

> **Project Summary and Outcomes**

In this project, students will be developing a workable Electronic Help Desk. This database application will be used by the following groups: staff members who have requests for hardware- and software-related computer help, technical support specialists who respond to the help requests, and student programmers. The objective will be to create a user-friendly program that addresses the needs of all three groups.

Student programmers will work with technical specialists to determine the type of data required to be able to respond effectively to help desk questions. They will also develop attractive, efficiently designed data entry forms to capture all of the required data. They may need to test out the entry-form format with potential end users. Students will also develop a second form to enable technical specialists to view current help requests. They will also design an update process by which help requests can be effectively closed out. To write to and read from a database, students will need to use Java Database Connectivity and Open Database Connectivity (JDBC/ODBC) drivers with the appropriate Structured Query Language (SQL) commands to read and update.

The application designed should be robust, as well as easy to use and update. The goal is to create a vehicle for staff members to respond in a timely manner to help requests, for technicians to access these requests, and, most important, for student programmers to learn to interact with a wider community in project development.

> **Student Assessment (see rubric)**

How well do students perform each of the following tasks:

- Create databases with well-defined tables
- Understand and implement the SQL commands
- Use JDBC/ODBC drivers to access a database

- Add new records [rows] to a database table
- Query existing records based on specified criteria
- Update records in a table
- Learn how to work with others on a project
- Work with users to determine needs
- Develop a project timeline
- Work cooperatively to design, implement, and test modules
- Develop prototypical input screens
- Allocate appropriate amounts of time to various tasks
- Complete tasks on time
- Upload and maintain sites on a network or intranet
- Provide comprehensive documentation of a finished system
- Train staff members to use the system

> Prerequisite Knowledge and Skills

Java:

- `java.sql.Connection`
- `java.sql.Statement`
- `java.lang.Class`

Windows Operating System:

- An understanding of ODBC would be helpful, but there are steps included in this project/unit plan that walk new users through the process
- Familiarity with Microsoft Access would be helpful, but there are steps included in this project/unit plan to walk new users through the process

Databases and SQL:

- Understanding of database, table, and field structures
- Basic familiarity with Structured Query Language (SQL) commands

> This Project Targets the Following Subject Areas(s):

| Pre-Java | Java Programming |
|-----------------------------------------------------|------------------------------------------------------------|
| <input type="checkbox"/> Hardware Basics | <input type="checkbox"/> Applet Programming |
| <input checked="" type="checkbox"/> Software Basics | <input type="checkbox"/> Subroutine Programming |
| <input type="checkbox"/> Networks and Servers | <input checked="" type="checkbox"/> Full Scale Programming |
| <input type="checkbox"/> HTML | |
| <input type="checkbox"/> Action Scripting | |
| <input type="checkbox"/> Java Scripting | |

> Project-Framing Questions

| | |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Essential Question | <ul style="list-style-type: none"> How can students build a prototype using staff input that will result in a system that satisfies a school-wide need? |
| Activity Questions | <ul style="list-style-type: none"> Who should have input into the screen-design process? What are the best mechanisms for ensuring the screens meet users' needs? What methods have been drawn up for testing and ensuring user satisfaction? |
| Sample Content Questions | <ul style="list-style-type: none"> What is included in the Date object? How can you access these elements? Which methods of Time have now been replaced by corresponding Calendar methods and why? |

- What methods of the Runnable interface must be implemented?
- Which fields represent data to be acquired from the user and, therefore, present on the request screen?
- What types of queries will be required?
- What SQL commands must be implemented in Java to populate the database and generate the required information?
- How will the test plan be developed and implemented?
- What type of training and documentation will be needed for this project?
- Are there plans for extending this project if demand requires it?

> Targeted Content Standards, Benchmarks, or State Frameworks

Massachusetts Curriculum Frameworks:

Technology/Engineering Engineering Design:

- Engineering design involves practical problem solving, research, development and invention, and requires designing, drawing, building, testing, and redesigning
- Demonstrate methods of representing solutions to a design problem

School-to-Career Competencies:

- Developing team skills and interacting with others/clients
- Developing individual skills and completing entire activities
- Developing individual skills and time management skills

International Society of Technology Education (ISTE):

- **TF-V.D** – Use technology to communicate and collaborate with peers, parents, and the larger community in order to nurture student learning

English Language Arts Curriculum:

- **3.17** – Deliver formal presentations for particular audiences using clear enunciation and appropriate organization, gestures, tone, and vocabulary
- **4.26** – Identify and use correctly new words acquired through study of their different relationships to other words
- **5.30** – Identify, describe, and apply all conventions of standard English
- **20.5** – Use different levels of formality, style, and tone when composing for different audiences
- **21.9** – Revise writing to

improve style, word choice, sentence variety, and subtlety of meaning after rethinking how well questions of purpose, audience, and genre have been addressed

- **22.10** – Use all conventions of standard English when writing and editing

> Materials and Resources Required for Project

| | |
|--------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Equipment | <ul style="list-style-type: none"> • Projector for presenting prototype and final project |
| Consumable Supplies | <ul style="list-style-type: none"> • Diskettes • Printed output for prototype |
| Textbooks/Lesson Guides | <ul style="list-style-type: none"> • <i>Core Java Volume II – Advanced Features</i> by Cay Horstmann and Gary Cornell. Sun Microsystems Press, 2000. • <i>Core Web Programming</i> by Marty Hall and Larry Brown. Sun Microsystems Press, 2000. |
| Technology | <ul style="list-style-type: none"> • Database and SQL commands • Java JDBC/ODBC syntax and implementation |
| Internet Resources | <ul style="list-style-type: none"> • http://java.sun.com/developer/onlineTraining/Database/index.html • http://java.sun.com/products/jdbc/learning/tutorial/index.html |

> Activity/Lesson Plan Outline

Overview: This is a database project. As such, students will begin with an Microsoft Access [or similar] database with two tables: one for user requests and another for staff names and passwords. In the addendum portion of this document, you will find steps for creating an Access database. To code to a database from Java, students will also need to establish a JDBC/ODBC connection, which allows a program to connect to a variety of datasets. You will find instructions in the addendum for establishing a JDBC/ODBC connection to an Access database. Once students have the tables in place, they will design and develop data entry modules to allow staff members to place and review requests. As this is an interactive project, student programmers will also consult with end users to determine their needs. After assessing and translating these needs into appropriate Access tables and input screens, students will begin coding. Assuming they have some familiarity with SQL commands and java Connection and Statement classes, students will design input screens translate user requests into database input and retrieval. See the previously listed Internet and text resources for help with this portion of the project.

Purpose: To streamline the process through which staff members request and receive assistance for computer problems. To enhance and apply knowledge of database activities (creating and using tables, queries, reports). To develop team and time management skills by working together and interacting with clients.

Assessment: Student projects are assessed based on how well student programmers handle the following various tasks:

- Screen design – How functional are the screens in terms of fulfilling the following tasks:
- Entering requests
- Reviewing requests (staff)
- Retrieving data (technicians)
- Updating request data

- SQL implementation – How successful is the project in terms of accomplishing the following:
 - Creating databases and tables
 - Connecting to databases using JDBC/ODBC drivers
 - Executing database queries using SQL
 - Closing connections upon completion
- Documentation – How thorough is the documentation provided:
 - Developing project prototype with screen design and database structures
 - Finishing system documentation with optional training materials
 - Documenting classes

Step 1 – Create an Access database with two tables, one for user requests and the other for staff names, passwords, and priority levels. Individual staff members should always be able to review their own requests and the status of those requests. Technicians should be able to both review all requests and update the status of requests as they change.

Tips, Hints, and Tricks: When defining the requests table, spend some time with technicians to determine all of the things that they need to know to respond to a help request. Use the priority field to indicate individuals eligible to retrieve and update requests. Make all fields text fields for ease of access.

Step 2 – User request screen. Create a data entry screen and code module for request input. The screen should reflect all fields deemed necessary for the project to succeed. See Sample Request Service Screen under Addendum 1, [Screen 1](#).

Tips, Hints, and Tricks: This step will require some back and forth communication with technical support professionals to determine that all of the needed fields have been included.

Step 3 – User query screen. Create a query screen and for staff access, including username and password verification. Once a username and password have been verified, a user should be able to review all his/her requests and their statuses. See

Sample Request Service Screen under Addendum 1, [Screen 2](#).

