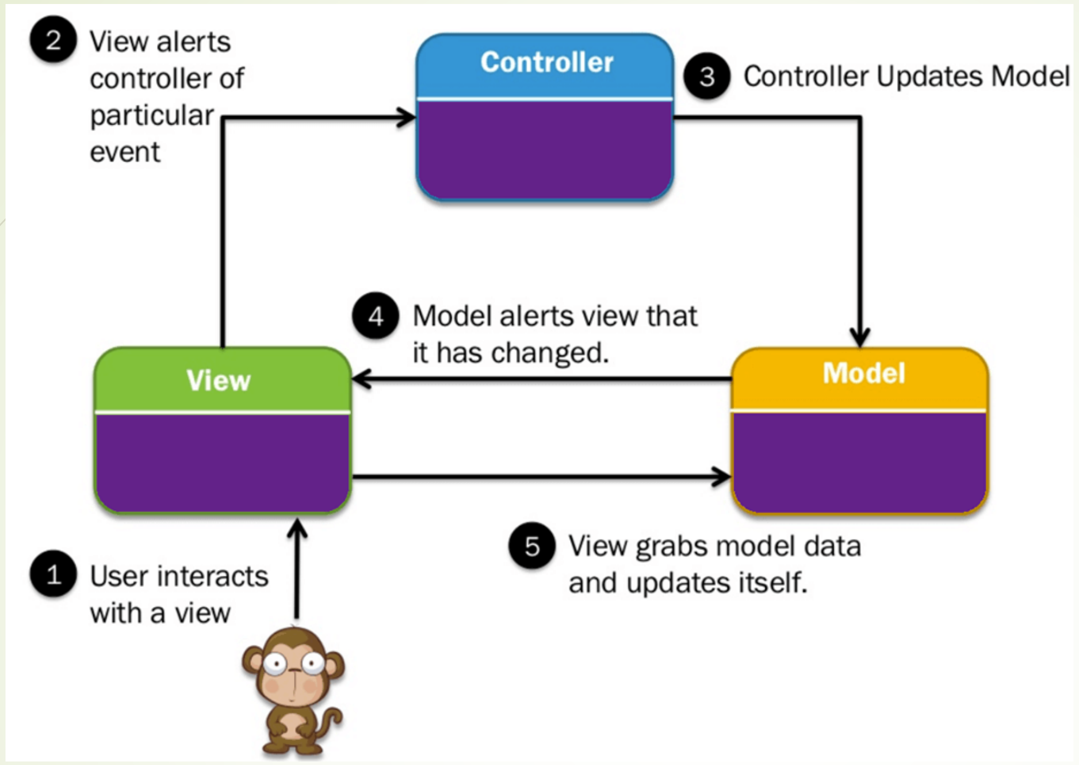
SOEN 387 WEB-BASED ENTERPRISE APPLICATIONS DESIGN

MVC Architecture


2 View alerts controller of particular event
Controller
3 Controller Updates Model
4 Model alerts view that it has changed.
View
Model
1 User interacts with a view
5 View grabs model data and updates itself.

