Bộ môn: Kỹ Thuật Phần Mềm

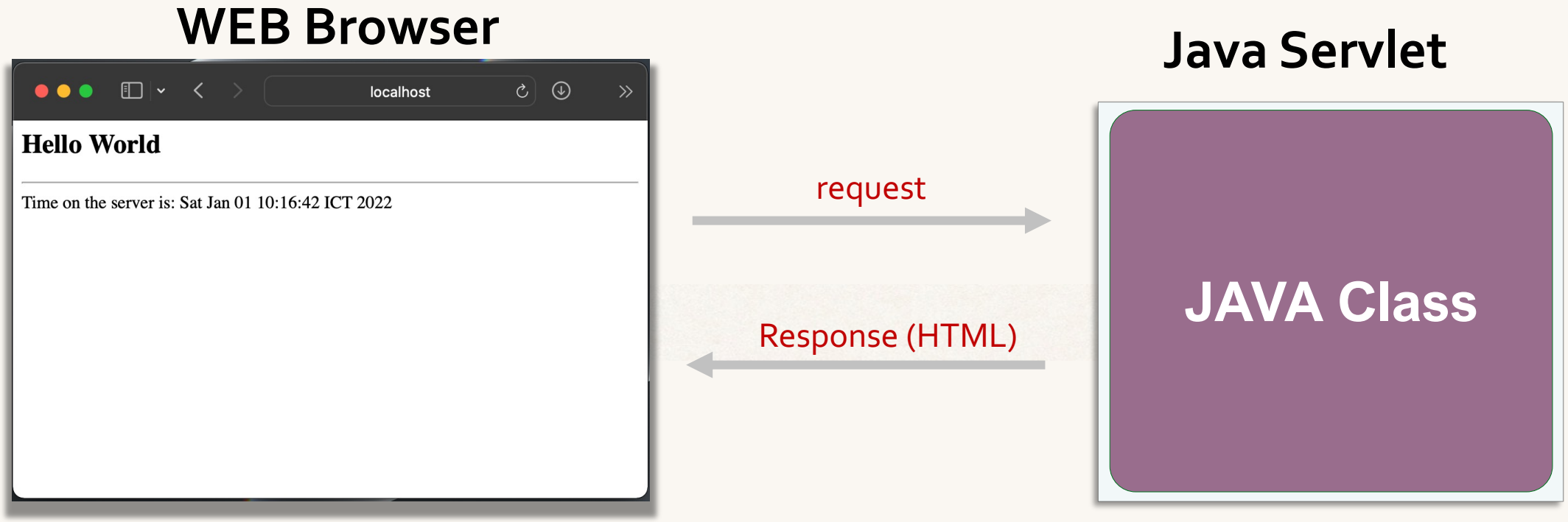
Lập trình WWW *Java*

Servlets Overview

What Are Servlets?

- Java class that is processed on the server
- Java class generates HTML that is returned to browser
- Can read HTML form data, use cookies and sessions etc...
- At a high-level, similar functionality to JSPs

What Are Servlets?



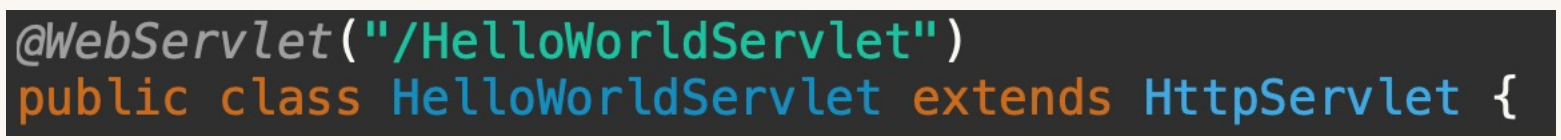
What Are Servlets - Example

```java
@WebServlet("/HelloWorldServlet")
public class HelloWorldServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    public HelloWorldServlet() {
        super();}
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // Step 1: set the content type
                response.setContentType("text/html");

                // Step 2: get the printwriter
                PrintWriter out = response.getWriter();

                // Step 3: generate HTML content
                out.println("<html><body>");

                out.println("<h2>Hello World</h2>");
                out.println("<hr>");
                out.println("Time on the server is: " + new java.util.Date());

                out.println("</body></html>");
    }
```

Access the Servlet

Use the path specified in WebServlet annotation

http://localhost:8080/<projectName>/HelloWorldServlet



```
@WebServlet("/HelloWorldServlet")
public class HelloWorldServlet extends HttpServlet {
```

Comparing JSP and Servlets

JSP

- HTML file with .jsp extension
- Contains static HTML
- JSP to generate HTML
- Has built-in JSP objects

Servlets

- Java class file
- Generate all HTML
- More steps to access web objects

Which One?

- Can use either one for building Java web apps ...
  - Build entire site using only Servlets ...

  or

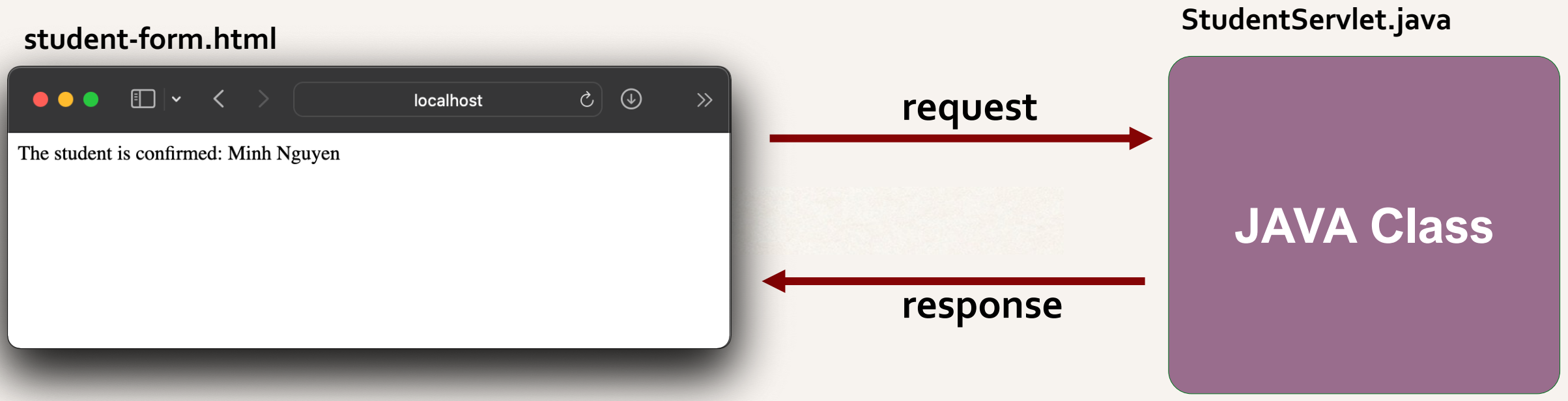
  - Build entire site using only JSPs

Best Practice

- Integrate them both together!
  - ✓ Servlet does the business logic
  - ✓ JSP handles the presentation view
- Model-View-Controller (MVC) Design Pattern

Reading HTML Form Data with Servlets

HTTP Request / Response

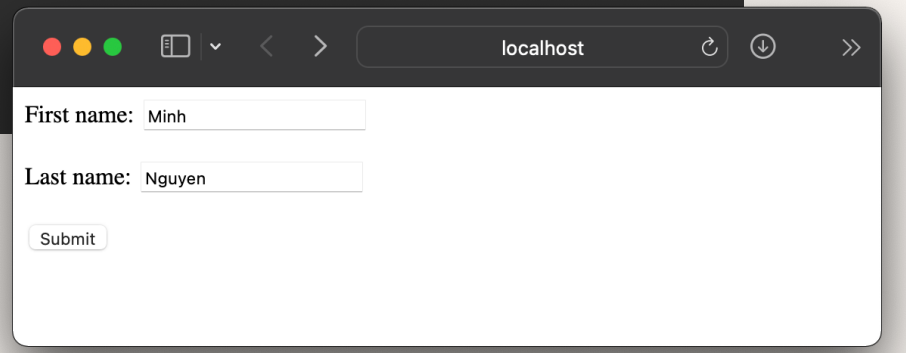


Step 1: Building HTML Form

Servlet class name

Calls to doGet method

```
<body>
<form action="StudentServlet" method="GET">
    First name: <input type="text" name="firstName" />
    <br/><br/>
    Last name: <input type="text" name="lastName" />
    <br/><br/>
    <input type="submit" value="Submit" />
</form>
</body>
```



Form GET method calls Servlet doGet() method

```
<body>
<form action="StudentServlet" method="GET">
```

```
@WebServlet("/StudentServlet")
public class StudentServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
```

Step 2: Reading Form Data with Servlet

```
First name: <input type="text" name="firstName" />
<br/><br/>
Last name: <input type="text" name="lastName" />
```

```
out.println("The student is confirmed: "
            + request.getParameter("firstName") + " "
            + request.getParameter("lastName"));
```

Form POST method calls Servlet doPost() method

```
<body>
<form action="StudentServlet" method="POST">
```

```
@WebServlet("/StudentServlet")
public class StudentServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
```

Servlet Configuration Parameters

Servlet Configuration Parameters

- Your web app can make use of configuration parameters
- Located in standard file: WEB-INF/web.xml

Deployment Descriptor: web.xml

```
<context-param>
    <param-name>max-shopping-cart-size</param-name>
    <param-value>99</param-value>
</context-param>

<context-param>
    <param-name>project-team-name</param-name>
    <param-value>The Coding Hero </param-value>
</context-param>
```

