LAB 1.1

Write a simple spring program to print 'Hello World!!!!' in the screen. But use different types of configurations.

Steps:

- Create a new java project in Eclipse.
- Right Click the project go to Build path→ Configure path. Add the following jars in the build path
    - antlr-runtime-3.0.1
    - org.springframework.aop-3.1.0.M2
    - org.springframework.asm-3.1.0.M2
    - org.springframework.aspects-3.1.0.M2
    - org.springframework.beans-3.1.0.M2
    - org.springframework.context.support-3.1.0.M2
    - org.springframework.context-3.1.0.M2
    - org.springframework.core-3.1.0.M2
    - org.springframework.expression-3.1.0.M2
    - commons-logging-1.1.1
- Create a new package org.capgemini. Add the class HelloWorld.java.

  HelloWorld.java

```
package org.capgemini;
public class HelloWorld {

     private String message;

     public String getMessage() {
          return message;
     }

     public void setMessage(String message) {
          this.message = message;
     }
}
```

- Include the configuration file under the src folder called Beans.xml

Beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/sch
ema/beans
http://www.springframework.org/schema/beans/spring-
beans-3.0.xsd">
<bean id="helloWorld"
class="org.capgemini.HelloWorld">
<property name="message" value="Hello World!"/>
</bean>
```

- Include the class MainApp.java under org.capgemini

```
package org.capgemini;
import
org.springframework.beans.factory.InitializingBean;
import
org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
public class MainApp {
public static void main(String[] args) {
XmlBeanFactory factory = new XmlBeanFactory
(new ClassPathResource("Beans.xml"));
HelloWorld obj = (HelloWorld)
factory.getBean("helloWorld");
System.out.println("Your Message :"
+obj.getMessage());
}
}
```

- Run the MainApp.java file.

Note:

In the above configuration change the highlighted XmlBeanFactory to ApplicationContext. Then explain the differences.

```
ApplicationContext  context=new
FileSystemXmlApplicationContext
("D:\vidavid\workspace1\BeanFactory\src\Beans.xml")
```

Output

```
Your Message : Hello World!
```

LAB 1.2

Write a Spring program which demonstrates the usage of singleton and prototype bean.

Steps:

- Create a new java project in Eclipse.
- Right Click the project goto Build path→ Configure path. Add the following jars in the build path
    - antlr-runtime-3.0.1
    - org.springframework.aop-3.1.0.M2
    - org.springframework.asm-3.1.0.M2
    - org.springframework.aspects-3.1.0.M2
    - org.springframework.beans-3.1.0.M2
    - org.springframework.context.support-3.1.0.M2
    - org.springframework.context-3.1.0.M2
    - org.springframework.core-3.1.0.M2
    - org.springframework.expression-3.1.0.M2
    - commons-logging-1.1.1

- Create a new package org.capgemini. Add the class HelloWorld.java.

HelloWorld.java

```java
package org.capgemini;
public class HelloWorld {

	private String message;

	public String getMessage() {
		return message;
	}

	public void setMessage(String message) {
		this.message = message;
	}
}
```

- Include the configuration file under the src folder called Beans.xml

Beans.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:context="http://www.springframework.org/schema/contex
t"
     xsi:schemaLocation="http://www.springframework.org/schema/b
eans http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
          http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-
3.0.xsd">

<bean id="helloWorld" class="org.capgemini.HelloWorld"
scope="singleton">

</bean>
</beans>
```

- Include the class MainApp.java under org.capgemini

MainApp.java

```
package org.capgemini;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationConte
xt;

public class MainApp {
          public static void main(String[] args) {
          ApplicationContext context=new
ClassPathXmlApplicationContext("Beans.xml");
          HelloWorld
hw1=(HelloWorld)context.getBean("helloWorld");
          hw1.setMessage("I am helloWorld");

          HelloWorld
hw2=(HelloWorld)context.getBean("helloWorld");
          System.out.println("My Message1 :" +
hw1.getMessage());
          System.out.println("My Message2 :" +
hw2.getMessage());
     }

}
```

Output1

```
My Message1 :I am helloWorld
My Message2 :I am helloWorld
```

Note:

*In the configuration **"Beans.xml"** file just change the bean definition's scope as **prototype.** And execute the file we will get the following output*

```
<bean id="helloWorld" class="org.capgemini.HelloWorld"
scope="prototype">
```

Output1

```
My Message1 :I am helloWorld
My Message2 :null
```

LAB 1.3

Write a Spring program which demonstrates the bean life cycle callbacks.

Steps:

- Create a new java project in Eclipse.
- Right Click the project goto Build path→ Configure path. Add the following jars in the build path
    - antlr-runtime-3.0.1
    - org.springframework.aop-3.1.0.M2
    - org.springframework.asm-3.1.0.M2
    - org.springframework.aspects-3.1.0.M2
    - org.springframework.beans-3.1.0.M2
    - org.springframework.context.support-3.1.0.M2
    - org.springframework.context-3.1.0.M2
    - org.springframework.core-3.1.0.M2
    - org.springframework.expression-3.1.0.M2
    - commons-logging-1.1.1

- Create a new package org.capgemini. Add the class HelloWorld.java.

HelloWorld.java

```
package org.capgemini;

public class HelloWorld {

     private String message;

     public void getMessage() {
          System.out.println("My Message :" + message);
     }

     public void setMessage(String message) {
          this.message = message;
     }
     public void init(){
          System.out.println("Bean Initialization Here.");
          }
     public void destroy(){
          System.out.println("Bean will destroy now.");
          }
}
```

- Include the configuration file under the src folder called Beans.xml

Beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:context="http://www.springframework.org/schema/contex
t"
     xsi:schemaLocation="http://www.springframework.org/schema/b
eans http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
          http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-
3.0.xsd">

<bean id="helloWorld" class="org.capgemini.HelloWorld" init-
method="init" destroy-method="destroy">
<property name="message" value="I am helloWorld"/>
</bean>
</beans>
```

- Include the class MainApp.java under org.capgemini

MainApp.java

```
package org.capgemini;

import 
org.springframework.context.support.AbstractApplicationContext;
import 
org.springframework.context.support.ClassPathXmlApplicationConte
xt;

public class MainApp {
          public static void main(String[] args) {
          AbstractApplicationContext context=new 
ClassPathXmlApplicationContext("Beans.xml");
          HelloWorld 
hw=(HelloWorld)context.getBean("helloWorld");
          hw.getMessage();
          context.registerShutdownHook();

          }
}
```

- Run the MainApp.java file you will be getting the following output.

Output:

```
Bean Initialization Here.
My Message :I am helloWorld
Bean will destroy now.
```

Note:

Here we used AbstractApplicationContext to call the registerShutdownHook method.

LAB 1.3.1

Write a Spring program to invoke BeanPostProcessors methods.

Steps:

- Create a new java project in Eclipse.
- Right Click the project goto Build path→ Configure path. Add the following jars in the build path
  - antlr-runtime-3.0.1
  - org.springframework.aop-3.1.0.M2
  - org.springframework.asm-3.1.0.M2
  - org.springframework.aspects-3.1.0.M2
  - org.springframework.beans-3.1.0.M2
  - org.springframework.context.support-3.1.0.M2
  - org.springframework.context-3.1.0.M2
  - org.springframework.core-3.1.0.M2
  - org.springframework.expression-3.1.0.M2
  - commons-logging-1.1.1

- Create a new package org.capgemini. Add the class HelloWorld.java.

HelloWord.java

```
package org.capgemini;

public class HelloWorld {
private String message;
public void setMessage(String message){
this.message = message;
}
public void getMessage(){
System.out.println("Your Message : " + message);
}
public void init(){
System.out.println("Bean is going through init.");
}
public void destroy(){
System.out.println("Bean will destroy now.");
}
 }
```

InitHelloWorld.java

```
package org.capgemini;
import
org.springframework.beans.factory.config.BeanPostProcessor;
import org.springframework.beans.BeansException;

public class InitHelloWorld implements BeanPostProcessor {

public Object postProcessBeforeInitialization(Object bean,
String beanName) throws BeansException {
System.out.println("BeforeInitialization : " + beanName);
return bean; // you can return any other object as well
}

public Object postProcessAfterInitialization(Object bean,
String beanName) throws BeansException {
System.out.println("AfterInitialization : " + beanName);
return bean; // you can return any other object as well
}
}
```

MainApp.java

```
package org.capgemini;
import
org.springframework.context.support.AbstractApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationConte
xt;

public class MainApp {
public static void main(String[] args) {
AbstractApplicationContext context =
new ClassPathXmlApplicationContext("Beans.xml");
HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
obj.getMessage();
context.registerShutdownHook();
}
}
```

Beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
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-
3.0.xsd">
<bean id="helloWorld" class="org.capgemini.HelloWorld"
init-method="init" destroy-method="destroy">
<property name="message" value="Hello World!"/>
</bean>
<bean class="org.capgemini.InitHelloWorld" />
</beans>
```

Output:

```
BeforeInitialization : helloWorld
Bean is going through init.
AfterInitialization : helloWorld
Your Message : Hello World!
Bean will destroy now.
```

LAB 1.4

Write a Spring program which demonstrates the constructor and setter based dependency injection.