Tips, Hints, and Tricks: Allow the user to review previously submitted requests without changing them.

Step 4 – Technician request retrieval screen. This should be designed to allow a technician to view requests by date, staff member, or problem. Ideally, this module will become a full-fledged report generation screen with options for providing a wide range of reports, both on screen and in print. See Sample Request Service Screen under Addendum 1, [Screen 3](#).

Step 5 – Request entry screen module. This code will generate the request screen that will enable a user to submit a new technical help request. (See Sample Request Service Screen under Addendum 1, [Screen 1](#)). A listing of the code is provided in Addendum 2, [Sample 1](#).

Step 6 – User query screen module. Create the code that executes when a user requests a status update on previously submitted requests. A listing is provided in Addendum 2, [Sample 2](#).

Step 7 – Technician request retrieval screen module. Create the code to build the technician request retrieval screen (Illustrated in Addendum 1, Screen 2). Sample code is provided in Addendum 3, Sample 3.code that executes when a staff member completes the request screen and presses the submit button. Sample code to access the JDBC/ODBC database referred to by the System DSN HelpDesk is provided in Sample 2. The database contains a requests table into which the values from this screen will be added as a new record within that table.

Step 8 – Process user request. Create code to process a request made by a user. This code will access the database, specifically, the requests table, create a new record and populate that record with the data provided by the user. A sample is given in Addendum 3, [Process Sample 1](#).

Step 9 – Process user query request. Code the module to retrieve request data. Sample code is provided in Addendum 3, [Process Sample 2](#).

Step 10 – Process technician query request. Code the module to respond to a technician’s request for data. Sample code is provided in Addendum 3, **Process Sample 3**.

Step 11 – Students design test plan. This should be a comprehensive set of data which will be run against the program.

Step 12 – Test and revise. Students should begin testing, revising the code as needed.

> Pacing/Timeline

- Step #1: 3 -5 days. Students learn how to use Microsoft Access or a similarly suitable database. Steps #1-4: 1 day. Students meet with technical professionals to determine the data needed to response to a help request.
- Steps #2-4: 1 week. Students design initial screen prototypes.
- Steps #2-4: 1 day. Students review prototypes with support professionals to determine modifications.
- Steps #2-4: 3 days. Students make needed modification and again review with technical support.
- Steps #5 –10: 2 weeks. Students begin coding modules – add records, query, modify.
- Step #11: 2 days. Students design comprehensive test plan.
- Step #12: 4 days. Students begin testing and making needed revisions.

> Teacher Reflection (For example, what worked well in this project? What would you change if you were to teach it again?)

- How did students handle the coding, specifically the database piece?
- Were students able to interact with staff members?
- What roadblocks did students find in implementing this project?
- What ideas did students have for enhancing or broadening the project?
- Was the timeline realistic – did students need more time on any activities?

> Help Desk Project - Rubric

| Attribute | Needs Improvement | Proficient | Advanced | Maximum Pts. |
|--------------------------------------------------------------------|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|--------------|
| Design Input and Query Screens | | | | |
| <i>Designs Request Screen</i> | Uses too few, mainly text, fields | Uses a number of fields addressing a wide array of questions. Uses combo boxes, radio buttons, and text area fields. | Includes data validation. Also includes username and password, possibly even supplying a password-hint option for forgotten passwords. | 8 |
| <i>Designs Query Screen for Staff to Review Requests Submitted</i> | Allows review, but without filtering | Includes review by date range, type of request, staff member or other relevant criteria. | Builds in open-ended query options for staff members. | 8 |
| <i>Designs Retrieval Screen</i> | Allows technicians to view requests | Allows technicians to view requests by date, staff member, type of problem, or any combination thereof | Builds in open-ended query options for technicians. | 8 |
| <i>Designs Update Screen to View Change Status of Requests</i> | Designs screen with minimal functionality | Provides a complete set of options for status. Adds comment areas. | Builds in notification to staff when requests have been resolved. May include flags for follow-ups. | 8 |

Implement SQL

| | | | | |
|------------------------------------|----------------------------------------|----------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|----|
| <i>Creates Database and Tables</i> | Creates insufficient tables and fields | Establishes database with appropriate tables. Makes ODBC connection (if using Access). | Includes password tables in database | 5 |
| <i>Establishes Connection</i> | Unable to access | Correctly uses ODBC, JDBC/ODBC drivers | Connects with error checking | 8 |
| <i>Executes Queries</i> | Cannot execute queries | Runs successful queries and produces desired result set | Produces a nicely formatted report with the option to save and print. May produce an entire module of query options. | 12 |
| <i>Executes Updates</i> | Develops no method | Runs update, which adds to or modifies table | Produces a module that confirms addition and that perhaps has the option to output an updated table or some part thereof | 12 |

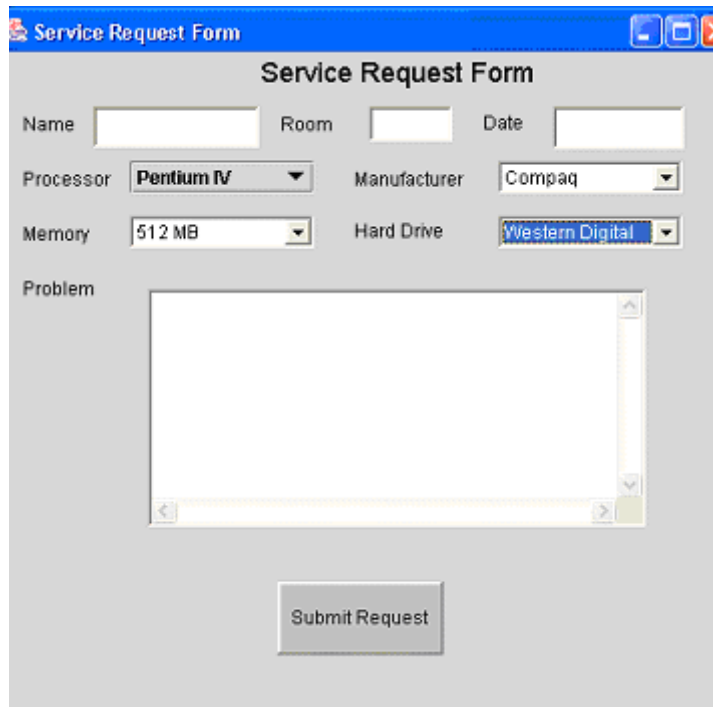
| | | | | |
|-------------------------------------------|-------------------------------------------------------|---------------------------------------------------------------------|-----------------------------------------------------------------------------|----|
| <i>Supplies an Option to Add New User</i> | Creates a way to add a user and password to the table | Creates a way to request data, which is then verified by technician | Creates a way to add a user and then send out a subsequent welcome greeting | 12 |
|-------------------------------------------|-------------------------------------------------------|---------------------------------------------------------------------|-----------------------------------------------------------------------------|----|

Provides Documentation

| | | | | |
|--------------------------------------------|--------------------------------------------|------------------------------------------------------------------------------|---------------------------------------|---|
| <i>Closes Connection</i> | Does not close connection | Successfully closes connection each time it is opened | Builds in error checking | 5 |
| <i>Documents Prototype</i> | Distributes screen shots | Distributes screen shots and database structure, providing descriptions | Illustrates screen and report options | 3 |
| <i>Documents Finished Product</i> | Distributes screen shots and notes for use | Provides documentation on entire system, including all screens and reporting | Also provides live training | 8 |
| <i>Provides Properly Commented Classes</i> | Supplies minimal comments | Adds Javadoc | Also provides live training | 3 |

