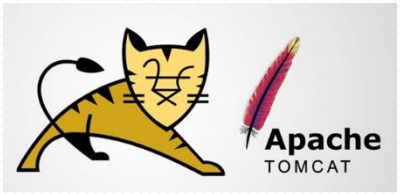
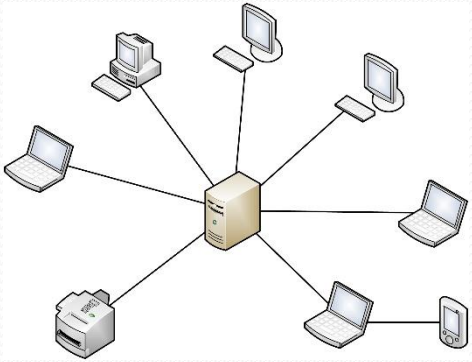
Server-side Programming and Client-Server Communication





Overview

- Java Servlet
- Java Server Page (JSP)
- **JDBC**
- HTTP session

JDBC

- JDBC (Java database connectivity) is used for accessing databases from Java applications
  - This is not specific for servlets
- Information is transferred from database to objects and vice-versa
  - databases optimised for searching/indexing
  - objects optimised for engineering/flexibility
- Another similar API by Microsoft called ODBC
- Different databases such as Oracle, MySQL, PostgreSQL, SQLite etc

Seven steps

- Load the driver
- Define the connection string
- Establish the connection
- Create a statement object
- Execute a query
- Process the result
- Close the connection

Packages to import

- In order to connect to the MySQL database from java, import the following packages:

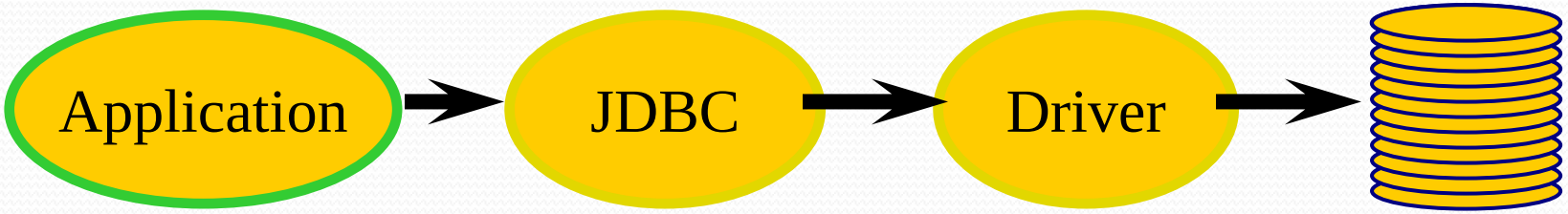
java.sql.*;

- Basic database features - such as executing SQL statements

javax.sql.*

- Advanced database features, such as connection pooling, scrollable ResultSet

JDBC architecture



- Java code calls JDBC library
- JDBC loads a driver
- Driver talks to a particular database
- Can have more than one driver
  - Required if we have more than one database
- Ideal: can change database engines without changing any application code

Initial steps

- Loading the driver using the Java statement:

  Class.forName("com.mysql.jdbc.Driver");

  - creates an instance of the driver
  - registers the driver with the DriverManager
- Establishing the connection using the Java statement:
  Connection connect = DriverManager.getConnection(*"jdbc:mysql://localhost/youDatabaseName","DB_USERNAME","YOUR_PASSWORD"*);
  - creates an instance of the connection
  - registers the connection with the DriverManager

Code example: connecting to MySQL

```
…
String host="localhost";
//Dept MySQL server “mysql.mcscw3.le.ac.uk”
String database="CO7102";
String user="root";
String password="helloworld";

Class.forName("com.mysql.jdbc.Driver");

String conn_string="jdbc:mysql://"+host+"/"+database;
// setup the connection with the DB.

Connection connect =
    DriverManager.getConnection(conn_string,user,passwd);
….
```

Statement Objects for executing SQL queries

- **Statement createStatement()**
  - Simple queries without parameters
  - vulnerable to SQL injection attacks
- **PreparedStatement prepareStatement(String sql)**
  - Queries with parameters – same statement can be executed multiple times with different parameters
  - Prevents SQL injection attacks by separating data from executable code
- **CallableStatement prepareCall(String sql)**
  - Stored procedures

Connection methods

- **Statement createStatement()**
  - returns a new Statement object
- **PreparedStatement prepareStatement(String sql)**
  - returns a new PreparedStatement object
- **CallableStatement prepareCall(String sql)**
  - returns a new CallableStatement object
  - For SQL stored procedures

- Why all these different kinds of statements?
  - Optimization.

