Developing Database Applications by Using Software Components

Nusret Conk

DEVELOPING DATABASE APPLICATIONS BY USING SOFTWARE COMPONENTS

Nusret Conk

Developing Database Applications By Using Software Components

by

Nusret Conk

A Thesis Submitted to the Faculty

of the College of Graduate Studies

at Georgia Southern University

in Partial Fulfillment of the Requirements for a Degree of

Master of Science

in the Department of

Mathematics and Computer Science

Statesboro, Georgia

June, 1999

Developing Database Applications by Using Software Components

by

Nusret Conk

[illegible]
Chong-Wei Xu, Chairperson

[illegible]
Ahmed Barbour

[illegible]
Mark Aulick

[illegible]
G. Lane Van Tassell
Associate Vice President for Academic Affairs
and Dean of Graduate Studies

7/21/99
Date

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The experts often claim that software development lags hardware development. Although it is debatable which area influences the other for innovations, an obvious fact stands true that there is a growing demand for more software in the market. Naturally, while people expect more sophistication and reliability from software, they demand less cost of ownership. The software engineering answers with a component-based design approach to create cheaper but better applications. By using pre-existing software components, developers speed up the application development cycle while incorporating rich features of the components into their application.

The idea of creating applications with software components drives itself from the Object Oriented design concepts. Historically, programmers have dealt with design complexities with identifying repetitious tasks and coding their logic in procedures and modules. The Object Oriented design brought the "object" concept to define entities that pose certain characteristics and behavior. These objects provide an interface to communicate with other objects to extend their functionality. In other words, objects work together to produce meaningful results. A collection of objects make up a software component which can be an application all by itself or it can be one of the many components that make up an application. For example, a web browser uses a window object which uses a menu object and the menu object uses button objects. This web browser is a stand-alone application, and also, it can be a component in a larger application, such as a web database application. Application design with pre-existing components obviously speeds up the process. There are other benefits for using components in design and development. Each component is considered highly specialized that it behaves as intended by its designers. Its overall functionality is what concerns the people who use it as part of their application. Components hide their inner workings so only their creators can alter them; therefore, well designed and tested components exhibit consistent behavior and reliability. This certainly translates to developing better applications.

Developing robust applications and deploying them rapidly gained particular importance with the widespread acceptance of the Internet/intranet networks. The Internet with its over three-hundred billion dollar economy is the major player in directing software development market. Especially, in the areas of commerce, companies and consumers demand reliable applications to do business on the Internet. On the other hand, there is a growing need for information exchange between organizations and individuals. Today, people on the Internet use tools, such as Yahoo or Excite to search for data all the time. Almost all these commercial or non-commercial activities on the Internet involve accessing remote databases. Customers can be thousands of miles away from the computer where the company maintains its inventory database, or a person in one part of the world could be accessing a library of information in another part in the world.

Users interact with those databases with an interface from their computers or terminals. Being able to interact with a database brings up an issue of accessing that database. In a most basic system, user interaction takes place on a computer where the actual database is stored. However, as the distance between the user computer (or the client system) and the database server system increases, complexities associated with remote accessing become apparent. Although accessing databases remotely is not a new topic, accessing over the Internet/Intranet environments is fairly new, and it is in the process of evolving with rapid advancements in the hardware and software technologies. It is almost daily that new tools become available to make databases accessible over the Internet environment.

The Sun Microsystem's Java language is a new addition and very popular in developing world wide web, or WWW, applications. The developers of Java designed it with the Internet in mind, and its suitability and features made Java a popular language. Java is an Object Oriented language and supports component-based design ideas in application development. For example, the Java platform provides the Java DataBase Connectivity, or JDBC, component to facilitate database access from Java Programs. The JDBC makes a database application development easier as it reduces the complexities of accessing and interacting databases to issuing method calls in its interface. By using such components, the application designers concentrate on the desired capabilities of the applications rather than spending time on implementation details of individual components of the application.

1.1 Thesis Objective

Today, the software application development process is more assembly work than a "build from scratch" approach. By placing pre-existing software components together, it is possible to create a complete application. Such components provide interfaces so that programs use them for their intended purposes. The objective of this thesis is to illustrate how software components work together to make a complete application. To illustrate the ideas and the components, this project presents a three-tiered web database application. This application, as a whole, is made up of the client side web browser, a database and the actual application programs which are Java servlets. The emphasis is placed on these servlets and how they use the Java Database Connectivity, or JDBC, to interface with the databases.

1.2 Thesis Outline

The first chapter provides highlighted information about web database applications and their components. Chapter two focuses on connectivity tools including Open Database Connectivity, or ODBC, and JDBC. Networking issues as well as popular methods for developing application interfaces are included as well. The third chapter concentrates on JAVA as it pertains to database application development and the alternative tools that it provides. Chapter three concludes by addressing security issues. The fourth chapter provides a detailed accounting of a working model incorporating design issues and implementation details. In conclusion, chapter five summarizes the concepts and ideas that are presented in previous chapters.

1.3 Databases

Almost all companies and organizations use some form of database for their specific needs. In fact, information gathering, maintaining and processing extends to all living beings as part of their need to survive. Although, while it may be a natural act for them, organizations put forth a serious effort to create and to utilize databases for their success and survival. It is that effort that sets one company from the other to pursue their organizational goals in this ever-challenging environment.

1.3.1 Relational Databases

The usefulness of any collected data directly relates to its storage medium and organization, as well as the available access methods. Data files arranged in tables with columns and rows based on predetermined rules have been the traditional approach. This way operations to search, update, or rearrange the data become more manageable. A database is a collection of these tables. In most cases, a single table can incorporate all data. As the size of this table grows due to redundancies, it can potentially outgrow storage space, become bulky while reducing performance of timely operations, and grow increasingly less fault tolerant. Therefore, it becomes necessary to maintain the database on several tables that are related by a common key. These systems of tables are called relational databases.

As the number of the tables in a database increases, normalization is necessary to minimize the data redundancies.

Employee database

Employee Table

EmpNo	EmpName	EmpDept
147	John	3
188	David	5
234	Mary	2
78	Alex	5

Department Table

DeptNo	DeptName
2	Sales
3	Accounting
5	Management

In order to produce meaningful results from these two tables, the employee department field (EmpDept) in the employee table and the department number (DeptNo) field in the department table need a link. Then, it becomes possible to know exactly where an employee works.

1.3.2 Access Methods

There are different approaches to accessing databases. Storage platforms, available management software, user needs and data security are some of the issues which play important roles in determining the most feasible method for design and implementation of a database system. A single user on a single computer with a small database is a basic system, and it may allow direct access. However, there are more involved systems where a database is spanned over a network of computers with many users who access it simultaneously. Based on the requirements, designers must choose a system to insure accessibility, security and the integrity of a database.

1.3.2.1 Single-Tier Access

This is the most basic system of access. In this type of configuration, as mentioned earlier, the database resides on a single computer and a single user accesses it. In other words, all the data and the software to manage that data are tightly coupled in one computer system with a single point of access. Figure 1.1 illustrates the single tier architecture in which a client accesses the database through the DBMS interface.

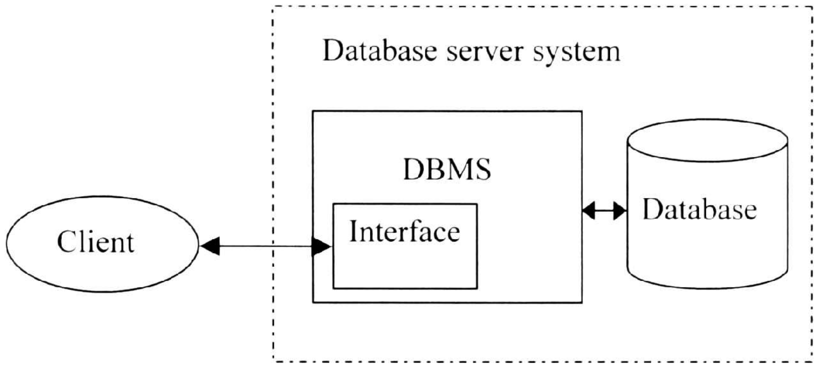


Fig. 1.1 Single-tier access method.

The user has direct access to the data in single-tier approach and can perform all the data creation, maintenance and retrieval functions through the interface of the DBMS. There are obvious disadvantages to single-tier systems: no multi-user access, less secure, and less fault tolerant. The cost of implementing and maintaining a single-tier system can be very low compared to multiple tier approaches, if it satisfies end user requirements.

1.3.2.2 Two-Tier Access

Often databases grow in size with increasing demand to store information electronically. Although single-tier systems can satisfy some needs they fall short of meeting the accessibility requirements of modern day businesses and organizations. There must be a way to allow multiple points of access without compromising security and data access speed. The two-tier approach provides methods to achieve such an architecture so that multiple users can access the same database. As one tier of the application is the server side that stores and manages data, while that other tier on the client machine provides the interface for input, output, and other functions for the enduser.

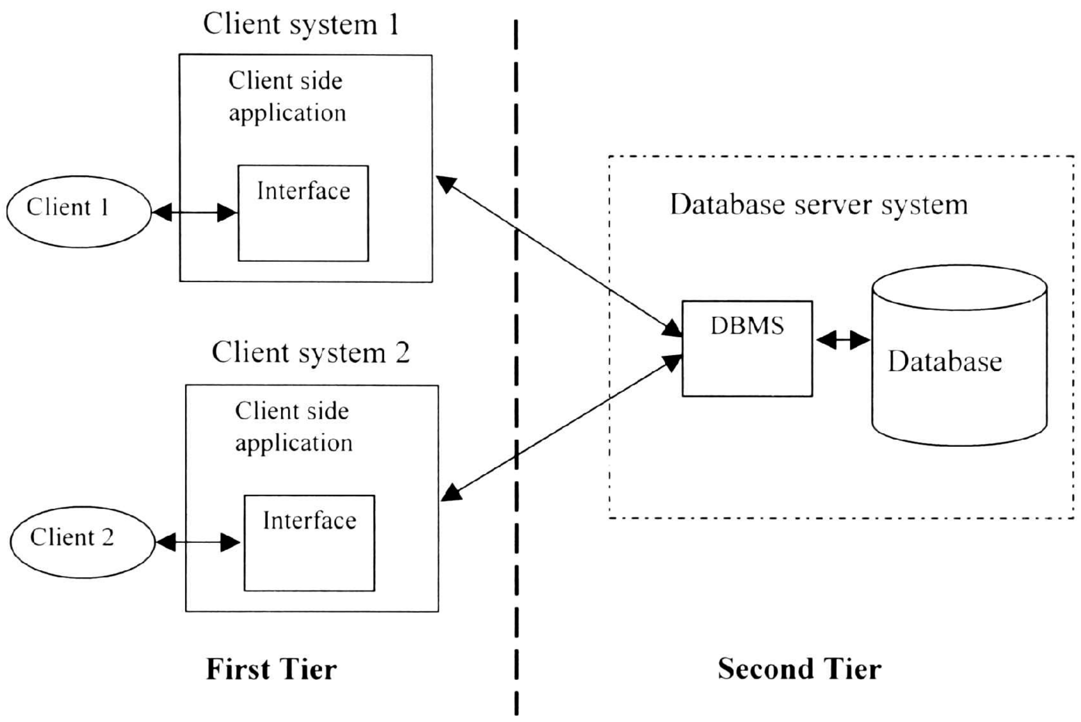


Fig. 1.2 Two-tier access with multiple clients.

The server side of the application works directly with the client side of the application as indicated in figure 1.2. There is no direct access to the database unless it is through the interface on the client system. The two-tier database management applications consist of client and server units. By doing so, the server side application offers more power and robustness for data management since the burden of the bulky user interface part of the application is handled by the client systems. In addition to improvements in speed, database access security gains an additional checkpoint in two-tier architectures.

Most Internet applications--email, telnet, ftp, gopher, and even the web--are simple two-tier applications. Without providing many data interpretation or processing, these applications provide a simple interface to access information across the internet[1].

While two-tier design is superior to single-tier design, it is not always the ultimate choice of design. Since the database resides on a single server, its potential growth must be taken into consideration to prevent performance pitfalls in the future. The host server will handle a limited amount of data processing with a limited number of clients. Any needed changes to the either side of the application can be costly, since both the client and the server are tightly coupled and therefore, such changes may require maintenance on both sides. The middle tier provides necessary flexibility to allow future data and client base expansion.

1.3.2.3 Three-Tier Access

The next approach is a three-tier system where an additional tier between the server side and the client side is added. As the complexity grows with the addition of a middle tier, the flexibility and the capability grows as well. The server side of the application now loses its heavy dependency on the client side application design, and interacts only with the middle tier. Therefore, the database engine solely deals with data handling and networking issues. The client side of the application also benefits with this approach in that it is not concerned with the specifics of that particular engine, and it is specialized in executing the user interface.

In the three-tiered system, there is a middle tier application between the client system and the backend database. Clients of such systems can access the database only through the middle tier. In other words, client side applications do not directly deal with the DBMS. The middle-tier application, also referred to as middleware application, facilitates connections and the database management activities that clients request. As figure 1.3 indicates, clients connect to the database via the application in the middle tier.

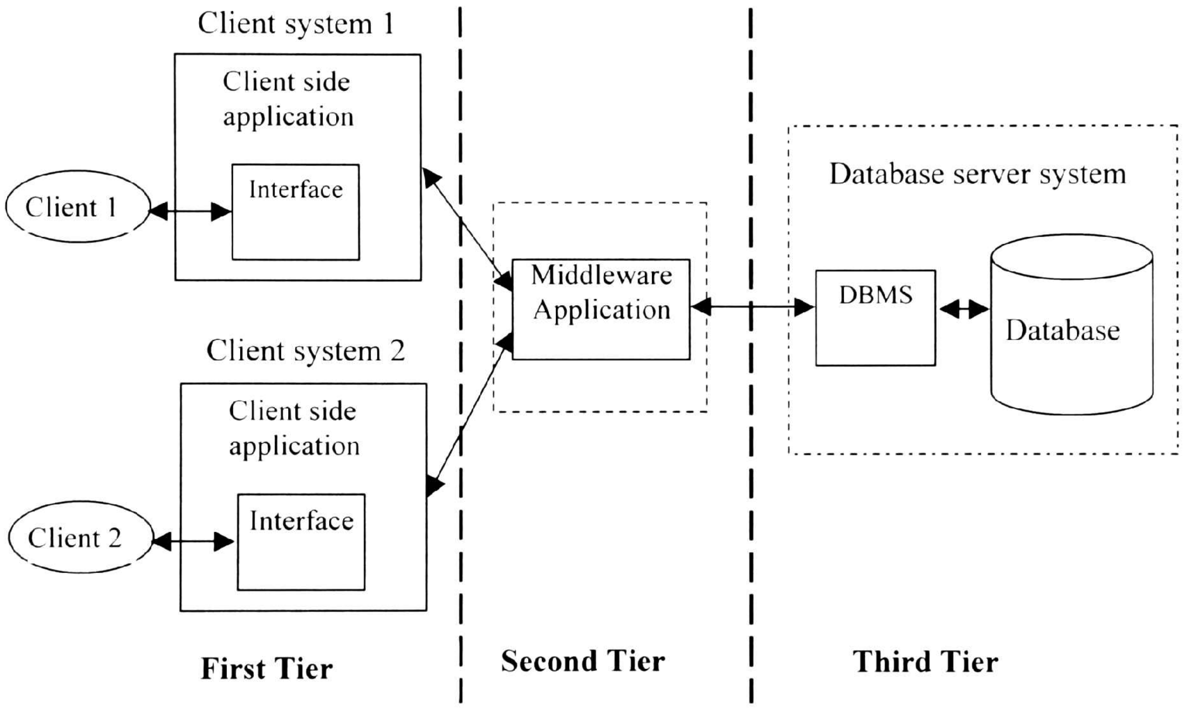


Fig. 1.3 Three tier access with multiple clients.

1.4 Web database applications

The term "web database application" refers to applications through which a client accesses some database via the Internet or Internet network. The span of the Internet is across the world; therefore, the physical distance between the client computer and the database can be thousands of miles. In a similar way, today's large organizations and businesses maintain their own private networks (intranets) which serve their members or employees in many offices, some of which may be widely separated. A web database application makes it possible to utilize distant databases as if they are local to a client on the Internet/Intranet.

There are three components that make up a web database application. The web browser on the client side, the actual application (the middleware) on the web server, and the database on some server. Although not limited to this particular configuration, a web database application program exists in the middle tier. Such systems are referred to as three-tiered systems of which the web browser is the first tier, the application programs are the second tier, and the database management system (DBMS) is the third tier. Figure 1.4 illustrates the three components of a web database application.

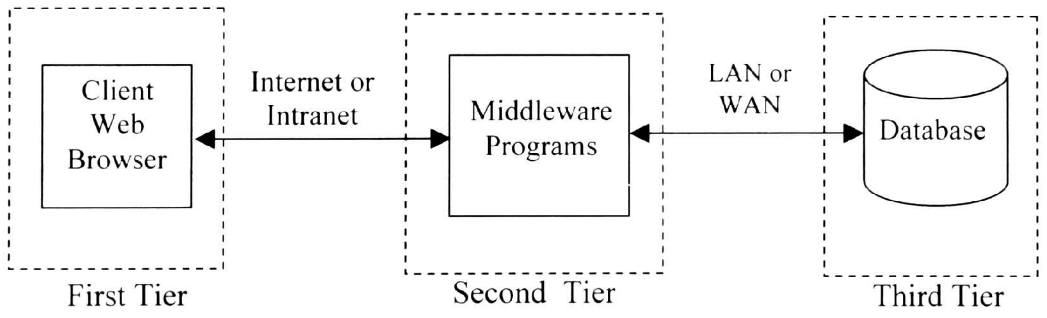


Fig. 1.4 Web database application components

The first tier application, such as Netscape or Internet Explorer, is on almost every personal computer today. Examples of third tier applications are MS Access, DB/2, Oracle, and MS SQL. These are very widely used DBMSs. The second tier applications make up a bridge between the first tier and the third tier.

1.4.1 First Tier - Web Browsers

Unlike typical client/server applications, the client side of a web database application is considered a "thin client." A web browser on the client computer provides the needed environment to interface with the server side application. Today, most client computer systems provide a web browser as part of their operating systems. Particularly, Microsoft's windows operating systems incorporate web browsing functionality into system tools and applications. There are other web browsers, such as Netscape, that are free or very reasonably priced. Since the client side system already provides the interface environment, the server side application needs to send the actual application interface to the client's browser. The application interface is an HTML file that the client's browser interprets and displays.

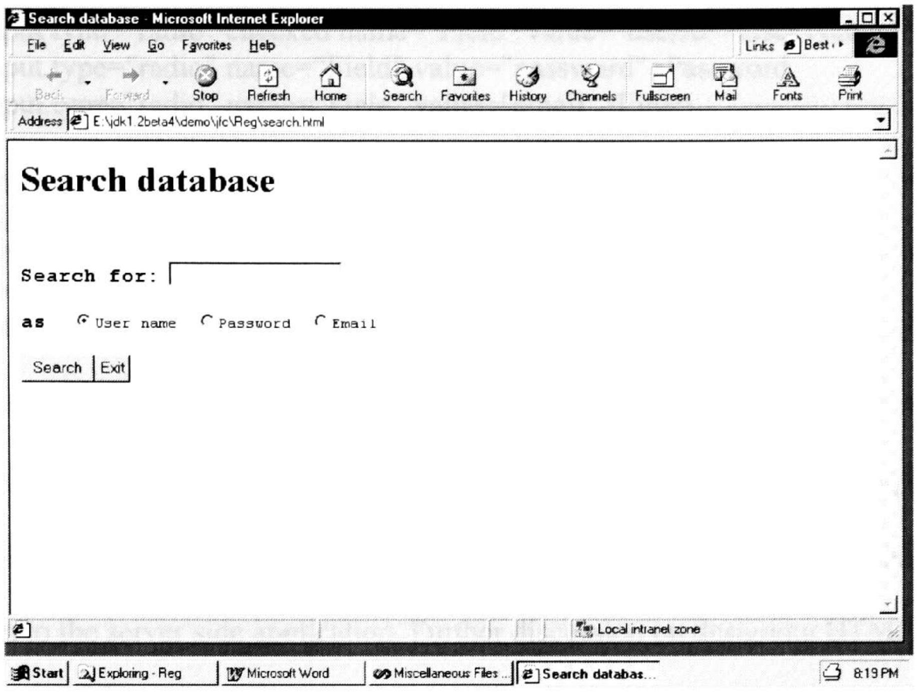
The middleware programs use HyperText Markup Language, or HTML, to construct the application interface that a client interacts with from his or her web browser. HTML is not a programming language but it defines how that interface is structured and displayed as a web page within the the browser environment. There are available set of HTML instructions, or tags, to construct web documents with sections, titles, paragraphs, lists, and tables. These instructions format the lines of text of a section to give it a certain desired appearance. Browsers display those HTML documents after interpreting the embedded tags and the returned data from the web server.

A basic HTML document is static in nature; once it is interpreted and displayed by the browser, it maintains its appearance until some type of user interaction takes place to go to another page. For example, the following HTML file displays a single line of text "Welcome to our page."

```
<HTML>
<HEAD>
<TITLE> A static HTML example </TITLE>
</HEAD>
<BODY>
   Welcome to our page
</BODY>
</HTML>
```

The tags <BODY> and </BODY> enclose the actual text that browser displays after formatting according to various formatting statements. The web server must send a new complete page if any information on this page needs to be changed or if any part of the page needs updating. There are, however, other tools or improved HTML versions such as DHTML that facilitate dynamic information display but browsers must support such extended features for them to work properly.

Data collection or manipulation requires user interaction. As mentioned earlier, HTML documents normally do not allow clients to interact with the page. It is, however, possible to create interactive HTML documents with visual components such as text boxes, check boxes, or drop down menus for data collection on the client side. Figure 1.5 indicates an HTML page which collects information to use in searching some database.


Search database - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Fonts Print
Address E:\jdk1.2beta4\demo\jfc\Reg\search.html
Search Exit
Local intranet zone
Start Exploring - Reg Microsoft Word Miscellaneous Files Search databas... 8:19 PM

Search database

Search for:

as ● User name ○ Password ○ Email

Fig. 1.5 HTML form with visual components.

This page, unlike a static HTML page, requires user interaction. The user enters a text to seek in the database and chooses a field to narrow the search operation. The search is initiated by clicking on the search button. By placing functional components on an HTML page, information gathering takes place within a browser. The following code represents the search form above.

```
<html>
<head>
<title>Search database</title>
</head>
<body bgcolor="#FFFFFF">
<p align="left"><font size="6"><strong>Search database</strong></font></p>

<form action="http://server1:8080/servlet/search" method="POST"
name="Search">
```

```
<input type="text" size="20" name="searchText">

<input type="radio" checked name="Field" value="userId">User Name
<input type="radio" name="Field" value="password">Password
<input type="radio" name="Field" value="email">Email

<input type="submit" name="Action" value="Search">
<input type="submit" name="Action" value="Exit"></pre>
</form>
</body>
</html>
```

A collection of such input fields in an HTML page makes up a form. Typically, a form contains various input fields as well as a "submit" button to send the entered information to the server side application. Further discussion on designing HTML pages is beyond the scope of this paper.

Web browsers make up the first tier, or the client side, of a web application. Basically, they provide an environment for the client interface through which data is gathered or presented. In a three-tiered application, the information collected from the client via HTML forms is channeled through the middle tier prior to being directed to the third tier where the database resides.

1.4.2 Second Tier - Web Servers and Middleware Programs

Web servers such as the Java Web Server, or JWS, and the actual application programs make up the middle tier. The first tier is connected to the middle tier via the HyperText Transfer Protocol (HTTP). It is the web server that hosts the HTML pages and the middle tier programs (middleware) to provide services to the client.

Any interaction with the web server starts with a client's request for an HTML page or a particular server side application program. For example, the following URL requests the "search" page from the web server "server 1" with port address 8080.

Http://server1:8080/search.html

In order for the server to locate the search form, search.html, it searches a designated directory for the HTML documents. During the web server installation, this directory is created and, by default, it is named as public_html. If the server locates the document, it sends it back to the client's browser via the HTTP protocol.

A client can request a server-side program execution by providing the program name with a URL or clicking on a designated area on a web page. For example, in the search page, fig 1.5, the submit button activates the search servlet. The browser sends the URL, http://server1:8080/servlet/search?searchText=John&Field=userId, to the server after the user enters "John" in the text field, clicks on the last name radio button and clicks on the search button. Figure 1.6 illustrates this process. At this point, the server calls the search servlet and passes the parameter search key and the search field. The middleware program, the search servlet, performs its task of connecting to the database

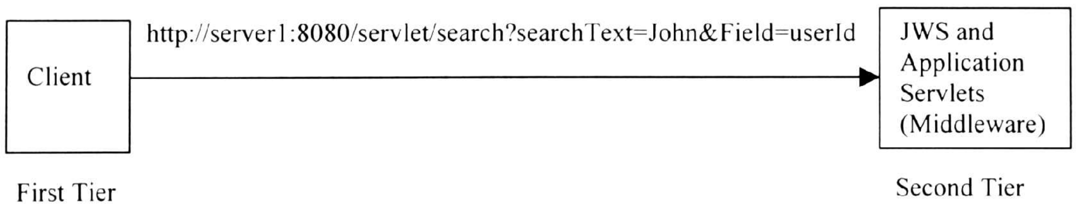

Client
http://server1:8080/servlet/search?searchText=John&Field=userId
JWS and Application Servlets (Middleware)
First Tier
Second Tier

Fig. 1.6 HTTP Requests from client to the middle-tier.

and requesting results from it with SQL statements. The DBMS receives the SQL statement as shown in the figure 1.7.

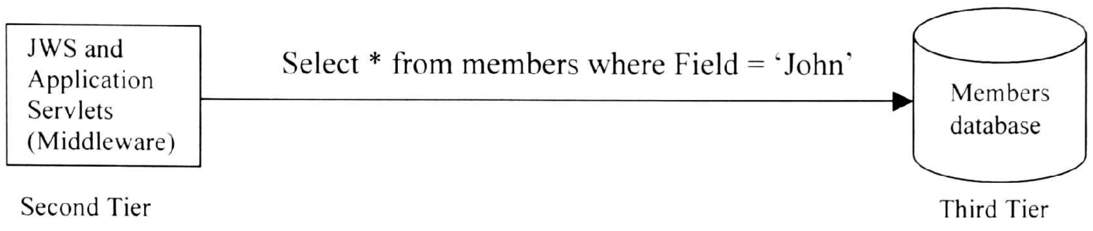

JWS and Application Servlets (Middleware)
Select * from members where Field = 'John'
Members database
Second Tier
Third Tier

Fig. 1.7 SQL requests from middle-tier to third-tier.

In this example, the middleware is a Java servlet program. Java servlets are server-side programs that are written in the Java language. Typically, those middleware programs are referred to as "gateway" programs, and the developers of gateway programs can use any language such as C, C++, or VisualBasic as long as the host computer supports them. The web server, or HTTP server, communicates with these gateway programs with a mechanism called Common Gateway Interface, or CGI. Middleware programs other than Java servlets use this interface to pass information back and forth to the HTTP server. Figure 1.8 shows the internal structure of the middle tier. The gateway programs that use the CGI are also called CGI programs.

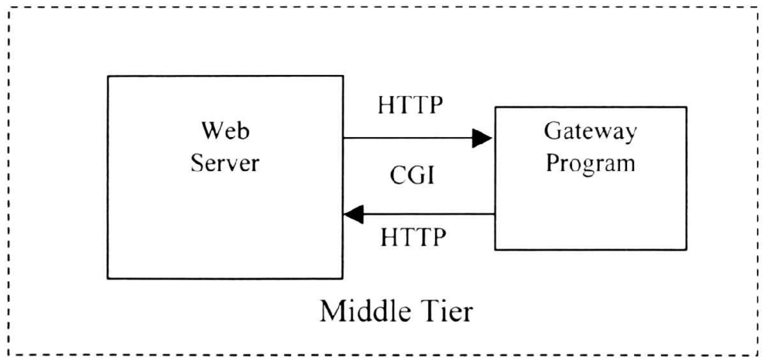


Fig. 1.8 Middle tier architecture.

The CGI programs are "stand alone" programs, unlike Java Servlet programs. Any Java-enabled web server invokes servlet programs as method calls. A single instantiation of a servlet can serve multiple clients whereas a new copy of CGI program must be loaded with each request from clients. Java servlets are much more efficient than CGI programs in this respect. For example, in figure 1.9, client 1 and client 2 request the same CGI program A and the web server loads a new copy for each client. The CGI programs execute independent of the web server. Therefore, the web server does not have any control over what a CGI program can do to the system. An ill-behaved CGI program can cause system crashes since a CGI program execution is not limited to designated areas in the system memory.

