PROG 43431
Multi-tier Programming I

Web Application Design Patterns

Acknowledgement

- These lecture slides are partly based on materials by Professor Pejman Salehi, Simon Hood, El Sayed Mahmoud, and Murach's Java Servlets and JSP (3rd Edition)

Design Pattern

“A design pattern is a general reusable solution to a commonly occurring problem within a given context in software design.”

http://en.wikipedia.org/wiki/Software_design_pattern

Example - Iterator pattern

- In object-oriented programming, the **iterator pattern** is a design pattern in which an iterator is used **to traverse a container and access the container's elements**. The iterator pattern decouples algorithms from containers; in some cases, algorithms are necessarily container-specific and thus cannot be decoupled.
  – https://en.wikipedia.org/wiki/Iterator_pattern

Architecture design & modeling

- important in software development
  - **Apply well defined design patterns can save time and increase quality of the software**
- **Agile software development practices**
  - Agile software development is supported by a number of concrete practices, covering areas like requirements, design, modeling, coding, testing, planning, risk management, process, quality, etc.
    - https://en.wikipedia.org/wiki/Agile_software_development

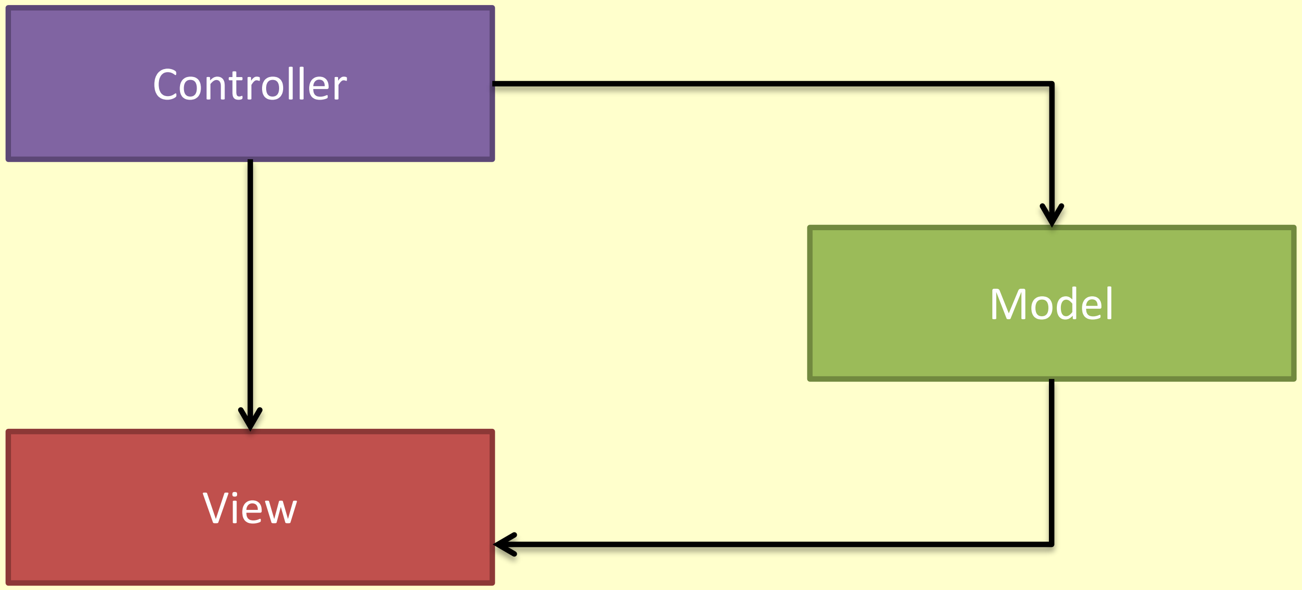
MVC - Model View Controller

- Model View Controller Pattern
  - Highly being used for web applications
  - Increases the flexibility of the application
  - an architecture in which we divide the application in 3 parts:
    - Input
    - Processing
    - Output

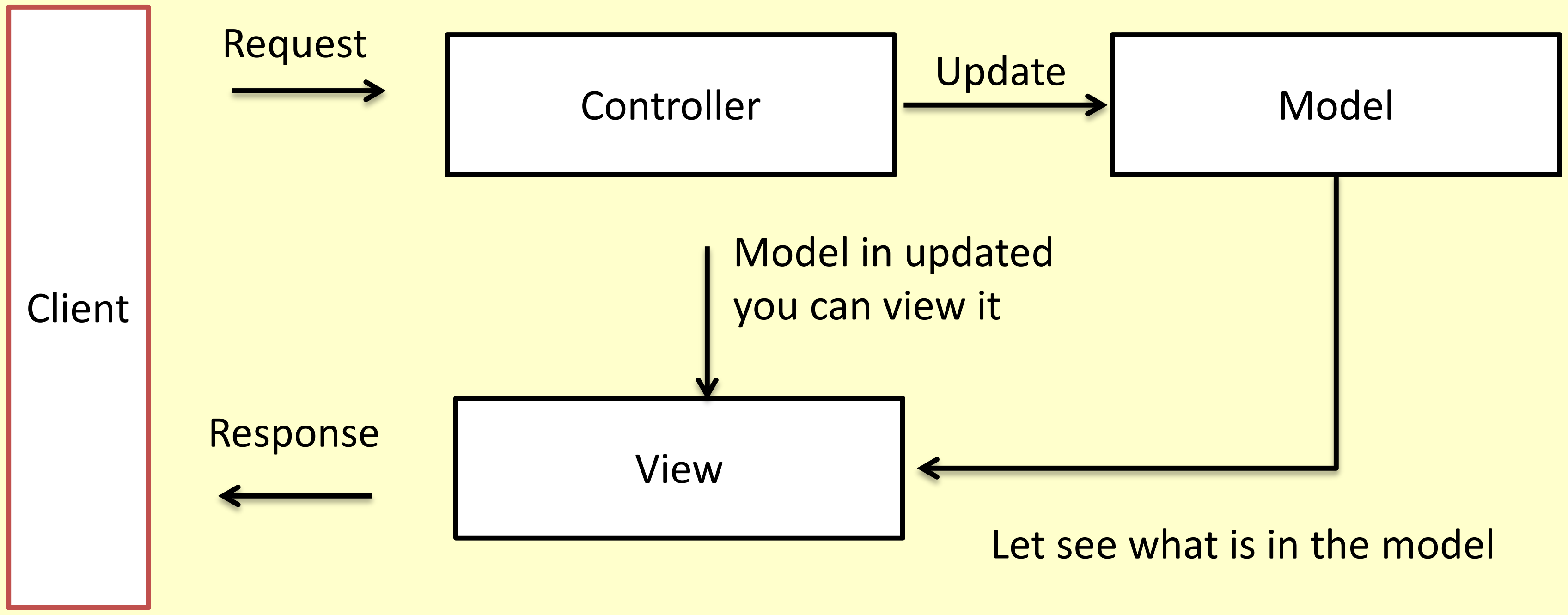
MVC Elements

- Model
  - Encapsulates core data and functionality
  - Independent of view
- View
  - Presents the output to the user
  - A view has an associated controller
- Controller
  - Receives inputs from the user
  - Processes inputs translating requests for the model