JDBC Example

- MySQL database (Student.sql)

```
CREATE TABLE `Student` (
  `ID` int(11) NOT NULL,
  `Name` varchar(45) NOT NULL,
  `Address` text,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO `Student` VALUES (1,'John Smith','G1,
Computer Science Building, University Road,
Leicester,  LE1 7RH'),(2,'Jane Borwn','23 St Leonards
Rd, Leicester, LE2 1WS');
```

Querying DB with createStatement()

- Statements are used for queries that are only issued once.
- The **executeQuery** method returns a **ResultSet** object representing the query result.

```
String sql="SELECT * from Student”;

Statement statement = connect.createStatement();

ResultSet resultSet = statement.executeQuery(sql);
```

Updating DB with createStatement()

- **executeUpdate** is used for data manipulation:
  - insert, delete, update, create table, etc. (anything other than querying!)
- **executeUpdate** returns the number of rows modified.

```
String sql=
    "UPDATE Student SET Address='23 St Leonards    Rd,
Leicester, LE2 1WS' WHERE id='2'";

Statement statement = connect.createStatement();
statement.executeUpdate(sql);
```

Querying with preparedStatement()

- Prepared Statements are used for queries that are executed many times.
  - They are parsed only once.
  - **setString(i, value), setInt(i, value)**, etc. set the i-th question mark to the given value

String sql="**SELECT** * from Student **WHERE** ID=?";

PreparedStatement pstmt = connect.prepareStatement(sql);
pstmt.setInt(1,studentID);
ResultSet rs = pstmt.executeQuery();

Bypassing authentication with createStatement()

- What is the security risk of the following statement?

```
…
String name= req.getParameter(“name”);
String pass= req.getParameter(“pass”);
Statement stmt =  con.createStatement( );
String SQL= “SELECT * FROM User WHERE” +
                “User=‘”+name+”’ AND Password=‘” +pass+”’”;
ResultSet rs =  stmt.executeQuery(sql);
//Checking password
```

What if user entered **OR '1’=‘1’;--** Into SQL query?

SQL statement will become

SELECT * FROM User WHERE User=‘name’ OR ‘1’=‘1’ ; -- AND Password=‘pass’;

Searching DB with prepareStatement()

```
String sql="SELECT * from Student WHERE ID=?";

PreparedStatement pstmt = connect.prepareStatement(sql);
pstmt.setInt(1,studentID);

ResultSet rs = pstmt.executeQuery();

while(rs.next()){
     String name=rs.getString("Name");
     String address=rs.getString("Address");
    //…
}
```

ResultSet

- A **ResultSet** provides access to a table of data generated by executing a Statement
- Only one **ResultSet** per Statement can be open at once
- A **ResultSet** maintains a cursor pointing to its current row of data
  - '**next()**' method moves the cursor to the next row
  - **‘previous()**' method moves the cursor to the previous row
  - **‘first()’** method moves the cursor to the first row
  - **‘last()**' method moves the cursor to the last row
  - **‘absolute (int row)**' method moves the cursor to a specific row
  - ‘**relative(int rows)**' method moves the cursor to a specific number of rows

ResultSet methods (1)

- **boolean next()**
  - activates the next row, the first call to next() activates the first row
  - returns false if there are no more rows
- **void close()**
  - disposes of the ResultSet
  - allows you to re-use the Statement that created it
  - automatically called by most Statement methods
- **getType(int columnIndex)**
  - retrieve the data type of a specific column
  - First column is indexed as 1 (not 0)
- **getType(String columnName)**
  - retrieve the data type of a specific column by its name
- **int findColumn(String columnName)**
  - looks up column index based on a given column name

ResultSet methods (2)

- String getString(int columnIndex)
- boolean getBoolean(int columnIndex)
- byte getByte(int columnIndex)
- short getShort(int columnIndex)
- int getInt(int columnIndex)
- long getLong(int columnIndex)
- float getFloat(int columnIndex)
- double getDouble(int columnIndex)
- Date getDate(int columnIndex)
- Time getTime(int columnIndex)
- Timestamp getTimestamp(int columnIndex)

http://docs.oracle.com/javase/8/docs/api/java/sql/ResultSet.html

Cleaning up

- Remember to close the **Connections, Statements, PreparedStatements** and **ResultSets**

```
con.close();
stmt.close();
pstmt.close();
rs.close()
```