> Addendum 1: Sample Screens

Screen 1: Service Request Form

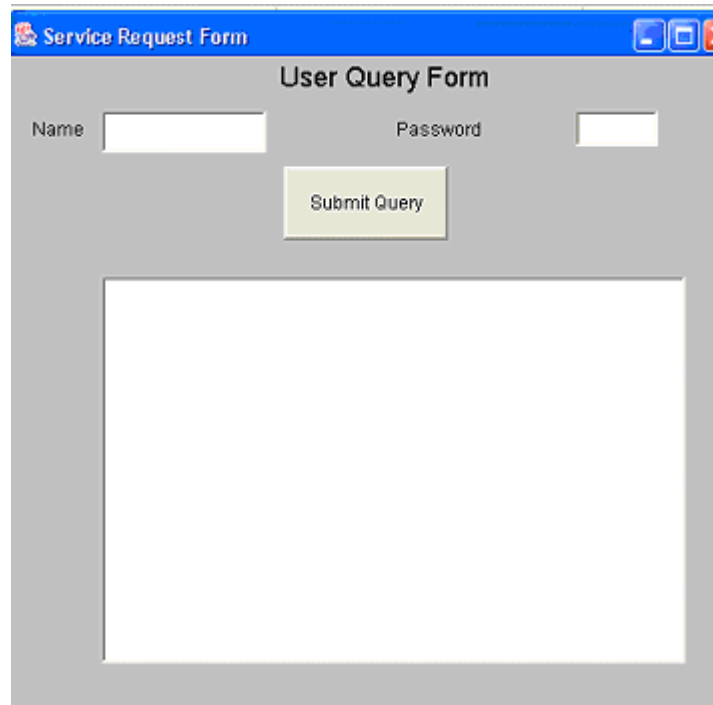


The screenshot shows a window titled "Service Request Form" with a blue title bar. The form itself has a grey background and is titled "Service Request Form". It contains several input fields and dropdown menus:

- Name:
- Room:
- Date:
- Processor:
- Manufacturer:
- Memory:
- Hard Drive:
- Problem:

At the bottom of the form is a "Submit Request" button.

Screen 2: User Query Form

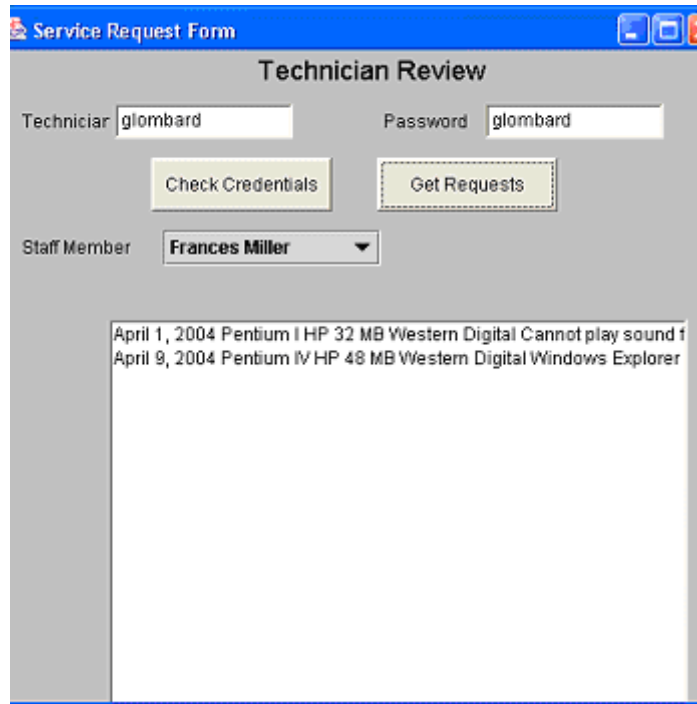


The screenshot shows a window titled "Service Request Form" with a blue title bar. The form itself has a grey background and is titled "User Query Form". It contains two input fields and a button:

- Name:
- Password:
- Submit Query:

Below the button is a large empty rectangular area, likely intended for a query or response.

Screen 3: Request Retrieval Form



The screenshot shows a Java-based application window titled "Service Request Form" with a subtitle "Technician Review". The window contains the following elements:

- Technician:
- Password:
- Buttons: "Check Credentials" and "Get Requests" (the latter is highlighted with a dashed border).
- Staff Member: A dropdown menu currently showing "Frances Miller".
- Request List: A scrollable area containing two entries:
 - April 1, 2004 Pentium I HP 32 MB Western Digital Cannot play sound f
 - April 9, 2004 Pentium IV HP 48 MB Western Digital Windows Explorer

> Addendum 2: Sample Code for Building Screens

Sample 1

RequestScreen.java – builds a screen for users to input requests for technical assistance

```
import java.awt.*;
import java.awt.event.*;

/*
   RequestScreen.java - Simple input screen
   asking for user name and definition of the
   problem
*/

class RequestScreen extends Frame
{
    RequestScreen()
    {
// Initialize and instantiate declared controls

        //{{INIT_CONTROLS
        setLayout(null);
        setBackground(Color.lightGray);
        setSize(479,435);
        setVisible(false);
        nameLabel.setText("Name");
        add(nameLabel);
        nameLabel.setBounds(12,36,36,24);
        add(name);
        name.setBounds(60,36,108,27);
        roomLabel.setText("Room");
        add(roomLabel);
        roomLabel.setBounds(180,36,36,24);
        add(room);
        room.setBounds(240,36,54,23);
        dateLabel.setText("Date");
        add(dateLabel);
        dateLabel.setBounds(312,36,40,22);
        add(date);
        date.setBounds(360,36,85,28);
        titleLabel.setText("Service Request Form");
        titleLabel.setAlignment(java.awt.Label.CENTER);
        add(titleLabel);
        titleLabel.setFont(new Font("Dialog", Font.PLAIN, 18));
        titleLabel.setBounds(156,0,204,28);
```

```

processorLabel.setText("Processor");
add(processorLabel);
processorLabel.setBounds(12,72,60,24);
add(cpu);
cpu.setBounds(84,72,120,21);
manufacturerLabel.setText("Manufacturer");
add(manufacturerLabel);
manufacturerLabel.setBounds(228,72,84,24);
add(manufacturer);
manufacturer.setBounds(324,72,120,21);
memoryLabel.setText("Memory");
add(memoryLabel);
memoryLabel.setBounds(12,108,48,24);
add(memory);
memory.setBounds(84,108,120,21);
driveLabel.setText("Hard Drive");
add(driveLabel);
driveLabel.setBounds(228,108,60,20);
add(hardDrive);
hardDrive.setBounds(324,108,120,21);
problemLabel.setText("Problem");
add(problemLabel);
problemLabel.setBounds(12,144,52,23);
add(problem);
problem.setBounds(96,156,324,156);
requestButton.setLabel("Submit Request");
add(requestButton);
requestButton.setBackground(java.awt.Color.lightGray);
requestButton.setBounds(180,348,108,48);
setTitle("Service Request Form");

// Populate processor Choice
cpu.addItem("Pentium IV");
cpu.addItem("Pentium III");
cpu.addItem("Pentium II");
cpu.addItem("Pentium I");
cpu.addItem("486");
cpu.addItem("386");
cpu.addItem("Other");

// Populate manufacturer choice
manufacturer.addItem("Compaq");
manufacturer.addItem("HP");
manufacturer.addItem("HiQ");
manufacturer.addItem("IBM");
manufacturer.addItem("Apple");
manufacturer.addItem("Other");

// Populate memory choice
memory.addItem("16 MB");
memory.addItem("32 MB");

```

```

        memory.addItem("48 MB");
        memory.addItem("128 MB");
        memory.addItem("256 MB");
        memory.addItem("512 MB");
        memory.addItem("1024 MB");
        memory.addItem("Other");

        // Populate drive choice
        hardDrive.addItem("Unknown");
        hardDrive.addItem("Fujitsu");
        hardDrive.addItem("Western Digital");
        hardDrive.addItem("Maxtor");
hardDrive.addItem("Other");

        addWindowListener(new WindowAdapter(){

            public void WindowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });

        //}}
    }

    public RequestScreen(String title)
    {
        this();
        setTitle(title);
    }

    /**
    * Shows or hides the component depending on the boolean
flag b.
    * @param b if true, show the component; otherwise,
hide the
    * component.
    * @see java.awt.Component#isVisible
    */
    public void setVisible(boolean b)
    {
        if(b)
        {
            setLocation(50, 50);
        }
        super.setVisible(b);
    }

    public void addNotify()