MVC Elements



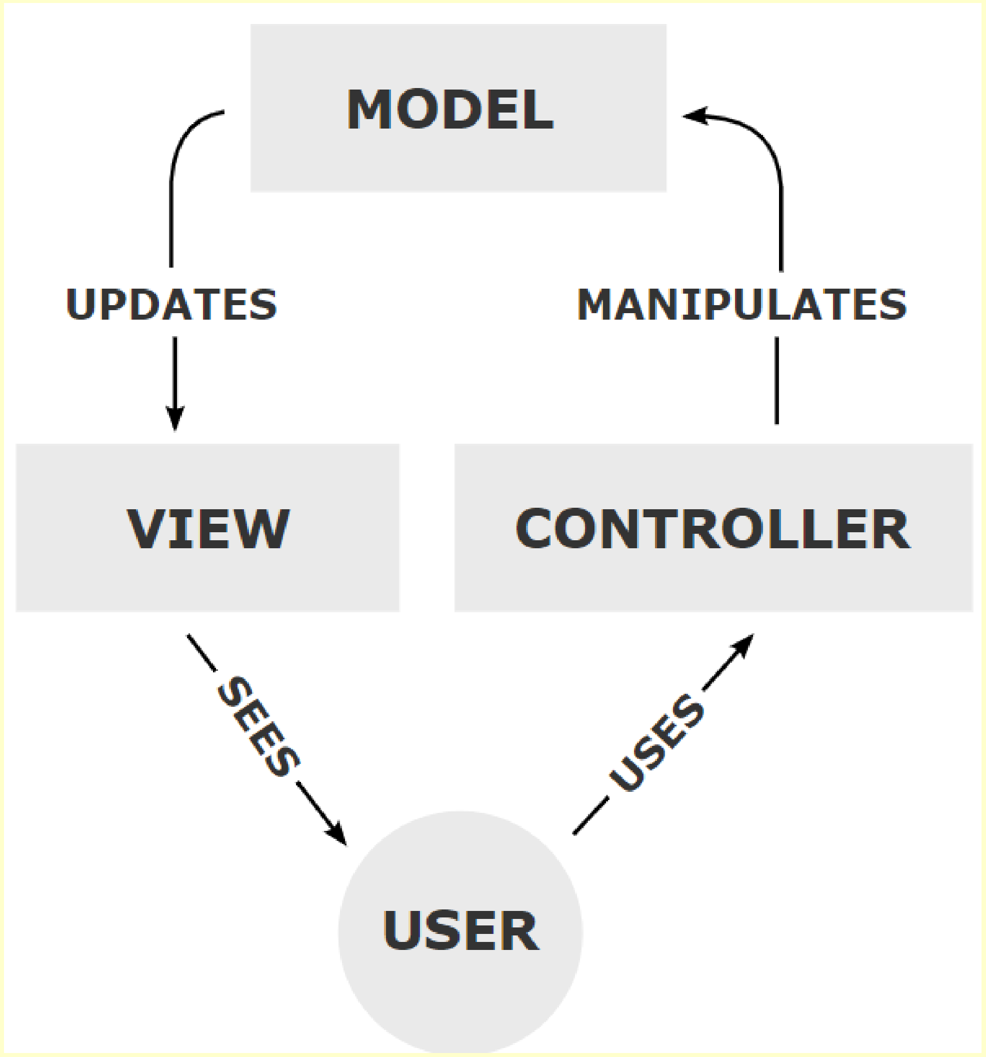
MVC in Web Application



MVC in Web Application

- Model
  - Implements the business logic and state of the application
  - Obtains and updates the state
  - The only part of the application that talks to the database
- View
  - Responsible for presentation of output and sending user requests to the controller
  - Obtains the state of the model through the controller but not directly – doesn't access the controller directly
- Controller
  - Processes user requests (HttpRequests) and translates them into requests for the model and/or view
  - Tells the model to update itself
  - Makes the model available to the view (through request and / or session attributes)

https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller#/media/File:MVC-Process.svg