Reading Configuration Parameters

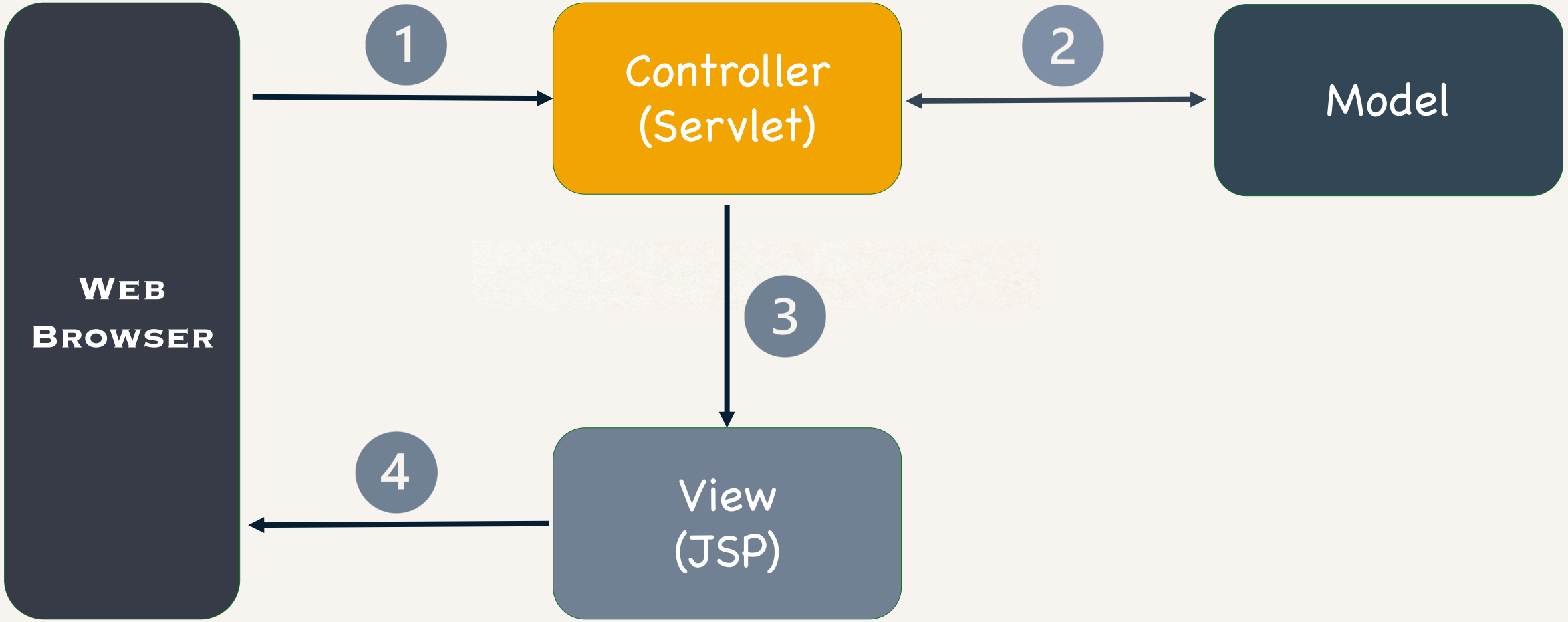
```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    ServletContext context = getServletContext();

    String maxCartSize = context.getInitParameter("max-shopping-cart-size");
    String teamName = context.getInitParameter("project-team-name");

    out.println("<html><body>");

    out.println("Max cart: " + maxCartSize);
    out.println("<br/><br/>");
    out.println("Team name: " + teamName);

    out.println("</body></html>");
}
```

MVC with Servlets and JSP

Best Practice

- Integrate them both together!
  - ✓ Servlet does the business logic
  - ✓ JSP handles the presentation view
- Model-View-Controller (MVC) Design Pattern

Model-View-Controller (MVC)



Benefits of MVC

- Minimizes HTML code in Servlet
  - no more: out.println(...) in Servlet code
- Minimize Java business logic in JSPs
  - no more large scriptlets in JSP code

Servlet Can Call a JSP

Servlet can call JSP using a request dispatcher

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    // Step 0: Add data
    String[] students = {"Xuan Xuan", "Ha Ha", "Thu Thu", "Dong Dong"};
    request.setAttribute("student_list", students);
    // Step 1: get request dispatcher
    RequestDispatcher dispatcher =
                request.getRequestDispatcher("/view_students.jsp");
    // Step 2: forward the request to JSP
    dispatcher.forward(request, response);
}
```

Sending Data to JSP

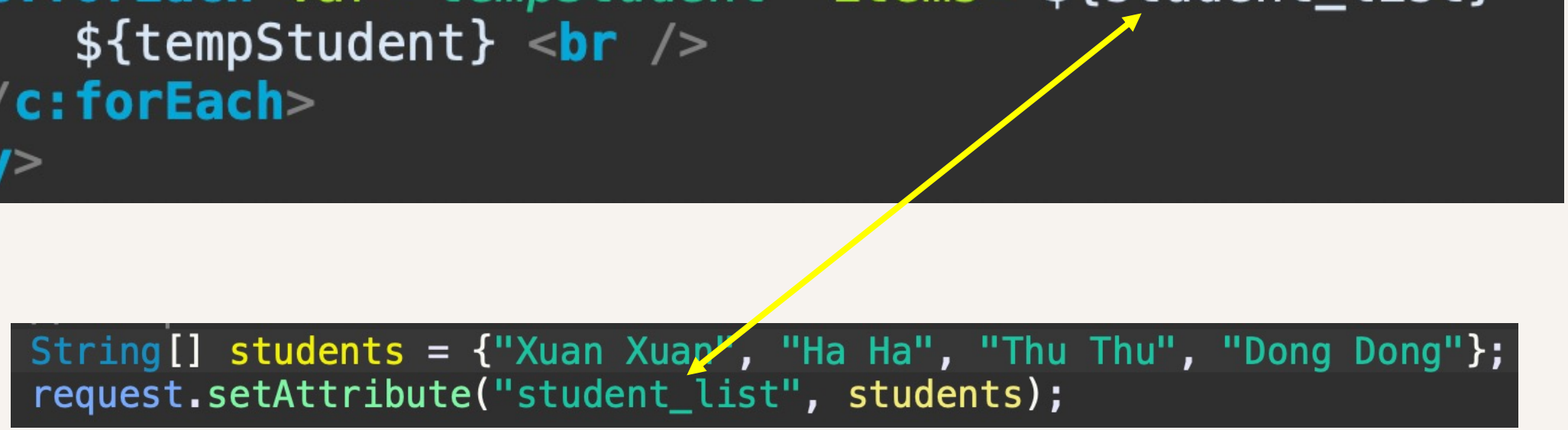
Servlet can add data to request object

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    // Step 0: Add data
    String[] students = {"Xuan Xuan", "Ha Ha", "Thu Thu", "Dong Dong"};
    request.setAttribute("student_list", students);
    // Step 1: get request dispatcher
    RequestDispatcher dispatcher =
                request.getRequestDispatcher("/view_students.jsp");
    // Step 2: forward the request to JSP
    dispatcher.forward(request, response);
}
```

JSP page to view data

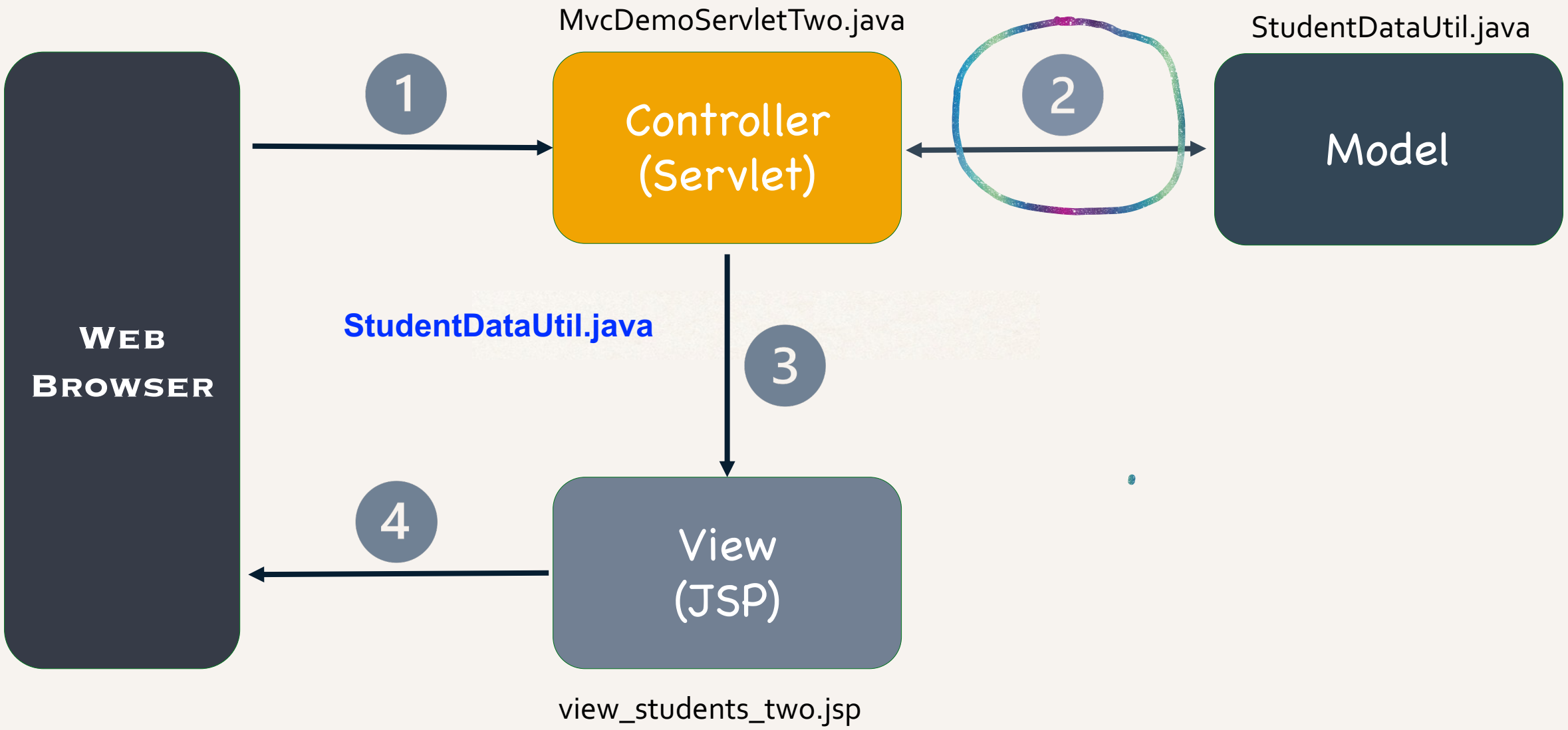
JSP use JSTL to access data

```
<body>
    <c:forEach var="tempStudent" items="${student_list}">
        ${tempStudent} <br />
    </c:forEach>
</body>
```