```

```

    {
        // Record the size of the window prior to calling
parent's
        // addNotify.
        Dimension d = getSize();

        super.addNotify();

        if (fComponentsAdjusted)
            return;

        // Adjust components according to the insets
        setSize(getInsets().left + getInsets().right +d.width,
                getInsets().top+getInsets().bottom+d.height);
        Component components[] = getComponents();
        for (int i = 0; i < components.length; i++)
        {
            Point p = components[i].getLocation();
                p.translate(getInsets().left, getInsets().top);
                components[i].setLocation(p);
            }
            fComponentsAdjusted = true;
        }

        // Used for addNotify check.
        boolean fComponentsAdjusted = false;

        //{{DECLARE_CONTROLS
        Label nameLabel = new Label();
        TextField name = new TextField();
        Label roomLabel = new Label();
        TextField room = new TextField();
        Label dateLabel = new Label();
        TextField date = new TextField();
        Label titleLabel = new Label();
        Label processorLabel = new Label();
        Choice cpu = new Choice();
        Label manufacturerLabel = new Label();
        Choice manufacturer = new Choice();
        Label memoryLabel = new Label();
        Choice memory = new Choice();
        Label driveLabel = new Label();
        Choice hardDrive = new Choice();
        Label problemLabel = new Label();
        TextArea problem = new TextArea();
        Button requestButton = new Button();
        //}}

```

```

class SymWindow extends java.awt.event.WindowAdapter
{
    public void windowClosing(java.awt.event.WindowEvent
event)
    {
        Object object = event.getSource();
        if (object == RequestScreen.this)
            RequestScreen_WindowClosing(event);
    }
}

void RequestScreen_WindowClosing(java.awt.event.WindowEvent
event)
{
    // to do: code goes here.

    RequestScreen_WindowClosing_Interaction1(event);
}

void RequestScreen_WindowClosing_Interaction1(
java.awt.event.WindowEvent event)
{
    try {
        // QuitDialog Create and show as modal
        (new QuitDialog(this, true)).setVisible(true);
    } catch (Exception e) {
    }
}
}

```

Sample 2

UserQuery.java– builds a screen for a user to query the database to view the previously submitted requests

```

import java.awt.*;
import java.awt.event.*;

/*
    UserQuery.java - Simple input screen
    allows a user to view requests he/she
    has made
*/

public class UserQuery extends Frame
{
    public UserQuery()
    {
        // Initialize and instantiate declared controls

```

```

        //{{INIT_CONTROLS
        setLayout(null);
        setBackground(java.awt.Color.lightGray);
        setSize(479,435);
        setVisible(false);
        nameLabel.setText("Name");
        add(nameLabel);
        nameLabel.setBounds(12,36,36,24);
        add(name);
        name.setBounds(60,36,108,27);
        passwordLabel.setText("Password");
        add(passwordLabel);
        passwordLabel.setBounds(252,36,60,24);
        add(password);
        password.setBounds(372,36,54,23);
        titleLabel.setText("User Query Form");
        titleLabel.setAlignment(java.awt.Label.CENTER);
        add(titleLabel);
        titleLabel.setFont(new Font("Dialog", Font.PLAIN, 18));
        titleLabel.setBounds(144,0,204,28);
        submitQueryBtn.setLabel("Submit Query");
        add(submitQueryBtn);
        submitQueryBtn.setBounds(180,72,108,48);
        add(listRequests);
        listRequests.setBounds(60,144,384,252);
        setTitle("User Query Form");
        //}}
    }

    public UserQuery(String title)
    {
        this();
        setTitle(title);
    }

    /**
     * Shows or hides the component depending on the boolean flag b.
     * @param b if true, show the component; otherwise, hide the
component.
     * @see java.awt.Component#isVisible
     */
    public void setVisible(boolean b)
    {
        if(b)
        {
            setLocation(50, 50);
        }
        super.setVisible(b);
    }

```

```

public void addNotify()
{
    // Record the size of the window prior to calling
parent's // addNotify.
    Dimension d = getSize();

    super.addNotify();

    if (fComponentsAdjusted)
        return;

    // Adjust components according to the insets
    setSize(getInsets().left+ getInsets().right+d.width,
            getInsets().top getInsets().bottom d.height);
    Component components[] = getComponents();
    for (int i = 0; i < components.length; i++)
    {
        Point p = components[i].getLocation();
        p.translate(getInsets().left, getInsets().top);
        components[i].setLocation(p);
    }
    fComponentsAdjusted = true;
}

// Used for addNotify check.
boolean fComponentsAdjusted = false;

//{{{DECLARE_CONTROLS
Label nameLabel = new Label();
TextField name = new TextField();
Label passwordLabel = new Label();
TextField password = new TextField();
Label titleLabel = new Label();
Button submitQueryBtn = new Button();
List listRequests = new List(0);
//}}}

class SymWindow extends java.awt.event.WindowAdapter
{
    public void windowClosing(java.awt.event.WindowEvent
event)
    {
        Object object = event.getSource();
        if (object == UserQuery.this)
            UserQuery_WindowClosing(event);
    }
}

```

```

void UserQuery_WindowClosing(java.awt.event.WindowEvent event)
{
    // to do: code goes here.

    UserQuery_WindowClosing_Interaction1(event);
}

void UserQuery_WindowClosing_Interaction1(
java.awt.event.WindowEvent
    event)
{
    try {
        // QuitDialog Create and show as modal
        (new QuitDialog(this, true)).setVisible(true);
    } catch (Exception e) {
    }
}
}

```

Sample 3

TechnicianScreen.java— builds a screen enabling a technician to request data on service requests.

```

import java.awt.*;
import javax.swing.*;

/*
 *
 * TechnicianScreen.java
 * author:  Ginny Lombard
 * Version: Version 1.0
 * - Simple input screen
 * asking for technician name and password. Assuming
 * technician is found and has the appropriate priority level
 * a list of users will appear and a user can be selected. Then
 * a set of requests from that user will appear.
 * Version 2.0 will include selecting by date
 */

public class TechnicianScreen extends Frame
{
    public TechnicianScreen()
    {
        // Initialize and instantiate declared controls
    }
}

```

```

    //{{INIT_CONTROLS
    setLayout(null);
    setBackground(java.awt.Color.lightGray);
    setSize(479,435);
    setVisible(false);
    nameLabel.setText("Technician");
    add(nameLabel);
    nameLabel.setBounds(12,36,60,24);
    add(name);
    name.setBounds(76,36,118,23);
    passwordLabel.setText("Password");
    add(passwordLabel);
    passwordLabel.setBounds(252,36,60,24);
    add(password);
    password.setBounds(322,36,118,23);
    titleLabel.setText("Technician Review");
    titleLabel.setAlignment(java.awt.Label.CENTER);
    add(titleLabel);
    titleLabel.setFont(new Font("Dialog", Font.PLAIN, 18));
    titleLabel.setBounds(144,0,204,28);
    submitQueryBtn.setLabel("Check Credentials");
    add(submitQueryBtn);
    submitQueryBtn.setBounds(100,72,120,36);
    getRequestBtn.setLabel("Get Requests");
    add(getRequestBtn);
    getRequestBtn.setBounds(250,72,120,36);
    getRequestBtn.setVisible(false);
    add(listRequests);
    listRequests.setBounds(72,180,384,452);
    staffLabel.setText("Staff Member");
    add(staffLabel);
    staffLabel.setBounds(12,120,84,24);
    add(staffList);
    staffList.setBounds(108,120,144,24);
    badUserPassword.setText("Bad User Name / Password");
    add(badUserPassword);
    badUserPassword.setBounds(324,72,147,41);
    badUserPassword.setVisible(false);
    setTitle("Service Request Form");
    //}}
}

public TechnicianScreen(String title)
{
    this();
    setTitle(title);
}

/**
 * Shows or hides the component depending on the boolean flag b.