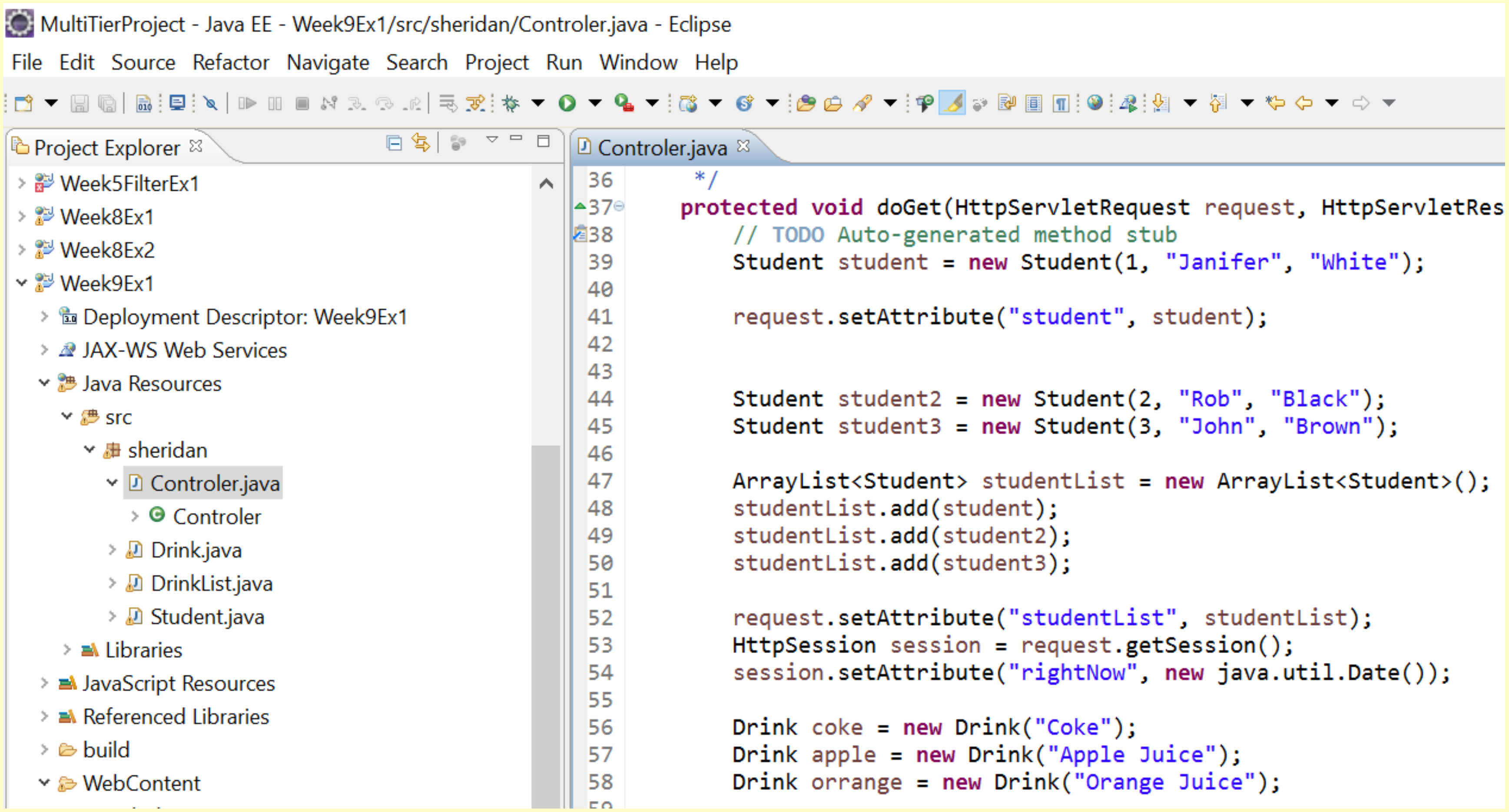


MVC

Available technologies

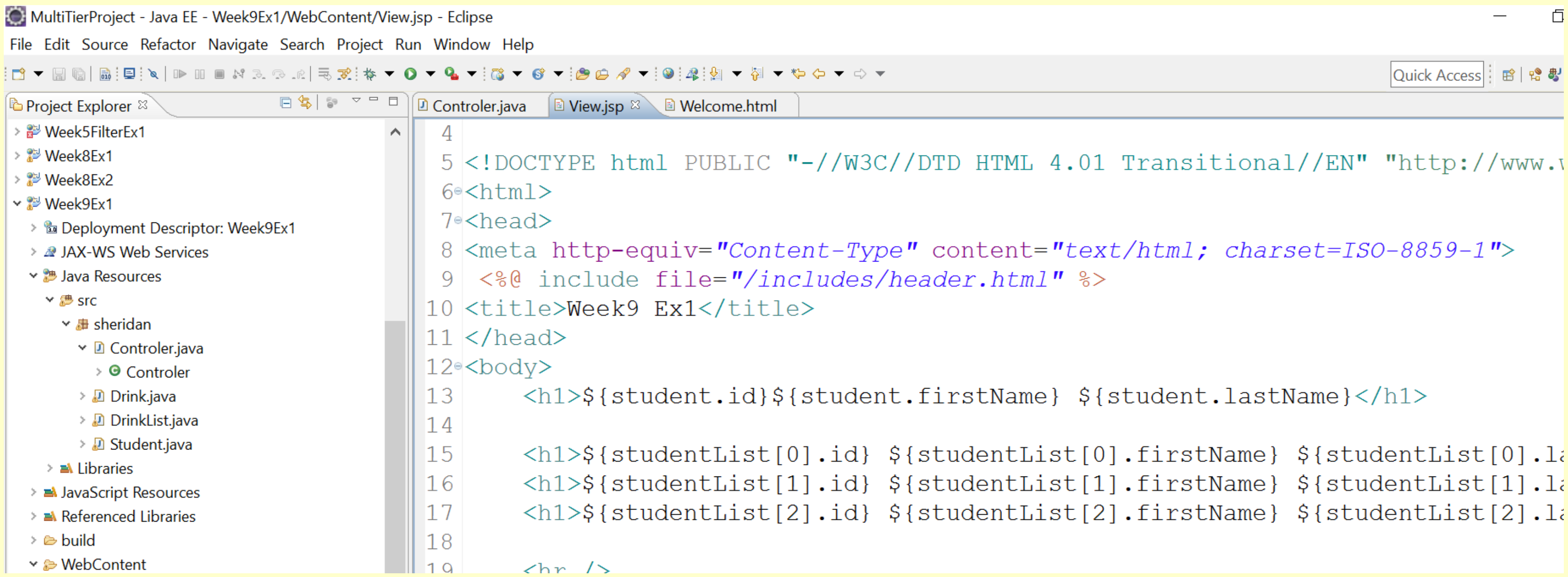
- View
  - HTML, Java Script, JSP, Applet
- Controller
  - Servlets, Rarely JSP
- Model
  - POJO, EJB, Web Service

Example - Controler



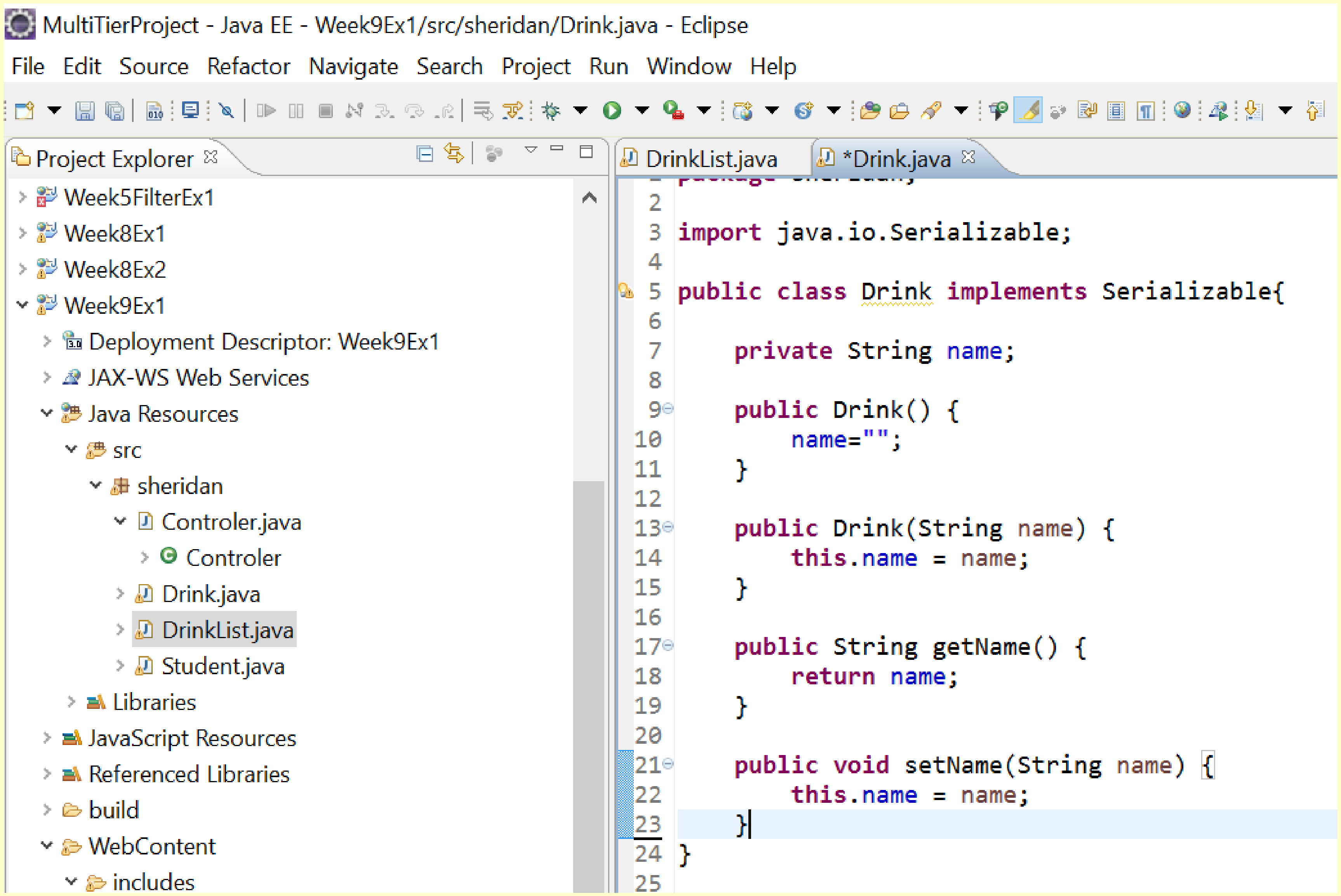
```
     */
    protected void doGet(HttpServletRequest request, HttpServletRes
        // TODO Auto-generated method stub
        Student student = new Student(1, "Janifer", "White");

        request.setAttribute("student", student);


        Student student2 = new Student(2, "Rob", "Black");
        Student student3 = new Student(3, "John", "Brown");

        ArrayList<Student> studentList = new ArrayList<Student>();
        studentList.add(student);
        studentList.add(student2);
        studentList.add(student3);

        request.setAttribute("studentList", studentList);
        HttpSession session = request.getSession();
        session.setAttribute("rightNow", new java.util.Date());

        Drink coke = new Drink("Coke");
        Drink apple = new Drink("Apple Juice");
        Drink orrange = new Drink("Orange Juice");
```