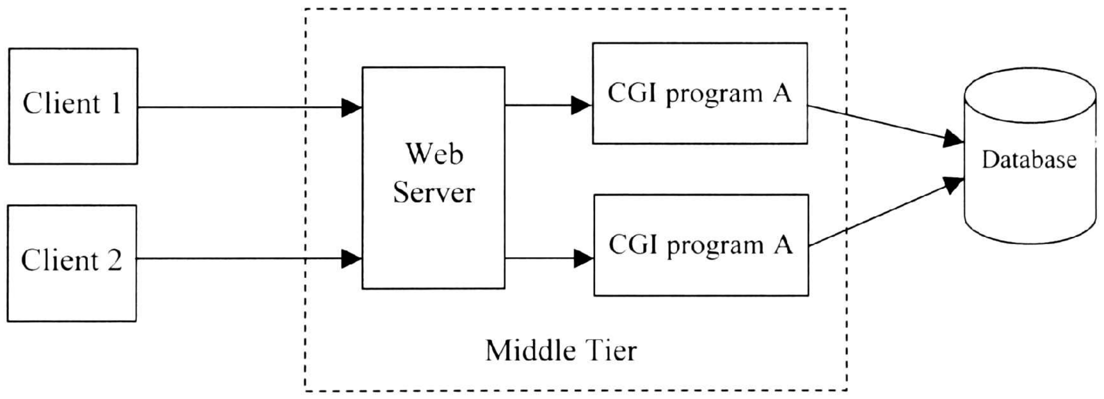


Fig. 1.9 Multiple clients requesting the same CGI program

Java middleware programs need a Java enabled web server or Java's own web server called Java Web Server, or JWS, for execution. In terms of software components, the JWS and a servlet are two separate components but they work together as one unit after the JWS loads that servlet as its extension. A servlet is component that performs a specific task within an application. However, its execution takes place within the JWS environment. Java servlets, although offer rich features for middleware application development, are limited to how they can access system resources as well as what they can do at the operating system level executions. The Java enabled server loads a single copy of a servlet in a safe area in the memory and that copy of that servlet serves all clients that call that servlet. Figure 1.10 illustrates the scenario with servlets.

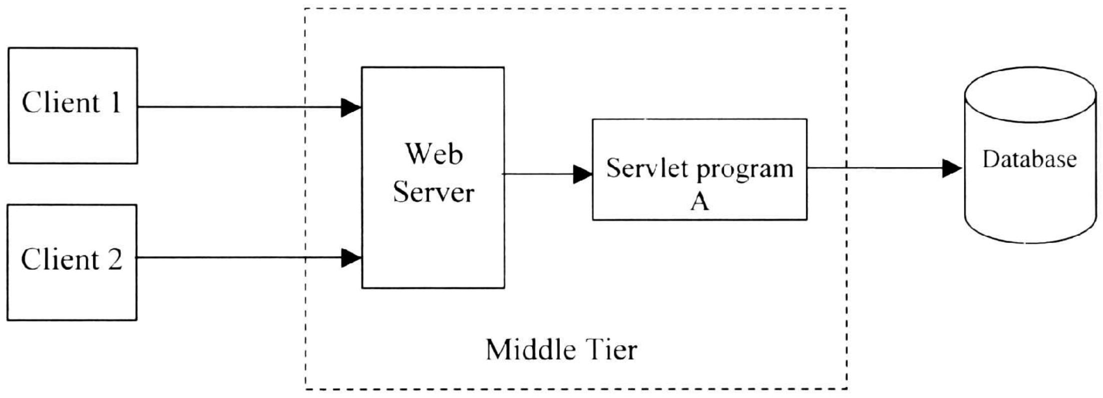


Fig. 1.10 Multiple clients requesting the same servlet program.

There is further discussion about servlets in chapter 3 and advantages of servlets over CGI programs. Further details on gateway programs and CGI are beyond the scope of this paper.

In summary, the middle tier of a three-tiered web database application includes a web server, or HTTP server, and a set of executable programs called middleware applications. Developers use mostly C, Java, C++, VisualBasic, or PERL to write the applications to serve the client requests to interface with the database in the third tier.

The example in this project is a three-tier system. The client side application is merely a web browser which provides the environment to display the application interface. The middleware application is set of Java servlets and HTML pages. The last and the third tier in this system is the MS Access that that acts as the backend DBMS for the database. The middleware application interacts with the MS Access database through the JDBC interface and the JDBC-ODBC driver. The next chapter explains the JDBC interface in detail.

1.4.3 Third Tier - Backend Databases

A web database application facilitates user interaction with a remote database on some database server computer. The term "backend database" refers to these databases. As previously discussed, a client on the Internet or an Intranet network uses his/her browser to connect to the middle tier, which accesses the database in the third tier. A DBMS at the third tier responds to the various requests from the middle tier applications. Clients utilize these middleware programs to interact with this backend database.

Most DBMSs respond to Sequential Query Language, or SQL, commands via some interface. The middleware applications use these SQL commands to interact with databases. In a web database application, user interface is an HTML form with text fields and selection boxes. It is the middleware program that extracts the information from that form and sends it to the DBMS after embedding the information in some SQL statement.

For example, the search servlet, after extracting the data from the URL

http://server1.8080/servlet/searchServlet?SearchKey = John&SearchField = LastName

it generates the SQL command,

select * from members where LastName = John

to send to the MS Access database.

There are components that facilitate connections to databases from the middleware application programs in the middle-tier. The interfaces of these components provide conventional methods for the middleware programs to talk to databases by translating the SQL requests into database specific codes. The next chapter explains the components Open Database Connectivity, or ODBC, and Java Database Connectivity, or JDBC, in further detail.

In summary, the job of developing a large application can be simplified by putting software components together rather than creating it from scratch. A good example to illustrate this process is a web database application. A web database application provides clients means to access remote databases over the Internet or intranet networks. These databases are relational databases and there are three ways clients can access them. When a client accesses a database directly, this type of access method is a single tier database access system. When a user connects to this database from a computer on the network via a client side application interface, then this type of architecture becomes a two-tier access system. Lastly, if a client, or user, connects to the database via a middle tier , then this system is said to be a three-tiered access system. Web database applications are typically three tiered systems that there is a middle tier between the client side and the remote database. The client side, as the first tier, is a web browser that connects to the web server in the middle-tier. The web server calls middleware programs, such as Java servlets or CGI programs that serve client requests by establishing connections to backend databases in the third tier. The components, web browser, web server, middleware programs, connectivity components, and backend database work together to make up a web database application.

CHAPTER 2

DATABASE CONNECTIVITY COMPONENTS

This chapter discusses how middle tier programs connects and interacts with a backend database via database connectivity components. By using these components, developers can avoid including database-specific details in their program design. Applications make connections to databases or perform read/write operations on databases through these interfaces by issuing method calls. The connectivity components translate these calls for a particular database by using the appropriate driver for that database.

Such components give database applications compatibility for wide range of database systems. In other words, the same application, for example, can be used with both the MS SQL DBMS and the Oracle DBMS without making any changes to the middle tier program. There are currently two components that are in the market for middleware programs to access backend databases. These are Microsoft's Open Dabatase Connectivity, or ODBC, and Sun Microsystem's Java Database Connectivity, or JDBC. The Java applications must use the JDBC and those applications that support C language libraries must use the ODBC for connecting to databases. However, the backend database systems must comform to the ODBC standards for those applications written in C, C++, or Visual Basic to use the ODBC component for connectivity. On the other hand, Java programs can use the JDBC to connect with JDBC compliant database systems.

There is also a JDBC-ODBC bridge that facilitates Java applications connecting to ODBC compliant databases if there is no pure JDBC driver available for that database.

2.1 Microsoft's ODBC Component

2.1.1 Background

Microsoft's Open Database Connectivity (ODBC) is an alternative to SunSoft's JDBC. These database interfaces are Call Level Interfaces (CLI), which provide access mechanisms for SQL databases by using a set of function calls. Without these CLIs, applications must directly talk to the database engines by calling special functions that are unique to that database engine. This approach to database application development has been common, but these applications support only a single DBMS. For example, an application that is designed for Microsoft's Access DBMS works only with MS Access database. This application must be modified in order for it work with Borland's InterBase database engine. As long as there is certainty that neither the database engine nor the database specifics are subject to change someday, an application can be developed specifically for a particular DBMS. However, it makes more sense to develop applications which are independent of any particular database platform.

The ODBC specifications define function calls that are accepted by the industry as standard. By using these functions, a database application can perform tasks on a given ODBC compliant database. Since this application does not incorporate any of the specifics of that DBMS, it can be used for any DBMSs as long as they are ODBC compliant.

2.1.2 ODBC Description

ODBC defines a common interface for function calls between the user application and the driver manager.

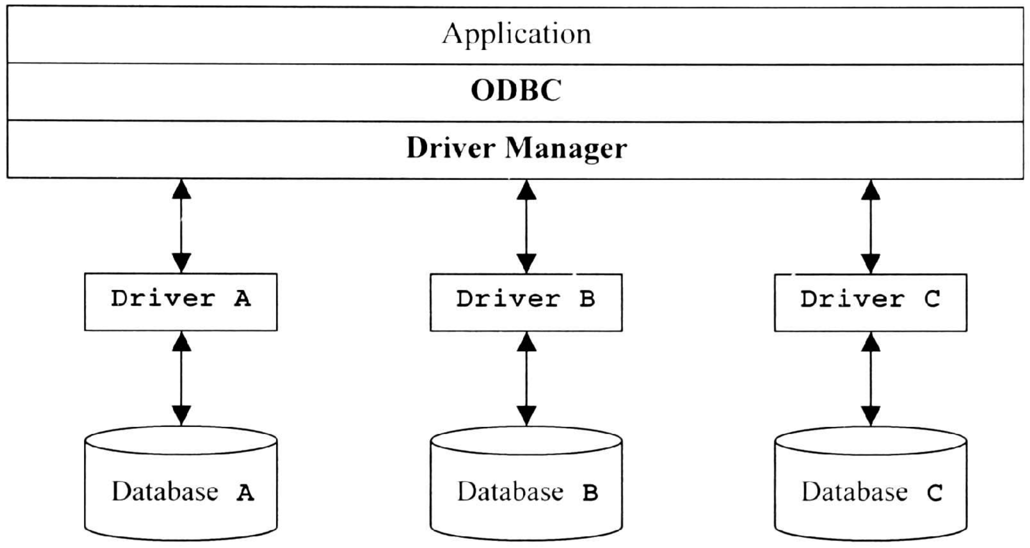


Fig. 2.1 ODBC system architecture.

The ODBC Application Programming Interface consists of a driver manager and one or more ODBC drivers. These drivers provide a mechanism to translate instructions passed from the application through the driver manager to database specific instructions. As shown in Figure 2.1, the ODBC works between the application and the database as an interface for the database. The application makes calls to the ODBC instead of making direct calls to the database. The process starts with an ODBC compliant application making calls to the driver manager with some data source access request. The driver manager first locates the appropriate ODBC driver and channels the request to that driver. The driver receives the request and converts it to a SQL statement. The DBMS specific details come into the picture here in that the driver knows exactly how to pass the SQL statement to this DBMS. The driver sends the request and receives a response from the DBMS. The response is either a set of rows of data or an error exception. The driver at this point translates the returned results and passes them back to the driver manager, which in turn forwards it to the application.

ODBC drivers are classified as one of the following two types [2].

- Single-tier drivers translate SQL statements into low-level instructions that operate directly on files. Single-tier drivers are required for relational DBMSs that don't process SQL statements directly. The widely used PC RDBMSs fall into this category, such as dBase IV or FoxBase.
- Multiple-tier drivers process ODBC instructions but pass SQL statements directly to the data source using SQL syntax that is acceptable to the back-end RDBMS. All popular client/server RDBMSs that can run on PCs and most mini and mainframe RDBMSs process SQL statements directly, such as MS SQL Server or Oracle.

2.2 Sun's JDBC Component

2.2.1 Background

Java's simplicity, robustness, security features and platform independent nature continue to attract developers to use it to develop database applications. SunSoft introduced Java Database Connectivity Application Programming Interface, or JDBC

API, for easy database access as the Java language made its way into database application development. The JDBC API was released in June 1996 the first time as version 1.0, and currently JDBC API version 2.0 is available from SunSoft.

The JDBC API defines a common low-level API which supports basic SQL functionality. As mentioned earlier, the JDBC is a Call Level Interface and it is based on the X/Open SQL CLI specifications. In its development, SunSoft used Microsoft's ODBC as a model for the JDBC, and therefore the JDBC offers the same functionality as the ODBC interface. In order to gain developer support and acceptance, SunSoft adopted the ODBC design principles for its JDBC interface.

There are currently ODBC drivers for almost all database management systems. Those vendors who developed ODBC drivers for their database engines started developing JDBC drivers as well. Although not as popular, JDBC as a database connectivity component is becoming a major player in the field. Java's wide acceptance in web database application development is especially encouraging vendors to write Java-compatible drivers in order to be on the leading edge in the market. Java applications cannot use the ODBC directly since it is a C language interface. To remedy this situation, there are JDBC-ODBC bridge drivers available. This way, a Java application can connect to an ODBC compliant database even if an appropriate JDBC driver is not available for that database. The JDBC and the ODBC interfaces are not interchangeable; one cannot be used in place of the other. JDBC is exclusively for Java programs, and ODBC is for the programs that incorporate C language libraries, such as Visual Basic or C++.

SunSoft provides JDBC as part of its Java Development Kit (JDK) free of charge. The JDK version 1.2 is available on SunSoft's web site to download, or it can be purchased on a CD with a nominal charge. This kit includes Java core classes, libraries, a Java interpreter, drivers and several development tools. The JDK includes all the necessary tools to develop Java database applications. SunSoft hopes to see Java and JDBC as industry leaders in database application development arena, especially in the Internet and Intranet environments.

2.2.2 JDBC Overview

The JDBC component, basically, facilitates Java applications in connecting to a DBMS to perform various SQL commands. Java database applications interface with various databases through the JDBC component and its database drivers. Microsoft's ODBC and SunSoft's JDBC offer the same functionality with their methods and interfaces. A Java application needs a Java compliant object oriented database interface and that is exactly what JDBC is.

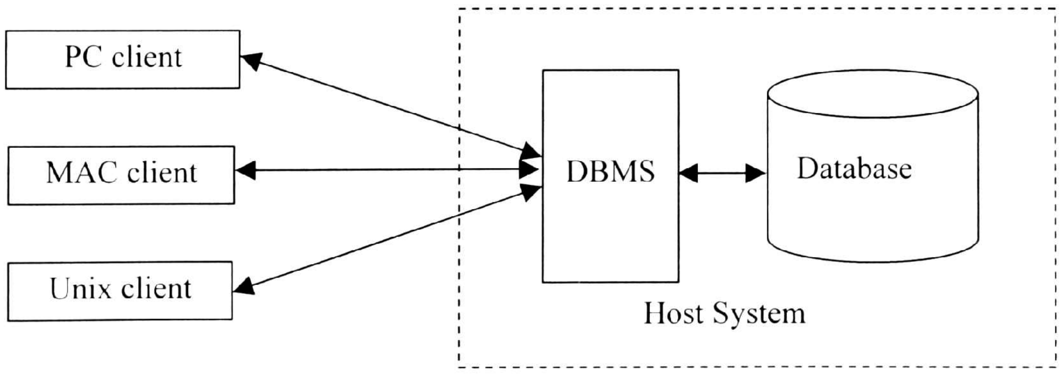


Fig. 2.2 JDBC applications are platform independent.

A connection to a specific database requires a specific JDBC driver for that database. However, unlike platform dependent ODBC applications, JDBC applications are compiled once to be used on different platforms such as PC, Mac, or Unix as the figure 2.2 depicts.

JDBC characteristics can be outlined as follows [3].

- **A SQL level API** - JDBC is a call level SQL interface for Java. Although, it is a low level interface, it is usable by programmers. Higher-level APIs can be built on top of this base level.
- **SQL Conformance** - JDBC allows any query string to be passed through to an underlying DBMS driver. JDBC drivers must support ANSI SQL 92 to be called "JDBC compliant".
- **JDBC must be implementable on top of common database interfaces** - ODBC is, especially, supported for this reason.
- **Consistency with the rest of the Java system** - JDBC interface is built on and reinforces the style and virtues of the existing core Java classes.
- **Simple** - JDBC provides a single mechanism to perform a particular task. The API may be extended if needed.
- **Strong static typing** - JDBC API is strongly typed for more error checking at compile time.
- **Common cases simple** - SQL statements such as SELECT, INSERT, UPDATE, or DELETE are the most common cases and they are simple.
- **Uses multiple methods to express multiple functionality** - This approach from core Java classes is extended to JDBC design. Using multiple methods, over multi-purpose methods with many flag arguments, simplifies JDBC.

The JDBC architecture resembles ODBC architecture in that it resides between a database application and a database. Figure 2.3 shows the JDBC architecture as well as how JDBC can utilize ODBC drivers.

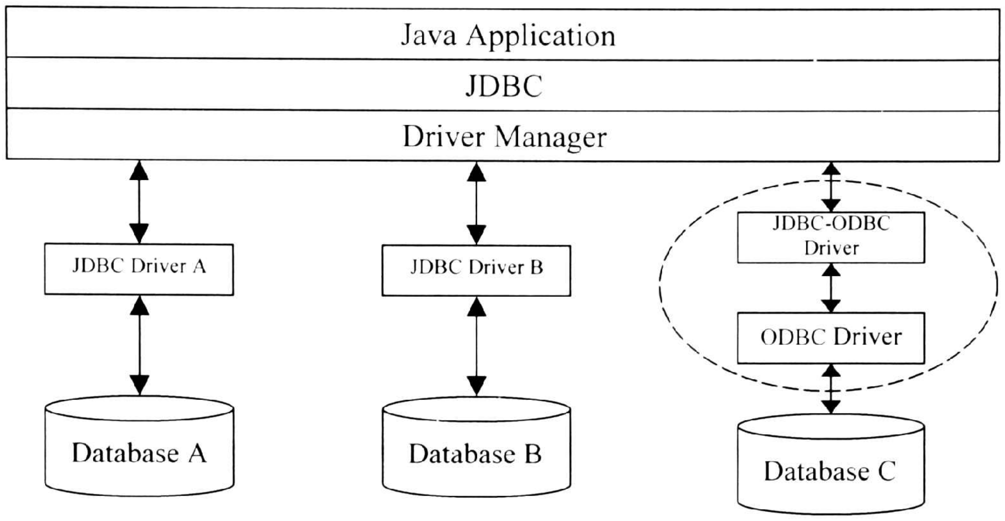


Fig. 2.3 JDBC system architecture.

In a typical JDBC application, the process starts with loading the JDBC driver. By calling the method *Class.forName*, the driver manager searches a list of drivers until it locates the right driver. Once the driver is loaded, the process continues by placing a connection request with JDBC's *DriverManager.getConnection* method. During a connection to the database, the JDBC driver channels the SQL statements from the application and the results from the database. For each datasource, a separate connection statement is needed. The following code establishes a connection to a database called "members" with a user name “admin” and “pass” as the password.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection(
                    "jdbc:odbc:members","admin","pass");
```

Once connections are established, the Statement class objects provide a means to carry SQL statements such as SELECT, INSERT, UPDATE and DELETE to the database as the user requests .

```
:
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM members");
:
```

The ResultSet object "rs" is where the database returns its response to the SQL statement "SELECT * FROM members" in the form of rows and columns. The executeQuery() method call to the "stmt" object sends the query request and prepares the result set object for the returned results. It is possible to get a different data type as a result than initially defined types by the application. Hence, in processing the returned results, the application must also handle possible returned error exceptions.

The final step is to close the statement object and the connection.

```
:
close.stmt();
close.con();
:
```

In summary, the JDBC application basically establishes the connection to the database or the datasource. Once the connection is established, SQL statements allow various operations on the database. The JDBC driver manager plays a major role in establishing connections to the databases. The connection statement provides driver identification and the driver manager locates the right driver based on that information. Once located, the driver makes the actual connection to the data source.

2.2.2.1 JDBC Drivers

The core idea behind using the JDBC interface is that the application stays independent of the specifics of the DBMS. The drivers perform the job of translating standard SQL statements into DBMS specific requests. Without these drivers, applications must include code to make the actual connections and other types of operations for each particular DBMS. For example, the code to make a connection to an MS SQL database server is useless if DBMS is changed to Oracle or any other DBMS. This means that reprogramming is necessary to make that application work for DBMSs other than MS SQL. The JDBC drivers establish connections, send application requests to the DBMS after necessary translations, and return results and errors back to the applications, again, after translations.

There are four types of JDBC drivers [4].

1. The JDBC-ODBC bridge provides JDBC access via most ODBC drivers. Note that some ODBC binary code and, in many cases, database client code must be loaded on each client machine that uses this driver, so this kind of driver is most appropriate on a corporate network, or for application server code written in Java in a 3-tier architecture.
2. A native-API partly-Java driver converts JDBC calls into calls on the client API for Oracle, Sybase, Informix, DB2, or other DBMS. Note that, like the other bridge

driver, this style of driver requires that some binary code be loaded on each client machine.

3. A net-protocol all-Java driver translates JDBC calls into a DBMS-independent net protocol which is then translated to a DBMS protocol by a server. This net server middleware is able to connect its all-Java clients to many different databases. The specific protocol used depends on the vendor. In general, this is the most flexible JDBC alternative. It is likely that all vendors of this solution will provide products suitable for Intranet use. In order for these products to support Internet access they must handle the additional requirements for security, access through firewalls, etc., that the Web imposes. Several vendors are adding JDBC drivers to their existing database middleware products.
4. A native-protocol all-Java driver converts JDBC calls into the network protocol used by DBMSs directly. This allows a direct call from the client machine to the DBMS server and is a practical solution for Intranet access. Since many of these protocols are proprietary, the database vendors themselves will be the primary source for this style of driver. Several database vendors have these in progress.

2.2.2.2 JDBC-ODBC Bridge

The JDBC-ODBC bridge provides ways for Java applications to connect to ODBC-compliant databases. The bridge driver translates JDBC method calls into ODBC function calls. It allows JDBC to leverage the database connectivity provided by the existing array of ODBC drivers. JDBC is designed to be efficiently implementable on ODBC, so the JDBC-ODBC bridge is the best way to use ODBC from Java. It is a joint development of JavaSoft and Intersolv[5]. Figure 2.4 illustrates that the JDBC-ODBC bridge driver works with the existing ODBC driver rather than interfacing directly with the backend database. The other types of JDBC drivers, types 2, 3, and 4, work directly with the database and don't need any intervention from the ODBC drivers.

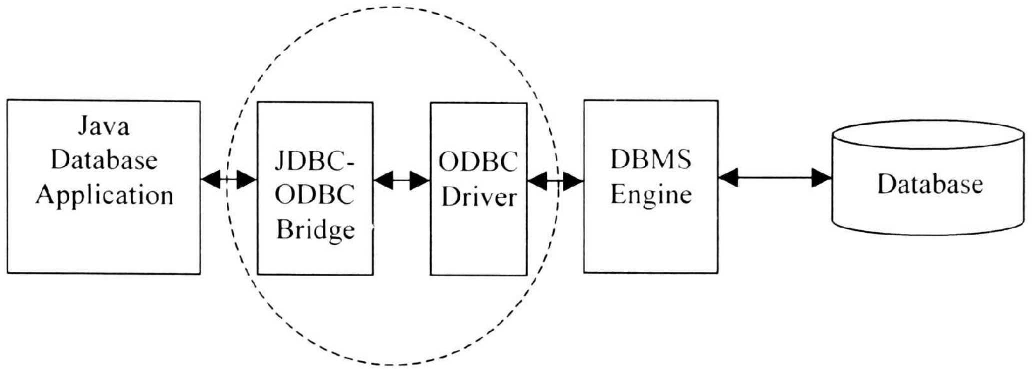


Fig. 2.4 JDBC-ODBC bridge driver.

If the backend database is ODBC compliant, a Java database application can still work with the JDBC-ODBC bridge driver as an adapter in the middle. However, this bridge driver can be considered a temporary solution to connecting Java applications until pure Java JDBC drivers become available for those ODBC compliant databases. It also serves as a cost-effective alternative to other types of JDBC drivers.

The example application in this project utilizes a JDBC-ODBC bridge driver to connect to the MS Access database. There are pure Java MS Access drivers in the market, but this bridge driver performs well for small applications.

2.2.2.3 JDBC Interfaces

JDBC provides a set of classes and methods for application development and for JDBC driver development. The JDBC API is a series of Java interfaces for application programmers to open connections to databases, execute SQL statements and process the results [2].

JDBC drivers implement the following interfaces for database application development [6].

- **Java.sql.DriverManager:** This class maintains a list of JDBC drivers that it loads during its initialization. There are methods to register drivers and to establish connections during execution. The "getConnection" method is particularly an important one in that it establishes the connection to a datasource. As its first parameter, a URL specifies the needed driver identification. Once the driver manager locates the driver, the getConnection method calls the "connect" method of java.sql.Driver to make the actual connection If the specified driver is not found in the list of drivers, it throws a SQL exception.

- **Java.sql.Connection:** This class actually represents a connection session to the desired datasource. Within the context of a Connection, SQL statements are executed and results are returned. There are methods in its body to conduct operations on the connected database such as "commit", "rollback", and "close." A Connection's database is able to provide information describing its tables, its supported SQL grammar, its stored procedures, the capabilities of this connection, etc. This information is obtained with the getMetaData method.

- **Java.sql.Statement:** A Statement object is used for executing a static SQL statement and obtaining the results produced by it. Only one ResultSet per Statement can be open at any point in time. Therefore, if the reading of one ResultSet is interleaved with the reading of another, each must have been generated by different Statements. All Statement execute methods implicitly close a statement's current ResultSet if an open one exists. Some of the most frequently used methods are "close" to immediately release a Statements's database and JDBC resources; "execute", "executeQuery", and "executeUpdate" to execute a SQl statements; "getResultSet" to receive a result set from the data source.

- **Java.sql.ResultSet**: A ResultSet provides access to a table of data generated by executing a Statement. The table rows are retrieved in sequence. Within a row, the column values can be accessed in any order. A ResultSet maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The 'next' method moves the cursor to the next row. A ResultSet is automatically closed by the Statement that generated it when that Statement is closed, re-executed, or is used to retrieve the next result from a sequence of multiple results. There are many "get" methods to capture any information from the returned results. The "next" method positions the pointer to the next row to travers the results.

- **Java.sql.ResultSetMetaData**: A ResultSetMetaData object can be used to find out about the types and properties of the columns in a ResultSet.

- **Java.sql.CallableStatement:** CallableStatement is used to execute SQL stored procedures. JDBC provides a stored procedure SQL escape that allows stored procedures to be called in a standard way for all RDBMS's. A Callable statement may return a ResultSet or multiple ResultSets. Multiple ResultSets are handled using operations inherited from Statement.

- **Java.sql.PreparedStatement:** This is smilar to CallableStatement but it is for executing prepared or precompiled SQL statements. A SQL statement is pre-compiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times. The setXXX methods for setting IN parameter values must specify types that are compatible with the defined SQL type of the input parameter. For instance, if the IN parameter has SQL type Integer then setInt should be used. Since PreparedStatement extends the "Statement" interface, it has "execute", "executeQuery", and "executeUpdate" methods to perform SQL statements.

- **Java.sql.Driver:** This class represents a specific JDBC implementation. When a Driver is loaded, it should create an instance of itself and register that instance with the DriverManager class. This allows applications to create instances of it using the Class.forName() call to load a driver. The Driver object then provides the ability for an application to connect to one or more databases. When a request for a specific database comes throug, the DriverManager will pass the data source request to each Driver registred as a URL. The first Driver to connect to the data source using that URL will be used[1]. The most important method "connect" tries to make the actual connection with the given URL. If Url is not correct, it returns null and if for some reason connection doesn't happen "connect" method returns an SQLException.

CHAPTER 3

JAVA in DATABASE APPLICATION DEVELOPMENT

Java, as a general-purpose application development language, provides excellent features for database application development. Particularly for applications that are Internet/Intranet accessible, the Java language gives total independence to both client and server platforms. Its object-oriented nature makes Java a great alternative to other object-oriented languages, such as C, C++, or Visual Basic, which rely heavily on proprietary interfaces.

Java's JDBC interface provides easy mechanisms for connecting to databases. Using the methods that the JDBC interface provides, developers design object-oriented application programs that work directly with relational databases.

Database applications that are developed with Java can run on any platform and connect to almost any database management system in the market today. Furthermore, there are ways to incorporate general-purpose template objects in application design to free the application from the actual data objects.

An Internet/Intranet Java database application can be in the form of "applets" or "servlets." A Java applet executes on the web browser on the client machine. A servlet, on the other hand, executes on the web server and it returns an HTML page to the client's browser. Although they both achieve the same results, careful considerations are necessary to choose between applets or servlets for application development.

3.1 Applets

An applet is a relatively small program that executes on the client computer after it is downloaded from the web server. The Java language made its way into the Internet/Intranet application programming language as an applet design language. The first applets appeared initially as small animating objects that were embedded in HTML pages. The browsers allowed these applets to execute and to animate an object or to respond to a user request. For example, a simple mortgage calculator, a spinning company logo, or some kind of game are common applets on the Internet today. The host browser, however, must be "Java enabled" in order to execute applets. With Netscape and Internet Explorer, enabling or disabling Java execution come as an option in their setup.

Java applets also frequently serve as client interfaces for client/server database applications. In two-tiered database applications, the client side application provides a user interface as well as most of the business logic of the application. The database connectivity issues, security issues, and the application functionality issues must be addressed by the application. Java applets work well when designing two-tiered applications.

There are a number of complex Graphical User Interface, or GUI, components available with applets that are not available with HTML input tags. Heavy usage of graphical interface tools increases the applet size, which, in turn, increases download time for the web page. Since applets execute on the local machine, the computationally intensive designs can be burden on system resources. Therefore, using applets make more sense if their advantages over servlets or HTML documents are much greater.

The following summarizes some of the advantages of applets[7].

- Applets allow local validation of data entered by user. Local validation of data is possible using HTML combined with JavaScript but variances in JavaScript implementations make JavaScript difficult to use.
- An applet can use the database to perform list of values, lookups and data validation. HTML (even if combined with JavaScript) cannot do that without invoking a CGI or a servlet program and drawing a new HTML page.
- Once an applet is downloaded, the amount of data transferred between the web browser and the server is reduced. HTML requires that the server transfer the presentation of the data (the HTML tags) along with the data itself. The HTML tags can easily be 1/4 to 1/2 of the data transferred from the server to the client.
- Applets allow the designer to use complex GUI widgets such as grids, spin controls, and scrollbars. These widgets are not available to HTML.

Since applets are actual executable programs, they pose a major security risk to the hosting client system. Therefore, applets loaded over the network are subject to various security restrictions. Although this can seem bothersome at times, it is necessary for network security. One of the major advantages of using the Java programming language is its strong security features.

The following restrictions exists for applets[8].

- Applets cannot load libraries or define native methods: Applets can use only their own Java code and the Java API the applet viewer provides. At a minimum, each applet viewer must provide access to the API defined in the java.* packages.

- An applet cannot ordinarily read or write files on the host that is executing it: The JDK Applet Viewer actually permits some user-specified exceptions to this rule, but Netscape Navigator 2.0, for example, does not. Applets in any applet viewer *can* read files specified with full URLs, instead of by a filename. A workaround for not being able to write files is to have the applet forward data to an application on the host the applet came from. This application can write the data files on its own host.
- An applet cannot make network connections except to the host that it came from: The workaround for this restriction is to have the applet work with an application on the host it came from. The application can make its own connections anywhere on the network.
- An applet cannot start any program on the host that is executing it: Again, an applet can work with a server-side application instead.
- An applet cannot read certain system properties.
- Windows that an applet brings up look different from windows that an application brings up: Applet windows have some warning text and either a colored bar or an image. This helps the user distinguish applet windows from those of trusted applications.

3.2 Servlets

JavaSoft released "Java Servlets" to develop middleware programs, also referred to as server side applications, much like CGIs. As explained in the previous section, Java applets are client side programs, which execute within a web browser environment. Java servlets, on the other hand, are Java programs which execute at the middle tier with a web server which supports servlets. Before SunSoft's introduction of servlets, programmers often used C, C++, or PERL to develop server-side CGI applications. Now, Java servlets are available to do what those languages can do to construct HTML pages with added advantages and features. Java's platform independence, advanced security, persistence, and easy database access features extend to servlets to make them superior to those other CGI development languages.

As mentioned earlier, a Java servlet is like a CGI program that executes at the server. In response to some user request on the client system, a web or HTTP server executes the servlet to perform the intended actions and returns the results back to the client in form of an HTML file. This is much like how a CGI program works. However, Java servlets are much more powerful than CGI programs for the following reasons[7].

- Servlets execute as a tread within the web server. Threaded execution avoids the overhead of creating separate processes for each CGI call.
- Servlets may retain data between executions. For example, a servlet could retain a network connection or an access counter between executions. However, cookies or similar solutions are still needed to retain data about an individual browser that accesses the servlet.
- A servlet may connect to any computer on the network or write files on the server. While CGI programs may also do these things, Java servlets allow a platform independent implementation.
- A servlet can use business objects that are part of a larger distributed system. This is difficult to impossible to accomplish with CGI.

A Java Virtual Machine, or JVM, on the server side facilitates execution of a servlet. For applet execution, the JVM must exist on the client machine. Of course, most web browsers provide a JVM in their package. Sun Microsystems's "Java Web Server", or JWS, is a servlet enabled server. Popular servers such as Netscape, Microsoft ISS, or Apache can support servlets with additional drivers, which SunSoft provides.

3.2.1 Java Servlet API

The Java Servlet API provides the classes and methods by which servlets interface with the web servers. Servlets execute as server extensions in that they become part of the web server execution process on the same host. The server, upon servlet invocation, loads the servlet and initializes it. The same server can extend its functionality with thousands of servlets if necessary. However, the servlets must be destroyed if they are not in use. The responsibility of destroying them also resides with the server. However, capability to destroy servlets is restricted exclusively to the server administrator. Servlets cannot destroy themselves or other servlets.

The javax.servlet.Servlet interface in the javax.servlet package of the Servlet API provides three major methods for servlet development:

- **init() method** - The init() method, as its name suggests, initializes the servlet. During the initialization, the server instantiates the servlet object. Implementing any time consuming tasks, such as establishing database connections or file accessing, inside the init() method increases servlet's efficiency during execution. The init() method executes once and it is not called again by the server. Its execution followed by the service() method.

- **service() method** - The service() method is basically the servlet's engine that does all the work to service the client. There are two parameter objects that this method uses to communicate with the HTTP client.
    - ServletRequest object - This represents the input from the client request. The incoming data is in the form of name/value pairs. The name identifies a particular HTTP input field and the value is the value in that field.
    - ServletResponse object - This parameter is the response object back to the client. The PrintWriter object returns the response data to the client through the ServletResponse parameter.
- **destroy() method** - A call to this method from the server releases any resources prior to unloading the servlet. Care must be taken in unloading a servlet in case the service method is still running. When the server administrator unloads a servlet, the server calls this method in that servlet.

A typical servlet application follows these steps [9].

1. The user enters information into an HTML form. The form data is passed to the Java servlet running on the web server.
2. The Java servlet parses the form data and constructs an SQL statement. The SQL statement is passed to the database server issuing the JDBC.
3. The database server executes the SQL statement and returns a result set to the Java servlet.
4. The Java servlet processes the result set and constructs an HTML page with the data. The HTML page is then returned to the user's Web browser.

The code below shows an example of a servlet structure.