Steps:

- Create a new java project in Eclipse.
- Right Click the project goto Build path→ Configure path. Add the following jars in the build path
  - antlr-runtime-3.0.1
  - org.springframework.aop-3.1.0.M2
  - org.springframework.asm-3.1.0.M2
  - org.springframework.aspects-3.1.0.M2
  - org.springframework.beans-3.1.0.M2
  - org.springframework.context.support-3.1.0.M2
  - org.springframework.context-3.1.0.M2
  - org.springframework.core-3.1.0.M2
  - org.springframework.expression-3.1.0.M2
  - commons-logging-1.1.1
- Create a new package org.capgemini and add one new class called TextEditor.java.

TextEditor.java

```
package org.capgemini;

public class TextEditor {
 private SpellChecker spellChecker;
 public TextEditor(SpellChecker spellChecker)
 {
      System.out.println("Text Editor Constructor");
      this.spellChecker=spellChecker;
 }


 public void spellCheck()
 {
      spellChecker.checkSpelling();
 }
}
```

- Add new class SpellChecker.java under the org.capgemini package

SpellChecker.java

```
package org.capgemini;

public class SpellChecker {
	public SpellChecker()
	{
		System.out.println("Inside SpellCheker 
Constructor....");
	}
	public void checkSpelling()
	{
		System.out.println("Inside SpellChecking");
	}
}
```

- Include Beans.xml file under scr folder. Then do the constructor based DI in the configuration.
  Beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/b
eans http://www.springframework.org/schema/beans/spring-
beans.xsd">

<bean id="textEditor" class="org.capgemini.TextEditor">
<constructor-arg ref="spellChecker" />
</bean>

<bean id="spellChecker" class="org.capgemini.SpellChecker">
</bean>

</beans>
```

Output:

```
Inside SpellCheker Constructor....
Text Editor Constructor
Inside SpellChecking
```

Note:

- If we want to do the setter based DI. First we should add the getters and setters in the TextEditor.java file as follows:

TextEditor.java

```
package org.capgemini;

     public class TextEditor {
      private SpellChecker spellChecker;

       public SpellChecker getSpellChecker() {
          return spellChecker;
        }

     public void setSpellChecker(SpellChecker spellChecker) {
          System.out.println("Inside setSpellChecker." );
          this.spellChecker = spellChecker;
     }


     public void spellCheck()
      {
           spellChecker.checkSpelling();
      }
     }
```

- Then change the Beans.xml file as follows. The highlighted area shows the changes where we made.
  Beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.springframework.org/schema/b
eans http://www.springframework.org/schema/beans/spring-
beans.xsd">

<bean id="textEditor" class="org.capgemini.TextEditor">
<property name="spellChecker" ref="spellChecker" />
</bean>

<bean id="spellChecker" class="org.capgemini.SpellChecker">
</bean></beans>
```

- Run the MainApp.java , you can feel the setter property DI.

```
package org.capgemini;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

		public static void main(String[] args) {
		ApplicationContext context=new ClassPathXmlApplicationContext("Beans.xml");
		TextEditor tx=(TextEditor)context.getBean("textEditor");
		tx.spellCheck();
	}

}
```

Output:

```
Inside SpellCheker Constructor....
Inside setSpellChecker.
Inside SpellChecking
```

Note:

If you have many setter methods then it is convenient to use p-namespace in the XML configuration file. Let us check the difference: Let us take the example of a standard XML configuration file with <property> tags:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/bean
s http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">
<bean id="john-classic" class="com.example.Person">
<property name="name" value="John Doe"/>
<property name="spouse" ref="jane"/>
</bean>
<bean name="jane" class="com.example.Person">
<property name="name" value="John Doe"/>
</bean>
</beans>
```

```
   <!—Using P-nameSpace-->

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/bean
s
http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">
<bean id="john-classic" class="com.example.Person"
p:name="John Doe"
p:spouse-ref="jane"/>
</bean>
<bean name="jane" class="com.example.Person"
p:name="John Doe"/>
</bean>
</beans>
```

LAB 1.5

Write a text editor program to perform the different types of auto-wiring.

Steps:

- Create a new java project in Eclipse.
- Right Click the project goto Build path→ Configure path. Add the following jars in the build path
    - antlr-runtime-3.0.1
    - org.springframework.aop-3.1.0.M2
    - org.springframework.asm-3.1.0.M2
    - org.springframework.aspects-3.1.0.M2
    - org.springframework.beans-3.1.0.M2
    - org.springframework.context.support-3.1.0.M2
    - org.springframework.context-3.1.0.M2
    - org.springframework.core-3.1.0.M2
    - org.springframework.expression-3.1.0.M2
    - commons-logging-1.1.1
- Create a new package org.capgemini and add one new class called TextEditor.java.

TextEditor.java

```
package org.capgemini;

public class TextEditor {
	private SpellChecker spellChecker;
	private String name;
	public SpellChecker getSpellChecker() {
		return spellChecker;
	}
	public void setSpellChecker(SpellChecker spellChecker) {
		this.spellChecker = spellChecker;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
```

```
      }
      public void checkSpell()
      {
            spellChecker.checkSplling();
      }

}
```

- Add new class SpellChecker.java under the org.capgemini package

SpellChecker.java

```
package org.capgemini;

public class SpellChecker {

      public SpellChecker()
      {
            System.out.println("Inside of SpellChecker
constructor.");
      }

      public void checkSplling()
      {
            System.out.println("Checking Spelling");
      }
}
```

- Include Beans.xml file under scr folder. The highlighted lines shows that auto wired 'byName'.
  Beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.springframework.org/schema/b
eans http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">

<!-- Definition for textEditor bean -->
<bean id="textEditor" class="org.capgemini.TextEditor"
autowire="byName">
<property name="name" value="Generic Text Editor" />
</bean>
```

```
<!-- Definition for spellChecker bean -->
<bean id="spellChecker" class="org.capgemini.SpellChecker">
</bean>
</beans>
```

Output:

```
Inside of SpellChecker constructor.
Checking Spelling
```

AutoWire 'byType':

Beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.springframework.org/schema/b
eans http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">

<!-- Definition for textEditor bean -->
<bean id="textEditor" class="org.capgemini.TextEditor"
autowire="byType">
<property name="name" value="Generic Text Editor" />
</bean>

<!-- Definition for spellChecker bean -->
<bean id="spellChecker" class="org.capgemini.SpellChecker">
</bean>
</beans>
```

AutoWire 'byConstructor':

Add one constructor in your TextEditor.java file

```
package org.capgemini;
public class TextEditor {
	private SpellChecker spellChecker;
	private String name;
	public TextEditor(SpellChecker spellChecker, String name) {
		this.spellChecker = spellChecker;
		this.name = name;
	}
	public SpellChecker getSpellChecker() {
		return spellChecker;
	}
	public String getName() {
		return name;
	}
	public void checkSpell()
	{
		spellChecker.checkSplling();
	}

}
```

Beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/b
eans http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">

<!-- Definition for textEditor bean -->
<bean id="textEditor" class="org.capgemini.TextEditor"
autowire="constructor">
<constructor-arg value="Generic Text Editor"/>
</bean>

<!-- Definition for spellChecker bean -->
<bean id="spellChecker" class="org.capgemini.SpellChecker">
</bean>
</beans>
```

LAB 1.6

Write a Spring MVC program which demonstrate the MVC in detail which will print 'Spring3 MVC, Hello World!!!!!'

Steps:

- Create a new Dynamic web project in Eclipse
- Add the below mentioned Spring MVC jar files under the lib directory which comes under the webcontent directory.
    - commons-logging-1.0.4.jar
    - jstl-1.2.jar
    - org.springframework.asm-3.1.0.RELEASE-A.jar
    - org.springframework.beans-3.1.0.RELEASE-A.jar
    - org.springframework.context-3.1.0.RELEASE-A.jar
    - org.springframework.core-3.1.0.RELEASE-A.jar
    - org.springframework.expression-3.1.0.RELEASE-A.jar
    - org.springframework.web.servlet-3.1.0.RELEASE-A.jar
    - org.springframework.web-3.1.0.RELEASE-A.jar
- Add one index.jsp file under webcontent.

Index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-
8859-1">
<title>Spring3 MVC - HelloWorld</title>
</head>
<body>
<a href="hello.html">Click Here</a>
</body>
</html>
```

- Create one new folder in the name of ‘jsp’ under WEB-INF. Add one ‘ hello.jsp’ file under WEB-INF/jsp folder.

hello.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-
8859-1">
<title>Spring3 MVC - HelloWorld</title>
</head>
<body>
${message}
</body>
</html>
```

- Create new java class called ‘HelloWorldController’ within org.capgemini package. This class act as a controller.

