Database Connectivity with JDBC

- The JDBC API makes it possible to access databases and other data sources from Java.
  
  ```java
  import java.sql.*;
  ...
  Class.forName("com.mysql.jdbc.Driver").newInstance();
  String jdbc = "jdbc:mysql://localhost:3306/db?user=smith&password=xxx";
  Connection con = DriverManager.getConnection(jdbc);
  Statement stmt = con.createStatement();
  ResultSet rs = stmt.executeQuery("select * from employee");
  while (rs.next())
      System.out.println(rs.getString("fname") + rs.getString("lname");
  rs.close();
  stmt.close();
  con.close();
  ```

- For updates/inserts/deletes, use
  ```java
  stmt.executeUpdate("update ... ");
  con.commit();
  ```

Working with ResultSet

- **ResultSet**: a table of data representing a database result set
  - generated by executing a statement that queries the database
  - It maintains a cursor pointing to its current row of data
    - Initially the cursor is positioned before the first row
    - The next method moves the cursor to the next row
  - Provides getter methods for retrieving column values from the current row
    - `get<String, getInt, getLong, ...`
  - Also provides setter methods for updating column values
    - `updateString, updateInt, ...`
  - Values can be retrieved/updated using either
    - the index number of the column (starting from 1)
    - `rs.getString(2)`, `rs.getString(2,"Smith")`
    - or the name of the column
    - `rs.getString("name")`, `rs.updateString("name","Smith")`

Updates

- To delete the current row from the ResultSet and from database
  ```java
  rs.deleteRow();
  ```
- To update a column value in the current row
  ```java
  rs.updateString("name", "Smith");
  rs.updateInt("salary", 100000);  
  rs.updateRow();
  ```
- To insert column values into the insert row
  ```java
  rs.moveToInsertRow();
  rs.updateString("name", "Smith");
  rs.updateInt("salary", 100000);  
  rs.insertRow();
  ```
Java Servlets

- A **servlet** is a small Java program that runs within a Web server.
- Servlets receive and respond to requests from Web clients.
- Need a **servlet container** (web container) to run servlets.

![Servlet Diagram]

**History**

- 1997: Sun released the Java Web Server and Java Servlet Developers Kit
- 1999: Sun introduced JavaServer Pages (JSPs)
- 2003: Java 2 Enterprise Edition (J2EE)
- 2000: NetBeans
  - open source IDE (Integrated Development Environment)
  - Java EE (Enterprise Edition)
  - Enterprise Java Beans (EJB), servlets, JSP pages, JAX-WS web services
  - Servlet engines (web containers): hosts for servlets and JSPs
    - Jakarta Tomcat by Apache
    - GlassFish
    - Sun’s Java System Application Server
    - BEA WebLogic
    - RedHat JBoss
    - IBM’s WebSphere

**Installing and Learning about NetBeans**

- Works on most platforms (Linux, Mac OS, Solaris, MS Windows)
- Install JDK 6 (Java SE Development Kit 6) from:
  

- Install NetBeans IDE 6.0 from:
  
  - Select to install both Tomcat and GlassFish

- To learn more about NetBeans:
  - The Help Contents in NetBeans Visual Designer (very useful)
  - Documentation about NetBeans Web applications:
    

  - Java Studio Creator Reference
    

  - The Java EE 5 Tutorial (similar to NetBeans)
    

  - The Java API
    

**The Servlet Interface**

- To implement this interface, you can write
  
  - a generic servlet that extends `javax.servlet.GenericServlet`
  - an HTTP servlet that extends `javax.servlet.http.HttpServlet`

- It defines methods to initialize/remove a servlet and to service requests
  
  Servlet life-cycle:
  
  - The servlet is constructed, then initialized with the `init()` method
  - Calls from clients to the service method are handled
  - The servlet is taken out of service, then destroyed with the `destroy()` method, then garbage collected and finalized

- Other methods:
  
  - `getServletConfig()`
  - `getServletInfo()`
**GenericServlet**

- Defines a generic, protocol-independent servlet
- Example:
  ```java
  import javax.servlet.Servlet;
  import javax.servlet.http.HttpServlet;
  
  public class MyServlet extends HttpServlet {
      public void service (HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
          response.setContentType("text/html");
          PrintWriter out = response.getWriter();
          out.println("<html>...<html>");
      }
  }
  ```
  - There are also default methods to initialize (init) and finalize (destroy) a servlet that can be overridden
  - To write an HTTP servlet for use on the Web, implement the HttpServlet interface instead

