TABLE OF CONTENT

Practical 1

AIM: Write a Java Program to demonstrate a Generic Class.

```
package genreic;
import java.util.ArrayList;
import java.util.Iterator;
public class Genreic {
   public static void main(String[] args) {
      ArrayList<String> list=new ArrayList<String>();
      list.add("Mehul");
      list.add("Amit");
      list.add("Jitesh");
      list.add("Chirag");
      list.add("Raj");

      String s=list.get(1);//type casting is not required
      System.out.println("element is: "+s);
      Iterator<String> itr=list.iterator();

      while(itr.hasNext()){
         System.out.println(itr.next());
      }
   }
}
```

Output:

```
element is: Amit
Mehul
Amit
Jitesh
Chirag
Raj
```

AIM: Write a Java Program to demonstrate Generic Methods.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package genreic;
import java.util.ArrayList;
import java.util.Iterator;
/**
```

```
 *
 * @author lenovo
 */
public class Genreic {

    /**
     * @param args the command line arguments
     */
public static void main(String[] args) {

    // initialize the class with Integer data
    DemoClass demo = new DemoClass();

    // generics method working with String
    demo.<String>genericsMethod("Python Programming");

    // generics method working with integer
    demo.<Integer>genericsMethod(50);
  }
}

class DemoClass {

  // creae a generics method
  public <T> void genericsMethod(T data) {
    System.out.println("Generics Method:");
    System.out.println("Data Passed: " + data);
  }
}
```

Output:

```
Generics Method:
Data Passed: Python Programming
Generics Method:
Data Passed: 50
```

Practical 2

AIM: Write a Java program to create a Set containing list of items of type String and print the items in the list using Iterator interface. Also print the list in reverse/ backword direction.

```
package listD;
import java.util.*;

public class listD{
 public static void main(String[] args)
    {
       List<String> list = new ArrayList<String>();
       list.add("C");
       list.add("C++");
       list.add("Java");
       for(String i:list )
       // 1. Print string representation of the list using `toString()`
       System.out.println(i);
    }
}
```

Output:

```
C

C++

Java
```

AIM: Write a Java program to create List containing list of items and use ListIterator interface to print items present in the list. Also print the list in reverse/ backward direction.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package listD;
import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;
public class listOperations {
   public static void main(String a[]){
    ListIterator<String> litr = null;
    List<String> names = new ArrayList<String>();
    names.add("Amit");
    names.add("Jitesh");
    names.add("Megha");
    names.add("Devendra");
    names.add("Kajol");
    //Obtaining list iterator
    litr=names.listIterator();

    System.out.println("Traversing the list in forward direction:");
    while(litr.hasNext()){
      System.out.println(litr.next());
    }
    System.out.println("\nTraversing the list in backward direction:");
    while(litr.hasPrevious()){
      System.out.println(litr.previous());
    }
  }
}
```

Output:

```
Traversing the list in forward direction:
Amit
Jitesh
Megha
Devendra
Kajol

Traversing the list in backward direction:
Kajol
Devendra
Megha
Jitesh
Amit
```

Practical 3

AIM: Write a Java program to create a Set containing list of items of type String and print the items in the list using Iterator interface. Also print the list in reverse/ backword direction.

```
import java.util.*;
package listD;
import java.util.*;
public class listD {
 public static void main(String args[])    {
        List<Integer> lstData = new ArrayList<Integer>();
        lstData.add(150);
        lstData.add(78);
        lstData.add(1000);
        lstData.add(23);
        lstData.add(450);

        System.out.println(lstData);
    }
}
```

Output

```
[150, 75, 1000, 23, 450]
```

Practical 4

AIM: Write a Java program using Map interface containing list of items having keys and associated values

```
import java.util.*;
class MapDemo {
public static void main(String args[])
   {
      Map<String, Integer> mapObj = new HashMap<String, Integer>();

      // Inserting pairs in above Map
      // using put() method
      mapObj.put("a", new Integer(100));
      mapObj.put("b", new Integer(200));
      mapObj.put("c", new Integer(300));
      mapObj.put("d", new Integer(400));

      // Traversing through Map using for-each loop
      for (Map.Entry<String, Integer> me :
          mapObj.entrySet()) {

         // Printing keys
         System.out.print(me.getKey() + ":");
         System.out.println(me.getValue());
      }
   }
}
```

Output:

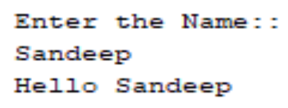
```
a:100

b:200

c:300

d:400
```

Practical 5

AIM: Write a Java program using Lambda Expression to print ”Hello World”.

```
package lamda;
interface HelloWorld {
            String sayHello(String name);
}
public class Lamda {

   /**
    * @param args the command line arguments
    */
 public static void main(String args[]) throws Exception
            {
          String name;
          Scanner scObj = new Scanner(System.in);
          System.out.println("Enter the Name::");
          name = scObj.next();
          HelloWorld helloWorld = (String) -> { return "Hello " + name; };
                         System.out.println(helloWorld.sayHello(name));
            }

}
```

Output:



```
Enter the Name::
Sandeep
Hello Sandeep
```

AIM: Write a Java program using Lambda Expression with multiple parameters to add two numbers.

```
package lamda;
import java.util.function.*;
public class Lamda {
public static void main(String args[]) {
     Message m = new Message();
     m.printStringInteger("Nikhil", 43,
               (String str, Integer number) -> {     // multiple parameters lambda
                 System.out.println("The values are: "+ str + " "+ number);
               });
     }
  private static class Message {
    public void printStringInteger(String str, Integer number, BiConsumer<String, Integer>
biConsumber) {
      biConsumber.accept(str, number);
    }
  }
}
```

Output:

```
The values are: Nikhil 43
```

Practical 6

AIM: Create a Telephone directory using JSP and store all the information within a database, so that later could be retrieved as per the requirement. Make your own assumptions.

Create table phone

```
create table phone(
name varchar2(20),
phone varchar2(15));
```

customer.jsp

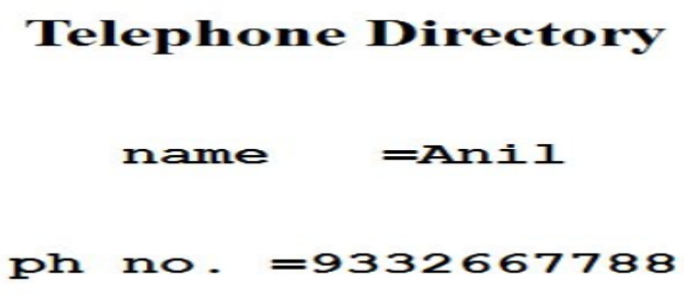
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">

