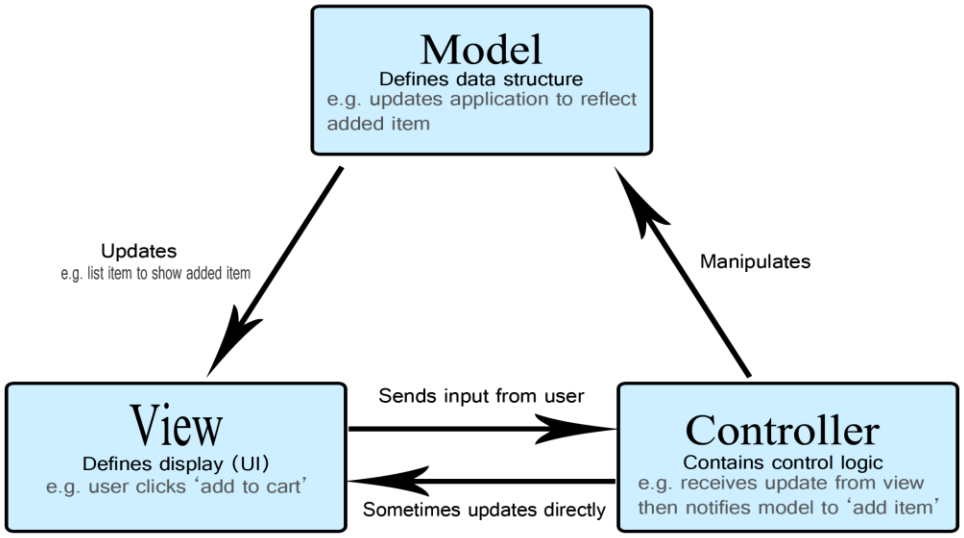
Spring MVC

1. What is Spring MVC?

- Spring MVC Framework is an open source Java platform that provides comprehensive infrastructure support for developing robust Java based Web applications very easily and very rapidly.

1.1 MVC (Model-View-Controller)

- The Model-View-Controller is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application.



- The **Model** component corresponds to all the data-related logic that the user works with.
- The **View** component is used for all the UI logic of the application.
- **Controllers** act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output.

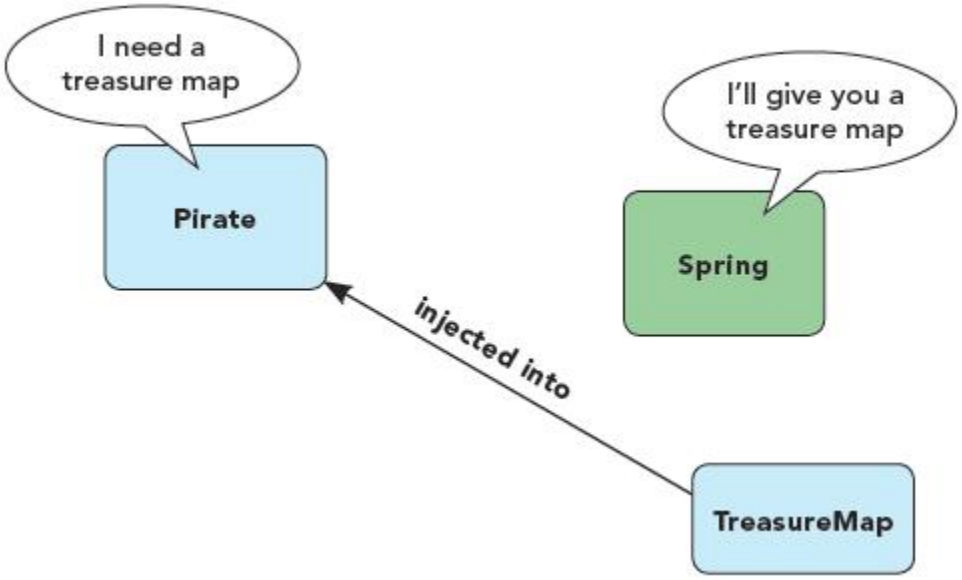
2. Spring IoC container

- The Spring container is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction.
- The Spring container uses Dependency Injection to manage the components that make up an application.

- Here, the focus is on Loose Coupling.
- This process is fundamentally the inverse (hence the name, Inversion of Control) of the bean itself controlling the instantiation or location of its dependencies on its own by using direct construction of classes or the Service Locator pattern.