**HttpServlet**

- The HttpServlet interface extends the GenericServlet interface to handle GET/POST requests
- Example:
  ```java
  import javax.servlet.ServletException;
  import javax.servlet.http.HttpServlet;
  
  public class Hello extends HttpServlet {
      public void doGet (HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
          response.setContentType("text/html");
          PrintWriter out = response.getWriter();
          out.println("<html>...<html>");
      }
  }
  ```
  - doPost is similar

**Cookies**

- To read the cookies associated with the servlet request:
  ```java
  Cookie[] cookies = request.getCookies();
  ```
  - Cookie methods:
    ```java
    cookie.getName()
    cookie.getValue()
    ```
  - To create a new cookie:
    ```java
    Cookie cookie = new Cookie("myCookie","some-value");
    cookie.setPath(request.getServletContext().getContextPath());
    cookie.setMaxAge(-1);
    response.addCookie(cookie);
    ```

**HttpSession**

- Use `getParameter()` to access GET/POST parameters:
  ```java
  String value = request.getParameter("parameter-name");
  ```
- To get all parameter names:
  ```java
  Enumeration parameters = request.getParameterNames();
  ```
- Method `getSession()` returns the current session associated with this request, or if the request does not have a session, creates one:
  ```java
  HttpSession session = request.getSession();
  ```
  - HttpSession methods:
    ```java
    session.getId();
    ```
    - To get the names of all session attributes:
      ```java
      Enumeration attributes = session.getAttributeNames();
      ```
      - Given the name of a session attribute, get its value:
        ```java
        Object value = session.getAttribute("name");
        ```
      - Change the value of a session attribute:
        ```java
        session.setAttribute("name",value);
        ```
**ServletContext**

- Contains the objects common to all sessions
  - particular to the web application
  - its a location to share global information (eg, a database of sale items)
- To extract:
  ServletContext context = getServletContext();
- Methods:
  Enumeration attributes = context.getAttributeNames();
  Object value = context.getAttribute("name");
  context.setAttribute("name",value);

**The Directory Structure**

- The directory for the application MyApplication has structure:
  - MyApplication/: contains all static HTML and JSP files
  - MyApplication/WEB-INF/web.xml: the deployment descriptor
  - MyApplication/WEB-INF/classes/: contains the Java classes
  - MyApplication/WEB-INF/lib/: contains the JAR files
- The easiest way to deploy the application is to convert it to a WAR file using JAR. Inside directory MyApplication do:
  - jar cvf MyApplication.war ...
- **WAR**: Web Application Archive file
- Then, you can deploy the file MyApplication.war using the Tomcat manager
- If you use the NetBeans Visual Studio
  - it will create a default deployment descriptor
  - it will deploy your application automatically

**The Deployment Descriptor**

- It's the file web.xml in the WEB-INF directory
  ```xml
  <?xml version="1.0" encoding="ISO-8859-1"?>
  <web-app ...
  <display-name>Hello, World Application</display-name>
  <description>... </description>
  <servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>mypackage.Hello</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
  </web-app>
  ```
- After you deploy with Tomcat, to run it on browser use:
  - http://localhost:8080/hello/

**Web Components**

- **Java Server pages** (JSP) are text documents that execute as<br>  servlets but allow a more natural approach to creating web content
  - They contain two types of text:
    - static data, which can be expressed as HTML or XML, and
    - JSP elements, which determine how the page constructs dynamic content
  - JavaServer Pages **Standard Tag Library** (JSTL) encapsulates core<br>    functionality common to many JSP applications
    - iterator and conditional tags
    - tags for manipulating XML documents
    - tags for accessing databases
- **JavaServer Faces** (JSF) technology provides a user interface<br>  component framework for web applications. Components:
  - a GUI component framework
  - a flexible model for rendering components in HTML
  - JSF pages are translated to JSP pages (lazily)
  - Need library descriptor files in WEB-INF to deploy JSP pages
  - Tomcat's Jasper
JSP Example

```jsp
<%@ taglib url="http://java.sun.com/jstl/core" prefix="c" %>
<jsp:useBean id="date" class="java.util.Date" />
<html>
<head><title>JSP Example</title></head>
<body>
  <h2>Today's Date</h2>
  <c:out value="\$\{date\}" />
</body>
</html>
```