Example - View



```

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
 <%@ include file="/includes/header.html" %>
<title>Week9 Ex1</title>
</head>
<body>
    <h1>${student.id}${student.firstName} ${student.lastName}</h1>

    <h1>${studentList[0].id} ${studentList[0].firstName} ${studentList[0].la
    <h1>${studentList[1].id} ${studentList[1].firstName} ${studentList[1].la
    <h1>${studentList[2].id} ${studentList[2].firstName} ${studentList[2].la

    <hr />
```

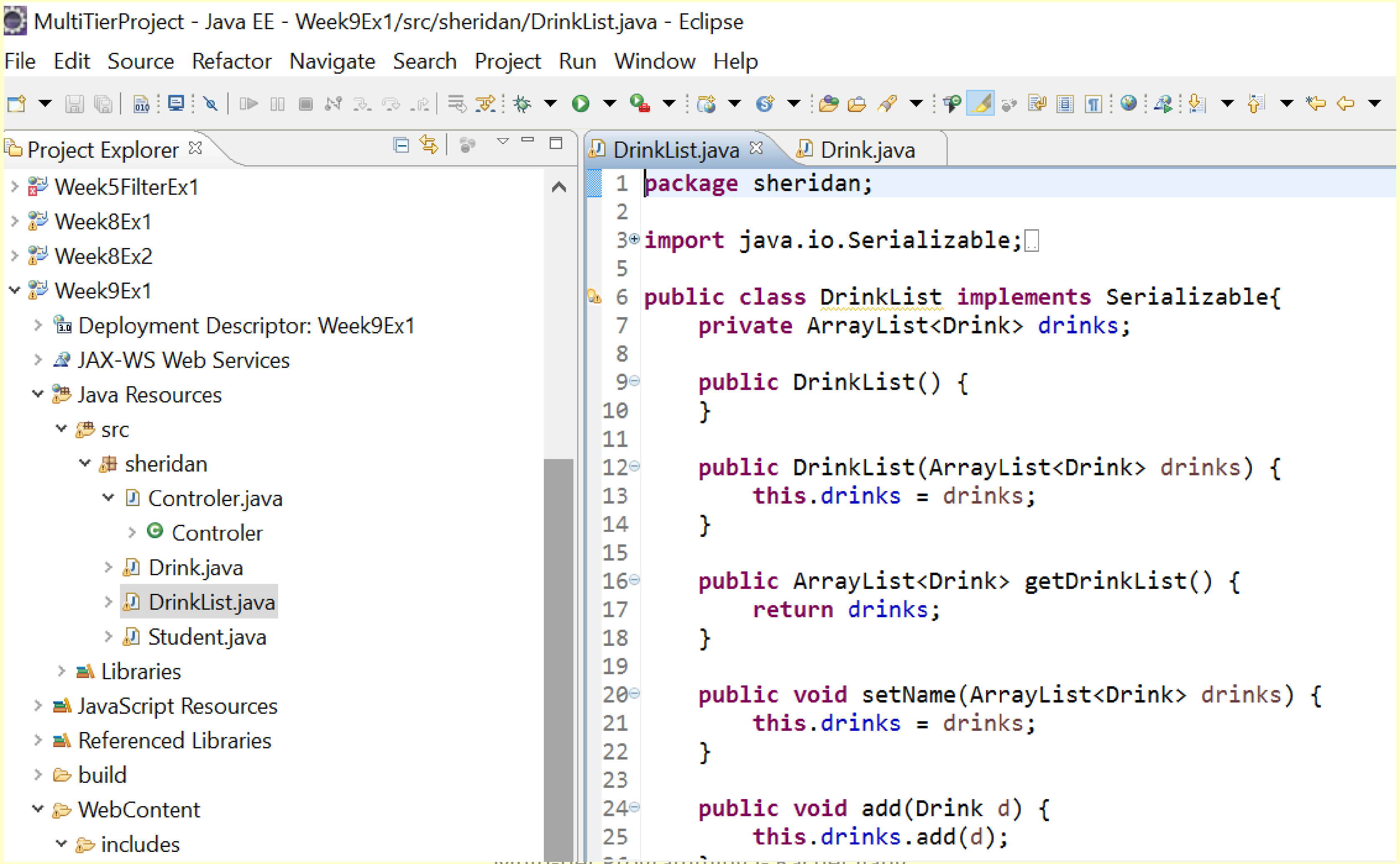
Example - model



MultiTierProject - Java EE - Week9Ex1/src/sheridan/Drink.java - Eclipse

```java

import java.io.Serializable;

public class Drink implements Serializable{

    private String name;

    public Drink() {
        name="";
    }

    public Drink(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