2.1. Dependency Injection

- In object-oriented programming (OOP) software design, dependency injection (DI) is the process of supplying a resource that a given piece of code requires.
- The required resource, which is often a component of the application itself, is called a dependency.
- Dependency injection can be useful when working with large applications because it relieves various code modules from the task of instantiating references to resources and allows dependencies to be swapped out easily, which can make unit testing easier.
- It is a process whereby objects define their dependencies (that is, the other objects they work with) only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method. The container then injects those dependencies when it creates the bean.
- Eg: Here, dependency is ‘Treasure Map’.



3. Hello World Example

- Step 1 : Go to https://start.spring.io/
- Step 2 : Fill the entries as follows :
    - Generate a : *Gradle Project*
    - with : *Java*
    - and Spring Boot : *2.0.6*
    - Group: *com.springdemo*
    - Artifact : *SpringDemo*
    - Search for dependencies :
        - Web
        - Spring Integration
- Step 3 : Click on Generate Project.
- Step 4 : Download the project, extract it and import it in IntelliJ.
- Step 5 : Navigate to -
  SpringDemo -> src -> main -> java -> com.springDemo.SpringDemo
- Step 6 : Create a java file named HelloController.java with the following code.

```
package com.springDemo.SpringDemo;
import org.springframework.stereotype.Controller;

@Controller
public class HelloController {
   @GetMapping("/")
   @ResponseBody
   public String index() {
       return "Hello World";
   }
}
```

- Step 6 : Run the application : SpringDemoApplication
- Step 7 : Go to your browser and in address-bar, type - localhost:8080
- Step 8 : Hit ‘Enter’ and you should be seeing a webpage with text “Hello World”.

4. Simple Form Example

- Step 1 : Go to https://start.spring.io/
- Step 2 : Fill the entries as follows :
    - Generate a : *Gradle Project*
    - with : *Java*
    - and Spring Boot : *2.0.6*
    - Group: *com.springdemo*
    - Artifact : *SpringDemo*
    - Search for dependencies :
        - Web
        - Spring Integration
- Step 3 : Click on Generate Project.
- Step 4 : Download the project, extract it and import it in IntelliJ.
- Step 5 : Navigate to -

  SpringDemo -> src -> main -> java -> com.springDemo.SpringDemo
- Step 6 : Create a java file named FormController.java with the following code.

```
package com.springDemo.SpringDemo;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.ModelAndView;

@SuppressWarnings("unused")
@Controller
public class FormController {
   @RequestMapping(value = "/student", method = RequestMethod.GET)
   public ModelAndView student() {
        return new ModelAndView("student", "command", new Student());
   }

   @RequestMapping(value = "/result", method = RequestMethod.POST)
   public String addStudent(Student student, ModelMap model) {
        model.addAttribute("name", student.getName());
        model.addAttribute("age", student.getAge());
        model.addAttribute("id", student.getId());
        return "result";
   }
}
```

- Step 6 : Create a folder webapp in *src\main*.
- Step 7 : Create a folder WEB-INF in *src\main\webapp*.
- Step 8 : Create a folder jsp in *src\main\webapp\WEB-INF*.
- Step 9 : Create file student.jsp in previously created jsp folder with the following code.

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<html>
  <head>
     <title>Spring MVC Form Handling</title>
  </head>
  <body>
     <h2>Student Information</h2>
     <form:form method = "POST" action = "/result">
        <table>
           <tr>
              <td><form:label path = "name">Name</form:label></td>
              <td><form:input path = "name" /></td>
           </tr>
           <tr>
              <td><form:label path = "age">Age</form:label></td>
              <td><form:input path = "age" /></td>
           </tr>
           <tr>
              <td><form:label path = "id">id</form:label></td>
              <td><form:input path = "id" /></td>
           </tr>
           <tr>
              <td colspan = "2">
                 <input type = "submit" value = "Submit"/>
              </td>
           </tr>
        </table>
     </form:form>
  </body>
</html>
```

- Step 10 : Create file result.jsp in previously created jsp folder with the following code.