Java Beans

- **Java Beans** are Java classes that have properties (variables) and have get and set methods for accessing the properties
- ```java
package org.myserver;
class MyResult {
  String result;
  public String getResult () { return result; }
  public void setResult (String s) { result = s; }
}
```
- There are 4 Java Bean categories (scopes) used by JSP:
  - application (global objects)
  - session (session objects)
  - request (objects passing from servlet to servlet through requestDispatch)
  - page (local to the current page)

JSP Syntax

- **JSP expressions $\{\ldots\}$** retrieve the value of object properties
- for deferred evaluation use: #(\ldots)
- Variables are properties in a scope bean (the default scope is page)
- The custom tag `c:set` sets a variable:
  - ```jsp
  <c:set var="y" value="$\{x+1\}" />
  <c:set var="user" value="$\{user\}" scope="session"/>
  ```
- There are custom tags to do
  - iterations over a collection
    - ```jsp
    <c:forEach var="x" items="\ldots" /> ...
    ```
  - conditions:
    - ```jsp
    <c:if test="\ldots" /> ...
    ```
  - XML and SQL stuff
    - ```jsp
    <sql:query var="x" sql="select * from PUBLIC.books where id = \$id\"/>
    ```
  - To create/update/use a Java Bean object:
    - ```jsp
    <jsp:useBean id="MyResult" class="org.myserver" scope="application"/>
    <jsp:setProperty name="MyResult" property="result" value="$\{\ldots\}\"/>
    $\{MyResult.result\}
    ```

GET/POST Parameters and Cookies

- Use the param object associated with the Map.Entry bean
  - ```jsp
  <html>
  <head><title>Posted Data</title></head>
  <body>
  <h1>Posted Data</h1>
  <p><c:out value="$\{param\}" />
  </p>
  </body>
  </html>
  ```
- For cookies, use the cookie object
  - ```jsp
  <c:forEach var="c" items="$\{cookie\}"
  <p><c:out value="$\{c.key\}" />
  </p>
  ```
Database Connectivity

- Custom tags for SQL:
  ```xml
  <sql:transaction>
  <sql:update>
      insert into person values('John Smith','smith@domain.com')
  </sql:update>
  <sql:query var="result">
      select * from person
  </sql:query>
  </sql:transaction>
  
  <c:forEach var="row" items="$[result.rows]">
    <c:forEach var="col" items="$[row]">
      <c:out value="${col.key}"/>
      : <c:out value="${col.value}"/>
    </c:forEach>
  </c:forEach>
  ```

User-defined Custom Tags

- You can create your own custom tag, `ct`, in the namespace `mytags`, by providing a Java bean, the `tag handler` `CTag`:
  ```xml
  <mytags:ct name="x">some content</mytags:ct>
  ```

- Code in mypackage.tag:
  ```java
  import javax.servlet.jsp.tagext.*;
  public class CTag extends BodyTagSupport {
      String name;
      public int doStartTag() throws JspException {}
      public int doStartTag() throws JspException {
          JspWriter out = pageContext.getOut();
          String content = getBodyContent().getString().trim();
          out.println(content);
      }
  }
  ```

- To import mytags:
  ```xml
  <%@taglib url="mypackage.tags" prefix="mytags" %>
  ```

Scriptlets and JavaScript

- You can embed Java code fragments (called scriptlets) into the JSP pages:
  ```java
  %
  java-code
  %
  ```

  - Syntax: `<% java-code %>`
  - Not recommended because the application programming should be detached from web page content
  - Use custom tags instead

- You can include JavaScript code to be executed at client side:
  ```html
  <c:import url="/WEB-INF/javascript/client.js"/>
  <form name="myForm" onsubmit="popup()">
  ```