<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>JSP Page</title>
   </head>
   <body><h1>
      <center>Enter Information<br>
      <form action=" telephone.jsp ">
         Enter the name :: <input type="text" name="n" >
            <br><br><input type="submit" value="Submit">
      </form>
   </center></h1>
   </body>
</html>
```

telephone.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!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=UTF-8">
      <title>JSP Page</title>
   </head>
   <body>
      <center>
```

```
    <h1>Telephone Directory</h1>
    <%@ page language="java" %>
    <%@ page import="java.sql.*" %>
     <%@ page import="java.sql.DriverManager.*" %>

    <%
    String m=null, mn=null;
    String s=request.getParameter("n");
    PreparedStatement ps=null;
    Connection con= null;
    ResultSet rs= null;
   Class.forName("oracle.jdbc.driver.OracleDriver");
   con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","root");
    Statement st=con.createStatement();
     rs=st.executeQuery("select * from phone where name='"+ s+"' ");
      while(rs.next())
          {
           m=rs.getString(1);
           mn=rs.getString(2);
       }
       out.println("<br>");

        out.println("<h1>");
        out.println("<pre>");
      out.println("name   =" +m);
     out.println("<br>");
      out.println("ph no. =" +mn);
       out.println("</pre>");
        out.println("</h1>");
    %>
    </center>
  </body>
</html>
```

Output:



```
Telephone Directory

name    =Anil

ph no. =9332667788
```

Practical 7

AIM: Write a program to demonstrate dependency injection via setter method.

To inject primitive and string-based values bythe setter method. We have created three files here:

- **Student.java**
- **applicationContext.xml**
- **Test.java**

Student.java

```
package com.MyPackage;
public class Student
{
    private int id;
    private String name;
    private String city;

    public int getId()
    {
            return id;
    }
    public void setId(int id)
    {
            this.id = id;

    }
    public String getName()
    {
            return name;
    }
    public void setName(String name)
    {
            this.name = name;
    }
    public String getCity()
    {
            return city;
    }
    public void setCity(String city)
    {
            this.city = city;
    }
    void display()
    {
        System.out.println("Student ID="+id+" "+"StudentName="+name+" "+" City="+city);
        }
}
```

applicationContext.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:p="http://www.springframework.org/schema/p"
   xsi:schemaLocation="http://www.springframework.org/schema/
   beans
      http://www.springframework.org/schema/beans/spring-beans-
   3.0.xsd">
   <bean id="obj" class="com.MyPackage.Student">
   <property name="id">
   <value>20</value>
   </property>

   <property name="name">
   <value>Umesh</value>
   </property>

   <property name="city">
   <value>Pune</value>
   </property>
   </bean>
</beans>
```

Test.java

```
package com.MyPackage;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.*;
public class Test{
        public static void main(String[] args){
           Resource r=new ClassPathResource("applicationContext.xml");
           BeanFactory factory=new
           XmlBeanFactory(r);Student
           s=(Student)factory.getBean("obj");
           s.display();
        }}
```

Output:

```
Student ID=20  Student Name=Umesh   City=Pune
```

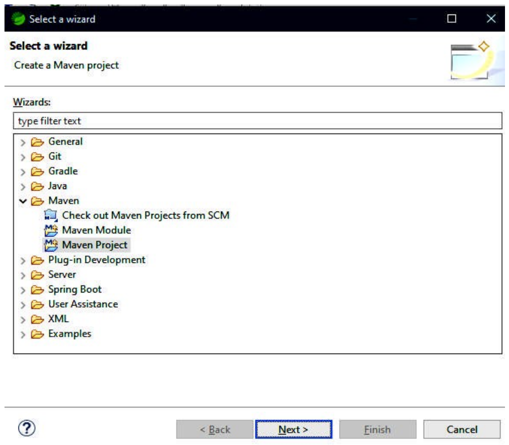
Practical 8

AIM: Write a program to demonstrate Spring AOP – before advice, after advice and around advice

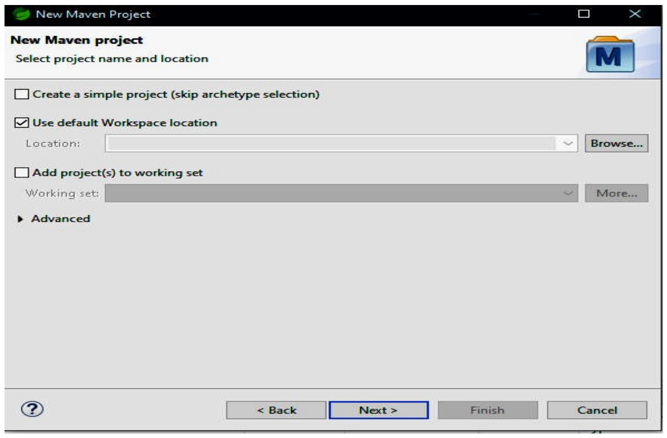
Step 1: Project Creation

- Open Spring Tool Suite

- In Spring Tools Suite IDE, go to File -> New ->Other ->Maven Project

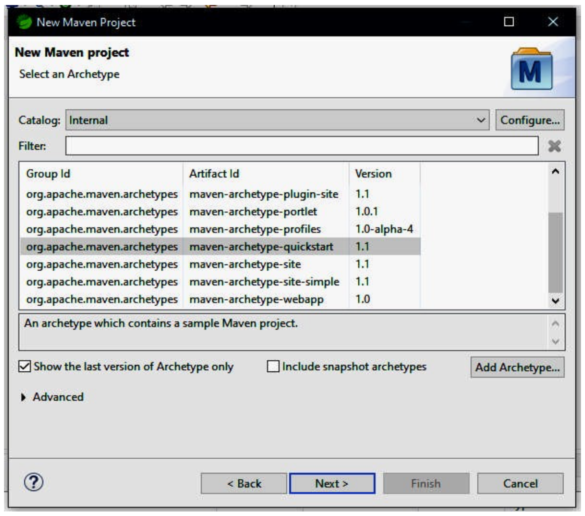


- The New Maven Project window will prompt you to select a project location. You may select the 'Use default workspace location' (by default selected) or may choose another project location to store


New Maven Project
New Maven project
Select project name and location
Create a simple project (skip archetype selection)
Use default Workspace location
Location:
Browse...
Add project(s) to working set
Working set:
More...
Advanced
< Back
Next >
Finish
Cancel

the project. Also uncheck the 'Create a simple project (skip archetype selection)' checkbox and click on the next button to proceed.

➢ Select maven-archetype-quickstart and click Next


New Maven Project

New Maven project

Select an Archetype

Catalog: Internal

Filter:

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-plugin-site	1.1
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1
org.apache.maven.archetypes	maven-archetype-site-simple	1.1
org.apache.maven.archetypes	maven-archetype-webapp	1.0

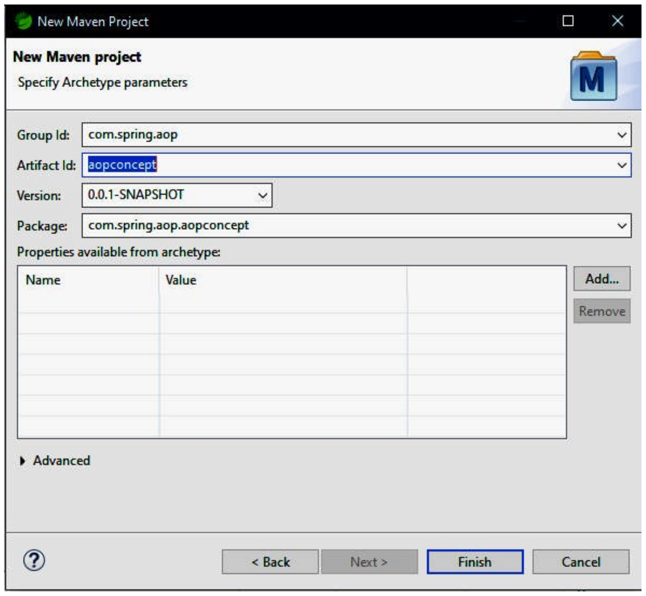
An archetype which contains a sample Maven project.

[✓] Show the last version of Archetype only   [ ] Include snapshot archetypes

➢ In the next window enter the group and artifact ID for the project as follows:

Group ID: com.spring.aop

Artifact ID: aopconcept


New Maven Project

New Maven project

Specify Archetype parameters

Group Id: com.spring.aop

Artifact Id: aopconcept

Version: 0.0.1-SNAPSHOT

Package: com.spring.aop.aopconcept

Properties available from archetype:

Name	Value

➢ Click on Finish and a maven project will be created. All the maven dependencies will be created and an XML file pom.xml will be created.

The contents of pom.xml will be as follows:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema- instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0. 0
http://maven.apache.   org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.spring.aop</groupId>
<artifactId>aopconcept</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>
<name>aopconcept</name>
<url>http://maven.apache.org</url>
<properties>
<project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
</properties>
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
</dependencies>
</project>
```