```
<html>
  <head> <title>Spring MVC Form Handling</title> </head>
  <body> <h2>Submitted Student Information</h2>
     <table>
        <tr> <td>Name</td> <td>${name}</td> </tr>
        <tr> <td>Age</td> <td>${age}</td> </tr>
        <tr> <td>ID</td> <td>${id}</td> </tr>
     </table>
  </body>
</html>
```

- Step 11 : In your *application.properties* file, add

```
spring.mvc.view.prefix:/WEB-INF/jsp/
spring.mvc.view.suffix:.jsp
```

- Step 12 : In your *build.gradle* file, add

```
apply plugin: 'war'
```

- Step 13 : Run the application : SpringDemoApplication
- Step 14 : Go to your browser and in address-bar, type - localhost:8080/student
- Step 15 : Hit ‘Enter’ and you should be seeing a webpage with a simple form.
- Step 16 : Fill the entries and hit enter. You should be seeing a webpage with a submitted data.

5. Form Tag Library

- All the jsp pages using a form need to add a tag library of prefix form as the first line of code.

```
<%@taglib uri = "http://www.springframework.org/tags/form" prefix = "form"%>
```

5.1. Textbox

```
<form:input path = "name" />
```

5.2. Password

```
<form:password path = "password" />
```

5.3. Textarea

```
<form:textarea path = "address" />
```

5.4. Checkbox

```
<form:checkbox path = "receivePaper" />
```

5.5. Checkboxes

student.jsp

```
<form:checkboxes items= "${webFrameworkList}" path = "favoriteFrameworks"/>
```

StudentFormController.java

```
@Controller
public class StudentFormController {
  .
  .
  @RequestMapping(value = "/result", method = RequestMethod.POST)
   public String addStudent(Student student, ModelMap model) {
   .
   .
   model.addAttribute("favoriteFrameworks",student.getFavoriteFrameworks());
```

```
        return "result";
    }
    .
    .
    @ModelAttribute("webFrameworkList")
    public List<String> getWebFrameworkList() {
        List<String> webFrameworkList = new ArrayList<>();
        webFrameworkList.add("Spring MVC");
        webFrameworkList.add("Struts 1");
        webFrameworkList.add("Struts 2");
        webFrameworkList.add("Apache Wicket");
        return webFrameworkList;
    }
    .
    .
}
```

5.6. Radiobutton

student.jsp

```
<form:radiobutton path = "gender" value = "M" label = "Male" />
<form:radiobutton path = "gender" value = "F" label = "Female" />
```

result.jsp

```
<td>Gender</td>
<td>${(gender=="M"? "Male" : "Female")}</td>
```

5.7. Dropdown

student.jsp

```
<form:select path = "country">
  <form:option value = "NONE" label = "Select"/>
  <form:options items = "${countryList}" />
</form:select>
```

StudentFormController.java

```
@Controller
public class StudentFormController {
  .
  .
  @RequestMapping(value = "/result", method = RequestMethod.POST)
   public String addStudent(Student student, ModelMap model) {
   .
   .
   model.addAttribute("country", student.getCountry());
       return "result";
   }
   .
   .
   @ModelAttribute("countryList")
   public Map<String, String> getCountryList() {
     Map<String, String> countryList = new HashMap<>();
     countryList.put("US", "United States");
     countryList.put("CH", "India");
     countryList.put("SG", "Singapore");
     countryList.put("MY", "Malaysia");
     return countryList;
   }
   .
   .
}
```

5.8. Listbox

student.jsp

```
<form:select path = "country">
  <form:option value = "NONE" label = "Select"/>
  <form:options items = "${countryList}" />
</form:select>
```

result.jsp

```
<form:select path = "country">
  <form:option value = "NONE" label = "Select"/>
  <form:options items = "${countryList}" />
</form:select>
```

StudentFormController.java

```
@Controller
public class StudentFormController {
  .
  .
  @RequestMapping(value = "/result", method = RequestMethod.POST)
   public String addStudent(Student student, ModelMap model) {
   .
   .
   model.addAttribute("skills", student.getSkills());
       return "result";
   }
   .
   .
   @ModelAttribute("skillsList")
   public Map<String, String> getSkillsList() {
     Map<String, String> skillList = new HashMap<>();
     skillList.put("Hibernate", "Hibernate");
     skillList.put("Spring", "Spring");
     skillList.put("Apache Wicket", "Apache Wicket");
     skillList.put("Struts", "Struts");
     return skillList;
   }
   .
   .
}
```