```
import java.io.*;
import javax.servlet.*;
public MyServlet implements Servlet
{
        private ServletConfig config;

        public void init(ServletConfig config) throws ServletException
        {
                super.init(config);
        }

        public void service(ServletRequest req, ServletRresponse res) throws
                                        ServletException, IOException
        {
                res.setContentType("text/html");
                PrintWriter out = res.getWriter();
                out.println("<html>"+
                        "<head>+
                        :
                        "</html>);
                out.close();
        }

        public void destroy() {}

} //end servlet
```

There are no specific details in the service() method in this example code. However, the service() method parses the InputStream to extract the name/value pairs that are coming from the HTTP client. It performs desired tasks with the received information and send a response back to the client in the form of HTML text within OutputStream.

A typical CGI program writes to a standard output to create an HTML page. With servlet programs, the ServletOutputStream provides the channel to send the output back to the web browser. The following ServletOutputStream methods are commonly used for this purpose [10].

- ServletOutputStream.println(String) - Sends a character string to the browser, with a terminating CRLF.
- ServletOutputStream.print(String) - Sends a character string to the browser, with no terminating CRLF.
- ServletOutputStream.close() - Shuts down the stream.

Once a servlet is compiled with the Java compiler, javac, it is placed in a servlets directory under the JWS' default directory. After registering the servlet with the JWS, the servlet becomes available for use.

Java applets can be burdensome for the client due to their size and demands on local system sources. This is especially true for small and simple applications that normally can be implemented efficiently in other ways. Java servlets, on the other hand, provide an excellent way to produce web database applications that are more powerful and yet simpler and faster. Therefore, servlets are a far better alternative to applets for developing most common web applications particularly, three-tiered datbase applications.

3.3 RMI/CORBA

Java applets and servlets provide practical and simple ways to develop two-tier or three-tier database applications. However, they fall short of providing methods for more robust three-tier design approaches. For powerful and critical applications, Java's Remote

Method Invocation, or RMI, and Common Object Request Broker Architecture, or CORBA, are excellent three-tier design tools.

In a typical two-tier design, the JDBC interface, and all the application logic remain on the client side. The server, the database engine, responde to client requests. There is really no middle tier between the parts of the application. As business rules change, however, both the client and the server side may require changes in this approach.

The middle tier provides a solution to this problem by housing the business logic in its design. The middle tier applications are referred to as middleware applications. Both RMI and CORBA applications serve in the middle tier between the client interface and the database server to conduct tasks such as data integrity checks, data recovery, and remote object executions as illustrated in Figure 3.1.

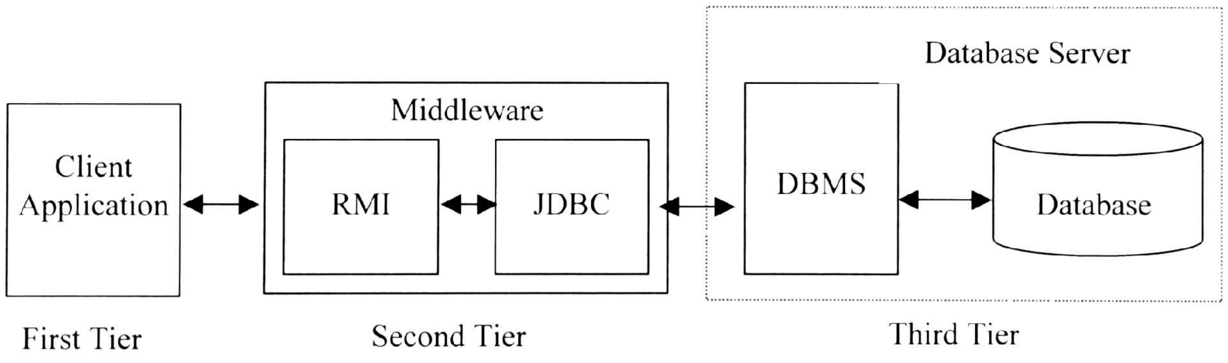


Figure 3.1 Three-Tiered RMI architecture

Instead of using sockets and streams for networking in three-tier designs, RMI provides method calls. This gives RMI a major role in three-tiered client/server applications development. RMI also supports Java objects communicating via their methods, regardless of where the objects are located.

RMI applications are usually big and involve more steps to implement them. These steps are highlighted as follows[11].

- Develop remote object code.
- Develop the server code.
- Develop the client code.
- Compile the code.
- Run the RMI compiler.
- Place .class files in directories.
- Start the registry.
- Start the server.
- Start the client.

Three-tier RMI application design and implementation is a vast topic all by itself. This section provides only a brief description, since the details are beyond the scope of this paper.

3.4 Security Issues

Java attracted the attention of developers because of its platform independence, its simplicity compared to other object-oriented languages, its robustness and its security. When the Internet started providing host computer access to millions of people across the globe, security become an extremely important issue. Java designers, with that in mind, built in security control mechanisms that made Java the language an excellent choice for developing very secure network applications.

The following items summarize some of Java's existing security features as well as the items to be implemented in the future[12].

- Safe from malevolent programs: Programs should not be allowed to harm a user's computing environment.
- Non-intrusive: Programs should be prevented from accessing private information on the host or the network.
- Authenticated: The identity of parties involved in the program should be verified.
- Encrypted: Data that the program sends and receives should be encrypted.
- Audited: Potentially sensitive operations should always be logged.
- Verified: Rules of operation should be set and verified.
- Well-behaved: Programs should be prevented from consuming too many system resources.
- C2 or B1 certified: Programs should have certification from the U.S. government that certain security procedures are included.

Java programs execute in predefined secured areas. The concept of a "Sandbox" as a designated playground gave Java designers the idea to limit the execution space in the computer's memory. This way, other areas in the memory, the file system, the web server, and the network are protected from direct access. A program must meet security clearance measures set by the execution environment before it accesses these areas. For example, Web browsers such Netscape or MS Internet Explorer allow users to define security policies for downloaded applets. If an applet does not have the proper security clearance, it can not execute in that browser. To safeguard this feature, applets, as a rule, cannot alter those security policies. Java, on the other hand, uses secured sandboxes that system administrators define.

The Java runtime environment provides several security control mechanisms in its architecture. The bytecode verifier makes sure that the program's bytecodes comply with Java language rules. The class loader loads program classes in a designated area defined by the CLASSPATH statement in the system environment setup. The security manager and the accesscontroller make decisions regarding access rights to the system resources. Figure 3.2 shows the security layer architecture for a Java program execution.

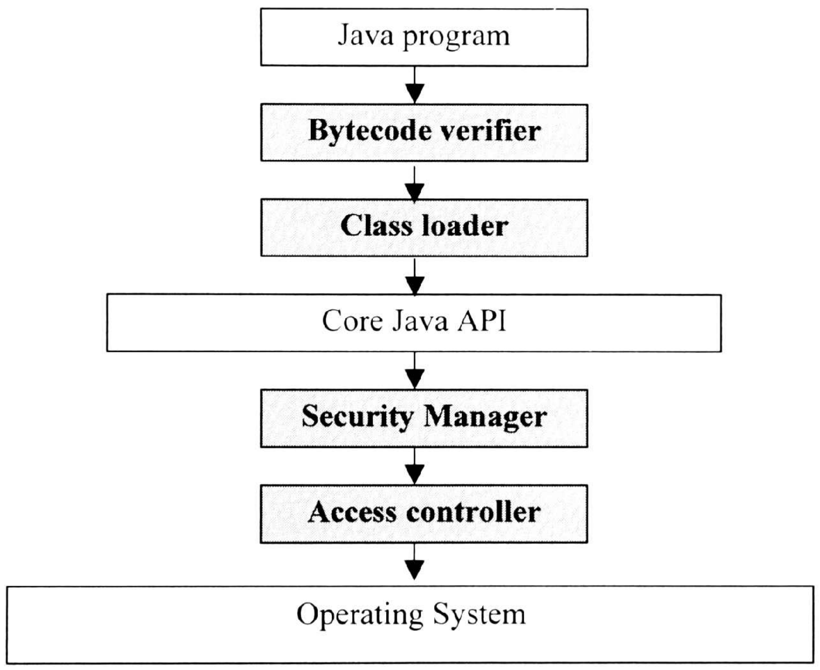


Fig 3.2 The path between a Java program and the operating system.

There is also a very complex authenticator, the security package, that deals with keys and certificates, digital signatures, and encryption. The security package works within the core Java API.

CHAPTER 4

AN EXAMPLE APPLICATION

4.1 Application Objectives

This chapter explains the example application which features the concepts and the software components that are explained in the previous chapters. The Alumni Registration Application, or ARA, will illustrate how a three-tiered web database application can be created and deployed by bringing several components together.

4.2 Application Description

The Mathematics and Computer Science department wishes for visiting alumni to be able to access the department's already existing alumni database via the department's web page. This access gives the visiting alumni registration, update and search operations on the alumni database. In order to maintain the correct information in the database, a registered person should be able to update his or her record. As a service to anyone who would like to search the database, a limited search functionality must be present as well.

Functionality of the application:

1. Membership Registration – All the information about the alumnus is collected with this function. The visitor provides name, degree, major, year of graduation, home and work addresses, and a message for the department. For future update of this

information, in order to use the update function, the visitor defines a user id and a password, as well.

2. Updating Registry – Once a membership is established with registration, updates on the membership record are performed with this function. The visitor must provide the user id and the password in order to have access to his or her record. Authenticating the user at this point gives protection to member records so that a member can modify or delete only his or her record.
3. Searching Registry – Any visitor of the Alumni page should be able to search the membership database using last name, major, graduation year, and city as search criteria.

A client, or visitor, utilizes a web browser, such as Netscape or Internet Explorer, to visit the web site of the department. A visiting alumnus or person reaches the main page of the application from a link on one of the pages on the department's web site.

4.3 Application Components

A three-tiered web application is made up of three main components: The first component is the client's browser, the second component is the middle-tier with the JWS and the application programs (middleware application), and the last component is the backend database.

The first tier component, the client browser, and the third tier component, the Microsoft Access database, are already existing components. As Figure 4.1 illustrates, the middle tier consists of two parts: the Java Web Server and the middleware application.

With the help of the middleware, all the components work together as one complete web database application.

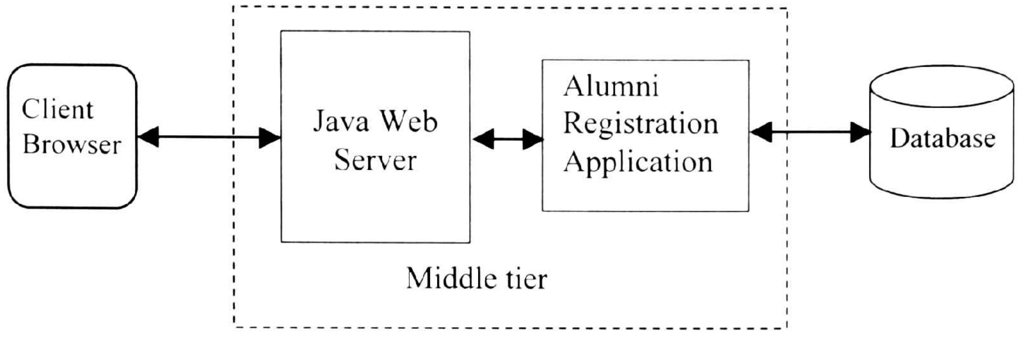


Fig. 4.1 Middle tier components.

The middleware application is also made up of several components. Depending on the detail and size of the application, there can be many servlets, HTML pages or CGI programs to provide the needed functionality for the applications. In this example, there are three Java servlets and two HTML pages that make up the middleware application. Figure 4.2 illustrates the application structure and its components.

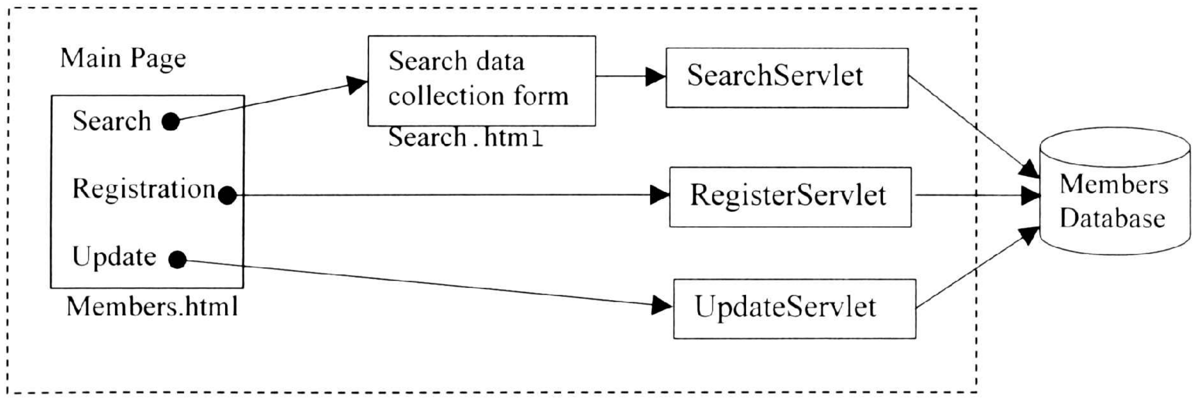
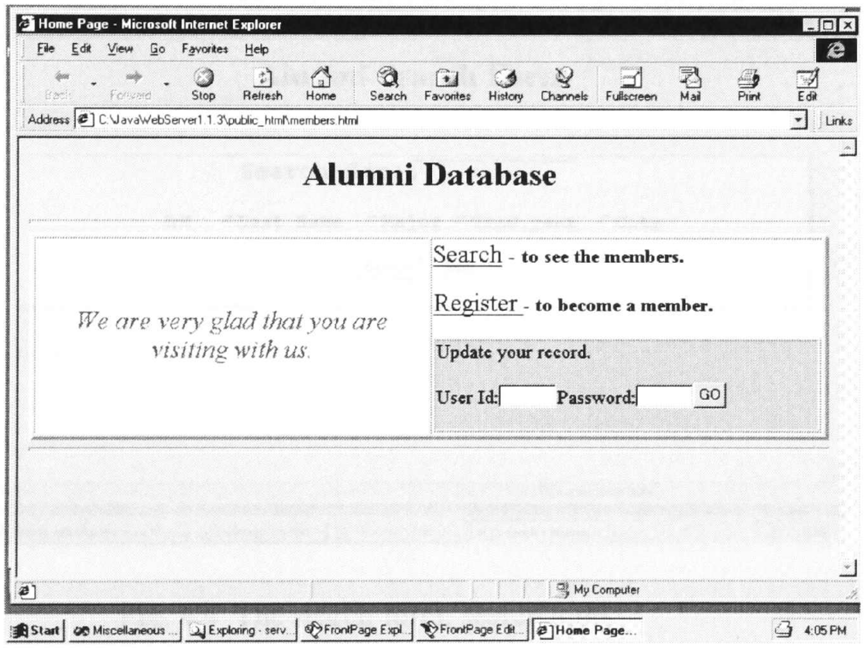


Fig. 4.2 Alumni Registration Application components.

4.3.1 HTML pages: Main Page and Search Page

The main page, named members.html, is the starting point for interacting with the ARA. Figure 4.3 shows the main page in a web browser. This HTML file is located in the public_html subdirectory under the default web server root directory. Typically, all HTML files, for the purposes of security and organization, are collected in a designated area in the server's directory structure. The JWS installation creates the public_html subdirectory automatically.


Home Page - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print Edit
Address
Links
My Computer
Start Miscellaneous ... Exploring - serv... FrontPage Expl... FrontPage Edit... Home Page...
4:05 PM

C:\JavaWebServer1.1.3\public_html\members.html

Alumni Database

We are very glad that you are visiting with us.

Search - **to see the members.**

Register - **to become a member.**

Update your record.

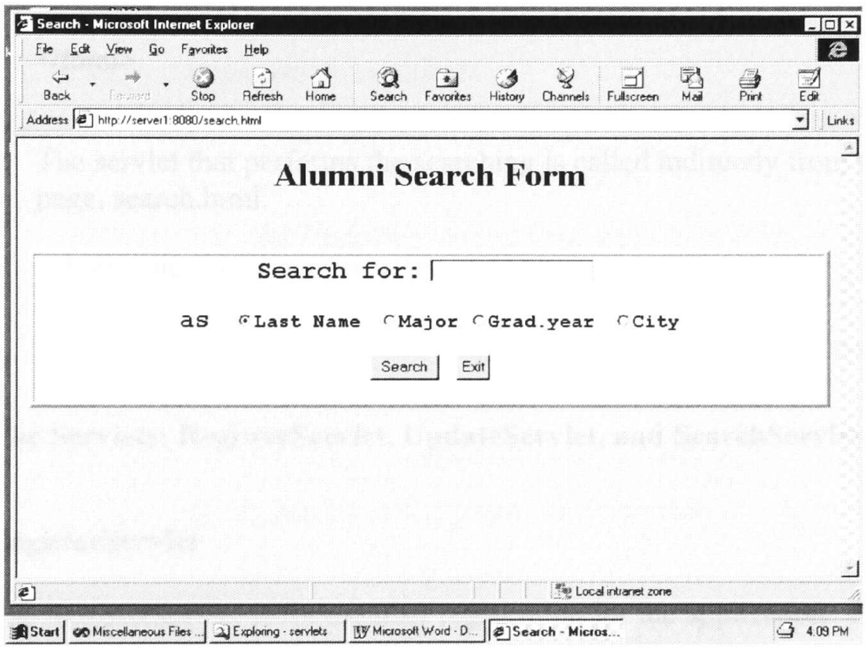
User Id: **Password:** GO

Fig. 4.3 ARA main page. (members.html)

The HTML tables and forms give the main page its appearance and organization. The main page includes a welcome message as well as a menu of functions to allow interaction with the alumni membership database.

Selecting the register link provides the user with the registration page for information collection. The update function requires user id and user password which can be entered in the provided text boxes on the main page. The "GO" button invokes the update program to give the user access to his or her record.

Searching the alumni database starts with the search link from the main page. This link takes the user to the search page where the user creates the search query by providing the text and field to search. Figure 4.4 shows a snapshot of the search page, search.html.


Search - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print Edit
Address http://server1:8080/search.html
Links
Search
Exit
Local intranet zone
Start
Miscellaneous Files
Exploring - servlets
Microsoft Word - D...
Search - Micros...
4:09 PM

Alumni Search Form

Search for:

as [✓ Last Name] Major Grad.year City

Fig. 4.4 The search form, search.html.

The design details of these pages are not pertinent at this point and their HTML source code is included in Appendix C. However, the following three points are important examples of using hyperlinks to servlet invocations from HTML pages.

1. The servlet for registration is called directly with the

   ```
   <ahref="http:servername:port/servlet/registerServlet>
   ```

   tag in the members.html file.

2. The servlet for updating function is called from a form with a button component.

   ```
   <form action="http://servername:port/servlet/updateServlet

   method="post" name="update">
   :
   <input type="submit" name="Submit" value="GO">
   :
   </form>
   ```

3. The servlet that performs the searching is called indirectly from the search page, search.html.

   ```
   <ahref= http://servername:port/search.html>
   ```

4.3.2 The Servlets: RegisterServlet, UpdateServlet, and SearchServlet

4.3.2.1 RegisterServlet

The RegisterServlet is for member registration for the application. It is invoked from the main page of the application. Initially, this servlet creates a registration form (See Appendix B, Figure B.1) with HTML tags and sends it back to the user's browser. After the user enters the data into the form, he/she submits it by clicking the submit button.

The servlet receives the entered data in the form of parameters attached to the URL. In order to place them into the member object, it parses the URL. The registration gets completed by sending that member object to the database via the SetMember method of the UserRegistryJdbcImpl class.

The RegisterServlet follows the following outline to serve client requests:

```
1)  Initialize the servlet; establish database connection
2)  Service method is called
    A)  If this is a first call:
        i)   Create a new session
        ii)  Send the registration form
    B)  Consecutive calls (with form already containing data)
        i)   Exit if exit requested
        ii)  Registration requested
             1)  Required fields are okay
                 a)  Verify user id uniqueness
                 b)  Existing user id. Send the form back
                 c)  New member. Perform registration
                     i)   Place the member into the database


                     ii)  Send confirmation and exit
             2) Incomplete form. Send the form back
```

The following explains this process in further detail:

1) Initialize the servlet.

The servlet must be initialized prior to servicing any request from the browser. The Java web server loads the servlet and initializes. This method is called once by the server, and once the servlet is loaded, it stays loaded until it is destroyed by the server.

```
Public void init (ServletConfig config) throws ServletException {

  Super.init (config);
  _registry = new UserRegisterJdbcImpl ( );
}
```

The servlet calls super.init (config) to initialize itself and to log the initialization with the JWS. The next line instantiates the UserRegistryJdbcImpl class. The reason for doing this here is that a single instance of that class serves all instance of the RegisterServlet class. Since that class establishes database connection and compiles prepared statements, it requires more time to load. It is a good idea to place such time-consuming operations in the servlet initialization.

2) Service the user requests.

In the service method all the functionality of this servlet exists. The service method does registration with these steps:

A) If this is a first call to this servlet

i) Create a new session

A session object allows for maintaining state information during the client's interaction with the HTML pages through the browser. Typically, the web pages are stateless, i.e., once a servlet program creates a page and sends it back to the browser, any data related to that page is no longer available to the program. In order to retain some or all of that information, some state tracking mechanism is needed. The session object maintains any information stored in it, as long as it is not destroyed exclusively. For example, if the client submits the form with missing data such as user id or password, the servlet sends the form back with existing data in the fields. It uses session tracking to identify the HTML pages so it can receive and send data to the right pages.

