Laboratory Manual

for

Applied Design Patterns and Application Frameworks (IT 618)

B.Tech (IT) SEM - VI

June 2016

Faculty of Technology
Department of Information Technology
Dharmsinh Desai University Nadiad.
www.ddu.ac.in

Dharmsinh Desai University, Nadiad
Faculty of Technology
Department of Information Technology
Laboratory Manual
B.Tech. – IT, Sem: 6, Subject Name: Applied Design Patterns and Application Frameworks

List of Experiments:

1. Implement programming solution that uses Abstract class, interface, Object composition, and inheritance
2. Implement programming solutions that use following design pattern
    a. Singleton
    b. AbstractFactory
    c. Prototype
3. Implement programming solutions that use following design pattern
    a. Composite
    b. Decorator
    c. Facade
4. Solving programming problems using Java-8 functional programming.
5. Designing and implementing web interface using JavaScript, and CSS
6. Designing and implementing web interface using Bootstrap and customizing CSS
7. Designing and implementing a single page web interface using Angular JS
8. Designing and implementing interactive web interface using AJAX
9. Implement data access using Hibernate
10. Implement relational data access using Hibernate
11. Implement form handling with data binding using SpringMVC
12. Implement form handling with form validation using SpringMVC

LABWORK BEYOND CURRICULA

1. Implement programming problem using ServiceLocator design pattern
2. Study of JavaServer Faces

Dharmsinh Desai University, Nadiad
Faculty of Technology
Department of Information Technology
Laboratory Manual
B.Tech. – IT, Sem: 6, Subject Name: Applied Design Patterns and Application Frameworks

COMMON PROCEDURE
The common procedure for solving programming related problems is as follows:

Step 1: For a given problem statement, prepare a design of the solution.
Step 2: Study and understand the usage of required API classes, functions, or tags.
Step 3: Prepare a solution project (Java, Web, or HTML/JavaScript)
Step 4: Prepare database and necessary DB tables, if required
Step 5: Write configuration files (e.g., for Hibernate and Spring project)
Step 6: Implement the solution using appropriate programming/scripting languages.
Step 7: Test the solution with valid/invalid input.

Experiment 1:

Aim: Implement programming solution that uses Abstract class, interface, Object composition, and inheritance

Tools / Apparatus: Netbeans 8.0.2 or higher

Procedure:
Step 1: Think of a scenario in which abstract class, interface, Object composition, and inheritance can be used.
Step 2: Prepare a design of the solution.
Step 3: Prepare a Java project (Java console application)
Step 4: Write the required Java classes and interfaces.
Step 5: Test the solution with valid/invalid input.

Experiment 2:

Aim: Implement programming solutions that use following design pattern

a. Singleton
b. AbstractFactory
c. Prototype

Tools / Apparatus: Netbeans 8.0.2 or higher

Procedure:
Step 1: For each design pattern, think of a scenario in which it can be used.
Step 2: Prepare a design of the solution.
Step 3: Prepare a Java project (Java console application)
Step 4: Write the required Java classes and interfaces.
Step 5: Test the solution with valid/invalid input.

