18 Sailboat Marina Project

In this section we look at an implementation of the Sailboat Marina project previously described in Section 1.5.

This project is a web-based application that is used to manage a large sailboat marina that employs a database to keep track of sailboats, owners, whether the owners have paid their sailboat rental (and for what dates), and also handling any sailboat maintenance requests that come in from owners.

The idea is that a boat has an owner (string), a boat is of a particular model (string), and a boat is parked in a particular slip (parking spot next to the dock) identified by a number (integer) in the marina. The begin date and end date (in Date types, depending on your programming language) refer to the timeperiod for which the owner has paid slip rental. The marina also has one or more maintenance persons on staff to handle the maintenance problems of a boat, when a problem arises one of these persons should be assigned (asynchronously) to the problem.

18.1 SAILBOAT MARINA PROJECT IN ENTERPRISE JAVA BEANS

Before running the code in this section, you must implement a MySQL database named “thething” containing a table named “marina” as described in Section 4.11.1.1.

In these sections we will be using the Java Database Connectivity (JDBC) libraries. We won’t go over the details of the connections as they are presented here, since we previously covered that in Section 4.11.1.2.

To deploy JDBC so it was accessible by an Enterprise Java Beans application, I installed it in GlassFish itself: To make the JDBC library available to all applications in a domain, put it in the lib directory under the GlassFish domain directory then restart GlassFish.

18.1.1 MARINA INTERFACE

Listing 18.1 shows the interface that we use to access the database server code. The methods in this interface are implemented in Java methods in the marinaBean class shown in Listing 18.2. There is a method to display all items in the database (iddisplay), a method to create a new record in the database (Create), a method to delete (or show the details of) a method in the interface (Delete), and an asynchronous method (requestmaintenance) to request a maintenance person to be assigned to a maintenance task.

LISTING 18.1 Interface to EJB Marina Project

package marina;

import java.util.concurrent.Future;
import javax.ejb.Remote;

@Remote
public interface themarina {
    public void init_stuff();
    public Future<String> requestmaintenance();
    public String create(String namestr, String modelstr, String slipstr,
                         String begindatestr, String enddatestr);
}
18.1.2 Marina Bean

The marinaBean class in Listing 18.2 is an Enterprise Java Bean (EJB) that implements the interface previously shown in Listing 18.1. As an EJB, an object of this type class will be deployed in a GlassFish EJB container.

I chose a Stateful Session Bean, but for this particular project a Stateless Session Bean would also have worked. The Stateless Session Bean would be much more scalable, whereas a Stateful Session Bean is basically allocated per-client. In this particular case, I’m thinking of one primary client, the manager of the Sailboat Marina, running this application, which is why I chose the Stateful Session Bean.

Let’s look at the iddisplay method first. Using JDBC, it connects to the database then creates a query that requests information about all the major items in a record (name, model, slip, begin_date, end_date, id). It then iterates through the results of the query, looking at the values for each major item (name, model, slip, begin_date, end_date). It creates a string that contains the values for each major item, separated by tab characters. It then closes the connection.

Let’s look next at the Create method. It uses JDBC to connect to the database. Then it tries to find an unused id number, since the id number is supposed to be unique. The code I did here counts the number of rows, then picks an id number as one greater than the total number of rows. This does NOT work well and sometimes you end up with two items with the same id number. Why do you think this is? (Exercise for the reader: Hint, sometimes records must be deleted). Note that in the other implementations of the Marina project in this textbook, I did this differently, in a way that works better. Look it up! (This is actually one of the Conceptual Questions in this chapter.)

Then Create makes a “query” (better called just a database statement) that includes an SQL INSERT command to stuff a new record into the database, with the new values that Create had passed to it as parameters. It executes this statement and if it is successful then a new record has been added to the database.

The Delete method allows you to delete a record based either on the slip number or the id number, as both are supposed to be unique (slip is unique except when storing something like a rowboat in the same slip as a sailboat). Whichever one (id or slip) you don’t want to use to specify a record has a “–1” passed in for that value, because both id and slip are supposed to be positive numbers. Then it uses JDBC to connect to the database, and creates a statement that says to delete a record from the Marina based on an id number, or else delete a record from the Marina based on a slip number, depending on which was chosen earlier. Then it executes the statement and if the delete was successful, it passes a cute little message back to the client to the effect that the delete actually happened.

The requestmaintenance method is asynchronous and returns a Future value. We previously saw an asynchronous Stateful Session Bean in Section 8.3.5.2—we saw a Future return type there. Requestmaintenance simulates the process of someone hunting down an available maintenance person by sleeping for 10 seconds (in practice, in a real sailboat marina, this might be several days or even a couple of weeks). Then it returns a message about who the maintenance person is and when the person will be available. I generally make the availability Friday after 3pm because it’s usually a time when the owner doesn’t really want to go down to the Sailboat Marina, and that’s usually what happens.☺

Why do you think we did a separate connect to the database in each method? Rather than having one connect in the marinaBean constructor and then saving that connection. Well, this is a Stateful
Session Bean, but it could have been implemented as a Stateless Session Bean, connecting separately
can make that easier (see the Conceptual Questions).

LISTING 18.2 Marina Bean

```java
package marina;

import java.io.*;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.*;
import java.sql.SQLException;
import java.lang.Class;
import java.util.Calendar;
import java.util.Date;
import java.text.SimpleDateFormat;
import javax.ejb.Stateful;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.ejb.Remove;
import javax.ejb.PostActivate;
import javax.ejb.PrePassivate;
import javax.ejb.AsyncResult;
import javax.ejb.Asynchronous;
import java.util.concurrent.Future;
import javax.ejb.Stateful;
import java.lang.*;

@Stateful
public class marinaBean implements themarina {
    private int mycount=0;
    // set name of the JDBC driver
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver"
    // set database URL
    static final String DB_URL = "jdbc:mysql://localhost/thething";
    // set your database username and password
    static final String myUN = "myownstuff";
    static final String myPW = "samsung";
    
    @PostConstruct
    @PostActivate
    @Override
    public void init_stuff() {
        System.out.println("Doing PostConstruct or PostActivate");
    }
}
```
@Asynchronous
public Future<String> requestmaintenance()
{
    try {
        Thread.sleep (10000); // sleep for 10 seconds
    }
    catch (InterruptedException e) {
        e.printStackTrace();
    }
    return new AsyncResult<String>("Your Maintenance person is Carlos. He will be available Friday after 3pm");
}

public String create( String namestr, String modelstr, String slipstr,
        String begindatestr, String enddatestr ) {
    StringBuffer myout = new StringBuffer("\n");
    System.out.println("Inside create method");
    Connection conn = null;
    Statement stmt = null;
    try{
        // Register the JDBC driver
        Class.forName("com.mysql.jdbc.Driver");

        // Connect to the database
        System.out.println("Attempting to connect to database. Trying hard!");
        conn = DriverManager.getConnection(DB_URL,myUN,myPW);

        // Create a query to the database
        System.out.println("Attempting to query database");

        // set the id to the number of rows plus one
        // to do this we must first read the number of rows from database
        String mysql_Query;
        stmt = conn.createStatement();
        mysql_Query = "SELECT COUNT(*) FROM marina";
        ResultSet rs = stmt.executeQuery(mysql_Query);
        rs.next();
        theid=rs.getInt(1);
        theid=theid+1;
        System.out.println("theid: "+theid);
        rs.close();

        String mysql_QueryA;
        mysql_QueryA = "INSERT INTO marina (name, model, slip,
        begin_date,end_date,id) VALUES (?, ?, ?, ?, ?)";
        PreparedStatement statement = conn.prepareStatement (mysql_QueryA);
statement.setString(1,namestr);
statement.setString(2,modelstr);

int theslip = Integer.parseInt(slipstr);
statement.setInt(3,theslip);

SimpleDateFormat sdf = new SimpleDateFormat("MM-dd-yyyy");

Date thedate = sdf.parse(begindatestr);
java.sql.Date sqlbegin_date= new java.sql.Date(thedate.getTime());
statement.setDate(4,sqlbegin_date);

thedate = sdf.parse(enddatestr);
java.sql.Date sqlend_date= new java.sql.Date(thedate.getTime());
statement.setDate(5,sqlend_date);

statement.setInt(6,theid); // id field

int rows_inserted = statement.executeUpdate();
if (rows_inserted>0)
{
    System.out.println("inserted number of rows: "+rows_inserted);
}

// finish up when done
conn.close();
}
catch(SQLException sqle){
    // JDBC problems
    sqle.printStackTrace();
}
catch(Exception e){
    e.printStackTrace();
}
}finally{
    try{
        if(stmt!=null)
            stmt.close();
    }catch(SQLException se){
    }
    try{
        if(conn!=null)
            conn.close();
    }catch(SQLException se){
        se.printStackTrace();
    } //end finally
} //end try

System.out.println("Be seein’ y’all");
myout.append("Successfully created new row in database");

String thestring = myout.toString();

return thestring;

public String iddisplay(String name) {

    StringBuffer myout = new StringBuffer(
        "");

    // Following output will show up in Glassfish log file
    // because System.out is standard output it will show up
    // on the console
    System.out.println("Inside receivedGet method");

    Connection conn = null;
    Statement stmt = null;
    try{
        // Register the JDBC driver
        Class.forName("com.mysql.jdbc.Driver");

        // Connect to the database
        System.out.println("Attempting to connect to database.
                     Trying hard!");
        conn = DriverManager.getConnection(DB_URL,myUN,myPW);

        // Create a query to the database
        String mysql_Query;
        System.out.println("Attempting to query database");
        stmt = conn.createStatement();
        mysql_Query = "SELECT name,model,slip,begin_date,end_date,id
                       FROM marina";
        ResultSet rs = stmt.executeQuery(mysql_Query);

        // Get all data in the database
        while(rs.next()){
            // grab the data in order by data column name
            String thename = rs.getString("name");
            String themodel = rs.getString("model");
            int theslip = rs.getInt("slip");
            String thebegindate = rs.getString("begin_date");
            //String theenddate = rs.getString("end_date");
            //Timestamp theenddate = rs.getTimestamp("end_date");
            Date theenddate = rs.getDate("end_date");
            Calendar mycalendar=Calendar.getInstance();
            mycalendar.setTime(theadendate);
            int month = mycalendar.get(Calendar.MONTH);

            //...
int day = mycalendar.get(Calendar.DAY_OF_MONTH);
int year = mycalendar.get(Calendar.YEAR);

int theid = rs.getInt("id");
myout.append(thename);
myout.append("\t");
myout.append(themodel);
myout.append("\t");
myout.append(theslip);
myout.append("\t");
myout.append(thedate);
myout.append("\t");
myout.append(thedate);
myout.append("\n");
}

// finish up when done
rs.close();
stmt.close();
conn.close();

} catch(SQLException sqle){
    // JDBC problems
    sqle.printStackTrace();
} catch(Exception e){
    e.printStackTrace();
} finally{
    // “finally” shut everything down
    try{
        if(stmt!=null)
            stmt.close();
        } catch(SQLException se){
            // ignore
        }
    try{
        if(conn!=null)
            conn.close();
        } catch(SQLException se){
            se.printStackTrace();
    } //end finally
} //end try

System.out.println("Be seein’ y’all");
String thestring = myout.toString();
return thestring;
}
public String delete(String slipstr, String idstr ) {

    // pass in -1 to signal not using that item (slipstr or idstr)
    StringBuffer myout = new StringBuffer(""");
    int whichslip;
    int whichid;

    whichid = 0;
    whichslip = 0;

    if ( !slipstr.equals("-1") )
    {
        System.out.println("got here a, slipstr is "+slipstr+" idstr is "+idstr);
        whichslip =Integer.parseInt(slipstr);
    }
    else if ( !idstr.equals("-1") )
    {
        System.out.println("got here b, slipstr is "+slipstr+" idstr is "+idstr);
        whichid =Integer.parseInt(idstr);
    }

    System.out.println("got here c, slipstr is "+slipstr+" idstr is "+idstr);

    Connection conn = null;
    try{
        // Register the JDBC driver
        Class.forName("com.mysql.jdbc.Driver");

        // Connect to the database
        System.out.println("Attempting to connect to database.
        Trying hard!");
        conn = DriverManager.getConnection(DB_URL,myUN,myPW);

        // Create a query to the database
        String mysql_stmt;
        System.out.println("Attempting to query database");

        if (whichslip !=0)
        {
            mysql_stmt = "DELETE FROM marina where slip="+whichslip;
        }
        else if (whichid !=0)
        {
            mysql_stmt = "DELETE FROM marina where id="+whichid;
        }
        else
        {
            mysql_stmt = "DELETE FROM marina where slip=0"
            // default to slip =0 which should result in no delete
        }
    }
}
PreparedStatement stmt = conn.prepareStatement(mysql_stmt);
int rows_deleted = stmt.executeUpdate();
if (rows_deleted > 0)
    System.out.println("Row was successfully deleted.");
myout.append("\n");
myout.append("Delete was performed");
myout.append("\n");

// finish up when done
conn.close();

} catch(SQLException sqle)
{
    // JDBC problems
    sqle.printStackTrace();
} catch(Exception e)
{
    e.printStackTrace();
}
finally
{
    // “finally” shut everything down
    try{
        if(conn!=null)
            conn.close();
    } catch(SQLException se)
    {
        se.printStackTrace();
    } //end catch
} //end finally
System.out.println("Be seein’ y’all");
String thestring = myout.toString();
return thestring;
}

18.1.3 MARINA CONSOLE CLIENT
The Console Client is implemented as a Plain Old Java Object (POJO), actually as a free standing Java application. Here in the Console Client, we use the Java Naming and Directory Interface (JNDI), see Section 8.3.1, to create a proxy that points to the marinaBean (as it is deployed in the GlassFish container).
It calls the requestmaintenance method on the marinaBean first (it uses the Future interface), and illustrates that it can do useful work while waiting for the asynchronous requestmaintenance method
to assign a maintenance person—it uses polling to determine when the requestmaintenance method is done.

Then it calls the Create method on the marinaBean to create an arbitrary record. I didn’t fix this so it could really read new data from the console—instead you would have to put in data to this call and re-run this Java application.

Then it deletes an arbitrary record that happens to have an id=9 (it sets the slip number to -1 so it’s not using a slip to specify the deleted value).

Then it does a call to the iddisplay method to display all the records in the database.

**LISTING 18.3 Marina Console Client**

```java
package marina;
import javax.naming.InitialContext;
import javax.naming.Context;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;
import javax.ejb.AsyncResult;
import javax.ejb.Asynchronous;
import java.util.concurrent.Future;
public class marinaClient {
    public static void main (String[] args) {
        try {
            Context ic= new InitialContext();
            // Get one proxy object
            // Names in the java:global JNDI namespace are accessible to any application.
            // (This is how EJB references may be accessed outside the application where
            // they’re defined.)
            Object object_ref = ic.lookup("java:global/marina/marinaBean");
            themarina proxyA= (themarina) PortableRemoteObject.narrow
            (object_ref,themarina.class);
            System.out.println("Now get the count from original proxy");
            Future<String> mystring = proxyA.requestmaintenance();
            int i=0;
            while (!mystring.isDone() )
            {
                // We could actually be doing something useful now.
                // I’ll just count to simulate the useful work :-)
                System.out.println("Doing useful counting, i = "+i);
                i++;
            }
            String myfinalstring = mystring.get();
            System.out.println("string returned was "+myfinalstring);
```
String mystuff = proxyA.create("name","model","18","07-07-2016","07-07-2016");

String mystuffA = proxyA.delete("-1","9");

String thestuff = proxyA.iddisplay("test");
System.out.println(thestuff);

} catch (NamingException ex) {
    // You might get a naming exception if you forgot to deploy the jar file
    // You might get a naming exception if you forgot to start Glassfish
    System.out.println("Naming exception");
    System.out.println("cause is"+ex.getCause());
    System.out.println("stack trace is ");
    ex.printStackTrace();
} catch (Exception ex) {
    System.out.println("general exception");
}

} // end Main()
} // end class

18.1.4 J AVA SWING MARINA GUI – EJB

This section describes a graphical user interface (GUI) for the EJB Sailboat Marina project, written using Java Swing. Some specifics about Java Swing were previously covered in Section 17.1.4.

If you recall from that section, buttons are set by creating a JButton, then setting an Action Command, and an associated Action Listener to handle the processing required by the button. To begin with, look at Figure 18.1. This is the main page for this GUI; it is called the “manage” page and is handled by class manage in Listing 18.4. The first button, b1, of type JButton is established as follows:

b1=new JButton("Display with ids");
b1.setBounds(100,50,500,50);
b1.setActionCommand("display");
b1.addActionListener(this);

When we call “new JButton”, the name we pass, in this case “Display with ids,” will be shown on the screen as the name of the button, see Figure 18.1.

Note the setActionCommand in the case of each button. We set the action command to “display” for the Display with ids button and we set it to “create” for the Add Sailor/Sailboat to Database button, etc. We use this in the actionPerformed method to figure out which button was pushed, as follows:

- if ("display").equals(e.getActionCommand())
- if ("create").equals(e.getActionCommand())
- etc.

Then the appropriate class to handle each separate button is instantiated. The iddisplayGUI class is instantiated to handle the “display” action command, that is, to display all the records in the database,
see Figure 18.2. The createGUI class is instantiated to handle the “create” action command, that is, to create a new record for the database, see Figure 18.3. The details_and_deleteGUI class is instantiated to handle the “details” action command, that is, to show the details of a single record or else delete a single record in the database, see Figure 18.4. The requestmaintenance class is instantiated to handle the “requestmaintenance” action command, that is, to handle requesting a person to perform maintenance, see Figure 18.5.

Note that as each new display is selected, the current Manage screen is set to not visible:

```java
this.setVisible(false);
```

And after the new display is created (after its appropriate class is instantiated) that new display is set visible:

```java
mydisplayB.setVisible(true);
```
LISTING 18.4  Java Swing Marina GUI – EJB

package marina;
import javax.naming.InitialContext;
import javax.naming.Context;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;
import javax.ejb.AsyncResult;
import javax.ejb.Asynchronous;
import java.util.concurrent.Future;

FIGURE 18.3  Create New Item in Database.

FIGURE 18.4  Display and/or Delete Item.

FIGURE 18.5  Request Maintenance.
```java
import javax.swing.*;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableCellRenderer;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

public class manage {
    static select_the_display the_display;
    static String data[][] = new String[][] {
        {
            "name", "boat model", "0", "01-23-1979", "01-23-1979", "1" },
        {
            "", "", "", "", "", "" },
        {
            "", "", "", "", "", "" },
        {
            "", "", "", "", "", "" },
        {
            "", "", "", "", "", "" },
        {
            "", "", "", "", "", "" },
        {
            "", "", "", "", "", "" },
    };

    static String columnHeaders[] = new String[] {
        { "name", "model", "slip", "begin_date", "end_date", "id" },
    };

    public static class iddisplayGUI extends JFrame implements ActionListener {
        class myRenderer implements TableCellRenderer {
        }
    }
```
// override getTableCellRendererComponent. It return
// component used to draw the cell. So you can make that
// component be a JTextArea
public Component getTableCellRendererComponent(JTable table,
                                               Object thevalue, boolean isSelected,
                                               boolean hasFocus, int row, int column) {
    JTextArea myTextArea = new JTextArea();
    if (thevalue != null)
        myTextArea.setText(thecode.toString());
    return myTextArea;
} // end nested class myRenderer

JTable mytable = new JTable(data, columnHeaders);
static int maxtablesize = 0;

String displayed_text = "";

JButton homebutton, updatebutton;

public idisplayGUI() // constructor
{
    Container container = getContentPane();

    // FlowLayout puts GUI items in order
    container.setLayout(new FlowLayout());

    // home Button
    homebutton = new JButton("Home");
    homebutton.addActionListener(this);
    container.add(homebutton);
    mytable.setDefaultRenderer(Object.class, new myRenderer());
    mytable.getTableHeader().setFont(new Font("Sans Serif",
                                               Font.BOLD, 10));
    mytable.getTableHeader().setBackground(Color.GREEN);
    container.add(new JScrollPane(mytable));

    // Update Button
    updatebutton = new JButton("Update");
    updatebutton.addActionListener(this);
    container.add(updatebutton);

    setSize(1000, 300);

} //end of constructor
public void actionPerformed(ActionEvent e) {
    if ("update".equals(e.getActionCommand())) {
        String response=call_the_iddisplay();
        extract_info_from_the_iddisplay(response);
    } else if ("home".equals(e.getActionCommand())) {
        this.setVisible(false);
        the_display.setVisible(true);
    }
} // end actionlistener

void extract_info_from_the_iddisplay(String mystring) {
    try {
        StringReader strreader = new StringReader(mystring);
        BufferedReader br = new BufferedReader(strreader);
        String currentline;
        int i=0;
        while ((currentline = br.readLine()) != null) {
            if (i>maxtablesize) // update the max table size
                maxtablesize=i;
            String [] items = currentline.split("\t");
            for (int j=0; j<items.length; j++) {
                mytable.setValueAt(items[j], i, j);
            }
            i++;
        } // end while

        // Clear out the remainder of the table
        if (maxtablesize<15) // make sure at least always
            maxtablesize=15;
        for (int k=i-1; k<maxtablesize; k++)
            
    }
} // end extract_info_from_the_iddisplay
for(int j=0; j<6; j++)
{
    mytable.setValueAt("", i, j);
}

} // end extract info from the display

String call_the_iddisplay()
{
    String thestuff;
    thestuff="";
    try {
        Context ic= new InitialContext();
        // Get one proxy object
        // Names in the java:global JNDI namespace are accessible to any
        // application.
        // (This is how EJB references may be accessed outside the application
        // where they're defined.)
        Object object_ref = ic.lookup("java:global/marina/marinaBean");
        themarina proxyA= (themarina)
            PortableRemoteObject.narrow(object_ref,themarina.class);
        thestuff = proxyA.iddisplay("test");
        System.out.println(thestuff);
    }
    catch (NamingException ex) {
        // You might get a naming exception if you forgot to deploy the jar file
        // You might get a naming exception if you forgot to start Glassfish
        System.out.println("Naming exception");
        System.out.println("cause is"+ex.getCause());
        System.out.println("stack trace is ");
        ex.printStackTrace();
    } catch (Exception ex) {
        System.out.println("general exception");
    }

    return thestuff;