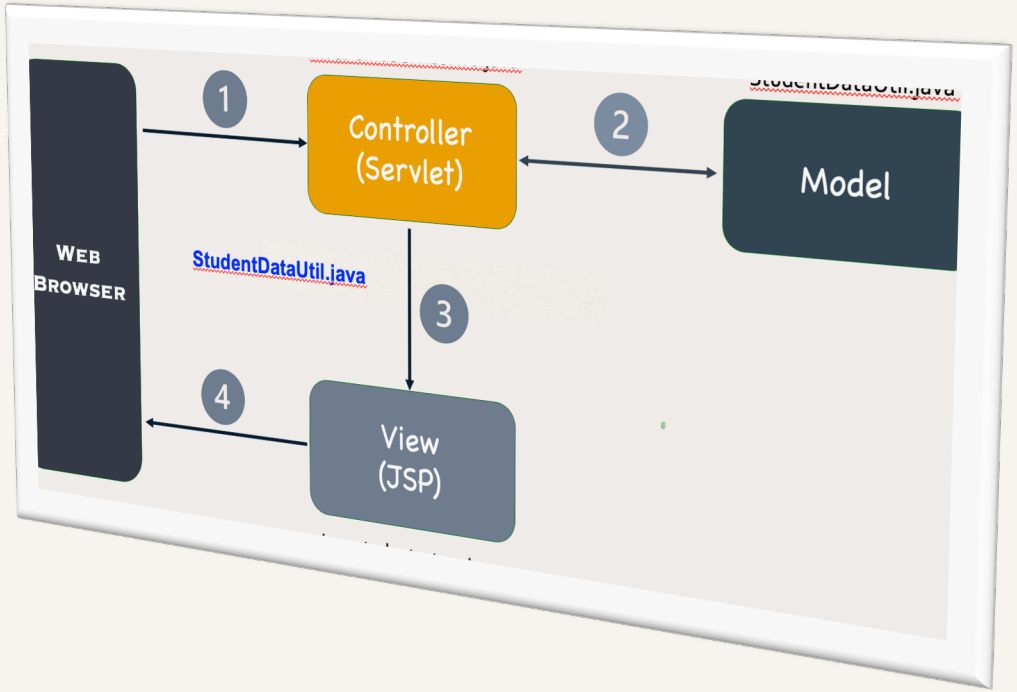
```
String[] students = {"Xuan Xuan", "Ha Ha", "Thu Thu", "Dong Dong"};
request.setAttribute("student_list", students);
```

MVC with Servlets and JSP (more)



Todo List

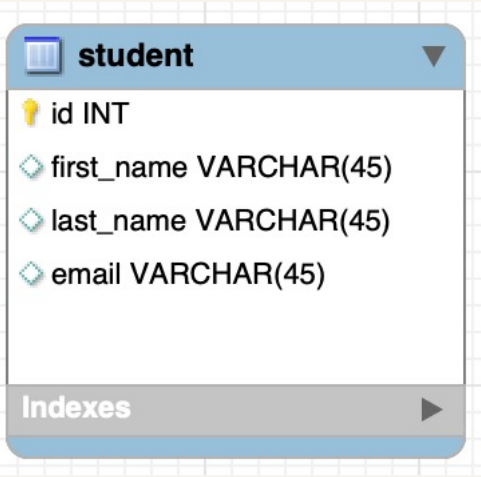
- Create Student class
- Create StudentDataUtil class
- Create MVC Servlet
- Create View JSP



MVC WEB App with JDBC

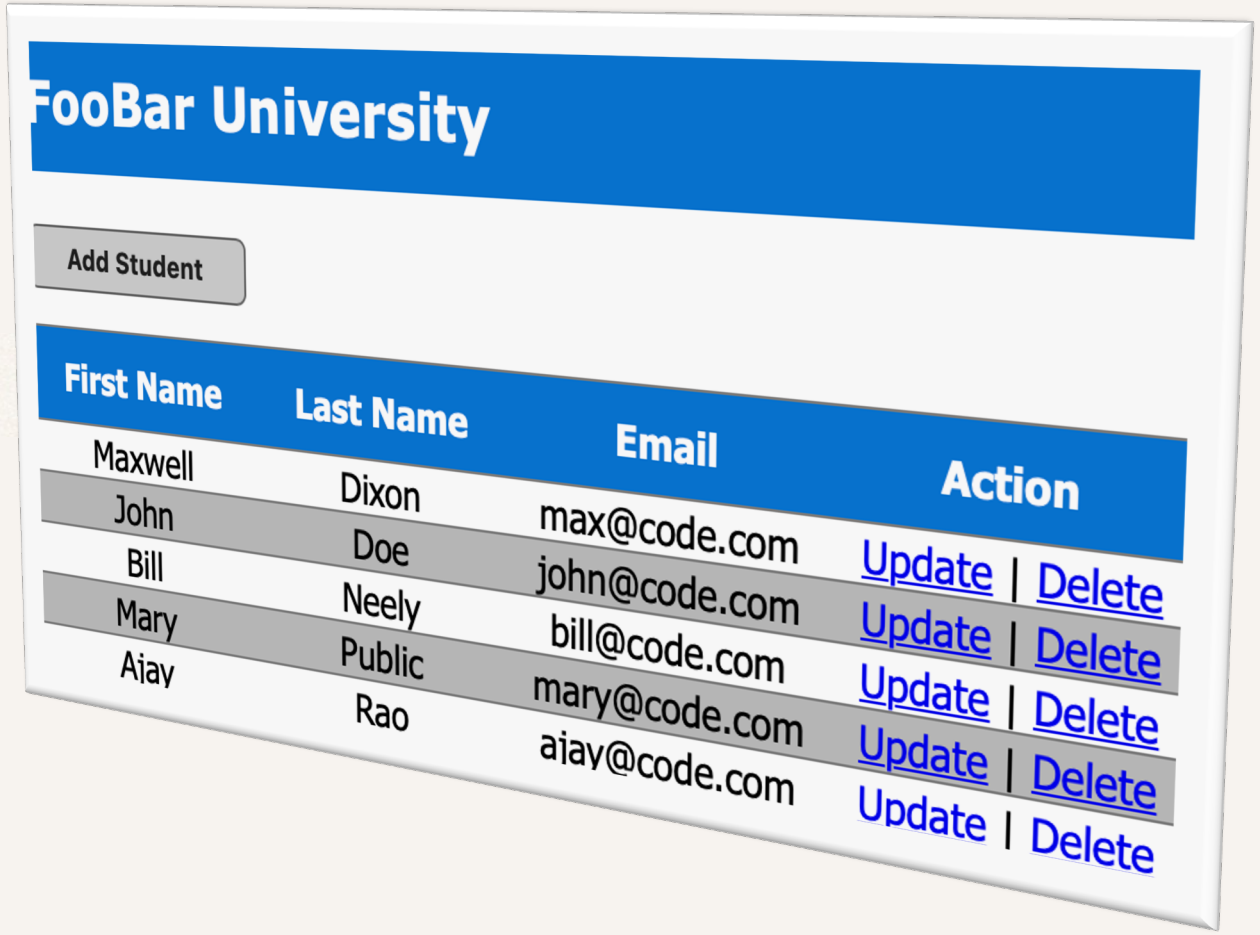
Project

Full working JSP and Servlet application that connects to a database



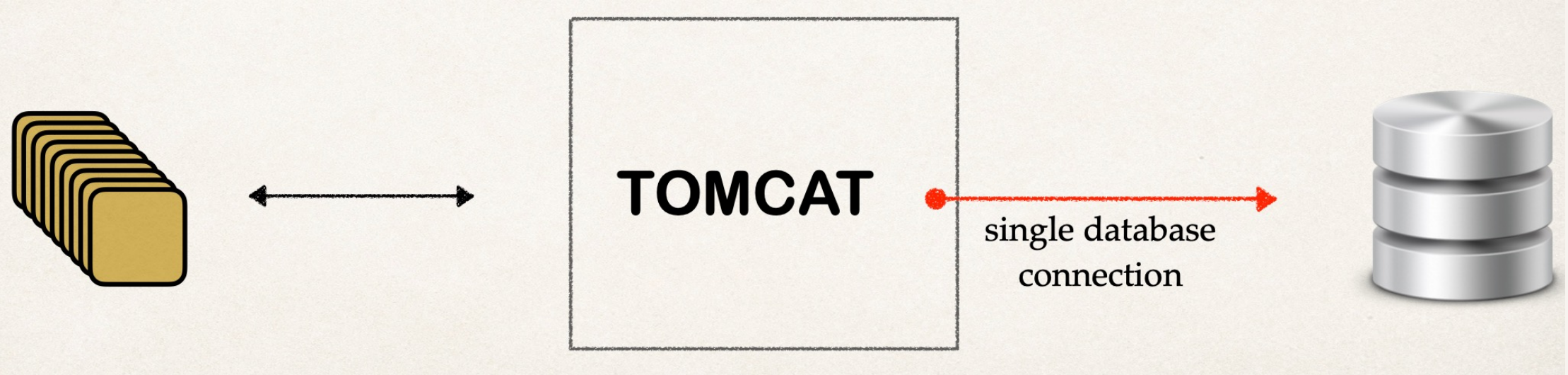
Road Map

- Set up Database Dev Environment (MySQL/SQL Server)
- List Students
- Add a new Student
- Update a Student
- Delete a Student



Setup Tomcat Database Connection Pool

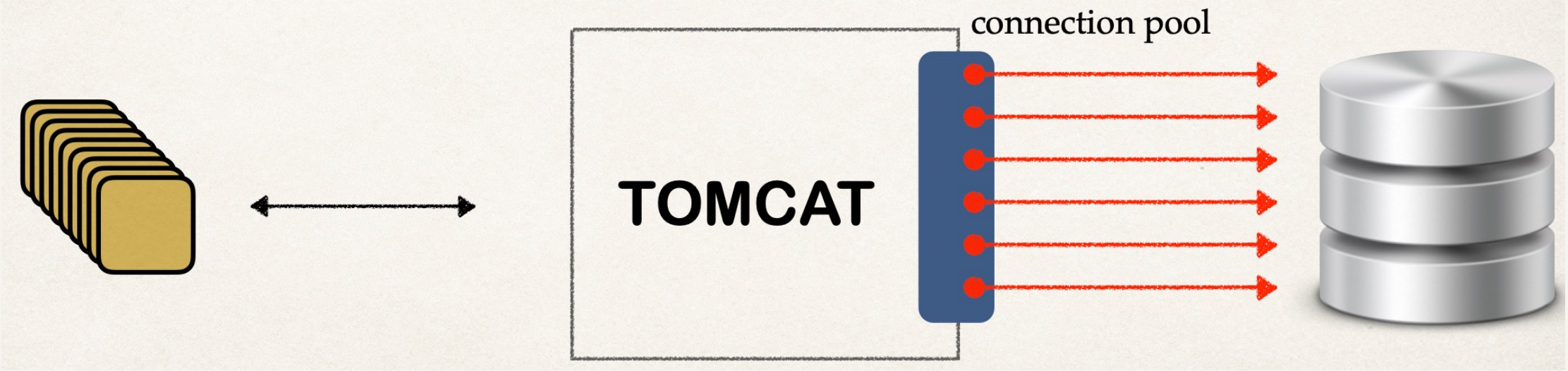
Database Connections in web apps: *You may think you only need a single database connection*



→Will not scale for multiple web users

Database Connection Pools

- Best practice is to use database connection pools



→ Allows your app to scale and handle multiple users quickly

Setup Tomcat Database Connection Pool

- Download JDBC Driver JAR file
- Define connection pool in META-INF/context.xml
- Get connection pool reference in Java code