```

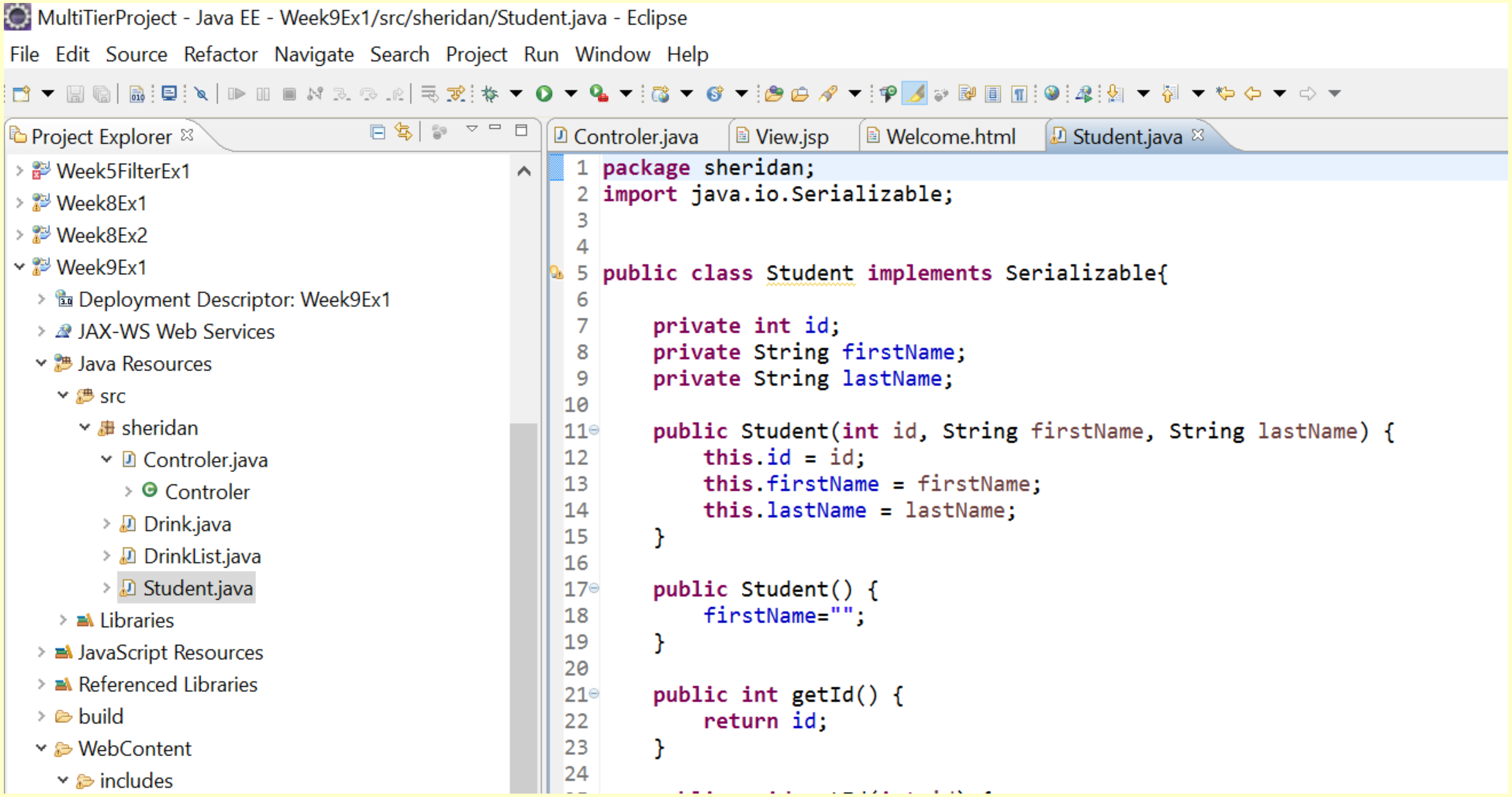
Example - model


MultiTierProject - Java EE - Week9Ex1/src/sheridan/DrinkList.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
DrinkList.java
Drink.java

```
Week5FilterEx1
Week8Ex1
Week8Ex2
Week9Ex1
  Deployment Descriptor: Week9Ex1
  JAX-WS Web Services
  Java Resources
    src
      sheridan
        Controler.java
          Controler
        Drink.java
        DrinkList.java
        Student.java
    Libraries
  JavaScript Resources
  Referenced Libraries
  build
  WebContent
    includes
```

```java
package sheridan;

import java.io.Serializable;

public class DrinkList implements Serializable{
    private ArrayList<Drink> drinks;

    public DrinkList() {
    }

    public DrinkList(ArrayList<Drink> drinks) {
        this.drinks = drinks;
    }

    public ArrayList<Drink> getDrinkList() {
        return drinks;
    }

    public void setName(ArrayList<Drink> drinks) {
        this.drinks = drinks;
    }

    public void add(Drink d) {
        this.drinks.add(d);
```

Example - model

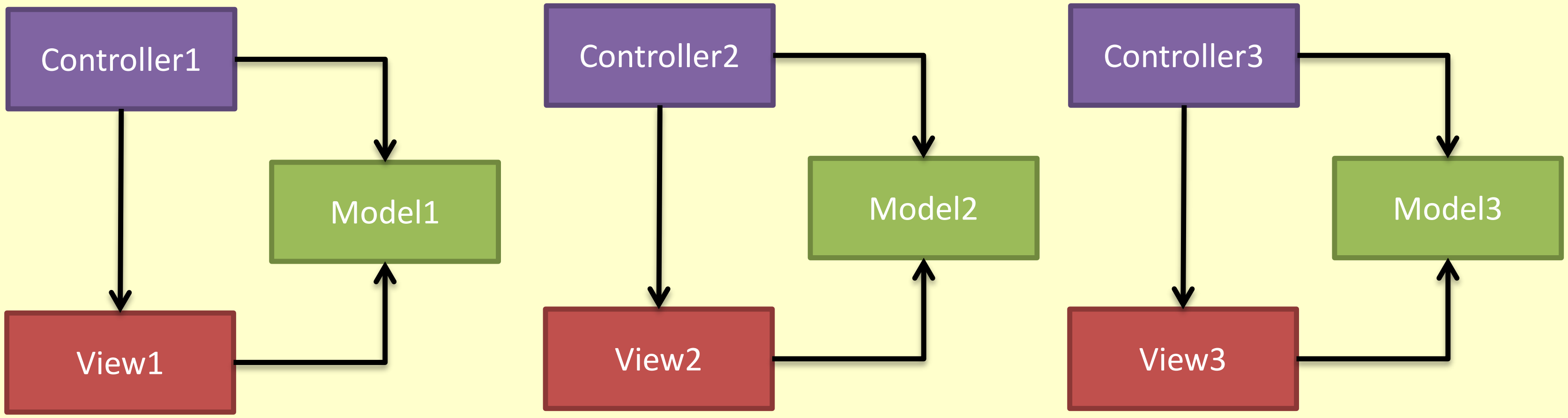


```java
package sheridan;
import java.io.Serializable;


public class Student implements Serializable{

    private int id;
    private String firstName;
    private String lastName;

    public Student(int id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public Student() {
        firstName="";
    }

    public int getId() {
        return id;
    }

```

MVC

- There is a problem with MVC
  - For each view you need a controller
  - Applying MVC in a web application may lead to a proliferation of controllers
  - Controllers are not cohesive and they are repetitive: parse arguments, validate, set, forward to view
  - Hard to manage
  - Hard to maintain