} // end call_the_iddisplay

} // end iddisplayGUI class
public static class createGUI extends JFrame implements ActionListener {
    class myRenderer implements TableCellRenderer {
        // override getTableCellRendererComponent. It return
        // component used to draw the cell. So you can make that
        // component be a JTextArea
        public Component getTableCellRendererComponent(JTable table, Object thevalue, boolean isSelected,
            boolean hasFocus, int row, int column) {
            JTextArea myTextArea = new JTextArea();
            if (theValue != null)
                myTextArea.setText(theValue.toString());
            return myTextArea;
        }
    } // end nested class myRenderer
    static String data[][] = new String [][] {
        {"name","boat model","0","01-23-1979","01-23-1979","1"},
    };
    static String columnHeaders[] = new String [] {
        {"name","model","slip","begin_date","end_date","id"};
    JTable mytable = new JTable(data,columnHeaders);
    String displayed_text = "";
    JButton homebutton, updatebutton;
    
    public createGUI() // constructor
    {
        Container container = getContentPane();
        // FlowLayout puts GUI items in order
        container.setLayout(new FlowLayout());
        // Home Button
        homebutton= new JButton("Home");
        //homebutton.setBounds(100,50,100,50);
        homebutton.setActionCommand("home");
        homebutton.addActionListener(this);
        container.add(homebutton);
    }
mytable.setDefaultRenderer(Object.class, new myRenderer());
mytable.getTableHeader().setFont(new Font("Sans Serif",Font.BOLD,10));
mytable.getTableHeader().setBackground(Color.GREEN);
container.add(new JScrollPane(mytable));

// Update Button
updatebutton = new JButton("Update");
//homebutton.setBounds(100,50,100,50);
updatebutton.setActionCommand("update");
updatebutton.addActionListener(this);
container.add(updatebutton);

setSize(1000, 300);
//setVisible(true);
} //end of constructor

public void actionPerformed(ActionEvent e)
{
    if ("update".equals(e.getActionCommand())) {
        String response=call_the_create();
    } else if ("home".equals(e.getActionCommand())) {
        this.setVisible(false);
        the_display.setVisible(true);
    }

    // JOptionPane.showMessageDialog(null, displayed_text);
} // end actionlistener

String call_the_create()
{
    String mystuff;
    mystuff="";
    String item0 = (mytable.getValueAt(0,0)).toString();
    String item1 = (mytable.getValueAt(0,1)).toString();
    String item2 = (mytable.getValueAt(0,2)).toString();
    String item3 = (mytable.getValueAt(0,3)).toString();
    String item4 = (mytable.getValueAt(0,4)).toString();
    String item5 = (mytable.getValueAt(0,5)).toString();

    try {
        Context ic= new InitialContext();
        // Get one proxy object
        // Names in the java:global JNDI namespace are accessible to any application.
// (This is how EJB references may be accessed outside the
// application where
// they’re defined.)
Object object_ref = ic.lookup("java:global/marina/marinaBean");
themarina proxyA = (themarina)
    PortableRemoteObject.narrow(object_ref,themarina.class);

mystuff = proxyA.create(item0,item1,item2,item3,item4);

} catch (NamingException ex) {
    // You might get a naming exception if you forgot to deploy the jar file
    // You might get a naming exception if you forgot to start Glassfish
    System.out.println("Naming exception");
    System.out.println("cause is"+ex.getCause());
    System.out.println("stack trace is ");
    ex.printStackTrace();
} catch (Exception ex) {
    System.out.println("general exception");
}

return mystuff;

} // end call_the_create
} // end createGUI class

public static class details_and_deleteGUI extends JFrame implements ActionListener {

    class myRenderer implements TableCellRenderer {

        // override getTableCellRendererComponent. It return
        // component used to draw the cell. So you can make that
        // component a JTextArea
        public Component getTableCellRendererComponent(JTable table,
            Object thevalue, boolean isSelected,
            boolean hasFocus, int row, int column) {

            JTextArea myTextArea = new JTextArea();
            if (thevalue != null)
                myTextArea.setText(thevalue.toString());
            return myTextArea;

        } // end nested class myRenderer

        static String data[][] = new String [][] {
            {"name","boat model","0","01-23-1979","01-23-1979","1"},
        };
static String columnHeaders[] = new String []
{"name","model","slip","begin_date","end_date","id"};

JTable mytable = new JTable(data,columnHeaders);
String displayed_text = "";

public details_and_deleteGUI() // constructor
{
    Container container = getContentPane();

    // FlowLayout puts GUI items in order
    container.setLayout(new FlowLayout());

    // home Button
    homebutton= new JButton("Home");
    homebutton.setActionCommand("home");
    homebutton.addActionListener(this);
    container.add(homebutton);

    mytable.setDefaultRenderer(Object.class, new myRenderer());
    mytable.getTableHeader().setFont( new Font("Sans Serif",Font.BOLD,10));
    mytable.getTableHeader().setBack ground(Color.GREEN);
    container.add( new JScrollPane(mytable) );

    // Update Button
    updatebutton= new JButton("Update");
    //homebutton.setBounds(100,50,100,50);
    updatebutton.setActionCommand("update");
    updatebutton.addActionListener(this);
    container.add(updatebutton);

    // Delete Button
    deletebutton= new JButton("Delete");
    //homebutton.setBounds(100,50,100,50);
    deletebutton.setActionCommand("delete");
    deletebutton.addActionListener(this);
    container.add(deletebutton);

    setSize(1000, 300);
    //setVisible(true);
    }

public void actionPerformed(ActionEvent e)
{

if ("update".equals(e.getActionCommand()) ) {
        String response=call_the_iddisplay();
        extract_info_from_the_iddisplay(response);
    }
    else if ("home".equals(e.getActionCommand()) ) {
        this.setVisible(false);
        the_display.setVisible(true);
    }
    else if ("delete".equals(e.getActionCommand()) ) {
        delete_the_item();
    }
} // end actionlistener

void extract_info_from_the_iddisplay(String mystring)
{
    try {
        StringReader strreader = new StringReader(mystring);
        BufferedReader br = new BufferedReader(strreader);
        String currentline;
        int i=0;

        while ((currentline = br.readLine()) != null)
        {
            String [] items = currentline.split("\t");
            if (items[5].equals(mytable.getValueAt(0,5)))
            {
                for (int j=0; j<items.length; j++)
                {
                    mytable.setValueAt(items[j], 0, j);
                }
                i++;
            }
        }
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
}

} // end extract info from the display
void delete_the_item()
{

    try {
        Context ic= new InitialContext();

        // Get one proxy object
        // Names in the java:global JNDI namespace are accessible to any
        // application.
        // (This is how EJB references may be accessed outside the application
        // where
        // they’re defined.)
        Object object_ref = ic.lookup("java:global/marina/marinaBean");
        themarina proxyA= (themarina)
        PortableRemoteObject.narrow(object_ref,themarina.class);

        String firstvalue="-1";
        String secondvalue=mytable.getValueAt(0,5).toString();

        System.out.println("secondvalue is "+secondvalue);
        String response = proxyA.delete(firstvalue,secondvalue);
        System.out.println("response is "+response);

        for (int j=0; j<6; j++)
        {
            mytable.setValueAt(firstvalue, 0, j);
        }
    }
    catch (NamingException ex) {
        // You might get a naming exception if you forgot to deploy the jar file
        // You might get a naming exception if you forgot to start Glassfish
        System.out.println("Naming exception");
        System.out.println("cause is "+ex.getCause());
        System.out.println("stack trace is ");
        ex.printStackTrace();
    } catch (Exception ex) {
        System.out.println("general exception");
    }

    } // delete_the_item

String call_the_iddisplay()
{

    String thestuff;
thestuff=""

try {
    Context ic = new InitialContext();

    // Get one proxy object
    // Names in the java:global JNDI namespace are accessible to any
    // application.
    // (This is how EJB references may be accessed outside the application
    // where
    // they’re defined.)
    Object object_ref = ic.lookup("java:global/marina/marinaBean");

    themarina proxyA = (themarina)
        PortableRemoteObject.narrow(object_ref, themarina.class);

    thestuff = proxyA.iddisplay("test");
    System.out.println(thestuff);
}

return thestuff;

} // end call_the_iddisplay

} // end details class

public static class requestmaintenance extends JFrame implements ActionListener {

    JTextArea mytextA;
    String displayed_text = "";

    JButton refreshbutton, updatebutton;

    public requestmaintenance() {
        Container container = getContentPane();
        container.setLayout(new FlowLayout());
        mytextA = new JTextArea("name", 10, 30);
        container.add(mytextA);
        refreshbutton= new JButton("Request Maintenance");
        refreshbutton.setBounds(100,50,100,50);
    }

} // end requestmaintenance class
refreshbutton.addActionListener(this);

container.add(refreshbutton);

setSize(1000, 500);
setVisible(true);

public void actionPerformed(ActionEvent e) {

try {
   Context ic= new InitialContext();

   // Get one proxy object
   // Names in the java:global JNDI namespace are accessible to any application.
   // (This is how EJB references may be accessed outside the application where
   // they’re defined.)
   Object object_ref = ic.lookup("java:global/marina/marinaBean");

   themarina proxyA= (themarina)
   PortableRemoteObject.narrow(object_ref,themarina.class);
   System.out.println(“about to request maintenance”);

   Future<String> mystring = proxyA.requestmaintenance();

   int i=0;
   while (!mystring.isDone() )
   {
      // We could actually be doing something useful now.
      // I’ll just count to simulate the useful work :-)
      System.out.println(“Doing useful work, i is ”+i);
      i++;
   }

   String myfinalstring = mystring.get();
   System.out.println(“string returned was ”+myfinalstring);

   String nameused = (mytextA.getText()).toString();
   System.out.println(“nameused is”+nameused);

   String thefinalresponse = “Hello, ”+nameused+“ ”+myfinalstring;
   mytextA.setText(thefinalresponse);

} catch (NamingException ex) {
   // You might get a naming exception if you forgot to deploy the jar file
   // You might get a naming exception if you forgot to start Glassfish
   System.out.println(“Naming exception”);
   System.out.println(“cause is”+ex.getCause());
System.out.println("stack trace is ");
ex.printStackTrace();
} catch (Exception ex) {
    System.out.println("general exception");
}
}
}
static class select_the_display extends JFrame implements ActionListener{

    JButton b,b1,b2, b3;

    select_the_display(){

        b1=new JButton("Display with ids");
b1.setBounds(100,50,500,50);
b1.setActionCommand("display");
b1.addActionListener(this);

        b2=new JButton("Add Sailor/Sailboat to Database ");
b2.setBounds(100,150,500,50);
b2.setActionCommand("create");
b2.addActionListener(this);

        b=new JButton("Choose Show Details or Delete Record in Database  
(enter id) ");
b.setBounds(100,250,500,50);
b.setActionCommand("details");
b.addActionListener(this);

        b3=new JButton("Request Maintenance");
b3.setBounds(100,350,500,50);
b3.setActionCommand("requestmaintenance");
b3.addActionListener(this);

        // Add components to current container
        add(b1);
        add(b2);
        add(b);
        add(b3);

        setSize(900,500);
        setLayout(null);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e){
        if ("display".equals(e.getActionCommand())) {
            this.setVisible(false);
        }
As we said before, the iddisplayGUI class is instantiated to handle the “display” action command, that is, to display all the records in the database. On Figure 18.2 there is a Home button and an Update button. The Home button takes you back to the Manage screen and the Update button reads all the records from the database (at least, all that will fit in the table on the screen).
This table on the screen is created as follows: First, column headers for the table are defined, and an array describing the table contents is defined as follows:

```java
static String data[][] = new String [][] {
    {"name","boat model","0","01-23-1979","01-23-1979","1"},
    {"",""},
    {""},
    {""},
    {""},
    {""},
    {""},
    {""},
    {""},
    {""},
    {""},
    {""},

static String columnHeaders[] = new String []{
    "name","model","slip","begin_date","end_date","id"};
```

Then the table is created using the following statement:

```java
JTable mytable = new JTable(data,columnHeaders);
```

Then a new (nested) "renderer" class is defined, myRenderer, which implements TableCellRenderer. What this new myRenderer class does is to set the individual items in a table to be TextAreas, where data can be displayed and entered (otherwise these items would have been just display items). The table which was just created (mytable) is set to use this new myRenderer class by using the following statement:

```java
mytable.setDefaultRenderer(Object.class, new myRenderer());
```

To pretty things up, then mytable has its header fonts set:

```java
mytable.getTableHeader().setFont( new Font("Sans Serif",Font.BOLD,10) );
```

And then its header has its background color set:

```java
mytable.getTableHeader().setBackground(Color.GREEN);
```

And finally the table is added to the current container. (The container is a pane that is employing a FlowLayout. We previously saw FlowLayout for Swing GUIs in Section 17.1.4):

```java
container.add( new JScrollPane(mytable) );
```
Then when the update button is pushed, the actionPerformed handler runs the code associated with the “update” action command. This results in calling the “call_the_iddisplay()” method, which uses JNDI to establish a connection to the Stateful Session marinaBean, which then performs the appropriate database access. Then the “extract_info_from_the_iddisplay” method is called, which uses the tab characters in the marinaBean response to break the response up into all the separate values, one for each column, and these are written into mytable (on the screen) as follows:

```java
String [] items = currentline.split("t");
for (int j=0; j<items.length; j++)
{
    mytable.setValueAt(items[j], i, j);
}
i++;
```  

The createGUI class is instantiated to handle the “create” action command, that is, to create a new record for the database. The table shown in Figure 18.3 works similarly to the table in Figure 18.2 from the iddisplayGUI class except that only one row of the class is shown, the row that will hold the data that will be written to the database. When the update button shown in Figure 18.3 is pushed, a connection will be created to the marinaBean using JNDI, and thus established; the marinaBean performs the appropriate update of the database. The data passed to the marinaBean to be used in creating the new record will be extracted from the table as follows:

```java
String item0 = (mytable.getValueAt(0,0)).toString();
String item1 = (mytable.getValueAt(0,1)).toString();
```  

The details_and_deleteGUI class is instantiated to handle the “details” action command, that is, to show the details of a single record or else delete a single record in the database. From Figure 18.4 you can see that the single row of the table shown is implemented in the GUI in the same way that it worked for Create (and similarly to how it worked for Display All Items, except with one row instead of several). Here, “update” will result in a call to the marinaBean to display a single row, whereas “delete” will result in a call to marinaBean to delete the item based on the id number shown in the table. It extracts the id to be passed to the marinaBean for the delete as follows:

```java
String secondvalue=mytable.getValueAt(0,5).toString();
```  

Note that this code always assumes it’s going to be deleting based on the id number; it sets the slip value passed to the marinaBean to a “–1”.

The requestmaintenance class is instantiated to handle the “requestmaintenance” action command, that is, to handle requesting a person to perform maintenance. The requestmaintenance class is a much simpler GUI than the others as it does not use a table; see Figure 18.5. It just sets up a TextArea to collect the name from the user, and employs a button that, when pressed, results in a call to requestmaintenance on the marinaBean.

### 18.2 SAILBOAT MARINA PROJECT IN JAX-WS

In this section we look at an implementation of the Sailboat Marina project previously described in Section 1.5.
Before running the code in this section, you must implement a MySQL database named “thething” containing a table named “marina” as described in Section 4.11.1.1.

In these sections we will be using the Java Database Connectivity (JDBC) libraries. We won’t go over the details of the connections as they are presented here, since we previously covered that in Section 4.11.1.2.

The code in this section is similar to the Enterprise Java Beans Sailboat Marina implementation in Section 18.1, in that the code in the server to access the database is the same. However, the way the server code is organized and published is different. With EJB, all the server code is in a single server instead of in multiple servers, and this single server is published inside the GlassFish container, whereas with this JAX-WS code, the server code is broken into several different servers at several locations, and is published using the Java self-publisher.

The console clients shown here are quite different from the console client used with EJB. The console clients in this section call code that was automatically generated using wsimport (accessing the WSDL of the running servers), whereas the EJB code was a POJO that used JNDI to access the server.

The graphical user interface client used here is based on Java swing, and the GUI interface is the same as that previously seen in Section 18.1. However, the client code employed in the JAX-WS GUI interface calls code that was automatically generated using wsimport.

18.2.1 MARINA SERVERS

The servers are all separately published using the Java self-publisher, at different port numbers on the same server (in this case, localhost). You could add other servers to this list as desired (in the code provided for the instructor with this textbook, I have a hello world server as well; it’s convenient for making sure things are working).

Note that when you use the graphical user interface, in Listing 18.16, this could be considered to be an example of a Service-Oriented Architecture, because several different services will be contacted by a single client to perform a particular task.

If you want to access the WSDL associated with each server, go to your browser and type something like (substitute the URL of the server whose WSDL you want to see):

http://localhost:8083/display?wsdl

Listing 18.5 Publisher for Servers

```java
package marina;

import javax.xml.ws.Endpoint;

import marina.HelloWorld;
import marina.displayitall;

//Endpoint publisher
public class marinaPublisher{

    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8083/display", new displayitall());
        Endpoint.publish("http://localhost:8084/iddisplay", new iddisplayitall());
        Endpoint.publish("http://localhost:8085/create", new create());
    }
}
```
Let’s look first at Listing 18.6, the displayitall server. Using JDBC, it connects to the database then creates a query that requests information about all the major items in a record (name, model, slip, begin_date, end_date, id). It then iterates through the results of the query, looking at the values for each major item (name, model, slip, begin_date, end_date). It creates a string that contains the values for each major item, separated by tab characters. Note that it does not write the id value into the string.

It then closes the connection.

LISTING 18.6 Display All Records Server

```java
package marina;

import java.io.*;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.*;
import java.sql.SQLException;
import java.util.Calendar;
import java.lang.Class;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.xml.ws.soap.Addressing;

@WebService
@WebMethod
public class displayitall {

    // set name of the JDBC driver
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    // set database URL
    static final String DB_URL = "jdbc:mysql://localhost/thething";

    // set your database username and password
    static final String myUN = "myownstuff";
    static final String myPW = "samsung";

    @WebMethod
    public String display(String name) {
        return "Endpoint.publish("http://localhost:8086/delete", new delete());
        Endpoint.publish("http://localhost:8087/requestmaintenance", new requestmaintenance());
    }
}
```
```java
StringBuffer myout = new StringBuffer("\n");

// Following output will show up in Glassfish log file
// because System.out is standard output it will show up
// on the console
System.out.println("Inside receivedGet method");

Connection conn = null;
Statement stmt = null;
try{
    // Register the JDBC driver
    Class.forName("com.mysql.jdbc.Driver");

    // Connect to the database
    System.out.println("Attempting to connect to database. Trying hard!");
    conn = DriverManager.getConnection(DB_URL,myUN,myPW);
    // Create a query to the database
    String mysql_Query;
    System.out.println("Attempting to query database");
    stmt = conn.createStatement();
    mysql_Query = "SELECT name,model,slip,begin_date,end_date,id FROM marina";
    ResultSet rs = stmt.executeQuery(mysql_Query);
    myout.append("Display All");
    myout.append("\n");

    // Get all data in the database
    while(rs.next()){
        // grab the data in order by data column name
        String thename = rs.getString("name");
        String themodel = rs.getString("model");
        int theslip = rs.getInt("slip");
        String thebegindate = rs.getString("begin_date");
        Date theenddate = rs.getDate("end_date");
        Calendar mycalendar=Calendar.GetInstance();
        mycalendar.setTime(thenddate);
        int month = mycalendar.get(Calendar.MONTH);
        int day = mycalendar.get(Calendar.DAY_OF_MONTH);
        int year = mycalendar.get(Calendar.YEAR);
        int theid = rs.getInt("id");

        // write values to glassfish console
        System.out.print("Name: " + thename);
        System.out.print("Model: " + themodel);
        System.out.println("Slip: " + theslip);
        System.out.println("begin_date: " + thebegindate);
        System.out.println("end_date: " + theenddate);
    }
}
```
System.out.println("end_date: month "+ month);
System.out.println("end_date: day" + day);
System.out.println("end_date: year" + year);
System.out.println("id" + theid);

myout.append("name:"+thename);
myout.append("@" );
myout.append("model:"+themodel);
myout.append("@" );
myout.append("slip:"+theslip);
myout.append("@" );
myout.append("begin_date:"+thebegindate);
myout.append("@" );
myout.append("end_date:"+theenddate);
myout.append("n");

} // finish up when done
rs.close();
stmt.close();
conn.close();
}

} // JDBC problems
sle.printStackTrace();
}

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

} // “finally” shut everything down
try{
if(stmt!=null)
stmt.close();
} catch(SQLException allse){
 // Ignore
}

} //end finally

System.out.println("Be seein’ y’all");

String thestring = myout.toString();

return thestring;

}
Next, let’s look at Listing 18.7, the iddisplayitall server. Using JDBC, it connects to the database then creates a query that requests information about all the major items in a record (name, model, slip, begin_date, end_date, id). It then iterates through the results of the query, looking at the values for each major item (name, model, slip, begin_date, end_date). It creates a string that contains the values for each major item, separated by tab characters.

Note that the only difference between the iddisplayitall server and the displayitall server is that the iddisplayitall server writes the id number into the string it returns, whereas the displayitall server does not.

It then closes the connection.

LISTING 18.7 Display All Records Including IDs Server

```java
package marina;

import java.io.*;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.*;
import java.sql.SQLException;
import java.util.Calendar;
import java.lang.Class;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.xml.ws.soap.Addressing;

@WebService
@WebMethod
public class iddisplayitall {

  // set name of the JDBC driver
  static final String JDBC_DRIVER = “com.mysql.jdbc.Driver”;

  // set database URL
  static final String DB_URL = “jdbc:mysql://localhost/thething”;

  // set your database username and password
  static final String myUN = “myownstuff”;
  static final String myPW = “samsung”;

  @WebMethod
  public String iddisplay(String name) {
    StringBuffer myout = new StringBuffer(“”);

    // Following output will show up in Glassfish log file
    // because System.out is standard output it will show up
    // on the console
    System.out.println(“Inside receivedGet method”);

    Connection conn = null;
    Statement stmt = null;
```
try{
    // Register the JDBC driver
    Class.forName(“com.mysql.jdbc.Driver”);

    // Connect to the database
    System.out.println(“Attempting to connect to database. Trying hard!”);
    conn = DriverManager.getConnection(DB_URL,myUN,myPW);

    // Create a query to the database
    String mysql_Query;
    System.out.println(“Attempting to query database”);
    stmt = conn.createStatement();
    mysql_Query = “SELECT name,model,slip,begin_date,end_date,id FROM marina”;
    ResultSet rs = stmt.executeQuery(mysql_Query);

    // Get all data in the database
    while(rs.next()){
        // grab the data in order by data column name
        String thename = rs.getString(“name”);
        String themodel = rs.getString(“model”);
        int theslip = rs.getInt(“slip”);
        String thebegindate = rs.getString(“begin_date”);
        Date theenddate = rs.getDate (“end_date”);
        Calendar mycalendar=Calendar.getInstance();
        mycalendar.setTime(thenddate);
        int month = mycalendar.get(Calendar.MONTH);
        int day = mycalendar.get(Calendar.DAY_OF_MONTH);
        int year = mycalendar.get(Calendar.YEAR);

        int theid = rs.getInt(“id”);

        // write values to glassfish console
        System.out.print(“Name: “ + thename);
        System.out.print(“Model: “ + themodel);
        System.out.print(“Slip: “ + theslip);
        System.out.print(“begin_date: “ + thebegindate);
        System.out.print(“end_date: “ + theenddate);
        System.out.print(“end_date: month “ + month);
        System.out.print(“end_date: day “ + day);
        System.out.print(“end_date: year “ + year);
        System.out.print(“id” + theid);

        myout.append(thename);
        myout.append(“	”);
        myout.append(themodel);
        myout.append(“	”);
        myout.append(theslip);
        myout.append(“	”);
        myout.append(thebegindate);
        myout.append(“	”);
        myout.append(thenddate);
        myout.append(“	”);
    }
Let’s look next at the Create method in Listing 18.8. It uses JDBC to connect to the database. Then it searches through the id numbers in the Marina database to find the maximum id number, and it adds one to this id number to create the id number for the new item that is being created.