HelloWorldController.java

```
package org.capgemini;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class HelloWorldController {

     @RequestMapping("/hello")
     public ModelAndView sayHello()
     {
          String msg="Spring3 MVC, Hello World!!!!!";
          return new ModelAndView("hello", "message", msg);
     }
}
```

- Mention the bellow configuration in web.xml file .

Web.xml

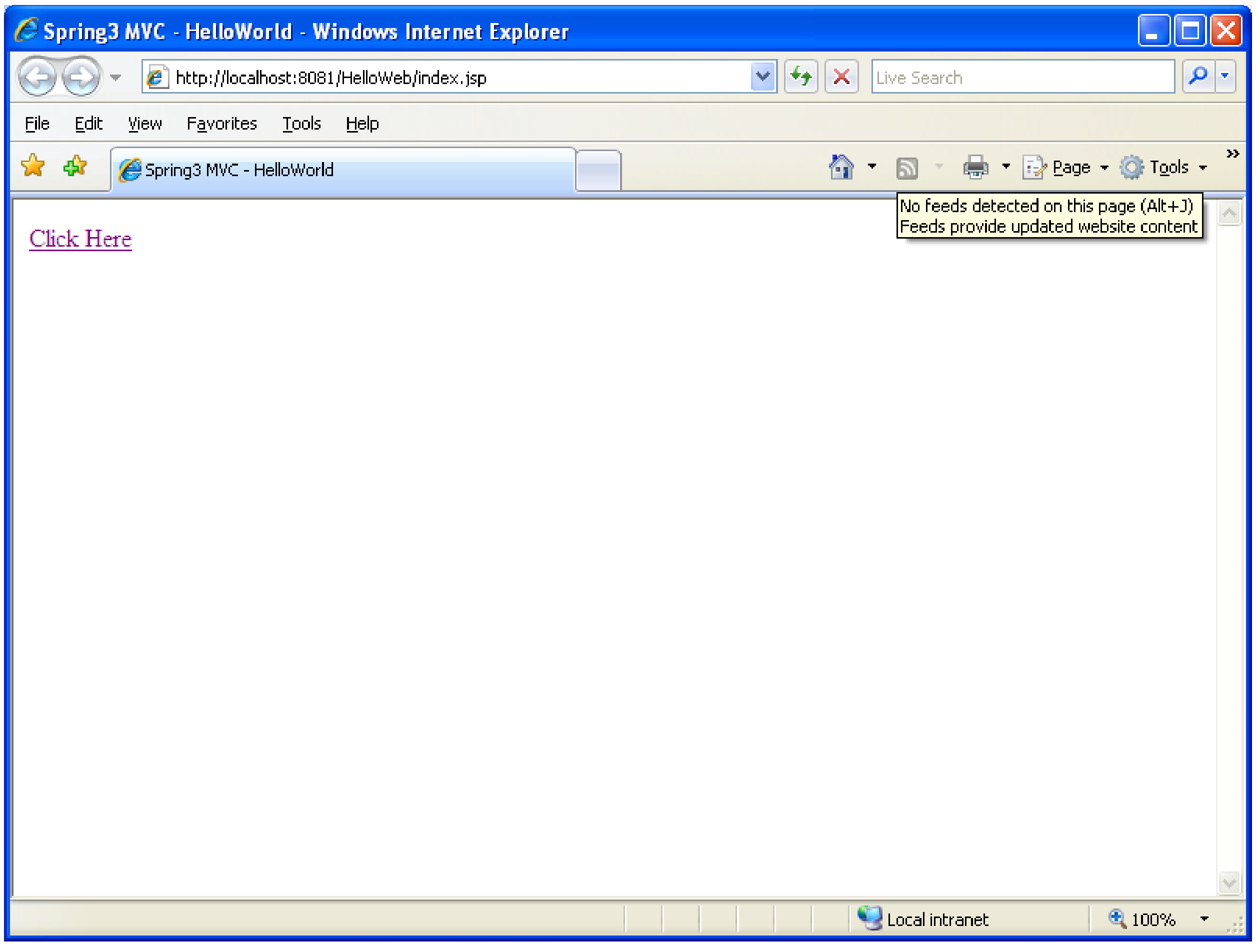
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="WebApp_ID" version="2.5">
  <display-name>HelloWeb</display-name>
  <welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <servlet>
  <servlet-name>springmvc</servlet-name>
  <servlet-class>
     org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
  <servlet-name>springmvc</servlet-name>
  <url-pattern>*.html</url-pattern>
  </servlet-mapping>
</web-app>
```

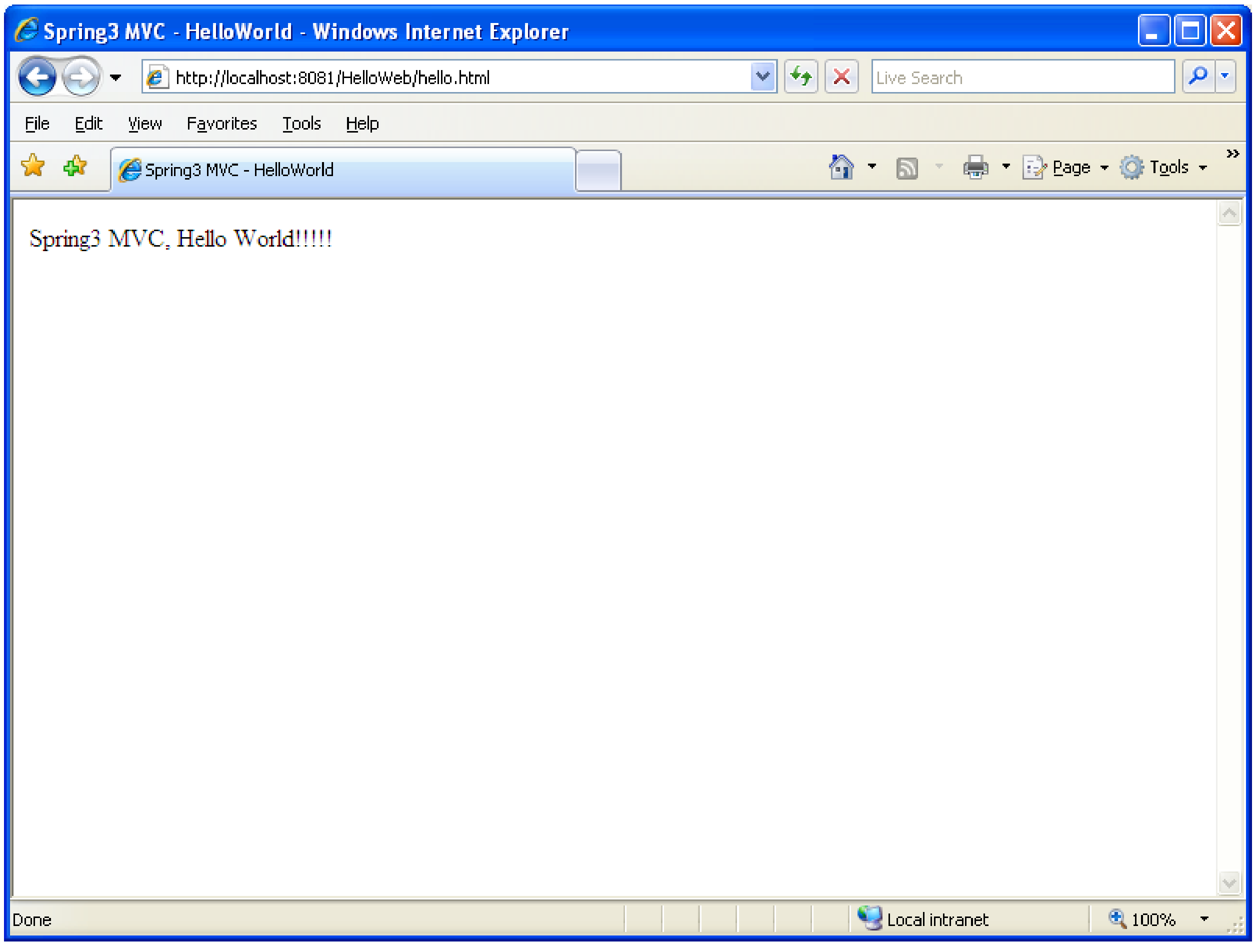
- Add new xml file in the name of ‘springmvc-servlet.xml’ under the WEB-INF directory.

springmvc-servlet.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-
3.0.xsd">

<!-- It will load all the components from the package
org.capgemini  -->
<context:component-scan base-package="org.capgemini"/>

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

</beans>
```

Output:


Spring3 MVC - HelloWorld - Windows Internet Explorer
http://localhost:8081/HelloWeb/index.jsp
Live Search
File Edit View Favorites Tools Help
Spring3 MVC - HelloWorld
Page Tools
No feeds detected on this page (Alt+J)
Feeds provide updated website content
Click Here
Local intranet
100%


Spring3 MVC - HelloWorld - Windows Internet Explorer
http://localhost:8081/HelloWeb/hello.html
Live Search
File Edit View Favorites Tools Help
Spring3 MVC - HelloWorld
Page Tools
Done
Local intranet
100%

Spring3 MVC, Hello World!!!!!