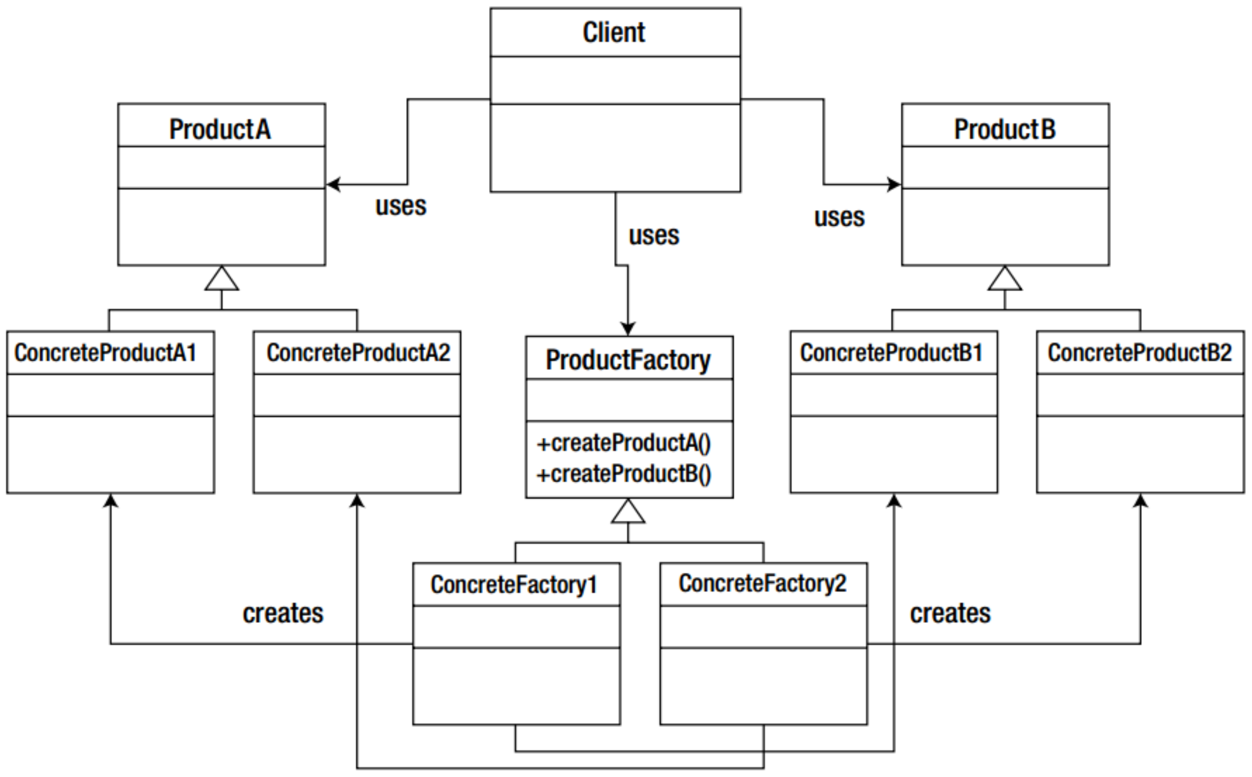
Hints:

a. Singleton

It is a creational design pattern. There are many situations in which you want to make sure that only one instance is present for a particular class. The pattern implementation provides a single point of access to the class.

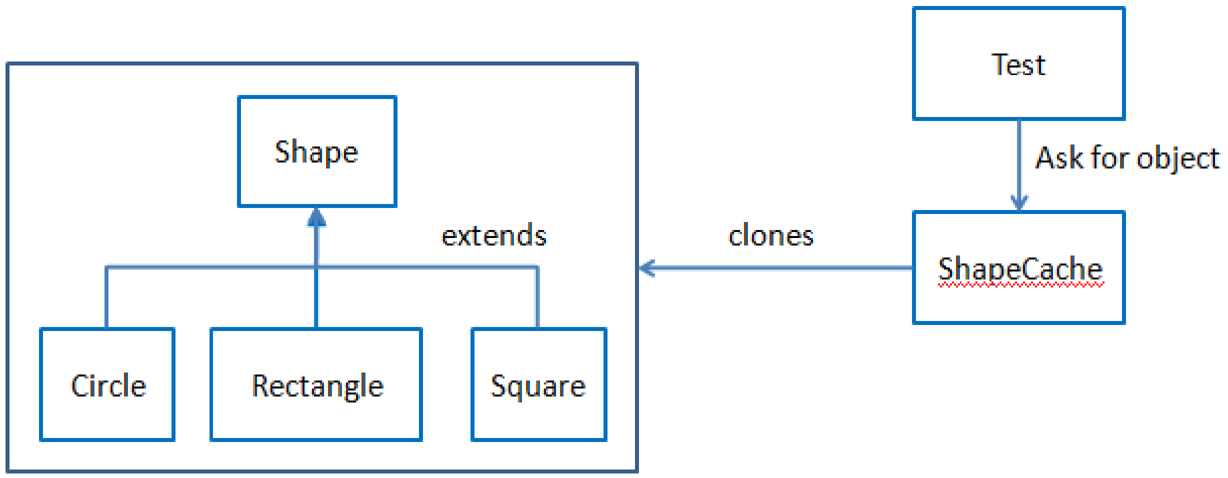
b. AbstractFactory

It is a creational design pattern. The abstract factory is basically a factory of factories. The abstract factory design pattern introduces one more indirection to create a specified object. A client of the abstract factory design pattern first requests a proper factory from the abstract factory object, and then it requests an appropriate object from the factory object.
A typical class diagram for an abstract factory design pattern is as follows:



c. Prototype

It is a creational design pattern. It can provide the best way to create duplicate object while keeping performance in mind. This pattern is useful in scenarios where creating objects directly is costly. E.g., an object is created after performing a costly database operation. Objects can be cached, and cached object can be returned when needed.