Step 2: Maven Dependencies

Add the following dependencies in pom.xml inside the <dependencies> element

```
<!-- Spring Core Dependency-->
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-core</artifactId>
<version>5.3.5</version>
</dependency>
<!-- Spring Context Dependency-->
```

```
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-context</artifactId>
<version>5.3.5</version>
</dependency>

<!-- Spring AOP Dependency-->
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-aop</artifactId>
<version>5.3.5</version>
</dependency>
<!-- AspectJ Runtime Dependency-->
<dependency>
<groupId>org.aspectj</groupId>
<artifactId>aspectjrt</artifactId>
<version>1.8.6</version>
</dependency>

<!-- AspectJ Weaver Dependency-->
<dependency>
<groupId>org.aspectj</groupId>
<artifactId>aspectjweaver</artifactId>
<version>1.8.6</version>
</dependency>
```

Update Project by Right Clicking on PackageName->Maven->Update Project

Step 3: JAVA Classes & XML File

- **Customer Model Class (Customer.java)**

```
package com.spring.aop.aopconcept;
import org.springframework.stereotype.Component;
@Component
public class Customer {
```

```
int cust_id;
String cust_name;11
String cust_loc;
public

Customer(){}

          public Customer(int cust_id, String cust_name,
Stringcust_loc) {
                    super();
                    this.cust_id = cust_id;
                    this.cust_name =
                    cust_name;
                    this.cust_loc =
                    cust_loc;
          }
          public int getCust_id() {
                    return cust_id;
          }
          public void setCust_id(int
                    cust_id) {this.cust_id
                    = cust_id;
          }
          public String
                    getCust_name()
                    {return
                    cust_name;
          }
          public void setCust_name(String cust_name)
                    {this.cust_name = cust_name;
          }
          public String
                    getCust_loc() {
                    return
                    cust_loc;
          }
          public void setCust_loc(String cust_loc) {this.cust_loc = cust_loc;
          }

          @Override
          public String toString() {
                    return "Customer [cust_id=" + cust_id + ",
cust_name=" + cust_name + ", cust_loc=" + cust_loc + "]";
```

```
                }
                public void display()
                {
                        System.out.println(this.toString());
                }
        }
```

➢ **Customer Aspect Class (CustomerAspect.java)**

```
package  com.spring.aop.aopconcept;
import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Around;

import org.springframework.stereotype.Component

@Aspect

@Component

        public class CustomerAspect {

                @Before("execution(*
        com.aop.aopconcept.Customer.display(..))")

                void beforeAdvice() {

                        System.out.println("This  method  is  called
        beforethe display method");

                }

                @After("execution(*
        com.aop.aopconcept.Customer.display(..))")

                void afterAdvice() {

                        System.out.println("This  method  is  called
        afterthe display method");

                }

                @Around("execution(*
        com.aop.aopconcept.Customer.display(..))")

                void   aroundAdvice(ProceedingJoinPoint  pjp)
                       throwsThrowable {

                 System.out.println("..................................................................");pjp.proceed();

                }

        }
```

- **Create a XML file, beans.xml**

<u>beans.xml</u>

```
<?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:aspectj-autoproxy/>

        <!-- - Definition for customer bean -->
        <bean id = "cust" class =
        "com.spring.aop.aopconcept.Customer">
        <property name = "cust_id" value = "001" />
        <property name = "cust_name"  value = "Vipul"/>
        <property name = "cust_loc"  value = "Mumbai"/>
        </bean>
        <!-- - Definition for Customer Aspect -->
        <bean id = "custAspect" class =
        "com.spring.aop.aopconcept.CustomerAspect"/>
        </beans>
```

- **<u>Implementation Class (App.java)</u>**

```
package com.spring.aop.aopconcept;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplication
Context;
public class App
        {
           public static void main( String[] args )
           {
            ApplicationContext
                  context =new
    ClassPathXmlApplicationContext("beans.xml");

            Customer cust = (Customer) context.getBean("cust");
               cust.display();
       }
    }
```

Step 4: Run the Program

Run App.java as **Java Application**.**Output**

```
This method is called before the display method
Customer [cust_id=1, cust_name=Vipul,
cust_loc=Mumbai]This method is called after the
display method
```

AIM: Write a program to demonstrate Spring AOP – after returning advice, after throwing advice and pointcuts

We will define a MathOp class with methods add(), sub(), prod() and mul() to perform mathematical operations.