Learning:

- From the above example we can understand how to write a simple MVC program in spring3.

LAB 1.7

Write a Spring MVC program which contains customer details (name, address, mobile, amount) in form. Accept the customer details and print it into the next page.

- Create a new Dynamic web project in Eclipse
- Add the below mentioned Spring MVC jar files under the lib directory which comes under the webcontent directory.
  - commons-logging-1.0.4.jar
  - jstl-1.2.jar
  - org.springframework.asm-3.1.0.RELEASE-A.jar
  - org.springframework.beans-3.1.0.RELEASE-A.jar
  - org.springframework.context-3.1.0.RELEASE-A.jar
  - org.springframework.core-3.1.0.RELEASE-A.jar
  - org.springframework.expression-3.1.0.RELEASE-A.jar
  - org.springframework.web.servlet-3.1.0.RELEASE-A.jar
  - org.springframework.web-3.1.0.RELEASE-A.jar
- Add one index.jsp file under webcontent.

Index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-
8859-1">
<title>Spring3 MVC - Customer Form</title>
</head>
<body>
<a href="customer.html">Customer Registration</a>
</body>
</html>
```

- Create one new folder in the name of ‘jsp’ under WEB-INF. Add one ‘customer.jsp’ file under WEB-INF/jsp folder.
  Customer.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Customer Registration Form</title>
</head>
<body>
<form:form method="post" action="showCustomer.html">
<fieldset>
    <legend>Customer Registration Form</legend>
<table>
<tr>
<td><form:label path="cname">Name :</form:label></td>
<td><form:input path="cname" size="20"/></td>
</tr>
<tr>
<td><form:label path="address">Address:</form:label></td>
<td><form:textarea path="address" rows="5" cols="20"/></td>
</tr>
<tr>
<td><form:label path="mobile">Mobile :</form:label></td>
<td><form:input path="mobile" size="20"/><br></td>
</tr>
<tr>
<td><form:label path="amount">Amount :</form:label></td>
<td><form:input path="amount" size="20"/></td>
</tr>
<tr><td></td>
<td><input type="submit" name="submit" value="Show Details"></td>
</tr>
</table>
</fieldset>
</form:form></body></html>
```

- Add one more jsp file in the name of ‘showCustomer.jsp’

showCustomer.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-
8859-1">
<title>Customer Registration Form</title>
</head>
<body>

<form>

<table>
<tr>
<th>Name :</th>
<td>${customer.cname}</td>
</tr>

<tr>
<th>Address :</th>
<td>${customer.address}</td>
</tr>


<tr>
<th>Mobile :</th>
<td>${customer.mobile}</td>
</tr>

<tr>
<th>Amount :</th>
<td>${customer.amount}</td>
</tr>
</table>
</form>
</body>
</html>
```

- Add one POJO class called customer.java under org.capgemini package

Customer.java

```
package org.capgemini;

public class Customer {
     private String cname;
     private String address;
     private String mobile;
     private Double amount;

     public String getCname() {
          return cname;
     }
     public void setCname(String cname) {
          this.cname = cname;
     }
     public String getAddress() {
          return address;
     }
     public void setAddress(String address) {
          this.address = address;
     }
     public String getMobile() {
          return mobile;
     }
     public void setMobile(String mobile) {
          this.mobile = mobile;
     }
     public Double getAmount() {
          return amount;
     }
     public void setAmount(Double amount) {
          this.amount = amount;
     }

}
```

- Add one CustomerController.java class under org.capgemini package

CustomerController.java

```
package org.capgemini;

import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import
org.springframework.web.bind.annotation.SessionAttributes;
import org.springframework.web.servlet.ModelAndView;

@Controller
@SessionAttributes
public class CustomerController {


/*@ModelAttribute will bind the data from request to the
customer object.*/
     @RequestMapping(value="/showCustomer",
method=RequestMethod.POST)
     public ModelAndView getCustomer(
@ModelAttribute("customer") Customer customer, BindingResult
result)
     {
          return new ModelAndView("showCustomer", "customer",
customer);
     }

     @RequestMapping("/customer")
     public ModelAndView showCustomer()
     {
          return new ModelAndView("customer","command",new
Customer());
     }

}
```

- The web.xml and springmvc-servlet.xml are same like the previous project.

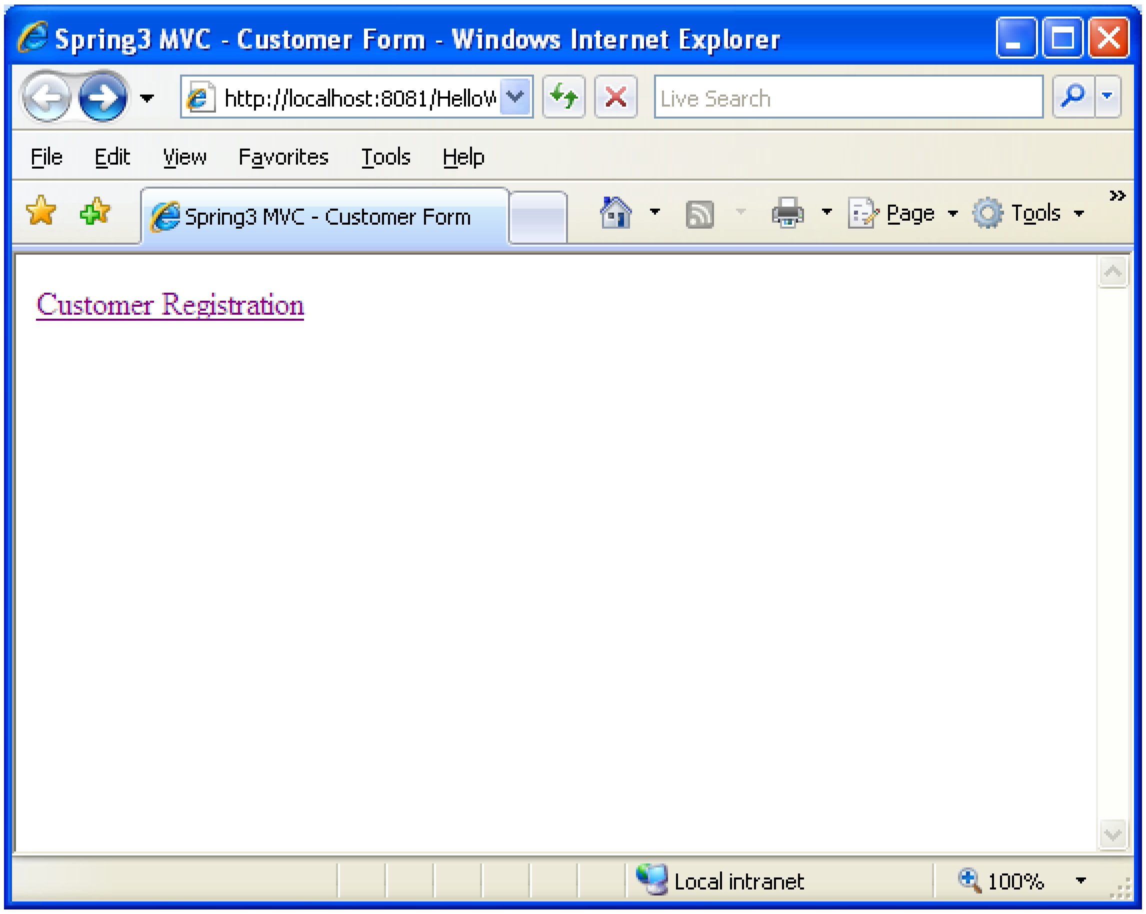
Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="WebApp_ID" version="2.5">
  <display-name>HelloWeb</display-name>
  <welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <servlet>
  <servlet-name>springmvc</servlet-name>
  <servlet-class>
     org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
  <servlet-name>springmvc</servlet-name>
  <url-pattern>*.html</url-pattern>
  </servlet-mapping>
</web-app>
```