Download JDBC Driver JAR file

- MySQL/SQL Server JDBC Driver

http://dev.mysql.com/downloads

Or

https://docs.microsoft.com/vi-vn/sql/connect/jdbc/microsoft-jdbc-driver-for-sql-server

- Place the JAR file in your app's WEB-INF/lib

Define connection pool: context.xml

File: `WebContent/META-INF/context.xml`

```
<Context>

  <Resource name="jdbc/web_student_tracker"
    auth="Container" type="javax.sql.DataSource"
            maxActive="20" maxIdle="5" maxWait="10000"
            username="webstudent" password="webstudent"
            driverClassName="com.mysql.cj.jdbc.Driver"
            url="jdbc:mysql://localhost:3306/web_student_tracker
</Context>
```

Define connection pool: context.xml

File: `WebContent/META-INF/context.xml`

For MS-SQL Server:

```
driverClassName="com.microsoft.sqlserver.jdbc.SQLServerDriver"

url="jdbc:sqlserver://localhost:1433/web_student_tracker"
```

Get connection pool in Java code

Leverage **resource injection** with the Servlets: This means that Tomcat will AUTOMAGICALLY set the connection pool/data source on your servlet

File: `TestServlet.java`

```
//define datasource/connection pool for resource
@Resource(name="jdbc/web_student_tracker")
private DataSource dataSource;
```

File: `context.xml`

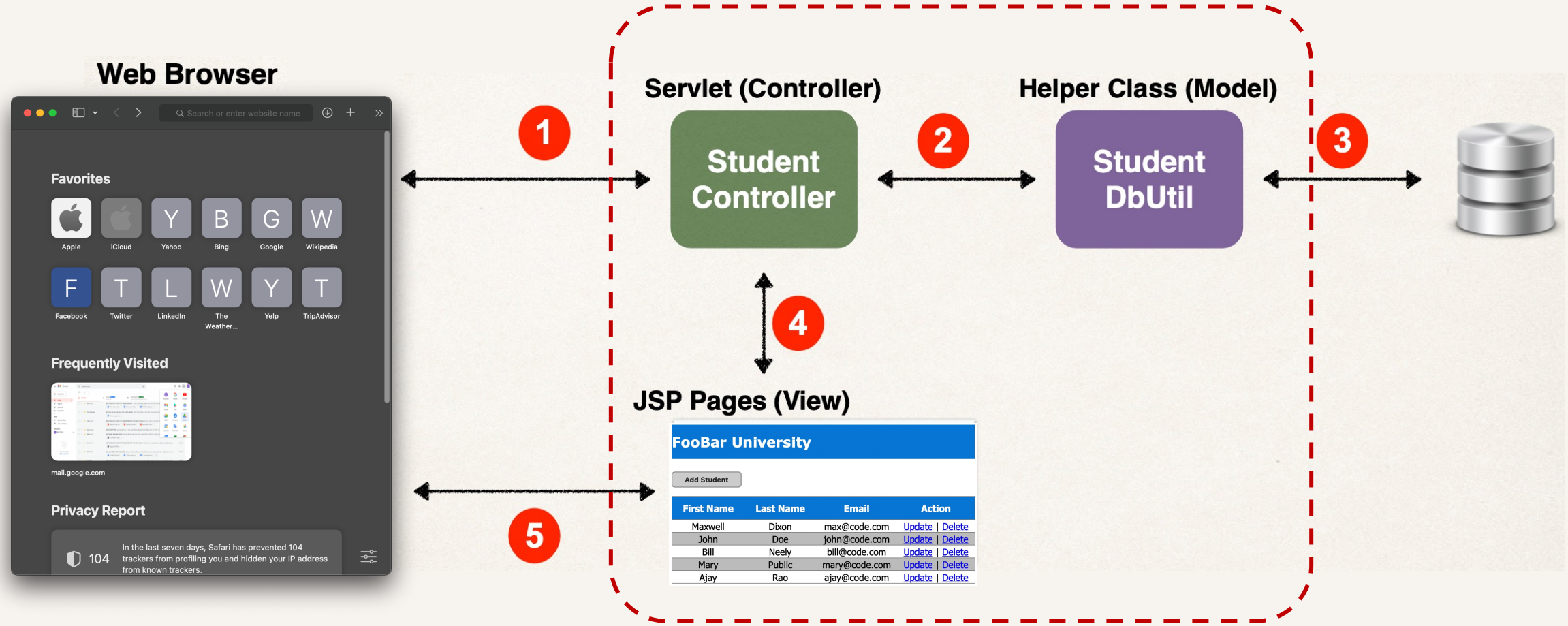
```
<Resource name="jdbc/web_student_tracker"
  auth="Container" type="javax.sql.DataSource"
          maxActive="20" maxIdle="5" maxWait="10000"
          username="webstudent" password="webstudent"
          driverClassName="com.mysql.cj.jdbc.Driver"
          url="jdbc:mysql://localhost:3306/web_student_tracker
```

Get Data in Java code

File: `TestServlet.java`

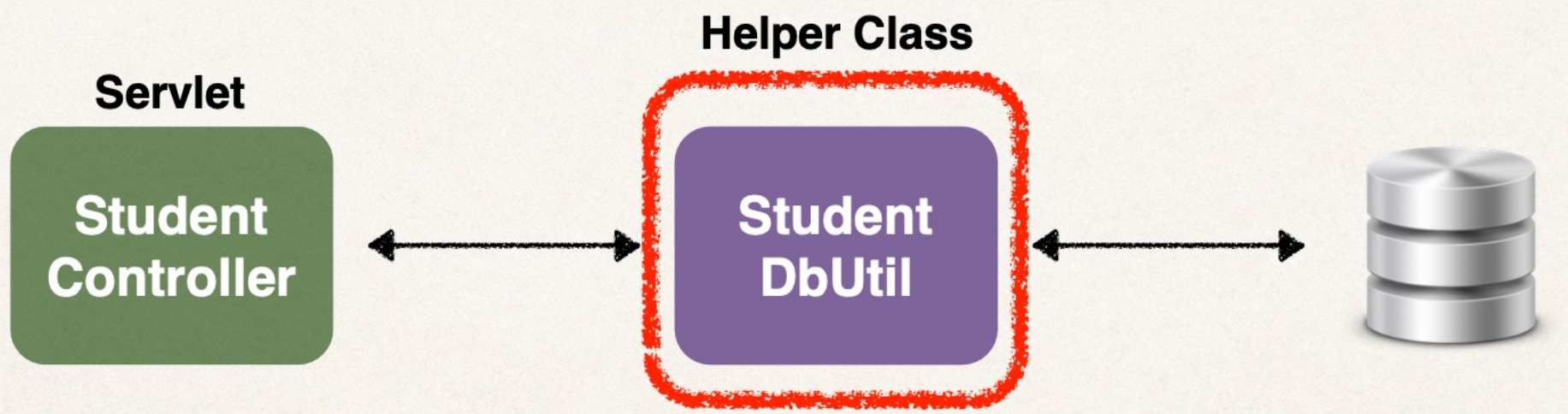
```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    // Step 1:  Set up the printwriter
    PrintWriter out = response.getWriter();
    response.setContentType("text/plain");
    // Step 2:  Get a connection to the database
    Connection myConn = null;
    Statement myStmt = null;
    ResultSet myRs = null;
    try {
        myConn = dataSource.getConnection();
        // Step 3:  Create a SQL statements
        String sql = "select * from student";
        myStmt = myConn.createStatement();
        // Step 4:  Execute SQL query
        myRs = myStmt.executeQuery(sql);
        // Step 5:  Process the result set
        while (myRs.next()) {
            String email = myRs.getString("email");
            out.println(email); }
    }
    catch (Exception exc) {
        exc.printStackTrace();}
}
```

MVC APP


Web Browser
Search or enter website name
Favorites
Apple
iCloud
Yahoo
Bing
Google
Wikipedia
Facebook
Twitter
LinkedIn
The Weather...
Yelp
TripAdvisor
Frequently Visited
mail.google.com
Privacy Report
104
In the last seven days, Safari has prevented 104 trackers from profiling you and hidden your IP address from known trackers.
Servlet (Controller)
Helper Class (Model)
Student Controller
Student DbUtil
1
2
3
4
5
JSP Pages (View)
FooBar University
Add Student
First Name
Last Name
Email
Action
Maxwell
Dixon
max@code.com
Update | Delete
John
Doe
john@code.com
Update | Delete
Bill
Neely
bill@code.com
Update | Delete
Mary
Public
mary@code.com
Update | Delete
Ajay
Rao
ajay@code.com
Update | Delete

Student DB Utility

- Responsible for interfacing with the database using JDBC code
- This is a common design pattern: Data Accessor Object (DAO)



List Students

- Create Student.java
- Create StudentDBUtil.java
- Create StudentControllerServlet.java
- Create JSP page: list-students.jsp

List Students – Create Student.java

File: iuh.fit.web.jdbc.Student.java

```java
package iuh.fit.web.jdbc;
public class Student {
    private int id;
    private String firstName;
    private String lastName;
    private String email;
    public Student(String firstName, String lastName, String email) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }
    public Student(int id, String firstName, String lastName, String email)
    {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }
```

List Students - Create StudentDBUtil.java (1)

File: iuh.fit.web.jdbc.StudentDBUtil.java (used by StudentControllerServlet)

```
import java.sql.Connection;

public class StudentDbUtil {

    private DataSource dataSource;

    public StudentDbUtil(DataSource theDataSource) {
        dataSource = theDataSource;
    }
```

List Students – Create StudentDBUtil.java (2.1)

File: iuh.fit.web.jdbc.StudentDBUtil.java (used by StudentControllerServlet)

```
public List<Student> getStudents() throws Exception {
    List<Student> students = new ArrayList<>();
    Connection myConn = null;
    Statement myStmt = null;
    ResultSet myRs = null;
    try {
        // get a connection
        myConn = dataSource.getConnection();
        // create sql statement
        String sql = "select * from student order by last_name";
        myStmt = myConn.createStatement();
        // execute query
        myRs = myStmt.executeQuery(sql);
```

List Students – Create StudentDBUtil.java (2.2)

File: iuh.fit.web.jdbc.StudentDBUtil.java (used by StudentControllerServlet)

```
  String sql = "select * from student order by last_name";
  myStmt = myConn.createStatement();
  // execute query
  myRs = myStmt.executeQuery(sql);
  // process result set
  while (myRs.next()) {
      // retrieve data from result set row
      int id = myRs.getInt("id");
      String firstName = myRs.getString("first_name");
      String lastName = myRs.getString("last_name");
      String email = myRs.getString("email");
      // create new student object
      Student tempStudent = new Student(id, firstName, lastName, email);
      // add it to the list of students
      students.add(tempStudent);}
  return students;}
finally {
  // close JDBC objects
  close(myConn, myStmt, myRs);}
```