Next we will define MathOpAspect class where we shall define pointcuts and the methods to be called after returning advice and after throwing advice

Repeat Step 1 & Step 2 as per Program 1

Step 3: JAVA Classes & XML File

- **MathOp Class (MathOp.java)**

```
package com.spring.aop.aopconcept;
import org.springframework.stereotype.Component;
@Component
public class MathOp {
        int add(int a,int b)
        {
                return a+b;
        }
        int sub(int a,int b)
        {
                return a-b;
        }
        int prod(int a,int b)
        {
                return a*b;
        }
        int div(int a,int b) throws Exception
        {
                return a/b;
        }
        }
```

- **MathOp Aspect Class (MathOpAspect.java)**

```
package   com.spring.aop.aopconcept;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.stereotype.Component;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
```

```
@Aspect
@Component

public class MathOpAspect {
        @Pointcut("execution(*
                com.aop.aopconcept.MathOp.*(..))")void
                callMe() {}
                @AfterReturning("callM
                e()")void
                afterReturning()
                {
        System.out.println("Operation successfully completed");
                }

                @AfterThrowing("callM
                e()")void
                afterThrowing()
                {
        System.out.println("An error has occured during the
        execution ofthe program");
                }
                }
```

- **Create a XML file, beans.xml**

  <u>beans.xml</u>

```
<?xml version = "1.0" encoding = "UTF-8"?>
        <beans xmlns =
        "http://www.springframework.org/schema/bean
        s"
          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/spri
          ng-aop-
        3.0.xsd ">
        <aop:aspectj-autoproxy/>
        <bean id = "mo" class = "com.aops.aopconcepts.MathOp"/>
        <bean id = "moa" class =
        "com.aops.aopconcepts.MathOpAspect"/>
        </beans>
```

➢ **Implementation Class (App.java)**

```
    package com.spring.aop.aopconcept;
    import org.springframework.context.ApplicationContext;
    import org.springframework.context.support.ClassPathXmlApplicationContext;
    public class App
    {
       public static void main( String[]
    args )ApplicationContext context =
             new
    ClassPathXmlApplicationContext("com/aop/aopconcept/beans.xml");

             MathOp mo = (MathOp)
             context.getBean("mo");
             System.out.println("Sum is "+mo.add(4,3));
             System.out.println("Diffference is
             "+mo.sub(5,2));System.out.println("Product
             is  "+mo.prod(2,1)); try
             {
               System.out.println("Division is "+mo.div(3,0));
             }
             catch(Exception e)
             {}
        }
```

Step 4: Run the Program

Run App.java as **Java**

Application.**Output**

```
Operation successfully completed
Sum is 7
Operation successfully completed
Diffference is 3
Operation successfully
completedProduct is 2
```

Practical 9

AIM: Write a program to insert, update and delete records from the given table

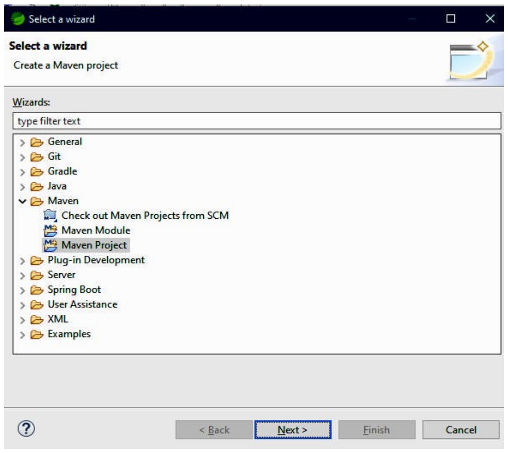
Step 1: MySQL database

```
Create database idol;
CREATE TABLE Customer (
ID INT NOT NULL AUTO_INCREMENT,
NAME VARCHAR(20) NOT NULL,
AGE INT NOT NULL,
PRIMARY KEY (ID));
```

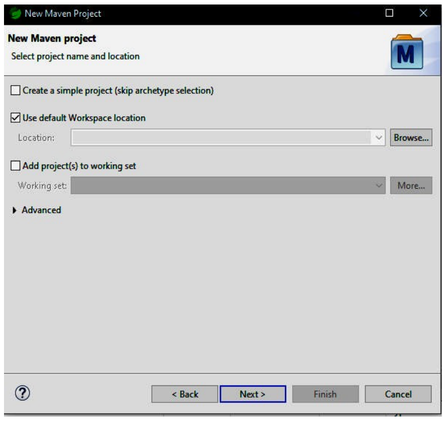
Step 2: Project Creation

- Open Spring Tool Suite

- In Spring Tools Suite IDE, go to File -> New ->Other -> Maven Project

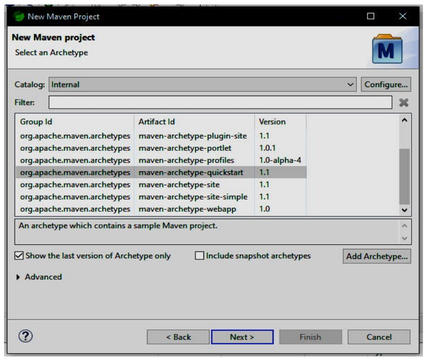


- The New Maven Project window will prompt you to select a project location. You may select the 'Use default workspace location' (by default selected) or may choose another project location to store the project. Also uncheck the 'Create a simple project (skip archetype selection)' checkbox and click on the next button to proceed.



[ ] Create a simple project (skip archetype selection)
[✓] Use default Workspace location
Location:
[ ] Add project(s) to working set
Working set:
Advanced

- Select maven-archetype-quickstart and click Next



Catalog: Internal
Filter:

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-plugin-site	1.1
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1
org.apache.maven.archetypes	maven-archetype-site-simple	1.1
org.apache.maven.archetypes	maven-archetype-webapp	1.0

An archetype which contains a sample Maven project.

[✓] Show the last version of Archetype only
[ ] Include snapshot archetypes
Advanced

- In the next window enter the group and artifact ID for the project as follows:

  Group ID: com.jdbc.springjdbc
  Artifact ID: jdbc

- Click on Finish and a maven project will be created. All the maven dependencies will be created and an XML file pom.xml will be created.

The contents of pom.xml will be as follows:

```
<project      xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema- instance"

        xsi:schemaLocation="http://maven.apache.org/P
        OM/4.0. 0 http://maven.apache.org/xsd/maven-
        4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.spring.aop</groupId>
        <artifactId>aopconcept</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>jar</packaging>
        <name>aopconcept</name>
        <url>http://maven.apache.org</url>
        <properties>
        <project.build.sourceEncoding>UTF-
        8</project.build.sourceEncoding>
```

```
                </properties>
                <dependencies>
                <dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>3.8.1</version>
                <scope>test</scope>
                </dependency>
                </dependencies>
        </project>
```

Step 3: Maven Dependencies

- Add the following dependencies in pom.xml inside the <dependencies> element

```
        <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>4.1.0.RELEASE</version>

        </dependency>
        <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.1.4.RELEASE</version>
        </dependency>
        <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-jdbc</artifactId>
                <version>4.1.4.RELEASE</version>
                </dependency>
                <dependency>
                <groupId>mysql</groupId>
                <artifactId>mysql-connector-java</artifactId>
                <version>5.0.5</version>
        </dependency>
```

Update Project by Right Clicking on PackageName->Maven->Update Project

Step 4: JAVA Classes & XML File

- **<u>Customer Model Class (Customer.java)</u>**

```
    package com.jdbc.springjdbc;
     public class Customer {
            private Integer age;
            private String name;
            private Integer id;
```

```
public void setAge(Integer
age) {
        this.age = age;
}
          public Integer
            getAge() {return
            age;
          }
          public void setName(String name)
            {this.name = name;
          }
          public String
            getName() {
            return name;
          }
          public void
            setId(Integer id) {
            this.id = id;
          }
          public Integer
            getId() {
                return id;
          }
}
```

