Architecture MVC (Model View Controller)

- **Objective:** to assure division of tasks;
- **Model:** classes that represent entities and manipulate data. It contains the business rules;
- **View**: presentation of results in a Web page;
- **Controller**: servlets with business logic.
- **Exercise 1:** In the project "`class5`", modify the class `InsertStudent`. Let's start to separate the logic of the Servlet from the answer to the user. Create a file named "`student-inserted.jsp`" with the code:

```
<body>
      Student ${param.name} successfully added!
</body>
```

Modify the class `InsertStudent` to redirect the answer instead of printing it in HTML. Right after the command `dao.insert(student)`, type:

```
// redirect the information to a JSP file
RequestDispatcher rd = request.getRequestDispatcher("/student-
inserted.jsp");
rd.forward(request, response);
```

Control: reducing the number of servlets

- **Problem of administration:** a servlet for each different logic for each model is impractical for real applications;
- **An option:** for each model, we group all operations inside the same servlet.

  Exercise 1: Modify the project "`class5`", grouping in one only servlet the operations of insertion, updating and removal for the model student. First, create a servlet called `ControllerServlet` with the following code:

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

	@Override
	protected void service(HttpServletRequest request, HttpServletResponse
				response) throws ServletException, IOException {
		String objective = request.getParameter("objective");

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

		Student student = new Student();
		student.setName(name);
		student.setEmail(email);
		student.setAddress(address);
		StudentDAO dao = new StudentDAO();

		if (objective.equals("Insert")){
			dao.insert(student);
			RequestDispatcher rd =
				request.getRequestDispatcher("/listtaglib.jsp");
			rd.forward(request, response);
		} else if (objective.equals("Update")){
			student.setId(Long.valueOf(request.getParameter("id")));
			dao.update(student);
			RequestDispatcher rd =
				request.getRequestDispatcher("/listtaglib.jsp");
			rd.forward(request, response);
		} else if (objective.equals("Remove")){
			student.setId(Long.valueOf(request.getParameter("id")));
			dao.remove(student);
			RequestDispatcher rd =
				request.getRequestDispatcher("/listtaglib.jsp");
			rd.forward(request, response);
		}
	}
}
```

- Click in each line with an error and add the following imports using Ctrl + 1:

```
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import br.edu.ufabc.prograd.dao.StudentDAO;
import br.edu.ufabc.prograd.model.Student;
```

- Modify `listtaglib.jsp` to construct a list of students with options to make insertions, updates and removals. Also, add the field `id` for each student in a new column of the table:

```
<td align="center">${student.id}</td>
```

- Below the table **add inputs so the user can provide information for a new student and/or to update the information of a student already in the database**. **Don't forget to insert the tag <form></form> with an action indicating the servlet.** The field `id` of the student should not allow changes, as follows:

```
Id:<input type="text" id="id" name="id" readonly style="color:#AAAAAA"/>
```

- **Adding radiobox in the table to list students.** Create a new column in the table containing a radiobox whose value is the student's id in the database.

```
<td align="center">
      <input type="radio" name="group1" value="${student.id}">
</td>
```

- **Implementing copy of data from a line of the table to the fields of the form.** For that, we need to identify the line containing the `radiobox` that is clicked, so we can take the data from the line and add them in the `input's`. We associate an id for each line:

```
<tr id="row_${student.id}" bgcolor="#${counter.count % 2 == 0 ?
'99FFFF' : 'FFFF99' }">
```

- In the event `onclick` of the radiobox, that is, every time that the user clicks in this radiobox, we will call a function named checkRadio(value) made in Javascript:

```
<td align="center">
            <input type="radio" name="group1" value="${student.id}"
            onClick="checkRadio(value)">
      </td>