```
String ticket;
HttpSession session = request.getSession (true);
ticket =  (String) session.getValue ("ticketNo");
```

The ticket object represents this session. The session id is assigned to a session variable called ticketNo.

```
:
Session putValue ("ticketNo", ticket);
:
```

ii) Send the registration form to the browser.

```
:
SendRegistrationForm (request, response, message);
:
```

The request parameter provides the received values from the form. Initially, since it is a blank form, there are no values associated with the fields of the registration form.
The response parameter represents the output stream that sends the form back to the browser.
The message parameter provides a text string with a message to be sent back to the browser.

B) If this call to the servlet is not the very first, perform insertion into the database.

At this point the registration form contains member's data.

i) If user decides to exit before requesting registration.

1) Close the session.

```
:
session.invalidate ( );
:
```

2) Send the main page back to the browser.

```
:
response.sendRedirect ("/members.html");
:
```

The servlet completes its service at this point.

ii) If the user requested registration

1. Check the required fields, user id, and password for validity.

a) Search the database to see if this is a pre-existing user id.

```
:
AlumniMember members[ ] =
_registry.getMember
      (request.getParameter ("userId"));
:
```

b) If the user id is not unique, send the registration form back to the browser with a warning message.

```
:
Send Registration Form (request, response,
                                 message);
```

c) This is a new member. Perform registration.

i) Create a new member object and assign the input values to the member object's attributes.

```
:
AlumniMember.member =
        new AlumniMember ( );
Member.setuserId (request.getParameter
                    ("userId"));
Member.setuserId (request.getParameter
                    ("password"))
:
```

Store the member object into the members database.

```
:
_registry.setMember (member);
:
```

ii) Job is complete; kill this session.

```
:
session.invalidate ( );
:
```

Send the browser a confirmation page to let the user know that the registration is complete (See Appendix B, Fig. B.2).

```
:
send confirmation (request, response);
:
```

The service method finishes its job here.

2. There are missing fields in the received form from the browser. Send a message with the incomplete form back to the user's browser.

```
:
String message="Form must be complete before
                        registration.";
SendRegistrationForm(request, response, message);
:
```

At this point, the RegisterServlet has completed its task.

There are two private methods, which are SendRegistrationForm and SendConfirmation. The following briefly explains their inner workings.

SendRegistrationForm Method: This method dynamically creates the registration form with HTML tags. Initially, field values in the form are blank. In consecutive calls to this method, it retains the existing field values in their proper areas.

1. The output stream back to the browser object requires the following code:

```
:
response.setContentType ("text/html")
PrintWriter out = response.getWriter ( );
:
```

2. Extract the field values from the received form, and initialize the field values in the new form.

```
:
String firstName = request.getParameter ("firstName");
:
```

3. If any field value is null, initialize it to an empty string.

```
:
If (firstName = null) firstName = " ";
:
```

4. Prepare the HTML form, and write it to the output with the PrintWriter object "out."

```
:
out.println ("<html><head> +
      :         +
      :         +
             "FirstName: <input type = \"text\"...
                    + firstName + "\" >" +
                    :
                    :
             "<input type = \"submit\"name = \"Action\"..
                    show: the next two lines
             "</body></html>");
```

Clear and close the output stream

```
:
out.flush ( );
out.close ( );
```

SendConfirmation Method: This method sends the user a confirmation page constructed with HTML tags. Its structure is basically like the method SendRegistration. They differ on the actual contents of the HTML page. Its code is provided in Appendix C.

4.3.2.2 UpdateServlet

The updateServlet facilitates modifications to the membership records in the members database. By using this servlet, the user can change any information on his/her record. Figure 4.5 illustrates the process of how the update servlet interacts with the other components.

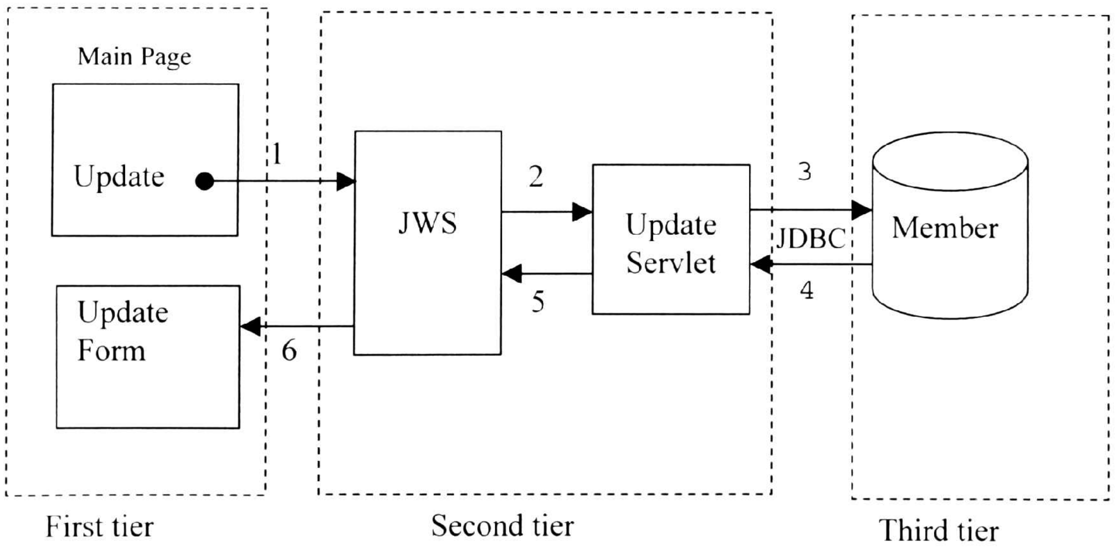


Fig. 4.5 Members database update process.

The interaction with the updateServlet starts by the user clicking on the GO button on the main page after providing the userid and the password (1). The Java Web Server, or JWS, calls the servlet with the password and the userid (2). The updateServlet connects to the members database and sends a search request with the userid (3). The database returns the search results back to the servlet. If the search result is not successful, an error message is returned; otherwise, the member's record is what the servlet receives (4). At this point, based on the return result, the servlet prepares either the update form with the membership information or an HTML page with a message about unsuccessful search. The prepared page is channeled back to the JWS (5) and the JWS sends it to the user's browser (6).

The update form is basically the same form as the registration form (See Appendix B, Figure B.3). During its the creation, the servlet places the membership information in the appropriate text fields.

As with any servlet, the JWS initializes the updateServlet before the service method starts. The servlet's structure is similar to the RegisterServlet in that it consists of two public methods: init( ) and service( ). The following outline shows this structure.

1) Initialize the servlet and establish database connection
2) Service the user requests
    - A) Create a new session if there is not one already
    - B) Initial call to the servlet
        - i) Get the userid and password
            - (1) Create a session identification
            - (2) Search the database
                - (a) Member located, password verified
                    - (i) Password verified; send the update form

(ii) Password mismatch; send a message back and exit servlet (kill session)

(b) Member userid not in records; send a message and exit servlet

(c) Problem with user id (duplicate userid); send a message: user needs to register again, exit servlet

(3) Problem with the database; send a message and exit

ii) User id and/or password not provided; send a message and exit

C) Consecutive calls to this servlet

i) User requested record update; rewrite the record to the database; send a confirmation page; exit the servlet

ii) User requested record delete; delete the record; send confirmation; exit

iii) User requested exit; send confirmation (no changes made); exit

In order to provide the client with a well-behaved web application, the servlet design must anticipate various situations. As the outline above indicates, there are several conditions that the user must be made aware of, such as a password mismatch or failure of the requested update or delete operation, etc.

The following explains the UpdateServlet in further detail:

1) Servlet initialization

The JWS calls the init( ) method to load the servlet to make it an extention for itself. The servlet is instantiated by the JWS once and that particular instantiation serves all requests to the update servlet. This way, once the servlet is initialized and loaded, the JWS directs all calls to the same copy of the servlet.
Other time-consuming operations take place during initialization, such as establishing database connection.

```
:
super.init(config);
_registry = new.UserRegistryJdbcImpl ( );
:
```

2) Service the user requests.

The service method performs all the work of receiving the user requests as well as responding them. First it checks for session information. If there is not one already, it creates a session object to maintain the state.

A) Create a new session

```
HttpSession session = request.getSession(true);
ticket = (String) session.getValue("ticketNo");
```

The value of the ticketNo variable indicates whether this is an existing session or not. The "null" value indicates that no ticket was issued to this user (or the session) previously.

B) First call to this servlet is detected by checking the ticket. If the ticket is null, then the user clicked the go button from the main page. Thus, a ticket will be issued.

```
:
if (ticket = = null)
:
```

i) Get the user id and password, and if they are not null,

```
:
String userId = request.getParameter ("UserId");
String userPassword = request.getParameter ("Password");
if (userId ! = null && userPassword ! = null)
:
```

(1) Create a session identification

The object "ticket" represents an identification tag for this session and the session id becomes its value.

```
:
ticket = session.get Jd ( );
session.putValue ("ticketNo", ticket);
:
```

(2) Search the database by calling the getMember method and place the returned results in the members array

```
:
AlumniMember members[ ] = _registry.getMember(userId);
:
```

(a) If the userid is in the database, then the member record is found. Now, authenticate the password.

```
:
if (userPassword.equals(member.getpassword( ) );
{
:
```

(i) The user supplied password matches what is in the member record, so send the browser the update form with all the information in the record.

```
:
send UpdateForm (request, response, member);
:
```

(ii) Password did not match the record. Send a message and kill the session.

```
:
session.invalidate( );
String message = "This password is not correct.";
SendProblemMessage (request, response, message);
:
```

(b) If the member is not in the database, i.e., the members array length is 0, send a message and kill the session (fig. B.5).

```
{
session.invalidate ( );
String message = "This userId is not in our records!";
sendProblemMessage (request, response, message);
}
```

(c) There is a problem with this user id. There is more than one occurrence of this userid in the database. The database administrator must resolve this problem.
The user is encouraged to re-register with a different user id. The session is killed after sending the message.

```
{
session.invalidate ( );
String message = "Please register again with a different
    userId.";
sendProblemMessage (request, response, message);
}
```

(3) There is a problem with the database. The servlet receives a SQL statement error (SQLException) from the database. The session is killed, and a message is sent back to the user.

```
        {
            e.printStackTrace ( );
            session.invalidate ( );
            String message = "There is a problem with our database.";
            sendProblemMessage (request, response, message);
        }
```

ii) The user left either the userid and/or the password field blank and clicked the "GO" button. These two fields are necessary to locate the member's record and to validate the password. Send a message and kill the session.

```
{
session.invalidate ( );
String message = "We need your userId and password.Try again!";
sendProblemMessage (request, response, message);
}
```

C) The servlet was triggered from the update form—not from the main page

i) User clicked on the update button to record the changes in the database. Get the values from the form and store them in a member object before sending that to the database.

```
{
AlumniMember member = new AlumniMember ( );
member.setuserId (request.getParameter ("userId"));
member.setpassword (request.getParameter ("password));
:
_registry.setMember (member);
session.invalidate ( );
String message = "Your record has been updated.";
sendConfirmation (request, response, message);
}
```

After updating the record, a confirmation page (See Appendix B, Fig. B.4) informs the user that the update was successful. The service method exits after killing (invalidating) the session. If a problem arises with the database, a message informs the user.

```
{
e.printStackTrace ( );
session.invalidate ( );
String message = "There is a problem with update operation.";
SendProblemMessage (request, response, message);
}
```

ii) User requested delete operation by clicking the delete button on the update form. The servlet calls the deleteMember method of the database engine, which in turn sends the proper SQL command to perform the operation. After deleting the record, a confirmation page with a proper message completes the action, and the session is terminated.

```
{
String userId = request.getParameter ("userId");
AlumniMember member = new AlumniMember ( );
member.setuserId (userId);
_registry.deleteMember (member);
session.invalidate ( );
String message = "Your record has been deleted.";
sendConfirmation (request, response, message);
```

If delete operation is not successful, invalidate the session and inform the user.

```
{
e.printStackTrace ( );
session.invalidate ( );
String message = "There is a problem with delete operation.";
sendProblemMessage (request, response, message);
```

iii) If the user clicks the exit button on the update form, the session gets completed, and again, a confirmation page is sent back with a message.

There are three public methods in the update servlet; namely, sendUpdate form, sendConfirmation and sendProblemMessage. All three of these methods are for the purpose of constructing HTML pages for the browser. The details are similar to the sendRegistrationForm method of the RegisterServlet. The Appendix C includes the actual code for these methods.

4.3.2.3 SearchServlet

The purpose of the SearchServlet, as its name suggets, is to provide a search functionality for the application. By using this function, a visitor to the alumni page can search for registered members by providing search keys such as last name, graduation year or city of residence. The HTML page, search.html, provides the necessary interface to create a desired search query. Once the user enters the text to search for and chooses the field to search in the database, clicking the seach button on the form activates the searchServlet.

There are three tiers that are involved in the search process. The first tier is the user's browser. The main page, as mentioned earlier, is an HTML page which provides links to servlets or other HTML pages. The search process flows as depicted in Figure 4.6. The "Search" link on the main page directs the user to the search page (1). This page is another HTML page with a text field for a search key and a set of radio buttons to choose a field. (See Figure 4.4). The "Exit" button on this page takes the user back to the main page of the ARA. The search button, on the other hand, calls the searchServlet to search the database (2).

The middle tier gets involved at this point. The servlet issues a SQL search statement to the database (3). The members database in the third tier returns the results of the request back to the servlet (4). The servlet prepares an HTML page with the returned results and sends that page back to the user's browser via the JWS (5).

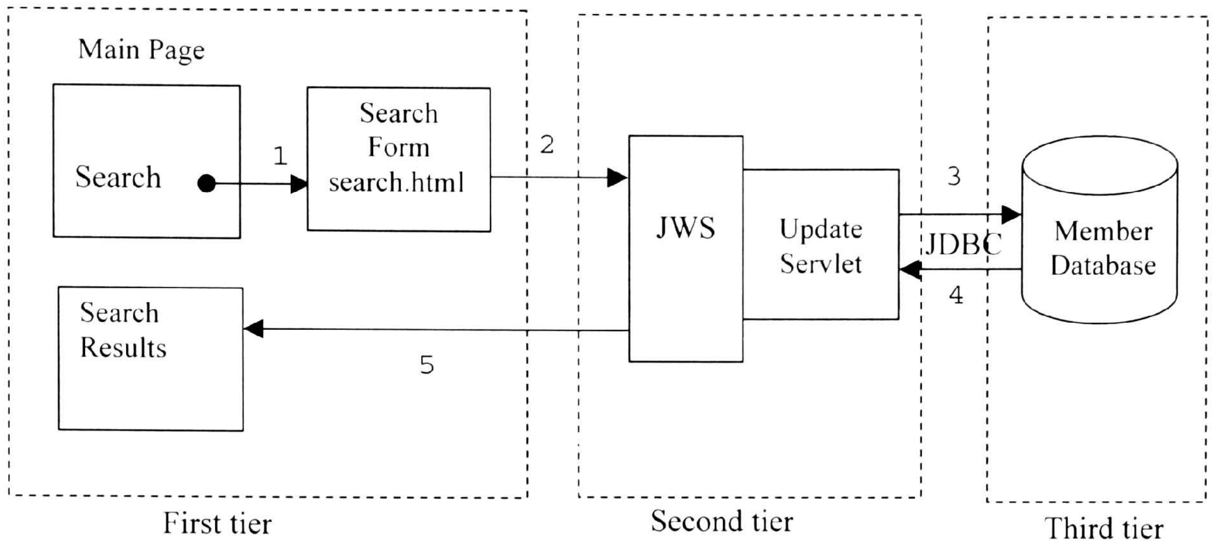


Fig. 4.6 Members database search process.

As with the other servlets, the SearchServlet is in the middle tier of the three-tiered application and is independent of the tiers on either side. The JWS deals with the browser platform and the JDBC bridge deals with the database. The program logic or even the compiled code of the servlet do not need any changes if either or both of tier 1 or tier 3 change platforms.

The SearchServlet program logic can be outlined as follows:

```
1) Initialize the servlet
    a) Connect to the database
2) Service client requests.
    a) If search requested
        i) Get the search key and the search field
            (1) Search the database
                (a) If not found send a message.
```

(b) Otherwise, return the results.

(2) Error returned from the database

ii) No search key or search field; send error message

b) If exit requested send the main page back.

The program code for the SearchServlet is shorter in comparison to the RegisterServlet and the UpdateServlet. The search process does not involve insertion, update or delete operations on the database; it is basically a read-only operation. Updating databases require care and caution to maintain the data safety and the integrity. Search operations on databases pose less risks to the data itself. However, what information is not accessible with a particular access right is the issue with searching databases. For example, the SearchServlet excludes the user id and the user password in the returned results page. Placing such security rules in the middle tier leaves the client side, tier 1, and the DBMS, tier 3, free of application details.

The actual program code for the SearchServlet and further explanations are as follows:

1) Initialize the servlet.

   As with the previous two servlets, the JWS calls this method to load the servlet. Other time consuming tasks, such as connections to database, take place during this initialization.

```
:
public void init (ServletConfig config) throws ServletException
{
        super.init(config);
        _registry = new UserRegistryJdbcImpl();
}
```

2) Service client requests.

   The service() method provides all the functionality for the SearchServlet.

a) If search is requested (user clicked on "search")

```
:
if ((request.getParameter("Action").equals("Search")))
{
  :
```

i) Get the search key and the search field.

```
:
String columnName = request.getParameter("Field");
String searchString = request.getParameter("searchString");
:
```

(1) Search the database by calling the searchRegistry() method of the _registry object.

```
:
AlumniMember members[] = _registry.searchRegistry (columnName,
searchString);
:
```

The columnName and the searchString parameters provide the necessary information for the search operation.

(a) No member is found with the given criteria.

```
if ( members.length == 0 )
{
    String message = "Can not find matching data !";
    sendProblemMessage(request, response, message);
}
```

(b) There is at least one record found.
Prepare the output stream to write back to the browser.

```
:
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String serverName = request.getServerName();
int serverPort = request.getServerPort();
out.println("<html>"+"<head>"+
            "<title>Results Page</title></head>"+
:
```

The length of the members array represents the number of the returned record from the database. The for loop generates an HTML table (fig. B.6) with rows and columns to display all the record with the matching criteria.

```
:
for (int i = 0; i < members.length; i++)
{
        out.println("<TR>");
        out.println("<TD>" + members[i].gettitle());
        out.println("<TD>" + members[i].getfirstName());
        out.println("<TD>" + members[i].getmiddleInitial());
:
```

The printWriter object gets closed after emptying its buffer.

```
:
out.flush();
out.close();
:
```

(2) Error occurred with the database.
The database returned an error condition so no search was performed. The user gets a message with SendProblemMessage() method.

```
:
catch (RegistryException e)
{
        e.printStackTrace();
        String message = "There is a problem with our database.";
        sendProblemMessage(request, response, message);
}
:
```

ii) User requested search operation without a search text and/or search field. The search form is sent back to the user with a message.

```
:
String message = "You did not enter any data to search. Try again!";
sendConfirmation(request, response, message);
:
```

b) If user clicked on Exit button on the search form, send the main page back to the browser.

```
:
response.sendRedirect("/members.html");
:
```

There are two private methods, sendProblemMessage and sendConfirmation in the SearchServlet. They basically send various messages back to the browser by placing them in HTML pages. The actual code is almost identical to those methods with the same names in the RegisterServlet and the UpdateServlet.

4.4 Design and Implementation Details

Java is a an object-oriented language. The servlet components of this application utilize the object classes and methods that are readily available from Java core API. Object-oriented design encourages more concentration on functionality of object components of an application than the actual implementation details. In creating abstractions for various classes, a designer identifies class methods, responsibilities, and relationships. The class's relationships for Alumni Registry Application can be diagrammed using Booch notation [13].

In Booch notation, as in Figure 4.7, the clouds represent classes, the open circles represent "uses a" relationship. The RegisterServlet, the UpdateServlet, and the SearchServlet use the UserRegistry class. This class is an abstract class and is differentiated with an inverted triangle. The solid circles represent "has a" relationship, so the UserRegistry class has an AlumniMember class. The arrows from the UserRegistryJdbcImpl indicate the class implementation for the UserRegistry abstraction class.

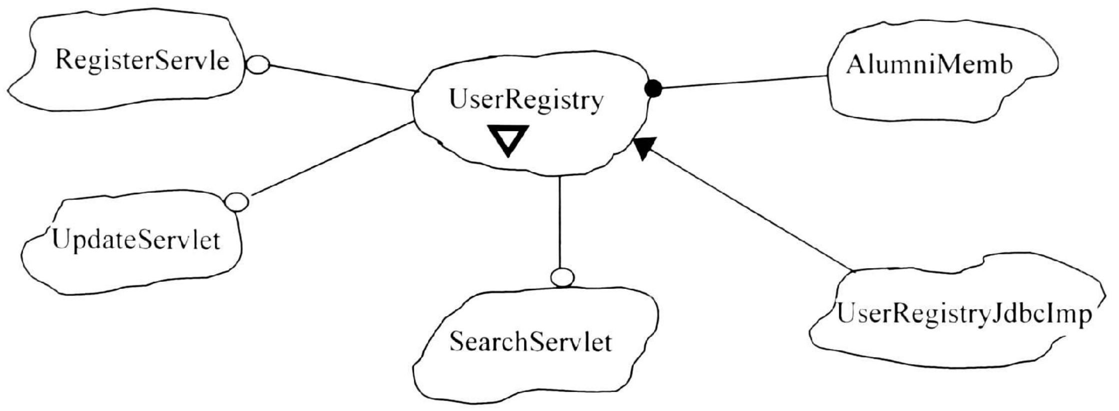


Fig. 4.7 Representation of class relations with Booch notation.

By representing the application components this way, how those components work with one another becomes more clear. Using this diagram, one can easily see that the three servlets use the UserRegistry class which serves as an interface for the class UserRegistryJdbcImpl. The UserRegistry uses the RegistryException class, and it has the AlumniMember as a subclass. The following briefly describes each class's characteristics:

- UserRegistry – provides interface for the methods getMember, setMember, deleteMember, and SearchRegistry
- UserRegistryJdbcImpl – this class contains the actual implementations of the above methods
- RegistryException – implementation class for extending more generic class called Exception
- AlumniMember – definition of data structures for the members database and implementation of set and get methods for member variables

By using an object-oriented paradigm to design this project, all the implementation specific details are compartmentalized in classes. When changes are needed in the applications, it is easier to identify the affected classes, therefore simplifying the process of maintenance. For example, all the database specific coding in this example is in the UserRegistryJdbcImpl class; the other classes don't care if the implementation for the deleteMember method changes, as long as it conforms to its interface in the UserRegistry abstract class.

The actual implementation where the database connections take place, the SQL statements for insert, delete, update, and search are constructed in the UserRegistryJdbcImpl class. Therefore, the following section explains this class in detail.

4.4.1 JDBC Implementation

UserRegistryJdbcImpl class primarily establishes the connection to the database and facilitates insert, update, delete, and search operations. There are four methods to perform those operations; namely getMember, setMember, deleteMember, and searchRegistry respectively.

Figure 4.8 illustrates the basic three stages in working with databases. First of all,

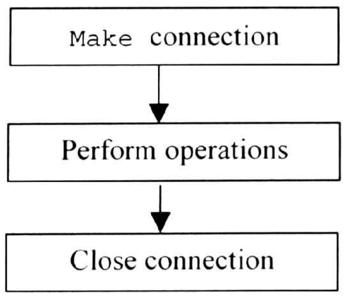


Fig. 4.8 Steps of working with databases.

a connection to the database must be established. Once a connection is available, various SQL statements allow interaction with the database. The final step is, naturally, closing the connection. Ideally, connections will be made as the need arrives, rather than keeping a connection alive for a long period of time. The process of establishing a connection is a time consuming task and frequent connection requests can create a burden on system resources.

Making a connection to a database requires defining a driver name and a database name. The database name is defined using URL syntax as follows:

```
Jdbc:<subprotocol>:<subname>
```

Jdbc indicates the JDBC protocol. The subprotocol is the name of the protocol that provides the database-dependent interface. The subname is the name of the database. Since the database is MS Access with name "members" and the ODBC interface is the driver for it, the following code defines the static variable representing the database:

```
_url = "jdbc:odbc:members"
```

The database driver name is required for a connection. For the "members" ODBC compliant database, a JDBC-ODBC bridge driver is necessary. Although there are pure Java drivers for MS Access, the JDBC-ODBC driver works well, and it comes as part of the JDK1.2 or Java2.

```
_driver = "sun.jdbc.odbc.JdbcOdbcDriver"
```

It is possible to provide more than one driver name to the JDBC Driver Manager by separating them with a colon between each driver. The Driver Manager tries each driver until it can connect to the database. After defining the driver name, it is loaded with _jdbcDriver = Class.forName(_driver) statement. The driver is placed in a static area, so all instances of the UserRegistryJdbcImpl class access the same instantiation of the driver.

The private method _init( ) creates the connection and performs initialization of the prepared statements.

```
static private void _init()
{
        synchronized (_jdbcDriver)
        {
                try
                {
                        _db = _jdbcDriver.connect (_url, _dbProperties);

                        _insertStatement = _db.prepareStatement ("insert into UTH (userId,...

                        :
                        _updateStatement = _db.prepareStatement ("update UTH set ...
                        :
                        _queryStatement  = _db.prepareStatement ("select * from UTH ...
                        :
                        _deleteStatement = _db.prepareStatement ("delete from UTH where
                                                                        userid=?");

                        }
                        catch (SQLException e)
                        {
                                e.printStackTrace();
                        }
                }
        }
```

The statement jdbcDriver.connect (_url, _dbProperties) actually establishes the connection with the given parameters. As mentioned earlier, the variable _url provides the driver name and the variable _dbProperties provides the user name and user password that are specific to the database. These values are not the same values as the member's user name and the password, which are maintained as part of the member record in the member's database.

Public Methods

getMember – This method accepts a user id as a string and searches the database for its occurrence. The returned results, member records, are stored in an array of member objects and returned. The logic of the method is as follows:

1. Check for database connection, and open connection if closed.

   ```
   If (_db.isClosed( ))
   {
       _init( );
   }
   ```

2. Prepare for search by setting the query statement.

   ```
   _queryStatement.setstring(1,userId)
   ```

   The _queryStatement is already loaded and compiled during initialization.

   ```
   _queryStatement  = _db.prepareStatement ("select * from UTH where
          userid=?");
   ```

   At this point only the value of userid is supplied with the setString method, where the parameter 1 points to the first parameter in the prepareStatement and the value of userId gets substituted where "?" is.

3. Perform the search.

   ```
   r = _queryStatement.executeQuery( );
   ```

   The JDBC interface sends the prepared SQL statement to the DBMS at this point, and the results are stored in the resultSet object "r". The resultSet is the rows of records that are returned from the members database.

4. Fill the member array with the return information.

   ```
   while there are more rows

   while(r.next( ))
   ```

```
{
 // create a new member object

 member = new AlumniMember( );
 member = setuserId(r.getString("userID"));
                               ("PASS
                               ("FIRSTNAME"));
 :
 member.addElement(member);
}
```

5. Clear the result set if it is not null.

```
if (r != null)
{
  r.close( );
}
```

6. Close the database connection if it is open

```
if (_db! = null)
{
  _db.close( );
}
```

7. Create a clone of the member array called memberArray, and return it

```
:
AlumniMember memberArray[] = new

             AlumniMember[members.size()];

members.copyInto (memberArray);

return memberArray;
```

SetMember – This method adds a new member to the database as well as replacing the existing member information with new information.

A member object as a parameter supplies all the information about the member. If setMember cannot find the member in the members database, it inserts it; otherwise, it replaces the record and returns the original record back. The steps in this method are the following:

1. Search the database with the member's userid.

   ```
   getMember (member.getUserId( ));
   ```

2. Check to see if the database connection is still there.

3. If there is exactly one member found
   a. Prepare the update statement

      ```
      :
      _updateStatement.setstring(1,member.getpassword( ));
      :
      ```

   b. Execute the update statement.

      ```
      _updateStatement.executeUpdate( );
      ```

4. If there is no existing member, insert the member record into the database.
   a. Prepare the insert Statement

      ```
      :
      _insertStatement.setstring(1,member.getuserId( ));
      :
      ```

   b. Execute the insert Statement

      ```
      _insertStatement.executeUpdate( ));
      ```

5. If the search returns multiple records, return an error message. UserId is supposed to be a unique key, and it should not have multiple copies.

   ```
   :
   throw new RegistryException("Duplicate UserId!")
   :
   ```

6. Clear prepared statement parameters

   ```
   :
   statement.clearParameters( );
   :
   ```

7. Close the connection

```
:
_db.close( );
:
```

8. Return the old member record. Although this example application does not utilize the returned value, it could be useful if later it was desired to undo the saved information.

DeleteMember – This method deletes a member from the database. As with setMember, it receives a member object as a parameter. It deletes this member if it finds it in the database; otherwise, it returns an error message. If there are multiple records with the given user id, it returns an error. Once a successful delete is performed, it returns the original record back in case a recovery is later needed.

Steps of delete method:

1. Search the database with getMember method.