To implement Prototype design pattern, support is available in Java. Java supports Cloneable interface. We need to override its clone() method.

Experiment 3:

Aim: Implement programming solutions that use following design pattern

a. Composite
b. Decorator
c. Facade

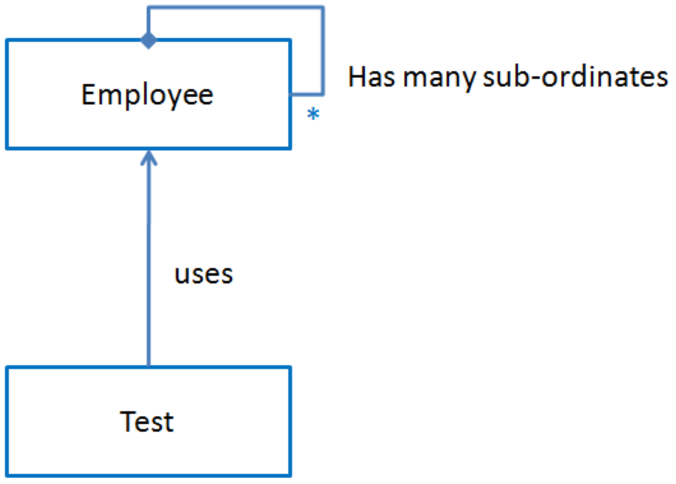
Tools / Apparatus: Netbeans 8.0.2 or higher

Procedure:
Step 1: For each design pattern, think of a scenario in which it can be used.
Step 2: Prepare a design of the solution.
Step 3: Prepare a Java project (Java console application)
Step 4: Write the required Java classes and interfaces.
Step 5: Test the solution with valid/invalid input.

Hints:

a. Composite

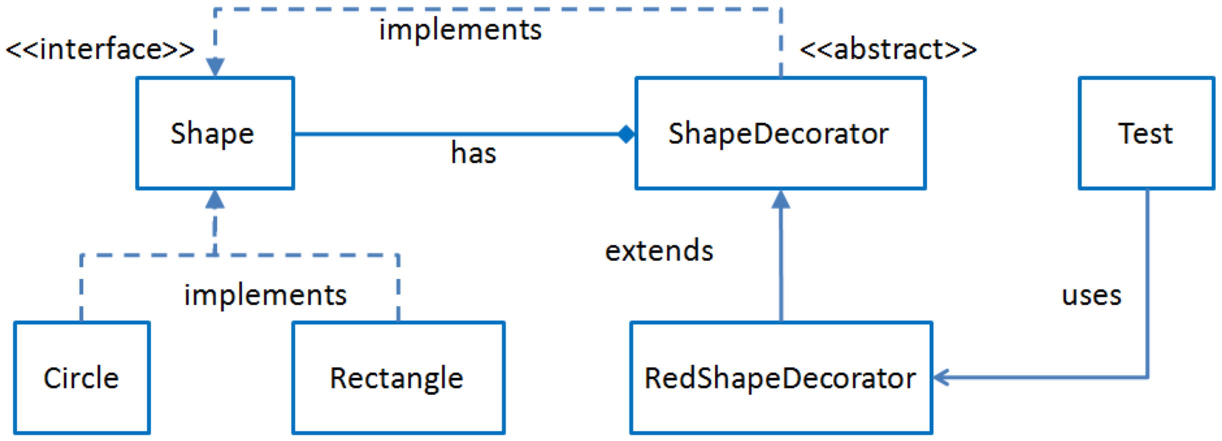
It is of type structural design pattern. The composite design pattern is used where we need to treat a group of objects in a similar way as a single object. Composite pattern composes objects in term of a tree structure to represent part as well as whole hierarchy. This pattern creates a class that contains a group of its own objects. This class provides ways to modify its group of same objects.
Example:



b. Decorator

It is of type structural design pattern. Decorator pattern allows a user to add new functionality to an existing object without altering its structure. This pattern acts as a wrapper to existing class. This pattern creates a decorator class which wraps the original class and provides additional functionality keeping class methods signature intact.