Then Create makes a “query” (better called just a database statement) that includes an SQL INSERT command to stuff a new record into the database, with the new values that Create had passed to it as parameters. It executes this statement and if it is successful, then a new record has been added to the database.
LISTING 18.8  Create New Record Server

package marina;
import java.io.*;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.*;
import java.sql.SQLException;
import java.lang.Class;
import java.util.Calendar;
import java.util.Date;
import java.text.SimpleDateFormat;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.xml.ws.soap.Addressing;

@WebService
@Addressing(required = true)
public class create {

    // set name of the JDBC driver
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver"
    // set database URL
    static final String DB_URL = "jdbc:mysql://localhost/thething";

    // set your database username and password
    static final String myUN = "myownstuff";
    static final String myPW = "samsung";

    StringBuffer myout = new StringBuffer("");

    @WebMethod
    public String create(String namestr, String modelstr, String slipstr,
                           String begindatestr, String enddatestr ) {
        System.out.println("Inside create method");
        Connection conn = null;
        Statement stmt = null;
        try{
            // Register the JDBC driver
            Class.forName("com.mysql.jdbc.Driver");

            // Connect to the database
            System.out.println("Attempting to connect to database. Trying hard!");
            conn = DriverManager.getConnection(DB_URL, myUN, myPW);

            // Create a query to the database
            System.out.println("Attempting to query database");

            /*
             *  Here is where you would set up your query according to what 
             *  you wish to happen
             */

            // Here you will run the query
            // stmt.executeQuery("SQL Query Here");

            myout.append("It worked");
            return myout.toString();
        }
        catch(Exception e){
            myout.append("Error. Here.");
            return myout.toString();
        }
    }
}
// set the id to biggest id number plus one
// to do this we must first read the ids from the database
String mysql_Query;
stmt = conn.createStatement();
mysql_Query = "SELECT id FROM marina";
ResultSet rs = stmt.executeQuery(mysql_Query);

int theid=0;
int i=0;
// Get all data in the database
while(rs.next()){
    int someid = rs.getInt("id");
    if (someid>i)
        i=someid;
}
theid=i+1;

String mysql_QueryA;
mysql_QueryA = "INSERT INTO marina (name, model, slip, begin_date,end_date,id) VALUES (?,?,?,?,?)";

PreparedStatement statement = conn.prepareStatement(mysql_QueryA);
statement.setString(1,namestr);
statement.setString(2,modelstr);
int theslip = Integer.parseInt(slipstr);
statement.setInt(3,theslip);
SimpleDateFormat sdf = new SimpleDateFormat("MM-dd-yyyy");
Date thedate = sdf.parse(begindatestr);
java.sql.Date sqlbegin_date= new java.sql.Date(thedate.getTime());
statement.setDate(4,sqlbegin_date);

thedate = sdf.parse(enddatestr);
java.sql.Date sqlend_date= new java.sql.Date(thedate.getTime());
statement.setDate(5,sqlend_date);
// int thisid = Integer.parseInt(idstr);
statement.setInt(6,theid); // id field

int rows_inserted = statement.executeUpdate();
if (rows_inserted>0)
{
    System.out.println("inserted number of rows: "+rows_inserted);
}
// finish up when done
conn.close();
}catch(SQLException sqle){
    // JDBC problems
    sqle.printStackTrace();
}catch(Exception e){
e.printStackTrace();
}finally{
    // “finally” we shut everything down
    try{
        if(stmt!=null)
            stmt.close();
    }catch(SQLException allse){
        // ignore
    }
    try{
        if(conn!=null)
            conn.close();
    }catch(SQLException se){
        se.printStackTrace();
    }//end try
}//end finally

System.out.println("Be seein’ y’all");

// Write a small simple web page into the HTTP response
myout.append("Successfully created new row in database");

System.out.println("Be seein’ y’all");

String thestring = myout.toString();

return thestring;
}
}

The Delete server in Listing 18.9 allows you to delete a record based either on the slip number or the
id number, as both are supposed to be unique (slip is unique except when storing something like a
rowboat in the same slip as a sailboat). Whichever one (id or slip) you don’t want to use to specify a
record has a “−1” passed in for that value, because both id and slip are supposed to be positive num-
bers. Then it uses JDBC to connect to the database, and creates a statement that says to delete a
record from the Marina based on an id number, or else delete a record from the Marina based on
a slip number, depending on which was chosen earlier. Then it executes the statement and if the
delete was successful, it passes a cute little message back to the client to the effect that the delete
actually happened.
LISTING 18.9  Delete Record Server

package marina;
import java.io.*;
import java.io.*;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.*;
import java.sql.SQLException;
import java.lang.Class;
import java.util.Calendar;
import java.util.Date;
import java.text.SimpleDateFormat;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.xml.ws.soap.Addressing;
@WebService
@Addressing(required = true)
public class delete {

    // set name of the JDBC driver
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    // set database URL
    static final String DB_URL = "jdbc:mysql://localhost/thething";

    // set your database username and password
    static final String myUN = "myownstuff";
    static final String myPW = "samsung";

    StringBuffer myout = new StringBuffer("");

    @WebMethod
    public String delete(String slipstr, String idstr) {

        // pass in -1 to signal not using that item (slipstr or idstr)
        int whichslip;
        int whichid;

        whichid = 0;
        whichslip = 0;
        if (!slipstr.equals("-1") )
            {
                System.out.println( "got here a, slipstr is "+slipstr+" idstr is "+idstr);
                whichslip = Integer.parseInt(slipstr);
            }
else if ( !idstr.equals("-1") )
{
    System.out.println("got here b, slipstr is "+slipstr+" idstr is "+idstr);
    whichid =Integer.parseInt(idstr);
}

System.out.println("got here c, slipstr is "+slipstr+" idstr is "+idstr);
Connection conn = null;
try{
    // Register the JDBC driver
    Class.forName("com.mysql.jdbc.Driver");
    // Connect to the database
    System.out.println("Attempting to connect to database. Trying hard!");
    conn = DriverManager.getConnection(DB_URL,myUN,myPW);
    // Create a query to the database
    String mysql_stmt;
    System.out.println("Attempting to query database");
    if (whichslip !=0)
    {
        mysql_stmt = "DELETE FROM marina where slip=\"+whichslip;
    }
    else if (whichid !=0)
    {
        mysql_stmt = "DELETE FROM marina where id=\"+whichid;
    }
    else
    {
        mysql_stmt = "DELETE FROM marina where slip=0";
    // default to slip =0 which should result in no delete
    }
    PreparedStatement stmt = conn.prepareStatement(mysql_stmt);

    int rows_deleted = stmt.executeUpdate();
    if (rows_deleted > 0)
        System.out.println("Row was successfully deleted.");
    myout.append("\n");
    myout.append("Delete was performed");
    myout.append("\n");
    // finish up when done
    conn.close();
}
catch(SQLException sqle)
{
The requestmaintenance server is intended to be called asynchronously, if you look back at the problem statement in Section 1.5. However, in this implementation in JAX-WS, all the asynchronous behavior is in the client. All the requestmaintenance server does after it is called is return a string that has a message about which person is being assigned to do sailboat maintenance, and when.

LISTING 18.10 Request Maintenance Server

```java
package marina;
import java.io.*;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.xml.ws.soap.Addressing;

@WebService
@Addressing(required = true)
public class requestmaintenance {
    StringBuffer myout = new StringBuffer("");

    @WebMethod
    public String requestmaintenance() {
        try{
            if(conn!=null)
                conn.close();
        } catch(SQLException se) {
            se.printStackTrace();
        } //end catch
    } //end finally

    String thestring = myout.toString();
    return thestring;
}
```

The requestmaintenance server is intended to be called asynchronously, if you look back at the problem statement in Section 1.5. However, in this implementation in JAX-WS, all the asynchronous behavior is in the client. All the requestmaintenance server does after it is called is return a string that has a message about which person is being assigned to do sailboat maintenance, and when.
System.out.println("Be seein’ y’all");
myout.append("Your maintenance person will be Sabrina. She will be available Friday after 3pm");
String thestring = myout.toString();
myout.setLength(0);
return thestring;
}

18.2.2 MARINA CONSOLE CLIENTS
As mentioned earlier, the console clients in this section call code that was automatically generated using wsimport, accessing the WSDL of the running servers.

**LISTING 18.11 Display All Records Client**

```java
package marina;
import marina.displayitall;
import marina.DisplayitallService;

public class displayClient {
    public static void main(String[] args) {
        DisplayitallService thedisplay = new DisplayitallService();
        Displayitall myinterface = thedisplay.getDisplayitallPort();

        String response = myinterface.display(args[0]);
        System.out.println("response is "+response);
    }
}
```

In the case of each console client, the proxy class provided by wsimport is instantiated, and the appropriate method is called on this proxy.

As you can see, there’s not a lot of difference between most of the console clients. Except for the requestmaintenance clients, they vary only in which wsimport-generated proxy is instantiated, and the name of the method called on that proxy.

The requestmaintenance console client, as seen in Listing 18.15, works differently from the other console clients because it implements an asynchronous interface:

```java
Response<RequestmaintenanceResponse>
    response = myinterface.requestmaintenanceAsync();
```
LISTING 18.12 Display All Records including IDs Client

```java
package marina;

public class iddisplayClient {
    public static void main(String[] args) {
        IddisplayitallService thedisplay = new IddisplayitallService();
        Iddisplayitall myinterface = thedisplay.getIddisplayitallPort();
        String response = myinterface.iddisplay(args[0]);
        System.out.println("response is "+response);
    }
}
```

LISTING 18.13 Create New Record Client

```java
package marina;

public class createClient {
    public static void main(String[] args) {
        CreateService thedisplay = new CreateService();
        Create myinterface = thedisplay.getCreatePort();
        String response = myinterface.create(args[0], args[1], args[2], args[3], args[4]);
        System.out.println("response is "+response);
    }
}
```

LISTING 18.14 Delete Record Client

```java
package marina;

public class deleteClient {
    public static void main(String[] args) {
        DeleteService thedisplay = new DeleteService();
```
Delete myinterface = thedisplay.getDeletePort();

String response = myinterface.delete(args[0], args[1]);

System.out.println("response is "+response);
}
}

We previously discussed in Section 10.3.6 how to create an asynchronous WSDL, and how to cause wsimport to generate the asynchronous interface. This example polls “response.isDone()” to determine when the server has replied. When the response is ready, to get the response it does:

// get the response itself
RequestmaintenanceResponse myresponse = response.get();

Then to extract the string from inside the response, it does:

// retrieve the string from within the response
String tmpresponse=myresponse.getReturn();

LISTING 18.15 Request Maintenance Client

package marina;
import java.util.concurrent.Future;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.xml.bind.annotation.XmlSeeAlso;
import javax.xml.ws.Action;
import javax.xml.ws.AsyncHandler;
import javax.xml.ws.RequestWrapper;
import javax.xml.ws.Response;
import javax.xml.ws.ResponseWrapper;

//import thehello.HelloWorld;
//import thehello.HelloWorldService;

public class requestmaintenanceClient {
    public static void main(String[] args) {

RequestmaintenanceService myHelloWorld = new RequestmaintenanceService();

Requestmaintenance myinterface = myHelloWorld.getRequestmaintenancePort();

Response<RequestmaintenanceResponse> response = myinterface.requestmaintenanceAsync();

String thereseponse="";
try {
    int i;
i=0;
    while ( !response.isDone() ) {

        if (i==0) {
            //do only once
            String thename= args[0];
            thereseponse="Hi, "+thename+" \n";
            System.out.println("doing useful work");
        }
        i=i+1;
    }

    // get the response itself
    RequestmaintenanceResponse myresponse = response.get();

    // retrieve the string from within the response
    String tmpresponse=myresponse.getReturn();
    thereseponse = thereseponse+tmpresponse;
}
catch (InterruptedException e) {
e.printStackTrace();
}
catch (Exception e) {
e.printStackTrace();
}

System.out.println(thereseponse );
}

18.2.3 Java Swing Marina GUI – JAX-WS

As mentioned earlier, the graphical user interface client used here is based on Java swing, and the GUI related interface is the same as that previously seen in Section 18.1. However, the client code employed in the JAX-WS GUI interface calls code that was automatically generated using wsimport. The client calls here are about the same as those previously seen in the various console clients, just encapsulated inside methods that are called from within the GUI code.
Since the GUI related code is the same, the user screens seen will be the same as with EJBs. Some specifics about Java Swing were previously covered in Section 17.1.4. If you recall from that section, buttons are set by creating a JButton, then setting an Action Command, and an associated ActionListener to handle the processing required by the button. To begin with, look at Figure 18.1. This is the main page for this GUI; it is called the “manage” page and is handled by class manage in Listing 18.16. The first button, b1, of type JButton is established as follows:

```java
b1=new JButton("Display with ids");
b1.setBounds(100,50,500,50);
b1.setActionCommand("display");
b1.addActionListener(this);
```

When we call “new JButton”, the name we pass, in this case “Display with ids,” will be shown on the screen as the name of the button. See Figure 18.1.

Note the setActionCommand in the case of each button. We set the action command to “display” for the Display with ids button and we set it to “create” for the Add Sailor/Sailboat to Database button, etc.. We use this in the actionPerformed method to figure out which button was pushed, as follows:

```java
if ("display".equals(e.getActionCommand())
if ("create".equals(e.getActionCommand() ) )
```

Then the appropriate class to handle each separate button is instantiated. The iddisplayGUI class is instantiated to handle the “display” action command, that is, to display all the records in the database; see Figure 18.2. The createGUI class is instantiated to handle the “create” action command, that is, to create a new record for the database, see Figure 18.3. The details_and_deleteGUI class is instantiated to handle the “details” action command, that is, to show the details of a single record or else delete a single record in the database, see Figure 18.4. The requestmaintenance class is instantiated to handle the “requestmaintenance” action command, that is, to handle requesting a person to perform maintenance; see Figure 18.5.

Note that as each new display is selected, the current Manage screen is set to not visible:

```java
this.setVisible(false);
```

And after the new display is created (after its appropriate class is instantiated) that new display is set visible:

```java
mydisplayB.setVisible(true);
```

**LISTING 18.16 Java Swing Marina GUI – JAX-WS**

```java
package marina;
import javax.swing.*;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
```

```java
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
```

```java
import javax.xml.ws.BindingProvider;
import javax.xml.ws.handler.MessageContext;
import java.util.Map;
```

```java
package marina;
```

```java
import java.util.ArrayList;
import java.util.List;
```

```java
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
```

```java
public class Manage extends JFrame {
```

```java
public Manage() {
    super("Manage");
    this.setVisible(true);
```

```java
public void start() {
    this.setVisible(true);
```

```java
private void manageGUI() {
    this.setVisible(true);
```

```java
private void iddisplayGUI() {
    this.setVisible(true);
```

```java
private void createGUI() {
    this.setVisible(true);
```

```java
private void details_and_deleteGUI() {
    this.setVisible(true);
```

```java
private void requestmaintenance() {
    this.setVisible(true);
```

```java
```
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableCellRenderer;
import java.awt.*;
import java.awt.event.*;
import java.awt.event.ActionListener;
import java.io.*;
import java.util.concurrent.Future;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
//import javax.xml.bind.annotation.XmlSeeAlso;
import javax.xml.ws.Action;
import javax.xml.ws.AsyncHandler;
import javax.xml.ws.RequestWrapper;
import javax.xml.ws.Response;
import javax.xml.ws.ResponseWrapper;

public class manage {

    static select_the_display the_display;
    static String data[][] = new String [][] {
        {"name","boat model","0","01-23-1979","01-23-1979","1"},
        {
            "", "", "", "", "", ""},
        {
            "", "", "", "", "", ""},
        {
            "", "", "", "", "", ""},
        {
            "", "", "", "", "", ""},
        {
            "", "", "", "", "", ""},
        {
            "", "", "", "", "", ""},
        {
            "", "", "", "", "", ""},
        {
            "", "", "", "", "", ""},
        {
            "", "", "", "", "", ""},
        {
            "", "", "", "", "", ""},
        {
            "", "", "", "", "", ""},
        {
            "", "", "", "", "", ""},
        {
            "", "", "", "", "", ""},
        {
            "", "", "", "", "", ""},
        {
            "", "", "", "", "", ""},
        {
            "", "", "", "", "", ""},
    };
    static String columnHeaders[] = new String []
    {
        "name","model","slip","begin_date","end_date","id"};
public static class iddisplayGUI extends JFrame implements ActionListener {

    class myRenderer implements TableCellRenderer {

        // override getTableCellRendererComponent. It return
        // component used to draw the cell. So you can make that
        // component be a JTextArea
        public Component getTableCellRendererComponent(JTable table, Object thevalue, boolean
                isSelected, boolean hasFocus, int row, int column) {
            JTextArea myTextArea = new JTextArea();
            if (theValue != null) myTextArea.setText(GetValue.toString());
            return myTextArea;
        }
    } // end nested class myRenderer

    JTable mytable = new JTable(data, columnHeaders);
    static int maxtablesize = 0;

    String displayed_text = "";

    JButton homebutton, updatebutton;

    public iddisplayGUI() // constructor
    {
        Container container = getContentPane();

        // FlowLayout puts GUI items in order
        container.setLayout(new FlowLayout());

        // home Button
        homebutton = new JButton("Home");
        homebutton.setActionCommand("home");
        homebutton.addActionListener(this);
        container.add(homebutton);

        mytable.setDefaultRenderer(Object.class, new myRenderer());
        mytable.getTableHeader().setFont(new Font("Sans Serif", Font.BOLD, 10));
        mytable.getTableHeader().setBackground(Color.GREEN);
        container.add(new JScrollPane(mytable));

        // Update Button
        updatebutton = new JButton("Update");
        updatebutton.setActionCommand("update");
        updatebutton.addActionListener(this);
        container.add(updatebutton);

        setSize(1000, 300);
    } // end of constructor
public void actionPerformed(ActionEvent e) {
    if ("update".equals(e.getActionCommand())) {
        String response=call_the_iddisplay();
        extract_info_from_the_iddisplay(response);
    }
    else if ("home".equals(e.getActionCommand())) {
        this.setVisible(false);
        the_display.setVisible(true);
    }
} // end actionlistener

void extract_info_from_the_iddisplay(String mystring) {
    try {
        StringReader strreader = new StringReader(mystring);
        BufferedReader br = new BufferedReader(strreader);
        String currentline;
        int i=0;
        while ((currentline = br.readLine()) != null) {
            if (i>maxtablesize) // update the max table size
                maxtablesize=i;
            String [] items = currentline.split("\t");
            for (int j=0; j<items.length; j++)
                mytable.setValueAt(items[j], i, j);
            i++;
        }
        // Clear out the remainder of the table
        for (int k=i-1; k<maxtablesize; k++)
            for(int j=0; j<6; j++)
            {
mytable.setValueAt("", i, j);
}
}

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

} // end extract info from the display

String call_the_iddisplay()
{

IddisplayitallService thedisplay = new IddisplayitallService();

// generated by wsimport as an interface class
Iddisplayitall myinterface = thedisplay.getIddisplayitallPort();

String response = myinterface.iddisplay("noname");

System.out.println("response is "+response);

return response;
}

} // end call_the_iddisplay

} // end iddisplayGUI class

public static class createGUI extends JFrame implements ActionListener {
class myRenderer implements TableCellRenderer {

    // override getTableCellRendererComponent. It return
    // component used to draw the cell. So you can make that
    // component be a JTextArea
    public Component getTableCellRendererComponent(JTable table,
           Object thevalue, boolean isSelected,
           boolean hasFocus, int row, int column) {

           JTextArea myTextArea = new JTextArea();

           return myTextArea;
    }

} // end myRenderer class
if (thevalue != null)
    myTextArea.setText(thevalue.toString());

    return myTextArea;
  }
} // end nested class myRenderer

static String data[][] = new String [][] {
    {"name","boat model","0","01-23-1979","01-23-1979","1"},
};
static String columnHeaders[] = new String []
    {"name","model","slip","begin_date","end_date","id"};
JTable mytable = new JTable(data,columnHeaders);

String displayed_text = "";

JButton homebutton, updatebutton;

public createGUI() // constructor
{
    Container container = getContentPane();

    // FlowLayout puts GUI items in order
    container.setLayout(new FlowLayout());

    // Home Button
    homebutton= new JButton("Home");
    homebutton.setActionCommand("home");
    homebutton.addActionListener(this);
    container.add(homebutton);
    mytable.setDefaultRenderer(Object.class, new myRenderer());
    mytable.getTableHeader().setFont( new Font("Sans Serif",Font.BOLD,10) );
    mytable.getTableHeader().setBackground(Color.GREEN);
    container.add( new JScrollPane(mytable) );

    // Update Button
    updatebutton= new JButton("Update");
    updatebutton.setActionCommand("update");
    updatebutton.addActionListener(this);
    container.add(updatebutton);
    setSize(1000, 300);
}
//end of constructor

public void actionPerformed(ActionEvent e)
{
    if ("update".equals(e.getActionCommand()) ) {

String response = call_the_create();

else if ("home".equals(e.getActionCommand())) {
    this.setVisible(false);
    the_display.setVisible(true);
}

} // end actionlistener

String call_the_create() {
    CreateService thedisplay = new CreateService();
    Create myinterface = thedisplay.getCreatePort();