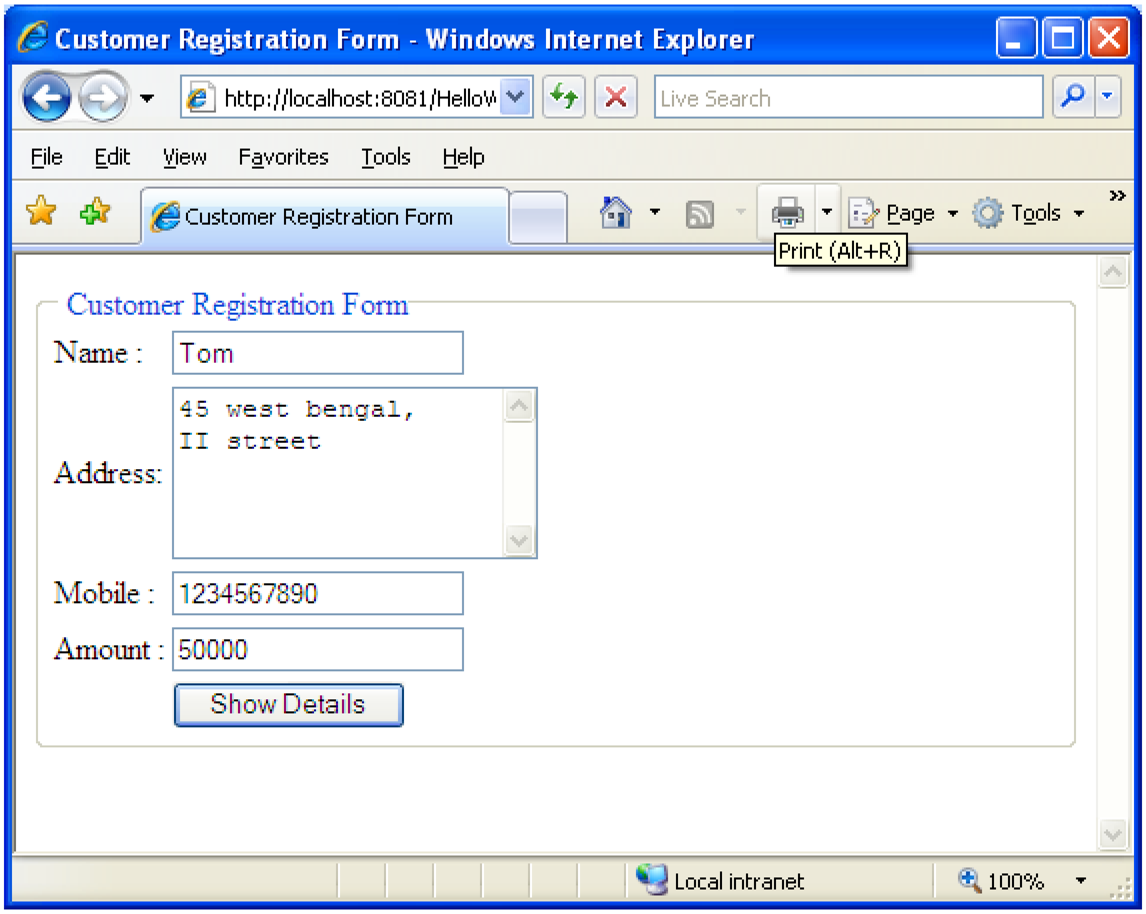
springmvc-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-
3.0.xsd">

<!-- It will load all the components from the package
org.capgemini  -->
<context:component-scan base-package="org.capgemini"/>

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

</beans>
```

output


Spring3 MVC - Customer Form - Windows Internet Explorer
http://localhost:8081/HelloW
Live Search
File Edit View Favorites Tools Help
Spring3 MVC - Customer Form
Page Tools
Local intranet
100%

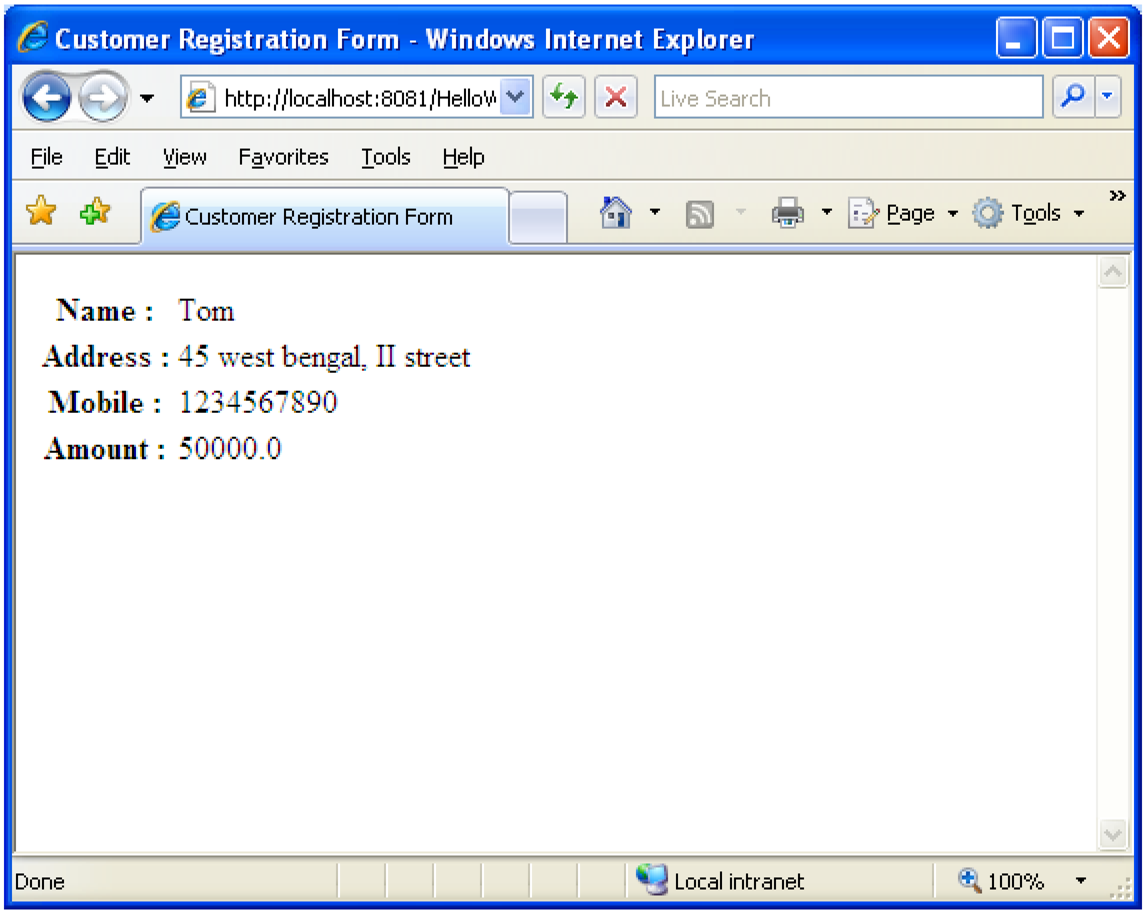
Customer Registration


Customer Registration Form - Windows Internet Explorer
http://localhost:8081/HelloW
Live Search
File Edit View Favorites Tools Help
Customer Registration Form
Page Tools
Print (Alt+R)
Local intranet
100%

Customer Registration Form

Name :	Tom
Address:	45 west bengal, II street
Mobile :	1234567890
Amount :	50000

Show Details


Customer Registration Form - Windows Internet Explorer
http://localhost:8081/HelloW
Live Search
File Edit View Favorites Tools Help
Customer Registration Form
Page Tools
Done
Local intranet
100%

Name : Tom
Address : 45 west bengal, II street
Mobile : 1234567890
Amount : 50000.0

Learning:

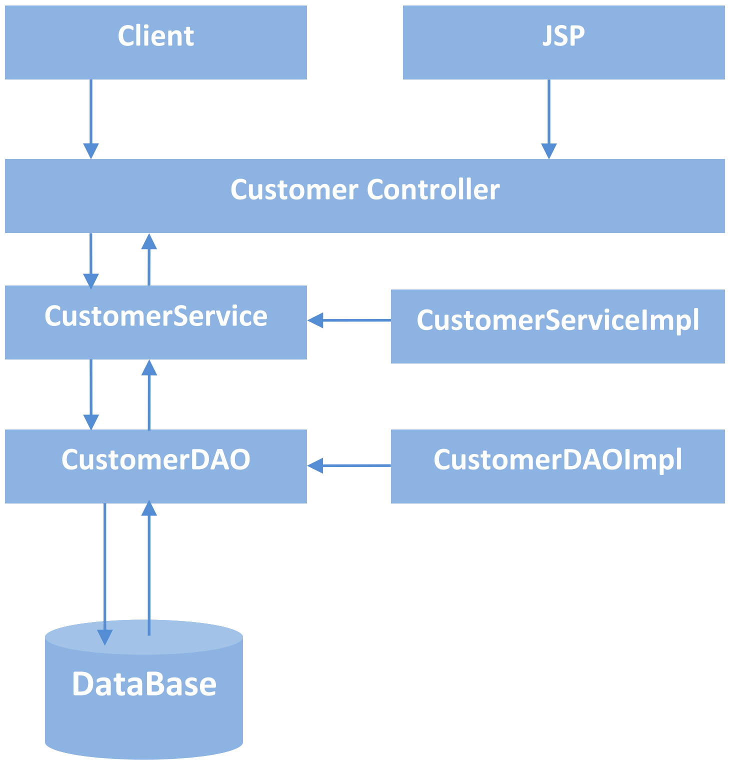
- From the above example we can understand that how to accept the form inputs from Spring3 MVC .

LAB 1.8

Write a Spring MVC program which contains customer details (name, address, mobile, amount) in form. Accept the customer details and store the data into the table named customer.

Step:

- Create a dynamic web project
- Add the Spring jars
- Add Hibernate jars.



- This is the architecture of implementation.
- Take a help of this architecture and implement it.

LAB 1.9

Write a Spring JDBC program which perform CRUD operations with database table customer (custid, custname, mobile, deposit, reg_date).

Steps:

- Create a customer table in mysql. The table creation DDL query is as follows.