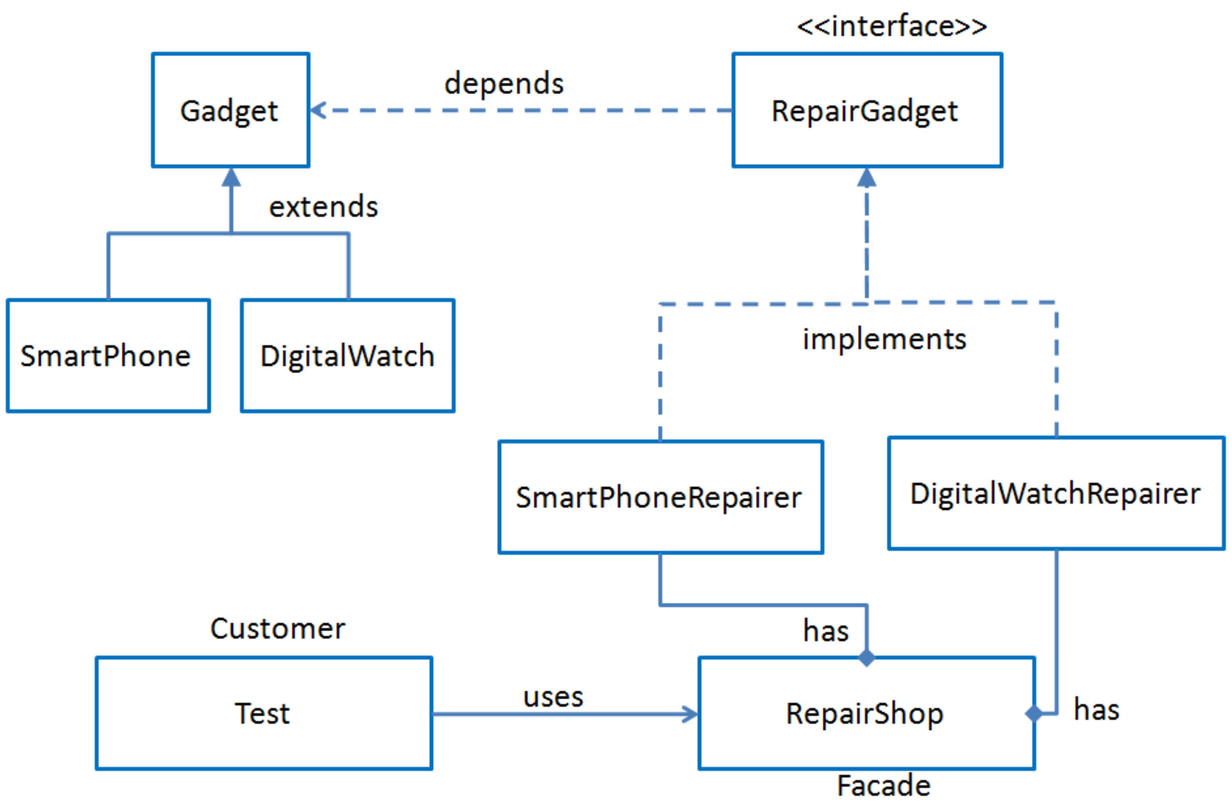
Example:



c. Façade

It is of type structural design pattern. Facade pattern hides the complexities of the system and provides an interface to the user using which the user can access the system. This design pattern involves a single class which provides simplified methods required by user and delegates calls to methods of existing system classes.

Example:



Experiment 4:

Aim: Solve programming problems using Java-8 functional programming.

Tools / Apparatus: Netbeans 8.0.2 or higher

Procedure:
Step 1: Think of a scenario in which functional programming (lambda expression and default methods) can be used.
Step 2: Prepare a design of the solution.
Step 3: Prepare a Java project (Java console application)
Step 4: Write the required Java classes and interfaces.
Step 5: Test the solution with valid/invalid input.