    String item0 = (mytable.getValueAt(0,0)).toString();
    String item1 = (mytable.getValueAt(0,1)).toString();
    String item2 = (mytable.getValueAt(0,2)).toString();
    String item3 = (mytable.getValueAt(0,3)).toString();
    String item4 = (mytable.getValueAt(0,4)).toString();
    String item5 = (mytable.getValueAt(0,5)).toString();
    String response = myinterface.create(item0, item1, item2, item3, item4);

    return response;
}

} // end call_the_create

} // end createGUI class

public static class details_and_deleteGUI extends JFrame implements ActionListener {

    class myRenderer implements TableCellRenderer {
        // override getTableCellRendererComponent. It return
        // component used to draw the cell. So you can make that
        // component be a JTextArea
        public Component getTableCellRendererComponent(JTable table, Object thevalue,
                                                       boolean isSelected, boolean hasFocus,
                                                       int row, int column) {
            JTextArea myTextArea = new JTextArea();
            if (thevalue != null)
                myTextArea.setText(thethevalue.toString());
            return myTextArea;
        }
    } // end nested class myRenderer

    static String data[][] = new String[][] {
        {"name","boat model","0","01-23-1979","01-23-1979","1"},
    };
}
static String columnHeaders[] = new String [] {
    "name", "model", "slip", "begin_date", "end_date", "id"};

JTable mytable = new JTable(data, columnHeaders);

String displayed_text = "";

JButton homebutton, updatebutton, deletebutton;

public details_and_deleteGUI() // constructor
{
    Container container = getContentPane();

    // FlowLayout puts GUI items in order
    container.setLayout(new FlowLayout());

    // home Button
    homebutton= new JButton("Home");
    homebutton.setActionCommand("home");
    homebutton.addActionListener(this);
    container.add(homebutton);

    mytable.setDefaultRenderer(Object.class, new myRenderer());
    mytable.getTableHeader().setFont(new Font("Sans Serif",Font.BOLD,10));
    mytable.getTableHeader().setBackground(Color.GREEN);
    container.add( new JScrollPane(mytable) );

    // Update Button
    updatebutton= new JButton("Update");
    updatebutton.setActionCommand("update");
    updatebutton.addActionListener(this);
    container.add(updatebutton);

    // Delete Button
    deletebutton= new JButton("Delete");
    deletebutton.setActionCommand("delete");
    deletebutton.addActionListener(this);
    container.add(deletebutton);

    setSize(1000, 300);
} //end of constructor

public void actionPerformed(ActionEvent e)
{
    if ("update".equals(e.getActionCommand()) ) {
        String response=call_the_iddisplay();
        extract_info_from_the_iddisplay(response);
    }
}
else if ("home".equals(e.getActionCommand()) ) {
    this.setVisible(false);
    the_display.setVisible(true);
}
else if ("delete".equals(e.getActionCommand()) ) {
    delete_the_item();
}

} // end actionlistener

void extract_info_from_the_iddisplay(String mystring)
{
try {
    StringReader strreader = new StringReader(mystring);
    BufferedReader br = new BufferedReader(strreader);
    String currentline;
    int i=0;
    while ((currentline = br.readLine()) != null)
    {
        String [] items = currentline.split("\t");
        if ( items[5].equals(mytable.getValueAt(0,5) ) )
        {
            for (int j=0; j<items.length; j++)
            {
                mytable.setValueAt(items[j], 0, j);
            }
            i++;
        }
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
}

} // end extract info from the display

void delete_the_item()
{
    DeleteService thedisplay = new DeleteService();
Delete myinterface = thedisplay.getDeletePort();

    String firstvalue="-1";
    String secondvalue=mytable.getValueAt(0,5).toString();

    System.out.println("secondvalue is"+secondvalue);
    String response = myinterface.delete(firstvalue,secondvalue);

    System.out.println("response is "+response);
    for (int j=0; j<6; j++)
    {
        mytable.setValueAt(firstvalue, 0, j);
    }

} // delete_the_item
String call_the_iddisplay() {

    IddisplayitallService thedisplay = new IddisplayitallService();
    // generated by wsimport as an interface class
    Iddisplayitall myinterface = thedisplay.getIdisplayitallPort();

    String response = myinterface.iddisplay("noname");

    return response;

} // end call_the_iddisplay

} // end details class

public static class requestmaintenance extends JFrame implements ActionListener {

    JTextArea mytextA;

    String displayed_text = "";

    JButton refreshbutton, updatebutton;

    public requestmaintenance() {
        Container container = getContentPane();
        container.setLayout(new FlowLayout());
        mytextA = new JTextArea("name", 10, 30);
        container.add(mytextA);
    }
refreshbutton = new JButton("Request Maintenance");
refreshbutton.setBounds(100, 50, 100, 50);

refreshbutton.addActionListener(this);

container.add(refreshbutton);

setSize(1000, 1000);
setVisible(true);
}

public void actionPerformed(ActionEvent e) {
    // displayed_text=mytextA.getText();
    try {
        RequestmaintenanceService myHelloWorld = new RequestmaintenanceService();

        // this interface was generated by wsimport as an interface class
        Requestmaintenance myinterface = myHelloWorld.getRequestmaintenancePort();
        Response<RequestmaintenanceResponse> response = myinterface.requestmaintenanceAsync();
        String theresponse="";
        int i;
        i=0;
        while ( !response.isDone() ) {
            if (i==0) {
                //do only once
                System.out.println("doing useful work");
            }
            i=i+1;
        }

        String thename= (mytextA.getText()).toString();
        theresponse="Hi, " + thename + " ";

        // get the response itself
        RequestmaintenanceResponse myresponse = response.get();
        System.out.println("myresponse is " + myresponse);

        // retrieve the string from within the response
        String tmpresponse=myresponse.getReturn();
        System.out.println("tmpresponse is " + tmpresponse);
        theresponse = theresponse + tmpresponse;
        System.out.println("theresponse is " + theresponse);

        mytextA.setText(theresponse);
    }
    catch (Exception ex) {
    }
System.out.println("general exception");
System.out.println("stack trace is ");

} // end requestmaintenance class

static class select_the_display extends JFrame implements ActionListener{

JButton b,b1,b2,b3;

select_the_display(){

b1=new JButton("Display with ids");
b1.setBounds(100,50,500,50);
b1.setActionCommand("display");
b1.addActionListener(this);

b2=new JButton("Add Sailor/Sailboat to Database ");
b2.setBounds(100,150,500,50);
b2.setActionCommand("create");
b2.addActionListener(this);

b=new JButton("Choose Show Details or Delete Record in Database (enter id)");
b.setBounds(100,250,500,50);
b.setActionCommand("details");
b.addActionListener(this);

b3=new JButton("Request Maintenance");
b3.setBounds(100,350,500,50);
b3.setActionCommand("requestmaintenance");
b3.addActionListener(this);

// Add components to current container
add(b1);
add(b2);
add(b);
add(b3);

setSize(900,500);
setLayout(null);
setVisible(true);
}

public void actionPerformed(ActionEvent e){
if ("display".equals(e.getActionCommand() ) ) {
    this.setVisible(false);
    iddisplayGUI mydisplay = new iddisplayGUI();
}
mydisplay.setVisible(true);
mydisplay.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// JOptionPane.showMessageDialog(this,"OptionA");
} else if ("create".equals(e.getActionCommand()) ) {
    this.setVisible(false);
    createGUI mydisplayA = new createGUI();
    mydisplayA.setVisible(true);
    mydisplayA.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
else if ("details".equals(e.getActionCommand()) ) {
    this.setVisible(false);
    details_and_deleteGUI mydisplayB = new details_and_deleteGUI();
    mydisplayB.setVisible(true);
    mydisplayB.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
else if ("requestmaintenance".equals(e.getActionCommand()) ) {
    this.setVisible(false);
    requestmaintenance mydisplayC = new requestmaintenance();
    mydisplayC.setVisible(true);
    mydisplayC.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String args[])
{
    the_display= new select_the_display();
}
} // end manage

As we said before, the iddisplayGUI class is instantiated to handle the “display” action command, that is, to display all the records in the database. On Figure 18.2 there is a Home button and an Update button. The Home button takes you back to the Manage screen and the Update button reads all the records from the database (at least, all that will fit in the table on the screen).
This table on the screen is created as follows. First column headers for the table are defined, and an array describing the table contents is defined as follows:

```java
static String data[][] = new String [][] {
    {"name", "boat model", "0", "01-23-1979", "01-23-1979", "1"},
    {...},
    {...},
    {...},
    {...},
    {...},
    {...}
};

static String columnHeaders[] = new String [] {
    "name", "model", "slip", "begin_date", "end_date", "id"};

Then the table is created using the following statement:

JTable mytable = new JTable(data, columnHeaders);

Then a new (nested) “renderer” class is defined, myRenderer, which implements TableCellRenderer. What this new myRenderer class does is to set the individual items in a table to be TextAreas, where data can be displayed and entered (otherwise these items would have been just display items). The table which was just created (mytable) is set to use this new myRenderer class by using the following statement:

mytable.setDefaultRenderer(Object.class, new myRenderer());

To pretty things up, then mytable has its header fonts set:

mytable.getTableHeader().setFont( new Font("Sans Serif", Font.BOLD, 10) );

And then its header has its background color set:

mytable.getTableHeader().setBackground(Color.GREEN);

And finally the table is added to the current container. (The container is a pane that is employing a FlowLayout. We previously saw FlowLayout for Swing GUIs in Section 17.1.4):

container.add( new JScrollPane(mytable) );

Then when the update button is pushed, the actionPerformed handler runs the code associated with
the “update” action command. This results in calling the “call_the_iddisplay()” method, which
connects to the iddisplayitall server, which then performs the appropriate database access. Then the
“extract_info_from_the_iddisplay” method is called, which uses the tab characters in the iddisplayitall
response to break the response up into all the separate values, one for each column, and these are
written into mytable (on the screen) as follows:

```java
String [] items = currentline.split("\t");
for (int j=0; j<items.length; j++)
{
    mytable.setValueAt(items[j], i, j);
}
i++;
} // end while
```

The createGUI class is instantiated to handle the “create” action command, that is, to create a new
record for the database. The table shown in Figure 18.3 works similarly to the table in Figure 18.2
from the iddisplayGUI class except that only one row of the class is shown, the row that will hold
the data that will be written to the database. When the update button shown in Figure 18.3 is pushed,
a connection will be created to the Create server that will be established, and the Create server per-
forms the appropriate update of the database. The data passed to the Create server to be used in creat-
ing the new record will be extracted from the table as follows:

```java
String item0 = (mytable.getValueAt(0,0)).toString();
String item1 = (mytable.getValueAt(0,1)).toString();
etc.
```

The details_and_deleteGUI class is instantiated to handle the “details” action command, that is, to
show the details of a single record or else delete a single record in the database. From Figure 18.4
you can see that the single row of the table shown is implemented in the GUI in the same way that
it worked for Create (and similarly to how it worked for Display All Items, except with one row
instead of several). Here, “update” results in a call to the iddisplayitall server to display a single
row, whereas “delete” results in a call to the Delete server to delete the item based on the id number
shown in the table. It extracts the id to be passed to the Delete server as follows:

```java
String secondvalue=mytable.getValueAt(0,5).toString();
```

Note that this code always assumes it’s going to be deleting based on an id number; it sets the slip
value passed to the delete to a “–1”.

The requestmaintenance class is instantiated to handle the “requestmaintenance” action command,
that is, to handle requesting a person to perform maintenance. The requestmaintenance class is a much
simpler GUI than the others, as it does not use a table; see Figure 18.5. It just sets up a TextArea to
collect the name from the user, and employs a button that, when pressed, results in a call to reques-
tmaintenance on the requestmaintenance server.

### 18.3 SAILBOAT MARINA PROJECT IN JAVASCRIPT/PHP USING AJAX

In this section we look at an implementation of the Sailboat Marina project previously described in
Section 1.5.

Before running the code in this section, you must implement a MySQL database named “thething”
containing a table named “marina” as described in Section 4.11.1.1.
18.3.1 **MARINA SERVERS**

The servers in this section are all written in PHP, and using the MySQLi functions we saw previously in Section 4.11.1.3 to access the MySQL database.

When these servers were tested, they were deployed on Linux using the Apache HTTP server. So this is an example of a LAMP web solution; see Section 4.11.1. All the files, including both the PHP files and the .html files were placed in the Apache web directory and accessed via localhost.

In each of these servers, there is a function named “connect_to_db()”. It uses a predefined username and password to create a connection to the MySQL database.

**LISTING 18.17 Display All Records Server**

```php
<?php

function connect_to_db()
{
    $username = "myownstuff";
    $password = "mypassword";
    $dbname = "thething";

    // Create connection
    $conn = new mysqli($servername, $username, $password, $dbname);
    // Check connection
    if ($conn->connect_error) {
        die("Connection failed: ". $conn->connect_error);
    }

    return $conn;
}

// Here begins the main program
$conn=connect_to_db();
$sql = "SELECT name, model,slip, begin_date, end_date, id FROM marina";
$result = $conn->query($sql);

$initialfile = fopen("/var/www/html/initialformat.html", "r") or die("missing initial format data");

// read html format and echo it
while(!feof($initialfile)) {
    $current_line=fgets($initialfile);

    // don’t print the id header
    if (strpos($current_line,"Id") == false)
        echo $current_line;
}
fclose($initialfile);

if ($result->num_rows > 0) {
```
The Display All Records server shown in Listing 18.17 first connects to the database. Then it performs a MySQLi query that extracts all instances of name, model, slip, begin_date, end_date, and id from the database.

Then, to create the output web page, it reads the initial boilerplate and the necessary code to set up an output table from the file that is shown in Listing 18.18.

In Listing 18.18, note that it is absolutely not an error that there is no terminating </table> statement, </body> statement, or </html> statement. The intent is for the current PHP server to read the code in Listing 18.18 to start up the web page, then write in the data (appropriate for the current server), then read in the code in Listing 18.19 to finish up the web page. So the ending statements are in Listing 18.19.

The data that is written (after the code from Listing 18.18 is read and output, and before the code from Listing 18.19 is read and output) consists of name, model, slip, begin date, and end date. Note that it does not include the id field.

LISTING 18.18 InitialFormat

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="mystyle.css"/>
  </head>
```

```php
// output data of each row
while($row = $result->fetch_assoc()) {
    print "<tr>
    print "<td>".$row["name"]."</td>
    print "<td>".$row["model"]."</td>
    print "<td>".$row["slip"]."</td>
    print "<td>".$row["begin_date"]."</td>
    print "<td>".$row["end_date"]."</td>
    print "</tr>
}
} else {
    echo “0 results”;
}

$finalfile = fopen("finalformat.html", "r") or die("missing final format data");

// read html format and echo it
while(!feof($finalfile)) {
    echo fgets($finalfile);
}
fclose($finalfile);
$conn->close();
?>
```
The server in Listing 18.20 shows all records in the database. It works exactly the same as the server in Listing 18.17, except that when it creates the output HTML table, it writes the id (in Listing 18.17); the id was read from the database along with all the other records, but was not written to the output HTML table.

```php
<?php

function connect_to_db()
{
    $servername = "myownstuff";
    $password = "mypassword";
    $dbname = "thething";

    // Create connection
    $conn = new mysqli($servername, $username, $password, $dbname);

    // Check connection
    if ($conn->connect_error) {
        die("Connection failed: ". $conn->connect_error);
    }
    return $conn;
}
```
// Here begins the main program
$conn = connect_to_db();

$sql = "SELECT name, model, slip, begin_date, end_date, id FROM marina";
$result = $conn->query($sql);

$initialfile = fopen("/var/www/html/initialformat.html", "r") or die("missing initial format data");

// read html format and echo it
while(!feof($initialfile)) {
    echo fgets($initialfile);
}
fclose($initialfile);

if ($result->num_rows > 0) {
    // output data of each row
    while($row = $result->fetch_assoc()) {
        print "<tr>");
        print "<td>".$row["name"]."</td>");
        print "<td>".$row["model"]."</td>");
        print "<td>".$row["slip"]."</td>");
        print "<td>".$row["begin_date"]."</td>");
        print "<td>".$row["end_date"]."</td>");
        print "<td>".$row["id"]."</td>");
        print "</tr>");
    }
} else {
    echo "0 results";
}

$finalfile = fopen("finalformat.html", "r") or die("missing final format data");

// read html format and echo it
while(!feof($finalfile)) {
    echo fgets($finalfile);
}
fclose($finalfile);
$conn->close();
?>

The server in Listing 18.21 creates a new record in the Marina table in the thething database. After connecting to the database, it searches through all the records of the database to determine the maximum id number. Then it sets the id number of the new record to one plus the previous maximum id number. Finally, it performs a MySQLi insert command to add the new record to the database.

When creating a new date field, it is important to remember the PHP 32 bit integer date problem: the biggest possible date is in January 2038; larger dates will wrap around. Try entering dates beyond January 2038 and see what happens. ☺
LISTING 18.21  Create New Record Server

<?php

function connect_to_db()
{
    $username = "myownstuff";
    $password = "mypassword";
    $dbname = "thething";

    // Create connection
    $conn = new mysqli($servername, $username, $password, $dbname);
    // Check connection
    if ($conn->connect_error) {
        die("Connection failed: ". $conn->connect_error);
    }

    return $conn;
}

// Here begins the main program
$conn=connect_to_db();

$name = $_REQUEST["name"];  
$model = $_REQUEST["model"];  
$slip = $_REQUEST["slip"];  
$begin_date = $_REQUEST["begin_date"];  
$end_date = $_REQUEST["end_date"];  

// change date format
$modified_begin_date = date("Y-m-d", strtotime($begin_date));  
$modified_end_date = date("Y-m-d", strtotime($end_date));  

// note: biggest possible date is in January 2038 due to 
// PHP 32 bit integer date problem, larger dates will wrap 
// around

// search for largest id number in the database
$sql = "SELECT id FROM marina";
$result = $conn->query($sql);

$i=0;
if ($result->num_rows > 0) {

    // output data of each row
    while($row = $result->fetch_assoc()) {
        $temp=$row["id"];
        if ($temp > $i)
        {
            // ...
$i = $temp;
}
}

$id = $i + 1;
$sql = "INSERT INTO marina (name, model, slip, begin_date, end_date, id) VALUES ('$name', '$model', '$slip', '$modified_begin_date', '$modified_end_date', '$id')";

if ($conn->query($sql) === TRUE) {
    $message = "New record created successfully";
    require_once 'success.php';
} else {
    echo "Error: " . $sql . "<br>
    " . $conn->error;
}

$conn->close();
?>

The server in Listing 18.22 can use either the id number or the slip number to choose a record to delete; it uses the "isset" statement to determine whether one or the other has a value.

If you remember from Section 1.5, there are cases where a slip in a marina might contain both a canoe and a sailboat, or maybe two canoes. ☺ However, if you use the slip to select a record, then you normally would be assuming that only one boat will be present in any slip. (However, note that the code works even if two boats share a slip, when it deletes by slip both boats will be deleted.)

**LISTING 18.22  Delete Record Server**

```php
<?php
function connect_to_db()
{
    $username = "myownstuff";
    $password = "mypassword";
    $dbname = "thething";

    // Create connection
    $conn = new mysqli($servername, $username, $password, $dbname);
    // Check connection
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }
    return $conn;
}

// Here begins the main program
$conn = connect_to_db();
```
The server in Listing 18.23 shows the details of a selected id number or a selected slip number. Note that it allows the use of either id or slip, and the code to select whether to use id or slip is very similar to the code we saw previously in Listing 18.22.

The details server in Listing 18.23, however, instead of deleting a single record, will allow you to show the details of multiple records, as follows:

```php
while($row = $result->fetch_assoc()) {
    // Print the details of the record
    print “<tr>”;
    print “<td>”. $row["name"] . “</td>”;
    print “<td>”. $row["model"] . “</td>”;
    print “<td>”. $row["slip"] . “</td>”;
    print “<td>”. $row["begin_date"] . “</td>”;
    print “<td>”. $row["end_date"] . “</td>”;
    print “<td>”. $row["id"] . “</td>”;
    print “</tr>”;
}
```

The server in Listing 18.23 shows the details of a selected id number or a selected slip number. Note that it allows the use of either id or slip, and the code to select whether to use id or slip is very similar to the code we saw previously in Listing 18.22.
Because of this while loop, the details server will show multiple records that have the same id. This isn’t supposed to happen but I didn’t do input checks because it cluttered up the code and made the technology hard to see. It will also show multiple boats in the same slip—whether you allow this or not is up to you! Or the manager of your sailboat marina.

**LISTING 18.23 Details Server**

```php
<?php

function connect_to_db()
{
    $username = "myownstuff";
    $password = "mypassword";
    $dbname = "thething";

    // Create connection
    $conn = new mysqli($servername, $username, $password, $dbname);
    // Check connection
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }

    return $conn;
}

// Here begins the main program
$conn=connect_to_db();

$slip = $_REQUEST["slip"];    
$id = $_REQUEST["id"];

$sql = "SELECT name, model,slip, begin_date, end_date, id FROM marina WHERE ";

// select using id if present else delete using slip
if ( isset($id) ) {
    // sql to select a record
    $sql = $sql."id=\"\".$id;
}
else if ( isset($slip) ) {
    // sql to select a record
    $sql = $sql."slip=\"\".$slip;
}
else {
    print "error input incorrect";
}
```
In Listing 18.24, we see a simple server to handle a request for maintenance. All it does is return a simple message that a maintenance person is assigned, and what time that person will be available.

If you recall from our previous Marina examples, and from the Sailboat Marina project statement in Section 1.5, requesting a maintenance person will normally be asynchronous. In this example, the request maintenance server is clearly synchronous. So the asynchronous behavior is in the web page that calls this server—in this case, we will be using JavaScript embedded inside the HTML web page. See Listing 18.32 later on. Note that this request maintenance server sleeps for 5 seconds in the middle of its processing. This is to simulate true asynchronous behavior, and will allow interesting results that exhibit the asynchronous behavior when we look at the request maintenance web page later on.
LISTING 18.24 Request Maintenance Server

```php
<?php
    $mydata = $_REQUEST['mydata'];
    sleep(5);
    print "<p>Hi, $mydata! Your maintenance person will be Priya. She will be available Friday after 3pm.</p>";
?>
```

18.3.2 MARINA WEB PAGES

The clients to access the servers from the previous section are all written in HTML with JavaScript and are meant to run in a standard browser.

The main web page for this Sailboat Marina application is shown in Listing 18.25, and what it looks like when it’s running in a browser is shown in Figure 18.6. In Listing 18.25, we can see that the two Display All Records servers (Display All Records without including ids, and Display All Records including ids) are called direct from inside the HTML, and the results are shown in Figure 18.7 and Figure 18.8:

- `<a href="displayitall.php">`
- `<a href="iddisplayitall.php">`

![Manage Web Page](http://localhost/manage.html)

**FIGURE 18.6** Manage Web Page.
To call the other servers, the main web page in Listing 18.25 calls different HTML web pages. These are shown later on in this section.