- **CustomerJDBCTemplateClass (CustomerJDBCTemplate.java)**

```
package com.jdbc.springjdbc;
import javax.sql.DataSource;
import org.springframework.jdbc.core.JdbcTemplate;
public class CustomerJDBCTemplate
{
        private DataSource dataSource;
        private JdbcTemplate jdbcTemplateObject;
        public void setDataSource(DataSource dataSource)
        {
                this.dataSource = dataSource;
                this.jdbcTemplateObject = new JdbcTemplate(dataSource);
        }
        public void insert(String name, Integer age)
        {
          String SQL = "insert into Customer (name,age)
      values('"+name+"','"+age+"')";
          jdbcTemplateObject.update(  SQL);
          System.out.println("Created Record Name = " + name +
          " Age= " + age);
           return;
        }
        public void delete(int id)
        {
          String SQL = "delete from Customer where id='"+id+"'";
          int n=jdbcTemplateObject.update( SQL);
          if(n>0)
          {
          System.out.println("Deleted Customer with ID = " + id);
          }
          return;
        }
         public void updateAge(Integer age, Integer id)
        {
              String SQL = "update Customer age='"+age+"' where
      id='"+id+"'";
             jdbcTemplateObject.update( SQL);
              System.out.println("Record
              Updated");return;
          }
}
```

- **Create a XML file, beans.xml**

<u>beans.xml</u>

```
<?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 ">

                <!-- Initialization for data source -->
                <bean id =
                    "dataSource"
                    class =
                "org.springframework.jdbc.datasource.DriverManagerDat
                aSourc e">
                <property name = "driverClassName" value =
                "com.mysql.jdbc.Driver"/>
                <property name = "url" value =
                "jdbc:mysql://localhost:3306/idol"/>
                <property name = "username" value = "root"/>
                <property name = "password" value = "rdnc"/>
                </bean>

                <!-- Definition for CustomerJDBCTemplate bean -->
                <bean id = "customerJDBCTemplate"
                    class = "com.jdbc.springjdbc.CustomerJDBCTemplate">
                <property name = "dataSource" ref = "dataSource" />
                </bean>
</beans>
```

- **Implementation Class (App.java)**

```
        package com.jdbc.springjdbc;
        import java.util.List;
        import
        org.springframework.context.ApplicationContext;
        import org.springframework.context.support.ClassPathXmlApplication
Context;
        import com.jdbc.springjdbc.CustomerJDBCTemplate;
        public class App {
                public static void main(String[] args) {
                      ApplicationContext context = new
                ClassPathXmlApplicationContext("Beans.xml");
                CustomerJDBCTemplate customerJDBCTemplate =(CustomerJDBCTemplate)
                context.getBean("customerJDBCTemplate");
                System.out.println("------Records Creation -------------" );
                customerJDBCTemplate.insert("Rahul", 25);
                customerJDBCTemplate.insert("Raj", 40);
                customerJDBCTemplate.insert("Manpreet", 29);
                customerJDBCTemplate.delete(3);
               customerJDBCTemplate.update(5,24);
                  }
    }
```

Step 5: Run the Program

Run App.java as **Java Application**.

Output

```
------Records Creation--------
Created Record Name = Rahul Age = 25
Created Record Name = Raj Age = 40
Created Record Name = Manpreet Age = 29
Deleted Cutomer with ID = 3
Record Updated
```

Aim: Write a program to demonstrate PreparedStatement in SpringJdbcTemplate

We will create a Customer table with column ID, Name and Age in MySQL

Next we will write a program to insert values into the Customer table using PreparedStatement in Spring JDBC

Repeat Step 1, Step 2, and Step 3 and create Customer Model Classof Step 4 as per Program 1

Step 4: JAVA Classes & XML File

- **CustomerJDBCTemplate1Class**

```
package com.jdbc.springjdbc;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.List;
import javax.sql.DataSource;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementSetter;

public class CustomerDBCTemplate1
        {
          private DataSource dataSource;
          private JdbcTemplate jdbcTemplateObject;
          public void setDataSource(DataSource dataSource)
        {
            this.dataSource = dataSource;
            this.jdbcTemplateObject = new JdbcTemplate(dataSource);
          }
          public void insert(final String name, final int age) {
             final String SQL = "insert into customer (name,age) values(?,?) ";
             jdbcTemplateObject.update(SQL, new PreparedStatementSetter() {
               public void setValues(PreparedStatement ps)throws
        SQLException
               {
                 ps.setString(1,name);
                 ps.setString(2,age);
         }

       });

             System.out.println("Record Created for customer "+name);

          }
         }
```

- **Create a XML 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"
 xsi:schemaLocation =
        "http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-
        beans- 3.0.xsd ">
        <!-- Initialization for data source -->
        <bean id ="dataSource"class =
        "org.springframework.jdbc.datasource.DriverManagerDataSourc e">
        <property name = "driverClassName" value =
        "com.mysql.jdbc.Driver"/>
        <property name = "url" value =
        "jdbc:mysql://localhost:3306/idol"/>
        <property name = "username" value = "root"/>
        <property name = "password" value = "rdnc"/>
        </bean>
        <bean id = "customerJDBCTemplate1"
            class = "com.jdbc.springjdbc.CustomerJDBCTemplate1">
        <property name = "dataSource" ref = "dataSource" />
        </bean>
</beans>
```

- **Implementation Class (App.java)**

```
package com.jdbc.springjdbc;
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplication Context;
import com.jdbc.springjdbc.CustomerJDBCTemplate;
    public class App {
      public static void main(String[] args) {
        ApplicationContext context = new
    ClassPathXmlApplicationContext("Beans.xml");
    CustomerJDBCTemplate1customerJDBCTemplate1 =
    (CustomerJDBCTemplate1)
        context.getBean("customerJDBCTemplate3");
      System.out.println("------Records Creation----- " );
    customerJDBCTemplate1.insert("Vipul", 40);
    customerJDBCTemplate1.insert("Jatin", 37);
```

```
            customerJDBCTemplate1.insert("Harpreet", 39);
              }
            }
```

Step 5: Run the Program

Run App.java as **Java Application**.

Output