Hints:
Lambda expression:
Lambda expression is an anonymous function. An anonymous function means a method. Without a declaration, i.e., without Name declaration, Access modifier, and Return value declaration. An anonymous function saves our effort of declaring and writing a separate method to the containing class.
Example, we want to write a method that takes no arguments and prints “Hello World” message. The lambda expression for the above is as follows:

```
        () -> System.out.println(“Hello World”);
```

Default methods:

In Java 8, it is now possible to add method bodies into interfaces
Example:

```
      public interface MathOperation{
            int add (int a,int b);
            default int multiply(int a, int b){
                  return a*b;
            }
      }
```

FunctionalInterface:
FunctionalInterface must have only one method.
Example:

```
@FunctionalInterface
public interface MathOperation{
    int operation(int a, int b);
}
```

Experiment 5:

Aim: Designing and implementing web interface using JavaScript, and CSS

Tools / Apparatus: Netbeans 8.0.2 or higher

Procedure:
Step 1: Think of a scenario in which JavaScript, and CSS can be used.
Step 2: Prepare a design of the solution.
Step 3: Prepare a Java web project
Step 4: Write required HTML, JavaScript, and css files.
Step 5: Test the solution with valid/invalid input.

Hints:
JavaScript code is included within <script> tags:

```
      <script type="text/javascript">
              document.write("<h1>Hello World!</h1>") ;
      </script>
```

We can write a JavaScript function and we can call it on some event. E.g., on bringing focus on a control.

CSS stands for Cascading Style Sheet. We can add a stylesheet to our web page using the following:

```
      <head>
              <title>Demo for website design</title>
                  <link rel="stylesheet" type="text/css" href="style.css" />
      </head>
```

Syntax for style selector is as follows: selector {declarations}
Examples:
1. Style for existing tag

```
body {
      background-color:#EEEEEE;
      font-family: Helvetica, Arial, sans-sarif;
}
```
2. ID Selector
```
#id-name{
}
```
3. Class selector
```
.class-name{
}
```

Experiment 6:

Aim: Designing and implementing web interface using Bootstrap and customizing CSS

Tools / Apparatus: Netbeans 8.0.2 or higher

Procedure:
Step 1: Think of appropriate layout/design of a web-page.
Step 2: Explore which bootstrap classes are suitable for layout and components.
Step 3: Prepare a Java web project
Step 4: Include bootstrap.js and jqueery.js in the HTML page. Use suitable bootstrap classes or customize, if required.
Step 5: Test the solution with valid/invalid input.

Hints:
Bootstrap can be added in our web page using the following way
```
    <body>
        <h2>How to add Bootstrap.</h2>
            <script src="js/bootstrap.min.js"></script>
            <script src="js/jquery-2.2.2.min.js"></script>
    </body>
```

jQuery code can be included using the following way
```
<script>
$(document).ready(
  function()
  {
      // Write jQuery code here...
  }
);
</script>
```

Experiment 7:

Aim: Designing and implementing a single page web interface using Angular JS

Tools / Apparatus: Netbeans 8.0.2 or higher

Procedure:

Step 1: Think of appropriate scenario where Angular JS (a framework for SPA-Single Page Application) can be used.
Step 2: Decide about controller(s).
Step 3: Prepare a Java web project
Step 4: Include angularjs framework in our web page.
Step 5: Test the solution with valid/invalid input.

Hints:
AngularJS extends HTML with new tag attributes

- ng-app directive defines an AngularJS application
- ng-model directive binds the value of HTML control (input, select, textarea) to application data.
- ng-bind directive binds application data to the HTML view.

Example 1: Display text as you type in a text box
```
<!DOCTYPE html>
<html data-ng-app="">
    <head>
        <title>First AngularJS Page</title>
            <script src="angular.min.js"></script>
    </head>
    <body>
            Name:
            <input type="text" data-ng-model="name" /> {{ name }}
      </body>
</html>
```

To create a module, we use the following:
```
      var demoApp = angular.module('demoApp', []);
```
Where demoApp is the name of the module

AngularJS controller:
A controller contains business logic for a single view. It is used to add data and methods to $scope. Syntax for creating a controller
```
      app.controller('name', function (services) { ... });
```
The services is a list of service names this controller uses as separate parameters, not an array services are provided via dependency injection.