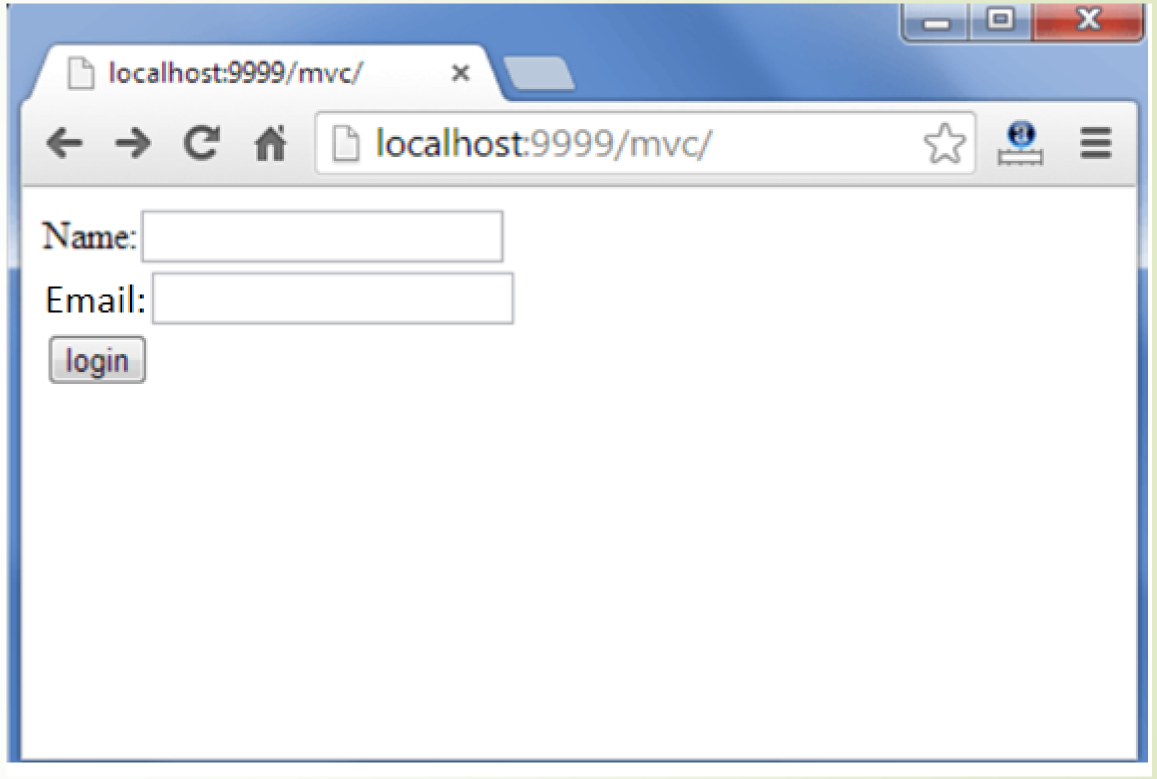
MVC Interaction

Example 1

- In this exercise we are implementing MVC for registration functionality
- A registration page displays fields for username and email
- By clicking the submit button, the system displays the successful registration message

View

```
<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=ISO-8859-1">
<title> Registration Page</title>
</head>
<body>
<form action="MyControllerServlet.jsp"
method="POST">
Name: <input type="text" name="username"><br>
Email: <input type="text" name="email"><br>
<br />
<input type="submit" value="Submit"/>
</form>
</body>
</html>
```



Controller

```
public class MyControllerServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();

        String name=request.getParameter("username");
        String email=request.getParameter("email");

        LoginBean bean=new LoginBean();
        bean.setName(name);
        bean.setEmail(email);
        request.setAttribute("bean",bean);
        RequestDispatcher rd=request.getRequestDispatcher("registration-success.jsp");
        rd.forward(request, response);
        }
}
```

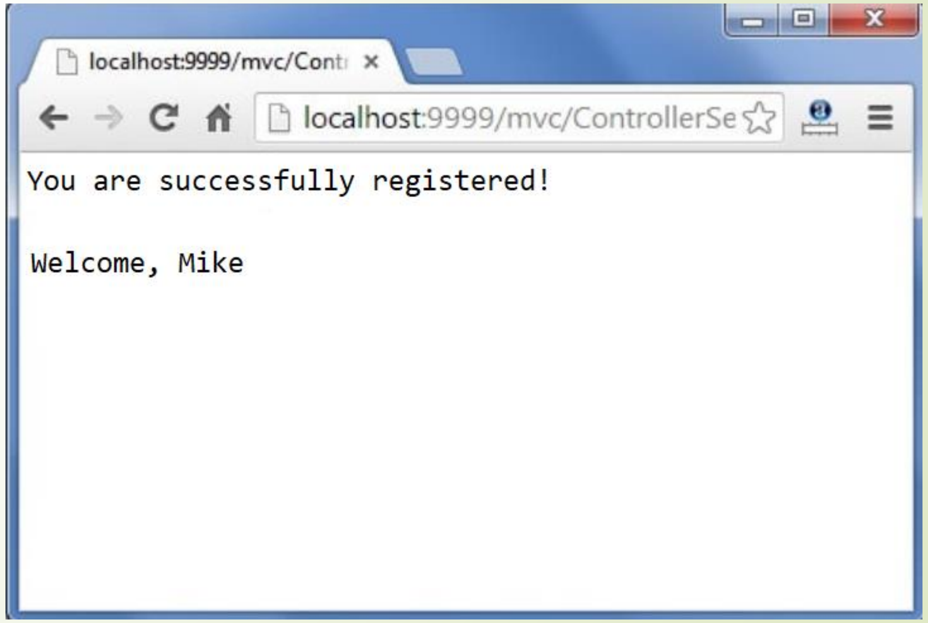
Model

```
public class LoginBean{
        private String name, email;
        public String getName() {
            return name;
        }
        public void setName(String name) {
            this.name = name;
        }
        public String getEmail() {
            return email;
        }
        public void setEmail(String email) {
            this.email = email;
        }
    }
}
```

