );
            value.append ( " '" + (String)content + "'," );
        }
        content = entry.get ( "Major" );
        if ( content != null )
        {
            header.append ( " MAJOR," );
            value.append ( " '" + (String)content + "'," );
        }
        content = entry.get ( "Degree" );
        if ( content != null )
        {
            header.append ( "DEGREE," );
            value.append ( " '" + (String)content + "'," );
        }
        content = entry.get ( "GDate" );
        if ( content != null )
        {
            header.append ( " GDATE," );
            value.append ( " '" + (String)content + "'," );
        }
        content = entry.get ( "G.P.A." );
        if ( content != null )
        {
            header.append ( " GPA," );
            value.append ( (String)content + "," );
        }
        header.append ( "ID)" );
        value.append ( "'" + id + "')" );

        header.append ( value.toString() );
```

```
    try
    {
        st.executeUpdate ( header.toString() );
    }
    finally
    {
        st.close();
    }
}


public static void removeBackgroundEntry ( String id, Hashtable entry )
throws SQLException
{
    Statement st = IcebergServer.getStatement();

    StringBuffer buffer = new StringBuffer (
    "DELETE FROM EDU_BACKGROUND WHERE ID='" );
    buffer.append ( id );
    buffer.append ( "'" );

    Object value;
    value = entry.get ( "College" );
    if ( value != null )
    {
        buffer.append ( " AND COLLEGE='" );
        buffer.append ( (String)value );
        buffer.append ( "'" );
    }
    value = entry.get ( "Location" );
    if ( value != null )
    {
        buffer.append ( " AND LOCATION='" );
        buffer.append ( (String)value );
        buffer.append ( "'" );
    }
    value = entry.get ( "Major" );
    if ( value != null )
    {
        buffer.append ( " AND MAJOR='" );
        buffer.append ( (String)value );
        buffer.append ( "'" );
    }
    value = entry.get ( "Degree" );
    if ( value != null )
    {
        buffer.append ( " AND DEGREE='" );
```

```
            buffer.append ( (String)value );
            buffer.append ( "'" );
        }
        value = entry.get ( "GDate" );
        if ( value != null )
        {
            buffer.append ( " AND GDATE='" );
            buffer.append ( (String)value );
            buffer.append ( "'" );
        }
        value = entry.get ( "G.P.A." );
        if ( value != null )
        {
            buffer.append ( " AND GPA=" );
            buffer.append ( value.toString() );
            buffer.append ( "" );
        }

        try
        {
            st.executeUpdate ( buffer.toString() );
        }
        finally
        {
            st.close();
        }
    }
}
```

REFERENCES

1. The swing connection. Javasoft, December, 1998 http://www.javasoft.com/products/jfc/tsc/index.html

2. JavaBeans "Glasgow" Draft Specifications JavaSoft, December 1998 http://www.javasoft.com/beans/glasgow/

3. eSuite Infobus Technology Brifing Lotus Inc., December 1998 http://java.sun.com/beans/infobus/index.html

4. Java Plug-in 1.2 Overview JavaSoft Inc, December 1998 http://www.javasoft.com/products/plugin/1.2/overview.html