To use a controller:
add ng-controller attribute to an HTML element

Creating a controller in a module, use the following syntax:
```
var demoApp = angular.module('demoApp', []);
demoApp.controller('SimpleController', function($scope){
       $scope.customers = [
            {name: 'Abc', city:'Vadodara' },
            {name:'Abc', city:'Nadiad' },
             {name:'Pqr', city:'Ahmedabad' },
             {name:'Xyz', city:'Surat' }
```

```
	];
	});
```

Example 2: Creating and using a controller:

```
<!DOCTYPE html>
<html ng-app="myApp">
    <head>
        <title>Create a basic controller and a view</title>
    </head>
    <body>
      <div ng-controller="SimpleController">
            <h3>Using Basic Controller</h3>
            <input type="text" ng-model="name"/>
            <ul>
            <li   ng-repeat="customer   in   customers   |   filter:name   |
orderBy:'city'"> {{customer.name}} -- {{customer.city}} </li>
            </ul>
      </div>
      <script src="angular.min.js"></script>
      <script>
            var myApp=angular.module("myApp", []);
            function SimpleController($scope){
                  $scope.customers = [
                         {name: 'Abc', city:'Vadodara' },
                         {name:'Abc', city:'Nadiad' },
                         {name:'Pqr', city:'Ahmedabad' },
                         {name:'Xyz', city:'Surat' }
                  ];
            }
            myApp.controller("SimpleController", SimpleController);
      </script>
      </body>
</html>
```

Experiment 8:

Aim: Designing and implementing interactive web interface using AJAX

Tools / Apparatus: Netbeans 8.0.2 or higher

Procedure:
Step 1: Think of appropriate scenario where AJAX call can be used.
Step 2: Design server side logic and client side logic.
Step 3: Prepare a Java web project
Step 4: Write a server logic in a servlet.
Step 5: Create a web page and call the servlet using AJAX call
Step 6: Test the solution with valid/invalid input.

Hints:
To make AJAX call, we require XMLHttpRequest, a JavaScript object.

We initialize AJAX call using
```
      req.open("GET", url, true);
```
Where url is the url of a servlet and req is the XMLHttpRequest object.

We set a callback function using
```
      req.onreadystatechange = callback;
```
Where callback is the callback function, which is to be executed when the servlet returns the response. Generally, a callback function parses the received response, in XML, and manipulates div to render the received response.

We initiate HTTP interaction using
```
req.send();
```

Experiment 9:

Aim: Implement data access using Hibernate

Tools / Apparatus: Netbeans 8.0.2 or higher

Procedure:
Step 1: Think of a problem for which data can be stored and retrieved from a database.
Step 2: Identify attributes (columns) of database table(s).
Step 3: Prepare a solution project (Java console application)
Step 4: Include Hibernate jar file in our project
5tep 5: Prepare database and necessary DB tables
Step 6: Write configuration files for Hibernate (database connection and hibernate mapping)
Step 7: Implement Hibernate entities and test class.
Step 8: Test the solution with valid/invalid input.
Step 9: Observe database records.

Hints:
Hibernate.cfg.xml (put it under src directory)
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE      hibernate-configuration      PUBLIC      "-//Hibernate/Hibernate
Configuration   DTD   3.0//EN"   "http://hibernate.sourceforge.net/hibernate-
configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property      name="hibernate.dialect">org.hibernate.dialect.MySQLDialect
</property>
    <property   name="hibernate.connection.driver_class">com.mysql.jdbc.Driver
</property>
    <property  name="hibernate.connection.url">jdbc:mysql://localhost:3306/ddu
</property>
    <property name="hibernate.connection.username">root</property>
    <!-- List of XML mapping files -->
    <mapping resource="Student.hbm.xml"/>
  </session-factory>
```

```
</hibernate-configuration>
```

Student.hbm.xml file (can be created using wizard)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE  hibernate-mapping  PUBLIC  "-//Hibernate/Hibernate  Mapping  DTD
3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="hibernate.model.Student" table="student">
        <id name="id" type="int" column="id">
            <generator class="native"/>
        </id>
        <property name="firstName" column="first_name" type="string"/>
        <property name="lastName" column="last_name" type="string"/>
        <property name="semester" column="semester" type="int"/>
    </class>
</hibernate-mapping>
```