```

```

    * @param b  if true, show the component; otherwise, hide the
component.
    * @see java.awt.Component#isVisible
    */
    public void setVisible(boolean b)
    {
        if(b)
        {
            setLocation(50, 50);
        }
        super.setVisible(b);
    }

    public void addNotify()
    {
        // Record the size of the window prior to calling
parent's
        // addNotify.
        Dimension d = getSize();

        super.addNotify();

        if (fComponentsAdjusted)
            return;

        // Adjust components according to the insets
        setSize(getInsets().left + getInsets().right+d.width,
            getInsets().top+getInsets().bottom + d.height);
        Component components[] = getComponents();
        for (int i = 0; i < components.length; i++)
        {
            Point p = components[i].getLocation();
            p.translate(getInsets().left, getInsets().top);
            components[i].setLocation(p);
        }
        fComponentsAdjusted = true;
    }

    // Used for addNotify check.
    boolean fComponentsAdjusted = false;

    //{{DECLARE_CONTROLS
    Label nameLabel = new Label();
    TextField name = new TextField();
    Label passwordLabel = new Label();
    TextField password = new TextField();
    Label titleLabel = new Label();
    Button submitQueryBtn = new Button();
    List listRequests = new List(40);

```

```

Label staffLabel = new Label();
JComboBox staffList = new JComboBox();
TextField badUserPassword = new TextField();
Button getRequestBtn = new Button();
//}}

class SymWindow extends java.awt.event.WindowAdapter
{
    public void windowClosing(java.awt.event.WindowEvent
event)
    {
        Object object = event.getSource();
        if (object == TechnicianScreen.this)
            Technician_WindowClosing(event);
    }
}

void Technician_WindowClosing(java.awt.event.WindowEvent
event)
{
    // to do: code goes here.

    Technician_WindowClosing_Interaction1(event);
}

void
Technician_WindowClosing_Interaction1(java.awt.event.WindowEvent
event)
{
    try {
        // QuitDialog Create and show as modal
        (new QuitDialog(this, true)).setVisible(true);
    } catch (Exception e) {
    }
}
}
}

```

> Addendum 3: Sample Screen Processing Code

Process Sample 1

ProcessRequest.java- processes the request input through the service request form.

```
import java.awt.event.*;
```

```
import java.net.*;
import java.sql.*;

/**
 * ProcessRequest.java. This code will receive input from a user
 concerning
 * a technical problem needing to be resolved. The data will be
 stored in a
 * requests table in the Help Desk database
 *
 * @author
 * @version 1.0
 */
public class ProcessRequest implements ActionListener
{
    // instance variables
    RequestScreen theRequest;
    Connection con;
    Statement stmt;

    /**
     * Constructor for objects of class ProcessRequest
     */
    public ProcessRequest()
    {
        // instance variables initialized
        theRequest = new RequestScreen();
        theRequest.requestButton.addActionListener(this);
    }

    public static void main(String[] args)
    {
        ProcessRequest pRequest = new ProcessRequest();
        pRequest.theRequest.setVisible(true);
    }

    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource() == theRequest.requestButton)
        {
            System.out.println("Action caught");
            getRequests();
        }
    }

    public void getRequests()
    {
        String name, room, date;
        int index;
        String cpu, manuf, mem, drive;
    }
}
```

```

String prob;
String url = "jdbc:odbc:HelpDesk";
String first, last;
boolean found = false;

name = theRequest.name.getText().trim();
room = theRequest.room.getText().trim();
date = theRequest.date.getText().trim();
index = theRequest.cpu.getSelectedIndex();
cpu = theRequest.cpu.getItem(index);
index = theRequest.manufacturer.getSelectedIndex();
manuf = theRequest.manufacturer.getItem(index);
index = theRequest.memory.getSelectedIndex();
mem = theRequest.memory.getItem(index);
index = theRequest.hardDrive.getSelectedIndex();
drive = theRequest.hardDrive.getItem(index);
prob = theRequest.problem.getText();

Connection con;
String updateString;
Statement stmt;

System.out.println("Creating update string");
updateString = "INSERT INTO Requests VALUES('";
    updateString += name;
    updateString += " , ";
    updateString += room;
    updateString += " , ";
    updateString += date;
    updateString += " , ";
    updateString += cpu;
    updateString += " , ";
    updateString += manuf;
    updateString += " , ";
    updateString += mem;
    updateString += " , ";
    updateString += drive;
    updateString += " , ";
    updateString += " ";
    updateString += " , ";
    updateString += " ";
    updateString += " , ";
    updateString += prob;
    updateString += "')";
System.out.println(updateString);

try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

} catch (java.lang.ClassNotFoundException e) {
    System.err.print("ClassNotFoundException: ");

```

```

        System.err.println(e.getMessage());
    }

    try {
        con = DriverManager.getConnection(url,
                                         "", "");

        stmt = con.createStatement();
        stmt.executeUpdate(updateString);
        stmt.close();
        con.close();

    } catch(SQLException ex) {
        System.err.println(updateString);
        System.err.println("SQLException in addAssignment: " +
                           ex.getMessage());
    }

    theRequest.name.setText("");
    theRequest.room.setText("");
    theRequest.date.setText("");
    theRequest.problem.setText("");

}

}

```

Process Sample 2

ProcessQuery.java- processes the user request for status update input through the user query form.

```

import java.awt.event.*;
import java.net.*;
import java.sql.*;

/**
 * this class will query the database to produce all requests for
 * technical assistance from a specific user
 *
 * @author
 * @version 1.0
 */
public class ProcessQuery implements ActionListener
{
    // instance variables
    UserQuery theQuery;
    Connection con;
    Statement stmt, stmt2;

    /**

```

```

    * Constructor for objects of class ProcessQuery
    */
public ProcessQuery()
{
    theQuery = new UserQuery();
    theQuery.submitQueryBtn.addActionListener(this);
}

public static void main(String[] args)
{
    ProcessQuery pQuery = new ProcessQuery();
    pQuery.theQuery.setVisible(true);
}

public void actionPerformed(ActionEvent e)
{
    if (e.getSource() == theQuery.submitQueryBtn)
    {
        getRequests();
    }
}

public void getRequests()
{
    // Verify that there are requests in the database
    // from the user before proceeding to list them
    String uname, pword;
    String url = "jdbc:odbc:HelpDesk";
    String first, last;
    boolean found = false;

    uname = theQuery.name.getText().trim();
    pword = theQuery.password.getText().trim();
    String queryString = "";
    queryString = "SELECT * FROM passwords WHERE username = '" +
uname + "'
                ";
    queryString += "and password = '" + pword + "'";
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    } catch (java.lang.ClassNotFoundException e) {
        System.err.print("ClassNotFoundException: ");
        System.err.println(e.getMessage());
    }

    try {
        con = DriverManager.getConnection(url,
                "", "");

        stmt = con.createStatement();

```

```

        ResultSet rs = stmt.executeQuery(queryString);
        int recordCount = 0;
        while (rs.next())
        {
            recordCount++;
        }
        found = (recordCount > 0);

        rs.close();

        stmt.close();
        con.close();

    } catch(SQLException ex) {
        System.err.println(queryString);
        System.err.println("SQLException in getRequests: " +
            ex.getMessage());
    }

    if (found)
    {
        populateQueryList(uname);
    }
    else
    {
        System.out.println("Username not found");
    }
}

public void populateQueryList(String uname)
{
    String url = "jdbc:odbc:HelpDesk";

    String queryString = "SELECT * FROM requests WHERE
name = '" +
                                uname + "' ";
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    } catch(java.lang.ClassNotFoundException e) {
        System.err.print("ClassNotFoundException: ");
        System.err.println(e.getMessage());
    }

    try {
        con = DriverManager.getConnection(url,
            "", "");

```

```

        stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(queryString);
        while (rs.next())
        {
            String line = "";
            line += (String)rs.getObject("date") + " ";
            line += (String)rs.getObject("cpu") + " ";
            line += (String)rs.getObject("manufacturer") + " ";
            line += (String)rs.getObject("hardDrive") + " ";
            line += (String)rs.getObject("problem") + " ";

            theQuery.listRequests.add(line);
        }

        rs.close();

        stmt.close();
        con.close();

    } catch(SQLException ex) {
        System.err.println(queryString);
        System.err.println("SQLException in getRequests Module:
" +
            ex.getMessage());
    }

        theQuery.name.setText("");
        theQuery.password.setText("");
    }
}

```

Process Sample 3

ProcessTechnician.java- processes the technician's request for data input through the user query form.

```

import java.awt.Window;
import java.awt.event.*;
import java.net.*;
import java.sql.*;
import java.util.*;
import javax.swing.*;

/**
 * ProcessTechnician - this module will process the requests of a
 technician
 * to view all requests by a user.
 *
 * @author

```

```

* @version 1.0
*/
public class ProcessTechnician implements ActionListener
{
    // instance variables - replace the example below with your own
    TechnicianScreen theRequest;
    Connection con;
    Statement stmt;
    Map allUsers;
    boolean validUser;

    /**
     * Constructor for objects of class ProcessTechnician
     */
    public ProcessTechnician()
    {
        // initialise instance variables
        theRequest = new TechnicianScreen();
        theRequest.submitQueryBtn.addActionListener(this);
        theRequest.getRequestBtn.addActionListener(this);
        allUsers = new HashMap();

        validUser = false;
    }

    public static void main(String[] args)
    {
        ProcessTechnician pRequest = new ProcessTechnician();

        pRequest.theRequest.setVisible(true);
    }

    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource() == theRequest.submitQueryBtn)
        {
            verifyUser();
        }
        else if (e.getSource() == theRequest.getRequestBtn)
        {
            populateQueryList();
        }
    }

    public void verifyUser()
    {
        String sName, sPassword;
        String url = "jdbc:odbc:HelpDesk";

