TASK 10
SPRING FRAMEWORK

10.1

Aim:-

To set up a spring MVC application that displays a welcome message to the user along with the ID, First name and Last name of the Employee from an existing EmployeeDb database utilizing Spring Tool Suite 4 IDE, Spring JAR Files, mySQL and Tomcat Apache latest version.

Algorithm:-

Step 1. Start

Step 2. Create a Dynamic Web Project 'SpringBootTrial' in Eclipse IDE

Step 3. Inside src > main > webapp > WEB-INF > lib folder of the project folder, paste the Spring Jars

Step 4. Configure Apache Tomcat Server and configure the Tomcat Server with the application

Step 5. Configure Dispatcher Servlet with the Spring MVC application by under src > main > webapp > WEB-INF > web.xml

Step 6. Create Spring Configuration file frontcontroller-dispatcher-servlet.xml under src > main > webapp > WEB-INF

Step 7. Create the Spring MVC Controller 'DemoController' under the package com.student.controllers

Step 8. Indicate that DemoController' is the controller class by using @Controller annotation

Step 9. Create displayWelcomeMessage() method inside the Controller class and use @RequestMapping and @ResponseBody annotation before the method

Step 10. Inside the displayWelcomeMessage() method, Open the existing Database connection EmployeeDB and retrieve the employee_ID,

FIRST_NAME and LAST_NAME of the last record from the employee table into Resultset using SQL select command

Step 11. The method returns the welcome string along with employee_ID, FIRST_NAME and LAST_NAME.

Step 12. Run the Spring MVC Application by right-clicking on the project > Run As > Run on Server and run the application

Step 13. Stop

Program:-

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
  <display-name>SpringBootTrial</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
   <absolute-ordering/>
<servlet>
      <!-- Provide a Servlet Name -->
    <servlet-name>frontcontroller-dispatcher</servlet-name>
    <!-- Provide a fully qualified path to the DispatcherServlet class -->
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
      <!-- Provide a Servlet Name that you want to map -->
    <servlet-name>frontcontroller-dispatcher</servlet-name>
    <!-- Provide a url pattern -->
    <url-pattern>/student.com/*</url-pattern>
  </servlet-mapping>

</web-app>
```

frontcontroller-dispatcher-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
             https://www.springframework.org/schema/beans/spring-beans.xsd
             http://www.springframework.org/schema/context
```

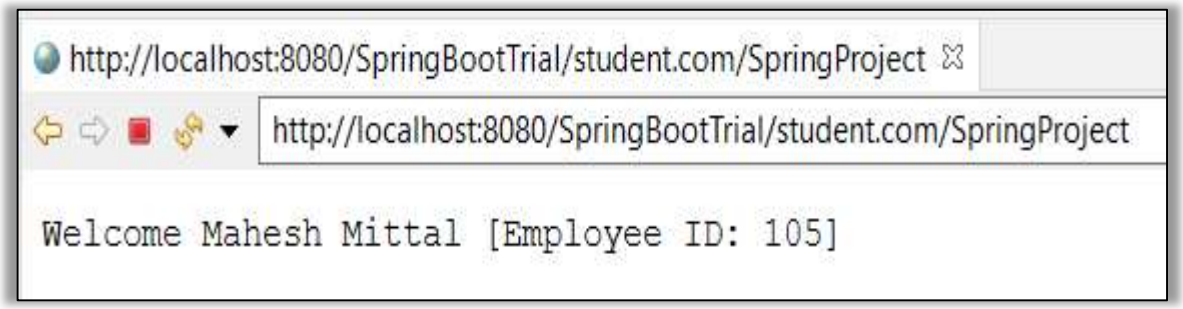
```
            https://www.springframework.org/schema/context/spring-context.xsd">
<context:component-scan base-
package="com.student.controllers"></context:component-scan>
</beans>
```