Write a test class

```
        Student st=new Student("Name", "Surname", 6);
        //Build session factory
        SessionFactory                   sessionFactory=                  new
Configuration().configure().buildSessionFactory();
        //Create a session
        Session session=sessionFactory.openSession();
      //Create a transaction to start interaction
        session.beginTransaction();
        session.save(st);
        session.getTransaction().commit();
        session.close();
        sessionFactory.close();
```

Experiment 10:

Aim: Implement relational data access using Hibernate

Tools / Apparatus: Netbeans 8.0.2 or higher

Procedure:
Step 1: Think of a problem for which data (tables) can be related (i.e., one-to-one, one-to-many, many-to-one, or many-to-many relation).
Step 2: Identify attributes (columns) of database table(s) and joining columns.
Step 3: Prepare a solution project (Java console application)
Step 4: Include Hibernate jar file in our project
5tep 5: Prepare database and necessary DB tables (we can use hbm2ddl.auto=create)
Step 6: Write configuration file for Hibernate (database connection)
Step 7: Implement annotation based Hibernate entities and test class.
Step 8: Test the solution with valid/invalid input.
Step 9: Observe database records.

Hints:
Mandatory annotations for entities:

@Entity

- To indicate that class is a Hibernate entity
- Class must have zero argument constructor

@Id

- To indicate that data field is the primary key

Other annotations

@Table

- When we want to use other name of the table

@GeneratedValue

- two parameters: strategy and generator are used to generate Id value automatically

To establish relation we have following annotations:

@OneToOne(cascade = CascadeType.ALL)
@ManyToOne(cascade=CascadeType.ALL)
@OneToMany (cascade=CascadeType.ALL)
@ManyToMany (cascade=CascadeType.ALL)
In bi-directional mapping, we use mappedBy property.
@JoinColumn is used to indicate that this entity is the owner of the relationship

In hibernate mapping file, we use the following:
<mapping class="FQ class name"/>

In view, we consume data using the following expression: ${data-name}

Experiment 11:

Aim: Implement form handling with data binding using SpringMVC

Tools / Apparatus: Netbeans 8.0.2 or higher

Procedure:
Step 1: Think of an HTML form with different types of data as input.
Step 2: Study working of SpringMVC framework, its components, and configuration file.
Step 3: Prepare a solution project (Java Web application)
Step 4: Include Spring jar file our project
5tep 5: Configure web.xml file
Step 6: Write and configure dispatcher servlet configuration file
Step 7: Implement annotation based SpringMVC controller with data binding and write view file.
Step 8: Test the solution with valid/invalid input.

Hints:
Web.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
```

```
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
       version="3.1">
    <servlet>
        <servlet-name>spring-dispatcher</servlet-name>
        <servlet-class>      org.springframework.web.servlet.DispatcherServlet
</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring-dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

@Controller annotation is used to make SpringMVC controller. We use to following to indicate that controllers are available and should be scanned automatically:
<context:component-scan base-package= "springmvcapp2.hellocontroller"/>

Spring-dispatcher-servlet.xml file under /WEB-INF

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        ">
    <context:component-scan base-package="springmvcapp2.hellocontroller" />

    <bean                                                     id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix">
            <value>/WEB-INF/</value>
        </property>
        <property name="suffix">
            <value>.jsp</value>
        </property>
    </bean>
</beans>
```

Example: HelloPage.jsp file under /WEB-INF

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Annotation Based Spring Controller</title>
    </head>
    <body>
        <h1>Annotation Based Spring Controller</h1>
        <h2>${msg}</h2>
    </body>
```

```
</html>
```

Where ${data-name} is used to access data.