```

```

boolean found = false;
int columns = 0;
ResultSet rs;
ResultSetMetaData rsmd;

sName = theRequest.name.getText().trim();
sPassword = theRequest.password.getText().trim();

Connection con = null;
String queryString;
Statement stmt = null;
int priority = 1;

queryString = "SELECT * FROM Passwords WHERE username = ";
queryString += sName;
queryString += " and password = ";
queryString += sPassword;
queryString += " ";

try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

} catch (java.lang.ClassNotFoundException e) {
    System.err.println("ClassNotFoundException: ");
    System.err.println(e.getMessage());
}

try {
    con = DriverManager.getConnection(url, "", "");

    stmt = con.createStatement();
    rs = stmt.executeQuery(queryString);
    rsmd = rs.getMetaData();
    columns = rsmd.getColumnCount();
    if (columns > 0)
    {
        rs.next();
        priority =
Integer.parseInt((String)rs.getObject("priority"));
    }

} catch (SQLException ex) {
    System.err.println(queryString);
    System.err.println("SQLException in verify User: " +
        ex.getMessage());
}

try
{
    stmt.close();
    con.close();

```

```

    }
    catch (SQLException ex)
    {
        System.err.println(queryString);
        System.err.println("SQLException in Closing Connection:
" +
            ex.getMessage());

    }
    if (columns == 0) // User not found
    {
        theRequest.badUserPassword.setText("User Not Found");
        theRequest.badUserPassword.setVisible(true);
    }
    else if (priority < 1) // Need priority 1 to access data
    {
        theRequest.badUserPassword.setText("This task not
accessible to
            this user");
        theRequest.badUserPassword.setVisible(true);
    }
    else
    {
        listStaff();
    }
}

public void populateQueryList()
{
    int index = theRequest.staffList.getSelectedIndex();
    String userName;
    String staffMember =
        (String)theRequest.staffList.getItemAt(index);
    ResultSet rs;
    ResultSetMetaData rsmd;

    String url = "jdbc:odbc:HelpDesk";

    String queryString = "SELECT * FROM requests ";
    if (!(staffMember.equals("All")))
    {
        // find username associated with selected name
        userName = (String)allUsers.get(staffMember);
        queryString += " WHERE name = '" + userName + "'
";
    }
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    } catch (java.lang.ClassNotFoundException e) {

```

```

        System.err.print("ClassNotFoundException: ");
        System.err.println(e.getMessage());
    }

    try {
        con = DriverManager.getConnection(url,
                                         "", "");

        stmt = con.createStatement();
        rs = stmt.executeQuery(queryString);
        rsmd = rs.getMetaData();
        int columnCount = rsmd.getColumnCount();
        int recordCount = 0;
        while (rs.next())
        {
            String line = "";
            line += (String)rs.getObject("date") + " ";
            line += (String)rs.getObject("cpu") + " ";
            line += (String)rs.getObject("manufacturer") + " ";
            line += (String)rs.getObject("memory") + " ";
            line += (String)rs.getObject("hardDrive") + " ";
            line += (String)rs.getObject("problem") + " ";

            theRequest.listRequests.add(line);
        }

        rs.close();

        stmt.close();
        con.close();

    } catch(SQLException ex) {
        System.err.println(queryString);
        System.err.println("SQLException in addAssignment: " +
                           ex.getMessage());
    }
}

void listStaff()
{
    String url = "jdbc:odbc:HelpDesk";
    boolean found = false;
    ResultSet rs;
    Connection con;
    String queryString;
    Statement stmt;
    String line;
    String user;

    queryString = "SELECT * FROM Passwords ";
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

```

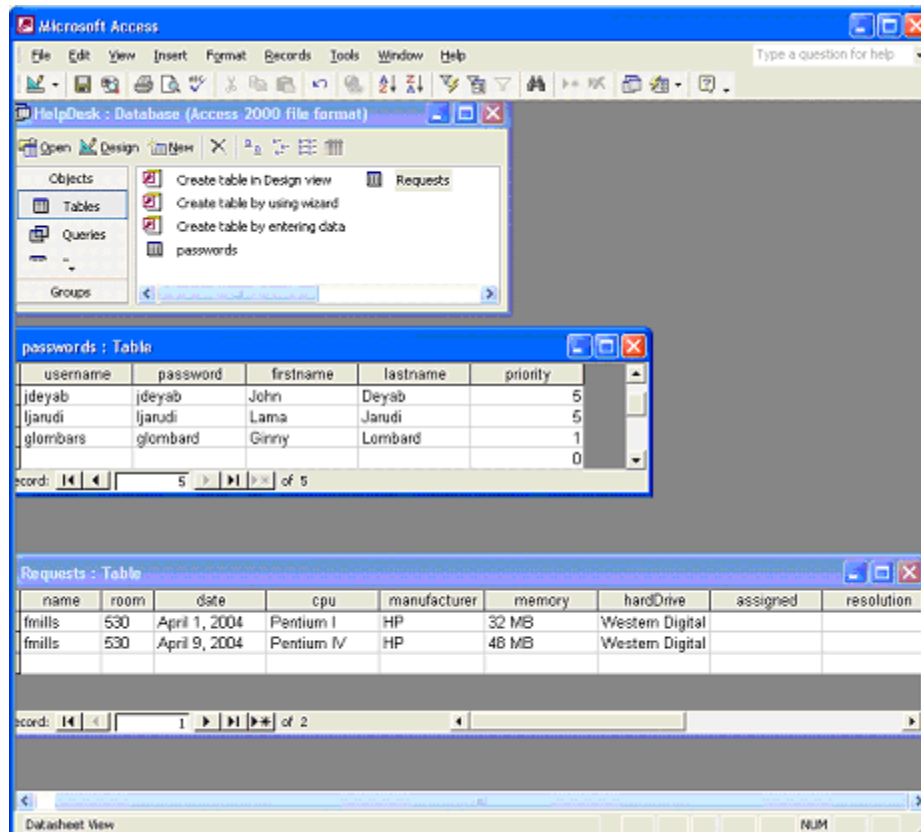
```
    } catch(java.lang.ClassNotFoundException e) {
        System.err.print("ClassNotFoundException: ");
        System.err.println(e.getMessage());
    }
}

try {
    con = DriverManager.getConnection(url,
                                     "", "");

    stmt = con.createStatement();
    rs = stmt.executeQuery(queryString);
    theRequest.staffList.addItem("All");
    while (rs.next())
    {
        user = (String)rs.getObject("username");
        line = rs.getObject("firstname") + " " +
              rs.getObject("lastname");
        theRequest.staffList.addItem(line);
        allUsers.put(line,user);
    }
    theRequest.getRequestBtn.setVisible(true);
    stmt.close();
    con.close();

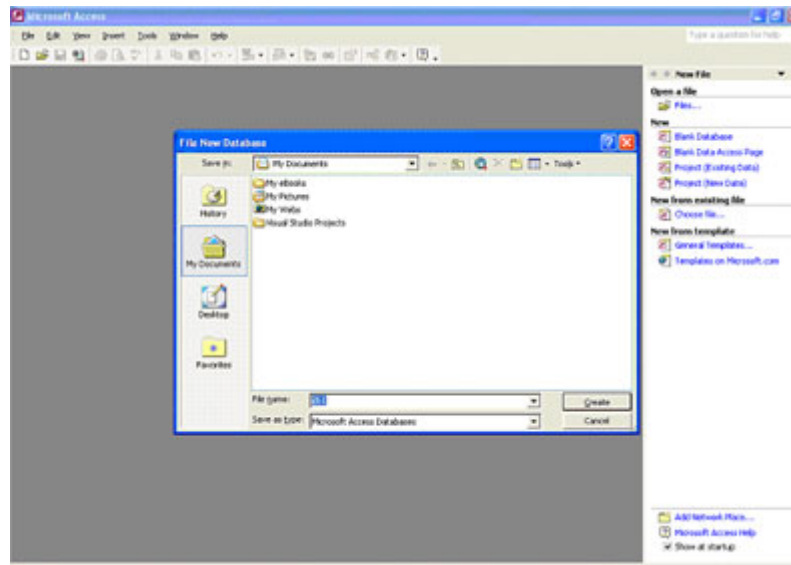
} catch(SQLException ex) {
    System.err.println(queryString);
    System.err.println("SQLException in ListStaff: " +
                       ex.getMessage());
}
}
```