LISTING 18.25 Manage Web Page

```html
<!DOCTYPE html>
<html>
  
  <head>
    <link rel="stylesheet" type="text/css" href="mystylemanage.css">
  </head>

  <body>
    
    <h1>Have a Windy Day!</h1>

    <table>
      <thead>
        <tr>
          <th>Name</th>
          <th>Model</th>
          <th>Slip</th>
          <th>Begin Date</th>
          <th>End Date</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>Puffy</td>
          <td>boat model</td>
          <td>9</td>
          <td>1969-12-31</td>
          <td>1989-12-11</td>
        </tr>
        <tr>
          <td>Cupid</td>
          <td>boat model</td>
          <td>3</td>
          <td>1979-01-23</td>
          <td>1979-01-23</td>
        </tr>
        <tr>
          <td>TroubleDoggy</td>
          <td>boat model</td>
          <td>9</td>
          <td>1979-01-23</td>
          <td>1979-01-23</td>
        </tr>
      </tbody>
    </table>

    <p>FIGURE 18.7 Display All Records Web Page.</p>

    <table>
      <thead>
        <tr>
          <th>Name</th>
          <th>Model</th>
          <th>Slip</th>
          <th>Begin Date</th>
          <th>End Date</th>
          <th>Id</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>Puffy</td>
          <td>boat model</td>
          <td>9</td>
          <td>1969-12-31</td>
          <td>1989-12-11</td>
          <td>5</td>
        </tr>
        <tr>
          <td>Cupid</td>
          <td>boat model</td>
          <td>3</td>
          <td>1979-01-23</td>
          <td>1979-01-23</td>
          <td>6</td>
        </tr>
        <tr>
          <td>TroubleDoggy</td>
          <td>boat model</td>
          <td>9</td>
          <td>1979-01-23</td>
          <td>1979-01-23</td>
          <td>7</td>
        </tr>
      </tbody>
    </table>

    <p>FIGURE 18.8 Display All Records with IDs Web Page.</p>
```

To call the other servers, the main web page in Listing 18.25 calls different HTML web pages. These are shown later on in this section.
A Cascading Style Sheet (CSS stylesheet) called “mystylemanage.css” is used to format the HTML for display in the browser. This stylesheet is shown in Listing 18.26. As you can see, it sets h2 headers to a light seagreen color, centers them and puts them in bold. It does the same with h1 headers, except it makes h1 headers bigger than h2 headers (220% compared to 180%).

For the table shown in the web page, it sets the color of even numbered rows to white and the color of odd numbered rows to mintcream. Parenthetical comment: Don’t you love the way GUI displays and also paint merchants use these special names for colors? Now I’m craving to be eating mint ice cream while walking along the sea shore.

Compare all the stylesheet settings to what the web page looks like in Figure 18.6. Try running it yourself and varying the style (this code is all included in the code that comes with this textbook).

**LISTING 18.26 CSS for Manage Web Page**

```html
<br>
<h2>Manage the Marina</h2>
<table>
  <caption>Select an Action</caption>
  <tr><td><a href="displayitall.php">Show Marina</a></td></tr>
  <tr><td><a href="iddisplayitall.php">Show Marina with Internal Ids</a></td></tr>
  <tr><td><a href="create.html">Add New Sailor/Sailboat to Marina</a></td></tr>
  <tr><td><a href="delete.html">Delete Sailor/Sailboat from Marina</a></td></tr>
  <tr><td><a href="details.html">Give Details of Sailor/Sailboat from Marina</a></td></tr>
  <tr><td><a href="requestmaintenance.html">Request Maintenance Person</a></td></tr>
</table>
</body>
</html>
```
In Listing 18.27 we use an HTML form to enter data and then send data to the Create server. It sends the data using an HTTP GET. The stylesheet that describes how this web page looks is shown in Listing 18.28. Compare Listing 18.27 together with Listing 18.28 to how the web page actually looks in Figure 18.9.

After you run the Create, the results returned by the Create server are shown in Figure 18.10. Compare this to what the Create server code returns, see Figure 18.7.

**LISTING 18.27 Create Web Page**

```html
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" type="text/css" href="mystylecreate.css">
</head>
<body>
  <h1>Have a Windy Day!</h1>
  <h2>Add new Sailor/Boat to Marina</h2>
  <p id="myform">
    <form action="create.php" method="get">
      name<br>
      <input type="text" name="name" value="name">
    </form>
  </p>
</body>
</html>
```
<br>
model<br>
<input type="text" name="model" value="boat model" />
<br>
<br>
slip<br>
<input type="number" name="slip" value=0 />
<br>
<br>
begin_date<br>
<input type="date" name="begin_date" value="01/23/1979" />
<br>
<br>
end_date<br>
<input type="date" name="end_date" value="01/23/1979" />
<br>
<br>
<input type="submit" value="Create" style="font-weight:bold" />
</form>
<p align="center"><a href="manage.html">Return to Home</a></p>
</body>
</html>

FIGURE 18.9  Create Web Page.
In Listing 18.29, we select *either* an id number or a slip number to be used to select an item to delete. Then we pass either the slip or the id to the Delete server (see Listing 18.22). The stylesheet for this web page is shown in Listing 18.30. Note how the actual Delete Web Page looks; see Figure 18.11, and compare it to Listing 18.29 and Listing 18.30.

The results after running the Delete are shown in Figure 18.12.
LISTING 18.29 Delete Web Page

```html
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="mystyledelete.css">
</head>
<body>
<h1>Have a Windy Day!</h1>
<h2>Delete Sailor/Boat from Marina</h2>
<form id="form">
    <input type="text" name="id" placeholder="Enter ID">
    <button type="submit" id="deleteById">Delete by Id</button>
</form>
<form id="form">
    <input type="text" name="slip" placeholder="Enter Slip Number">
    <button type="submit" id="deleteBySlip">Delete by Slip</button>
</form>
</body>
</html>
```

FIGURE 18.11 Delete Web Page.

FIGURE 18.12 Results After Delete Web Page.

LISTING 18.29 Delete Web Page

```html
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="mystyledelete.css">
</head>
<body>
<h1>Have a Windy Day!</h1>
<h2>Delete Sailor/Boat from Marina</h2>
</body>
</html>
```
LISTING 18.30  CSS for Delete Web Page and Details Web Page

```html
<form action="delete.php" method="get">
  id<br>
  <input type="number" name="id" value=0>
  <br>
  <input type="submit" value="Delete by Id" style="font-weight:bold">
</form>
<br>
<br>
<form action="delete.php" method="get">
  slip<br>
  <input type="number" name="slip" value=0>
  <br>
  <input type="submit" value="Delete by Slip" style="font-weight:bold">
  <br>
  <br>
  <a id="tohome" href="manage.html">Return to Home</a></p>
</form>
</body>
</html>
```
The details web page code in Listing 18.31 is very similar to the Delete Web Page in Listing 18.29. Also, its stylesheet (see Listing 18.30) is the same as for the Delete Web Page. See Figure 18.13 for what this web page looks like, and see Figure 18.14 for what the details server returns after it has been called.

FIGURE 18.13  Details Web Page.

Have a Windy Day!
Details

<table>
<thead>
<tr>
<th>Id</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Return to Home

FIGURE 18.14  Results After Details Web Page.

The details web page code in Listing 18.31 is very similar to the Delete Web Page in Listing 18.29. Also, its stylesheet (see Listing 18.30) is the same as for the Delete Web Page. See Figure 18.13 for what this web page looks like, and see Figure 18.14 for what the details server returns after it has been called.

LISTING 18.31  Details Web Page

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="mystyledelete.css">
  </head>
  <body>
    <h1>Have a Windy Day!</h1>
    <h2>Details</h2>
    <br>
    Id
    <br>
    5
    <br>
    Details
    
    Return to Home
  
    Slip
    
    9
    
    Details
    
    Return to Home

  </body>
</html>
```

Sailboat Marina Project
Okay, we’re finally now at our asynchronous web client, that is, the request maintenance web page shown in Listing 18.32 (its stylesheet is shown in Listing 18.33). The web page in the browser is shown in Figure 18.15.

**Figure 18.15** Request Maintenance Web Page.

**Have a Windy Day!**

**RequestMaintenance**

We do all kinds of maintenance

The Best Sailboat Maintenance People on the Planet

Enter your name: Letha

Request Maintenance Person

Return to Home
When a name is entered in the HTML form and the submit button is pushed, a small JavaScript program is called (this JavaScript code is embedded within the <script> and </script> tags in the HTML itself).

The JavaScript code calls XMLHttpRequest to send an HTTP GET to the request maintenance server that we previously saw in Figure 18.10. We saw how to use XMLHttpRequest to do asynchronous operations back in Section 11.1.3. This works the same way. We set up a function that runs when onreadystatechange changes.

In this particular case, when the request maintenance server is called, a portion of the request maintenance web page called "that part" is set to an in-progress message using the document object model (DOM) accessed by JavaScript. You can see the result of this in Figure 18.16. There was time to see this in-progress message because the request maintenance server (see Listing 18.24) delays for 5 seconds to allow this to occur before sending back a message.

When the request has been successfully processed, a portion of the web page called "that part" will be set to the message that says a maintenance person has been scheduled. You can see the result of this in Figure 18.17.

LISTING 18.32 Request Maintenance Web Page

```html
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="mystylerequestmaintenance.css">
  </head>
  <body>
    <h1>Have a Windy Day!</h1>
  
  </body>
</html>
```
<h1 id="myHeader">Request Maintenance</h1>

We do all kinds of maintenance

The Best Sailboat Maintenance People on the Planet

<form action = "requestmaintenance.php"
    <p> Enter your name: <input type="text" name="myname" /></p>
    <button type = "button" onclick = "doAJAXstuff(this.form)"
        Request Maintenance Person
        </button>
        </p>
        </form>
        </p>
        </form>

<script type = "text/javascript">
    function doAJAXstuff(theform) {
        request = new XMLHttpRequest();
        request.open("GET","requestmaintenance.php?mydata="+theform.myname.value, true);
        request.onreadystatechange = function () {
            if (request.readyState == 4)
                my_callback();
            
            request.send();
            
            document.getElementById("thatpart").style.font = "bold 30px Times New Roman sans-serif";
            document.getElementById("thatpart").innerHTML = "Your Request is Being Processed";
        }
    
    function my_callback() {
        if (request.status == 200) {
            document.getElementById("thispart").innerHTML = request.responseText;
            document.getElementById("thispart").style.font = "italic bold 20px arial, serif";
            document.getElementById("thatpart").innerHTML = "The Best Sailboat Maintenance People on the Planet";
        }
    
</script>
else if (request.status=404) {
    alert("Server was not found: data returned is :\n"+request.responseText);
}
<p align=“center”><a href=“manage.html”>Return to Home</a> </p>
</body>
</html>

LISTING 18.33  CSS for Request Maintenance Web Page

```css
#thispart, #thatpart {
  color: lightseagreen;
  text-align: center;
  font-size: 150%;
}

a {
  color: lightseagreen;
}

h2 {
  color: lightseagreen;
  text-align: center;
  font-size: 150%;
  font-weight: bold;
}

h1 {
  color: lightseagreen;
  text-align: center;
  font-size: 200%;
  font-weight: bold;
}

form {
  color: lightseagreen;
  width: 300px;
  margin: 0 auto;
}
```
18.4 SAILBOAT MARINA PROJECT USING SERVLETS

In this section we look at an implementation of the Sailboat Marina project previously described in Section 1.5.

Before running the code in this section, you must implement a MySQL database named "thething" containing a table named "marina" as described in Section 4.11.1.1.

In these sections, we will be using the Java Database Connectivity (JDBC) libraries. We won’t go over the details of the connections as they are presented here, since we previously covered that in Section 4.11.1.2.

18.4.1 MARINA SERVERS

When the servers in this section were tested, they were placed in the same Web Archive (war) file, and deployed on GlassFish and accessed at localhost. All were at the same port number (8080). The servers were all deployed using @WebServlet instead of using a web.xml file. We’ll talk more about deployment later on.

All the servlets in this section are accessed using a doGet or a doPost. What this means, as we saw previously in Section 11.2.4, is that when an HTTP GET is sent to the servlet in question, the doGet routine is called, and when an HTTP POST is sent to the servlet in question, the doPost routine is called.

Let’s look first at Listing 18.34, the displayitall server. Using JDBC, it connects to the database then creates a query that requests information about all the major items in a record (name, model, slip, begin_date, end_date, id). It then iterates through the results of the query, looking at the values for each major item (name, model, slip, begin_date, end_date). It uses PrintWriter to write the values for each major item, separated by tab characters, to a text output stream. Note that it does not write the id value into the string.

After creating the web page, it then closes the connection.
In Listing 18.34, Listing 18.35, and Listing 18.36, to create the output web page, it reads the initial boilerplate and the necessary code to set up an output table from the file that was previously shown in Listing 18.18.

In Listing 18.18, note that it is absolutely not an error that there is no terminating </table> statement, </body> statement, or </html> statement. The intent is for the current PHP server to read the code in Listing 18.18 in first to start up the web page, then write in the data (appropriate for the current server), then read in the code in Listing 18.19 to finish up the web page. So the ending statements are in Listing 18.19.

**LISTING 18.34 Display All Records Server**

```java
package marina;
import java.io.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.*;
import java.sql.SQLException;
import java.util.Calendar;
@WebServlet ( urlPatterns="display"
)
public class displayitall extends HttpServlet {
    // set name of the JDBC driver
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver"
    // set database URL
    static final String DB_URL = "jdbc:mysql://localhost/thething";

    // set your database username and password
    static final String myUN = "myownstuff";
    static final String myPW = "mypassword";

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
        {
            // set the content type in the HTTP response
            response.setContentType("text/html");

            // Following output will show up in Glassfish log file
            // because System.out is standard output it will show up
            // on the console
            System.out.println("Inside displayitall in doGet method");

            Connection conn = null;
            Statement stmt = null;
```
try{
    // Register the JDBC driver
    Class.forName("com.mysql.jdbc.Driver");

    // Connect to the database
    System.out.println("Attempting to connect to database. Trying hard!");
    conn = DriverManager.getConnection(DB_URL,myUN,myPW);

    // Create a query to the database
    String mysql_Query;
    System.out.println("Attempting to query database");
    stmt = conn.createStatement();
    mysql_Query = "SELECT name,model,slip,begin_date,end_date,id FROM marina";
    ResultSet rs = stmt.executeQuery(mysql_Query);

    // Create the response web page
    PrintWriter myout = response.getWriter();

    // Read initial HTML information from file also in WEB-INF/classes/marina
    InputStream in =
        displayitall.class.getResourceAsStream("initialformatwithoutid.html");
    BufferedReader r = new BufferedReader(new InputStreamReader(in) );
    String line;
    line = r.readLine();
    while (line != null)
    {
        myout.print(line);
        line = r.readLine();
    }
    in.close();

    // Get all data in the database
    while(rs.next()){
        // grab the data in order by data column name
        String thename = rs.getString("name");
        String themodel = rs.getString("model");
        int theslip = rs.getInt("slip");
        String thebegindate = rs.getString("begin_date");
        Date theenddate = rs.getDate("end_date");
        Calendar mycalendar=Calendar.getInstance();
        mycalendar.setTime(theenddate);
        int month = mycalendar.get(Calendar.MONTH);
        int day = mycalendar.get(Calendar.DAY_OF_MONTH);
int year = mycalendar.get(Calendar.YEAR);

int theid = rs.getInt("id");

myout.print("<tr>");
myout.print("<td>name: "+thename);
myout.print("</td>");
myout.print("<td>model: "+themodel);
myout.print("</td>");
myout.print("<td>slip: "+theslip);
myout.print("</td>");
myout.print("<td>begin_date "+thebegindate);
myout.print("</td>");
myout.print("<td>end_date "+theenddate);
myout.print("</td>");
myout.print("</tr>");

} // finish up when done
rs.close();
stmt.close();
conn.close();

// close off the web page
// Read final HTML information from file also in WEB-INF/classes/marina
in = displayitall.class.getResourceAsStream("finalformat.html");
r = new BufferedReader(new InputStreamReader(in) );

line = r.readLine();
while (line != null)
{
    myout.print(line);
    line = r.readLine();
}
in.close();

}catch(SQLException sqle){
    // problems with JDBC access
    sqle.printStackTrace();
}catch(Exception e){
    e.printStackTrace();
}finally{
    // “finally” shut everything down
try{
    if(stmt!=null)
        stmt.close();
}catch(SQLException allse){
    // ignore
} //end try catch
try{
    if(conn!=null)
        conn.close();
}catch(SQLException se){
    se.printStackTrace();
} //end try catch
//end finally

System.out.println("Be seein’ y’all");
}

Note that the only difference between the iddisplayitall server in Listing 18.35 and the displayitall server in Listing 18.34 is that the iddisplayitall server writes the id number into the string it returns, whereas the displayitall server does not.

**LISTING 18.35 Display All Records including IDs Server**

```java
package marina;
import java.io.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.*;
import java.util.Calendar;

@WebServlet (
    urlPatterns="/iddisplay"
)
public class iddisplayitall extends HttpServlet {
    // set name of the JDBC driver
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver"
    // set database URL
    static final String DB_URL = "jdbc:mysql://localhost/thething";

    // set your database username and password
    static final String myUN = "myownstuff";
    static final String myPW = "mypassword";
```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // set the content type in the HTTP response
    response.setContentType("text/html");

    // Following output will show up in Glassfish log file
    // because System.out is standard output it will show up
    // on the console
    System.out.println("Inside iddisplayitall in doGet method");

    Connection conn = null;
    Statement stmt = null;
    try {
        // Register the JDBC driver
        Class.forName("com.mysql.jdbc.Driver");

        // Connect to the database
        System.out.println("Attempting to connect to database. Trying hard!");
        conn = DriverManager.getConnection(DB_URL, myUN, myPW);

        // Create a query to the database
        String mysql_Query;
        System.out.println("Attempting to query database");
        stmt = conn.createStatement();
        mysql_Query = "SELECT name,model,slip,begin_date,end_date,id FROM marina";
        ResultSet rs = stmt.executeQuery(mysql_Query);

        // Create the response web page
        PrintWriter myout = response.getWriter();
        // Read initial HTML information from file also in WEB-INF/classes/marina
        InputStream in = iddisplayitall.class.getResourceAsStream("initialformat.html");
        BufferedReader r = new BufferedReader(new InputStreamReader(in));
        String line;
        line = r.readLine();
        while (line != null) {
            myout.print(line);
            line = r.readLine();
        }
        in.close();

        // Get all data in the database
        while (rs.next()) {
            // Code to process each row of the result set
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
// grab the data in order by data column name
String thename = rs.getString("name");
String themodel = rs.getString("model");
int theslip = rs.getInt("slip");
String thebegindate = rs.getString("begin_date");
Date theenddate = rs.getDate("end_date");
int theid = rs.getInt("id");

Calendar mycalendar = Calendar.getInstance();
mycalendar.setTime(thenddate);
int month = mycalendar.get(Calendar.MONTH);
int day = mycalendar.get(Calendar.DAY_OF_MONTH);
int year = mycalendar.get(Calendar.YEAR);

myout.print("<tr>");
myout.print("<td>");
myout.print("name: "+thename);
myout.print("</td>");
myout.print("<td>");
myout.print("model: "+themodel);
myout.print("</td>");
myout.print("<td>");
myout.print("slip: "+theslip);
myout.print("</td>");
myout.print("<td>");
myout.print("begin_date "+thebegindate);
myout.print("</td>");
myout.print("<td>");
myout.print("end_date "+theenddate);
myout.print("</td>");
myout.print("<td>");
myout.print("id "+theid);
myout.print("</td>");
myout.print("</tr>");

} // finish up when done
rs.close();
stmt.close();
conn.close();

// close off the web page

// Read final HTML information from file also in WEB-INF/classes/marina
in = iddisplayitall.class.getResourceAsStream("finalformat.html");
r = new BufferedReader(new InputStreamReader(in));

line = r.readLine();
while (line != null) {
    myout.print(line);
    line = r.readLine();
}
in.close();
}
} catch(SQLException sqle){
    // problems with JDBC access
    sqle.printStackTrace();
}
} catch(Exception e){
    e.printStackTrace();
}
} finally{
    // “finally” shut everything down
    try{
        if(stmt!=null)
            stmt.close();
    } catch(SQLException allse){
        // ignore
    } // end try catch
    try{
        if(conn!=null)
            conn.close();
    } catch(SQLException se){
        se.printStackTrace();
    } //end try catch
}//end finally

System.out.println("Be seein’ y’all");
}

Let’s look next at the Create method in Listing 18.36. It uses JDBC to connect to the database. Then it searches through the id numbers in the Marina database to find the maximum id number, and it adds one to this id number to create the id number for the new item that is being created.

Then Create makes a “query” (better called just a database statement) that includes an SQL INSERT command to stuff a new record into the database, with the new values that Create had passed to it as parameters. It executes this statement and if it is successful then a new record has been added to the database. A web page that shows the newly created item is returned.

LISTING 18.36 Create New Record Server