List Students - Create StudentDBUtil.java (3)

File: iuh.fit.web.jdbc.StudentDBUtil.java (used by StudentControllerServlet)

```
public void addStudent(Student theStudent) throws Exception {
    Connection myConn = null;
    PreparedStatement myStmt = null;
    try {
        // get db connection
        myConn = dataSource.getConnection();
        // create sql for insert
        String sql = "insert into student "
                   + "(first_name, last_name, email) "
                   + "values (?, ?, ?)";
        myStmt = myConn.prepareStatement(sql);
        // set the param values for the student
        myStmt.setString(1, theStudent.getFirstName());
        myStmt.setString(2, theStudent.getLastName());
        myStmt.setString(3, theStudent.getEmail());
        // execute sql insert
        myStmt.execute();
    }
    finally {
        // clean up JDBC objects
        close(myConn, myStmt, null);
    }
}
```

List Students - Create StudentDBUtil.java (4)

File: iuh.fit.web.jdbc.StudentDBUtil.java (used by StudentControllerServlet)

```
public void updateStudent(Student theStudent) throws Exception {
    Connection myConn = null;
    PreparedStatement myStmt = null;
    try {
        // get db connection
        myConn = dataSource.getConnection();
        // create SQL update statement
        String sql = "update student "
                   + "set first_name=?, last_name=?, email=? "
                   + "where id=?";
        // prepare statement
        myStmt = myConn.prepareStatement(sql);
        // set params
        myStmt.setString(1, theStudent.getFirstName());
        myStmt.setString(2, theStudent.getLastName());
        myStmt.setString(3, theStudent.getEmail());
        myStmt.setInt(4, theStudent.getId());
        // execute SQL statement
        myStmt.execute();}
    finally {
        // clean up JDBC objects
        close(myConn, myStmt, null);}
}
```

List Students – Create StudentDBUtil.java (5)

File: iuh.fit.web.jdbc.StudentDBUtil.java (used by StudentControllerServlet)

```
public void deleteStudent(String theStudentId) throws Exception {
    Connection myConn = null;
    PreparedStatement myStmt = null;
    try {
        // convert student id to int
        int studentId = Integer.parseInt(theStudentId);
        // get connection to database
        myConn = dataSource.getConnection();
        // create sql to delete student
        String sql = "delete from student where id=?";
        // prepare statement
        myStmt = myConn.prepareStatement(sql);
        // set params
        myStmt.setInt(1, studentId);
        // execute sql statement
        myStmt.execute();}
    finally {
        // clean up JDBC code
        close(myConn, myStmt, null);}
}
```

List Students - Create StudentDBUtil.java (6.1)

File: iuh.fit.web.jdbc.StudentDBUtil.java (used by StudentControllerServlet)

```
public Student getStudent(String theStudentId) throws Exception {
    Student theStudent = null;
    Connection myConn = null;
    PreparedStatement myStmt = null;
    ResultSet myRs = null;
    int studentId;
    try {
        // convert student id to int
        studentId = Integer.parseInt(theStudentId);
        // get connection to database
        myConn = dataSource.getConnection();
        // create sql to get selected student
        String sql = "select * from student where id=?";
        // create prepared statement
        myStmt = myConn.prepareStatement(sql);
        // set params
        myStmt.setInt(1, studentId);
        // execute statement
        myRs = myStmt.executeQuery();
        // retrieve data from result set row
```

List Students – Create StudentDBUtil.java (6.2)

File: iuh.fit.web.jdbc.StudentDBUtil.java (used by StudentControllerServlet)

```
        if (myRs.next()) {
            String firstName = myRs.getString("first_name");
            String lastName = myRs.getString("last_name");
            String email = myRs.getString("email");
            // use the studentId during construction
            theStudent = new Student(studentId, firstName, lastName, email);
        }
        else {
            throw new Exception("Could not find student id: " + studentId); }
        return theStudent;
    }
    finally {
        // clean up JDBC objects
        close(myConn, myStmt, myRs);}
}
```

List Students – Create StudentDBUtil.java (7)

File: iuh.fit.web.jdbc.StudentDBUtil.java (used by StudentControllerServlet)

```
private void close(Connection myConn, Statement myStmt, ResultSet myRs) {
    try {
        if (myRs != null) {
            myRs.close(); }
        if (myStmt != null) {
            myStmt.close(); }
        if (myConn != null) {
            myConn.close();
            // doesn't really close it ... just puts back in connection pool
        }
    }
    catch (Exception exc) {
        exc.printStackTrace();
    }
}
```

List Students - Create StudentControllerServlet.java (1)

File: iuh.fit.web.jdbc.StudentControllerServlet.java

```java
@WebServlet("/StudentControllerServlet")
public class StudentControllerServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    private StudentDbUtil studentDbUtil;

    @Resource(name="jdbc/web_student_tracker")
    private DataSource dataSource;

    @Override
    public void init() throws ServletException {
        super.init();

        // create our student db util ... and pass in the conn pool / datasource
        try {
            studentDbUtil = new StudentDbUtil(dataSource);
        }
        catch (Exception exc) {
            throw new ServletException(exc);
        }
    }
```

List Students - Create StudentControllerServlet.java (2)

File: iuh.fit.web.jdbc.StudentControllerServlet.java

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
try {
    // read the "command" parameter
    String theCommand = request.getParameter("command");
    // if the command is missing, then default to listing students
    if (theCommand == null) {
        theCommand = "LIST";}
    // route to the appropriate method
    switch (theCommand) {
    case "LIST":    listStudents(request, response);
        break;
    case "ADD":     addStudent(request, response);
        break;
    case "LOAD":    loadStudent(request, response);
        break;
    case "UPDATE":  updateStudent(request, response);
        break;
    case "DELETE":  deleteStudent(request, response);
        break;
    default:
        listStudents(request, response);
    }
}
```

List Students - Create StudentControllerServlet.java (3)

File: iuh.fit.web.jdbc.StudentControllerServlet.java

```
private void listStudents(HttpServletRequest request, HttpServletResponse response)
    throws Exception {

    // get students from db util
    List<Student> students = studentDbUtil.getStudents();

    // add students to the request
    request.setAttribute("STUDENT_LIST", students);

    // send to JSP page (view)
    RequestDispatcher dispatcher = request.getRequestDispatcher("/list-students.jsp");
    dispatcher.forward(request, response);
}
```

List Students - Create StudentControllerServlet.java (4)

File: iuh.fit.web.jdbc.StudentControllerServlet.java

```
private void addStudent(HttpServletRequest request, HttpServletResponse response)
    // read student info from form data
    String firstName = request.getParameter("firstName");
    String lastName = request.getParameter("lastName");
    String email = request.getParameter("email");
    // create a new student object
    Student theStudent = new Student(firstName, lastName, email);
    // add the student to the database
    studentDbUtil.addStudent(theStudent);
    // send back to main page (the student list)
    listStudents(request, response);
}
```

List Students - Create StudentControllerServlet.java (5)

File: iuh.fit.web.jdbc.StudentControllerServlet.java

```
private void loadStudent(HttpServletRequest request, HttpServletResponse response)
    throws Exception {
    // read student id from form data
    String theStudentId = request.getParameter("studentId");
    // get student from database (db util)
    Student theStudent = studentDbUtil.getStudent(theStudentId);
    // place student in the request attribute
    request.setAttribute("THE_STUDENT", theStudent);
    // send to jsp page: update-student-form.jsp
    RequestDispatcher dispatcher =
            request.getRequestDispatcher("/update-student-form.jsp");
    dispatcher.forward(request, response);
}
```

List Students - Create StudentControllerServlet.java (6)

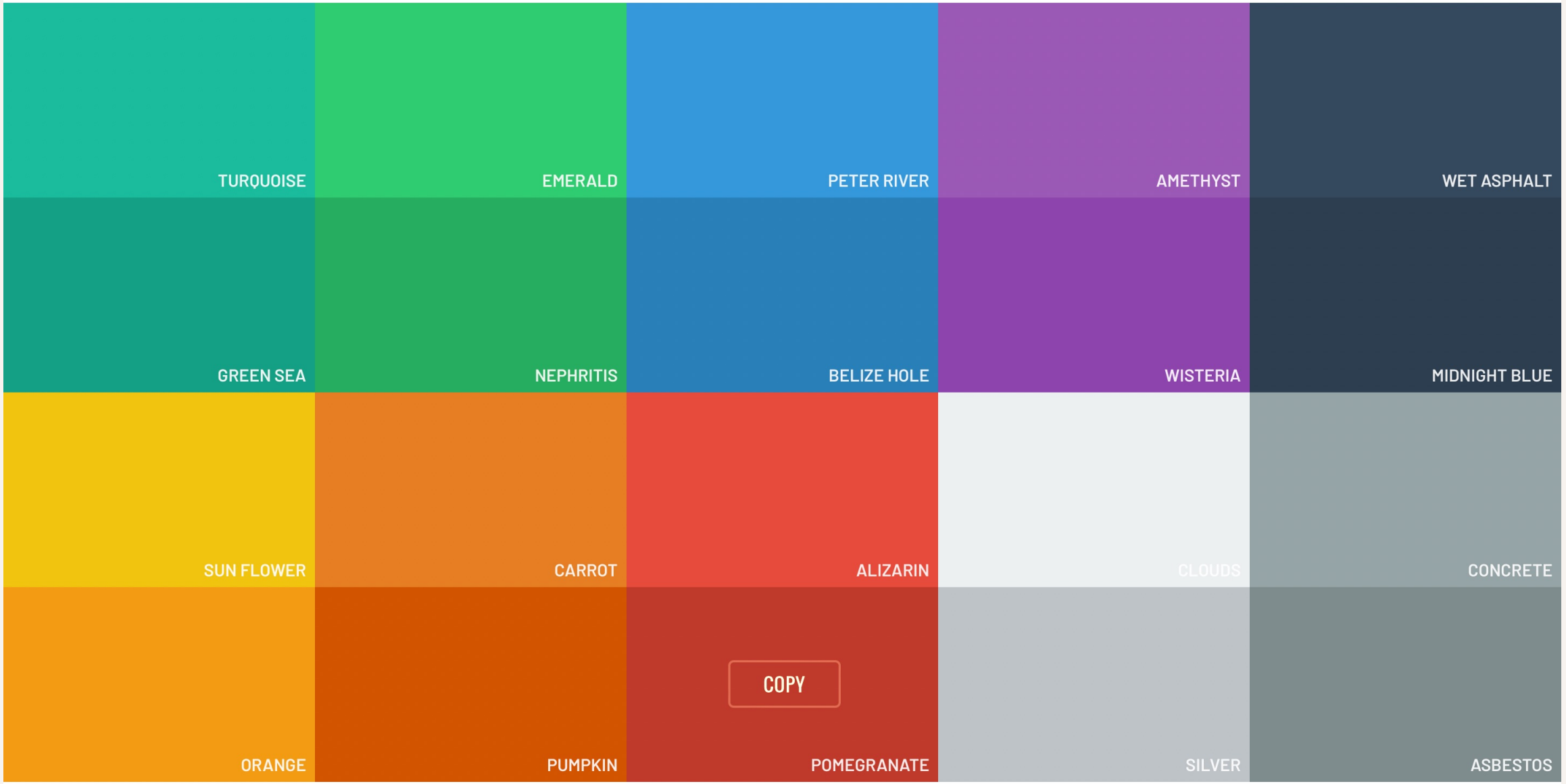
File: iuh.fit.web.jdbc.StudentControllerServlet.java