View Class

registration-success.jsp

```
<jsp:useBean id= “bean" class= “com.LoginBean“/>
 <% out.print(“You are successfully registered!
Welcome, "+bean.getName()); %>
</jsp:useBean>
```



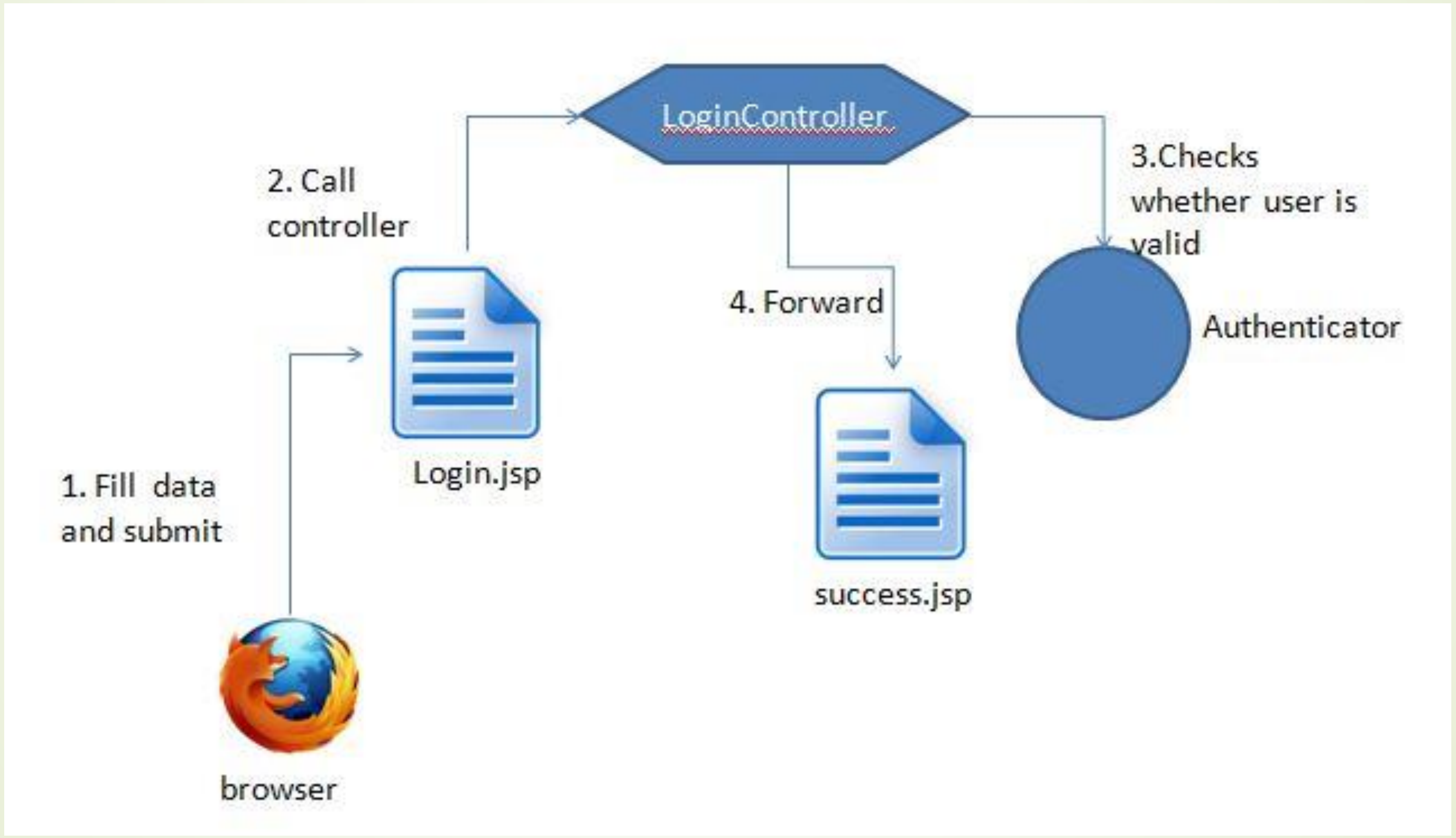
```
You are successfully registered!

Welcome, Mike
```