> Addendum 4: Database Setup

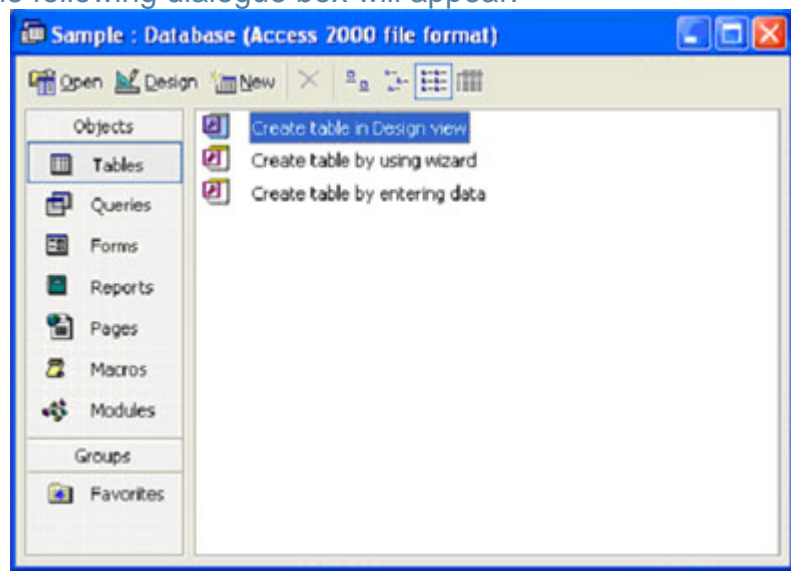


Instructions for Setting up an Access Database

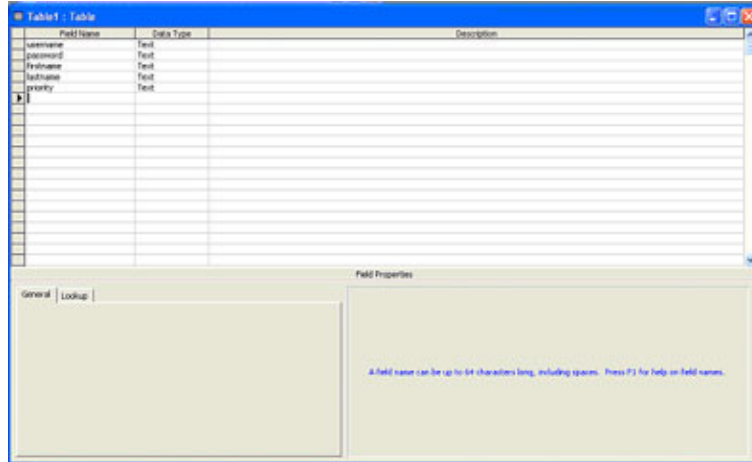
- 1 Decide upon the tables needed for your application and the specific data items (fields) that will be required. In this example, we will use a database called HelpDesk and two tables called passwords and requests. Open Microsoft Access and select File→New→ Blank Database. The following dialogue box will appear.



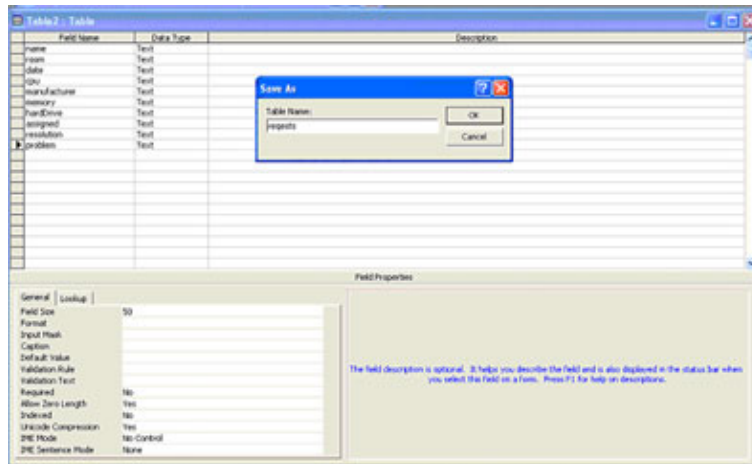
2. Name the new database 'HelpDesk'. Select an appropriate location and click Create. The following dialogue box will appear.



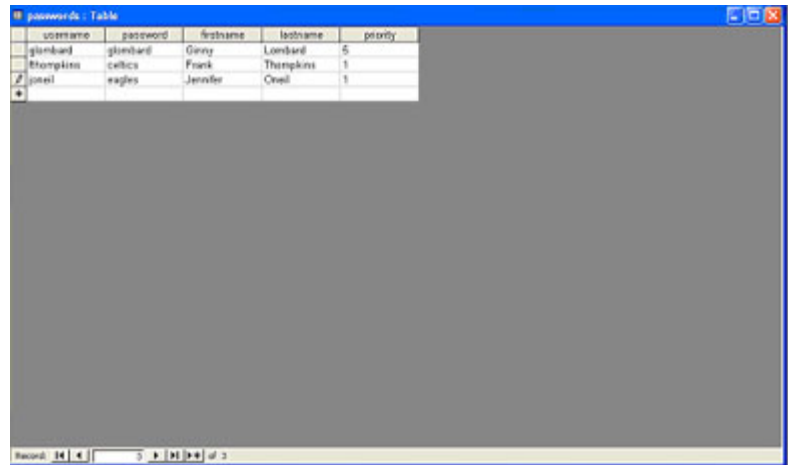
3. Select the option to create a table in Design View and then when the Design View screen appears, enter the field names as shown in the following screen view. Note: When setting up fields, just use the default type Text and let Access set the default size, which is 50.



- Once the design has been completed select File→Save. You will be prompted to name the table. Use passwords as the name here. Press Enter. You will be prompted for a key, but you do not need one here. Select No, that you do not wish to enter a key.
- Repeat Step 3 above to create a second table with fields as illustrated.



- Name this table requests and Save. No key is necessary.
- Open the passwords table by clicking on the passwords table and then selecting Open.
- Populate the passwords table. Input data for your institution. Use the priority field to differentiate technicians (here indicated by a priority 5) from casual users (priority 1). If you like, you can assign a wider range of priorities with different functionality associated with each of them.

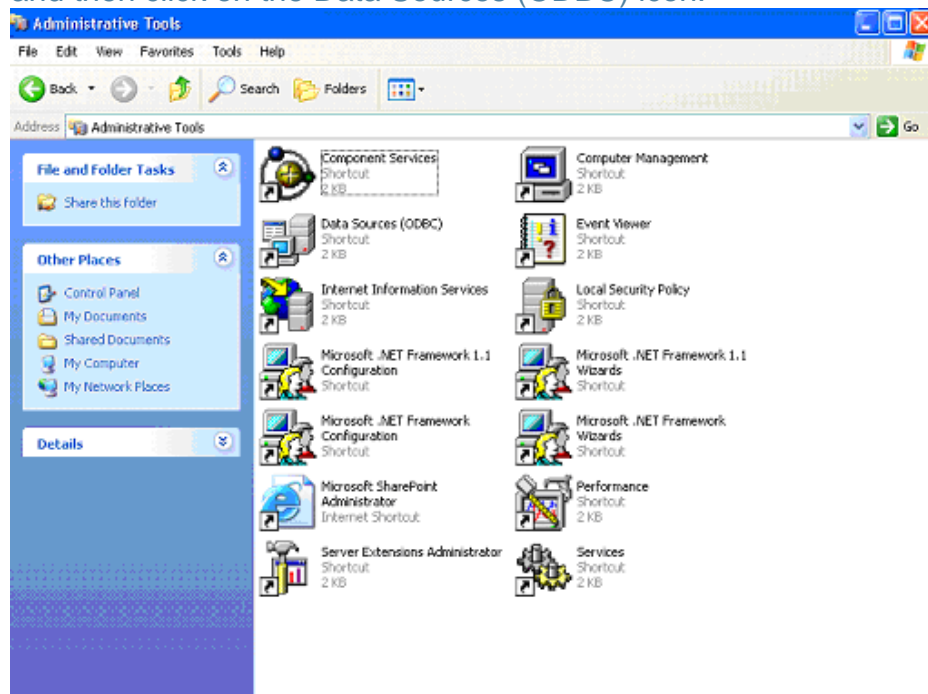


| username | password | firstame | lastame | priority |
|----------|----------|----------|----------|----------|
| glombard | glombard | Genz | Lowbard | 5 |
| thompson | celtics | Frank | Thompson | 1 |
| pete | eagles | Jennifer | Ozell | 1 |