- Or you can use Java's new **try-with-resource** statement. The try-with-resources statement ensures that each resource is closed at the end of the statement.

```
try (ResultSet rs = pstmt.executeQuery();){
    while(rs.next()){ … }
} catch(SQLException ex){…}
```

Printing ResultSet – code example

```
public void preparedStatementSearch(int studentID){

try( Connection connect = getConnection();
     String sql="SELECT * from Student WHERE ID=?”;)
     {
     PreparedStatement pstmt = connect.prepareStatement(sql);
     pstmt.setInt(1,studentID);

ResultSet rs = pstmt.executeQuery();
while(rs.next()){
          String name=rs.getString("Name");
          String address=rs.getString("Address");
          System.out.println("name:"+name+"
address:"+address);
}
          }catch(Exception ex){ ex.printStackTrace();}
```

Java 7's try-with-resources statement

Miscellaneous (1)

- Exceptions
  - **SQLException** – a checked exception
  - Checked exceptions are checked at compile time and must be handled in a try catch block
- Transactions
  - Transaction = more than one statement which must all succeed (or all fail) together
  - If one fails, the system must reverse all previous actions
  - **COMMIT** = complete transaction
  - **ROLLBACK** = abort

Miscellaneous (2)

- Transactions are not explicitly opened and closed
- Two ways of handling transactions
  - JTA (Java Transaction API)
  - **AutoCommit** feature in JDBC
    - if **AutoCommit** is true, then every statement is automatically committed
    - if **AutoCommit** is false, then multiple statements are executed as one single transaction
    - Default: true

AutoCommit

```
connection.setAutoCommit(boolean val)
```

- If you set **AutoCommit** to false, you must explicitly commit or rollback the transaction using **Connection.commit()** and **Connection.rollback()**
- Note: DDL statements (CREATE, DROP TABLE) in a transaction may be ignored or may cause a commit to occur. The behavior is DBMS dependent

Overview

- Java Servlet
- Java Server Page (JSP)
- JDBC
- **HTTP session**

State?

- Recall that HTTP is stateless
  - Once a webserver has dealt with a request the connection vanishes
  - The server cannot recognize that a sequence of requests is from the same client
- Need a way to store data between HTTP requests.
  - How to passing data between servlets?
  - How to recognise requests from same user?
  - How to avoid unauthorised access to a certain page?

State?

- Recall that HTTP is stateless
  - Once a webserver has dealt with a request the connection vanishes
  - The server cannot recognize that a sequence of requests is from the same client
- Need a way to store data between HTTP requests.
  - How to passing data between servlets?
  - How to recognise requests from same user?
  - How to avoid unauthorised access to a certain page?

Session tracking - Possible solutions (1)

- Possible solutions
  - Hidden form fields

    <input type="hidden" name="session" value="…">
    - The web server cannot tell the difference between a user entered value and a hidden form field value
  - URL rewriting
    - http://host/path/file.html?sessionid=455hh
    - Must rewrite all URLs

Session tracking - Possible solutions (2)

- Cookies
  - Cookies are client-side files. A cookie is a short text string storing in user's browser
  - Persistent cookies can remain on your device for a longer period (depending on the lifetime of the specific cookie)
  - User can disable cookies
- HTTP sessions
  - Sessions are server-side files that contain user information
  - A session is available as long as the browser is opened.
  - User cannot disable sessions.

Tracking with HTTP session

- A session is a logical link between pages requests by the same user
- Every user has a **HttpSession** object to store and retrieve user information
- Per user per browser
- Can be shared between pages such as servlet and JSP
- Can be implemented using cookies/URL rewriting

Using HTTP session

- Retrieve the user's session: (from a request)
  - **HttpSession session= request.getSession()**
    - Obtain current session or create a new one if no valid session
  - **HttpSession session= request.getSession(boolean create)**
    - Check session (if exists)
      - True: Obtain current session
      - False: Obtain current session
    - Check session (if does not exist)
      - True: Create a new session
      - False: Do not create a new session and return null
- Identify if a session is newly created or not

  session.isNew()
  - returns true if the session is newly created
  - returns false if the session already exists

Using HTTP session

- Store data to a session

  session.setAttribute(String name, Object value)

  - replaces any object that is bound in the session and has the same name
- Retrieve data from a session

  session.getAttribute(String name)

  - returns null if no object is bound to the name
- Terminate a session

  session.invalidate()