```
private void updateStudent(HttpServletRequest request, HttpServletResponse response)
    throws Exception {
    // read student info from form data
    int id = Integer.parseInt(request.getParameter("studentId"));
    String firstName = request.getParameter("firstName");
    String lastName = request.getParameter("lastName");
    String email = request.getParameter("email");
    // create a new student object
    Student theStudent = new Student(id, firstName, lastName, email);
    // perform update on database
    studentDbUtil.updateStudent(theStudent);
    // send them back to the "list students" page
    listStudents(request, response);
}
```

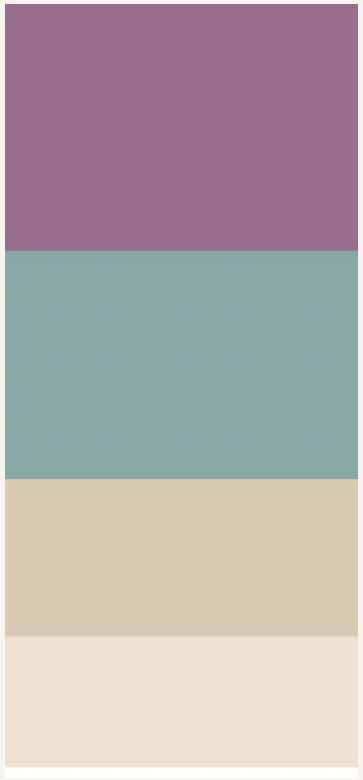
List Students - Create StudentControllerServlet.java (7)

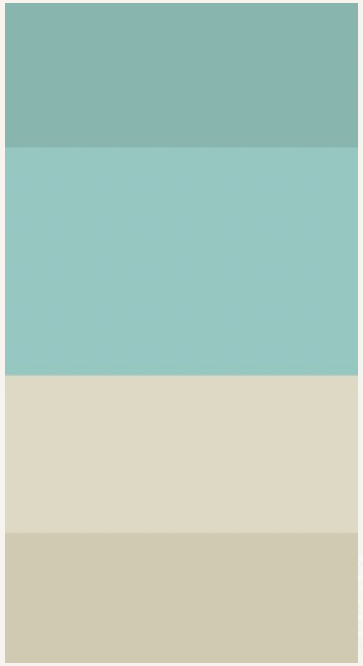
File: iuh.fit.web.jdbc.StudentControllerServlet.java

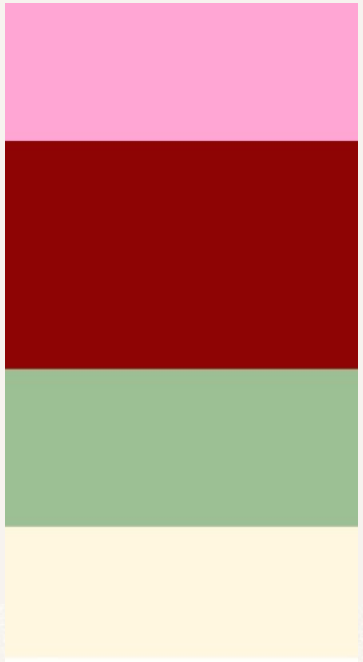
```
private void deleteStudent(HttpServletRequest request, HttpServletResponse response)
    throws Exception {
    // read student id from form data
    String theStudentId = request.getParameter("studentId");
    // delete student from database
    studentDbUtil.deleteStudent(theStudentId);
    // send them back to "list students" page
    listStudents(request, response);
}
```

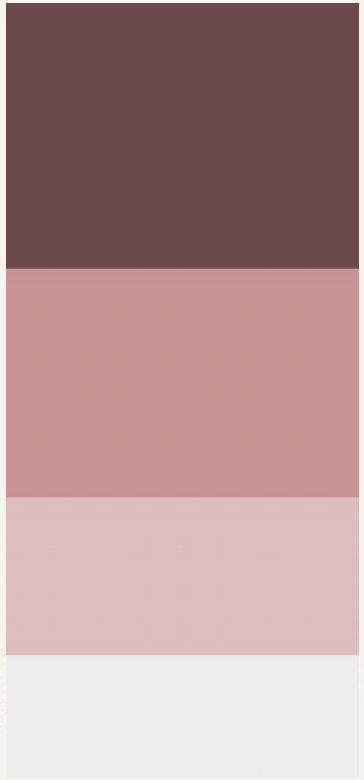


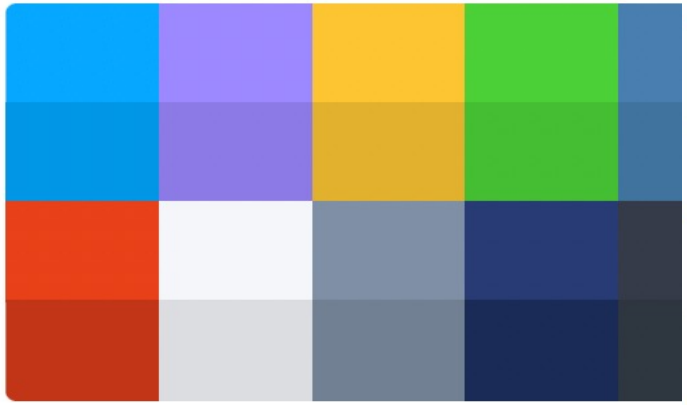
Flat UI Palette v1 by Flat UI Colors









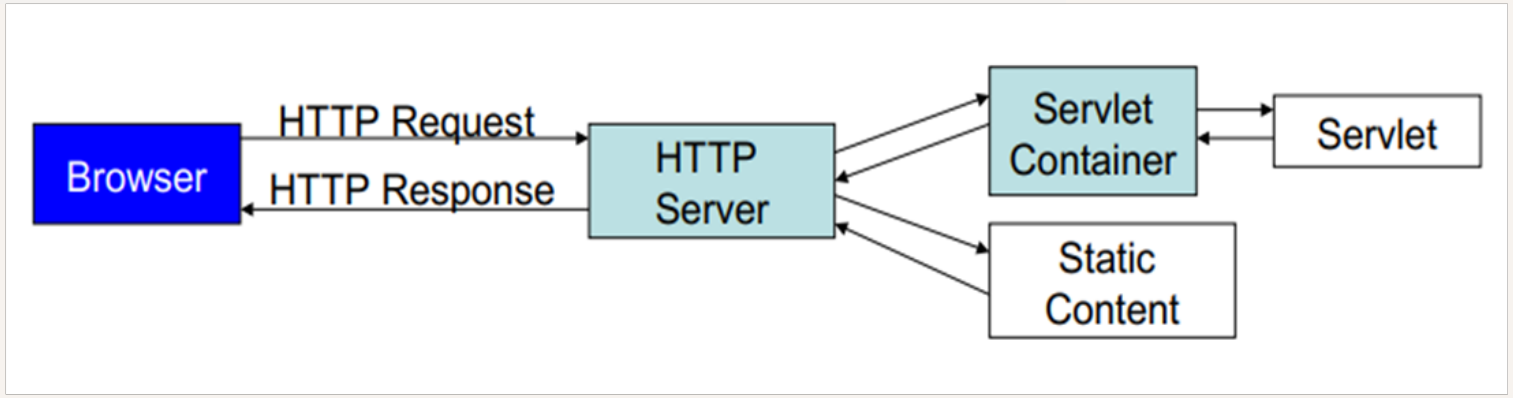


British Palette

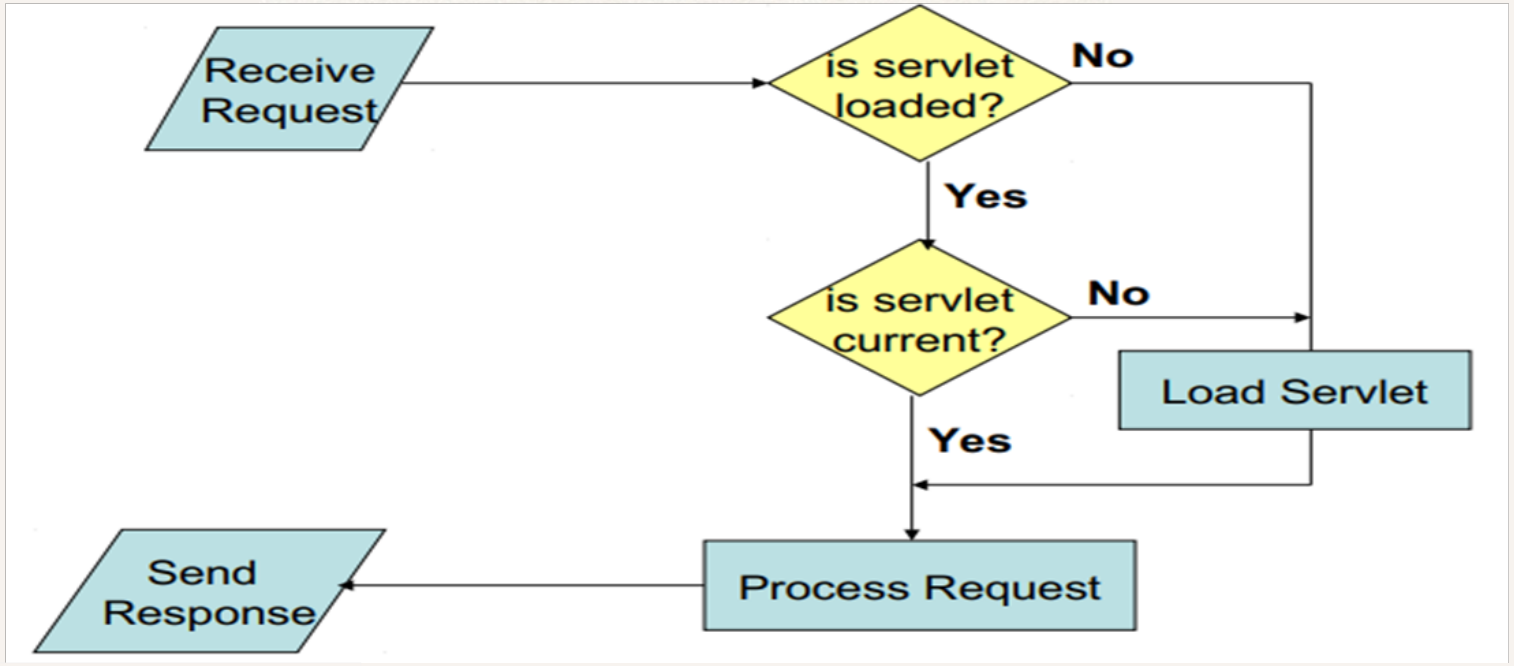
German Palette 🇩🇪

Indian Palette 🇮🇳

Servlet Container Architecture



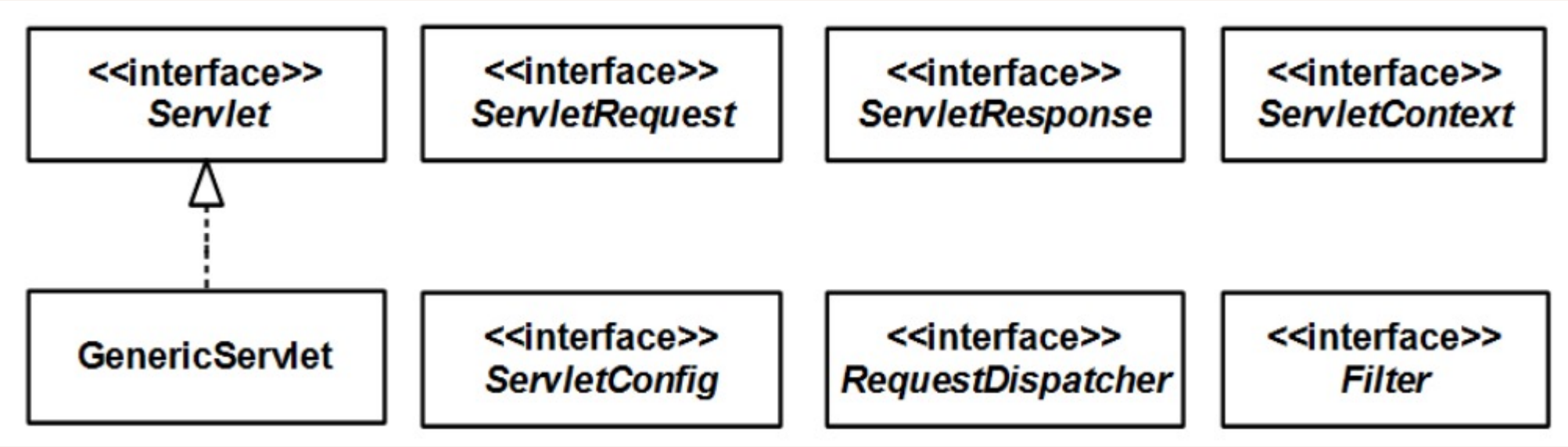
- How Servlets Work



Servlet APIs

- **javax.servlet**: contains a number of classes and interfaces that describe and define the contracts between a **servlet class** and the **runtime environment** provided for an instance of such a class by a conforming **servlet container**.
- **javax.servlet.http**: contains a number of classes and interfaces that describe and define the contracts between a servlet class running under the HTTP protocol and the runtime environment provided for an instance of such a class by a conforming servlet container.
- **javax.servlet.annotation:** Contains annotations to annotate servlets, filters, and listeners. It also specifies metadata for annotated components.
- **javax.servlet.descriptor:** Contains types that provide programmatic access to a web application's configuration information.

The javax.servlet Package

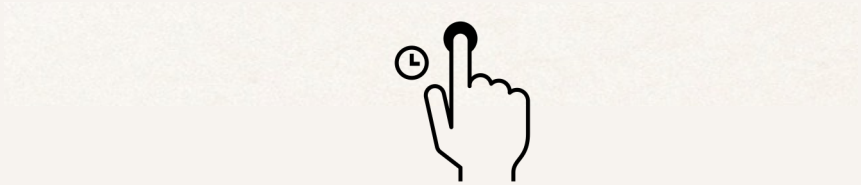


Servlet interface

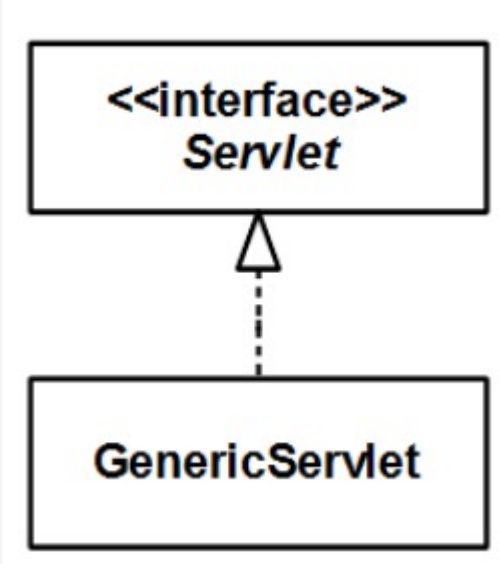
init, **service**, **and destroy** are **lifecycle methods**

- void init(ServletConfig config): Initializes the servlet.
- void service(ServletRequest req, ServletResponse res): Carries out a single request from the client.
- void destroy(): Cleans up whatever resources are being held (e.g., memory, file handles, threads) and makes sure that any persistent state is synchronized with the servlet's current in-memory state.
- ServletConfig getServletConfig(): Returns a servlet config object, which contains any initialization parameters and startup configuration for this servlet.
- String getServletInfo(): Returns a string containing information about the servlet, such as its author, version, and copyright.

Writing A Basic Servlet Application



GenericServlet interface



GenericServlet: an Abstract class, which implements both **Servlet, ServletConfig** and perform the following tasks:

- Assign the **ServletConfig** in the **init** method to a class level variable so that it can be retrieved by calling **getServletConfig**.
- Provide default implementations of all methods in the **Servlet** interface.
- Provide methods that wrap the methods in the **ServletConfig**.

ServletRequest

- For every HTTP request, the servlet container creates an instance of ServletRequest and passes it to the servlet's service method. The ServletRequest encapsulates information about the request
- Some of the methods in the **ServletRequest**:
  - **int getContentLength():** Returns the number of bytes in the request body. If the length is not known, this method returns -1.
  - **java.lang.String getContentType():** Returns the MIME type of the request body or null if the type is not known
  - **java.lang.String getParameter(java.lang.String *name*):** Returns the value of the specified request parameter.

ServletResponse

- Prior to invoking a servlet's **service method**, the **servlet container** creates a **ServletResponse** and pass it as the **second argument** to the **service method**
- Some of the methods in the **ServletResponse**:
  - **java.io.PrintWriter getWriter ():** returns a PrintWriter that can send text to the client. Before sending any HTML tag, you should set the content type of the response by calling the setContentType method, passing "text/html" as an argument

    ➔ `response.setContentType("text/html");`

  - **ServletOutputStream getOutputStream():** returns a suitable for writing binary data in the response.

ServletConfig

- The **servlet container** passes a **ServletConfig** to the **servlet's init method** when the servlet container **initializes the servlet**
- The **ServletConfig** encapsulates configuration information that you can pass to a servlet through **@WebServlet** or the deployment descriptor (in web.xml)
- Some of the methods in the **ServletConfig**:
  - **java.lang.String getInitParameter(java.lang.String name):** return the value of an initial parameter from inside a servlet
  - **java.util.Enumeration<java.lang.String> getInitParameterNames():** returns an Enumeration of all initial parameter names