  - NetBeans provides a library of Ajax JavaScript templates

JavaServer Faces

- Based on a special tag library

- Pages are created using User Interface Components:
  - they represent common user interface components, such as buttons, output fields, input fields, etc
  - they are organized in a tree-like structure
  - they are separated from `renderers` (which map to HTML)
    - Renderers can be redefined (in render kit)

- The event and listener model lets developers register listeners on components to handle events:
  - `Action event`: An action event is fired when a user does something, such as pressing a button or clicking a hyperlink
  - `Value-change event`: A value-change event is fired when a user changes a component's value, such as by clicking a checkbox or entering a value in a text field

- You can define a listener to an event as a `backing bean` method

- You can have multiple registered listeners (observers) to an event
Navigation Model

- Must define page navigation separately
  - Navigation is a set of rules for choosing the next page to be displayed after a button or hyperlink is clicked
  - Instead of a URL, use a tag name
  - Tag names are mapped to URLs in page navigation configuration file
    ```xml
    <navigation-rule>
      <from-view-id>/greeting.jsp</from-view-id>
      <from-outcome>success</from-outcome>
      <to-view-id>/response.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-view-id>/greeting.jsp</from-view-id>
      <from-outcome>failure</from-outcome>
      <to-view-id>/error.jsp</to-view-id>
    </navigation-case>
    </navigation-rule>
    ```
  - They can be returned from event listeners
    ```java
    public String button1_action() {
      return "success";
    }
    ```

- NetBeans provides a GUI to draw navigation graphs

Backing Beans

- These are the back-end objects that provide the User Interface functionality
  - They can validate a component’s data
  - They can handle an event fired by a component
  - They can perform processing to determine the next page to which the application must navigate

- Types:
  - **User Interface Backing Beans**: page and page fragment beans
    - contains everything necessary to manage the server-side logic for a web page
  - Component properties and events

- Data backing beans:
  - **Application beans** are created at the application level and available to all users, sessions, and requests
  - **Session beans** are created and are available at the user session level
  - **Request beans** are created for each request and are only available during the request. They are useful for passing information between two pages

Data Providers and RowSets

- **Data providers** provide a simple interface to various data sources
- The **RowSet** interface provides JDBC code that reads and updates data from a data provider (eg, a database table)
  - Extends the standard JDBC ResultSet Interface
  - ... but can be used as a JavaBeans component
    - supports JavaBeans events, allowing other components in an application to be notified when an event occurs on a RowSet, such as a change in its value

- Can have parameter placeholders:
  ```java
  rs.setCommand("select name, lname from CUSTOMER" +
              "where credit > ? and region = ? ");
  ```
- Which can be instantiated and executed later:
  ```java
  rs.setInt(1, 5000);
  rs.setString(2, "West");
  rs.execute();
  ```

CachedRowSet

- A **CachedRowSet** object is a container that caches database rows in memory
  - It is scrollable, updatable, and serializable
  - It extends the RowSet Interface

- When you drag and drop a database table to a web page, you create a data provider along with the CachedRowSet for the table
  - eg, if you drop the table CUSTOMER, you add the methods
    ```java
    customerRowSet = (CachedRowSet)session.getAttribute("CUSTOMER");
    customerDataProvider = (DataBean)session.getAttribute("DataProvider");
    ```
  - (plus some JDBC code in the session_init() method to connect to the DB)
A Simple Application

- Create a new project, called MyProject:
  - Open the NetBeans Visual Designer and click on “New Project”
  - On Name and Location, put Project Name: “MyProject”, Server: “Tomcat 6.0”, push “Next”
- On the Projects window, expand MyProject and “Web Pages”
  - Double-click to rename Page1.jsp to Main.jsp
- Create a new Database Customer:
  - On the Tools menu, select “Java DB Database”/“Create Database ...”
  - Put Database Name: customer, and pick a User Name and Password
  - On the Services window, expand Databases, right click on the jdbc:derby driver for customer and select “Connect ...”
  - Left-click on the customer driver to expand it
  - Right-click on Tables and select “Create Table ...”

The Main Page

- ... then create the table Person:

![Image of a form with fields: username, password, email, full name]

- From the Palette, drag and drop into the Main Design window
  - Label, Text Field, Password, and Button components as follows:

  ```
  username: 
  password: **********
  full name: 
  email: 
  ```

The Main Page (cont.)