```
------Records Creation--------
Created Record Name = Rahul
Created Record Name = Raj
Created Record Name = Manpreet
```

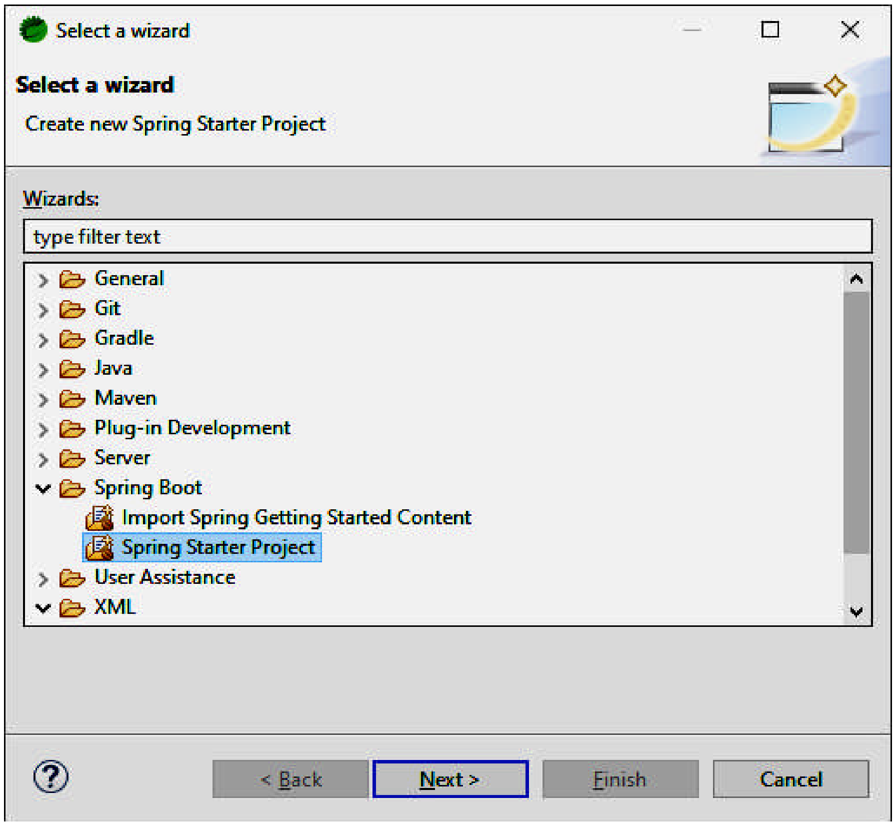
Practical 10

AIM: Write a program to create a simple Spring Boot application that prints a message

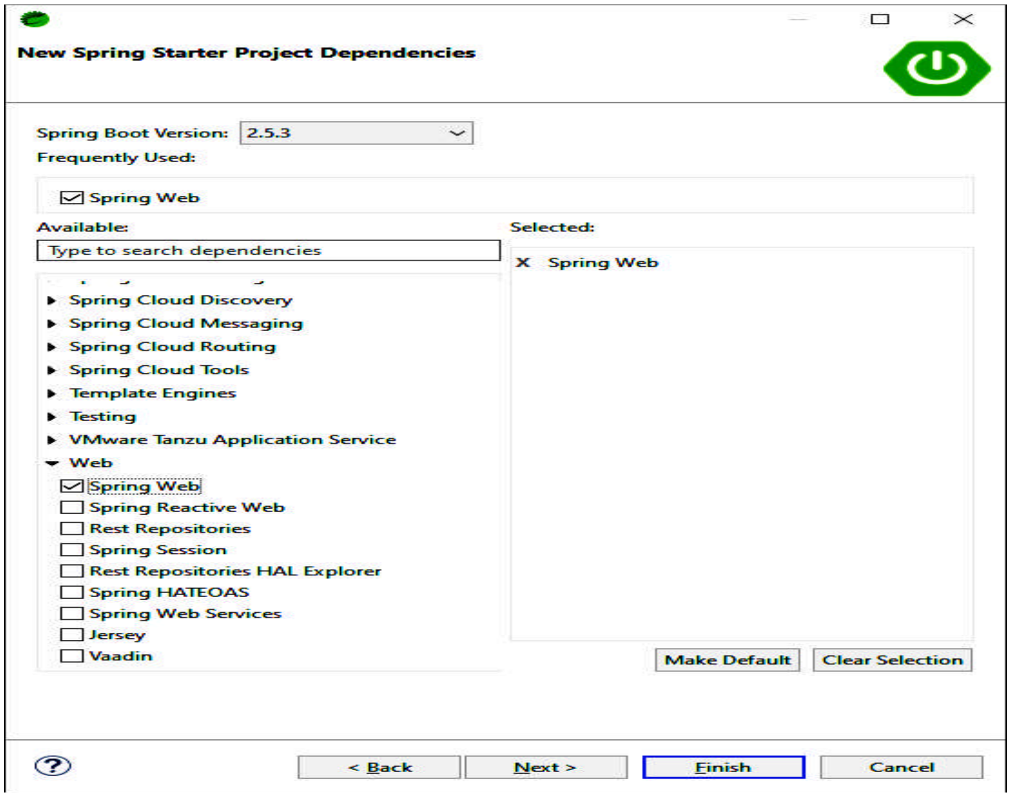
Step 1: Project Creation

- Open Spring Tool Suite

- In Spring Tools Suite IDE, go to File -> New ->Other -> Spring Boot -> Spring Starter Project



- In the next window enter the Name as **FirstBoot**
- In the next window Select Web->Spring Web as shown in the figure below:



New Spring Starter Project Dependencies

Spring Boot Version: 2.5.3

Frequently Used:

☑ Spring Web

Available: Type to search dependencies

- ▸ Spring Cloud Discovery
- ▸ Spring Cloud Messaging
- ▸ Spring Cloud Routing
- ▸ Spring Cloud Tools
- ▸ Template Engines
- ▸ Testing
- ▸ VMware Tanzu Application Service
- ▾ Web
  - ☑ Spring Web
  - ☐ Spring Reactive Web
  - ☐ Rest Repositories
  - ☐ Spring Session
  - ☐ Rest Repositories HAL Explorer
  - ☐ Spring HATEOAS
  - ☐ Spring Web Services
  - ☐ Jersey
  - ☐ Vaadin

Selected: X Spring Web

- Click on Finish and a maven project will be created. All the dependencies will be created and an XML file pom.xml will be created.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.
0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.5.3</version>
        <relativePath/><!-- lookup parent from repository -->
         </parent>
        <groupId>com.example</groupId>
         <artifactId>FirstBoot</artifactId>
         <version>0.0.1-SNAPSHOT</version>
        <name>FirstBoot</name>
         <description>Demo project for Spring Boot
         </description>
        <properties>
                <java.version>11</java.version>
```

```
                                        </properties>
                                       <dependencies>
                                               <dependency>
                                       <groupId>org.springframework.boot</groupId>
                                       <artifactId>spring-boot-starter-web</artifactId>
                                               </dependency>
                                                <dependency>
                                       <groupId>org.springframework.boot</groupId>
                                       <artifactId>spring-boot-starter-test</artifactId>
                                                       <scope>test</scope>
                                               </dependency>
                                        </dependencies>
                                       <build>
                                               <plugins>
                                                       <plugin>
                                        <groupId>org.springframework.boot</groupId>
                                       <artifactId>spring-boot-maven-plugin</artifactId>
                                                       </plugin>
                                               </plugins>
                                       </build>
                               </project>
```

Step 2: JAVA Classes

- **First BootApplication Class (Customer.java)**

```
package com.example;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication

public class FirstBootApplication {
          public static void main(String[] args) {
          SpringApplication.run(FirstBootApplication.class, args);
          }
  }
```

- **First Boot ControllerClass (FirstBootController.java)**

```
package com.example;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController

public class FirstBootController {
  @GetMapping("/hello")
  public String sayHello() {
  return "Hello to IDOL";
}}
```