      <script type="application/javascript">
      1     function checkRadio(name){
      2           document.getElementById('id').value=name;
      3           var elTableRow = document.getElementById('row_'+name);
      4           var elTableCells = elTableRow.getElementsByTagName("td");
      5           document.getElementById('name').value=trim(elTableCells[2
                        ].textContent);
      6           document.getElementById('email').value=trim(elTableCells[3]
                        .textContent);
      7           document.getElementById('address').value=trim(elTableCells[4]
                        .textContent);
      8     }

            function trim(str) {
                  return str.replace(/^\s+|\s+$/g, '');
            }
      </script>
```

In the line 2 of the function `checkRadio(name)`, we take the `id` of the student from the database (which is also the name of the `radiobox` in that line), and we add it into the `input` Id. In the line 3, we search the line containing the selected `radiobox`. In the line 4, we take the columns inside the selected line. In the lines 5, 6 and 7, we add the data from the table to the corresponding `input's`.

- **Adding many `submit` buttons to the form.** Add buttons with the same value that is expected in the servlet (Insert, Update or Remove). This information must be added to the `hidden input` named `"objective"`. This parameter is responsible to indicate to the servlet which action must be performed. Therefore, in the button's `onclick` event, we add the value to the `hidden input`.

```
<input type="hidden" id="objective" name="objective" value="" /><br />
<input type="submit" value="Insert" onclick="set(value)"/>
<input type="submit" value="Update" onclick="set(value)"/>
<input type="submit" value="Remove" onclick="set(value)"/>

<script type="application/javascript">
      function set(action){
            document.getElementById('objective').value=action;
      }
</script>
```

- **Returning to the same page after submitting the form.** Instead of redirecting to another page, we redirect the application to the same page from which the servlet was requested.

```
RequestDispatcher rd = request.getRequestDispatcher("/listtaglib.jsp");
rd.forward(request, response);
```

- However, if the user reloads the page now, the system can do again the action just made. In order to avoid this, we can replace the two lines above by the following line that just redirects without sending data:

```
response.sendRedirect("/class5/listtaglib.jsp");
```

- **Adding a hyperlink "Remove" for each line of the table.** Remove the button “Remove” from the form. Add a new column to the table with a hyperlink "Remove":

```
<td>
      <a href="#" id="link_${student.id}"
            onclick="set('Remove'); prepareRemoval(this.id);
            document.getElementById('formStudents').submit(); return
            false;">Remove</a>
</td>
```

Modify the form adding an id "formStudents" to it:

```
<form name="formStudents" id="formStudents" action="mvc" method="POST">
```

Add `"#"` in the `"a href"` of the hyperlink to indicate that its action is given in another place. In the `onclick` of the link, we add three functions, two of them we write in Javascript. The first one sets the `hidden input` for removal, and the second one adds data from the selected line inside the `input's` (which is the same used by the `radiobox's`). The last one is the action that submits the form. And finally, we have the `return false`, just in case the user has Javascript deactivated in his/her browser.

```
<script type="application/javascript">
      function prepareRemoval(str){
            var i=str.indexOf("_");
            str=str.substr(i+1,str.length);
            checkRadio(str);
      }
</script>
```

Control – reducing the number of servlets even more

- **Administration problem:** a servlet for each different logic and each model is impracticable for real applications;
- **Option that we already tried:** reducing to one servlet for each model, grouping all model operations inside a same servlet.
- **Even better:** reduce to **ONE** servlet for **ALL** application! In theory, we could implement one only servlet for all application by sending every logic and its operations to the same servlet, and inside it we would indicate the corresponding action for each one by using a `switch` or `if` statement.

  Disadvantage: the servlet would be gigantic.
- Let's use polymorphism. Modify the project "`class5`" and make it completely MVC.

- **Step 1.** In the package "`br.edu.ufabc.prograd.servlet`", modify the class `ControllerServlet` as follows:

```
	@WebServlet("/mvc")
	protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		String objective = request.getParameter("objective");
		String nameOfClass = "br.edu.ufabc.prograd.logic." + objective;

		try {
			Class myClass = Class.forName(nameOfClass);
			Logic logic = (Logic) myClass.newInstance();
			logic.execute(request, response);
		} catch (Exception e) {
			throw new ServletException("Error in the business logic",e);
		}
	}
```

- **Step 2.** Create a package "`br.edu.ufabc.prograd.logic`" and, inside it, create an interface called `Logic` with the following:

```
	public interface Logic {
		void execute(HttpServletRequest request, HttpServletResponse response)
			throws Exception;
	}
```

- **Step 3.** Still in the package "`br.edu.ufabc.prograd.logic`", create a class `InsertStudent` to insert students into the database:

```
public class InsertStudent implements Logic {
	@Override
	public void execute(HttpServletRequest request,HttpServletResponse response)
			throws Exception {
		// it searches parameters from request object
		String name = request.getParameter("name");
		String email = request.getParameter("email");
		String address = request.getParameter("address");

		// it instantiates object Student and inserts it into the database
		Student student = new Student();
		student.setName(name);
		student.setEmail(email);
		student.setAddress(address);
		StudentDAO dao = new StudentDAO();
		dao.insert(student);

		// it redirects the data to a JSP
		response.sendRedirect("/class5/listtaglib2.jsp");
	}
}
```

- **Step 4.** In the *JSP* file, attention to the details of the form whose action must be equal to the one in the `@WebServlet` mapped in the servlet:

```
<form name="register" id="form" action="mvc" method="POST">
```

- **Step 5.** Still in the *JSP* file, attention to the `hidden input`. The value sent to the servlet through this field must be equal to the name of the class that must be instantiated inside the servlet:

```
<input type="submit" value="Insert" onclick="set('InsertStudent')"/>
```

- **Exercise 1**. Create classes "*UpdateStudent*" and "*RemoveStudent*" to modify and remove a student, respectively. Add the three operations (insertion, update and removal) and also listing all students, where each line presents a `checkbox` to ease the choice of the record (as we made in the previous class).

- **Exercise 2.** Instead of implementing each logic to redirect to a *JSP* file, make the method `execute` return a string with the page address to which we must redirect. Then make the servlet use this string to redirect.

- **Exercise 3.** The logics `InsertStudent` and `UpdateStudent` are very similar. Try to create one only version for both.

Attaching data to a request

Suppose we want to make an operation in a database, and after the operation is successful, we want to forward the request to another page. For example, suppose we want to insert a student through the application "`class5`". Since we shouldn't abuse of Java code in the middle of HTML, we are going to use taglib. We do not need to change anything in the JSP file from which the request is coming. We just need to change the logic and the JSP file that will subsequently receive the request. So in the class `InsertStudent` we would make (in this case, the method `execute()` is not implemented with `return` and the redirecting is made directly in the logic):

```
package br.edu.ufabc.class5.logic;

import java.util.List;
import javax.servlet.RequestDispatcher;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import br.edu.ufabc.prograd.dao.StudentDAO;
import br.edu.ufabc.prograd.model.Student;

public class InsertStudent implements Logic {
      @Override
      public void execute(HttpServletRequest request, HttpServletResponse
response) throws Exception {
            // searching parameters from the request to insert
            String name = request.getParameter("name");
            String email = request.getParameter("email");
            String address = request.getParameter("address");

            // it instantiates object Student and inserts it into the database
            Student student = new Student();
            student.setName(name);
            student.setEmail(email);
            student.setAddress(address);
            StudentDAO dao = new StudentDAO();
            dao.insert(student);

            // it gets an updated list of the records from the table
            List<Student> students = dao.getList();
            // append the list to the request
            request.setAttribute("listStudents", students);
            RequestDispatcher dispatcher =
            request.getRequestDispatcher("student-added.jsp");
            // it forwards the request with the updated list
            dispatcher.forward(request,response);
      }
}
```

student-added.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
      pageEncoding="UTF-8"%>
<%@ page import="br.edu.ufabc.prograd.model.*"%>

<html>

<body>
      <table border='1'>
            <tr>
                  <td><b>Id</b></td>
                  <td><b>Name</b></td>
                  <td><b>E-mail</b></td>
                  <td><b>Address</b></td>
            </tr>
            <c:forEach var="student" items="${requestScope.listStudents}">
                  <tr align='center'>
                        <td>${student.id}</td>
                        <td>${student.name}</td>
                        <td>${student.email}</td>
                        <td>${student.address}</td>
                  </tr>
            </c:forEach>
      </table>
</body>
</html>
```