- Click on the Insert button and, in the Properties menu (right bottom), change its id to insertButton
- Go to the Main JavaBean by clicking on Java on the Main window and add the following properties inside the class Main:
  - String newUsername;
  - String newPassword;
  - String newFullname;
  - String newEmail;
- Right click and select “Refactor”/“Encapsulate Fields ...”
- Click on boxes to create getters and setters for the four new properties
- Push Refactor.
- Go back to the Main.jsp Design
- Right-click on the Text Field (the rectangle) next to “username:” and “Select Bind to Data ...”, then “Bind to Object”, then select newUsername
- Do the similar thing for the other Text Field rectangles

Inserting New Data

- Drag and drop the PERSON table from the Servers menu to the Main Design window (don't drop it on a component)
- Notice that there is now a personDataProvider in the Navigator menu
- Double-click on the Insert button to add code

```java
public String insertButton_action() {
  try {
    SessionBean1 session = getSessionFactory1();
    CachedRowSet rs = session.getPersonRowSet();
    rs.moveToInsertRow();
    rs.updateString1(newUsername);
    rs.updateString2(newPassword);
    rs.updateString3(newFullname);
    rs.updateString4(newEmail);
    rs.insertRow();
    rs.acceptChanges();
    newUsername = ""; newPassword = ""; newFullname = ""; newEmail = "";
  } catch (Exception ex) {
    throw new FacesException(ex);
  }
  return "main";
}
```
Displaying Data in a Table

- Go back to the Main Design window and drag and drop a Table from the Palette into the Design window
- Drag and drop the PERSON table from the Services window onto the header of the new Table component in the Design Window
  - On the popup menu, select “Use personRowSet”, press OK
  - Right-click on the Table header and change the title to Person

Testing

- Right-click on the Design window and choose “Page Navigation”
  - Push on the plus sign in Main.jsp to see its buttons
  - Drag and drop the insertButton link into Main.jsp forming a loop
  - Select the loop line, right-click, choose Rename..., and rename case1 to main
    - Select the insertButton_action() returns “main”, which loops back
  - Go back to the Main Design window and save the project
  - Push “Run Main Project” to build, install, and run your program
    - It will run using Tomcat on a web browser at the URL: http://localhost:8080/MyProject/
    - Insert few data and remember one username/password combination to use it for log-in

The Login Page

- Click on “New File” and Select “JavaServer Faces” and “Visual Web JSF Design”, click Next, put File Name: Login, push Finish
- On the Login Design window, drag and drop the following

```
Please Login
```

- Click on the login button and change its id to loginButton in the Properties window
- Go to the Login class by clicking on Java and add the properties
  ```java
  String loginUsername;
  String loginPassword;
  ```
  - Use the refactor as before to add getter/setter methods
  - Go back to the design and bind the username/password to these new properties (as before using “Bind to an Object”)

Login using the Database

- Drag and drop the PERSON table from the Services window into the Login design window
  - Choose Create SessionBean1 with personRowSet1
  - On the Navigator menu, right-click on personRowset1 in SessionBean1 and choose “Edit SQL Statement”
  - Use the SQL editor to add query criteria (parameters) and construct the SQL query
    ```sql
    SELECT ALL ADMIN.PERSON.USERNAME
    FROM ADMIN.PERSON
    WHERE ADMIN.PERSON.USERNAME = ?
    AND ADMIN.PERSON.PASSWORD = ?
    ```
  - Right-click on Login.jsp in the Projects window and select “Set as Start Page”
The Login Action

- Double-click on login button to edit the action:
  ```java
  public String loginButton_action() {
      try {
          SessionBean1 session = getSessionBean1();
          CachedRowSet rs = session.getPersonRowSet1();
          rs.setObject1(loginUsername);
          rs.setObject2(loginPassword);
          rs.executeQuery();
          loginUsername = ""; loginPassword = "";
          if (rs.first())
              return "main";
      } catch (Exception ex) {
          throw new FacesException(ex);
      }
      return "login";
  }
  ```

Navigation

- Right-click on the Login design page and select Page Navigation
  - Draw the following navigation (based on the loginButton action)

- Save and run the project again
- Login using one of the Person accounts
- Question: if we add a logout button in Main, what would be its action?