MVC

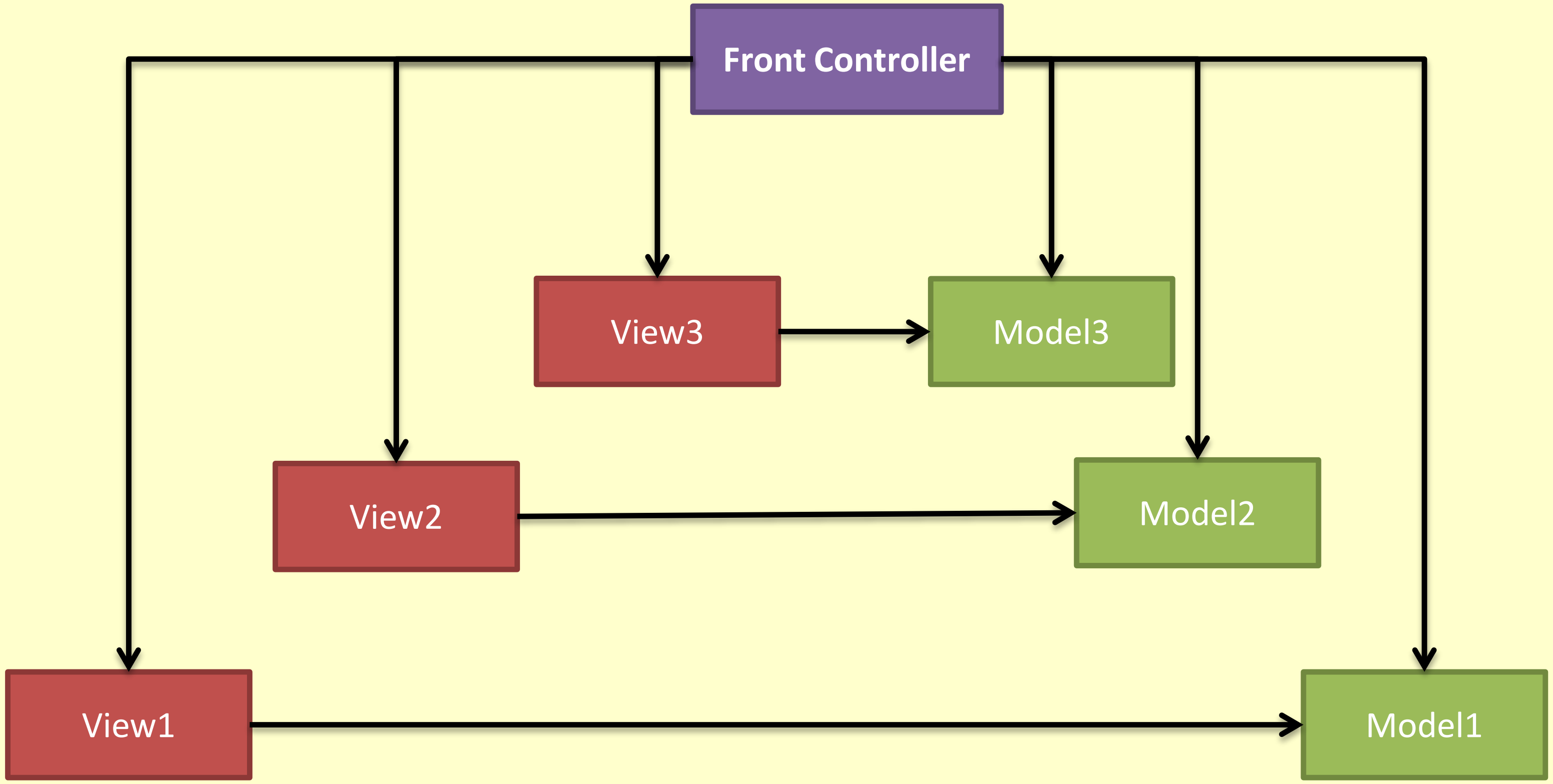

Controller1
Model1
View1
Controller2
Model2
View2
Controller3
Model3
View3

Front Controller Pattern

- Have only one controller which acts like a dispatcher – the front controller

- The controller is too large?
  - Distribute controller responsibilities to smaller more cohesive objects (e.g. validators, action objects)

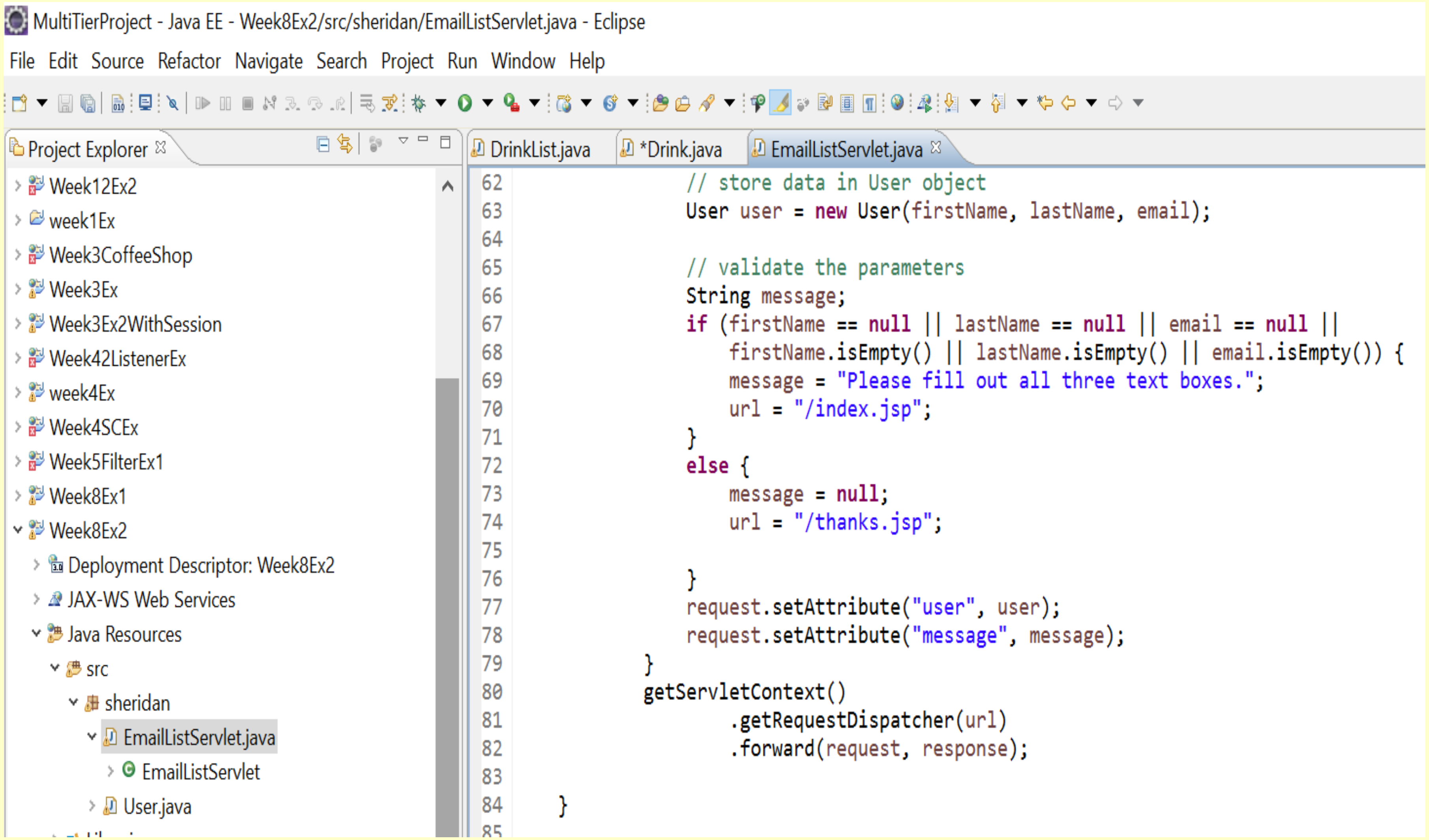
Front Controller Pattern



How does it look like at implementation?