```
CREATE TABLE Customer(
CUSTID INT NOT NULL AUTO_INCREMENT,
CUSTNAME VARCHAR(20) NOT NULL,
MOBILE VARCHAR(10),
DEPOSIT NUMERIC(8,2) NOT NULL,
REG_DATE DATE NOT NULL,
PRIMARY KEY (ID) );
```

- Create a new Java project in Eclipse
- Right Click the project goto Build path → configure build path → add the following jars.
    - org.springframework.aop-3.1.0.M2
    - org.springframework.asm-3.1.0.M2
    - org.springframework.aspects-3.1.0.M2
    - org.springframework.beans-3.1.0.M2
    - org.springframework.context.support-3.1.0.M2
    - org.springframework.context-3.1.0.M2
    - org.springframework.core-3.1.0.M2
    - org.springframework.expression-3.1.0.M2
    - org.springframework.jdbc.jar
    - org.springframework.transaction.jar
    - commons-logging-1.1.1
    - mysql-connector-java.jar

Note:

In this project we used mysql database so that we included **mysql-connector-java.jar** file. If you are using different database you should include the relevant jars.

- Create a new package org.capgemini and add a new class called Customer.java under this package. This is a POJO class.

Customer.java

```
package org.capgemini;
import java.util.Date;
     public class Customer
     {
          private Integer custid;
          private String custname;
          private String mobile;
          private Double deposit;
          private Date reg_date;

          public Integer getCustid() {
               return custid;
          }
          public void setCustid(Integer custid) {
               this.custid = custid;
          }
          public String getCustname() {
               return custname;
          }
          public void setCustname(String custname) {
               this.custname = custname;
          }
          public String getMobile() {
               return mobile;
          }
          public void setMobile(String mobile) {
               this.mobile = mobile;
          }
          public Double getDeposit() {
               return deposit;
          }
          public void setDeposit(Double deposit) {
               this.deposit = deposit;
          }
          public Date getReg_date() {
               return reg_date;
          }
          public void setReg_date(Date reg_date) {
               this.reg_date = reg_date;
          }

          @Override
```

```
           public String toString() {
                return "\nID"+custid + "Name :" + custname + "\nMobile
: " + mobile + "\nDeposit :" + deposit + "\nRegistration Date:" +
reg_date;
           }

   }
```

- Add CustomerDAO.java interface under the org.capgemini package

CustomerDAO.java

```
package org.capgemini;
import java.util.GregorianCalendar;
import java.util.List;
import javax.sql.DataSource;

public interface CustomerDAO {
     /*** This is the method to be used to initialize
      *    * database resources ie. connection.
      */
     public void setDataSource(DataSource ds);


     /**This is the method to be used to create
      * a record in the  Customer table. */
     public void create(String cname,String mobile, Double deposit,
GregorianCalendar regdate);

     /** This is the method to be used to list down
      * a record from the Customer table corresponding
      * to a passed Customer id. */
     public Customer getCustomer(Integer custid);

     /** This is the method to be used to list down
      * all the records from the Customer table. */
     public List<Customer> listCustomers();

     /** This is the method to be used to delete
      * a record from the Customer table corresponding
      * to a passed Customer id. */
     public void delete(Integer id);

     /**This is the method to be used to update
      * a record into the Customer table. */
     public void update(Integer id, String mobile);

}
```

- Add CustomerMapper.java class under the org.capgemini package. This Class used to map a single as a pojo object.

CustomerMapper .java

```
import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;

public class CustomerMapper implements RowMapper<Customer> {

	@Override
	public Customer mapRow(ResultSet rs, int rownum) throws 
SQLException {
		Customer cust=new Customer();
		cust.setCustid(rs.getInt("custid"));
		cust.setCustname(rs.getString("custname"));
		cust.setMobile(rs.getString("mobile"));
		cust.setDeposit(rs.getDouble("deposit"));
		cust.setReg_date(rs.getDate("reg_date"));
		return cust;
	}

}
```

- Add CustomerJDBCTemplater.java class under the org.capgemini package. This class contains all the implementation of CRUD operations which implements the DAO interface.

CustomerJDBCTemplate .java

```
package org.capgemini;
import java.util.GregorianCalendar;
import java.util.List;
import javax.sql.DataSource;
import org.springframework.jdbc.core.JdbcTemplate;

public class CustomerJDBCTemplate implements CustomerDAO {

	private DataSource dataSource;
	private JdbcTemplate jdbcTemplateObject;

	@Override
	public void setDataSource(DataSource ds) {
		this.dataSource=ds;
		this.jdbcTemplateObject=new JdbcTemplate(dataSource);

	}
	@Override
	public void create(String cname, String mobile, Double deposit, 
GregorianCalendar regdate) {
		String sql="insert into 
customer(custname,mobile,deposit,reg_date) values(?,?,?,?)";

	jdbcTemplateObject.update(sql,cname,mobile,deposit,regdate);
		System.out.println("Created Record Name=" + cname );

	}

	@Override
	public Customer getCustomer(Integer custid) {
		String sql="select * from customer where custid=?";
		Customer customer=jdbcTemplateObject.queryForObject(sql, 
new Object[]{custid},new CustomerMapper());
		return customer;
	}

	@Override
	public List<Customer> listCustomers() {
		String sql="select * from customer";
		List<Customer> customers=jdbcTemplateObject.query(sql,new 
CustomerMapper());
		return customers;
	}

	@Override
	public void delete(Integer id) {
		String sql="delete from customer where custid=?";
		jdbcTemplateObject.update(sql, id);
		System.out.println("Record " + id + " Deleted successfully" 
);
	}

	@Override
	public void update(Integer id, String mobile) {
		String sql="update customer set mobile=? where custid=?";
		jdbcTemplateObject.update(sql, mobile,id);
		System.out.println("Record " + id + " Updated successfully" 
);
	}

}
```

- Add new file in the name of MainApp.java that contains the user interaction menu details to perform CRUD operations.

MainApp.java

```
package org.capgemini;

import java.util.GregorianCalendar;
import java.util.List;
import java.util.Scanner;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

      public static void main(String[] args) {

            ApplicationContext context=new
ClassPathXmlApplicationContext("Beans.xml");
            CustomerJDBCTemplate custjdbctemp=
(CustomerJDBCTemplate)context.getBean("jdbctemp");

            int choice=0;
            do
            {
                  System.out.println("Menu\n1.Insert Record\n2.Find
\n3.ListAll \n4.Update \n5.Delete \n6.Exit");
                  System.out.println("Enter Your Choice(1to4):");
                  Scanner sc=new Scanner(System.in);
                  choice=sc.nextInt();

                  switch(choice)
                  {
                  case 1:
                        System.out.println("\nEnter Name:");
                        String cname=sc.next();
                        System.out.println("\nEnter Mobile:");
                        String mobile=sc.next();
                        System.out.println("\nEnter Deposit Amount:");
                        Double amt=sc.nextDouble();
                        System.out.println("\nEnter RegistrationDate");
                        System.out.println("\nEnter Date:");
                        int date=sc.nextInt();
                        System.out.println("\nEnter Month(0-11):");
                        int month=sc.nextInt();
                        System.out.println("\nEnter Year:");
                        int year=sc.nextInt();
                        GregorianCalendar regdate=new
GregorianCalendar(year, month, date);
```

```
                              custjdbctemp.create(cname,mobile,amt,regdate);
                              break;

                    case 2:
                              System.out.println("\nEnter Customer ID to 
Search:");
                              int custid=sc.nextInt();
                              Customer cust=custjdbctemp.getCustomer(custid);
                              System.out.println("Customer Details\n"+cust);
                              break;
                    case 3:

                              List<Customer> 
clist=custjdbctemp.listCustomers();

                              for(Customer customer :clist)
                              {
                                      System.out.println(customer);
                              }
                              break;
                    case 4:
                              System.out.println("\nEnter Customer ID to 
Update:");
                              int cust_id=sc.nextInt();
                              System.out.println("\nEnter new Mobile number to 
upadate");
                              String new_mobile=sc.next();
                              custjdbctemp.update(cust_id, new_mobile);
                              break;
                    case 5:
                              System.out.println("\nEnter Customer ID to 
Delete:");
                              int del_cust_id=sc.nextInt();
                              custjdbctemp.delete(del_cust_id);
                              break;
                    case 6:
                              System.exit(0);
                    default:
                              System.out.println("Invalid Choice");
                    }

            }while(choice>0);


        }

}
```

- The Beans.xml file should be placed under the src folder. It contains the spring data source configuration details.

Beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      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-3.0.xsd">

<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name="driverClassName" value="com.mysql.jdbc.Driver" />
<property name="url" value="jdbc:mysql://localhost:3306/cap" />
<property name="username" value="root"/>
<property name="password" value="capmysql"/>
</bean>

<bean id="jdbctemp" class="org.capgemini.CustomerJDBCTemplate">
<property name="dataSource" ref="dataSource" />
</bean>
</beans>
```