```
:
getMember(member.getuserId( ));
:
```

2. Check for database connection. Open a connection if there is no connection.

```
:
if ( _db.isClosed () )
{
        _init();
}
```

3. If only one record found prepare the delete statement and execute.

```
:
_deleteStatement.setString(1,member.getuserId( ));
_deleteStatement.executeUpdate( );
:
```

4. If duplicate userids found or no userid found, return error message.

```
:
throw new RegistryException ("Duplicate or unavailable userId!");
:
```

5. Clear prepared statement parameters.

```
:
Statement.clearParameters( );
```

6. Close the connection.

```
:
_db.close( );
```

7. Return the deleted record in case it is needed

```
:
return oldMember;
```

searchRegistry – There are two parameters for this method: columnName and searchString. The searchString provides the key value to search, and the columnName provides the name of the column to search for the given key value. A successful search returns an array of member objects to the calling object.

Steps of searchRegistry method:

1. Create the SQL statement to do the search

```
:
query = "select * from UTH where (" + columnName + " like " +
                    searchString + "%')";
:
```

2. Check for database connection; if it is closed, open it.

```
:
if ( _db.isClosed () )
{
_init();
}
:
```

3. Prepare the query statement and execute it.

```
:
_searchStatement = _dbprepareStatement(query);
r = _searchStatement.executeQuery( );
```

4. If there are results returned, create an array of member objects

```
:
member = new AlumniMember( );
member.setuserId(nulltoString(r.getString("userId"));
members.addElement(member)
:
```

5. If there are no results, return an error message

```
:
String s = e.getMessage();
throw new RegistryException (e.getMessage());
:
```

6. Clear the query statement parameters

```
:
_searchStatement.clearParameters();
:
```

7. Close the resultSet

```
:
r.close();
:
```

8. Close the connections

```
:
_db.close();
:
```

9. Return the results in member array

```
:
AlumniMember memberArray[] = new
          AlumniMember[members.size()];
members.copyInto (memberArray);
return memberArray;
```

CHAPTER 5

CONCLUSION

The Internet is shaping the way companies do business, as well as the way individuals think in our society. This vast network is spanning the globe and has already penetrated the daily activities of nearly all computer users. New possibilities, opportunities and ideas are fueling the growth of the Internet, WWW, and web applications. At the same time, the software industry has grown exponentially. Software manufacturers, in order to meet the demand for the massive World Wide Web application market, must adopt new concepts and tools. Putting software components together to develop applications is the new trend in the industry.

Using components to develop applications is the core idea behind OO design concept. Although different vendors may develop the components, they provide interfaces so that the individual components can work with one another to make up a complete application. This project is about integrating software components to build a web database application with Java and Java's JDBC interface.

The first chapter discussed the three components of a web database application. The concept of three-tiered client/server application design serves as a model for the example application. The first tier is the client's web browser that interacts with the middleware programs in the middle tier. The third tier is the database, and it serves the middle tier for client requests.

The second chapter covered the single most important component between the middle application and the backend database in the third tier. The JDBC interface captures attention in the chapter since the example application is a Java application, and JDBC is what Java applications use to connect to databases.

Java language offers applets and servlets to develop applications. With a three-tier database application design, Java servlets surpass applets or even languages like C++ or PERL for efficiency and simplicity of development. Therefore, chapter three covered servlets more in depth than applets. There is also a brief description of RMI and CORBA in this chapter as Java tools for more resource-intensive application development.

Chapter four covered the example application in detail. This middleware application consists of components such as HTML pages and Java servlets. There is a detailed explanation of the inner workings of the servlets in this chapter.

The client side browser, the middleware application and the backend database make up the complete application. Each one of these components carry no dependency on the others in their internal designs; they are the "plug-and-play" parts of an application.

Appendices

APPENDIX A: The data dictionary for members database

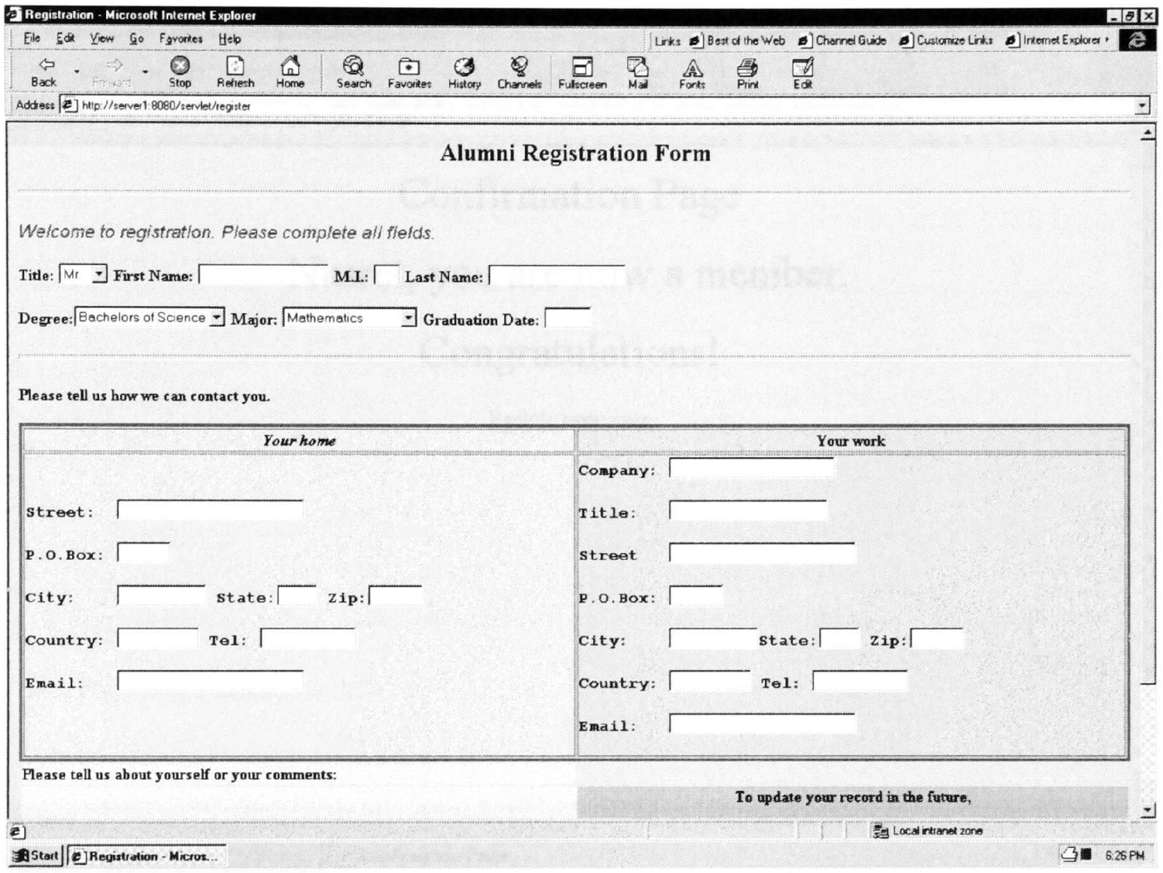
Columns

Name	Type	Size
title	Text	4
firstName	Text	20
middleInitial	Text	2
lastName	Text	20
degree1	Text	22
major1	Text	15
gradDate1	Text	8
degree2	Text	10
major2	Text	15
gradDate2	Text	8
homeStreet	Text	25
homePObox	Text	6
homeCity	Text	20
homeState	Text	2
homeZip	Text	5
homeCountry	Text	20
homeEmail	Text	50
homeTel	Text	15
company	Text	25
workTitle	Text	20
workStreet	Text	25
workPObox	Text	6
workCity	Text	20
workState	Text	2
workZip	Text	5
workCountry	Text	20
workEmail	Text	50
worktel	Text	15
comment	Text	50
userid	Text	10
password	Text	10

Table Indexes

Name	Number of Fields
PrimaryKey	1
Fields: userid, Ascending	

APPENDIX B: Application interface screens


Registration - Microsoft Internet Explorer
File Edit View Go Favorites Help
Links Best of the Web Channel Guide Customize Links Internet Explorer
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Fonts Print Edit
Address http://server1:8080/servlet/register
Local intranet zone
Start Registration - Micros...
6:26 PM

Alumni Registration Form

Welcome to registration. Please complete all fields.

Title: Mr First Name: M.I.: Last Name:

Degree: Bachelors of Science Major: Mathematics Graduation Date:

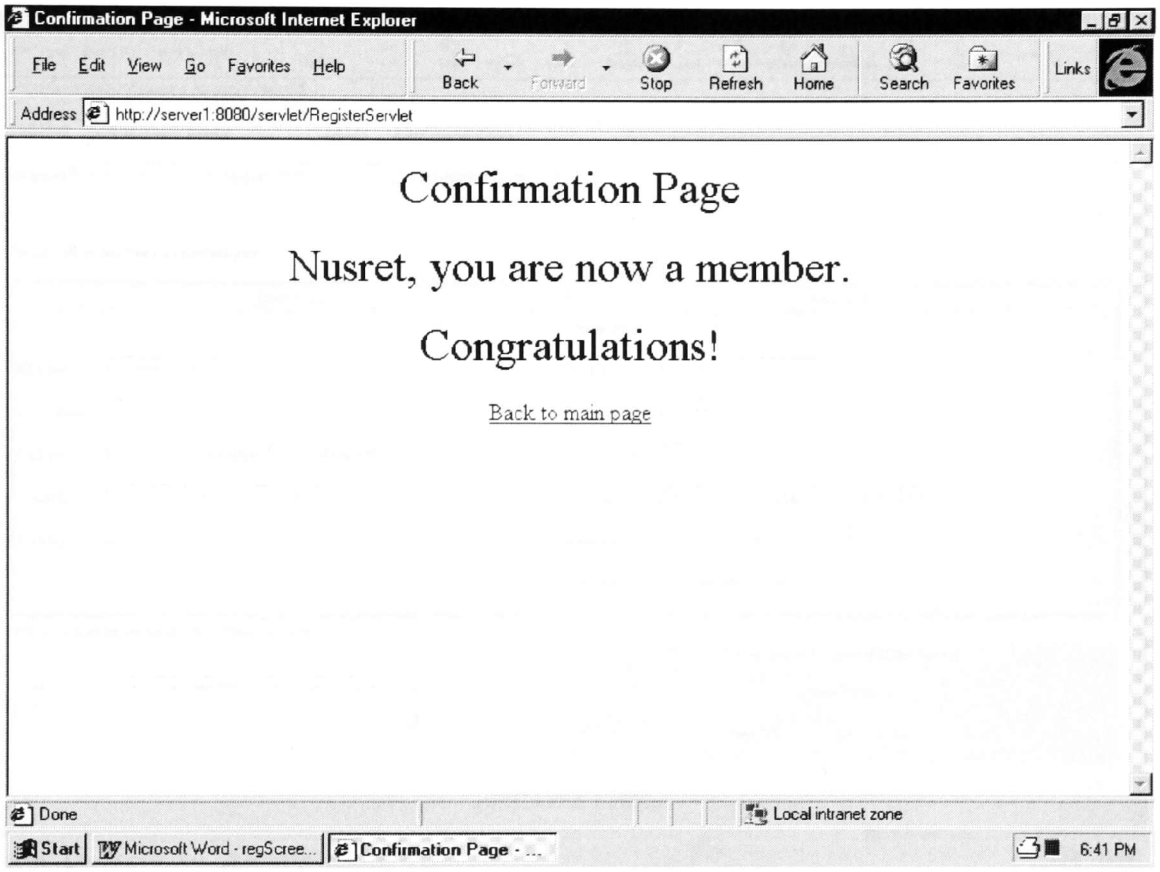
Please tell us how we can contact you.

Your home	Your work
	Company:
Street:	Title:
P.O.Box:	Street
City: State: Zip:	P.O.Box:
Country: Tel:	City: State: Zip:
Email:	Country: Tel:
	Email:

Please tell us about yourself or your comments:

To update your record in the future,

Fig. B1 Blank membership registration form.


Confirmation Page - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites
Links
Address http://server1:8080/servlet/RegisterServlet
Done
Local intranet zone
Start
Microsoft Word - regScree...
Confirmation Page - ...
6:41 PM

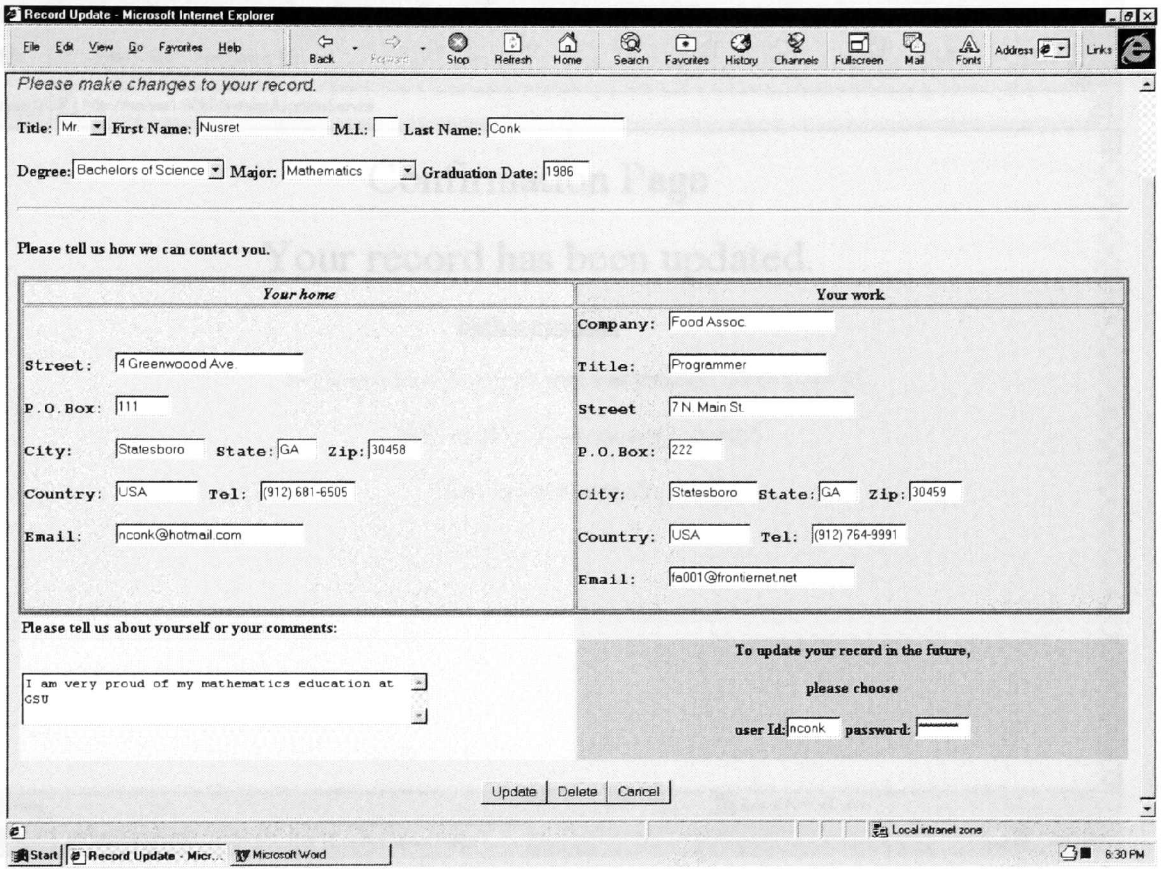
Confirmation Page

Nusret, you are now a member.

Congratulations!

Back to main page

Fig. B2 Confirmation message after successful registration.



Please make changes to your record.

Title: Mr First Name: Nusret M.I.: Last Name: Conk

Degree: Bachelors of Science Major: Mathematics Graduation Date: 1986

Please tell us how we can contact you.

Your home		Your work	
		Company:	Food Assoc.
Street:	4 Greenwoood Ave.	Title:	Programmer
P.O.Box:	111	Street	7 N. Main St.
City:	Statesboro State: GA Zip: 30458	P.O.Box:	222
Country:	USA Tel: (912) 681-6505	City:	Statesboro State: GA Zip: 30459
Email:	nconk@hotmail.com	Country:	USA Tel: (912) 764-9991
		Email:	fa001@frontiernet.net

Please tell us about yourself or your comments:

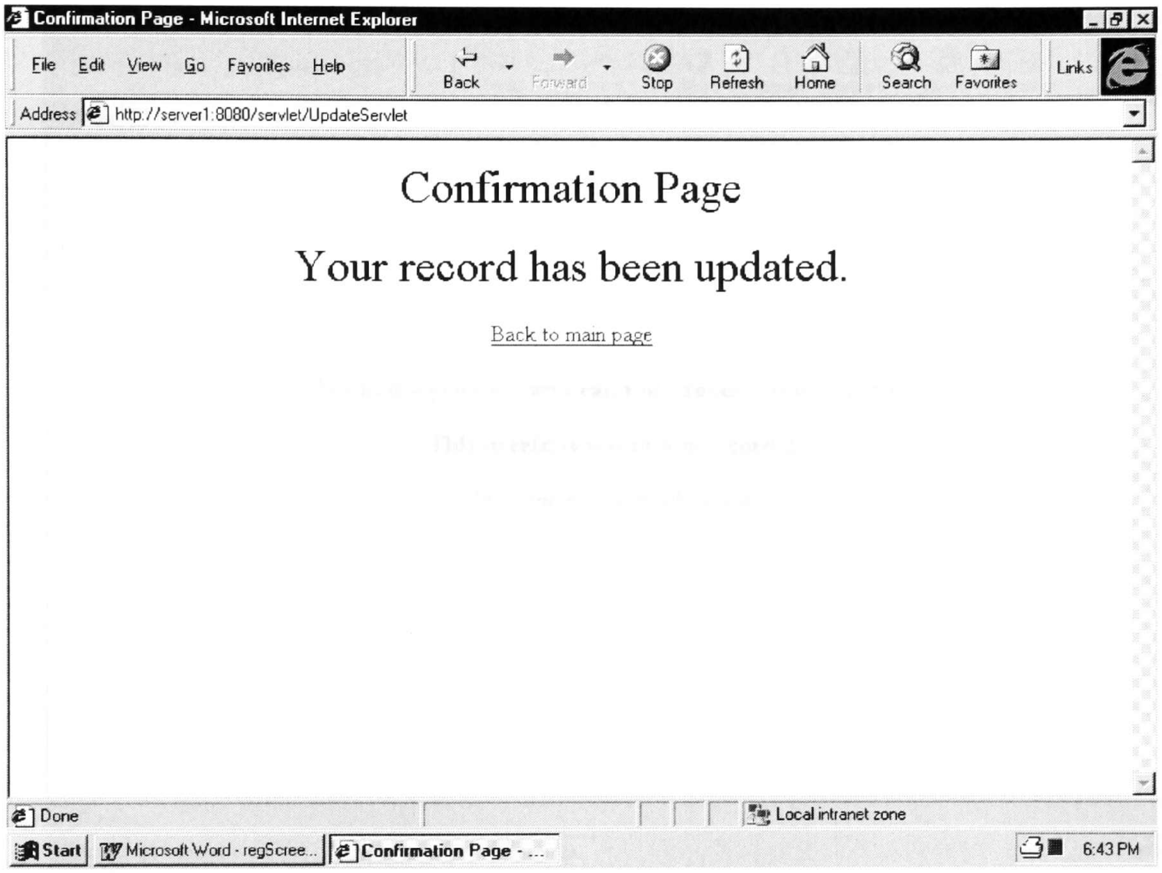
I am very proud of my mathematics education at GSU

To update your record in the future,

please choose

user Id: nconk password: *******

Fig. B3 Update form with a member information.

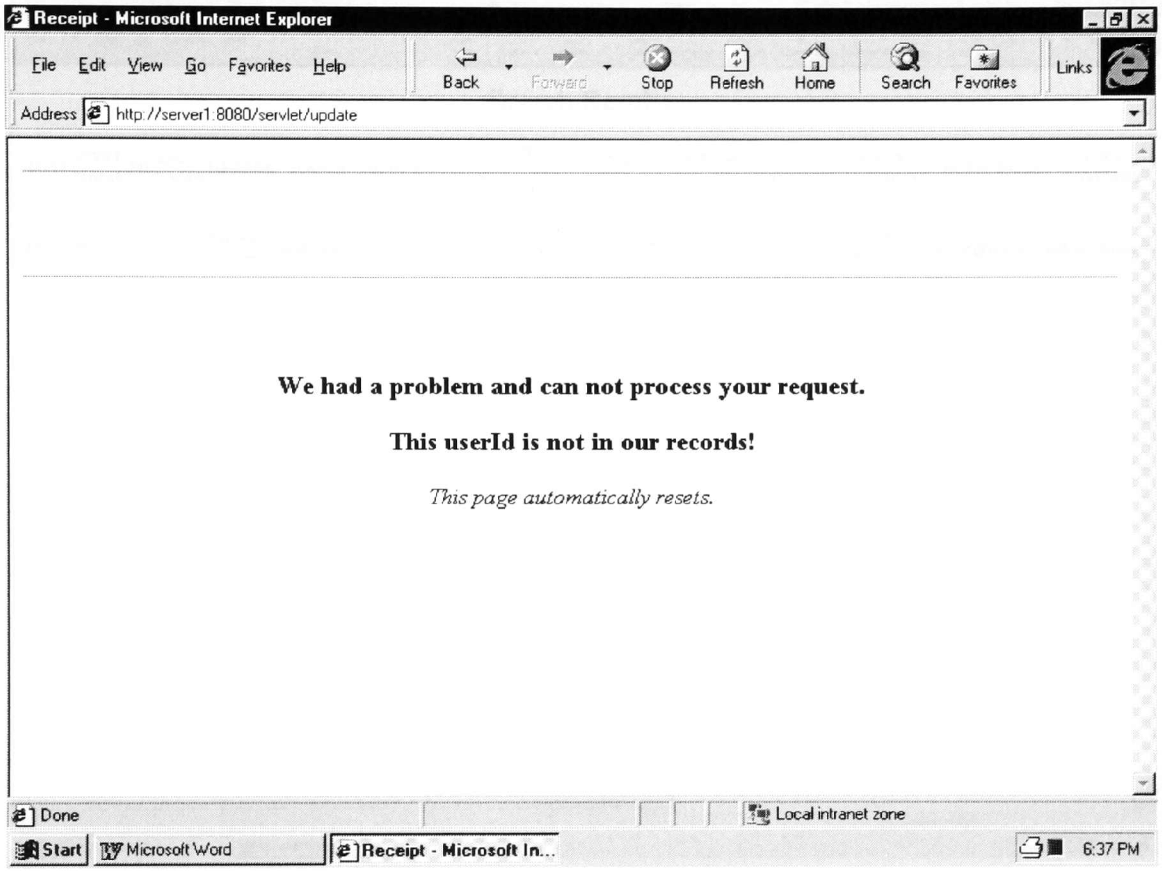

Confirmation Page - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites
Links
Address http://server1:8080/servlet/UpdateServlet
Done
Local intranet zone
Start
Microsoft Word - regScree...
Confirmation Page - ...
6:43 PM

Confirmation Page

Your record has been updated.

Back to main page

Fig. B4 Confirmation message after a successful update.

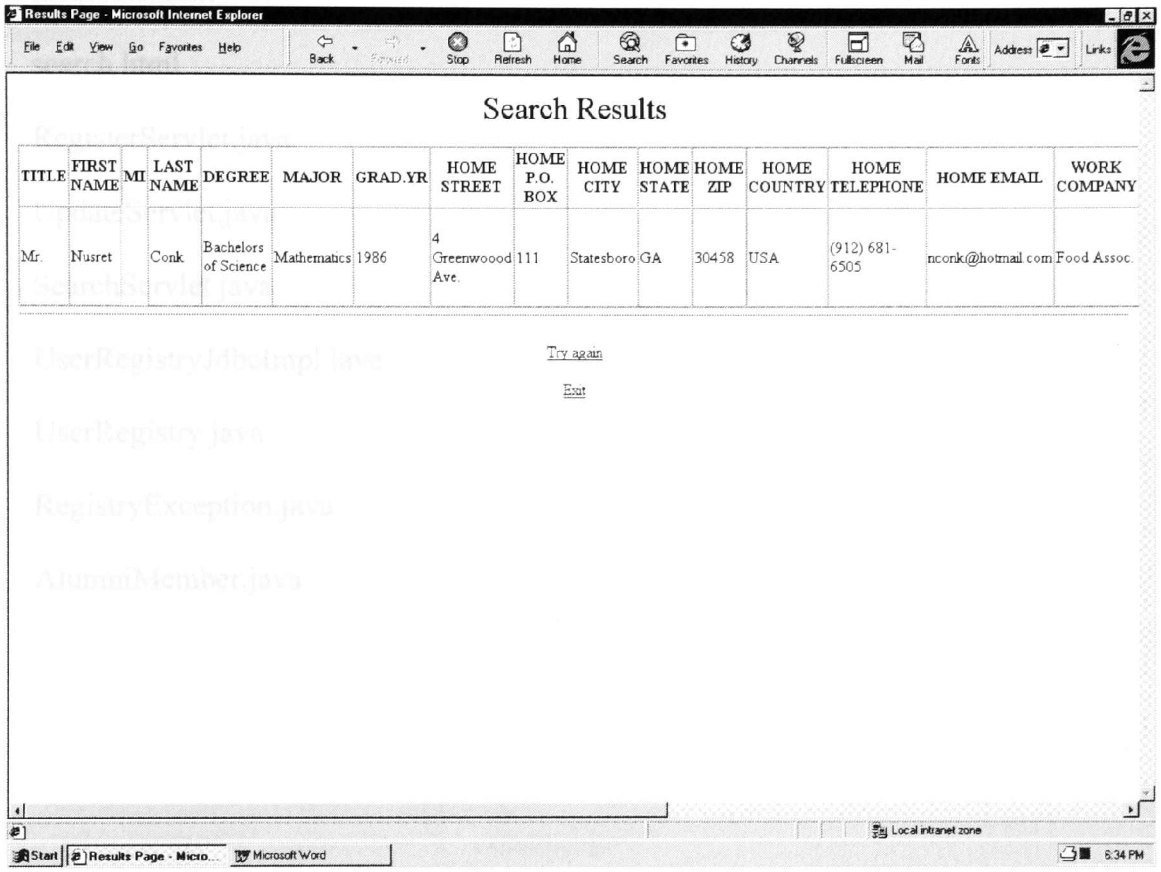

Receipt - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites
Links
Address http://server1:8080/servlet/update
Done
Local intranet zone
Start
Microsoft Word
Receipt - Microsoft In...
6:37 PM

We had a problem and can not process your request.

This userId is not in our records!

This page automatically resets.

Fig. B5 Update function returns refusal for database access.


Results Page - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Fonts Address Links
Local intranet zone
Start Results Page - Micro... Microsoft Word 6:34 PM

Search Results

TITLE	FIRST NAME	MI	LAST NAME	DEGREE	MAJOR	GRAD.YR	HOME STREET	HOME P.O. BOX	HOME CITY	HOME STATE	HOME ZIP	HOME COUNTRY	HOME TELEPHONE	HOME EMAIL	WORK COMPANY
Mr.	Nusret		Conk	Bachelors of Science	Mathematics	1986	4 Greenwoood Ave.	111	Statesboro	GA	30458	USA	(912) 681-6505	nconk@hotmail.com	Food Assoc.

Try again

Exit

Fig. B6 Search results in a HTML table.

APPENDIX C: Application source code

- members.html
- search.html
- RegisterServlet.java
- UpdateServlet.java
- SearchServlet.java
- UserRegistryJdbcImpl.java
- UserRegistry.java
- RegistryException.java
- AlumniMember.java

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>

<head>
<meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1">
<meta name="GENERATOR" content="Microsoft FrontPage 2.0">
<title>Home Page</title>
</head>

<body bgcolor="#CAD0C6">

<p align="center"><font size="6"><strong>Alumni Database </strong></font></p>

<hr size="5">
<div align="center"><center>

<table border="3" width="100%">
  <tr>
    <td width="50%" bgcolor="#C8F2C9"><p align="center"><font
    color="#008000" size="5"><em>We are very glad that you
    are visiting with us.</em></font></p>
    </td>
    <td width="50%" bgcolor="#F3EDBC"><a
    href="http://server1:8080/search.html"><font size="5">Search</font></a><font
    size="5"> </font><font size="4"><strong>-</strong></font><font
    size="5"><strong> </strong></font><font size="4"><strong>to
    see the members.</strong></font><p><a
    href="http://server1:8080/servlet/register"><font
    size="5">Register </font></a><font size="4"><strong>- to
    become a member.</strong></font></p>
    <table border="1" cellpadding="0" cellspacing="1"
    width="100%" bgcolor="#EC9BDC">
      <tr>
        <td><font size="4">Update your record. </font><form
        action="http://server1:8080/servlet/update"
        method="post" name="Update">
          <p><font size="4">User Id:<input type="text"
          size="6" name="UserId">Password:<input
          type="password" size="6" name="Password"><input
          type="submit" name="Submit" value="GO"></font></p>
        </form>
        </td>
      </tr>
    </table>
    </td>
  </tr>
```

```
</table>
</center></div>

<hr size="5">

<p>&nbsp;</p>
</body>
</html>
```