DemoController.java

```
// Java Program to Illustrate DemoController Class

package com.student.controllers;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

// Importing required classes
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

// Annotation
@Controller
// Class
public class DemoController {


      // Annotation
      @ResponseBody
      @RequestMapping("/SpringProject")

      // Method
      public String displayWelcomeMessage() throws SQLException,
ClassNotFoundException
      {

                Statement stmt=null;
                ResultSet rs=null;
                Connection conn= null;

            // Open a connection
                Class.forName("com.mysql.jdbc.Driver");
       conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/EmployeeDB", "root",
"admin");

        // Execute SQL query
        stmt = conn.createStatement();
        String sql;
        sql = "SELECT * FROM employee";
        rs = stmt.executeQuery(sql);
        int id = 0;
        String first = null,last = null;
```

```
        // Extract data from result set
        while(rs.next())
        {
              //Retrieve by column name
               id  = rs.getInt("employee_ID");
               first = rs.getString("FIRST_NAME");
               last = rs.getString("LAST_NAME");

        }

        // Clean-up environment
        rs.close();
        stmt.close();
        conn.close();
        return "Welcome"+" "+first+" "+last+" [Employee ID: "+id+"]";
  }
      }
```

Output:-



Result:-

Developed a spring MVC application that displayed a welcome message to the user along with Employee details utilizing Spring Tool Suite 4 IDE, Spring JAR Files, mySQL and Tomcat Apache latest version.

10.2

Aim:-

To Demonstrate a Spring Standalone application by creating a basic simple Java Project, then including Spring Framework and common logging API libraries, actual source files that displays a console-based text to the user. Also to have the Spring Beans Configuration performed in a Bean class with class-level annotation - @Configuration in order to display the console text to the user.

Algorithm:-

Step 1. Start

Step 2. Create Java Bean Class containing message property only with its getters and setters method

Step 3. Create a Spring Bean using @Bean Annotation which specifies that the annotation applied on displayWelcomeMessage() method returns a bean to be managed by Spring context

Step 4. Create the test class - ApplicationClass.java which gets the object of WelcomeMessage class from the IOC container using the getBean() method of BeanFactory

Step 5. Load the jar files required for spring framework

Step 6. Run the ApplicationClass

Step 7. Stop

Program:-

WelcomeMessage.java

```
package com.Spring.Examples;

public class WelcomeMessage {

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

WelcomeMessageConfigClass.java

```
package com.Spring.Examples;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

//This is a class-level annotation
@Configuration
public class WelcomeMessageConfigClass {

     @Bean
       public WelcomeMessage displayWelcomeMessage(){
          return new WelcomeMessage();
       }
}
```

ApplicationClass.java

```
package com.Spring.Examples;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class ApplicationClass {

    private static ApplicationContext ctx;

    public static void main(String[] args) {
        //The Context object differs on the basis of whether we are using
Annotations or xml.
        ctx = new
AnnotationConfigApplicationContext(WelcomeMessageConfigClass.class);

        WelcomeMessage welcomeObject = ctx.getBean(WelcomeMessage.class);
        welcomeObject.setMessage("Welcome to Spring Framework!!");
        String message =welcomeObject.getMessage();
        System.out.println(message);
    }
}
```

Output:-

Result:-

Demonstrated a Spring Standalone application by creating a basic simple Java Project, then including Spring Framework and common logging API libraries, actual source files that displays a console-based text to the user. Also, the Spring Beans Configuration is performed in a Bean class with class-level annotation - @Configuration in order to display the console text to the user..

10.3

Aim:-

To Develop a Java program that calculates the Future Investment Value with present value, Interest rate and Time period entered via Scanner class from the user.

Algorithm:-

Step 1. Start

Step 2. Input the present value, Interest rate and Time period for which Future Investment Value is to be computed.

Step 3. Calculate future investment value=present value * Math.pow((1+rate/100),time period);

Step 4. Print the future investment value to the user.

Step 5. Stop

Program:-

```
import java.util.Scanner;
class InvestmentCalculator
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);

       System.out.print("Enter present value: ");
      double p=sc.nextInt();
       System.out.print("Enter the interest rate: ");
       double r=sc.nextInt();
      System.out.print("Enter the time period in years: ");
       double y=sc.nextInt();
      double f=p*Math.pow((1+r/100),y);
      System.out.print("Future Investment value  is: "+f);
  }
}
```

Output:-

```
Enter present value: 1000
Enter the interest rate: 10
Enter the time period in years: 2
Future Investment value  is: 1210.0000000000002
```

Result:-

Developed a Java program that calculated the Future Investment Value with present value, Interest rate and Time period entered via Scanner class from the user.