How does it look like at implementation?

- See example of week8 Ex2
  - If () url=… else url=……, else url = ……….
- getServletContext().getRequestDispatcher(url)
  .forward(request, response);

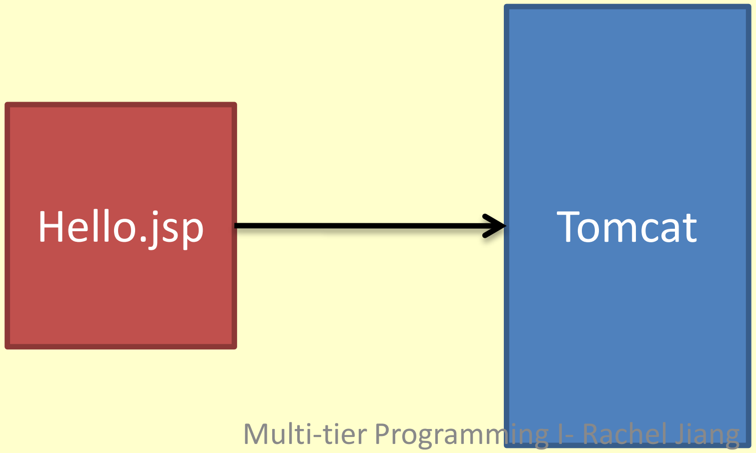


```java
            // store data in User object
            User user = new User(firstName, lastName, email);

            // validate the parameters
            String message;
            if (firstName == null || lastName == null || email == null ||
                firstName.isEmpty() || lastName.isEmpty() || email.isEmpty()) {
                message = "Please fill out all three text boxes.";
                url = "/index.jsp";
            }
            else {
                message = null;
                url = "/thanks.jsp";

            }
            request.setAttribute("user", user);
            request.setAttribute("message", message);
        }
        getServletContext()
                .getRequestDispatcher(url)
                .forward(request, response);

    }
```

JSP Life Cycle

- Web application develops the JSP pages and deploys them on the container.

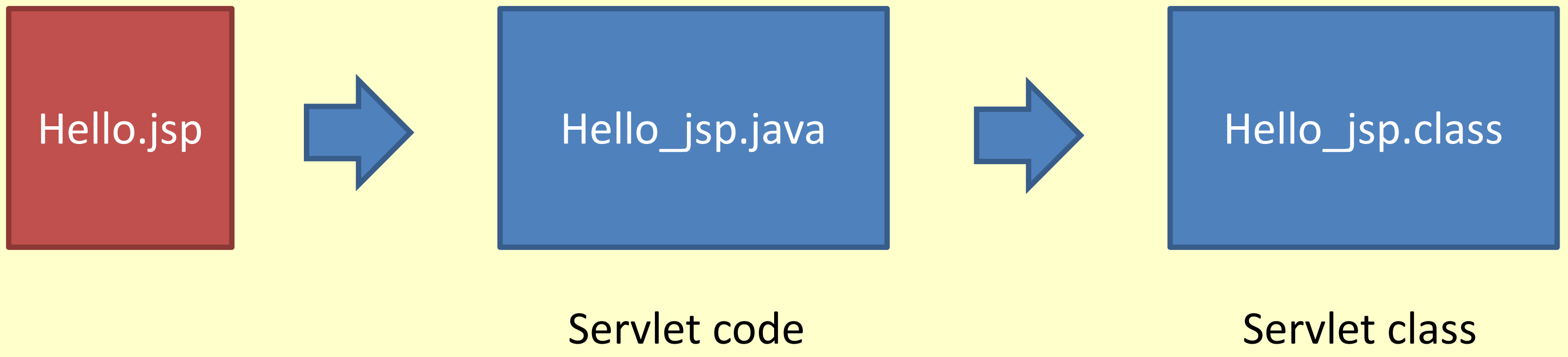
- JSP pages remain on the server until someone sends a request



JSP Life Cycle

- When the user requests the page for the first time
  - The container
    - translates the JSP to corresponding servlet (Hello_jsp.java)
    - compiles the servlet and generates the .class file (Hello_jsp.class)
    - loads the newly generated servlet class
    - calls the init() method
  - The service() method takes care of the request

JSP Life Cycle



- This happens only the first time
- After this everything is the same as servlets since container is dealing with the servlet

Generated Servlet

- After translating the JSP to servlet the following methods will be created
  - jspInit()
  - jspDestroy()
  - _jspService()
- Our code goes to ?

Generated Servlet

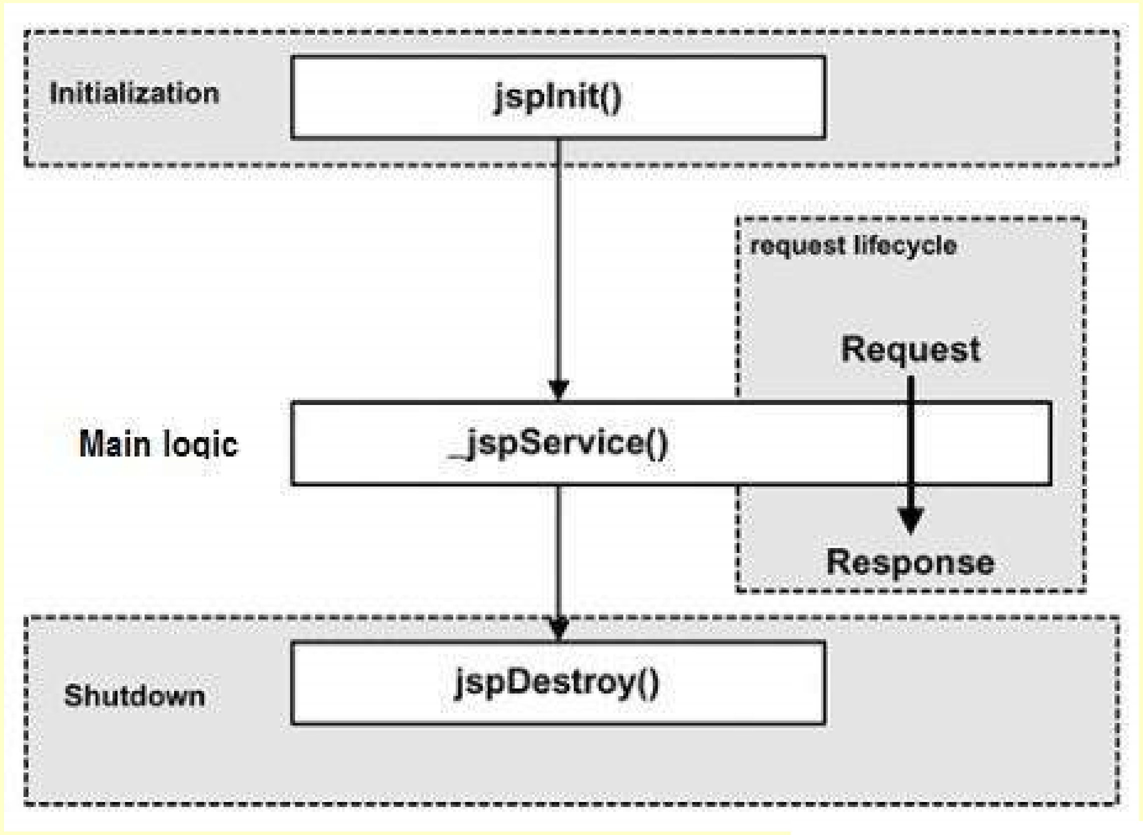
- What if we want to add stuff to jspInit() and jspDestroy()
  - We can override them
  - How?