```java
package marina;
import java.io.InputStream;
import java.io.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.*;
import javax.servlet.http.*;
```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.*;
import java.sql.SQLException;
import java.util.Calendar;
import java.util.Date;
import java.text.SimpleDateFormat;

@WebServlet (urlPatterns="/create")
public class create extends HttpServlet {
    // set name of the JDBC driver
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver"
    // set database URL
    static final String DB_URL = "jdbc:mysql://localhost/thething";

    // set your database username and password
    static final String myUN = "myownstuff";
    static final String myPW = "mypassword";

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        // Start creating the response
        PrintWriter myout = response.getWriter();

        // First, let’s get the parameters from the HTTP message
        String namestr,modelstr,slipstr,begindatestr,enddatestr;
        namestr=request.getParameter("name");
        modelstr=request.getParameter("model");
        slipstr=request.getParameter("slip");
        int theslip =Integer.parseInt(slipstr);
        System.out.println("slip is "+theslip);

        begindatestr=request.getParameter("begin_date");
        enddatestr=request.getParameter("end_date");

        // set the content type in the HTTP response
        response.setContentType("text/html");

        // Following output will show up in Glassfish log file
        // because System.out is standard output it will show up
        // on the console
        System.out.println("Inside create in doPost method");}
Connection conn = null;
Statement stmt = null;
try{
    // Register the JDBC driver
    Class.forName("com.mysql.jdbc.Driver");
    // Connect to the database
    System.out.println("Attempting to connect to database. Trying hard!");
    conn = DriverManager.getConnection(DB_URL,myUN,myPW);
    // Create a query to the database
    System.out.println("Attempting to query database");
    // set the id to biggest id number plus one
    // to do this we must first read the ids from the database
    String mysql_Query;
    stmt = conn.createStatement();
    mysql_Query = "SELECT id FROM marina";
    ResultSet rs = stmt.executeQuery(mysql_Query);

    int theid=0;
    int i=0;
    // Get all data in the database
    while(rs.next()){
        int someid = rs.getInt("id");
        if (someid>i)
            i=someid;
    }
    theid=i+1;
    // finish up when done
    rs.close();
    String mysql_QueryA;
    mysql_QueryA = "INSERT INTO marina (name, model, slip, begin_date,end_date,id) VALUES (?,?,?,?,?)";
    PreparedStatement statement = conn.prepareStatement(mysql_QueryA);
    statement.setString(1,namestr);
    statement.setString(2,modelstr);
    statement.setInt(3,theslip);
    SimpleDateFormat sdf = new SimpleDateFormat("MM-dd-yyyy");
    Date thedate = sdf.parse(begindatestr);
    java.sql.Date sqlbegin_date= new java.sql.Date(thedate.getTime());
    statement.setDate(4,sqlbegin_date);
    thedate = sdf.parse(enddatestr);
    java.sql.Date sqlend_date= new java.sql.Date(thedate.getTime());
statement.setDate(5, sqlend_date);

statement.setInt(6, theid); // id field

int rows_inserted = statement.executeUpdate();
if (rows_inserted > 0)
{
    System.out.println("inserted number of rows: "+rows_inserted);
}

// Create a query to the database
String mysql_QueryB;
System.out.println("Attempting to query database");
stmt = conn.createStatement();

mysql_QueryB = "SELECT name, model, slip, begin_date, end_date, id FROM marina where id="+theid;

ResultSet rsB = stmt.executeQuery(mysql_QueryB);

String aname="";
String amodel="";
int aslip=0;
String abegindate="";
Date aenddate = new Date();
int aid=0;

if (rsB.next())
{
    // grab the data in order by data column name
    aname = rsB.getString("name");
    amodel = rsB.getString("model");
    aslip = rsB.getInt("slip");
    abegindate = rsB.getString("begin_date");
    aenddate = rsB.getDate("end_date");
    aid = rsB.getInt("id");
}

rs.close();
stmt.close();
conn.close();

// finish up when done
conn.close();

// Read initial HTML information from file also in WEB-INF/classes/marina
InputStream in = create.class.getResourceAsStream("initialformat.html");
BufferedReader r = new BufferedReader(new InputStreamReader(in));
String line;
line = r.readLine();

while (line != null)
{
    myout.print(line);
    line = r.readLine();
}
in.close();

myout.print("<h2>");
myout.print("Successfully created new row in database:");
myout.print("</h2>");
myout.print("<tr>");
myout.print("<td>");
myout.print(aname);
myout.print("</td>");
myout.print("<td>");
myout.print(amodel);
myout.print("</td>");
myout.print("<td>");
myout.print(aslip);
myout.print("</td>");
myout.print("<td>");
myout.print(abegindate);
myout.print("</td>");
myout.print("<td>");
myout.print(aenddate);
myout.print("</td>");
myout.print("<td>");
myout.print(aid);
myout.print("</td>");
myout.print("</tr>");

// Read HTML information from file also in WEB-INF/classes/marina
in = create.class.getResourceAsStream("finalformat.html");
r = new BufferedReader(new InputStreamReader(in) );

line = r.readLine();

while (line != null)
{
    myout.print(line);
    line = r.readLine();
}
in.close();

}catch(SQLException sqle){
The Delete server in Listing 18.37 allows you to delete a record based either on the slip number or the id number, as both are supposed to be unique (slip is unique except when storing something like a rowboat in the same slip as a sailboat). Whichever one (id or slip) you don’t want to use to specify a record has a zero value passed in for that value. Then it uses JDBC to connect to the database, and creates a statement that says to delete a record from the Marina based on an id number, or else delete a record from the Marina based on a slip number, depending on which was chosen earlier. Then it executes the statement and if the delete was successful, it returns a small web page showing the item that was deleted. Note that the style of the web page was written to the web page at the time the data was written.

**Listing 18.37 Delete Record Server**

```java
package marina;
import java.io.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.*;
import java.util.Calendar;
@WebServlet (
    urlPatterns="/delete"
)
```
public class delete extends HttpServlet {
    // set name of the JDBC driver
    static final String JDBC_DRIVER = “com.mysql.jdbc.Driver”
    // set database URL
    static final String DB_URL = “jdbc:mysql://localhost/thething”;

    // set your database username and password
    static final String myUN = “myownstuff”;
    static final String myPW = “samsung”;

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        // Get the HTTP parameter
        String slipstr;
        String idstr;

        int whichslip;
        int whichid;

        whichid = 0;
        whichslip = 0;
        int whichitem=0;  // always default to 0
        if (request.getParameter(“slip”)!=null)
        {
            slipstr=request.getParameter(“slip”);
            whichslip =Integer.parseInt(slipstr);
        }
        else if (request.getParameter(“id”)!=null)
        {
            idstr=request.getParameter(“id”);
            whichid =Integer.parseInt(idstr);
        }

        // set the content type in the HTTP response
        response.setContentType(“text/html”);

        // Following output will show up in Glassfish log file
        // because System.out is standard output it will show up
        // on the console
        System.out.println(“Inside myservlet in doPost method”);

        Connection conn = null;
        try{

        }
// Register the JDBC driver
Class.forName("com.mysql.jdbc.Driver");

// Connect to the database
System.out.println("Attempting to connect to database. Trying hard!");
conn = DriverManager.getConnection(DB_URL, myUN, myPW);

// Create a query to the database
String mysql_stmt;
System.out.println("Attempting to query database");
if (whichslip != 0)
{
    mysql_stmt = "DELETE FROM marina where slip="+whichslip;
}
else if (whichid != 0)
{
    mysql_stmt = "DELETE FROM marina where id="+whichid;
}
else
{
    mysql_stmt = "DELETE FROM marina where slip=0"
    // default to slip =0 which should result in no delete
}
PreparedStatement stmt = conn.prepareStatement(mysql_stmt);

int rows_deleted = stmt.executeUpdate();

if (rows_deleted > 0)
    System.out.println("Row was successfully deleted.");

PrintWriter myout = response.getWriter();

myout.print("<!DOCTYPE html>");
myout.print("<html>");
myout.print("<head>");
myout.print("<style>");
myout.print("a { color:lightseagreen; text-align:center }"");
myout.print("h2 { color:lightseagreen; text-align:center; font-size:150%; font-weight: bold; }"");
myout.print("p { color:lightseagreen; text-align:center; }");
myout.print("</style>");
myout.print("</head>");
myout.print("<body>");
myout.print("<h2>Delete was performed"");
myout.print("<br>");
myout.print("</h2>");
myout.print("<br>");
myout.print("<p align="center">
  <a href="http://localhost:8080/marina/manage.html">Return to Home</a>
</p>");
myout.print("</body>");
myout.print("</html>");

// finish up when done
conn.close();

} catch(SQLException sqle) {
  // problems with JDBC access
  sqle.printStackTrace();
} catch(Exception e) {
  e.printStackTrace();
}

} finally {
  // “finally” shut everything down
  try{
    if(conn!=null)
      conn.close();
  } catch(SQLException se) {
    se.printStackTrace();
  } //end try catch
  //end finally

  System.out.println("Be seein’ y’all");
}

The Details server in Listing 18.38 works similarly to the Delete server in Listing 18.37 in the way that it accepts the parameters slip and id. After that, it works similarly to the “Display all records” including ids server in Listing 18.35 in the way it queries the database, except that it only requests the records that match the given id or slip (whichever was used). It then displays the data the same way the “Display all records” server did.

LISTING 18.38  Details of Record Server

package marina;
import java.io.InputStream;
import java.io.*;
import java.io.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.*;
import java.sql.SQLException;
import java.util.Calendar;

@WebServlet (urlPatterns="/details")
public class details extends HttpServlet {

    // set name of the JDBC driver
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver"
    // set database URL
    static final String DB_URL = "jdbc:mysql://localhost/thething";

    // set your database username and password
    static final String myUN = "myownstuff";
    static final String myPW = "samsung";

    public void doPost(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {

        // Get the HTTP parameter
        String slipstr;
        String idstr;

        int whichslip;
        int whichid;

        whichid = 0;
        whichslip = 0;
        int whichitem=0; // always default to 0
        if (request.getParameter("slip")!=null)
        {
            slipstr=request.getParameter("slip");
            whichslip =Integer.parseInt(slipstr);
        }
        else if (request.getParameter("id")!=null)
        {
            idstr=request.getParameter("id");
            whichid =Integer.parseInt(idstr);
        }

        // set the content type in the HTTP response
        response.setContentType("text/html");
// Following output will show up in Glassfish log file
// because System.out is standard output it will show up
// on the console
System.out.println("Inside myservlet in doPost method");

Connection conn = null;
Statement stmt = null;
try{
    // Register the JDBC driver
    Class.forName("com.mysql.jdbc.Driver");

    // Connect to the database
    System.out.println("Attempting to connect to database. Trying hard!");
    conn = DriverManager.getConnection(DB_URL,myUN,myPW);

    // Create a query to the database
    String mysql_Query;
    System.out.println("Attempting to query database");
    stmt = conn.createStatement();

    if (whichslip !=0)
    {
        mysql_Query = 
        "SELECT name,model,slip,begin_date,end_date,id 
        FROM marina where slip=":+whichslip;
    } else if (whichid !=0)
    {  
        mysql_Query = "SELECT name,model,slip,begin_date,end_date,id 
        FROM marina where id=":+whichid;
    } else { 
        mysql_Query = "SELECT name,model,slip,begin_date,end_date,id 
        FROM marina where slip=0" // default to slip =0 which should
        result in no output
    }
    
    ResultSet rs = stmt.executeQuery(mysql_Query);

    // Create the response web page
    PrintWriter myout = response.getWriter();

    // Get all data in the database
    while(rs.next()){
        // grab the data in order by data column name
        String thename = rs.getString("name");
        String themodel = rs.getString("model");
        int theslip = rs.getInt("slip");
        String thebegindate = rs.getString("begin_date");
        Date theenddate = rs.getDate("end_date");
Calendar mycalendar=Calendar.getInstance();
mycalendar.setTime(theenddate);
int month = mycalendar.get(Calendar.MONTH);
int day = mycalendar.get(Calendar.DAY_OF_MONTH);
int year = mycalendar.get(Calendar.YEAR);

int theid = rs.getInt("id");

// Read initial HTML information from file also in WEB-INF/classes/marina
InputStream in =
    details.class.getResourceAsStream("initialformat.html");
BufferedReader r = new BufferedReader(new InputStreamReader(in));
String line;
line = r.readLine();
while (line != null)
{
    myout.print(line);
    line = r.readLine();
}
in.close();

myout.print("<tr>");
myout.print("<td>");
myout.print(thename);
myout.print("</td>");
myout.print("<td>");
myout.print(themodel);
myout.print("</td>");
myout.print("<td>");
myout.print(theslip);
myout.print("</td>");
myout.print("<td>");
myout.print(thebegindate);
myout.print("</td>");
myout.print("<td>");
myout.print(theenddate);
myout.print("</td>");
myout.print("<td>");
myout.print(theid);
myout.print("</td>");
myout.print("</tr>");

// Read HTML information from file also in WEB-INF/classes/marina
in = details.class.getResourceAsStream("finalformat.html");
r = new BufferedReader(new InputStreamReader(in));
In the asynchronous Request Maintenance server shown in Listing 18.39, the asynchronous handling actually takes place in the server. Compare this to the JAX-WS implementation of the Request Maintenance server in Section 18.2 and the JavaScript/PHP/AJAX implementation of the Request Maintenance server in Section 18.3, where the asynchronous handling was done on the client side.

Here, a separate thread is spun off to do the asynchronous work, while the current process continues doing “useful work.” After a 10-second delay, the separate thread returns a web page containing the name of the person assigned to do maintenance.
LISTING 18.39 Request Maintenance Server

package example.myservlet;
import java.io.*;
import java.lang.Thread;
import javax.servlet.annotation.WebServlet;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.AsyncContext;
import javax.servlet.AsyncEvent;
import javax.servlet.AsyncListener;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.AsyncContext;
import javax.servlet.AsyncEvent;
import javax.servlet.AsyncListener;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletResponse;

class doWork implements Runnable {
    private AsyncContext ac;
    private PrintWriter myout;
    public doWork(AsyncContext the_ac, PrintWriter myout) {
        // set the context of the thread to the original context
        this.ac = the_ac;
        this.myout = myout;
    }
    public void run() {
        try {
            Thread.sleep(10000); // sleep for 10 sec to simulate hard work
            // Put some stuff in the response so it get done after completion
            myout.println("Your maintenance person will be Sabrina. She is available after 3pm next Friday");
            myout.println("<h2>");
            myout.println("<br>");
            myout.println("<br>");
            myout.println("<p><a href=
                \"http://localhost:8080/marina/manage.html\"
            >Return to Home</a></p>");
            System.out.println("just wrote some stuff into response");
        } catch (Exception e) {
            System.out.println("Inside doWork exception");
        }
    }
}
System.out.println("Some kind of Exception");
}
ac.complete(); // when hard work is done, signify completion
@

@WebServlet (urlPatterns="/requestmaintenance",asyncSupported = true)
public class requestmaintenance extends HttpServlet {

public PrintWriter myout;

public void doPost(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {

// Following output will show up in Glassfish log file
// because System.out is standard output it will show up
// on the console
System.out.println("Inside myservlet in doGet method");

AsyncContext ac = request.startAsync();

ac.addListener(new AsyncListener()
{
    @Override
    public void onComplete(AsyncEvent event) throws IOException {
        System.out.println("onComplete: Asynchronous task is now done");
    }

    @Override
    public void onTimeout(AsyncEvent event) throws IOException {
        System.out.println("onTimeout");
    }

    @Override
    public void onStartAsync(AsyncEvent event) throws IOException {
        System.out.println("onStartAsync");
    }

    @Override
    public void onError(AsyncEvent event) throws IOException {
        System.out.println("onError");
    }
});
System.out.println("just before thread start");

PrintWriter myout = ac.getResponse().getWriter();
new Thread (new doWork(ac, myout) ).start();

myout.print("<!DOCTYPE html>");
myout.print("<html>");
myout.print("<head>");
myout.print("<style>");
myout.print("a { color:lightseagreen; text-align:center }");
myout.print("p { color:lightseagreen; text-align:center; }");
myout.print("h2 { color:lightseagreen; text-align:center; font-size:150%; font-weight: bold; }");
myout.print("</style>");
myout.print("</head>");
myout.print("<body>");
myout.print("<h2>");
myout.print("Hi there");
String myname;
if (request.getParameter("name")!=null)
{
  myname=request.getParameter("name");
  myout.print(","+myname);
}
myout.print("!");
myout.print("</h2>");
myout.print("<br>");
myout.print("<h2>");

18.4.2 MARINA WEB PAGES

The clients to access the servers from the previous section are all written in HTML with JavaScript and are meant to run in a standard browser.

The main web page for this Sailboat Marina application is shown in Listing 18.40, and what it looks like when it’s running in a browser is shown in Figure 18.18.

In Listing 18.40, we can see that the two Display All Records servers (Display All Records without including ids, and Display All Records including ids) are called direct from inside the HTML:

- <a href="http://localhost:8080/marina/display">
- <a href="http://localhost:8080/marina/iddisplay">

The results after calling the display server are shown in Figure 18.19.
The results after calling the display server that includes id numbers are shown in Figure 18.20. To call the other servers, the main web page in Listing 18.40 calls different HTML web pages. These are shown later on in this section.

A Cascading Style Sheet (CSS stylesheet) called “mystylemanage.css” is used to format the HTML for display in the browser. This stylesheet is shown in Listing 18.41. As you can see, it sets h2 headers to a light seagreen color, centers them and puts them in bold. It does the same with h1 headers, except it makes h1 headers bigger than h2 headers (220% compared to 180%).

For the table shown in the web page, it sets the color of even numbered rows to white and the color of odd numbered rows to mintcream.
Compare all the stylesheet settings to what the web page looks like in Figure 18.18. Try running it yourself and varying the style (this code is all included in the code that was provided for the instructor with this textbook).

![Image of a web page with a table showing details of sailboats in a marina.](image.png)

**FIGURE 18.20** Results Returned by Display Server (Includes ID Numbers).

LISTING 18.40 Manage Web Page

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="mystylemanage.css">
  </head>
  <body>
    <br>
    <h1>Have a Windy Day!</h1>
    <br>
    <br>
    <h2>Manage the Marina</h2>
    <table>
      <caption>Select an Action</caption>
      <tr>
        <td><a href="http://localhost:8080/marina/display">Show Marina</a></td>
      </tr>
      <tr>
        <td><a href="http://localhost:8080/marina/iddisplay">Show Marina with Internal Ids</a></td>
      </tr>
      <tr>
        <td><a href="create.html">Add New Sailor/Sailboat to Marina</a></td>
      </tr>
      <tr>
        <td><a href="delete.html">Delete Sailor/Sailboat from Marina</a></td>
      </tr>
      <tr>
        <td><a href="details.html">Give Details of Sailor/Sailboat from Marina</a></td>
      </tr>
    </table>
  </body>
</html>
```
LISTING 18.41  CSS for Manage Web Page

```css
h2 {
  color: lightseagreen;
  text-align: center;
  font-size: 180%;
  font-weight: bold;
}

h1 {
  color: lightseagreen;
  text-align: center;
  font-size: 220%;
  font-weight: bold;
}

table, th, td {
  border: 1px solid black;
}

caption {
  color: lightseagreen;
  font-size: 150%;
  font-weight: bold;
  margin-bottom: 5px;
  text-align: center;
}

table {
  width: 500px;
  margin-left: auto;
  margin-right: auto;
}

th, td {
  text-align: center;
  padding: 10px;
}

th {
  background-color: lightseagreen;
  color: black;
}

tr:nth-child(odd) { background-color: mintcream }
tr:nth-child(even) { background-color: white }
```
In Listing 18.42 we use an HTML form to enter data and then send the data to the Create server. It sends the data using an HTTP POST. Compare this to the similar Create Web Page in Listing 18.27 (part of the JavaScript/PHP/AJAX Sailboat Marina example) which sends a GET instead.

The stylesheet that describes how this web page looks is shown in Listing 18.43. Compare Listing 18.42 together with Listing 18.43 to how the web page actually looks in Figure 18.21.

After you run the Create, the results returned by the Create server are shown in Figure 18.22.

LISTING 18.42  Create Web Page

```html
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="mystylecreate.css">
</head>
<body>
    <h1>Have a Windy Day!</h1>
    <h2>Add new Sailor/Boat to Marina</h2>
    <p id="myform">
        <form action="http://localhost:8080/marina/create" method="post">
            <input type="text" name="name" value="name">
            <br>
            <input type="text" name="model" value="boat model">
            <br>
            <input type="number" name="slip" value=0>
            <br>
            <br>
            <input type="date" name="begin_date" value="01-23-1979">
            <br>
            <br>
            <input type="date" name="end_date" value="01-23-1979">
            <br>
            <br>
            <input type="submit" value="Create" style="font-weight:bold">
        </form>
    </p>
    <p align="center"><a href="manage.html">Return to Home</a></p>
</body>
</html>
```
LISTING 18.43 CSS for Create Web Page

```css
a {
  color: lightseagreen;
}

h2 {
  color: lightseagreen;
  text-align: center;
  font-size: 150%;
  font-weight: bold;
}

h1 {
  color: lightseagreen;
  text-align: center;
  font-size: 200%;
  font-weight: bold;
}

form {
  color: lightseagreen;
  width: 300px;
  margin: 0 auto;
}
```

Have a Windy Day!

Add new Sailor/Boat to Marina

```html
name  
name  

model  
boat model  

slip  
0  

begin date  
01-23-1979  

end date  
01-23-1979  

Create  
```

FIGURE 18.21 Create Web Page.
In Listing 18.44, we select either an id number or a slip number to be used to select an item to delete. Then we pass either the slip or the id to the Delete server (see Listing 18.37). The stylesheet for this web page is shown in Listing 18.45. Look at how the actual Delete Web Page looks, see Figure 18.23. The results after running the delete are shown in Figure 18.24.

LISTING 18.44 Delete Web Page

```html
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="mystyledelete.css">
</head>
<body>
    <h1>Have a Windy Day!</h1>
    <h2>Delete Sailor/Boat from Marina</h2>
    <form action="http://localhost:8080/marina/delete" method="post">
        id<br>
        <input type="number" name="id" value=0>
        <br>
        <input type="submit" value="Delete by Id" style="font-weight:bold">
    </form>
    <br>
    <br>
    <form action="http://localhost:8080/marina/delete" method="post">
        slip<br>
        <input type="number" name="slip" value=0>
        <br>
        <input type="submit" value="Delete by Slip" style="font-weight:bold">
    </form>
    <br>
    <br>
    <a id="tohome" href="manage.html">Return to Home</a>
</body>
</html>
```

FIGURE 18.22 Results Returned by Create Server.

In Listing 18.44, we select either an id number or a slip number to be used to select an item to delete. Then we pass either the slip or the id to the Delete server (see Listing 18.37). The stylesheet for this web page is shown in Listing 18.45. Look at how the actual Delete Web Page looks, see Figure 18.23. The results after running the delete are shown in Figure 18.24.
LISTING 18.45  CSS for Delete Web Page and Details Web Page

```css
a {
  color: lightseagreen;
}

h2 {
  color: lightseagreen;
  text-align: center;
  font-size: 150%;
  font-weight: bold;
}

h1 {
  color: lightseagreen;
  text-align: center;
  font-size: 200%;
  font-weight: bold;
}

form {
  color: lightseagreen;
  width: 300px;
  margin: 0 auto;
}
```

The details web page code in Listing 18.46 is very similar to the Delete Web Page in Listing 18.44 (and to Listing 18.31, which is part of the JavaScript/PHP/AJAX Sailboat Marina project). Also, its stylesheet (see Listing 18.45) is the same as for the Delete Web Page. See Figure 18.25 for what this web page looks like, and see Figure 18.26 for what the details server returns after it has been called.