- **Create a XML 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"
xsi:schemaLocation =
"http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans- 3.0.xsd ">
<!-- Initialization for data source -->
<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/idol"/>
<property name = "username" value = "root"/>
<property name = "password" value = "rdnc"/>
</bean>
<!-- Definition for CustomerJDBCTemplate bean -->
<bean id = "customerJDBCTemplate"
class = "com.jdbc.springjdbc.CustomerJDBCTemplate">
<property name = "dataSource" ref = "dataSource" />
</bean>
</beans>
```

- **Implementation Class (App.java)**

```
package com.jdbc.springjdbc;
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.jdbc.springjdbc.CustomerJDBCTemplate;
public class App {
        public static void main(String[] args) {
                ApplicationContext context = new ClassPathXmlApplicationContext
                                                ("Beans.xml");
                CustomerJDBCTemplate customerJDBCTemplate =
                                                (CustomerJDBCTemplate)
                context.getBean("customerJDBCTemplate");
                System.out.println("------Records Creation--------" );
                customerJDBCTemplate.insert("Vipul", 40);
                customerJDBCTemplate.insert("Jatin", 37);
                customerJDBCTemplate.insert("Harpreet", 39);
                customerJDBCTemplate.delete(3);
                customerJDBCTemplate.update(5,24);
        }
    }
```

Step 3: Run the Program

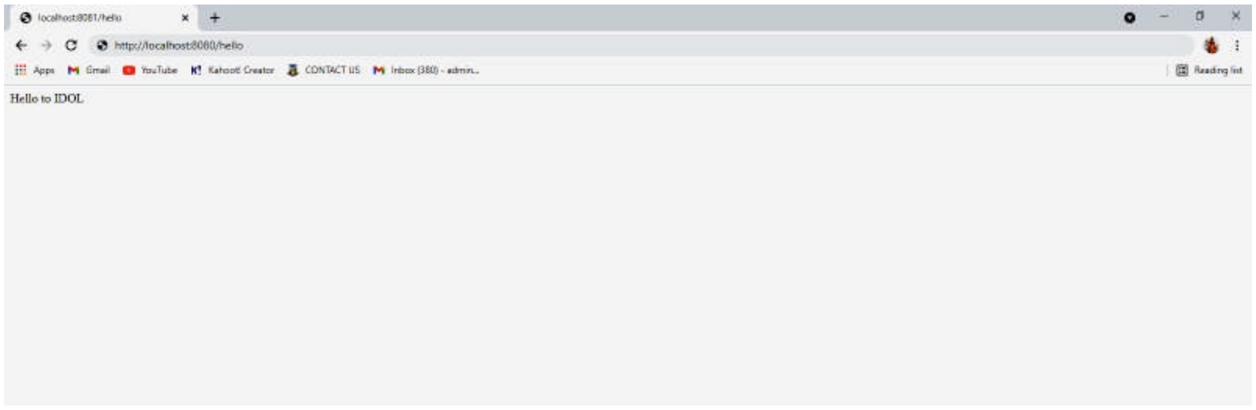
Run App.java as **Spring Boot App**.

Output

View the output in any Browser by typing the URL:

http://localhost:8080/hello

The output will be as follows:



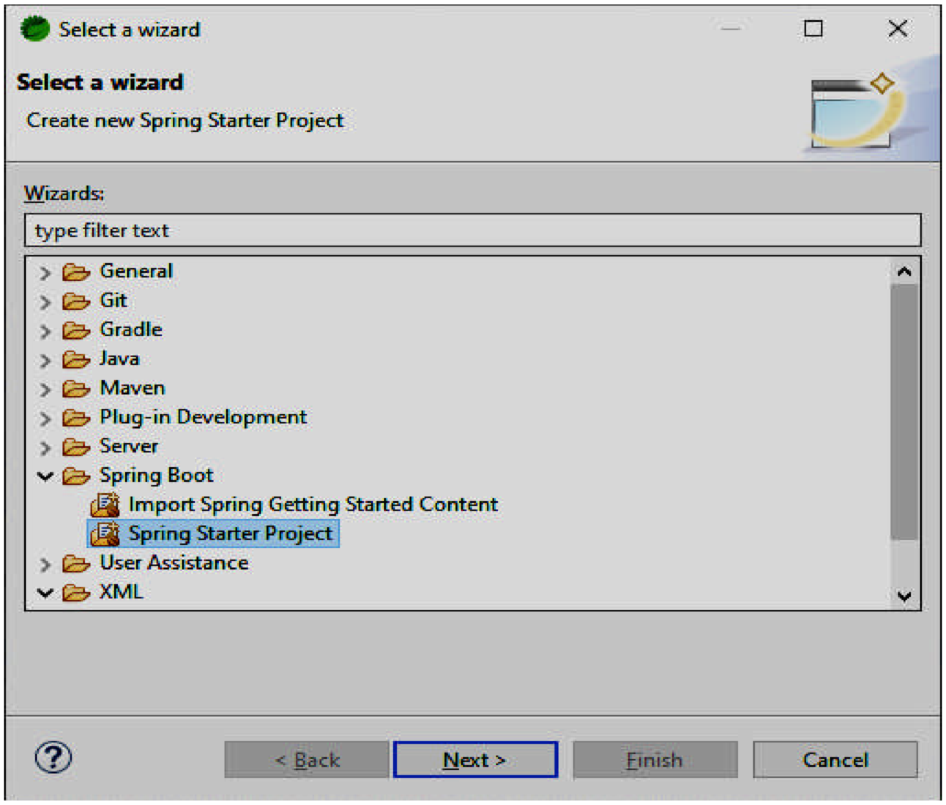
```
Hello to IDOL
```

Aim: Write a program to demonstrate RESTful Web Services with spring boot

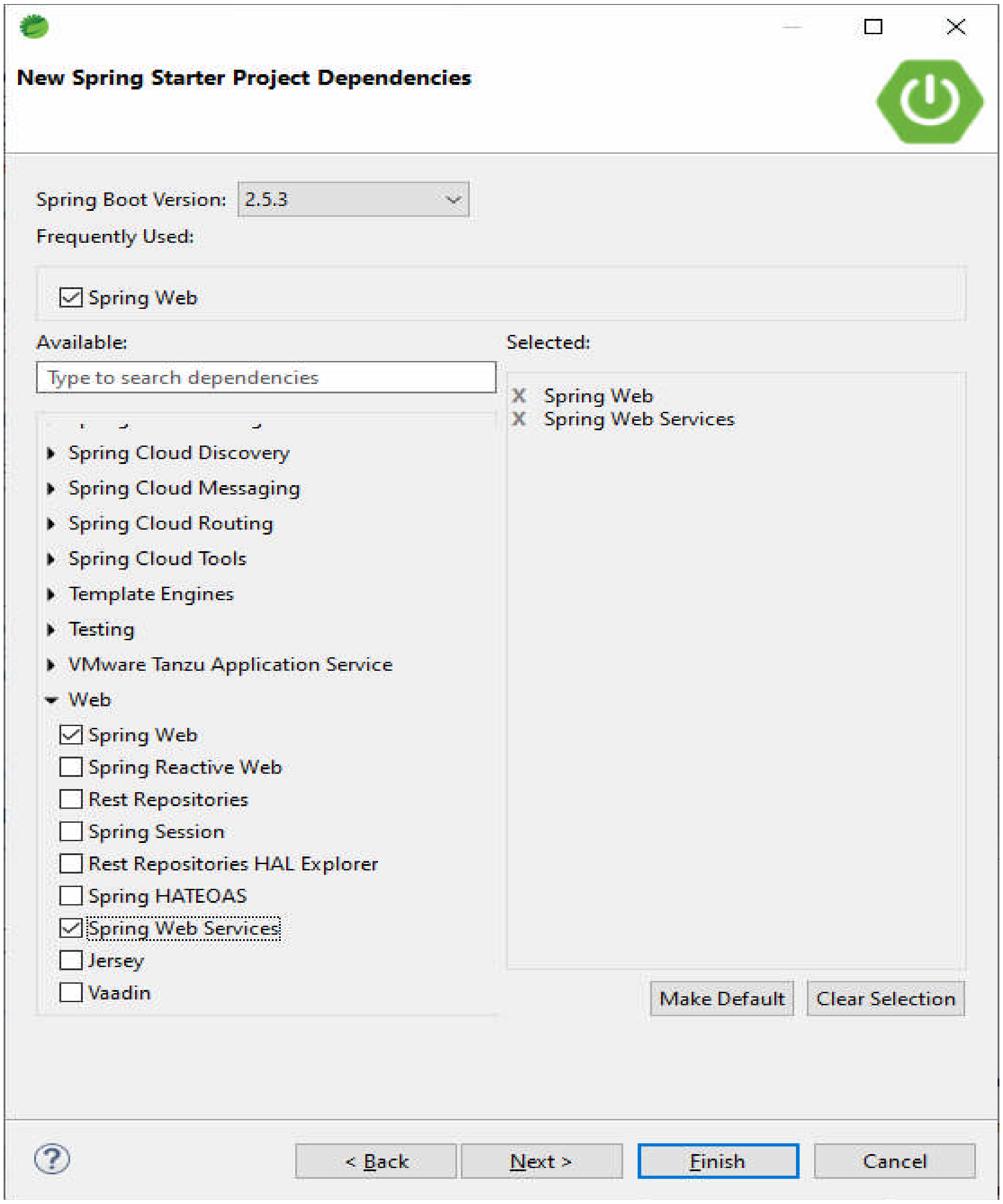
Step 1: Project Creation

- Open Spring Tool Suite

- In Spring Tools Suite IDE, go to File -> New ->Other -> Spring Boot -> Spring Starter Project



- In the next window enter the Name as **restful-webservice**
- In the next window Select Web->Spring Web and Spring Web Services as shown in the figure below:


New Spring Starter Project Dependencies

Spring Boot Version: 2.5.3

Frequently Used:

☑ Spring Web

Available:

Type to search dependencies

- Spring Cloud Discovery
- Spring Cloud Messaging
- Spring Cloud Routing
- Spring Cloud Tools
- Template Engines
- Testing
- VMware Tanzu Application Service
- Web
  - ☑ Spring Web
  - ☐ Spring Reactive Web
  - ☐ Rest Repositories
  - ☐ Spring Session
  - ☐ Rest Repositories HAL Explorer
  - ☐ Spring HATEOAS
  - ☑ Spring Web Services
  - ☐ Jersey
  - ☐ Vaadin

Selected:

X Spring Web
X Spring Web Services

➢ Click on Finish and a maven project will be created. All the dependencies will be created and an XML file pom.xml will be created.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.
0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
```

```
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.5.3</version>
<relativePath/><!-- lookup parent from repository -->
</parent>
<groupId>com.example</groupId>
<artifactId>restful-webservice</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>restful-webservice</name>
<description>Demo project for Spring Boot</description>
<properties>
<java.version>11</java.version>
</properties>
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web-services</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
  </dependency>
  </dependencies>
  <build>
  <plugins>
  <plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  </plugin>
  </plugins>
</build>
</project>
```