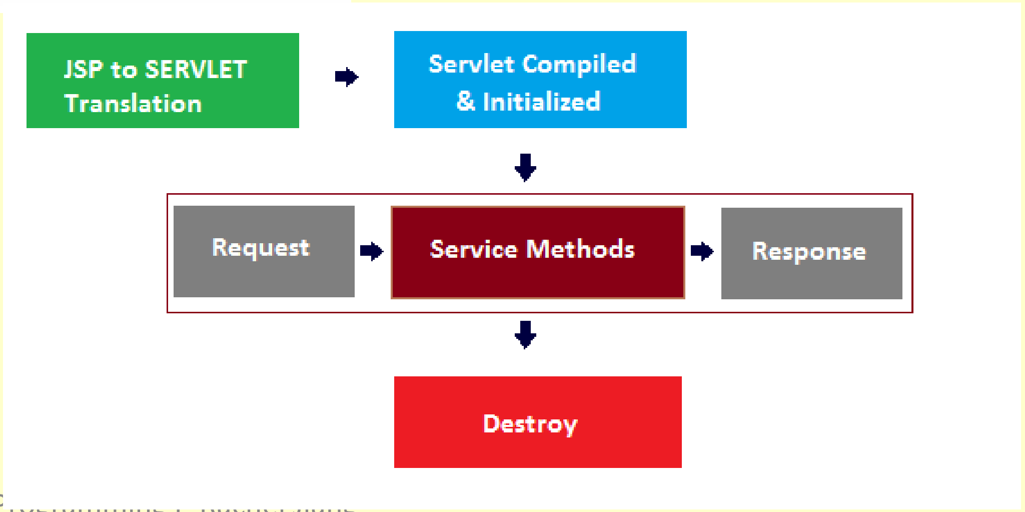
Managing Life-Cycle

```
<%! public void jspInit()
{
  // ...perform one time initialization.
  // ...this method is called only once per JSP, not per request
} %>
```

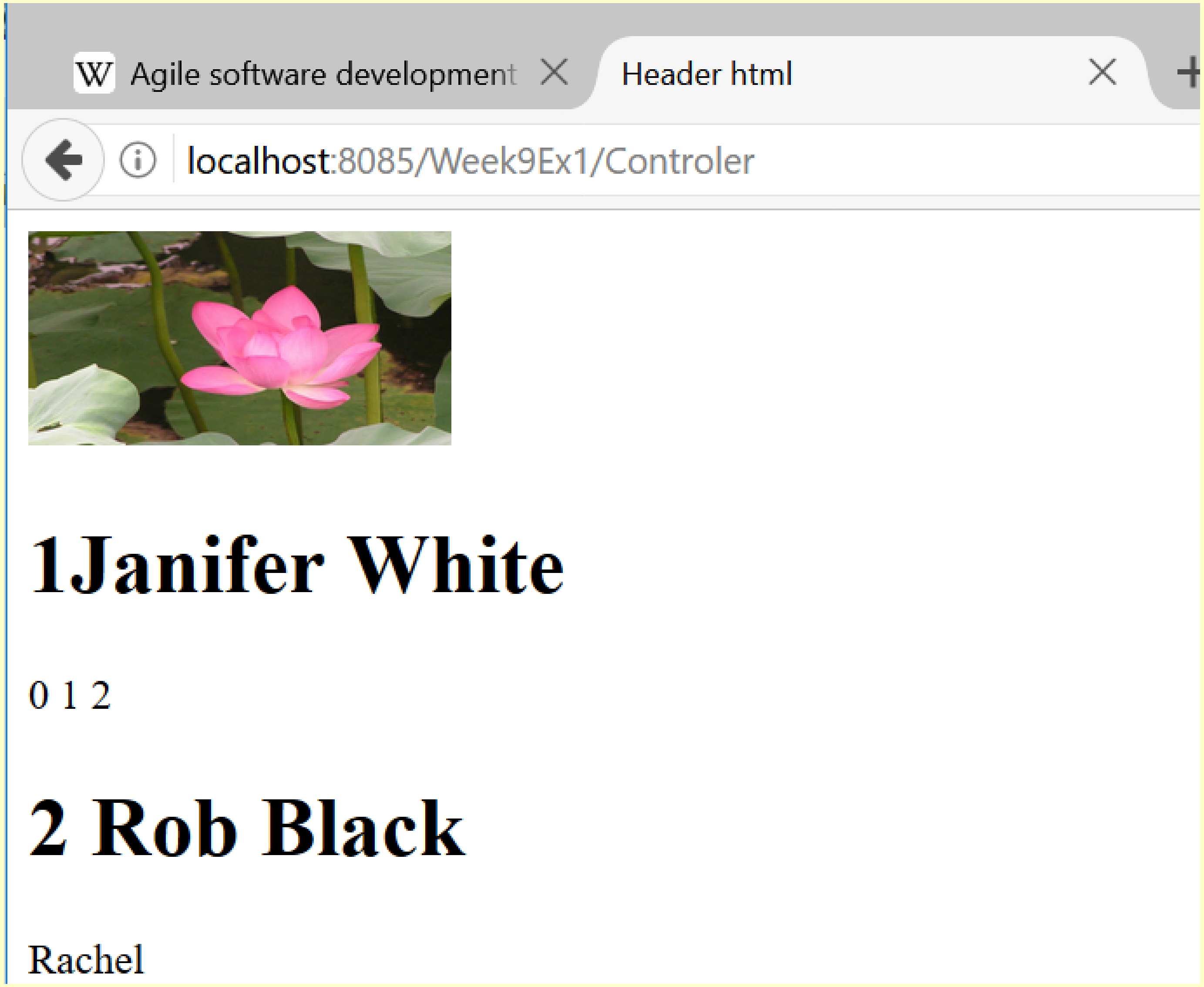
```
<%! public void jspDestroy()
{
   // ...perform one time cleanup of resources
} %>
```

You cannot override _jspService()


Initialization
jspInit()
request lifecycle
Request
Main logic
_jspService()
Response
Shutdown
jspDestroy()


JSP to SERVLET Translation
Servlet Compiled & Initialized
Request
Service Methods
Response
Destroy

example



```
localhost:8085/Week9Ex1/Controler
```

```
1Janifer White

0 1 2

2 Rob Black

Rachel
```

Inclass Ex

- Create a new project called MVCex
  - can get the web.xml (DD) option check box checked
- Create a javabean class called “Student” under a package named “sheridan”.
- Create a controller (Servlet) called “Controler” to get inputs from a html/jsp that will have a form defined to collect student information (ID, firstname, last name)
    - <form action=*“Controler" method="Post"*>
- Output the name(s) in the view.jsp witch is dispatched by the controller and sent to the browser.

References

- Head First Servlets and JSP , Second Edition
  - Chapter 4 and 7