Output:

1.Creating new Row into the Customer table

```
Menu
1.Insert Record
2.Find
3.ListAll
4.Update
5.Delete
6.Exit
Enter Your Choice(1to4):
1

Enter Name:
Jack
```

```
Enter Mobile:
9923129033

Enter Deposit Amount:
80000

Enter RegistrationDate

Enter Date:
24

Enter Month(0-11):
8

Enter Year:
2001
Created Record Name=Jack
```

2.Reading a particular record from the Customer table

```
Menu
1.Insert Record
2.Find
3.ListAll
4.Update
5.Delete
6.Exit
Enter Your Choice(1to4):
2

Enter Customer ID to Search:
7
Customer Details

ID7Name :Jack
Mobile : 9923129033
Deposit :80000.0
Registration Date:2001-09-24
```

3.Updating a particular record from the Customer table

```
Menu
1.Insert Record
2.Find
3.ListAll
4.Update
5.Delete
6.Exit
Enter Your Choice(1to4):
4

Enter Customer ID to Update:
7

Enter new Mobile number to upadate
8123499000
Record 7 Updated successfully
```

4.Deleting a particular record from the Customer table

```
Menu
1.Insert Record
2.Find
3.ListAll
4.Update
5.Delete
6.Exit
Enter Your Choice(1to4):
5

Enter Customer ID to Delete:
7
Record 7 Deleted successfully
```

Learning:

- From the above example we can understand how to write a simple JDBC program in spring3. This program also demonstrates the usage of DAO design pattern. And how the Row Mapper interface helps us to retrieve the record from the table.

LAB 1.10

- *Write a Spring JDBC program which interact with the customer table (custid, custname, mobile, deposit, reg_date). Write one stored procedure in Data base to retrieve the customer details for a particular customer id. Call the procedure in spring application.*

- Use the same customer table.

- Create one new stored procedure in mysql as follows to retrieve the data from the Customer table. The stored procedure name is **getCustomerRecord.**

  This is a stored procedure called <u>getCustomerRecord</u> in CAP database.

```
DELIMITER $$
DROP PROCEDURE IF EXISTS `CAP`.`getCustomerRecord` $$

CREATE PROCEDURE `CAP`.`getCustomerRecord` ( IN cust_id INTEGER, OUT cust_name
VARCHAR(20),OUT mobile1 VARCHAR(10), OUT deposit1 NUMERIC(8,2),OUT regdate DATE)
BEGIN
SELECT custname,mobile,deposit,reg_date INTO cust_name,mobile1,deposit1,regdate FROM
Customer where custid = cust_id;
END $$

DELIMITER ;
```

- Simply make few changes in CustomerJDBCTemplate.java file**.** Means that we have a method called **getCustomer** just update the getCustomer method. Inside of this method call the procedure which we have created earlier. The following code snippet will show the updations.

CustomerJDBCTemplate.java

```
package org.capgemini;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.List;
import java.util.Map;

import javax.sql.DataSource;
import org.springframework.jdbc.core.JdbcTemplate;
```

```
import
org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import
org.springframework.jdbc.core.namedparam.SqlParameterSource;
import org.springframework.jdbc.core.simple.SimpleJdbcCall;

public class CustomerJDBCTemplate implements CustomerDAO {

	private DataSource dataSource;
	private JdbcTemplate jdbcTemplateObject;
	private SimpleJdbcCall jdbccall;

	@Override
	public void setDataSource(DataSource ds) {
		this.dataSource=ds;
		this.jdbcTemplateObject=new JdbcTemplate(dataSource);
		this.jdbccall=new
SimpleJdbcCall(ds).withProcedureName("getCustomerRecord");
	}

	@Override
	public void create(String cname, String mobile, Double
deposit, GregorianCalendar regdate) {
		String sql="insert into
customer(custname,mobile,deposit,reg_date) values(?,?,?,?)";

	jdbcTemplateObject.update(sql,cname,mobile,deposit,regdate)
;
		System.out.println("Created Record Name=" + cname );

	}

	@Override
	public Customer getCustomer(Integer custid) {
		/*String sql="select * from customer where custid=?";
		Customer
customer=jdbcTemplateObject.queryForObject(sql, new
Object[]{custid},new CustomerMapper());*/

		SqlParameterSource in=new
MapSqlParameterSource("cust_id", custid);

		Map<String,Object> out=jdbccall.execute(in);

		Customer customer=new Customer();
		customer.setCustid(custid);
```

```
          customer.setCustname((String)out.get("cust_name"));
          customer.setDeposit(new
Double(out.get("deposit1").toString()));
          customer.setMobile((String)out.get("mobile1"));
          customer.setReg_date((Date)out.get("regdate"));
          return customer;
     }

     @Override
     public List<Customer> listCustomers() {
          String sql="select * from customer";
          List<Customer>
customers=jdbcTemplateObject.query(sql,new CustomerMapper());
          return customers;
     }

     @Override
     public void delete(Integer id) {
          String sql="delete from customer where custid=?";
          jdbcTemplateObject.update(sql, id);
          System.out.println("Record " + id + " Deleted
successfully" );
     }

     @Override
     public void update(Integer id, String mobile) {
          String sql="update customer set mobile=? where
custid=?";
          jdbcTemplateObject.update(sql, mobile,id);
          System.out.println("Record " + id + " Updated
successfully" );
     }

}
```

Output:

```
Menu
1.Insert Record
2.Find
3.ListAll
4.Update
5.Delete
6.Exit
Enter Your Choice(1to4):
2

Enter Customer ID to Search:
1
Customer Details

ID1Name :TOM
Mobile : 3243243212
Deposit :34000.0
Registration Date:2001-03-27
```

LAB 1.11

Write a simple Spring program to perform JDBC transaction.

Steps:

- Create the following tables in mysql. The table creation DDL query is as follows.

```
CREATE TABLE Customer(
CUSTID INT NOT NULL AUTO_INCREMENT,
CUSTNAME VARCHAR(20) NOT NULL,
MOBILE VARCHAR(10),
DEPOSIT NUMERIC(8,2) NOT NULL,
REG_DATE DATE NOT NULL,
PRIMARY KEY (ID) );


CREATE TABLE Orders(
OID INT NOT NULL AUTO_INCREMENT,
CNO INT NOT NULL,
DEPOSIT NUMERIC(8,2) NOT NULL,
PRIMARY KEY (OID) );
```

- Create a new java project in Eclipse.
- Right Click the project go to **build path→ Configure path**. Add the following jars in the build path
  - antlr-runtime-3.0.1
  - org.springframework.aop-3.1.0.M2
  - org.springframework.asm-3.1.0.M2
  - org.springframework.aspects-3.1.0.M2
  - org.springframework.beans-3.1.0.M2
  - org.springframework.context.support-3.1.0.M2
  - org.springframework.context-3.1.0.M2
  - org.springframework.core-3.1.0.M2
  - org.springframework.expression-3.1.0.M2
  - commons-logging-1.1.1
  - org.springframework.transaction.jar

- mysql-connector-java.jar
- org.springframework.jdbc.jar

- Create new package in the name of org.capgemini and include the Customer.java file in that package.
  Customer.java

```
package org.capgemini;
import java.util.Date;

public class Customer
{
          private Integer custid;
          private String custname;
          private String mobile;
          private Double deposit;
          private Date reg_date;

          public Integer getCustid() {
               return custid;
          }
          public void setCustid(Integer custid) {
               this.custid = custid;
          }
          public String getCustname() {
               return custname;
          }
          public void setCustname(String custname) {
               this.custname = custname;
          }
          public String getMobile() {
               return mobile;
          }
          public void setMobile(String mobile) {
               this.mobile = mobile;
          }
          public Double getDeposit() {
               return deposit;
          }
          public void setDeposit(Double deposit) {
               this.deposit = deposit;
          }
          public Date getReg_date() {
                return reg_date;
            }
            public void setReg_date(Date reg_date) {
                this.reg_date = reg_date;
            }

            @Override
            public String toString() {
                return "\nID"+custid + "Name :" + custname +
"\nMobile : " + mobile + "\nDeposit :" + deposit +
"\nRegistration Date:" + reg_date;
            }
}
```

- Include CustomerDAO.java within the same package.

CustomerDAO.java

```
package org.capgemini;
import java.util.GregorianCalendar;
import java.util.List;
import javax.sql.DataSource;

public interface CustomerDAO {
     /*** This is the method to be used to initialize
      *   * database resources ie. connection.
      */
     public void setDataSource(DataSource ds);


     /**This is the method to be used to create
      * a record in the  Customer table. */
     public void create(String cname,String mobile, Double
deposit, GregorianCalendar regdate);


     /** This is the method to be used to list down
      * all the records from the Customer table. */
     public List<Customer> listCustomers();

}
```

- Add CustomerJDBCTemplate.java under the same package.