Data binding can be performed in request handling method (submitAdmissionForm() in the following example) using the following way:

```
@RequestMapping(value="/submitAdmissionForm.html", method=RequestMethod.POST)
    public ModelAndView submitAdmissionForm(
      @RequestParam("studentName")                String                 name,
@RequestParam("studentContact")  String contact) {
        ModelAndView modelView=new ModelAndView("AdmissionSuccess");
        modelView.addObject("msg",  "We  have  registered  your  details  as,
Name:"+name+" Contact No.:"+contact);
        return modelView;
    }
}
```

We can also use @RequestParam Map<String,String> requestParams instead of individual parameters:

```
@RequestMapping(value="/submitAdmissionForm.html", method=RequestMethod.POST)
    public  ModelAndView  submitAdmissionForm(@RequestParam  Map<String,String>
requestParams) {
        String name, contact;
        name=requestParams.get("studentName");
        contact=requestParams.get("studentContact");
        ModelAndView modelView=new ModelAndView("AdmissionSuccess");
        modelView.addObject("msg",  "We  have  registered  your  details  as,
Name:"+name+" Contact No.:"+contact);
        return modelView;
    }
```

Experiment 12:

Aim: Implement form handling with form validation using SpringMVC

Tools / Apparatus: Netbeans 8.0.2 or higher

Procedure:
Step 1: Think of an HTML form with different types of data as input.
Step 2: Explore annotations available in SpringMVC for data validation.
Step 3: Prepare a solution project (Java Web application)
Step 4: Include Spring jar file and hibernate validation jar (hibernate-validator) file in our project
5tep 5: Configure web.xml file
Step 6: Write and configure dispatcher servlet configuration file
Step 7: Implement annotation based SpringMVC controller with validation controllers and view file.
Step 8: Test the solution with valid/invalid input.

Hints:

To allow validation framework to work, we need to use @Valid annotation in front of @ModelAttribute annotation (applied on parameter of request handling method)

Annotation for size of a string data
@Size(min=3, max=40)
We can add error message using message="…" parameter to @Size

For regular expression pattern, we can use @Pattern annotation
Example: @Pattern(regexp = "[^0-9]*")

@Past is used to indicate that a date has to be past date, not future date

@Max annotation is used to indicate that the maximum value should be less than the specified value
Example: @Max(999) indicates that any value greater than 999 should generate a violation error.

@Min is used to check whether the field value is higher than or equal to the specified minimum

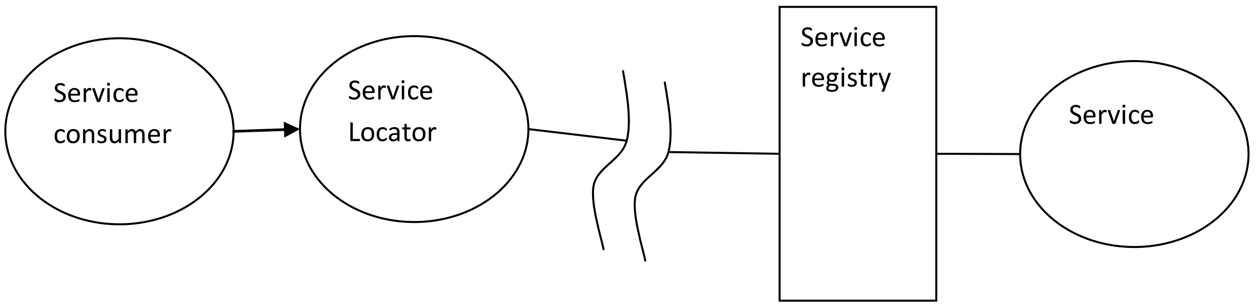
LABWORK BEYOND CURRICULA
Experiment 1:

Aim: Implement programming problem using ServiceLocator design pattern

Tools / Apparatus: Netbeans 8.0.2 or higher

Procedure:
Step 1: Think of any existing problem solution for which service locator design pattern can be applied.
Step 2: Prepare a design of the solution.
Step 3: Prepare a Java project (Java console application)
Step 4: Write the required Java classes and interfaces.
Step 5: Test the solution with valid/invalid input.

Hints:
Service locator pattern is used when we use any remote service or object. Service locator allows transparent access of a service even if the service is moved to another location.



Experiment 2:

Aim: Study of JavaServer Faces

Tools / Apparatus: Netbeans 8.0.2 or higher

Procedure:
Step 1: Study and understand JavaServer Faces framework.
Step 2: Understand configuration files.
Step 3: Understand Server side JSF components
Step 4: Compare working of JSF with that of SpringMVC.