Using HTTP session

- With **HttpSession**
  - There is a single session per user, per session.
  - Different servlets will **get the same HttpSession object**, when calling getSession on different HTTPServletRequest objects during the same session

Forward vs sendRedirect

- **forward(request, response)**
  - forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the same server
  - URL remains unchanged
- **sendRedirect(String location)**
  - 302 HTTP status code is generated
  - requires extra communication on part of the client – new request and objects are created
  - ends up with a different URL in the browser

Further reading:
https://tomcat.apache.org/tomcat-8.0-doc/servletapi/javax/servlet/http/HttpServletResponse.html

Passing data on

- Different ways to set parameters for the forwarded servlet or JSP to see
  - Data that will be used only for one request:

    request.setAttribute("key", value);
  - Data will be used for one client (multiple requests):

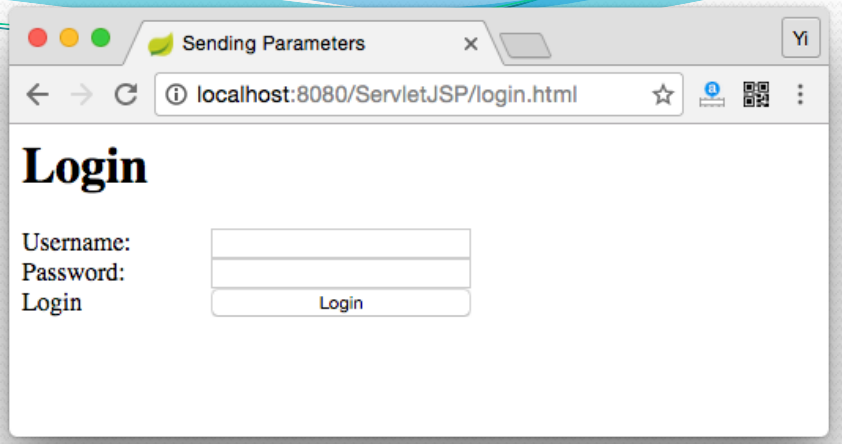
    session.setAttribute("key", value);
  - Data that will be used in the future for any client

    ServletContext context =
    request.getSession().getServletContext();

    context.setAttribute("key", value);

A Complete Example (1)

- User login page *(login.html)*




```
<h1>Login</h1>
<form action="./servlets/Login" method="GET">
<div>
    <label for="name">Username:</label>
        <input type=”text" name="name"/><br/>
    <label for="pass">Password:</label>
        <input type ="password" name="pass"/><br/>
    <label for="submit">Login</label>
        <input type=”submit" name="submit" value="Login">
</div>
</form>
```

A Complete Example (2)

- Servlet for user authentication *(Login.java)*

```
//doGet method in Login.java
..
String name=req.getParameter("name");
String pass=req.getParameter("pass");
UserVerification dbOperator=new UserVerification();
User u=dbOperator.checkUser(name, pass);
HttpSession se=req.getSession();

if(u!=null){
    se.setAttribute("User",u);
    res.sendRedirect("../user.jsp");
}else{
     res.sendRedirect("../error.jsp?errorid=1");
}
…
```

A Complete Example (3)

- User page *(user.jsp)*

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" import="uk.ac.le.cs.CO3098.bean.*"%><%
HttpSession se=request.getSession();
User u=null;
if(se.getAttribute("User")!=null){ u=(User)se.getAttribute("User");}
  else{response.sendRedirect("error.jsp?errorid=2");}
if(u!=null){
%>
<html>
<head><title>User page</title></head><body>
<h1>Hello</h1>
<div>
Welcome, <%= u.getFullname() %>! <br/>
<a href="./servlets/Logout">Logout</a>
</div>
</body></html>
<%}%>
```

A Complete Example (3)

- User page *(user.jsp)*

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" import="uk.ac.le.cs.CO3098.bean.*"%><%
HttpSession se=request.getSession();
User u=null;
if(se.getAttribute("User")!=null){ u=(User)se.getAttribute("User");}
  else{response.sendRedirect("error.jsp?errorid=2");}
if(u!=null){
%>
<html>
<head><title>User page</title></head><body>
<h1>Hello</h1>
<div>
Welcome, <%= u.getFullname() %>! <br/>
<a href="./servlets/Logout">Logout</a>
</div>
</body></html>
<%}%>
```

A Complete Example (4)

- Servlet for logout *(Logout.java)*

```
//doGet method in Logout.java
HttpSession se=req.getSession();
se.removeAttribute("User");
res.sendRedirect("../login.html");
```