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>

<head>
<meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1">
<meta name="GENERATOR" content="Microsoft FrontPage 2.0">
<title>Search</title>
</head>

<body bgcolor="#DBD1E9">

<p align="center"><font size="6"><strong>Alumni Search Form</strong></font></p>

<form action="http://server1:8080/servlet/search" method="POST"
name="Search">
    <pre><font face="Times New Roman">                              </font></pre>
    <div align="center"><center><table border="3" width="100%">
        <tr>
            <td width="100%" bgcolor="#F7FBC6"><div
            align="center"><center><pre><font size="5"><strong>Search for:</strong></font> < ↙
                input
type="text" size="20" name="searchString"> </pre>
            </center></div><div align="center"><center><pre><font
size="5"><strong>as</strong></font>  <font face="Times New Roman">  </font><input
type="radio" checked name="Field" value="lastName"><font size="4"><strong>Last Name</ ↙
    strong></font>  <input
type="radio" name="Field" value="major1"><font size="4"><strong>Major</strong></font> < ↙
    input
type="radio" name="Field" value="gradDate1"><font size="4"><strong>Grad.year</strong></ ↙
    font>  <input
type="radio" name="Field" value="homeCity"><font size="4"><strong>City</strong></font></↙
    pre>
            </center></div><div align="center"><center><pre><input
type="submit" name="Action" value="Search">  <input type="submit"
name="Action" value="Exit"></pre>
            </center></div></td>
        </tr>
    </table>
    </center></div><pre><font face="Times New Roman">                              </↙
        font></pre>
</form>
</body>
</html>
```

```
import java.net.URL;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.io.*;

/**
 * @author Nusret Conk
 * @version 1.0 April 3, 1999
 */

public class RegisterServlet extends HttpServlet
{
  /**
   * The UserRegistry implementation that provides access to a storage of users.
   */
  private UserRegistry _registry;
  private final String servletName = "RegisterServlet";


  /**
   * @param stub Arguments passed by the Servlet loader
   * @return void
   */
  public void init (ServletConfig config) throws ServletException
  {
    super.init(config);
    _registry = new UserRegistryJdbcImpl();

  }


  /**
   * Main service routine.
   * @param request The HttpServletRequest.
   * @param response The HttpServletResponse.
   * @throws ServletException
   * @throws IOException
   */
  public void service (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
  {
    String ticket;

    HttpSession session = request.getSession(true);
    ticket = (String)session.getValue("ticketNo");
```

```
if (ticket == null)
{

   ticket = session.getId();
   session.putValue("ticketNo",ticket);
   String message = "Welcome to registration. Please complete all fields.";
   sendRegistrationForm(request, response, message);

}
else
{
   if ((request.getParameter("Action").equals("Exit")))
   {
      session.invalidate();
      response.sendRedirect("/members.html");
   }


   if (!(request.getParameter("userId").equals("")) &&
      !(request.getParameter("password").equals("")))
   {
      try
      {

         AlumniMember members[] = _registry.getMember (request.getParameter("userId"));

         if ( members.length > 0 )
         {
            // Duplicate
            //response.sendError (HttpServletResponse.SC_INTERNAL_SERVER_ERROR ↙
               , "The UserId you've chosen already exist, please go back and pick another one ↙
               .");
            String message ="This username is assign to someone else. Please pick another  ↙
               username.";
            sendRegistrationForm(request, response, message);

         }
         else
         {
            AlumniMember member = new AlumniMember ();

            member.setuserId (request.getParameter("userId"));
            member.setpassword (request.getParameter("password"));
            member.settitle (request.getParameter("title"));
            member.setfirstName (request.getParameter("firstName"));
            member.setmiddleInitial (request.getParameter("middleInitial"));
            member.setlastName (request.getParameter("lastName"));
            member.setdegree1 (request.getParameter("degree1"));
            member.setmajor1 (request.getParameter("major1"));
            member.setgradDate1 (request.getParameter("gradDate1"));
            member.setdegree2 (request.getParameter("degree2"));
            member.setmajor2 (request.getParameter("major2"));
            member.setgradDate2 (request.getParameter("gradDate2"));
            member.sethomeStreet (request.getParameter("homeStreet"));
            member.sethomePObox (request.getParameter("homePObox"));
            member.sethomeCity (request.getParameter("homeCity"));
            member.sethomeState (request.getParameter("homeState"));
            member.sethomeZip (request.getParameter("homeZip"));
            member.sethomeCountry (request.getParameter("homeCountry"));
            member.sethomeTel (request.getParameter("homeTel"));
            member.sethomeEmail (request.getParameter("homeEmail"));
            member.setcompany (request.getParameter("company"));
            member.setworkTitle (request.getParameter("workTitle"));
            member.setworkStreet (request.getParameter("workStreet"));
            member.setworkPObox (request.getParameter("workPObox"));
            member.setworkCity (request.getParameter("workCity"));
            member.setworkState (request.getParameter("workState"));
            member.setworkZip (request.getParameter("workZip"));
            member.setworkCountry (request.getParameter("workCountry"));
            member.setworktel (request.getParameter("worktel"));
            member.setworkEmail (request.getParameter("workEmail"));
            member.setcomment (request.getParameter("comment"));

            _registry.setMember (member);
            session.invalidate();
            sendConfirmation(request, response);

        }
    }
    catch (RegistryException e)
    {
        e.printStackTrace();
        session.invalidate();

        // set content type header before accessing the Writer
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // then write the response
        out.println("<html>" +
                "<head><title> Receipt </title>" +
                "<meta http-equiv=\"refresh\" content=\"4; url=" +
                "http://" + request.getHeader("Host") +
                "/members.html;\"></head><body  bgcolor=\"#FFFFFF\">" +
                "<center><hr> <br> &nbsp;<br> &nbsp; <hr> <br> &nbsp;");

         out.println("<h3>Thank you for trying to register."+
                "<p>Unfortunately, something went wrong with our database."+
                "<p>Please try this again later.</h3>" +
                "<p><i>This page automatically resets.</i>" +
                "</body></html>");
         out.flush();
         out.close();
       }
     }
     else //missing fields
     {
       // Show the register page again.
       String message = "Form must be completed before registration.";
       sendRegistrationForm(request, response, message);

     }
   }//ticket check
 }//end of service

 private void sendRegistrationForm(HttpServletRequest req, HttpServletResponse res, String ↙
   message)
   throws ServletException, IOException
 {
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   String serverName = req.getServerName();
   int serverPort = req.getServerPort();

   String userId = req.getParameter("userId");
   String password = req.getParameter("password");
   String title = req.getParameter("title");
   String firstName = req.getParameter("firstName");
   String middleInitial = req.getParameter("middleInitial");
   String lastName = req.getParameter("lastName");
   String degree1 = req.getParameter("degree1");
   String major1 = req.getParameter("major1");
   String gradDate1 = req.getParameter("gradDate1");
   String degree2 = req.getParameter("degree2");
   String major2 = req.getParameter("major2");
   String gradDate2 = req.getParameter("gradDate2");
   String homeStreet = req.getParameter("homeStreet");
   String homePObox = req.getParameter("homePObox");
   String homeCity = req.getParameter("homeCity");
   String homeState = req.getParameter("homeState");
```

```
    String homeZip = req.getParameter("homeZip");
    String homeCountry = req.getParameter("homeCountry");
    String homeTel = req.getParameter("homeTel");
    String homeEmail = req.getParameter("homeEmail");
    String company = req.getParameter("company");
    String workTitle = req.getParameter("workTitle");
    String workStreet = req.getParameter("workStreet");
    String workPObox = req.getParameter("workPObox");
    String workCity = req.getParameter("workCity");
    String workState = req.getParameter("workState");
    String workZip = req.getParameter("workZip");
    String workCountry = req.getParameter("workCountry");
    String worktel = req.getParameter("worktel");
    String workEmail = req.getParameter("workEmail");
    String comment = req.getParameter("comment");

    if (userId == null) userId = "";
    if (password == null) password = "";
    if (title == null) title = "";
    if (firstName == null) firstName = "";
    if (middleInitial == null) middleInitial = "";
    if (lastName == null) lastName = "";
    if (degree1 == null) degree1 = "";
    if (major1 == null) major1 = "";
    if (gradDate1 == null) gradDate1 = "";
    if (degree2 == null) degree2 = "";
    if (major2 == null) major2 = "";
    if (gradDate2 == null) gradDate2 = "";
    if (homeStreet == null) homeStreet = "";
    if (homePObox == null) homePObox = "";
    if (homeCity == null) homeCity = "";
    if (homeState == null) homeState = "";
    if (homeZip == null) homeZip = "";
    if (homeCountry == null) homeCountry = "";
    if (homeTel == null) homeTel = "";
    if (homeEmail == null) homeEmail = "";
    if (company == null) company = "";
    if (workTitle == null) workTitle = "";
    if (workStreet == null) workStreet = "";
    if (workPObox == null) workPObox = "";
    if (workCity == null) workCity = "";
    if (workState == null) workState = "";
    if (workZip == null) workZip = "";
    if (workCountry == null) workCountry = "";
    if (worktel == null) worktel = "";
    if (workEmail == null) workEmail = "";
    if (comment == null) comment = "";
```

```
out.println("<html><head><meta http-equiv=\"Content-Type\" content=\"text/html; charset=↙
  iso-8859-1\">"+
      "<title>Registration</title></head>"+
      "<body bgcolor=\"#CAD0C6\">"+
      "<p align=\"center\"><font size=\"5\"><strong>Alumni Registration Form</strong ↙
        ></font></p>"+
      "<hr size=\"5\"><pre><font color=\"#FF0080\" size=\"4\" face=\"Arial\"><em>< ↙
        blink>" + message +
      "</blink></em></font></pre>"+
      "<form action=http://" + serverName + ":" + serverPort + "/servlet/" + servletName ↙
        + " method=\"POST\" name=\"Registration\">"+
      "<p><font size=\"3\"><strong>Title: <select name=\"title\" size=\"1\">"+
      "<option>Mr.</option><option>Mrs.</option><option>Ms.</option><option>Dr.</↙
        option>"+
      "</select>"+
      " First Name: <input type=\"text\" size=\"17\" name=\"firstName\" value=\"" + ↙
        firstName + "\">"+
      " M.I.: <input type=\"text\" size=\"2\" name=\"middleInitial\" value=\"" + ↙
        middleInitial + "\">"+
      " Last Name: </strong></font><font size=\"3\" face=\"System\">"+
      "<strong><input type=\"text\" size=\"18\" name=\"lastName\" value=\"" + lastName↙
        + "\"></strong></font></p>"+
      "<p><font size=\"3\"><strong>Degree:<select name=\"degree1\" size=\"1\">"+
      "   <option>Bachelors of Science</option>"+
      "   <option>Masters of Science</option>"+
      "</select> Major: <select name=\"major1\" size=\"1\">"+
      "   <option>Mathematics</option>"+
      "   <option>Computer Science</option>"+
      "</select> Graduation Date: <input type=\"text\" size=\"5\" name=\"gradDate1\" ↙
        value=\"" + gradDate1 + "\">"+
      "</strong></font></p>"+
      //"<p><font size=\"3\"><strong>Degree 2:<select name=\"degree2\" size=\"1\">"+
      //"   <option selected>Masters of Science</option>"+
      //"   <option>Bachelors of Science</option>"+
      //"</select> Major: <select name=\"major2\" size=\"1\">"+
      //"   <option selected>Mathematics</option>"+
      //"   <option>Computer Science</option>"+
      //"</select> Graduation Date: <input type=\"text\" size=\"5\" name=\"gradDate2\" ↙
        value=\"" + gradDate2 + "\"></strong></font></p>"+
      "<hr size=\"5\">"+
      "<p><font size=\"3\"><strong>Please tell us how we can contact you.</strong></ ↙
        font></p>"+
      "<table border=\"3\" width=\"100%\" bgcolor=\"#00FFFF\" bordercolor=\"#0080C0↙
        \">"+
        "<tr>"+
          "<th align=\"left\" width=\"50%\" bgcolor=\"#AFE0C0\"><div align=\"center ↙
```

```
            \"><center><address>"+
          "<font size=\"3\"><strong>Your home </strong></font></address></center></ ↙
            div></th>"+
          "<td width=\"50%\" bgcolor=\"#C0C0C0\"><p align=\"center\"><font size=\"3 ↙
            \"><strong>Your work</strong></font></p>"+
          "</td>"+
        "</tr>"+
        "<tr>"+
          "<td width=\"50%\" bgcolor=\"#AFE0C0\"><pre><font size=\"3\">"+
          "<strong>Street:  <input type=\"text\" size=\"25\" name=\"homeStreet\" value  ↙
            =\"" + homeStreet + "\"></strong></font></pre>"+
          "<pre><font size=\"3\"><strong>P.O.Box: <input type=\"text\" size=\"6\" name↙
            =\"homePObox\" value=\"" + homePObox + "\"></strong></font></pre>"+
          "<pre><font size=\"3\"><strong>City:   <input type=\"text\" size=\"11\" name ↙
            =\"homeCity\" value=\"" + homeCity + "\">"+
          " State:<input type=\"text\" size=\"4\" name=\"homeState\" value=\"" +       ↙
            homeState + "\">"+
          " Zip:<input type=\"text\" size=\"6\" name=\"homeZip\" value=\"" + homeZip  ↙
            + "\"></strong></font></pre>"+
          "<pre><font size=\"3\"><strong>Country: <input type=\"text\" size=\"10\"     ↙
            name=\"homeCountry\" value=\"" + homeCountry + "\">"+
          " Tel: <input type=\"text\" size=\"12\" name=\"homeTel\" value=\"" + homeTel↙
            + "\"></strong></font></pre>"+
          "<pre><font size=\"3\"><strong>Email:  <input type=\"text\" size=\"25\" name↙
            =\"homeEmail\" value=\"" + homeEmail + "\"></strong></font></pre>"+
          "</td><td width=\"50%\" bgcolor=\"#C0C0C0\"><pre><font size=\"3\">"+
          "<strong>Company: <input type=\"text\" size=\"22\" name=\"company\" value ↙
            =\"" + company + "\"></strong></font></pre>"+
          "<pre><font size=\"3\"><strong>Title:  <input type=\"text\" size=\"21\" name ↙
            =\"workTitle\" value=\"" + workTitle + "\"></strong></font></pre>"+
          "<pre><font size=\"3\"><strong>Street  <input type=\"text\" size=\"25\" name ↙
            =\"workStreet\" value=\"" + workStreet + "\"></strong></font></pre>"+
          "<pre><font size=\"3\"><strong>P.O.Box: <input type=\"text\" size=\"6\" name↙
            =\"workPObox\" value=\"" + workPObox + "\"></strong></font></pre>"+
          "<pre><font size=\"3\"><strong>City:   <input type=\"text\" size=\"11\" name ↙
            =\"workCity\" value=\"" + workCity + "\">"+
          "State:<input type=\"text\" size=\"4\" name=\"workState\" value=\"" +         ↙
            workState + "\"> Zip:<input type=\"text\" size=\"6\" name=\"workZip\" value↙
            =\"" + workZip + "\"></strong></font></pre>"+
          "<pre><font size=\"3\"><strong>Country: <input type=\"text\" size=\"10\"     ↙
            name=\"workCountry\" value=\"" + workCountry + "\">"+
          " Tel: <input type=\"text\" size=\"12\" name=\"worktel\" value=\"" + worktel  ↙
            + "\"></strong></font></pre>"+
          "<pre><font size=\"3\"><strong>Email:  <input type=\"text\" size=\"25\" name↙
            =\"workEmail\" value=\"" + workEmail + "\"></strong></font></pre>"+
          "</td>"+
        "</tr></table>");
```

```
    out.println("<table border=\"0\" width=\"100%\">"+
            "<tr>"+
              "<td width=\"50%\" bgcolor=\"#ECEFC7\"><font size=\"3\"><strong>Please"+
              " tell us about yourself or your comments:</strong></font></td>"+
              "<td width=\"50%\"><font size=\"3\"><strong></strong></font>&nbsp;</td>"+
            "</tr>"+
            "<tr>"+
              "<td width=\"50%\" bgcolor=\"#ECEFC7\"><font size=\"3\"><strong><textarea  ↙
                "+
              "name=\"comment\" rows=\"3\" cols=\"48\">"+comment+"</textarea></strong><↙
                font></td>"+
              "<td width=\"50%\" bgcolor=\"#D39D96\"><p align=\"center\"><font size=\"3   ↙
                \">"+
              "<strong>To update your record in the future,</strong></font></p>"+
              "<p align=\"center\"><font size=\"3\"><strong>please choose </strong></font></↙
                p>"+
              "<div align=\"center\"><center><pre><font size=\"3\" face=\"Times New Roman↙
                \">"+
              "<strong>user Id:</strong></font><font size=\"3\">"+
              "<strong><input type=\"text\" size=\"6\" name=\"userId\" value=\"" + userId    ↙
                + "\">"+
              "</strong></font><font size=\"3\" face=\"Times New Roman\"><strong>           ↙
                password: </strong></font>"+
              "<font size=\"3\"><strong>"+
              "<input type=\"password\" size=\"6\" name=\"password\" value=\"" + password   ↙
                + "\"></strong></font></pre>"+
              "</center></div></td>"+
            "</tr>"+
          "</table>"+
          "<p align=\"center\"><font size=\"3\"><strong>"+
          "<input type=\"submit\" name=\"Action\" value=\"Register\">"+
          "<input type=\"reset\" name=\"Reset\" value=\"Clear Form\">"+
          "<input type=\"submit\" name=\"Action\" value=\"Exit\"></p></strong></font>"+
          "</form>"+
          "</body>"+
          "</html>");
    out.flush();
    out.close();
  }//end of sendRegistrationForm

  private void sendConfirmation(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    String serverName = req.getServerName();
    int serverPort = req.getServerPort();
    String firstName = req.getParameter("firstName");

    out.println("<html>"+
            "<head>"+
            "<title>Confirmation Page</title></head>"+
            "<body bgcolor=\"#FFFFFF\">"+
            "<p align=\"center\"><font size=\"6\">Confirmation Page</font></p>"+
            "<p align=\"center\"><font size=\"6\">" + firstName + ", you are now a member. </ ↙
              font></p>"+
            "<p align=\"center\"><font size=\"6\">Congratulations!</font></p>"+
            "<p align=\"center\"><a href=http://" + serverName + ":" + serverPort + "/members.↙
              html>Back to main page</a></p>"+
            "</body></html>");
    out.flush();
    out.close();
  }//end of confirmation

}//end of servlet
```

```
import java.net.URL;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.io.*;

/**
 * @author Nusret Conk
 * @version 1.0 April 3, 1999
 */

public class UpdateServlet extends HttpServlet
{
   /**
    * The UserRegistry implementation that provides access to a storage of users.
    */
   private UserRegistry _registry;
   private final String servletName = "UpdateServlet";

   /**
    * @param stub Arguments passed by the Servlet loader
    * @return void
    */
   public void init (ServletConfig config) throws ServletException
   {
      super.init(config);
      _registry = new UserRegistryJdbcImpl();

   }


   /**
    * Main service routine.
    * @param request The HttpServletRequest.
    * @param response The HttpServletResponse.
    * @throws ServletException
    * @throws IOException
    */
   public void service (HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException
   {
      String ticket;

      HttpSession session = request.getSession(true);
      ticket = (String)session.getValue("ticketNo");
    if (ticket == null)
    {
      String userId = request.getParameter("UserId");
      String userPassword = request.getParameter("Password");

      if (userId != null && userPassword != null)
      {
        ticket = session.getId();
        session.putValue("ticketNo",ticket);
        try
        {

          AlumniMember members[] = _registry.getMember (userId);

          if ( members.length == 1 )
          {
            AlumniMember member = members[0];

            if (userPassword.equals(member.getpassword()))
            {
              sendUpdateForm(request, response, member);

            }
            else //password did not match
            {
              session.invalidate();
              String message = "This password is not correct.";
              sendProblemMessage(request, response, message);
            }


          }
          else if ( members.length == 0 )
          {
            session.invalidate();
            String message = "This userId is not in our records!";
            sendProblemMessage(request, response, message);
          }
          else
          {
            session.invalidate();
            String message = "Please register again with a different userId.";
            sendProblemMessage(request, response, message);
          }
        }
        catch (RegistryException e)
        {
                e.printStackTrace();
                session.invalidate();
                String message = "There is a problem with our database.";
                sendProblemMessage(request, response, message);

            }
        }
        else //bad userId and/or userPassword
        {
            session.invalidate();
            String message = "We need your userId and password. Try again!";
            sendProblemMessage(request, response, message);
        }
    }
    else // ticket is not null
    {
        if (request.getParameter("Action").equals("Update"))
        {
            try
            {
                AlumniMember member = new AlumniMember ();

                member.setuserId (request.getParameter("userId"));
                member.setpassword (request.getParameter("password"));
                member.settitle (request.getParameter("title"));
                member.setfirstName (request.getParameter("firstName"));
                member.setmiddleInitial (request.getParameter("middleInitial"));
                member.setlastName (request.getParameter("lastName"));
                member.setdegree1 (request.getParameter("degree1"));
                member.setmajor1 (request.getParameter("major1"));
                member.setgradDate1 (request.getParameter("gradDate1"));
                member.setdegree2 (request.getParameter("degree2"));
                member.setmajor2 (request.getParameter("major2"));
                member.setgradDate2 (request.getParameter("gradDate2"));
                member.sethomeStreet (request.getParameter("homeStreet"));
                member.sethomePObox (request.getParameter("homePObox"));
                member.sethomeCity (request.getParameter("homeCity"));
                member.sethomeState (request.getParameter("homeState"));
                member.sethomeZip (request.getParameter("homeZip"));
                member.sethomeCountry (request.getParameter("homeCountry"));
                member.sethomeTel (request.getParameter("homeTel"));
                member.sethomeEmail (request.getParameter("homeEmail"));
                member.setcompany (request.getParameter("company"));
                member.setworkTitle (request.getParameter("workTitle"));
                member.setworkStreet (request.getParameter("workStreet"));
                member.setworkPObox (request.getParameter("workPObox"));
                member.setworkCity (request.getParameter("workCity"));
            member.setworkState (request.getParameter("workState"));
            member.setworkZip (request.getParameter("workZip"));
            member.setworkCountry (request.getParameter("workCountry"));
            member.setworktel (request.getParameter("worktel"));
            member.setworkEmail (request.getParameter("workEmail"));
            member.setcomment (request.getParameter("comment"));

            _registry.setMember (member);
            session.invalidate();
            String message = "Your record has been updated.";
            sendConfirmation(request, response, message);
        }
        catch (RegistryException e)
        {
            e.printStackTrace();
            session.invalidate();
            String message = "There is a problem with update operation.";
            sendProblemMessage(request, response, message);

        }
    }
    else if (request.getParameter("Action").equals("Delete"))
    {
        try
        {
            String userId = request.getParameter("userId");
            AlumniMember member = new AlumniMember ();
            member.setuserId (userId);
            _registry.deleteMember(member);
            session.invalidate();
            String message = "Your record has been deleted.";
            sendConfirmation(request, response, message);
        }
        catch (RegistryException e)
        {
            e.printStackTrace();
            session.invalidate();
            String message = "There is a problem with delete operation.";
            sendProblemMessage(request, response, message);

        }
    }
    else
    {
        session.invalidate();
        String message = "No changes are made to your record.";
        sendConfirmation(request, response, message);
        }
    }//ticket check
}//end of service

private void sendUpdateForm(HttpServletRequest req, HttpServletResponse res,                ↙
    AlumniMember member)
    throws ServletException, IOException
{
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    String serverName = req.getServerName();
    int serverPort = req.getServerPort();



    String userId = member.getuserId();
    String password = member.getpassword();

    String title = member.gettitle();
    String firstName = member.getfirstName();
    String middleInitial = member.getmiddleInitial();
    String lastName = member.getlastName();
    String degree1 = member.getdegree1();
    String major1 = member.getmajor1();
    String gradDate1 = member.getgradDate1();
    String degree2 = member.getdegree2();
    String major2 = member.getmajor2();
    String gradDate2 = member.getgradDate2();
    String homeStreet = member.gethomeStreet();
    String homePObox = member.gethomePObox();
    String homeCity = member.gethomeCity();
    String homeState = member.gethomeState();
    String homeZip = member.gethomeZip();
    String homeCountry = member.gethomeCountry();
    String homeTel = member.gethomeTel();
    String homeEmail = member.gethomeEmail();
    String company = member.getcompany();
    String workTitle = member.getworkTitle();
    String workStreet = member.getworkStreet();
    String workPObox = member.getworkPObox();
    String workCity = member.getworkCity();
    String workState = member.getworkState();
    String workZip = member.getworkZip();
    String workCountry = member.getworkCountry();
    String worktel = member.getworktel();
    String workEmail = member.getworkEmail();
    String comment = member.getcomment();

    if (userId == null) userId = "";
    if (password == null) password = "";
    if (title == null) title = "";
    if (firstName == null) firstName = "";
    if (middleInitial == null) middleInitial = "";
    if (lastName == null) lastName = "";
    if (degree1 == null) degree1 = "";
    if (major1 == null) major1 = "";
    if (gradDate1 == null) gradDate1 = "";
    if (degree2 == null) degree2 = "";
    if (major2 == null) major2 = "";
    if (gradDate2 == null) gradDate2 = "";
    if (homeStreet == null) homeStreet = "";
    if (homePObox == null) homePObox = "";
    if (homeCity == null) homeCity = "";
    if (homeState == null) homeState = "";
    if (homeZip == null) homeZip = "";
    if (homeCountry == null) homeCountry = "";
    if (homeTel == null) homeTel = "";
    if (homeEmail == null) homeEmail = "";
    if (company == null) company = "";
    if (workTitle == null) workTitle = "";
    if (workStreet == null) workStreet = "";
    if (workPObox == null) workPObox = "";
    if (workCity == null) workCity = "";
    if (workState == null) workState = "";
    if (workZip == null) workZip = "";
    if (workCountry == null) workCountry = "";
    if (worktel == null) worktel = "";
    if (workEmail == null) workEmail = "";
    if (comment == null) comment = "";

    out.println("<html><head><title>Record Update</title></head>"+
            "<body bgcolor=\"#CAD0C6\">"+
            "<p align=\"center\"><font size=\"5\"><strong>Member Record Update</strong></ ↙
              font></p>"+
            "<hr size=\"5\">"+
            "<pre><font color=\"#FF0080\" size=\"4\" face=\"Arial\"><em><blink>" +
            "Please make changes to your record.</blink></em></font></pre>"+
            "<form action=http://" + serverName + ":" + serverPort + "/servlet/" + servletName ↙
              + " method=\"POST\" name=\"Registration\">"+
            "<p><font size=\"3\"><strong>Title: <select name=\"title\" size=\"1\">"+
            "<option selected>"+ title +"</option><option>Mr.</option><option>Mrs.</option ↙
              ><option>Ms.</option><option>Dr.</option>"+
            "</select>"+
" First Name: <input type=\"text\" size=\"17\" name=\"firstName\" value=\"" +   ↙
  firstName + "\">"+
" M.I.: <input type=\"text\" size=\"2\" name=\"middleInitial\" value=\"" +      ↙
  middleInitial + "\">"+
" Last Name: </strong></font><font size=\"3\" face=\"System\">"+
"<strong><input type=\"text\" size=\"18\" name=\"lastName\" value=\"" + lastName↙
  + "\"></strong></font></p>"+
"<p><font size=\"3\"><strong>Degree:<select name=\"degree1\" size=\"1\">"+
"   <option selected>" + degree1 +
"   <option>Bachelors of Science</option>"+
"   <option>Masters of Science</option>"+
"</select> Major: <select name=\"major1\" size=\"1\">"+
"   <option selected>" + major1 +
"   <option>Mathematics</option>"+
"   <option>Computer Science</option>"+
"</select> Graduation Date: <input type=\"text\" size=\"5\" name=\"gradDate1\"  ↙
  value=\"" + gradDate1 + "\">"+
"</strong></font></p>"+
//"<p><font size=\"3\"><strong>Degree 2:<select name=\"degree2\" size=\"1\">"+
//"   <option selected>Masters of Science</option>"+
//"   <option>Bachelors of Science</option>"+
//"</select> Major: <select name=\"major2\" size=\"1\">"+
//"   <option selected>Mathematics</option>"+
//"   <option>Computer Science</option>"+
//"</select> Graduation Date: <input type=\"text\" size=\"5\" name=\"gradDate2\"  ↙
  value=\"" + gradDate2 + "\"></strong></font></p>"+
"<hr size=\"5\">"+
"<p><font size=\"3\"><strong>Please tell us how we can contact you.</strong></  ↙
  font></p>"+
"<table border=\"3\" width=\"100%\" bgcolor=\"#00FFFF\" bordercolor=\"#0080C0↙
  \">"+
  "<tr>"+
    "<th align=\"left\" width=\"50%\" bgcolor=\"#AFE0C0\"><div align=\"center  ↙
      \"><center><address>"+
    "<font size=\"3\"><strong>Your home </strong></font></address></center></↙
      div></th>"+
    "<td width=\"50%\" bgcolor=\"#C0C0C0\"><p align=\"center\"><font size=\"3↙
      \"><strong>Your work</strong></font></p>"+
    "</td>"+
  "</tr>"+
  "<tr>"+
    "<td width=\"50%\" bgcolor=\"#AFE0C0\"><pre><font size=\"3\">"+
    "<strong>Street:  <input type=\"text\" size=\"25\" name=\"homeStreet\" value ↙
      =\"" + homeStreet + "\"></strong></font></pre>"+
    "<pre><font size=\"3\"><strong>P.O.Box: <input type=\"text\" size=\"6\" name↙
      =\"homePObox\" value=\"" + homePObox + "\"></strong></font></pre>"+
    "<pre><font size=\"3\"><strong>City:   <input type=\"text\" size=\"11\" name ↙
```