**FIGURE 18.23**  Delete Web Page.

---

**Have a Windy Day!**

Delete Sailor/Boat from Marina

```
id
0:
  Delete by id

slip
0:
  Delete by Slip
```

**Return to Home**
LISTING 18.46  Details Web Page

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="mystyledelete.css">
  </head>
  <body>
    <h1>Have a Windy Day!</h1>
    <h2>Details</h2>
    <form action="details" method="post">
      Id<br>
      <input type="number" name="id" value=0>
      <br>
      Slip<br>
      <input type="number" name="slip" value=0>
    </form>
    <br>
    <input type="submit" value="Details" style="font-weight:bold">
    <br>
    <br>
    <a id="tohome" href="manage.html">Return to Home</a>
  </body>
</html>
```

FIGURE 18.24  Results Returned by Delete Server.
The Request Maintenance web page in Listing 18.47 is very simple. When a name is entered and a button is pushed, it calls the Request Maintenance server in Listing 18.39. Its stylesheet is shown in Listing 18.48. For what this web page looks like, see Figure 18.27.

See Figure 18.28 for what the Request Maintenance server returns after it has been called.

FIGURE 18.25 Details Web Page.

FIGURE 18.26 Results Returned by Details Server.

The Request Maintenance web page in Listing 18.47 is very simple. When a name is entered and a button is pushed, it calls the Request Maintenance server in Listing 18.39. Its stylesheet is shown in Listing 18.48. For what this web page looks like, see Figure 18.27.

See Figure 18.28 for what the Request Maintenance server returns after it has been called.

LISTING 18.47 Request Maintenance Web Page

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="mystylerequestmaintenance.css">
  </head>
  <body>
    <h1>Have a Windy Day!</h1>
    <h2>RequestMaintenance</h2>
  </body>
</html>
```
<form action="http://localhost:8080/marina/requestmaintenance" method="post">
  name<br/>
  <input type="text" name="name" value="name">
  <br/>
  <input type="submit" value="RequestMaintenance" style="font-weight:bold">
  <br/>
  <br/>
  <a id="tohome" href="manage.html">Return to Home</a></p>
</form>
</body>
</html>

**LISTING 18.48 CSS for Request Maintenance Web Page**

```css
#thispart, #thatpart {
  color: lightseagreen;
  text-align: center;
  font-size: 150%;
}

a {
  color: lightseagreen;
}

h2 {
  color: lightseagreen;
  text-align: center;
  font-size: 150%;
  font-weight: bold;
}

h1 {
  color: lightseagreen;
  text-align: center;
  font-size: 200%;
  font-weight: bold;
}

form {
  color: lightseagreen;
  width: 300px;
  margin: 0 auto;
}
```
In Listing 18.49, where we show the directory structure in the WEB-INF file, note first that there is no need for a web.xml file because we are using the @WebServlet annotation, as previously shown in Section 11.2.3.

The lib directory contains the jar file for the Java Database Connectivity (JDBC) libraries.

LISTING 18.49 Deployment WEB-INF Directory Structure

```
\WEB-INF\classes
\WEB-INF\lib
```

LISTING 18.50 Deployment – How to Create the War File

```
jar -cvf marina.war WEB-INF create.html delete.html details.html finalformat.html initialformat.html manage.html requestmaintenance.html mystylecreate.css mystyle.css mystyledelete.css mystylemanage.css mystylerequestmaintenance.css
```
In Listing 18.50, note that the various html files and associated stylesheets are loaded directly instead of inside the WEB-INF directory. It’s a little confusing here because I have included finalformat.html and initialformat.html in this list. However, in practice they must (also or instead) be loaded into the \WEB-INF\classes directory or the java .class files will not be able to find them to load them. I put them in here as well as where they really need to be (that is, inside the \WEB-INF\classes directory) just to remind myself that I have to stuff them in the WEB-INF\classes directory to make the code work. Otherwise I might forget them entirely since there’s no build command directly related to them.

18.5 SAILBOAT MARINA PROJECT USING JAX-RS

In this section we look at an implementation of the Sailboat Marina project previously described in Section 1.5.

Before running the code in this section, you must implement a MySQL database named “thething” containing a table named “marina” as described in Section 4.11.1.1.

In these sections we will be using the Java Database Connectivity (JDBC) libraries. We won’t go over the details of the connections as they are presented here, since we previously covered that in Section 4.11.1.2.

18.5.1 MARINA SERVERS

When the servers in this section were tested, they were placed in the same Web Archive (war) file, and deployed on GlassFish and accessed at localhost. All were at the same port number (8080). The servers were all deployed using a web.xml file in the WEB-INF directory. We’ll talk more about deployment later on.

Some of the JAX-RS routines here were accessed with an HTTP GET and others were accessed with an HTTP POST. Let us consider Listing 18.51, the Display All Records server. The JAX-RS annotations at the top are:

```java
// called when request sends a GET with data in format “variable=value”
@GET
@Consumes("application/x-www-form-urlencoded")
@Produces(MediaType.TEXT_HTML)
```

If you compare this to Figure 18.25, the Display All Records server from the Java servlet implementation of the Sailboat Marina project there, we had:

```java
public void doGet(HttpServletRequest request,
    HttpServletResponse response)
```

So in this particular case both the JAX-RS and the servlet expected an HTTP GET. However, the JAX-RS implementation was able to further constrain the HTTP messages that it received by requiring headers related to the type of data it was willing to receive (application/x-www-form-urlencoded) and type of data it would produce (MediaType.TEXT_HTML).

Note also that the JAX-RS version simply returns a string, whereas in the servlet version, the HttpServletResponse is formatted as follows:

```java
// set the content type in the HTTP response
response.setContentType("text/html");
PrintWriter myout = response.getWriter();
```
then various print statements such as:

```java
myout.print("<tr>");
```

There is also a difference between servlets and JAX-RS in the way parameters are passed. Here is a method that receives a POST from the JAX-RS Create server, shown later on in Listing 18.53, it is accepting input parameters from an html form:

```java
@POST
@Consumes("application/x-www-form-urlencoded")
@Produces(MediaType.TEXT_HTML)
public String received_POST(
    @FormParam("name") String namestr,
    @FormParam("model") String modelstr,
    @FormParam("slip") int theslip,
    @FormParam("begin_date") String begindatestr,
    @FormParam("end_date") String enddatestr)
```

In the Create server that we saw previously as part of the servlet implementation of the Sailboat Marina project, shown in Listing 18.36, these parameters were extracted from the HttpServletRequest parameter as follows:

```java
String namestr,modelstr,slipstr,begindatestr,enddatestr;
namestr=request.getParameter("name");
modelstr=request.getParameter("model");
slipstr=request.getParameter("slip");
int theslip =Integer.parseInt(slipstr);
```

Other than these differences the JAX-RS implementation of the Sailboat Marina project is nearly the same as the Java servlet implementation. The only other real difference is in the way it is deployed. We’ll see this later on.

Let’s look further at Listing 18.51, the Display All Records server. Using JDBC, it connects to the database then creates a query that requests information about all the major items in a record (name, model, slip, begin_date, end_date, id). It then iterates through the results of the query, looking at the values for each major item (name, model, slip, begin_date, end_date). It uses PrintWriter to write the values for each major item, separated by tab characters, to a text output stream. Note that it does not write the id value into the string.

```
After creating the web page, it then closes the connection.
```

In Listing 18.51, Listing 18.52, and Listing 18.53, to create the output web page, it reads the initial boilerplate and the necessary code to set up an output table from the file that was previously shown in Listing 18.18.

In Listing 18.18, note that it is absolutely not an error that there is no terminating </table> statement, </body> statement, or </html> statement. The intent is for the current PHP server to read the code in Listing 18.18 to start up the web page, then write in the data (appropriate for the current server), then read in the code in Listing 18.19 to finish up the web page. So the ending statements are in Listing 18.19.

In Listing 18.18, note that it is absolutely not an error that there is no terminating </table> statement, </body> statement, or </html> statement. The intent is for the current PHP server to read the code in Listing 18.18 to start up the web page, then write in the data (appropriate for the current server), then read in the code in Listing 18.19 to finish up the web page. So the ending statements are in Listing 18.19.

Note that the only difference between the iddisplayitall server in Listing 18.51 and the displayitall server in Listing 18.51 is that the iddisplayitall server writes the id number into the string it returns, whereas the displayitall server does not.
package marina;
import java.io.InputStream;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.FormParam;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.MediaType;
import java.io.FileOutputStream;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.SQLException;
import java.util.Calendar;
import java.lang.Class;

@Path("/display")
public class displayitall {

    // set name of the JDBC driver
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    // set database URL
    static final String DB_URL = "jdbc:mysql://localhost/thething";
    // set your database username and password
    static final String myUN = "myownstuff";
    static final String myPW = "samsung";

    // called when request sends a GET with data in format "variable=value"
    @GET
    @Consumes("application/x-www-form-urlencoded")
    @Produces(MediaType.TEXT_HTML)
    public String received_GET() {
        StringBuffer myout = new StringBuffer(");
            // Following output will show up in Glassfish log file
            // because System.out is standard output it will show up
            // on the console
            System.out.println("Inside receivedGet method");

            Connection conn = null;
            Statement stmt = null;
            try{
                // Register the JDBC driver
                Class.forName("com.mysql.jdbc.Driver");

                // Attempt to open connection
                try {
                    conn = DriverManager.getConnection(DB_URL, myUN, myPW);
                    System.out.println("Connection is opened");
                    stmt = conn.createStatement();
                } catch (SQLException e) {
                    System.out.println("Connection failed");
                    e.printStackTrace();
                }

                // Execute Query
                String sql = "Select * from table1 and table2";
                Statement stmt = conn.createStatement();
                ResultSet rs = stmt.executeQuery(sql);

                while(rs.next()) {
                    // Do something with the ResultSet
                    System.out.println(rs.getString("column1").toString());
                    System.out.println(rs.getString("column2").toString());
                }
            }
            finally{
                if(stmt != null){ stmt.close(); } // Close connection
                if(conn != null){ conn.close(); } // Close connection
            }
        }
    }

    // called when request sends POST
    @POST
    @Consumes(MediaType.MULTIPART_FORM_DATA)
    @Produces(MediaType.TEXT_HTML)
    public String received_POST() {
        // Do something with the FormParams
        return "Success!";
    }
}

Listing 18.51 Display All Records Server
// Connect to the database
System.out.println("Attempting to connect to database. Trying hard!");
conn = DriverManager.getConnection(DB_URL, myUN, myPW);

// Create a query to the database
String mysql_Query;
System.out.println("Attempting to query database");
Stmt = conn.createStatement();
mysql_Query = "SELECT name, model, slip, begin_date, end_date, id FROM marina";
ResultSet rs = stmt.executeQuery(mysql_Query);

// Read initial HTML information from file also in WEB-INF/classes/marina
InputStream in = displayitall.class.getResourceAsStream("initialformatwithoutid.html");
BufferedReader r = new BufferedReader(new InputStreamReader(in));
String line;
line = r.readLine();
while (line != null)
{
    myout.append(line);
    line = r.readLine();
}
in.close();
System.out.println("Cjust finished reading a line");

// Get all data in the database
while(rs.next()){
    // grab the data in order by data column name
    String thename = rs.getString("name");
    String themodel = rs.getString("model");
    int theslip = rs.getInt("slip");
    String thebegindate = rs.getString("begin_date");
    Date theenddate = rs.getDate("end_date");
    Calendar mycalendar=Calendar.getInstance();
    mycalendar.setTime(thecddate);
    int month = mycalendar.get(Calendar.MONTH);
    int day = mycalendar.get(Calendar.DAY_OF_MONTH);
    int year = mycalendar.get(Calendar.YEAR);
    int theid = rs.getInt("id");

    myout.append("<tr>");
    myout.append("<td>");
    myout.append("name: "+thename);
    myout.append("</td>");
myout.append("<td>");
myout.append("model: "+themodel);
myout.append("</td>");
myout.append("<td>");
myout.append("slip: "+theslip);
myout.append("</td>");
myout.append("<td>");
myout.append("begin_date "+thebegindate);
myout.append("</td>");
myout.append("<td>");
myout.append("end_date "+theenddate);
myout.append("</td>");
myout.append("</tr>");

// Read final HTML information from file also in WEB-INF/classes/marina
in = displayitall.class.getResourceAsStream("finalformat.html");
r = new BufferedReader(new InputStreamReader(in));
line = r.readLine();
while (line != null)
{
    myout.append(line);
    line = r.readLine();
}
in.close();
// finish up when done
rs.close();
Stmt.close();
conn.close();

} catch(SQLException sqle){
    // problems with JDBC access
    Sqle.printStackTrace();
} catch(Exception e){
    e.printStackTrace();
} finally{
    // “finally” shut everything down
    try{
        if(stmt!=null)
            Stmt.close();
    } catch(SQLException allse){
        // ignore
    } // end try catch
    try{
        if(conn!=null)
            conn.close();
    } catch(Exception allse){
        // ignore
    } // end try catch
}catch(SQLException se){
    se.printStackTrace();
}//end try catch
} //end try

System.out.println("Be seein’ y’all");

String thestring = myout.toString();

return thestring;


LISTING 18.52  Display All Records Including IDs Server

package marina;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.FormParam;
import javax.ws.rs.core.Response;
import javax.ws.rs.Produces;
import javax.ws.rs.Consumes;
import javax.ws.rs.core.MediaType;
import java.io.*;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.*;
import java.sql.SQLException;
import java.util.Calendar;
import java.lang.Class;

@Path("/iddisplay")
public class iddisplayitall {
    // set name of the JDBC driver
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    // set database URL
    static final String DB_URL = "jdbc:mysql://localhost/thething";

    // set your database username and password
    static final String myUN = "myownstuff";
    static final String myPW = "samsung";

    // get the database driver
    static public void main(String[] args) {
        try{
            Class.forName(JDBC_DRIVER);
// called when request sends a POST with data in format “variable=value”
@GET
@Consumes("application/x-www-form-urlencoded")
@Produces(MediaType.TEXT_HTML)
public String received_GET() {
    StringBuffer myout = new StringBuffer("\n");
    // Following output will show up in Glassfish log file
    // because System.out is standard output it will show up
    // on the console
    System.out.println("Inside receivedGet method");
    Connection conn = null;
    Statement stmt = null;
    try{
        // Register the JDBC driver
        Class.forName("com.mysql.jdbc.Driver");
        // Connect to the database
        System.out.println("Attempting to connect to database. Trying hard!");
        conn = DriverManager.getConnection(DB_URL,myUN,myPW);
        // Create a query to the database
        String mysql_Query;
        System.out.println("Attempting to query database");
        stmt = conn.createStatement();
        mysql_Query = "SELECT name,model,slip,begin_date,end_date,id FROM marina";
        ResultSet rs = stmt.executeQuery(mysql_Query);
        // Get all data in the database
        while(rs.next()){
            // grab the data in order by data column name
            String thename = rs.getString("name");
            String themodel = rs.getString("model");
            int theslip = rs.getInt("slip");
            myout.append(line);
            line = r.readLine();
        }
    }
    in.close();
    return myout.toString();
}

String thebegindate = rs.getString("begin_date");
//String theenddate = rs.getString("end_date");
//Timestamp theenddate = rs.getTimestamp("end_date");
Date theenddate = rs.getDate("end_date");

Calendar mycalendar = Calendar.getInstance();
mycalendar.setTime(theenddate);
int month = mycalendar.get(Calendar.MONTH);
int day = mycalendar.get(Calendar.DAY_OF_MONTH);
int year = mycalendar.get(Calendar.YEAR);

int theid = rs.getInt("id");

myout.append("<tr>");
myout.append("<td>");
myout.append("name: "+thename);
myout.append("</td>");
myout.append("<td>");
myout.append("model: "+themodel);
myout.append("</td>");
myout.append("<td>");
myout.append("slip: "+theslip);
myout.append("</td>");
myout.append("<td>");
myout.append("begin_date "+thebegindate);
myout.append("</td>");
myout.append("<td>");
myout.append("end_date "+theenddate);
myout.append("</td>");
myout.append("<td>");
myout.append("id "+theid);
myout.append("</td>");
myout.append("</tr>");
}

// finish up when done
rs.close();
stmt.close();
conn.close();

// Read initial HTML information from file also in WEB-INF/classes/marina
in = iddisplayitall.class.getResourceAsStream("finalformat.html");
r = new BufferedReader(new InputStreamReader(in));
line = r.readLine();
System.out.println("Ajust read a line");
System.out.println("line is : ");
System.out.println(line);
while (line != null)
Let’s look next at the Create method in Listing 18.53. It uses JDBC to connect to the database. Then it searches through the id numbers in the Marina database to find the maximum id number, and it adds one to this id number to create the id number for the new item that is being created.

Then Create makes a “query” (better called just a database statement) that includes an SQL INSERT command to stuff a new record into the database, with the new values that Create had passed to it as parameters. It executes this statement and if it is successful then a new record has been added to the database. A web page that shows the newly created item is returned.

The Delete server in Listing 18.54 allows you to delete a record based either on the slip number or the id number, as both are supposed to be unique (slip is unique except when storing something like a rowboat in the same slip as a sailboat). Whichever one (id or slip) you don’t want to use to specify a record has a zero value passed in for that value. Then it uses JDBC to connect to the database, and creates a statement that says to delete a record from the Marina based on an id number, or else delete a record from the Marina based on a slip number, depending on which was chosen earlier. Then it
executes the statement and if the delete was successful, it returns a small web page showing the item that was deleted. Note that the style of the web page was written to the web page at the time the data was written.

**LISTING 18.53  Create New Record Server**

```java
package marina;
import java.io.InputStream;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.FormParam;
import javax.ws.rs.core.Response;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import java.io.*;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.*;
import java.lang.Class;
import java.util.Calendar;
import java.util.Date;
import java.text.SimpleDateFormat;
@Path("/create")
public class create {
    // set name of the JDBC driver
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    // set database URL
    static final String DB_URL = "jdbc:mysql://localhost/thething";

    // set your database username and password
    static final String myUN = "myownstuff";
    static final String myPW = "samsung";

    StringBuffer myout = new StringBuffer("");

    // called when request sends a POST with data in format "variable=value"
    @POST
    @Consumes("application/x-www-form-urlencoded")
    @Produces(MediaType.TEXT_HTML)
    public String received_POST(
        @FormParam("name") String namestr,
        @FormParam("model") String modelstr,
        @FormParam("slip") int theslip,
```
@FormParam("begin_date") String begindatestr,
@FormParam("end_date") String enddatestr)
{

    // Following output will show up in Glassfish log file
    // because System.out is standard output it will show up
    // on the console
    System.out.println("Inside receivedPOST method");
    Connection conn = null;

    Statement stmt = null;
    try{
        // Register the JDBC driver
        Class.forName("com.mysql.jdbc.Driver");

        // Connect to the database
        System.out.println("Attempting to connect to database. Trying hard!");
        conn = DriverManager.getConnection(DB_URL,myUN,myPW);

        // Create a query to the database
        System.out.println("Attempting to query database");

        // set the id to biggest id number plus one
        // to do this we must first read the ids from the database
        String mysql_Query;
        stmt = conn.createStatement();
        mysql_Query = "SELECT id FROM marina";
        ResultSet rs = stmt.executeQuery(mysql_Query);

        int theid=0;
        int i=0;
        // Get all data in the database
        while(rs.next()){
            int someid = rs.getInt("id");
            if (someid>i)
                i=someid;
            }
        theid=i+1;  // set id of new record to biggest previous id plus 1

        // finish up when done
        rs.close();

        String mysql_QueryA;
        mysql_QueryA = "INSERT INTO marina (name, model, slip, begin_date,end_date,id) VALUES (?,?,?,?)";
        PreparedStatement statement = conn.prepareStatement(mysql_QueryA);
statement.setString(1,namestr);
statement.setString(2,modelstr);
statement.setInt(3,theslip);

SimpleDateFormat sdf = new SimpleDateFormat("MM-dd-yyyy");

String mystring = begindatestr;
System.out.println("begindatestr is"+begindatestr);
System.out.println("mystring is"+mystring);

Date thedate = sdf.parse(mystring);
java.sql.Date sqlbegin_date= new java.sql.Date(thedate.getTime());
statement.setDate(4,sqlbegin_date);
thedate = sdf.parse(enddatestr);
java.sql.Date sqlend_date= new java.sql.Date(thedate.getTime());
statement.setDate(5,sqlend_date);

statement.setInt(6,theid); // id field

int rows_inserted = statement.executeUpdate();
if (rows_inserted>0)
{
 System.out.println("inserted number of rows: "+rows_inserted);
}

// Create a query to the database
String mysql_QueryB;
System.out.println("Attempting to query database");
stmt = conn.createStatement();

mysql_QueryB = "SELECT name,model,slip,begin_date,end_date,id FROM marina where id="+theid;

ResultSet rsB = stmt.executeQuery(mysql_QueryB);

String aname="";
String amodel="";
int aslip=0;
String abegindate="";
Date aenddate= new Date();
int aid=0;

if (rsB.next() )
{
 // grab the data in order by data column name
 aname = rsB.getString("name");
 amodel = rsB.getString("model");
 aslip = rsB.getInt("slip");
 abegindate = rsB.getString("begin_date");
aenddate = rsB.getDate("end_date");
aid = rsB.getInt("id");
}
rs.close();
stmt.close();
conn.close();

// finish up when done
conn.close();

// Read initial HTML information from file also in WEB-INF/classes/marina
InputStream in = create.class.getResourceAsStream("initialformat.html");
BufferedReader r = new BufferedReader(new InputStreamReader(in));

String line;
line = r.readLine();
while (line != null)
{
    myout.append(line);
    line = r.readLine();
}
in.close();

myout.append("<h2>");
myout.append("Successfully created new row in database:");
myout.append("</h2>");
myout.append("<tr>");
myout.append("<td>");
myout.append(aname);
myout.append("</td>");
myout.append("<td>");
myout.append(amodel);
myout.append("</td>");
myout.append("<td>");
myout.append(aslip);
myout.append("</td>");
myout.append("<td>");
myout.append(abegindate);
myout.append("</td>");
myout.append("<td>");
myout.append(aenddate);
myout.append("</td>");
myout.append("<td>");
myout.append(aid);
myout.append("</td>");
myout.append("</tr>");
// Read HTML information from file also in WEB-INF/classes/marina
in = create.class.getResourceAsStream("finalformat.html");
r = new BufferedReader(new InputStreamReader(in));

line = r.readLine();

while (line != null)
{
    myout.append(line);
    line = r.readLine();
}
in.close();
}catch(SQLException sqle){
    // problems with JDBC access
    sqle.printStackTrace();
}catch(Exception e){
    e.printStackTrace();
}finally{
    // “finally” shut everything down
    try{
        if(stmt!=null)
            stmt.close();
    }catch(SQLException allse){
        // ignore
    } // end try catch
    try{
        if(conn!=null)
            conn.close();
    }catch(SQLException se){
        se.printStackTrace();
    }//end try
}//end finally

System.out.println("Be seein’ y’all");