Refer Link

Example 2

- In this exercise we are implementing MVC for login functionality for user
- A login page displays fields for username and password
- By clicking the login button, the system will validate the user
- Displays welcome message on successful login or If the login fails, it will redirect to an error page

MVC implementation for Login



refer Link

Example 3

- In this exercise we are implementing MVC architecture to perform the basic operations of addition, subtraction, multiplication and division for a calculator
- A Calculator page displays 2 textbox for entering integers and a select option for operators(+,-,*,/)
- By clicking calculate button, the system will display results

View Class

Calculator.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<html>
<head>
<title>calculator</title>
</head>
<body>
<h2>Calculator</h2><br>
<form method=post action="calculator">
 <input type=text name=number1  class="a">
 <select name=operator class="a">
  <option selected>+</option>
  <option>-</option>
  <option>*</option>
  <option>/</option>
 </select>
 <input type=text name=number2 >
 <input type=submit value="=" >
</form> </body> </html>
```

Controller Class

CalculatorServlet.java (Servlet)

```
public class CalculatorServlet extends HttpServlet {
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
  int number1, number2;
  String operator;
  response.setContentType("text/html");
    response.setCharacterEncoding("UTF-8");
    PrintWriter out = response.getWriter();
    number1 = Integer.parseInt(request.getParameter("number1"));
    number2 = Integer.parseInt(request.getParameter("number2"));
    operator = request.getParameter("operator");

    OperatingClass oc = new OperatingClass(number1, number2, operator);
    request.setAttribute("bean",oc.calc());
    RequestDispatcher rd=request.getRequestDispatcher("calculation.jsp");
    rd.forward(request, response);
   }
}
```

Model Class

OperatingClass.java

```
public class OperatingClass {
 private int number1;
 private int number2;
 private String operator;
 private int result1;

 public OperatingClass(int number1, int number2,
String operator) {
  this.number1 = number1;
  this.number2 = number2;
  this.operator = operator;
 }