```
                =\"homeCity\" value=\"" + homeCity + "\">"+
              " State:<input type=\"text\" size=\"4\" name=\"homeState\" value=\"" +      ↙
                homeState + "\">"+
              " Zip:<input type=\"text\" size=\"6\" name=\"homeZip\" value=\"" + homeZip ↙
                + "\"></strong></font></pre>"+
              "<pre><font size=\"3\"><strong>Country: <input type=\"text\" size=\"10\"    ↙
                name=\"homeCountry\" value=\"" + homeCountry + "\">"+
              " Tel: <input type=\"text\" size=\"12\" name=\"homeTel\" value=\"" + homeTel↙
                + "\"></strong></font></pre>"+
              "<pre><font size=\"3\"><strong>Email:  <input type=\"text\" size=\"25\" name↙
                =\"homeEmail\" value=\"" + homeEmail + "\"></strong></font></pre>"+
              "</td><td width=\"50%\" bgcolor=\"#C0C0C0\"><pre><font size=\"3\">"+
              "<strong>Company: <input type=\"text\" size=\"22\" name=\"company\" value ↙
                =\"" + company + "\"></strong></font></pre>"+
              "<pre><font size=\"3\"><strong>Title:  <input type=\"text\" size=\"21\" name ↙
                =\"workTitle\" value=\"" + workTitle + "\"></strong></font></pre>"+
              "<pre><font size=\"3\"><strong>Street   <input type=\"text\" size=\"25\" name ↙
                =\"workStreet\" value=\"" + workStreet + "\"></strong></font></pre>"+
              "<pre><font size=\"3\"><strong>P.O.Box: <input type=\"text\" size=\"6\" name↙
                =\"workPObox\" value=\"" + workPObox + "\"></strong></font></pre>"+
              "<pre><font size=\"3\"><strong>City:   <input type=\"text\" size=\"11\" name ↙
                =\"workCity\" value=\"" + workCity + "\">"+
              "State:<input type=\"text\" size=\"4\" name=\"workState\" value=\"" +       ↙
                workState + "\"> Zip:<input type=\"text\" size=\"6\" name=\"workZip\" value↙
                =\"" + workZip + "\"></strong></font></pre>"+
              "<pre><font size=\"3\"><strong>Country: <input type=\"text\" size=\"10\"    ↙
                name=\"workCountry\" value=\"" + workCountry + "\">"+
              " Tel: <input type=\"text\" size=\"12\" name=\"worktel\" value=\"" + worktel ↙
                + "\"></strong></font></pre>"+
              "<pre><font size=\"3\"><strong>Email:  <input type=\"text\" size=\"25\" name↙
                =\"workEmail\" value=\"" + workEmail + "\"></strong></font></pre>"+
              "</td>"+
            "</tr></table>");
    out.println("<table border=\"0\" width=\"100%\">"+
          "<tr>"+
            "<td width=\"50%\" bgcolor=\"#ECEFC7\"><font size=\"3\"><strong>Please"+
            " tell us about yourself or your comments:</strong></font></td>"+
            "<td width=\"50%\"><font size=\"3\"></font>&nbsp;</td>"+
          "</tr>"+
          "<tr>"+
            "<td width=\"50%\" bgcolor=\"#ECEFC7\"><font size=\"3\"><strong><textarea ↙
              "+
            "name=\"comment\" rows=\"3\" cols=\"48\">"+ comment +"</textarea></strong ↙
              ></font></td>"+
            "<td width=\"50%\" bgcolor=\"#D39D96\"><p align=\"center\"><font size=\"3   ↙
              \">"+
            "<strong>To update your record in the future,</strong></font></p>"+
import java.net.URL;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.io.*;

/**
 * @author Nusret Conk
 * @version 1.0 April 3, 1999
 */

public class SearchServlet extends HttpServlet
{
  /**
   * The UserRegistry implementation that provides access to a storage of users.
   */
  private UserRegistry _registry;
  private final String servletName = "SearchServlet";

  /**
   * @param stub Arguments passed by the Servlet loader
   * @return void
   */
  public void init (ServletConfig config) throws ServletException
  {
    super.init(config);
    _registry = new UserRegistryJdbcImpl();

  }


  /**
   * Main service routine - do a search.
   * @param request The HttpServletRequest.
   * @param response The HttpServletResponse.
   * @throws ServletException
   * @throws IOException
   */
  public void service (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
  {


    if ((request.getParameter("Action").equals("Search")))
    {
      String columnName = request.getParameter("Field");
String searchString = request.getParameter("searchString");

if (columnName != null && searchString != null)
{

   try
   {
     AlumniMember members[] = _registry.searchRegistry (columnName, searchString);

     if ( members.length == 0 )
     {
        //session.invalidate();
        String message = "Can not find matching data !";
        sendProblemMessage(request, response, message);
     }
     else
     {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String serverName = request.getServerName();
        int serverPort = request.getServerPort();
        out.println("<html>"+
                "<head>"+
                "<title>Results Page</title></head>"+
                "<body bgcolor=\"#FFFFFF\">"+
                "<p align=\"center\"><font size=\"6\">Search Results</font></p>"+
                "<TABLE><table border=\"1\" cellspacing=\"0\" width=\"100%\">");

        out.println("<th>TITLE"+
                "<th>FIRST NAME"+
                "<th>MI"+
                "<th>LAST NAME"+
                "<th>DEGREE    "+
                "<th>MAJOR     "+
                "<th>GRAD.YR"+
                //"<th>DEGREE2"+
                //"<th>MAJOR2"+
                //"<th>GRAD.YR"+
                "<th>HOME STREET"+
                "<th>HOME P.O. BOX"+
                "<th>HOME CITY"+
                "<th>HOME STATE"+
                "<th>HOME ZIP"+
                "<th>HOME COUNTRY"+
                "<th>HOME TELEPHONE"+
                "<th>HOME EMAIL      "+
                "<th>WORK COMPANY   "+
                member.setlastName (request.getParameter("lastName"));
                member.setdegree1 (request.getParameter("degree1"));
                member.setmajor1 (request.getParameter("major1"));
                member.setgradDate1 (request.getParameter("gradDate1"));
                member.setdegree2 (request.getParameter("degree2"));
                member.setmajor2 (request.getParameter("major2"));
                member.setgradDate2 (request.getParameter("gradDate2"));
                member.sethomeStreet (request.getParameter("homeStreet"));
                member.sethomePObox (request.getParameter("homePObox"));
                member.sethomeCity (request.getParameter("homeCity"));
                member.sethomeState (request.getParameter("homeState"));
                member.sethomeZip (request.getParameter("homeZip"));
                member.sethomeCountry (request.getParameter("homeCountry"));
                member.sethomeTel (request.getParameter("homeTel"));
                member.sethomeEmail (request.getParameter("homeEmail"));
                member.setcompany (request.getParameter("company"));
                member.setworkTitle (request.getParameter("workTitle"));
                member.setworkStreet (request.getParameter("workStreet"));
                member.setworkPObox (request.getParameter("workPObox"));
                member.setworkCity (request.getParameter("workCity"));
                member.setworkState (request.getParameter("workState"));
                member.setworkZip (request.getParameter("workZip"));
                member.setworkCountry (request.getParameter("workCountry"));
                member.setworktel (request.getParameter("worktel"));
                member.setworkEmail (request.getParameter("workEmail"));
                member.setcomment (request.getParameter("comment"));

                _registry.setMember (member);
                session.invalidate();
                sendConfirmation(request, response);

            }
        }
        catch (RegistryException e)
        {
            e.printStackTrace();
            session.invalidate();

            // set content type header before accessing the Writer
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();

            // then write the response
            out.println("<html>" +
                    "<head><title> Receipt </title>" +
                    "<meta http-equiv=\"refresh\" content=\"4; url=" +
                    "http://" + request.getHeader("Host") +
                "/members.html;\"></head><body bgcolor=\"#FFFFFF\">" +
                "<center><hr> <br> &nbsp;<br> &nbsp; <hr> <br> &nbsp;");

            out.println("<h3>Thank you for trying to register."+
                "<p>Unfortunately, something went wrong with our database."+
                "<p>Please try this again later.</h3>" +
                "<p><i>This page automatically resets.</i>" +
                "</body></html>");
            out.flush();
            out.close();
          }
        }
        else //missing fields
        {
          // Show the register page again.
          String message = "Form must be completed before registration.";
          sendRegistrationForm(request, response, message);

        }
      }//ticket check
    }//end of service

    private void sendRegistrationForm(HttpServletRequest req, HttpServletResponse res, String ↙
      message)
      throws ServletException, IOException
    {
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      String serverName = req.getServerName();
      int serverPort = req.getServerPort();

      String userId = req.getParameter("userId");
      String password = req.getParameter("password");
      String title = req.getParameter("title");
      String firstName = req.getParameter("firstName");
      String middleInitial = req.getParameter("middleInitial");
      String lastName = req.getParameter("lastName");
      String degree1 = req.getParameter("degree1");
      String major1 = req.getParameter("major1");
      String gradDate1 = req.getParameter("gradDate1");
      String degree2 = req.getParameter("degree2");
      String major2 = req.getParameter("major2");
      String gradDate2 = req.getParameter("gradDate2");
      String homeStreet = req.getParameter("homeStreet");
      String homePObox = req.getParameter("homePObox");
      String homeCity = req.getParameter("homeCity");
      String homeState = req.getParameter("homeState");
```

```
String homeZip = req.getParameter("homeZip");
String homeCountry = req.getParameter("homeCountry");
String homeTel = req.getParameter("homeTel");
String homeEmail = req.getParameter("homeEmail");
String company = req.getParameter("company");
String workTitle = req.getParameter("workTitle");
String workStreet = req.getParameter("workStreet");
String workPObox = req.getParameter("workPObox");
String workCity = req.getParameter("workCity");
String workState = req.getParameter("workState");
String workZip = req.getParameter("workZip");
String workCountry = req.getParameter("workCountry");
String worktel = req.getParameter("worktel");
String workEmail = req.getParameter("workEmail");
String comment = req.getParameter("comment");

if (userId == null) userId = "";
if (password == null) password = "";
if (title == null) title = "";
if (firstName == null) firstName = "";
if (middleInitial == null) middleInitial = "";
if (lastName == null) lastName = "";
if (degree1 == null) degree1 = "";
if (major1 == null) major1 = "";
if (gradDate1 == null) gradDate1 = "";
if (degree2 == null) degree2 = "";
if (major2 == null) major2 = "";
if (gradDate2 == null) gradDate2 = "";
if (homeStreet == null) homeStreet = "";
if (homePObox == null) homePObox = "";
if (homeCity == null) homeCity = "";
if (homeState == null) homeState = "";
if (homeZip == null) homeZip = "";
if (homeCountry == null) homeCountry = "";
if (homeTel == null) homeTel = "";
if (homeEmail == null) homeEmail = "";
if (company == null) company = "";
if (workTitle == null) workTitle = "";
if (workStreet == null) workStreet = "";
if (workPObox == null) workPObox = "";
if (workCity == null) workCity = "";
if (workState == null) workState = "";
if (workZip == null) workZip = "";
if (workCountry == null) workCountry = "";
if (worktel == null) worktel = "";
if (workEmail == null) workEmail = "";
if (comment == null) comment = "";
```

```
out.println("<html><head><meta http-equiv=\"Content-Type\" content=\"text/html; charset=↙
  iso-8859-1\">"+
        "<title>Registration</title></head>"+
        "<body bgcolor=\"#CAD0C6\">"+
        "<p align=\"center\"><font size=\"5\"><strong>Alumni Registration Form</strong ↙
          ></font></p>"+
        "<hr size=\"5\"><pre><font color=\"#FF0080\" size=\"4\" face=\"Arial\"><em><  ↙
          blink>" + message +
        "</blink></em></font></pre>"+
        "<form action=http://" + serverName + ":" + serverPort + "/servlet/" + servletName ↙
          + " method=\"POST\" name=\"Registration\">"+
        "<p><font size=\"3\"><strong>Title: <select name=\"title\" size=\"1\">"+
        "<option>Mr.</option><option>Mrs.</option><option>Ms.</option><option>Dr.</↙
          option>"+
        "</select>"+
        " First Name: <input type=\"text\" size=\"17\" name=\"firstName\" value=\"" +    ↙
          firstName + "\">"+
        " M.I.: <input type=\"text\" size=\"2\" name=\"middleInitial\" value=\"" +       ↙
          middleInitial + "\">"+
        " Last Name: </strong></font><font size=\"3\" face=\"System\">"+
        "<strong><input type=\"text\" size=\"18\" name=\"lastName\" value=\"" + lastName↙
          + "\"></strong></font></p>"+
        "<p><font size=\"3\"><strong>Degree:<select name=\"degree1\" size=\"1\">"+
        "   <option>Bachelors of Science</option>"+
        "   <option>Masters of Science</option>"+
        "</select> Major: <select name=\"major1\" size=\"1\">"+
        "   <option>Mathematics</option>"+
        "   <option>Computer Science</option>"+
        "</select> Graduation Date: <input type=\"text\" size=\"5\" name=\"gradDate1\"   ↙
          value=\"" + gradDate1 + "\">"+
        "</strong></font></p>"+
        //"<p><font size=\"3\"><strong>Degree 2:<select name=\"degree2\" size=\"1\">"+
        //"   <option selected>Masters of Science</option>"+
        //"   <option>Bachelors of Science</option>"+
        //"</select> Major: <select name=\"major2\" size=\"1\">"+
        //"   <option selected>Mathematics</option>"+
        //"   <option>Computer Science</option>"+
        //"</select> Graduation Date: <input type=\"text\" size=\"5\" name=\"gradDate2\"  ↙
          value=\"" + gradDate2 + "\"></strong></font></p>"+
        "<hr size=\"5\">"+
        "<p><font size=\"3\"><strong>Please tell us how we can contact you.</strong></  ↙
          font></p>"+
        "<table border=\"3\" width=\"100%\" bgcolor=\"#00FFFF\" bordercolor=\"#0080C0↙
          \">"+
          "<tr>"+
            "<th align=\"left\" width=\"50%\" bgcolor=\"#AFE0C0\"><div align=\"center  ↙
          \"><center><address>"+
        "<font size=\"3\"><strong>Your home </strong></font></address></center></↙
          div></th>"+
        "<td width=\"50%\" bgcolor=\"#C0C0C0\"><p align=\"center\"><font size=\"3↙
          \"><strong>Your work</strong></font></p>"+
        "</td>"+
      "</tr>"+
      "<tr>"+
        "<td width=\"50%\" bgcolor=\"#AFE0C0\"><pre><font size=\"3\">"+
        "<strong>Street:  <input type=\"text\" size=\"25\" name=\"homeStreet\" value ↙
          =\"" + homeStreet + "\"></strong></font></pre>"+
        "<pre><font size=\"3\"><strong>P.O.Box: <input type=\"text\" size=\"6\" name↙
          =\"homePObox\" value=\"" + homePObox + "\"></strong></font></pre>"+
        "<pre><font size=\"3\"><strong>City:   <input type=\"text\" size=\"11\" name ↙
          =\"homeCity\" value=\"" + homeCity + "\">"+
        " State:<input type=\"text\" size=\"4\" name=\"homeState\" value=\"" +        ↙
          homeState + "\">"+
        " Zip:<input type=\"text\" size=\"6\" name=\"homeZip\" value=\"" + homeZip ↙
          + "\"></strong></font></pre>"+
        "<pre><font size=\"3\"><strong>Country: <input type=\"text\" size=\"10\"      ↙
          name=\"homeCountry\" value=\"" + homeCountry + "\">"+
        " Tel: <input type=\"text\" size=\"12\" name=\"homeTel\" value=\"" + homeTel↙
          + "\"></strong></font></pre>"+
        "<pre><font size=\"3\"><strong>Email:   <input type=\"text\" size=\"25\" name↙
          =\"homeEmail\" value=\"" + homeEmail + "\"></strong></font></pre>"+
        "</td><td width=\"50%\" bgcolor=\"#C0C0C0\"><pre><font size=\"3\">"+
        "<strong>Company: <input type=\"text\" size=\"22\" name=\"company\" value ↙
          =\"" + company + "\"></strong></font></pre>"+
        "<pre><font size=\"3\"><strong>Title:   <input type=\"text\" size=\"21\" name ↙
          =\"workTitle\" value=\"" + workTitle + "\"></strong></font></pre>"+
        "<pre><font size=\"3\"><strong>Street   <input type=\"text\" size=\"25\" name ↙
          =\"workStreet\" value=\"" + workStreet + "\"></strong></font></pre>"+
        "<pre><font size=\"3\"><strong>P.O.Box: <input type=\"text\" size=\"6\" name↙
          =\"workPObox\" value=\"" + workPObox + "\"></strong></font></pre>"+
        "<pre><font size=\"3\"><strong>City:   <input type=\"text\" size=\"11\" name ↙
          =\"workCity\" value=\"" + workCity + "\">"+
        "State:<input type=\"text\" size=\"4\" name=\"workState\" value=\"" +         ↙
          workState + "\"> Zip:<input type=\"text\" size=\"6\" name=\"workZip\" value↙
          =\"" + workZip + "\"></strong></font></pre>"+
        "<pre><font size=\"3\"><strong>Country: <input type=\"text\" size=\"10\"      ↙
          name=\"workCountry\" value=\"" + workCountry + "\">"+
        " Tel: <input type=\"text\" size=\"12\" name=\"worktel\" value=\"" + worktel  ↙
          + "\"></strong></font></pre>"+
        "<pre><font size=\"3\"><strong>Email:   <input type=\"text\" size=\"25\" name↙
          =\"workEmail\" value=\"" + workEmail + "\"></strong></font></pre>"+
        "</td>"+
      "</tr></table>");
```

```
    out.println("<table border=\"0\" width=\"100%\">"+
            "<tr>"+
              "<td width=\"50%\" bgcolor=\"#ECEFC7\"><font size=\"3\"><strong>Please"+
              " tell us about yourself or your comments:</strong></font></td>"+
              "<td width=\"50%\"><font size=\"3\"><strong></strong></font>&nbsp;</td>"+
            "</tr>"+
            "<tr>"+
              "<td width=\"50%\" bgcolor=\"#ECEFC7\"><font size=\"3\"><strong><textarea ↙
                "+
              "name=\"comment\" rows=\"3\" cols=\"48\">"+comment+"</textarea></strong><↙
                font></td>"+
              "<td width=\"50%\" bgcolor=\"#D39D96\"><p align=\"center\"><font size=\"3 ↙
                \">"+
              "<strong>To update your record in the future,</strong></font></p>"+
              "<p align=\"center\"><font size=\"3\"><strong>please choose </strong></font></↙
                p>"+
              "<div align=\"center\"><center><pre><font size=\"3\" face=\"Times New Roman↙
                \">"+
              "<strong>user Id:</strong></font><font size=\"3\">"+
              "<strong><input type=\"text\" size=\"6\" name=\"userId\" value=\"" + userId     ↙
                + "\">"+
              "</strong></font><font size=\"3\" face=\"Times New Roman\"><strong>            ↙
                password: </strong></font>"+
              "<font size=\"3\"><strong>"+
              "<input type=\"password\" size=\"6\" name=\"password\" value=\"" + password   ↙
                + "\"></strong></font></pre>"+
              "</center></div></td>"+
            "</tr>"+
          "</table>"+
          "<p align=\"center\"><font size=\"3\"><strong>"+
          "<input type=\"submit\" name=\"Action\" value=\"Register\">"+
          "<input type=\"reset\" name=\"Reset\" value=\"Clear Form\">"+
          "<input type=\"submit\" name=\"Action\" value=\"Exit\"></p></strong></font>"+
          "</form>"+
          "</body>"+
          "</html>");
    out.flush();
    out.close();
  }//end of sendRegistrationForm

  private void sendConfirmation(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    String serverName = req.getServerName();
    int serverPort = req.getServerPort();
```

```
            "<p align=\"center\"><font size=\"3\"><strong>please choose </strong></font></↙
              p>"+
            "<div align=\"center\"><center><pre><font size=\"3\" face=\"Times New Roman↙
              \">"+
            "<strong>user Id:</strong></font><font size=\"3\">"+
            "<strong><input type=\"text\" size=\"6\" name=\"userId\" value=\"" + userId     ↙
              + "\">"+
            "</strong></font><font size=\"3\" face=\"Times New Roman\"><strong>              ↙
              password: </strong></font>"+
            "<font size=\"3\"><strong>"+
            "<input type=\"password\" size=\"6\" name=\"password\" value=\"" + password   ↙
              + "\"></strong></font></pre>"+
            "</center></div></td>"+
          "</tr>"+
        "</table>"+
        "<p align=\"center\"><font size=\"3\"><strong>"+
        "<input type=\"submit\" name=\"Action\" value=\"Update\">"+
        "<input type=\"submit\" name=\"Action\" value=\"Delete\">"+
        "<input type=\"submit\" name=\"Action\" value=\"Cancel\"></p></strong></font>"+
        "</form>"+
        "</body>"+
        "</html>");

    out.flush();
    out.close();
  }//end of sendUpdateForm

  private void sendConfirmation(HttpServletRequest req, HttpServletResponse res, String      ↙
    message)
    throws ServletException, IOException
  {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    String serverName = req.getServerName();
    int serverPort = req.getServerPort();
    String userId = req.getParameter("UserId");

    out.println("<html>"+
          "<head>"+
          "<title>Confirmation Page</title></head>"+
          "<body bgcolor=\"#FFFFFF\">"+
          "<p align=\"center\"><font size=\"6\">Confirmation Page</font></p>"+
          "<p align=\"center\"><font size=\"6\">" + message + "</font></p>"+
          "<p align=\"center\"><a href=http://" + serverName + ":" + serverPort + "/members.↙
            html>Back to main page</a></p>"+
          "</body></html>");
    out.flush();
    out.close();
  }//end of confirmation

  private void sendProblemMessage(HttpServletRequest req, HttpServletResponse res, String   ↙
    message)
    throws ServletException, IOException
  {
    // set content type header before accessing the Writer
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    // then write the response
    out.println("<html>" +
            "<head><title> Receipt </title>" +
            "<meta http-equiv=\"refresh\" content=\"4; url=" +
            "http://" + req.getHeader("Host") +
            "/members.html;\"></head><body  bgcolor=\"#FFFFFF\">" +
            "<center><hr> <br> &nbsp;<br> &nbsp; <hr> <br> &nbsp;");

    out.println("<h3>We had a problem and can not process your request."+
            "<p>" + message + "</h3>" +
            "<p><i>This page automatically resets.</i>" +
            "</body></html>");
    out.flush();
    out.close();
  }//end of sendProblemMessage

}//end of servlet
```

```
                "<th>TITLE          "+
                "<th>WORK STREET    "+
                "<th>WORK P.O. BOX "+
                "<th>WORK CITY"+
                "<th>WORK STATE"+
                "<th>WORK ZIP"+
                "<th>WORK COUNTRY"+
                "<th>WORK TELEPHONE"+
                "<th>WORK EMAIL       "+
                "<th>COMMENTS                          </font></th>");


        for (int i = 0; i < members.length; i++)
        {
          out.println("<TR>");
          out.println("<TD>" + members[i].gettitle());
          out.println("<TD>" + members[i].getfirstName());
          out.println("<TD>" + members[i].getmiddleInitial());
          out.println("<TD>" + members[i].getlastName());
          out.println("<TD>" + members[i].getdegree1());
          out.println("<TD>" + members[i].getmajor1());
          out.println("<TD>" + members[i].getgradDate1());
          //out.println("<TD>" + members[i].getdegree2());
          //out.println("<TD>" + members[i].getmajor2());
          //out.println("<TD>" + members[i].getgradDate2());
          out.println("<TD>" + members[i].gethomeStreet());
          out.println("<TD>" + members[i].gethomePObox());
          out.println("<TD>" + members[i].gethomeCity());
          out.println("<TD>" + members[i].gethomeState());
          out.println("<TD>" + members[i].gethomeZip());
          out.println("<TD>" + members[i].gethomeCountry());
          out.println("<TD>" + members[i].gethomeTel());
          out.println("<TD>" + members[i].gethomeEmail());
          out.println("<TD>" + members[i].getcompany());
          out.println("<TD>" + members[i].getworkTitle());
          out.println("<TD>" + members[i].getworkStreet());
          out.println("<TD>" + members[i].getworkPObox());
          out.println("<TD>" + members[i].getworkCity());
          out.println("<TD>" + members[i].getworkState());
          out.println("<TD>" + members[i].getworkZip());
          out.println("<TD>" + members[i].getworkCountry());
          out.println("<TD>" + members[i].getworktel());
          out.println("<TD>" + members[i].getworkEmail());
          out.println("<TD>" + members[i].getcomment());

        }
        out.println("</TABLE><HR WIDTH=100%>"+
                    "<p align=\"center\"><a href=http://" + serverName + ":" + serverPort + "/↙
                      search.html>Try again</a></p>"+
                    "<p align=\"center\"><a href=http://" + serverName + ":" + serverPort + "/↙
                      members.html>Exit</a></p>"+
                    "</body></html>");
             out.flush();
             out.close();

           }
         }
         catch (RegistryException e)
         {
           e.printStackTrace();
           String message = "There is a problem with our database.";
           sendProblemMessage(request, response, message);

         }
       }
       else //no search data was entered
       {
         //session.invalidate();
         String message = "You did not enter any data to search. Try again!";
         sendConfirmation(request, response, message);
         // need to go back to search servlet
       }
     }
     else // exit requested
     {
       response.sendRedirect("/members.html");
     }

   }//end of service


   private void sendConfirmation(HttpServletRequest req, HttpServletResponse res, String      ↙
     message)
     throws ServletException, IOException
   {
     res.setContentType("text/html");
     PrintWriter out = res.getWriter();
     String serverName = req.getServerName();
     int serverPort = req.getServerPort();

     out.println("<html>"+
           "<head>"+
           "<title>Confirmation Page</title></head>"+
           "<body bgcolor=\"#FFFFFF\">"+
            "<p align=\"center\"><font size=\"6\">Confirmation Page</font></p>"+
            "<p align=\"center\"><font size=\"6\">" + message + "</font></p>"+
            "<p align=\"center\"><a href=http://" + serverName + ":" + serverPort + "/search.  ↙
              html>Try again</a></p>"+
            "<p align=\"center\"><a href=http://" + serverName + ":" + serverPort + "/members.↙
              html>Exit</a></p>"+
            "</body></html>");
    out.flush();
    out.close();
  }//end of confirmation

  private void sendProblemMessage(HttpServletRequest req, HttpServletResponse res, String   ↙
    message)
    throws ServletException, IOException
  {
    // set content type header before accessing the Writer
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    // then write the response
    out.println("<html>" +
            "<head><title> Problem </title>" +
            "<meta http-equiv=\"refresh\" content=\"4; url=" +
            "http://" + req.getHeader("Host") +
            "/members.html;\"></head><body  bgcolor=\"#FFFFFF\">" +
            "<center><hr> <br> &nbsp;<br> &nbsp; <hr> <br> &nbsp;");

    out.println("<h3>Sorry, we can not process your request."+
            "<p>" + message + "</h3>" +
            "<p><i>This page automatically resets.</i>" +
            "</body></html>");
    out.flush();
    out.close();
  }//end of sendProblemMessage