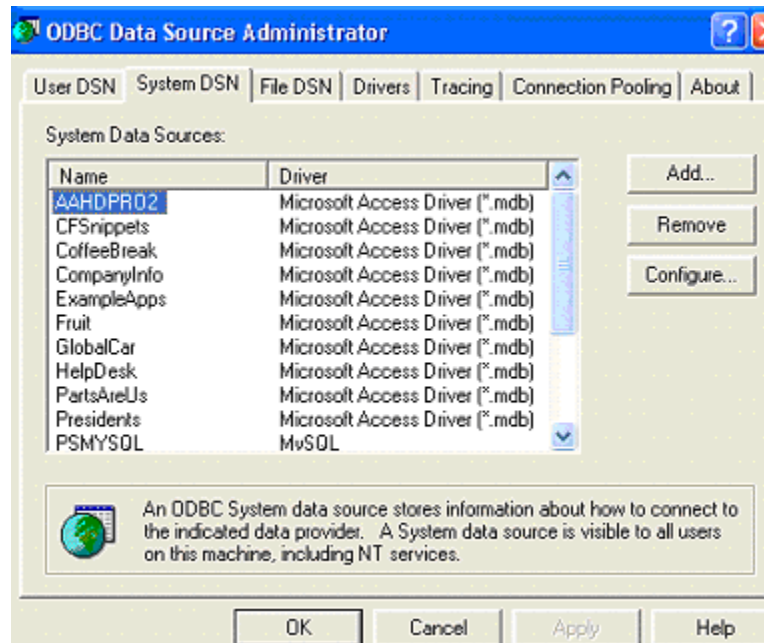
Addendum 5: ODBC Setup

Instructions for setting up a file in ODBC

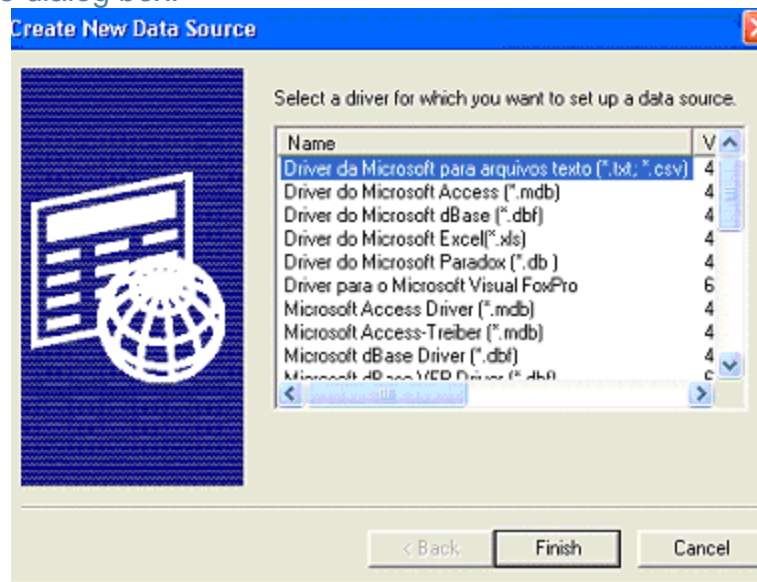
1. For older operating systems, call up the Control Panel and select ODBC. If you are using Windows XP, select Performance and Maintenance → Administrative Tools and then click on the Data Sources (ODBC) icon.



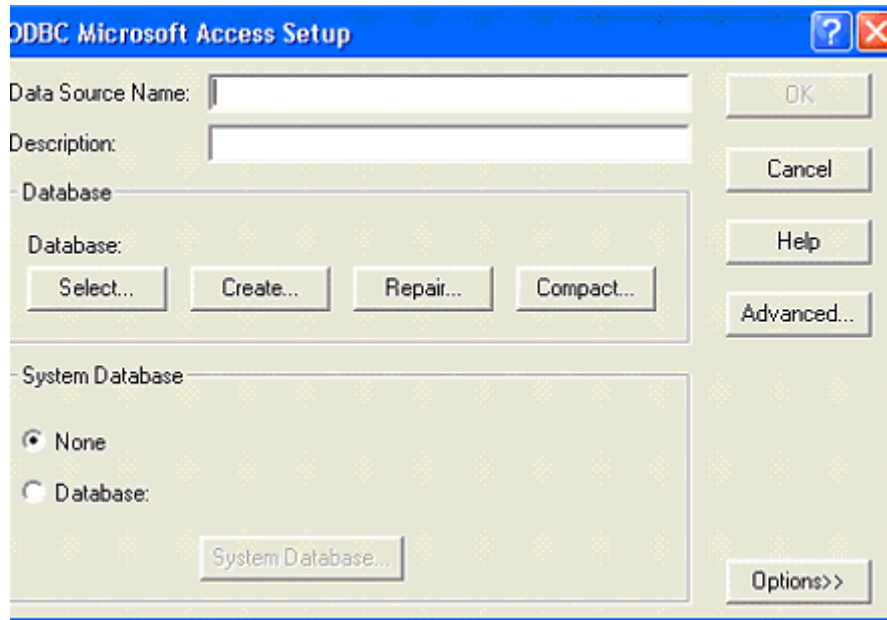
2. When the ODBC dialog box appears, select the System DSN panel.



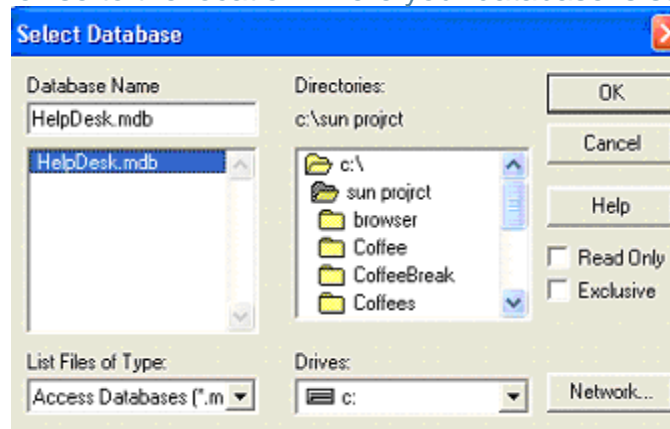
- Click Add to add a new Data Source. This action will bring up the Create New Data Source dialog box.



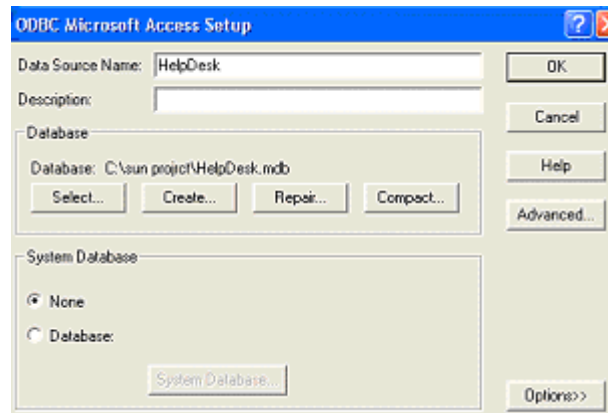
- Select Microsoft Access Driver and click Finish to bring up the Microsoft Access Setup screen.



6. Enter a Data Source Name. This is the name that will be used to reference the database. HelpDesk is the name of the database being referenced. What is entered here will be a name to be associated with the HelpDesk database – the name 'HelpDesk' might be useful as it will clearly relate to the Microsoft Access database with the same name. The name used here need not be the same as the name of the actual database. Filling in the description is optional. Next, click on the Select button just below the Database label in the center panel. That allows you to browse to the location where your database is stored.



7. Locate your database and click OK to make the connection between DSN Name and the physical database. When you return to the Microsoft Access Select dialog box, the database will be listed.



8. Click OK to finalize the ODBC setup. You can now use the reference or Data Source Name HelpDesk when you need to make any connections to your physical database.