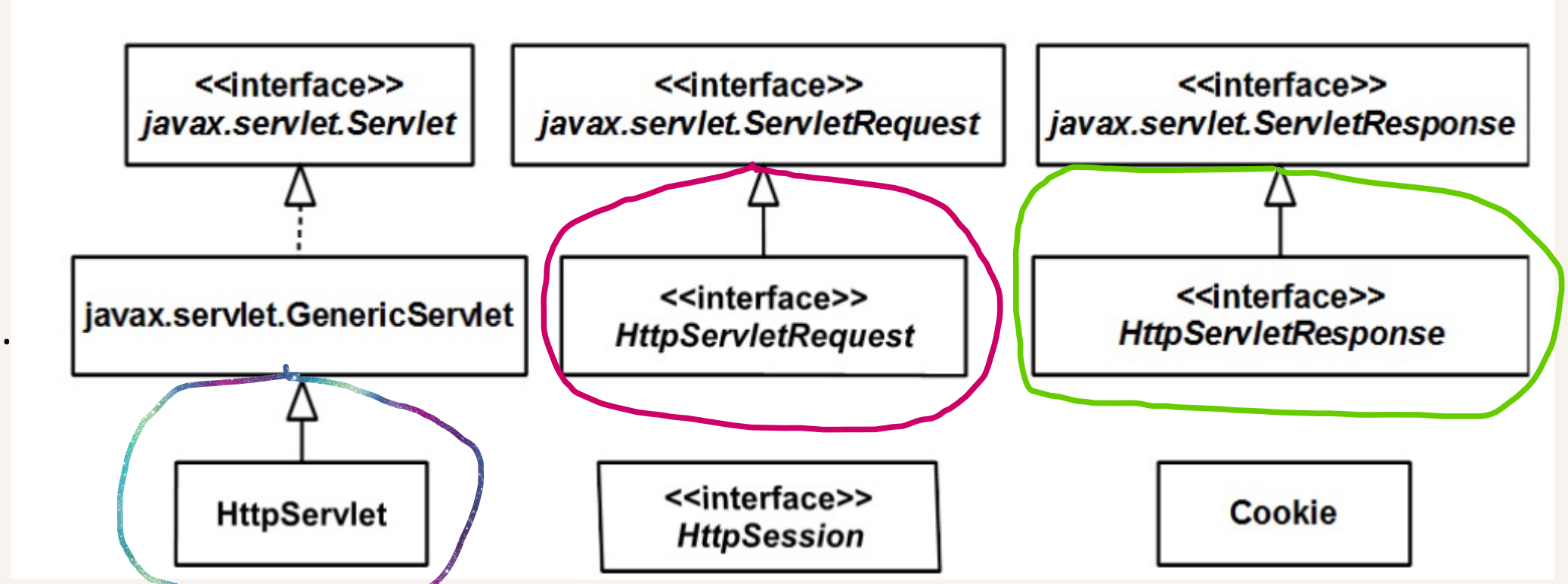
ServletContext

- The **ServletContext** represents the **servlet application**. There is **only one context per web application**
- We can obtain the **ServletContext** by calling the **getServletContext** method on the **ServletConfig** or **getServletContext** method in **ServletRequest**
- We can share information that can be accessed **from all resources** in the **application** and to **enable dynamic registration of web objects. Objects** stored in ServletContext are called **attributes**
  - **java.lang.Object getAttribute(java.lang.String *name*)**
  - **java.util.Enumeration<java.lang.String> getAttributeNames()**
  - **void setAttribute(java.lang.String *name*, java.lang.Object *object*)**
  - **void removeAttribute(java.lang.String name)**

HTTP Servlets

HTTP Servlets

- Most, servlet applications we write will work with HTTP. This means, we can make use of the features offered by HTTP.
- The javax.servlet.http package is the second package in the Servlet API that contains classes and interfaces for writing servlet applications



HTTP Servlets

- When using **HttpServlet**, you will work with the **HttpServletRequest** and **HttpServletResponse** objects that represent the **servlet request** and the **servlet response**, respectively
- **HttpServlet** overrides the service method in **GenericServlet** and adds **another service** method
- The **new service** method **in HttpServlet** then examines the **HTTP method used to send the request** (by calling request.getMethod) and call one of the following methods: **doGet**, **doPost**, **doHead**, **doPut**, **doTrace**, **doOptions**, and **doDelete**
- We rarely need to override the **service methods** anymore. Instead, you override **doGet** or **doPost** or both **doGet and doPost**

HttpServlets Methods

- void **doGet** (HttpServletRequest request, HttpServletResponse response): handles **GET** requests
- void **doPost** (HttpServletRequest request, HttpServletResponse response): handles **POST** requests
- void **doPut** (HttpServletRequest request, HttpServletResponse response): handles **PUT** requests
- void **doDelete** (HttpServletRequest request, HttpServletResponse response): handles **DELETE** requests

HttpServletRequest

HttpServletRequest represents the servlet request in the **HTTP environment**

- **java.lang.String getContextPath():** Returns the portion of the request URI that indicates the context of the request.
- **Cookie[] getCookies():** Returns an array of **Cookie** objects.
- **java.lang.String getHeader(java.lang.String name):** Returns the value of the specified HTTP header.
- **java.lang.String getMethod():** Returns the name of the HTTP method with which this request was made.
- **java.lang.String getQueryString():** Returns the query string in the request URL.
- **HttpSession getSession():** Returns the session object associated with this request. If none is found, creates a new session object.
- **HttpSession getSession(boolean create):** Returns the current session object associated with this request. If none is found and the create argument is **true**, create a new session object.

HttpServletResponse

HttpServletResponse represents the **servlet response** in the **HTTP environment.** Here are some of the methods defined in it:

- **void addCookie(Cookie *cookie*):** Adds a cookie to this response object.
- **void addHeader(java.lang.String name, java.lang.String value):** Adds a header to this response object.
- **void sendRedirect(java.lang.String location):** Sends a response code that redirects the browser to the specified location

HTML Forms

Working with HTML Forms

- A web application almost always contains **one or more HTML forms** to **take user input**. We can easily send an HTML form from a servlet to the browser. When the user submits the form, **values entered in the form are sent to the server as request**

- The value of an **HTML input field** (a text field, a hidden field, or a password field) or **text area** is sent to the server **as a string**. An **empty input field or text area** sends an **empty string**. As such, **ServletRequest.getParameter** that takes an **input field name never returns null**

- An HTML **select element** also sends a string to the server. If none of the options in the select element is selected, the **value of the option that is displayed is sent**

- A **multiple-value select element** sends a **string array** and has to be handled by **ServletRequest.getParameterValues**

Working with HTML Forms

- A **checked checkbox** sends the string **“on”** to the server. **An unchecked checkbox** sends **nothing to the server** and ServletRequest.getParameter(fieldName) **returns null**

- **Radio buttons** send the value of **the selected button** to the server. If none of the buttons is selected, **nothing is sent to the server** and ServletRequest.getParameter(fieldName) **returns null**

- If a form contains **multiple input elements with the same name**, **all values will besubmitted** and we have to use **ServletRequest.getParameterValues** to retrieve them. **ServletRequest.getParameter** will only return the **last value**

Servlet Mapping

Map a servlet with an annotation - @WebServlet

With **@WebServlet** we can specify:

- **Name**: defines the name of servlet
- **urlPatterns:** maps the servlet to the pattern
- **loadOnStartup:** loads the servlet at the time of deployment or server start if value is positive
- **Description:** description about servlet
- **initParams:** takes multiple **@WebInitParam** annotation

Map a servlet with an annotation - @WebServlet