A Complete Example (5)

- Error page *(error.jsp)*

```
..
<h1>Error</h1>
<%
String errorMsg="Access denied";
String errorid=request.getParameter("errorid");
if(errorid!=null){
   if(errorid.equals("1")){errorMsg+= " - Wrong password.”;}
   else if(errorid.equals("2")){errorMsg+=" - Session expired.”;}
   else{ errorMsg+=" - You are not authorized to access this
page.”;}
}else{errorMsg+=" - An unexpected error has occurred.";
}%>
<div>
        <label><%=errorMsg%></label>
        <label>Please <a href="login.html">login</a></label>
</div>
…
```

A Complete Example (6)

- Checking Database *(UserVerification.java)*

```
public User checkUser(String user,String password){

String sql="SELECT * from User WHERE UserName=? AND
PasswordHash=?";
User u=null;

try( Connection connect = getConnection();
    PreparedStatement pstmt = connect.prepareStatement(sql);){

    pstmt.setString(1,user);
    pstmt.setString(2,MD5HashGenerator.getMD5Hash(password
    ));
     //see next page
```

A Complete Example (6) cont.

- Java bean for database *(UserVerification.java)*

```
//cont.
try (ResultSet rs = pstmt.executeQuery();){
    while(rs.next()){
        String uname=rs.getString("UserName");
        String fname=rs.getString("FullName");
        String pass=rs.getString("PasswordHash");
        u=new User(uname,fname,pass);
        break;
        }
}catch(SQLException ex){ex.printStackTrace(); }
return u;
}
```

Complete source code available on blackboard

Cookies

- Server creates cookies and sends it to a client then it is stored on the client's browser
- On subsequent requests, the client sends the relevant cookies back to server
- Good to retain information about the user through repeating visits to a website such as login information
- Used for
  - Identifying a user during an e-commerce (or other) session
  - Avoiding user-name and password
  - Customizing a site according to user preference – language or theme
  - Focusing advertising

Problems

- Some browser do not support cookies
  - Tor, Brave
- Browsers that allow users to disable cookies
  - Chrome, Safari and Firefox
- Cookies cannot be larger than 4kb
- Only 20 cookies per domain
- No more than 300 cookies stored at client
- Expiry is up to browser (even though you can specify a value)

Servlets and cookies

- Cookies are represented by the class
  javax.servlet.http.Cookie
- A cookie object can be created by the cookie constructor
- The name and the value of the constructor should not include
  []() = , “ / ? @ : ;
- You create cookies and then add them to the HttpServletResponse
  response.addCookie(cookie)
- You can retrieve cookies from the HttpServletRequest
  Cookie[] coockie = request.getCookies()

Properties of cookies

- **getDomain() / setDomain()**
  - The domain for which the cookie belongs
- **getMaxAge() / setMaxAge()**
  - Positive value = How long (in seconds) will the cookie last
  - Negative value = per browser session
  - Zero – delete immediately
- **getName()**
  - The name of the cookie

Properties of cookies

- **getPath() / setPath()**
  - Defines the path for which the cookie relates
  - Cookie.setPath("/") means that all the pages on host will get the cookie
  - Default: Entire host
- **getSecure() / setSecure()**
  - Should the cookie be sent with SSL secured line
- **getValue() / setValue()**
  - The value that the cookie holds

An example: html

```
<html> <head>
<title>Login Page</title>
</head>

<body>
<h1>Logon to My Site</h1>
    <form action="servlets/WelcomeBack">
          Your Name:
         <input type="text" name="username">
         <input type="submit">
     </form>
</body>
</html>
```

An example: servlet (1)

```
import java.io.*; import javax.servlet.*;
import javax.servlet.http.*;

public class WelcomeBack extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse
   res)  throws ServletException, IOException   {
          String user = req.getParameter("username");
          if (user == null) {
                    Cookie[] cookies = req.getCookies();
                    for (int i = 0 ; i < cookies.length ; i++) {
                    if (cookies[i].getName().equals("username"))
                                        user = cookies[i].getValue();
                    }
          } else { res.addCookie(new Cookie("username", user)); }
```

An example: servlet (2)

.. for private study ..

```
        if (user != null) {
                res.setContentType("text/html");
                PrintWriter out = res.getWriter();
                out.println("<html><body>Welcome Back" +
                        user + "</html></body>");
        } else {
                res.sendRedirect("../login.html");
        }
    }
}
```