}//end of servlet
```

```
/**
 * Implementation of the UserRegistry.
 */
import java.util.*;
import java.sql.*;


public class UserRegistryJdbcImpl implements UserRegistry
{
  /**
   * Name of Table entries are queried from.
   */
  final static private String _driver = "sun.jdbc.odbc.JdbcOdbcDriver";
  final static private String _user = "";
  final static private String _pass = "";
  final static private String _url = "jdbc:odbc:members";

  /**
   * Database Connection.
   */
  static private Properties _dbProperties = null;
  static private Driver _jdbcDriver = null;
  static private Connection _db = null;

  /**
   * Insert and Update statements
   */
  static private PreparedStatement _insertStatement;
  static private PreparedStatement _updateStatement;
  static private PreparedStatement _queryStatement;
  static private PreparedStatement _deleteStatement;
  static private PreparedStatement _searchStatement;

  /**
   * search query
   */
  static private String query;


  /**
   * Initialize driver. We do this here so that we only have
   * one driver to use for all instances of this class.
   */
  static
  {
    try
    {
        _jdbcDriver = (Driver)Class.forName (_driver).newInstance();
        _dbProperties = new Properties();
        _dbProperties.put ("user", _user);
        _dbProperties.put ("password", _pass);
        _init();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

/**
 * Returns an array of found AlumniMembers
 * @param userId Search text.
 * @return AlumniMember[] Array AlumniMembers instances found in the database.
 * @throws RegistryException Error occured in database.
 */
public AlumniMember[] getMember (String UserId)  throws RegistryException
{
    Vector members = new Vector();
    String sql;
    ResultSet r = null;
    Statement statement = null;

    synchronized (_jdbcDriver)
    {
        try
        {
            // Check to see if the database connection
            // is still open, if not re-open.
            if ( _db.isClosed () )
            {
                _init();
            }

            // Now, lets do the query.
            AlumniMember member;
            _queryStatement.setString (1, UserId);
            r = _queryStatement.executeQuery();
            while ( r.next() )
            {
                member = new AlumniMember ();

                member.setuserId (r.getString ("USERID"));
            member.setpassword (r.getString ("PASSWORD"));
            member.settitle (r.getString ("TITLE"));
            member.setfirstName (r.getString ("FIRSTNAME"));
            member.setmiddleInitial (r.getString ("MIDDLEINITIAL"));
            member.setlastName (r.getString ("LASTNAME"));
            member.setdegree1 (r.getString ("DEGREE1"));
            member.setmajor1 (r.getString ("MAJOR1"));
            member.setgradDate1 (r.getString ("GRADDATE1"));
            member.setdegree2 (r.getString ("DEGREE2"));
            member.setmajor2 (r.getString ("MAJOR2"));
            member.setgradDate2 (r.getString ("GRADDATE2"));
            member.sethomeStreet (r.getString ("HOMESTREET"));
            member.sethomePObox (r.getString ("HOMEPOBOX"));
            member.sethomeCity (r.getString ("HOMECITY"));
            member.sethomeState (r.getString ("HOMESTATE"));
            member.sethomeZip (r.getString ("HOMEZIP"));
            member.sethomeCountry (r.getString ("HOMECOUNTRY"));
            member.sethomeTel (r.getString ("HOMETEL"));
            member.sethomeEmail (r.getString ("HOMEEMAIL"));
            member.setcompany (r.getString ("COMPANY"));
            member.setworkTitle (r.getString ("WORKTITLE"));
            member.setworkStreet (r.getString ("WORKSTREET"));
            member.setworkPObox (r.getString ("WORKPOBOX"));
            member.setworkCity (r.getString ("WORKCITY"));
            member.setworkState (r.getString ("WORKSTATE"));
            member.setworkZip (r.getString ("WORKZIP"));
            member.setworkCountry (r.getString ("WORKCOUNTRY"));
            member.setworktel (r.getString ("WORKTEL"));
            member.setworkEmail (r.getString ("WORKEMAIL"));
            member.setcomment (r.getString ("COMMENT"));

            member.clearChanged();
            members.addElement (member);
          }
        }
        catch (SQLException e)
        {
          //throw new RegistryException (e.getMessage());
          throw new RegistryException ("error in getMember");
        }
        finally
        {
          try
          {
            _queryStatement.clearParameters();
          }
          catch (SQLException e)
```

```
            {
               e.printStackTrace();
            }
            try
            {
               if ( r != null )
               {
                  r.close();
               }
            }
            catch (SQLException e)
            {
               e.printStackTrace();
            }
            try
            {
               if ( _db != null )
               {
                  _db.close();
               }
            }
            catch (SQLException e)
            {
               e.printStackTrace();
            }
         }
      }
      AlumniMember memberArray[] = new AlumniMember[members.size()];
      members.copyInto (memberArray);
      return memberArray;
   }

   /**
    * Adds a member to the database.
    * @param member The AlumniMember to add/replace.
    * @return AlumniMember The original value of the member or null if
    * it didn't exist.
    * @throws RegistryException
    */
   public AlumniMember setMember (AlumniMember member) throws RegistryException
   {
      AlumniMember hold[];
      AlumniMember oldMember = null;
      ResultSet results = null;
      PreparedStatement statement = null;

      // See if member exists
    hold = getMember (member.getuserId());

    synchronized (_jdbcDriver)
    {
      try
      {
        // Check to see if the database connection
        // is still open, if not re-open.
        if ( _db.isClosed () )
        {
          _init();
        }
        int resultCode;

        // We've found an existing member, so keep it because
        // we return the old value. Also, we must do an update
        // not an insert.
        if ( hold.length == 1 )
        {
          oldMember = hold[0];

          _updateStatement.setString (31, member.getuserId());
          _updateStatement.setString (1, member.getpassword());
          _updateStatement.setString (2, member.gettitle());
          _updateStatement.setString (3, member.getfirstName());
          _updateStatement.setString (4, member.getmiddleInitial());
          _updateStatement.setString (5, member.getlastName());
          _updateStatement.setString (6, member.getdegree1());
          _updateStatement.setString (7, member.getmajor1());
          _updateStatement.setString (8, member.getgradDate1());
          _updateStatement.setString (9, member.getdegree2());
          _updateStatement.setString (10, member.getmajor2());
          _updateStatement.setString (11, member.getgradDate2());
          _updateStatement.setString (12, member.gethomeStreet());
          _updateStatement.setString (13, member.gethomePObox());
          _updateStatement.setString (14, member.gethomeCity());
          _updateStatement.setString (15, member.gethomeState());
          _updateStatement.setString (16, member.gethomeZip());
          _updateStatement.setString (17, member.gethomeCountry());
          _updateStatement.setString (18, member.gethomeTel());
          _updateStatement.setString (19, member.gethomeEmail());
          _updateStatement.setString (20, member.getcompany());
          _updateStatement.setString (21, member.getworkTitle());
          _updateStatement.setString (22, member.getworkStreet());
          _updateStatement.setString (23, member.getworkPObox());
          _updateStatement.setString (24, member.getworkCity());
          _updateStatement.setString (25, member.getworkState());
        _updateStatement.setString (26, member.getworkZip());
        _updateStatement.setString (27, member.getworkCountry());
        _updateStatement.setString (28, member.getworktel());
        _updateStatement.setString (29, member.getworkEmail());
        _updateStatement.setString (30, member.getcomment());

        statement = _updateStatement;
        resultCode = statement.executeUpdate();
      }

      // No existing member, so do an insert.
      else if ( hold.length == 0 )
      {
        _insertStatement.setString (1, member.getuserId());
        _insertStatement.setString (2, member.getpassword());
        _insertStatement.setString (3, member.gettitle());
        _insertStatement.setString (4, member.getfirstName());
        _insertStatement.setString (5, member.getmiddleInitial());
        _insertStatement.setString (6, member.getlastName());
        _insertStatement.setString (7, member.getdegree1());
        _insertStatement.setString (8, member.getmajor1());
        _insertStatement.setString (9, member.getgradDate1());
        _insertStatement.setString (10, member.getdegree2());
        _insertStatement.setString (11, member.getmajor2());
        _insertStatement.setString (12, member.getgradDate2());
        _insertStatement.setString (13, member.gethomeStreet());
        _insertStatement.setString (14, member.gethomePObox());
        _insertStatement.setString (15, member.gethomeCity());
        _insertStatement.setString (16, member.gethomeState());
        _insertStatement.setString (17, member.gethomeZip());
        _insertStatement.setString (18, member.gethomeCountry());
        _insertStatement.setString (19, member.gethomeTel());
        _insertStatement.setString (20, member.gethomeEmail());
        _insertStatement.setString (21, member.getcompany());
        _insertStatement.setString (22, member.getworkTitle());
        _insertStatement.setString (23, member.getworkStreet());
        _insertStatement.setString (24, member.getworkPObox());
        _insertStatement.setString (25, member.getworkCity());
        _insertStatement.setString (26, member.getworkState());
        _insertStatement.setString (27, member.getworkZip());
        _insertStatement.setString (28, member.getworkCountry());
        _insertStatement.setString (29, member.getworktel());
        _insertStatement.setString (30, member.getworkEmail());
        _insertStatement.setString (31, member.getcomment());

        statement = _insertStatement;
        resultCode = statement.executeUpdate ();
```

```
            }

            // More then one member with the given user id is in the
            // database - this should never happen if we are
            // using user id as a unqiue key.
            else
            {
               throw new RegistryException ("Duplicate UserId!");
            }

            // Problem occured while doing the update/insert.
            if ( resultCode == -1 )
            {
               throw new RegistryException ("Result Code: " + resultCode);
            }
         }
         catch (SQLException e)
         {
            String s = e.getMessage();
            throw new RegistryException (e.getMessage());
         }
         finally
         {
            if ( statement != null )
            {
               try
               {
                  statement.clearParameters();
               }
               catch (SQLException e)
               {
                  e.printStackTrace();
               }
            }
            try
            {
               if ( _db != null )
               {
                  _db.close();
               }
            }
            catch (SQLException e)
            {
               e.printStackTrace();
            }
         }
      }
```

```
    return oldMember;
}

/**
 * Deletes a member from the database.
 * @param member The AlumniMember to delete.
 * @return AlumniMember The original value of the member or null if
 * it didn't exist.
 * @throws RegistryException
 */
public AlumniMember deleteMember (AlumniMember member) throws RegistryException
{
    AlumniMember hold[];
    AlumniMember oldMember = null;
    ResultSet results = null;
    PreparedStatement statement = null;

    // See if member exists
    hold = getMember (member.getuserId());

    synchronized (_jdbcDriver)
    {
        try
        {
            // Check to see if the database connection
            // is still open, if not re-open.
            if ( _db.isClosed () )
            {
                _init();
            }
            int resultCode;

            // We've found an existing member, so keep it because
            // we return the old value. Also, we must do an update
            // not an insert.
            if ( hold.length == 1 )
            {
                oldMember = hold[0];

                _deleteStatement.setString (1, member.getuserId());
                statement = _deleteStatement;
                resultCode = statement.executeUpdate();
            }

            // More then one member with the given user id or record is not
            // available for delete.
            else
            {
                throw new RegistryException ("Duplicate or unavailable userId!");
            }

        }
        catch (SQLException e)
        {
            String s = e.getMessage();
            throw new RegistryException (e.getMessage());
        }
        finally
        {
            if ( statement != null )
            {
                try
                {
                    statement.clearParameters();
                }
                catch (SQLException e)
                {
                    e.printStackTrace();
                }
            }
            try
            {
                if ( _db != null )
                {
                    _db.close();
                }
            }
            catch (SQLException e)
            {
                e.printStackTrace();
            }
        }
    }
    return oldMember;
  }

  /**
   * Searches the database with a given column name and a string.
   * @param columnName
   * @param searchString
   * @return the array of members with matching criteria.
   * @throws RegistryException
   */
  public AlumniMember[] searchRegistry (String columnName, String searchString) throws     ↙
```

```
    RegistryException
  {
    Vector members = new Vector();
    ResultSet r = null;
    PreparedStatement statement = null;

    synchronized (_jdbcDriver)
    {
      try
      {
        query = "select * from UTH where (" + columnName + " like '" + searchString + "%')";

        // Check to see if the database connection
        // is still open, if not re-open.
        if ( _db.isClosed () )
        {
          _init();
        }

        // Now, lets do the query.
        AlumniMember member;

        _searchStatement = _db.prepareStatement (query);

        r = _searchStatement.executeQuery();

        while ( r.next() )
        {
          member = new AlumniMember ();

          member.setuserId (nullToString(r.getString ("USERID")));
          member.setpassword (nullToString(r.getString ("PASSWORD")));
          member.settitle (nullToString(r.getString ("TITLE")));
          member.setfirstName (nullToString(r.getString ("FIRSTNAME")));
          member.setmiddleInitial (nullToString(r.getString ("MIDDLEINITIAL")));
          member.setlastName (nullToString(r.getString ("LASTNAME")));
          member.setdegree1 (nullToString(r.getString ("DEGREE1")));
          member.setmajor1 (nullToString(r.getString ("MAJOR1")));
          member.setgradDate1 (nullToString(r.getString ("GRADDATE1")));
          member.setdegree2 (nullToString(r.getString ("DEGREE2")));
          member.setmajor2 (nullToString(r.getString ("MAJOR2")));
          member.setgradDate2 (nullToString(r.getString ("GRADDATE2")));
          member.sethomeStreet (nullToString(r.getString ("HOMESTREET")));
          member.sethomePObox (nullToString(r.getString ("HOMEPOBOX")));
          member.sethomeCity (nullToString(r.getString ("HOMECITY")));
          member.sethomeState (nullToString(r.getString ("HOMESTATE")));
          member.sethomeZip (nullToString(r.getString ("HOMEZIP")));
          member.sethomeCountry (nullToString(r.getString ("HOMECOUNTRY")));
          member.sethomeTel (nullToString(r.getString ("HOMETEL")));
          member.sethomeEmail (nullToString(r.getString ("HOMEEMAIL")));
          member.setcompany (nullToString(r.getString ("COMPANY")));
          member.setworkTitle (nullToString(r.getString ("WORKTITLE")));
          member.setworkStreet (nullToString(r.getString ("WORKSTREET")));
          member.setworkPObox (nullToString(r.getString ("WORKPOBOX")));
          member.setworkCity (nullToString(r.getString ("WORKCITY")));
          member.setworkState (nullToString(r.getString ("WORKSTATE")));
          member.setworkZip (nullToString(r.getString ("WORKZIP")));
          member.setworkCountry (nullToString(r.getString ("WORKCOUNTRY")));
          member.setworktel (nullToString(r.getString ("WORKTEL")));
          member.setworkEmail (nullToString(r.getString ("WORKEMAIL")));
          member.setcomment (nullToString(r.getString ("COMMENT")));

          member.clearChanged();
          members.addElement (member);
        }


      }
      catch (SQLException e)
      {
        String s = e.getMessage();
        throw new RegistryException (e.getMessage());
      }
      finally
      {
        try
        {
          _searchStatement.clearParameters();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
        try
        {
          if ( r != null )
          {
            r.close();
          }
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
```

```
          try
          {
             if ( _db != null )
             {
                _db.close();
             }
          }
          catch (SQLException e)
          {
             e.printStackTrace();
          }
       }
    }
    AlumniMember memberArray[] = new AlumniMember[members.size()];
    members.copyInto (memberArray);
    return memberArray;
 }
 /**
  * Convience method to initalize the database connection
  * and create the Prepared statements.
  * @return void
  */
 static private void _init()
 {
    synchronized (_jdbcDriver)
    {
       try
       {
          _db = _jdbcDriver.connect (_url, _dbProperties);

          _insertStatement = _db.prepareStatement ("insert into UTH (userId,"+
             "password,title,firstName,middleInitial,lastName,degree1,major1,gradDate1,"+
             "degree2,major2,gradDate2,homeStreet,homePObox,homeCity,homeState,homeZip ↙
                ,"+
             "homeCountry,homeTel,homeEmail,company,workTitle,workStreet,workPObox,    ↙
                workCity,"+
             "workState,workZip,workCountry,worktel,workEmail,comment) "+
             "values (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)");

          _updateStatement = _db.prepareStatement ("update UTH set "+
             "password=?,title=?,firstName=?,middleInitial=?,lastName=?,degree1=?,major1=?, ↙
                gradDate1=?,"+
             "degree2=?,major2=?,gradDate2=?,homeStreet=?,homePObox=?,homeCity=?,       ↙
                homeState=?,homeZip=?,"+
             "homeCountry=?,homeTel=?,homeEmail=?,company=?,workTitle=?,workStreet=?, ↙
                workPObox=?,workCity=?,"+
             "workState=?,workZip=?,workCountry=?,worktel=?,workEmail=?,comment=?     ↙
```

```
                where userid=?");
            _queryStatement = _db.prepareStatement ("select * from UTH where userid=?");
            _deleteStatement = _db.prepareStatement ("delete from UTH where userid=?");


        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }
    }
}

/**
 * This method returns an empty string if a given field has a null value in it.
 */
static private String nullToString(String fieldValue)
{
    if (fieldValue == null)
        return "";
    else
        return fieldValue;
}
}
```

```
/**
 * Interface for UserRegistryJDBCImpl class.
 */

public interface UserRegistry
{
  /**
   * Returns an array of AlumniMembers based on the search criteria.
   * @param user id User id to search for.
   * @return AlumniMember[] Array AlumniMembers instances found in the database.
   * @throws RegistryException
   */
  public AlumniMember[] getMember (String text)  throws RegistryException;


  /**
   * Adds a member to the database.
   * @param member The AlumniMember to add/replace.
   * @return AlumniMember The original value of the member or null if
   * it didn't exist.
   * @throws RegistryException
   */
  public AlumniMember setMember (AlumniMember member) throws RegistryException;


/**
   * Adds a member to the database.
   * @param member The AlumniMember to delete.
   * @return AlumniMember The original value of the member or null if
   * it didn't exist.
   * @throws RegistryException
   */
  public AlumniMember deleteMember (AlumniMember member) throws RegistryException;

  /**
   * Searches the database with a given column name and a string.
   * @param columnName
   * @param searchString
   * @return the result set.
   * @throws RegistryException
   */
  public AlumniMember[] searchRegistry (String columnName, String searchString) throws   ↙
    RegistryException;

}
```

```
/**
 * Exception thrown if an error occurs in the UserRegistry.
 *
 */
public class RegistryException extends Exception
{
   /**
    * Create a new RegistryException instance
    * with a message.
    * @param s Exception message.
    */
   public RegistryException (String s)
   {
      super(s);
   }

   /**
    * Create a new RegistryException with no message
    */
   public RegistryException()
   {
      super();
   }
}
```

```
/**
 * Implementation of AlumniMember class.
 *
 */

public class AlumniMember implements Cloneable
{

   private String userId;
   private String password;
   private String title;
   private String firstName;
   private String middleInitial;
   private String lastName;
   private String degree1;
   private String major1;
   private String gradDate1;
   private String degree2;
   private String major2;
   private String gradDate2;
   private String homeStreet;
   private String homePObox;
   private String homeCity;
   private String homeState;
   private String homeZip;
   private String homeCountry;
   private String homeTel;
   private String homeEmail;
   private String company;
   private String workTitle;
   private String workStreet;
   private String workPObox;
   private String workCity;
   private String workState;
   private String workZip;
   private String workCountry;
   private String worktel;
   private String workEmail;
   private String comment;
   private Boolean _changed;

   /**
    * Create a new AlumniMember instance.
    */
   public AlumniMember()
   {
        userId = "";
        password = "";
        title = "";
        firstName = "";
        middleInitial = "";
        lastName = "";
        degree1 = "";
        major1 = "";
        gradDate1 = "";
        degree2 = "";
        major2 = "";
        gradDate2 = "";
        homeStreet = "";
        homePObox = "";
        homeCity = "";
        homeState = "";
        homeZip = "";
        homeCountry = "";
        homeTel = "";
        homeEmail = "";
        company = "";
        workTitle = "";
        workStreet = "";
        workPObox = "";
        workCity = "";
        workState = "";
        workZip = "";
        workCountry = "";
        worktel = "";
        workEmail = "";
        comment = "";
        _changed = new Boolean (false);
    }

    public void setuserId (String s) {userId = s;}
    public void setpassword (String s) {password = s;}
    public void settitle (String s) {if (s != null) title = s; else title = "";}
    public void setfirstName (String s) {if (s != null) firstName = s; else firstName = "";}
    public void setmiddleInitial (String s) {if (s != null) middleInitial = s; else middleInitial = "";}
    public void setlastName (String s) {if (s != null) lastName = s; else lastName = "";}
    public void setdegree1 (String s) {if (s != null) degree1 = s; else degree1 = "";}
    public void setmajor1 (String s) {if (s != null) major1 = s; else major1 = "";}
    public void setgradDate1 (String s) {if (s != null) gradDate1 = s; else gradDate1 = "";}
    public void setdegree2 (String s) {if (s != null) degree2 = s; else degree2 = "";}
    public void setmajor2 (String s) {if (s != null) major2 = s; else major2 = "";}
    public void setgradDate2 (String s) {if (s != null) gradDate2 = s; else gradDate2 = "";}
    public void sethomeStreet (String s) {if (s != null) homeStreet = s; else homeStreet = "";}
```

```
public void sethomePObox (String s) {if (s != null) homePObox = s; else homePObox = "";}
public void sethomeCity (String s) {if (s != null) homeCity = s; else homeCity = "";}
public void sethomeState (String s) {if (s != null) homeState = s; else homeState = "";}
public void sethomeZip (String s) {if (s != null) homeZip = s; else homeZip = "";}
public void sethomeCountry (String s) {if (s != null) homeCountry = s; else homeCountry ↙
  = "";}
public void sethomeTel (String s) {if (s != null) homeTel = s; else homeTel = "";}
public void sethomeEmail (String s) {if (s != null) homeEmail = s; else homeEmail = "";}
public void setcompany (String s) {if (s != null) company = s; else company = "";}
public void setworkTitle (String s) {if (s != null) workTitle = s; else workTitle = "";}
public void setworkStreet (String s) {if (s != null) workStreet = s; else workStreet = "";}
public void setworkPObox (String s) {if (s != null) workPObox = s; else workPObox = "";}
public void setworkCity (String s) {if (s != null) workCity = s; else workCity = "";}
public void setworkState (String s) {if (s != null) workState = s; else workState = "";}
public void setworkZip (String s) {if (s != null) workZip = s; else workZip = "";}
public void setworkCountry (String s) {if (s != null) workCountry = s; else workCountry = "";}
public void setworktel (String s) {if (s != null) worktel = s; else worktel = "";}
public void setworkEmail (String s) {if (s != null) workEmail = s; else workEmail = "";}
public void setcomment (String s) {if (s != null) comment = s; else comment = "";}


public String getuserId () {return new String (userId);}
public String getpassword () {return new String (password);}
public String gettitle () {return new String (title);}
public String getfirstName () {return new String (firstName);}
public String getmiddleInitial () {return new String (middleInitial);}
public String getlastName () {return new String (lastName);}
public String getdegree1 () {return new String (degree1);}
public String getmajor1 () {return new String (major1);}
public String getgradDate1 () {return new String (gradDate1);}
public String getdegree2 () {return new String (degree2);}
public String getmajor2 () {return new String (major2);}
public String getgradDate2 () {return new String (gradDate2);}
public String gethomeStreet () {return new String (homeStreet);}
public String gethomePObox () {return new String (homePObox);}
public String gethomeCity () {return new String (homeCity);}
public String gethomeState () {return new String (homeState);}
public String gethomeZip () {return new String (homeZip);}
public String gethomeCountry () {return new String (homeCountry);}
public String gethomeTel () {return new String (homeTel);}
public String gethomeEmail () {return new String (homeEmail);}
public String getcompany () {return new String (company);}
public String getworkTitle () {return new String (workTitle);}
public String getworkStreet () {return new String (workStreet);}
public String getworkPObox () {return new String (workPObox);}
public String getworkCity () {return new String (workCity);}
public String getworkState () {return new String (workState);}
public String getworkZip () {return new String (workZip);}
public String getworkCountry () {return new String (workCountry);}
public String getworktel () {return new String (worktel);}
public String getworkEmail () {return new String (workEmail);}
public String getcomment () {return new String (comment);}

/**
 * Clone a AlumniMember instance
 * @return Object AlumniMember instance copy.
 */
public Object clone()
{
  AlumniMember clone = new AlumniMember();

  clone.setuserId (this.getuserId());
  clone.setpassword (this.getpassword());
  clone.settitle (this.gettitle());
  clone.setfirstName (this.getfirstName());
  clone.setmiddleInitial (this.getmiddleInitial());
  clone.setlastName (this.getlastName());
  clone.setdegree1 (this.getdegree1());
  clone.setmajor1 (this.getmajor1());
  clone.setgradDate1 (this.getgradDate1());
  clone.setdegree2 (this.getdegree2());
  clone.setmajor2 (this.getmajor2());
  clone.setgradDate2 (this.getgradDate2());
  clone.sethomeStreet (this.gethomeStreet());
  clone.sethomePObox (this.gethomePObox());
  clone.sethomeCity (this.gethomeCity());
  clone.sethomeState (this.gethomeState());
  clone.sethomeZip (this.gethomeZip());
  clone.sethomeCountry (this.gethomeCountry());
  clone.sethomeTel (this.gethomeTel());
  clone.sethomeEmail (this.gethomeEmail());
  clone.setcompany (this.getcompany());
  clone.setworkTitle (this.getworkTitle());
  clone.setworkStreet (this.getworkStreet());
  clone.setworkPObox (this.getworkPObox());
  clone.setworkCity (this.getworkCity());
  clone.setworkState (this.getworkState());
  clone.setworkZip (this.getworkZip());
  clone.setworkCountry (this.getworkCountry());
  clone.setworktel (this.getworktel());
  clone.setworkEmail (this.getworkEmail());
  clone.setcomment (this.getcomment());
  return clone;
}
```

```
    /**
     * Indicates if the instance has changed, meaning
     * one of the set methods was called.
     * @return Boolean True if changed, False otherwise
     */
    public Boolean isChanged()
    {
        Boolean retvalue = new Boolean (_changed.booleanValue());
        _changed = new Boolean (false);
        return retvalue;
    }

    /**
     * Clear the value of changed
     * @return void
     */
    public void clearChanged()
    {
        _changed = new Boolean (false);
    }
}
```

BIBLIOGRAPHY

[1] George Reese, Database Programming with JDBC and Java. O'Reilly, 1997.

[2] Roger Jennings, Using Access 97. Que Corporation, 1997.

[3] G. Hamilton, R. Cattell, JDBC: A Java SQL API. JavaSoft, January 1997.

[4] JavaSoft Web site, "JDBC Drivers". Sun Microsystems Inc., Source: "http://java.sun.com/products/jdbc/jdbc.drivers.html" October 1998.

[5] JavaSoft Web site, "The JDBC Database Access API". Sun Microsystems Inc., Source: "http://java.sun.com/products/jdbc/JDBC-ODBC Bridge Driver" October 1997.

[6] Sun Microsystems, "The JDBC ™ API Version 1.20" Source: "http://splash.javasoft.com/jdbc" January 1997.

[7] Caribou Lake Software Web site, "Java Servlets and Applets". Caribou Lake Software Inc., Source: "http://www.cariboulake.com/techinfo/servlet_applet.html"

[8] M. Champione, K. Walwrath, The Java Tutorial. Addison-Wesley, 1998.

[9] Sys-Con Web site, "Developing 3-Tier Database Applications with Java Servlets", Sys-Con Publications, Source: http://www.sys-con.com/java/feature/3-2/3-tier/index.html", 1998.

[10] William Crawford, "Developing Java Servlets". Source: "http://webreview.com/97/10/10/feature/main.html"

[11] JavaSoft Web site, "Remote Method Invocation: Creating Distributed Java-to-Java Applications", Sun Microsystems, Source: "http://developer.java.sun.com/devel…technicalArticles/Mccluskey/rmi.html"

[12] Scott Oaks, Java security. O'Reilly. 1998

[13] Grady Booch, Object Solutions: Managing the Object Oriented Project. Addison-Wesley, October 1995.