```
@WebServlet(name="/WelcomeServlet",urlPatterns="/WelcomeServlet"
,loadOnStartup=1, description="Welcome Servlet")
-

}

@WebServlet(name="/WelcomeServlet",urlPatterns={"/WelcomeServlet",
"/HelloServlet"}, description="Welcome Servlet")



@WebServlet (urlPatterns= {"/emailList", "/email/*"})
```

Map a servlet with an annotation - @WebInitParam

@WebInitParam enables us to configure one init param and provides name , value and description attribute.

- **Name**: name of init param
- **Value**: value of init param
- **Description**: description of init param

Map a servlet with an annotation - @WebInitParam

```
@WebServlet(name="/WelcomeServlet",urlPatterns={"/WelcomeServlet,

initParams={

@WebInitParam(name="Param1",value="Param 1 Value",description="param1"),

@WebInitParam(name="Param2",value="Param 2 Value",description="param1")
})
```

Map a servlet with Deployment Descriptor (web.xml)

XML elements for working with **servlet mapping**:

- **<servlet-name>:** Specifies a unique name for the servlet that's used to identify the servlet within the web.xml file. This element is required for both the **servlet element** and the **servle mapping element**
- **<servlet-class>:** Specifies the class for the servlet. Note that this element **includes the package and name for the class but not the .class extension.**
- **<url-pattern>:** Specifies the URL or URLs that are mapped to the specified servlet. This pattern **must begin with a front slash.**

Map a servlet with Deployment Descriptor (web.xml)

```
<servlet>
	<servlet-name>MyGenericServletDemo</servlet-name>
	<servlet-class>servletapi.MyServletDemo</servlet-class>
</servlet>

<servlet-mapping>
	<servlet-name>MyGenericServletDemo</servlet-name>
	<url-pattern>/my</url-pattern>
</servlet-mapping>
```

Map a servlet with Deployment Descriptor (web.xml)

```
<servlet>
      <servlet-name>MyGenericServletDemo</servlet-name>
      <servlet-class>servletapi.MyServletDemo</servlet-class>
      <init-param>
            <description> param1 </description>
            <param-name>Param1</param-name>
            <param-value>Param 1 Value </param-value>
    </init-param>
</servlet>
<servlet-mapping>
      <servlet-name>MyGenericServletDemo</servlet-name>
      <url-pattern>/my</url-pattern>
</servlet-mapping>
```

Session management

Session management

- **Session management** or **session tracking** is a very important topic in web application.
- **HTTP protocol** is a **stateless protocol** which means no user information is pertained and server considers **every request as a new request**
- A **session** is a **conversation between the server and a client**. To maintain the conversational state , session tracking is needed.

Session management - Techniques

- URL Rewriting
- Hidden Fields
- Cookies
- Session Tracking API

Session management – Techniques - URL Rewriting

- **URL Rewriting:** a session tracking technique whereby you add a **token** or **multiple tokens** as **a query string to a URL**

- Note that the URL and the tokens are separated by a question mark. Two tokens are separated by an ampersand character (&)

  - http://localhost:8080/HelloWorld/SourceServlet?**Session1Id=XYZ&Session2id=123**

- URL rewriting is **only suitable** if the information that needs to be retained **does not span too many pages** and the information is **not sensitive**

- **Drawbacks:** you **cannot easily** append values to links in **static pages**; this would **present a challenge** if there **are a lot of links in a page**.

Session management – Techniques - Hidden Fields

- **Hidden fields :** Using hidden fields to retain states is similar to employing the URL rewriting technique
- Put **values** in **hidden fields** within an **HTML form**
- Only suitable if the page contains a form
- The advantage of this technique over URL rewriting is you can pass **much more characters** to the server and **no character encoding is necessary**
- This technique is only good if the **information to be passed doesn't need to span many pages**.

Session management – Techniques - Cookies

- a small piece of information that is passed back and forth between the web server and the browser automatically
- Cookies are **embedded** as **HTTP headers**, the process of transferring them is **handled** by the **HTTP protocol.** We can make **a cookie live** as **long** or as **short** as you want. A web browser is expected to support up to **twenty cookies per web server**
- The downside of cookies is a user can **refuse to accept cookies** by **changing** his/her **browser settings**.
- Cookies are **suitable** for **information** that needs to **span many pages**
- To use cookies, you need to be familiar with the **javax.servlet.http.Cookie** class as well as a couple of methods in the **HttpServletRequest** and **HttpServletResponse** interfaces.

Session management – Techniques - Cookies

- To create a cookie , pass a name and a value to the Cookie class's constructor: **Cookie cookie = new Cookie(name, value);**
- After you create a Cookie, we can set its **domain**, **path**, and **maxAge** (maximum age in seconds) properties.
- To **send a cookie to the browser**, call the add method on the HttpServletResponse: **httpServletResponse.addCookie(cookie).**
- To **access a cookie** sent by the browser, use the **getCookies** method on the **HttpServletRequest**. This method returns a **Cookie array** or **null** if no cookie is found in the request
- To find a **cookie** by **name**, we have to **iterate over the array**.
- To **delete a cookie**, you need to **create** an **identically-named cookie**, set its **maxAge to 0;** add the new cookie to the **HttpServletResponse**

Session management – Techniques - Session Tracking API

- Servlets provide a convenient and stable session-tracking solution using the **HttpSession API**
- Session tracking in servlet is very simple and it involves following steps:
  - Get the **associated session object (HttpSession)** using **request.getSession()**
  - To **get** the **specific value out of session object**, call **getAttribute(String)** on the **HttpSession object**.
  - To **store** any **information in a session** call **setAttribute(key,object)** on a **session object.**
  - To **remove** the **session data** , call **removeAttribute(key)** to discard a object with a **given key**.
  - To **invalidate** the **session**, call **invalidate() on session object**. This is used to logout the logged in user.

Session management – Techniques - Session Tracking API

- A **value put in an HttpSession** is **stored in memory**. As such, we should only **store the smallest possible objects** in it and not too many of them
- **Values** stored in an HttpSession **are not sent to the client side**.
- The **servlet container** generates **a unique identifier for every HttpSession** it creates and **sends this identifier as a token to the browser**, either as a **cookie named JSESSIONID** or by **appending it to URLs** as a **jsessionid parameter**.
- On **subsequent requests**, the **browser sends back the token to the server**, allowing the server to tell which user is making the request

QUESTIONS