String thestring = myout.toString();

    return thestring;
}

The Details server in Listing 18.55 works similarly to the Delete server in Listing 18.54 in the way that it accepts the parameters slip and id. After that, it works similarly to the Display All Records including the ids server in Listing 18.52 in the way it queries the database, except that it only requests the records that match the given id or slip (whichever was used). It then displays the data the same way the Display All Records server did.
package marina;
import java.io.*;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.FormParam;
import javax.ws.rs.core.Response;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import java.io.*;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.*;
import java.sql.SQLException;
import java.util.Calendar;
@Path("/delete")
public class delete {

    // set name of the JDBC driver
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    // set database URL
    static final String DB_URL = "jdbc:mysql://localhost/thething";

    // set your database username and password
    static final String myUN = "myownstuff";
    static final String myPW = "samsung";

    StringBuffer myout = new StringBuffer("");

    @POST
    @Consumes("application/x-www-form-urlencoded")
    @Produces(MediaType.TEXT_HTML)
    public String received_POST(
        @FormParam("slip") int whichslip,
        @FormParam("id") int whichid
    )
    {
        // Following output will show up in Glassfish log file
        // because System.out is standard output it will show up
        // on the console
        System.out.println("Inside received_POST method");

        Connection conn = null;
        try{
            // Register the JDBC driver
            Class.forName("com.mysql.jdbc.Driver");
            // set name of the JDBC driver
            static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
            // set database URL
            static final String DB_URL = "jdbc:mysql://localhost/thething";

            // set your database username and password
            static final String myUN = "myownstuff";
            static final String myPW = "samsung";

            StringBuffer myout = new StringBuffer("");
// Connect to the database
System.out.println("Attempting to connect to database. Trying hard!");
conn = DriverManager.getConnection(DB_URL,myUN,myPW);

// Create a query to the database
String mysql_stmt;
System.out.println("Attempting to query database");
if (whichslip !=0)
    {
        mysql_stmt = "DELETE FROM marina where slip="+whichslip;
    }
else if (whichid !=0)
    {
        mysql_stmt = "DELETE FROM marina where id="+whichid;
    }
else
    {
        mysql_stmt = "DELETE FROM marina where slip=0"
        // default to slip =0 which should result in no delete
    }
PreparedStatement stmt = conn.prepareStatement(mysql_stmt);
int rows_deleted = stmt.executeUpdate();
if (rows_deleted > 0)
    System.out.println("Row was successfully deleted.");
myout.append("<!DOCTYPE html>");
myout.append("<html>");
myout.append("<head>");
myout.append("<style>");
myout.append("a { color:lightseagreen; text-align:center }");
myout.append("h2 { color:lightseagreen; text-align:center; font-size:150%; font-weight: bold; }");
myout.append("p { color:lightseagreen; text-align:center; }");
myout.append("<style>");
myout.append("</style>");
myout.append("<head>");
myout.append("<body>");
myout.append("<h2>");
myout.append("Delete was performed");
myout.append("<br>");
myout.append("<br>");
myout.append("<a href="http://localhost:8080/marina/manage.html">Return to Home</a>");
myout.append("</body>");
LISTING 18.55  Details of Record Server

package marina;
import java.io.InputStream;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.FormParam;
import javax.ws.rs.core.Response;
import javax.ws.rs.Produces;
import javax.ws.rs.Consumes;
import javax.ws.rs.core.MediaType;
import java.io.*;
import java.sql.DriverManager;
import java.io.*;
import java.sql.DriverManager;

myout.append("<html>");
// finish up when done
conn.close();
)
catch(SQLException s ql e)
{
  // problems with JDBC access
  s ql e.printStackTrace();
} catch(Exception e)
{
  e.printStackTrace();
}
finally
{
  // “finally” shut everything down
  try{
    if(conn!=null)
      conn.close();
  } catch(SQLException se)
  {
    se.printStackTrace();
  }
  System.out.println("Be seein’ y’all");
  String thestring = myout.toString();
  return thestring;
}
}
import java.sql.Connection;
import java.sql.*;
import java.sql.SQLException;
import java.util.Calendar;
import java.lang.Class;

@Path("/details")
public class details {
    // set name of the JDBC driver
    static final String JDBC_DRIVER = ":com.mysql.jdbc.Driver";
    // set database URL
    static final String DB_URL = "jdbc:mysql://localhost/thething";

    // set your database username and password
    static final String myUN = "myownstuff";
    static final String myPW = "samsung";

    // called when request sends a POST with data in format “variable=value”
    @POST
    @Consumes("application/x-www-form-urlencoded")
    @Produces(MediaType.TEXT_HTML)
    public String received_POST(
        @FormParam("slip") int slip,
        @FormParam("id") int id) {
        int whichslip;
        int whichid;

        whichid = 0;
        whichslip = 0;
        if (slip!=0)
            whichslip = slip;
        else if (id!=0) // null?
            whichid =id;

        // Following output will show up in Glassfish log file
        // because System.out is standard output it will show up
        // on the console
        System.out.println("Inside receivedPOST method");

        StringBuffer myout = new StringBuffer("");

        Connection conn = null;
        Statement stmt = null;
        try{
            // Register the JDBC driver
            // Set the connection properties
            // Establish a Connection to the database
            Connection conn = DriverManager.getConnection(DB_URL, myUN, myPW);
            // Create a Statement object
            Statement stmt = conn.createStatement();

            // Execute the SQL statement
            String sql = "SELECT * FROM my_table WHERE slip = ? AND id = ?";

            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setInt(1, whichslip);
            pstmt.setInt(2, whichid);

            ResultSet rs = pstmt.executeQuery();

            while (rs.next()) {
                String name = rs.getString("name");
                int slip = rs.getInt("slip");
                int id = rs.getInt("id");

                // Process the data
                myout.append(name + " slip: " + slip + " id: " + id + ";
            }

            // Close resources
            rs.close();
            stmt.close();
            conn.close();
        } catch (SQLException e) {
            System.out.println("Error: "+e.getMessage());
        } finally {
            // Close resources
            if (conn!=null) conn.close();
            if (stmt!=null) stmt.close();
            if (rs!=null) rs.close();
        }

        return myout.toString();
    }
}
Class.forName("com.mysql.jdbc.Driver");

// Connect to the database
System.out.println("Attempting to connect to database. Trying hard!");
conn = DriverManager.getConnection(DB_URL, myUN, myPW);

// Create a query to the database
String mysql_Query;
System.out.println("Attempting to query database");
stmt = conn.createStatement();
if (whichslip != 0) {
    mysql_Query = "SELECT name, model, slip, begin_date, end_date, id 
    FROM marina where slip=\"\"+whichslip;
}
else if (whichid != 0) {
    mysql_Query = "SELECT name, model, slip, begin_date, end_date, id 
    FROM marina where id=\"\"+whichid;
}
else {
    mysql_Query = "SELECT name, model, slip, begin_date, end_date, id 
    FROM marina where slip=0"; // default to slip =0 which should result in no output
}
ResultSet rs = stmt.executeQuery(mysql_Query);

// Create the web page to be returned
myout.append("<html>\n\n<h1>Details</h1>
<body>
<br>

// Get all data in the database
while(rs.next()){
    // grab the data in order by data column name
    String thename = rs.getString("name");
    String themodel = rs.getString("model");
    int theslip = rs.getInt("slip");
    String thebegindate = rs.getString("begin_date");
    Date theenddate = rs.getDate("end_date");
    Calendar mycalendar=Calendar.getInstance();
    mycalendar.setTime(thenddate);
    int month = mycalendar.get(Calendar.MONTH);
    int day = mycalendar.get(Calendar.DAY_OF_MONTH);
int year = mycalendar.get(Calendar.YEAR);
int theid = rs.getInt("id");

// Read initial HTML information from file also
// in WEB-INF/classes/marina
InputStream in = details.class.getResourceAsStream("initialformat.html");
BufferedReader r = new BufferedReader(new InputStreamReader(in));
String line;
line = r.readLine();
while (line != null)
{
    myout.append(line);
    line = r.readLine();
}
in.close();
myout.append("<tr>");
myout.append("<td>");
myout.append(thename);
myout.append("</td>");
myout.append("<td>");
myout.append(themodel);
myout.append("</td>");
myout.append("<td>");
myout.append(theslip);
myout.append("</td>");
myout.append("<td>");
myout.append(thebegindate);
myout.append("</td>");
myout.append("<td>");
myout.append(theenddate);
myout.append("</td>");
myout.append("<td>");
myout.append(theid);
myout.append("</td>");
myout.append("</tr>");

// Read HTML information from file that is
// also in WEB-INF/classes/marina
in = details.class.getResourceAsStream("finalformat.html");
r = new BufferedReader(new InputStreamReader(in));
line = r.readLine();
while (line != null)
{
    myout.append(line);
    line = r.readLine();
In the asynchronous Request Maintenance server shown in Listing 18.56, the asynchronous handling actually takes place in the server. Compare this to the JAX-WS implementation of the Request Maintenance server in Section 18.2.1 and the JavaScript/PHP/AJAX implementation of the Request Maintenance server in Section 18.3.1, where the asynchronous handling was done on the client side. Here, a separate thread is spun off to do the asynchronous work, while the current process continues doing “useful work.” After a 10-second delay, the separate thread returns a web page containing the name of the person assigned to do maintenance.

LISTING 18.56 Request Maintenance Server

```java
package marina;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.ResponseEntity;
import java.sql.SQLException;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.Connection;
import java.sql.DriverManager;

public class RequestMaintenanceServer {

    public static void main(String[] args) throws SQLException {
        Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/marina", "root", "password");
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("select name from assigned where request = 'maintenance'");
        try {
            while (rs.next()) {
                System.out.println("Be seein’ y’all");
                String thestring = rs.getString("name");
                return thestring;
            }
        } catch (SQLException sqle) {
            sqle.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            rs.close();
            stmt.close();
            conn.close();
            System.out.println("Be seein’ y’all");
        }
    }
}
```
import javax.ws.rs.core.Response;
import javax.ws.rs.container.AsyncResponse;
import javax.ws.rs.container.Suspended;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.ArrayBlockingQueue;

@Path("/requestmaintenance")
public class requestmaintenance {

    @GET
    public void asyncGet(@Suspended final AsyncResponse asyncResponse) {
        new Thread (new Runnable() {
            @Override
            public void run() {

                System.out.println("inside requestmaintenance asyncGet");

            }
        }).start();
    }
}

18.5.2 Marina Web Pages

The clients to access the servers from the previous section are all written in HTML with JavaScript and are meant to run in a standard browser.

The main web page for this Sailboat Marina application is shown in Listing 18.57, and what it looks like when it’s running in a browser is shown in Figure 18.29.

In Listing 18.57, we can see that the two Display All Records servers (Display All Records without including ids, and Display All Records including ids) are called direct from inside the HTML:

- `<a href="http://localhost:8080/marina/restful/display">`
- `<a href="http://localhost:8080/marina/restful/iddisplay">`
Have a Windy Day!

Manage the Marina

Select an Action

- Show Marina
- Show Marina with Internal IDs
- Add New Sailor/Sailboat to Marina
- Delete Sailor/Sailboat from Marina
- Give Details of Sailor/Sailboat from Marina
- Request Maintenance Person

FIGURE 18.29  Manage Web Page.

Have a Windy Day!

Marina Sailboats

<table>
<thead>
<tr>
<th>Name</th>
<th>Model</th>
<th>Slip</th>
<th>Begin Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: Etzkorn</td>
<td>model: Catalina27</td>
<td>slip: 17</td>
<td>begin_date 2013-07-16</td>
<td>end_date 2016-07-19</td>
</tr>
<tr>
<td>name: Smith</td>
<td>model: Hunter25</td>
<td>slip: 16</td>
<td>begin_date 2012-02-05</td>
<td>end_date 2016-08-17</td>
</tr>
<tr>
<td>name: Blackjack</td>
<td>model: boat model</td>
<td>slip: 5</td>
<td>begin_date 1981-01-23</td>
<td>end_date 2011-01-23</td>
</tr>
<tr>
<td>name: Spartacus</td>
<td>model: boat model</td>
<td>slip: 0</td>
<td>begin_date 1979-01-23</td>
<td>end_date 1979-01-23</td>
</tr>
<tr>
<td>name: name</td>
<td>model: boat model</td>
<td>slip: 0</td>
<td>begin_date 1979-01-23</td>
<td>end_date 1979-01-23</td>
</tr>
</tbody>
</table>

FIGURE 18.30  Results Returned by Display Server.

Note that these URLs are different from the URLs contained in the Manage Web Page for the Java servlet, in that the word “restful” is inserted after “marina.” This is due to the way the JAX-RS code is deployed.

The results after calling the Display server are shown in Figure 18.30.

The results after calling the Display server that includes id numbers are shown in Figure 18.31.

To call the other servers, the main web page in Listing 18.57 calls different HTML web pages. These are shown later on in this section.

A Cascading Style Sheet (CSS stylesheet) called “mystylemanage.css” is used to format the HTML for Display in the browser. This stylesheet is shown in Listing 18.58. As you can see, it sets h2 headers to a light seagreen color, centers them and puts them in bold. It does the same with h1 headers, except it makes h1 headers bigger than h2 headers (220% compared to 180%).
LISTING 18.57  Manage Web Page

```html
<!DOCTYPE html>
<html>
  
  <head>
    <link rel="stylesheet" type="text/css" href="mystylemanage.css">
  </head>

  <body>
    <h1>Have a Windy Day!</h1>

    <table>
      <caption>Select an Action</caption>
      <tr><td><a href="http://localhost:8080/marina/restful/display">Show Marina</a></td></tr>
      <tr><td><a href="http://localhost:8080/marina/restful/iddisplay">Show Marina with Internal Ids</a></td></tr>
      <tr><td><a href="create.html">Add New Sailor/Sailboat to Marina</a></td></tr>
      <tr><td><a href="delete.html">Delete Sailor/Sailboat from Marina</a></td></tr>
      <tr><td><a href="details.html">Give Details of Sailor/Sailboat from Marina</a></td></tr>
      <tr><td><a href="requestmaintenance.html">Request Maintenance Person</a></td></tr>
    </table>

  </body>
</html>
```

FIGURE 18.31  Results Returned by Display Server (Includes ID Numbers).
For the table shown in the web page, it sets the color of even-numbered rows to white and the color of odd-numbered rows to mintcream.

Compare all the stylesheet settings to what the web page looks like in Figure 18.29. Try running it yourself and varying the style (this code is all included in the code that comes with this textbook).

**LISTING 18.58  CSS for Manage Web Page**

```css
h2 {
  color: lightseagreen;
  text-align: center;
  font-size: 180%;
  font-weight: bold;
}

h1 {
  color: lightseagreen;
  text-align: center;
  font-size: 220%;
  font-weight: bold;
}

table, th, td {
  border: 1px solid black;
}

caption {
  color: lightseagreen;
  font-size: 150%;
  font-weight: bold;
  margin-bottom: 5px;
  text-align: center;
}

table {
  width: 500px;
  margin-left: auto;
  margin-right: auto;
}

th, td {
  text-align: center;
  padding: 10px;
}
```
In Listing 18.59 we use an HTML form to enter data and then send data to the Create server. It sends the data using an HTTP POST. Compare this to the similar Create web page in Listing 18.27 (part of the JavaScript/PHP/AJAX Sailboat Marina example), which sends a GET instead.

The stylesheet that describes how this web page looks is shown in Listing 18.60. Compare Listing 18.59, together with Listing 18.60, to how the web page actually looks in Figure 18.32.

After you run the Create, the results returned by the Create server are shown in Figure 18.33.

LISTING 18.59 Create Web Page

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="mystylecreate.css">
  </head>
  <body>
    <h1>Have a Windy Day!</h1>
    <h2>Add new Sailor/Boat to Marina</h2>
    <form action="http://localhost:8080/marina/restful/create" method="post">
      name<br>
      <input type="text" name="name" value="name">
      <br>
      <br>
      model<br>
      <input type="text" name="model" value="boat model">
      <br>
      <br>
      slip<br>
      <input type="number" name="slip" value=0>
      <br>
      <br>
      begin_date<br>
      <input type="date" name="begin_date" value="01-23-1979">
      <br>
      <br>
      end_date<br>
      <input type="date" name="end_date" value="01-23-1979">
    </form>
  </body>
</html>
```
FIGURE 18.32  Create Web Page.

FIGURE 18.33  Results Returned by Create Server.
LISTING 18.60 CSS for Create Web Page

```css
a {
  color: lightseagreen;
}

h2 {
  color: lightseagreen;
  text-align: center;
  font-size: 150%;
  font-weight: bold;
}

h1 {
  color: lightseagreen;
  text-align: center;
  font-size: 200%;
  font-weight: bold;
}

form {
  color: lightseagreen;
  width: 300px;
  margin: 0 auto;
}
```

In Listing 18.61, we select either an id number or a slip number to be used to select an item to delete. Then we pass either the slip or the id to the Delete server (see Listing 18.54). The stylesheet for this web page is shown in Listing 18.62. Look at how the actual Delete Web Page looks; see Figure 18.34.

The results after running the Delete are shown in Figure 18.35.

LISTING 18.61 Delete Web Page

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="mystyledelete.css">
  </head>
  <body>
    <h1>Have a Windy Day!</h1>
    <h2>Delete Sailor/Boat from Marina</h2>
    <form action="http://localhost:8080/marina/restful/delete" method="post">
      id<br>
      <input type="number" name="id" value=0>
      <br>
      <input type="submit" value="Delete by Id" style="font-weight:bold">
    </form>
  </body>
</html>
```
The details web page code in Listing 18.63 is very similar to the Delete Web Page in Listing 18.61 (and to Listing 18.31, which is part of the JavaScript/PHP/AJAX Sailboat Marina project). Also, its stylesheet (see Listing 18.62) is the same as for the Delete Web Page. See Figure 18.36 for what this web page looks like, and see Figure 18.37 for what the details server returns after it has been called.
Have a Windy Day!
Delete Sailor/Boat from Marina

id
0
Delete by id

slip
0
Delete by Slip.

Return to Home

FIGURE 18.34 Delete Web Page.

Delete was performed

Return to Home

FIGURE 18.35 Results Returned by Delete Server.

Have a Windy Day!
Details

Id
0
Details

Return to Home

Slip
0
Details

Return to Home

FIGURE 18.36 Details Web Page.
The Request Maintenance web page in Listing 18.64 is very simple. When a name is entered and a button is pushed, it calls the Request Maintenance server in Listing 18.56. Its stylesheet is shown in Listing 18.65. For what this web page looks like, see Figure 18.38.

See Figure 18.39 for what the Request Maintenance server returns after it has been called.
LISTING 18.64 Request Maintenance Web Page

```html
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="mystylerequestmaintenance.css">
</head>
<body>
    <h1>Have a Windy Day!</h1>
    <h2>RequestMaintenance</h2>
    <form action="http://localhost:8080/marina/restful/request_the_maintenance" method="post">
        name<br>
        <input type="text" name="name" value="name">
        <br>
        Your maintenance person is Sabrina. She will be available Friday after 3pm
        <br>
        <br>
    </form>
    <a id="tohome" href="manage.html">Return to Home</a>
</body>
</html>
```

FIGURE 18.38 Request Maintenance Web Page.

FIGURE 18.39 Results Returned by Request Maintenance Server.
LISTING 18.65  CSS for Request Maintenance Web Page

```css
#thispart, #thatpart {
  color: lightseagreen;
  text-align: center;
  font-size: 150%;
}

a {
  color: lightseagreen;
}

h2 {
  color: lightseagreen;
  text-align: center;
  font-size: 150%;
  font-weight: bold;
}

h1 {
  color: lightseagreen;
  text-align: center;
  font-size: 200%;
  font-weight: bold;
}

form {
  color: lightseagreen;
  width: 300px;
  margin: 0 auto;
}
```

In Listing 18.66, where we show the directory structure in the WEB-INF file, note first that here we use a web.xml file for deployment.

The lib directory contains the jar file for the Java Database Connectivity (JDBC) libraries.

LISTING 18.66  Deployment WEB-INF Directory Structure

```
WEB-INF/classes
WEB-INF/lib
web.xml
```

LISTING 18.67  Deployment – How to Create the War File

```
jar -cvf marina.war WEB-INF create.html delete.html details.html finalformat.html initialformat.html manage.html requestmaintenance.html mystylecreate.css mystyle.css mystyledelete.css mystylemanage.css mystylerequestmaintenance.css
```
In Listing 18.67, note that the various html files and associated stylesheets are loaded directly instead of inside the WEB-INF directory. It’s a little confusing here because I have included finalformat.html and initialformat.html in this list. However, in practice they must (also or instead) be loaded into the \WEB-INF\classes directory or the java .class files will not be able to find them to load them. I put them in here as well as where they really need to be (that is, inside the \WEB-INF\classes directory) just to remind myself that I have to stuff them in the WEB-INF\classes directory to make the code work. Otherwise I might forget them entirely, since there’s no build command directly related to them.

<table>
<thead>
<tr>
<th>LISTING 18.68</th>
<th>web.xml File used for Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;!– load application class all @Path and @Provider classes included in application will be automatically registered in the application –&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;servlet&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;servlet-name&gt;javax.ws.rs.core.Application&lt;/servlet-name&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;load-on-startup&gt;1&lt;/load-on-startup&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;async-supported&gt;true&lt;/async-supported&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;/servlet&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;servlet-mapping&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;servlet-name&gt;javax.ws.rs.core.Application&lt;/servlet-name&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;url-pattern&gt;/restful/*&lt;/url-pattern&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;/servlet-mapping&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;/web-app&gt;</td>
<td></td>
</tr>
</tbody>
</table>

In the web.xml file, we see that asynchronous operation is supported:

<async-supported>true</async-supported>

We also see that the URL pattern being used is as follows:

<url-pattern>/restful/*</url-pattern>

This is why our URLs for this project are similar to the following (this is the URL for the Display All Records server):

http://localhost:8080/marina/restful/display

CONCEPTUAL QUESTIONS

1. In the EJB implementation of the Sailboat Marina project, when the Create method tries to find an unused id number (since the id number is supposed to be unique), the code counts the number of rows, then picks an id number as one greater than the total number of rows. This does NOT work well and sometimes you end up with two items with the same id number. Why do you think this is? (Hint: sometimes records must
be deleted). (Note that in the other implementations of the Marina project in this textbook, I did this differently, in a way that works better. Look it up!)

2. What changes would you have to make to implement the Sailboat Marina project as a Stateless Session Bean instead of what we did, which is use a Stateful Session Bean? Is it even possible? Why would you do one instead of the other?

3. Compare the Java Swing libraries (this chapter and Chapter 17) and HTML interfaces (this chapter) to Windows Presentation Foundation (Chapter 17). What are the advantages and disadvantages of each?