CustomerJDBCTemplate.java

```
package org.capgemini;
import java.util.GregorianCalendar;
import java.util.List;
import javax.sql.DataSource;

import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
import
org.springframework.transaction.PlatformTransactionManager;
import
org.springframework.transaction.TransactionDefinition;
import org.springframework.transaction.TransactionStatus;
import
org.springframework.transaction.support.DefaultTransactionD
efinition;

public class CustomerJDBCTemplate implements CustomerDAO {

	private DataSource dataSource;
	private JdbcTemplate jdbcTemplateObject;
	private PlatformTransactionManager transactionManager;


	public void setTransactionManager(
PlatformTransactionManager transactionManager) {
		this.transactionManager = transactionManager;
		}

	@Override
	public void setDataSource(DataSource ds) {
		this.dataSource=ds;
		this.jdbcTemplateObject=new
JdbcTemplate(dataSource);

	}

	@Override
```

```
    public void create(String cname, String mobile, Double
deposit, GregorianCalendar regdate) {

        TransactionDefinition def = new
DefaultTransactionDefinition();

        TransactionStatus status =
transactionManager.getTransaction(def);
        try
        {
        String sql="insert into
customer(custname,mobile,deposit,reg_date)
values(?,?,?,?)";

    jdbcTemplateObject.update(sql,cname,mobile,deposit,regd
ate);

        String sql1="select max(custid) from customer";
        int cid=jdbcTemplateObject.queryForInt(sql1);

        String sql2="insert into orders(cno,deposit)
values(?,?)";
        jdbcTemplateObject.update(sql2, cid,deposit);

        transactionManager.commit(status);



        System.out.println("Created Record Name=" + cname
);
        }catch(DataAccessException ex)
        {
            System.out.println("Error in Creating Record,
Rolling back");
            transactionManager.rollback(status);
            throw ex;

        }
    }


    @Override
    public List<Customer> listCustomers() {
        String sql="select * from customer";
        List<Customer>
customers=jdbcTemplateObject.query(sql,new
CustomerMapper());
        return customers;
    }

}
```

- Add CustomerMapper.java class used to map the object as a record in the customer table.
  CustomerMapper.java

```
package org.capgemini;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;

public class CustomerMapper implements RowMapper<Customer>
{
    @Override
    public Customer mapRow(ResultSet rs, int rownum) throws
SQLException {
        Customer cust=new Customer();
        cust.setCustid(rs.getInt("custid"));
        cust.setCustname(rs.getString("custname"));
        cust.setMobile(rs.getString("mobile"));
        cust.setDeposit(rs.getDouble("deposit"));
        cust.setReg_date(rs.getDate("reg_date"));
        return cust;
    }

}
```

- Then add Beans.xml file under src folder.
  Beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.springframework.org/schema/b
eans http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">

<!-- Initialization for DataSource -->
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSour
ce">
<property name="driverClassName" value="com.mysql.jdbc.Driver"
/>
<property name="url" value="jdbc:mysql://localhost:3306/cap" />
<property name="username" value="root"/>
<property name="password" value="capmysql"/>
</bean>

<!-- Initialization for TransactionManager -->
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransaction
Manager">
<property name="dataSource" ref="dataSource" />
</bean>

<!-- Definition for CustomerTemplate Bean -->
<bean id="jdbctemp" class="org.capgemini.CustomerJDBCTemplate">
<property name="dataSource" ref="dataSource" />
<property name="transactionManager" ref="transactionManager" />
</bean>

</beans>
```

- At last include MainApp.java file. Then run it.

```
package org.capgemini;
import java.util.GregorianCalendar;
import java.util.List;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationConte
xt;

public class MainApp {

     public static void main(String[] args) {

          ApplicationContext context=new
ClassPathXmlApplicationContext("Beans.xml");
          CustomerJDBCTemplate custjdbctemp=
(CustomerJDBCTemplate)context.getBean("jdbctemp");

          custjdbctemp.create("Jessy", "7780912354", 4000.00,new
GregorianCalendar(2000,10,11) );
          custjdbctemp.create("Thompson", "8823121231",
7000.00,new GregorianCalendar() );
          custjdbctemp.create("Jhon", "9923100345", 6000.00,new
GregorianCalendar() );


          List<Customer> clst=custjdbctemp.listCustomers();

          for(Customer record:clst)
          {
               System.out.println(record);
          }

     }

}
```

Output:

```
Created Record Name=Jessy
Created Record Name=Thompson
Created Record Name=Jhon

ID1Name :TOM
Mobile : 3243243212
Deposit :34000.0
Registration Date:2001-03-27

ID2Name :Jerry
Mobile : 9043243212
Deposit :34000.0
Registration Date:2000-07-03

ID4Name :Ram
Mobile : 9912345678
Deposit :45000.0
Registration Date:3912-04-21

ID5Name :Ram
Mobile : 9912345678
Deposit :45000.0
Registration Date:2009-04-12

ID6Name :pooja
Mobile : 1234567890
Deposit :67000.0
Registration Date:2011-04-23

ID9Name :Jessy
Mobile : 7780912354
Deposit :4000.0
Registration Date:2000-11-11

ID10Name :Thompson
Mobile : 8823121231
Deposit :7000.0
Registration Date:2012-11-29

ID11Name :Jhon
Mobile : 9923100345
Deposit :6000.0
Registration Date:2012-11-29
```

Output:

Now we get to know that how to perform the database transactions in spring. The above example explains how to do the simple database transactions commit and rollback.

LAB 1.12

Write a Spring program to demonstrate the different types of AOP advice.

Steps:

- Create a new java project in Eclipse.
- Right Click the project goto Build path→ Configure path. Add the following jars in the build path
    - antlr-runtime-3.0.1
    - org.springframework.aop-3.1.0.M2
    - org.springframework.asm-3.1.0.M2
    - org.springframework.aspects-3.1.0.M2
    - org.springframework.beans-3.1.0.M2
    - org.springframework.context.support-3.1.0.M2
    - org.springframework.context-3.1.0.M2
    - org.springframework.core-3.1.0.M2
    - org.springframework.expression-3.1.0.M2
    - commons-logging-1.1.1

  Incule the following additional jars for AOP.

    - aspectj.jar
    - aspectjweaver.jar
    - aspectjrt.jar
- Create a new package called org.capgemini and then include the Student.java file

Student.java

```
package org.capgemini;

public class Student {

     private Integer age;
     private String name;
     public Integer getAge() {
          System.out.println("Age : " + age );
          return age;
     }
     public void setAge(Integer age) {
          this.age = age;
     }
     public String getName() {
          System.out.println("Name : " + name );
          return name;
     }
     public void setName(String name) {
          this.name = name;
     }

     public void printThrowException(){
     System.out.println("Exception raised");
     throw new IllegalArgumentException(); }

}
```

- Add the file Logging.java under the same package which contains the advice methods.

Logging.java

```
package org.capgemini;

public class Logging {
     /** * This is the method which I would like to execute *
      * before a selected method execution. */
     public void beforeAdvice(){
          System.out.println("Going to setup student profile.");
          }
     /** * This is the method which I would like to execute *
      *  after a selected method execution. */

     public void afterAdvice(){
          System.out.println("Student profile has been setup.");
          }

     /** * This is the method which I would like to execute *
      * when any method returns. */
     public void afterReturningAdvice(Object retVal){
          System.out.println("Returning:" + retVal.toString() );
          }
     /** * This is the method which I would like to execute *
      * if there is an exception raised. */
     public void AfterThrowingAdvice(IllegalArgumentException
ex){
          System.out.println("There has been an exception: " +
ex.toString());
          }
}
```

- Include the configuration file Beans.xml

Beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd ">

<aop:config>
<aop:aspect id="log" ref="logging">
<aop:pointcut id="selectAll" expression="execution(*
org.capgemini.*.*(..))"/>
<aop:before pointcut-ref="selectAll" method="beforeAdvice"/>
<aop:after pointcut-ref="selectAll" method="afterAdvice"/>
<aop:after-returning pointcut-ref="selectAll" returning="retVal"
method="afterReturningAdvice"/>
<aop:after-throwing pointcut-ref="selectAll" throwing="ex"
method="AfterThrowingAdvice"/>
</aop:aspect>
</aop:config>


<!-- Definition for student bean -->
<bean id="student" class="org.capgemini.Student">
 <property name="name" value="Tom" />
 <property name="age" value="21"/>
 </bean>


 <!-- Definition for logging aspect -->
<bean id="logging" class="org.capgemini.Logging"/>
```

```
</beans>
```

- Finally include the MainApp.java file. And run it.

MainApp.java

```
package org.capgemini;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

                    public static void main(String[] args) {

                    ApplicationContext context = new
ClassPathXmlApplicationContext("Beans.xml");

                    Student student = (Student) context.getBean("student");

                    student.getName();

                    student.getAge();

                    student.printThrowException();

          }

}
```

Output:

```
Going to setup student profile.
Name : Tom
Student profile has been setup.
Returning:Tom
Going to setup student profile.
Age : 21
Student profile has been setup.
Returning:21
Going to setup student profile.
Exception raised
Student profile has been setup.
There has been an exception: java.lang.IllegalArgumentException
```