  public int getResult1() {
  return result1;
        }
```

```
public void calc() {
  switch(operator) {
     case "+":
      result1=number1+number2;
      break;
     case "-":
      result1=number1-number2;
      break;
     case "*":
      result1=number1*number2;
      break;
     case "/":
      result1=number1/number2;
      break;
     }
 }
 }
}
```

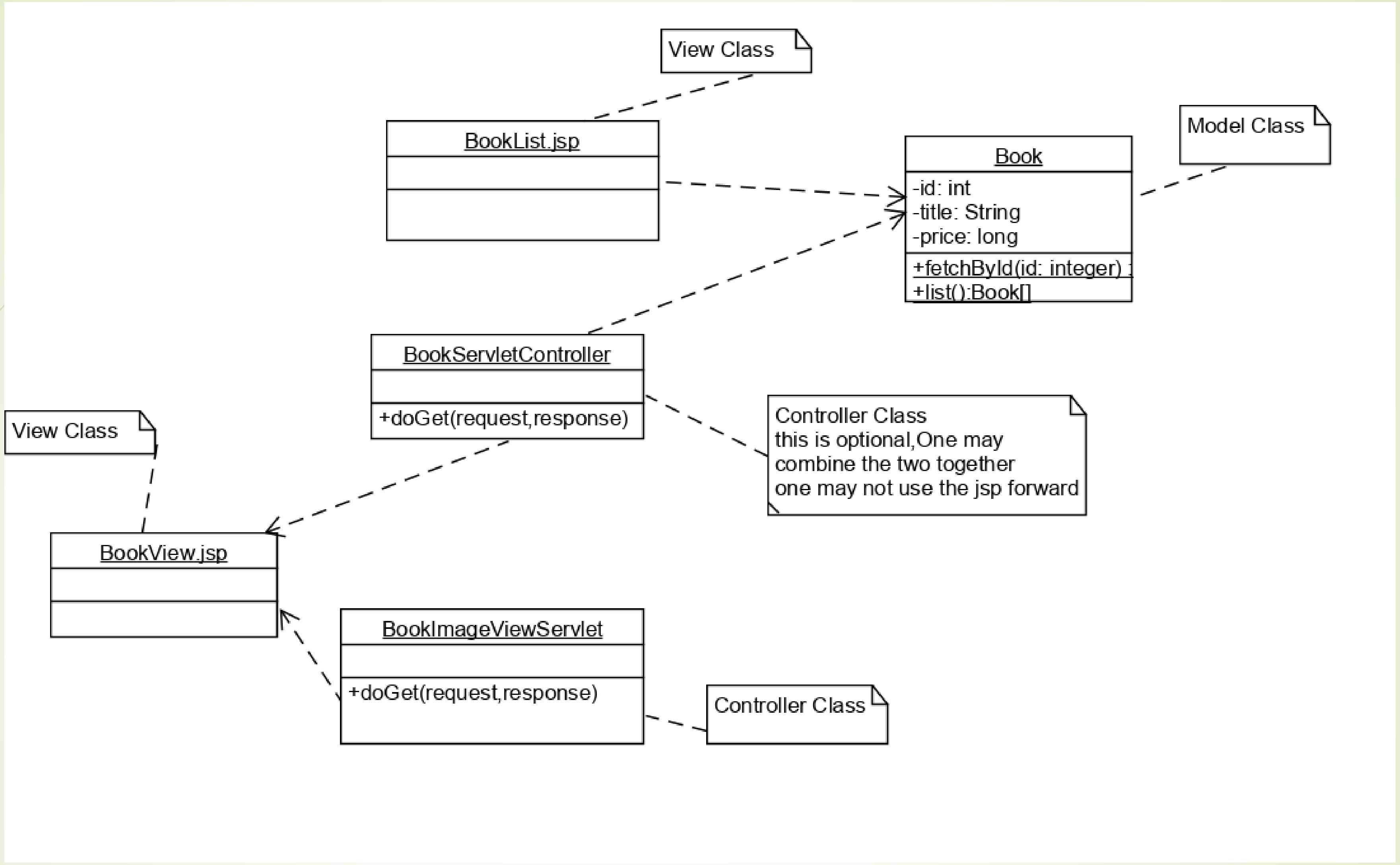
View Class

Calculation.jsp

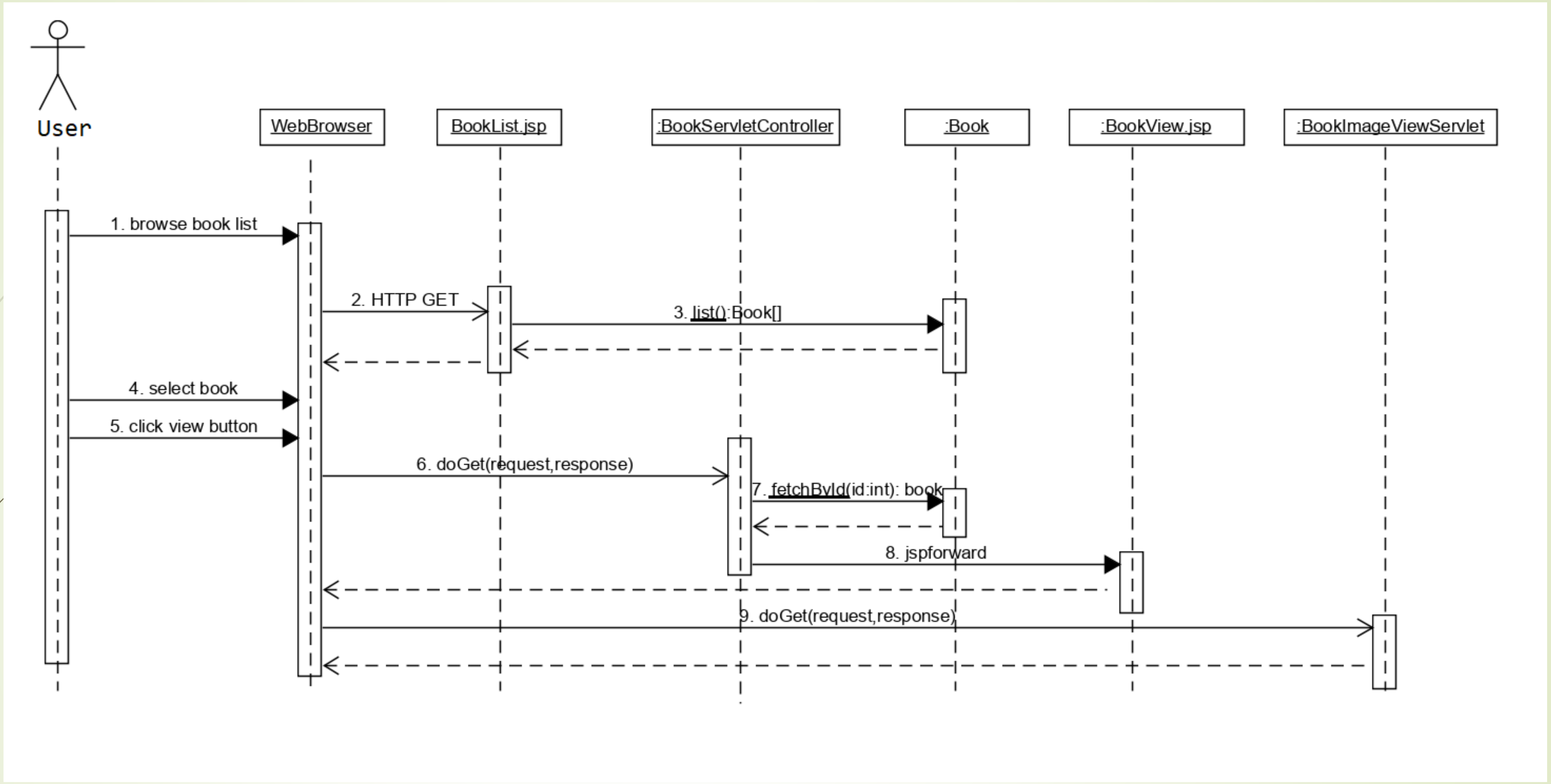
```
<jsp:useBean id= “operatingClassBeanObj" class= “com.OperatingClass“/>
 <% out.print (“Answer=, "+ operatingClassBeanObj. getResult1()) ); %>
</jsp:useBean>
```

Example 4

- In this exercise we are implementing a book list and a book view page.
- A book list page displays a list of books that are in the inventory.
- By selecting a book from the book list and clicking on the view button, the system displays the book information in book view page.
- The book view page displays the book cover image using a servlet.



Class diagram



Sequence Diagrams

References

- https://www.javatpoint.com/MVC-in-jsp
- https://www.guru99.com/mvc-tutorial.html
- https://www.baeldung.com/mvc-servlet-jsp
- https://phitchuria.wordpress.com/2017/11/16/jspservlet-simple-calculator/