Step 2: JAVA Classes

- Create a Hello IDOL Service in the class HelloIDOLController

  HelloIDOLController.java

```
package com.example;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
```

```
            //Controller
    @RestController
    public class HelloIDOLController
    {
                    //using get method and hello-world as URI
            @RequestMapping(method=RequestMethod.GET, path="/helloworld")
            public String helloWorld()
            {
                    return "Hello IDOL";
            }
    }
```

- **RestfulWebserviceApplicationClass**

  (RestfulWebserviceApplication.java)

```
package com.example;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
 @SpringBootApplication
 public class RestfulWebserviceApplication {
 public static void main(String[] args) {
 SpringApplication.run(RestfulWebserviceApplication.class, args);
 }}
```

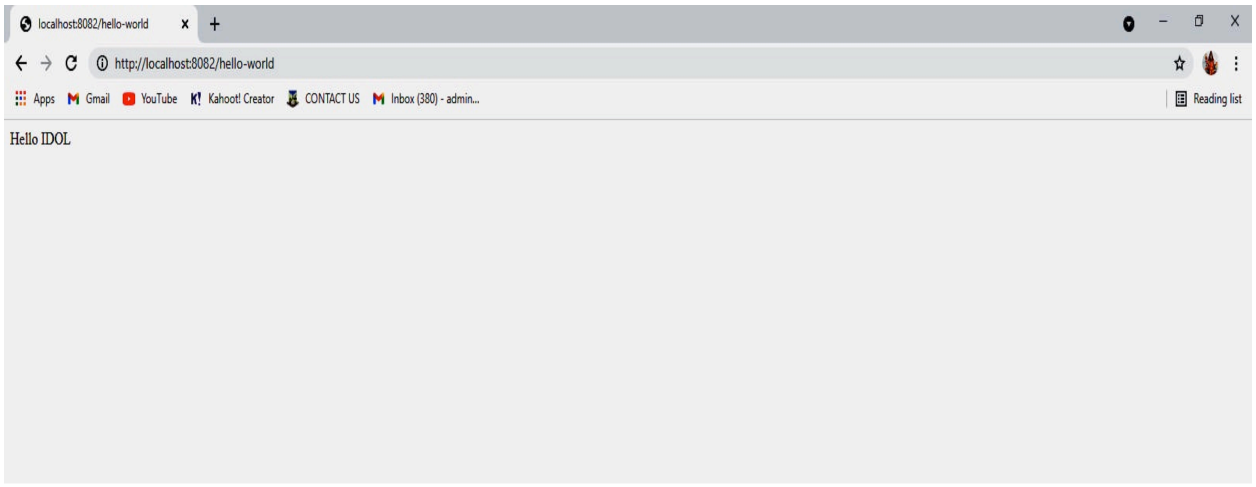
Step 3: Run the Program

Run App.java as **Spring Boot App**.

Output

View the output in any Browser by typing the URL:
http://localhost:8080/hello-world

The output will be as follows:



```
Hello IDOL
```