A.Y. 2022-2023

**Journal of Advanced Java Programming**

Institute of Management & Computer Studies, Thane



Course Name: Master of Computer Application Subject Name: Advanced Java Programming

Submitted To: Prof.Rajshree Thete Submitted by:.

Roll Number:

Semester: 1st semester

**ASM’s**

**Institute of Management & Computer Studies**

# MCA Institute

Affiliated to University of Mumbai & Approved by AICTE, C-4, Wagle Industrial Estate, Near Mulund Check Naka,Opp. to Aplab, Thane (W) – 400604.

Certificate

This is to certify that Mr./Ms.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.Student of **MCA** Course, First year, **Semester 1,** Roll No. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_has successfully completed the required number of practical in subject of **Advanced Java Programming** as prescribed by the **University of Mumbai** under our supervision during the academic-year 2022-2023.

**Practical In-Charge Internal Examiner**

**Date: Date:**

**External Examiner Director**

**Date: IMCOST**

**College Seal:**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Topic** | **Sign** |
| **1.** | **Practical No. 1** |  |
|  | **Assignments on Java Generics** |  |
|  | 1. Write a Java Program to demonstrate a Generic Class. |  |
|  | 2. Write s Java Program to demonstrate Generic Methods. |  |
|  | 3. Write a Java Program to demonstrate Wildcards in Java  Generics. |  |
| **2.** | **Practical No. 2** |  |
|  | **Assignments on List Interface** |  |
|  | 1. Write a Java program to create List containing list of items of  type String and use for---each loop to print the items of the list. |  |
|  | 2. 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/ backword direction. |  |
| **3.** | **Practical No. 3** |  |
|  | **Assignments on Set Interface** |  |
|  | 1. Write a Java program to create a Set containing list of items of  type String and printthe items in the list using Iterator interface.  Also print the list in reverse/ backword direction. |  |
|  | 1. Write a Java program using Set interface containing list of items   and perform the following operations:   * 1. Add items in the set.   2. Insert items of one set in to other set.   3. Remove items from the set   4. Search the specified item in the set |  |
| **4.** | **Practical No. 4** |  |
|  | **Assignments on Map Interface** |  |
|  | 1. Write a Java program using Map interface containing list of   items having keys and associated values and perform the following operations:   * 1. Add items in the map.   2. Remove items from the map |  |

Advanced JAVA

|  |  |  |
| --- | --- | --- |
|  | 1. Search specific key from the map 2. Get value of the specified key 3. Insert map elements of one map in to other map. 4. Print all keys and values of the map. |  |
| **5.** | **Practical No. 5** |  |
|  | **Assignments on Lambda Expression** |  |
|  | 1. Write a Java program using Lambda Expression to print ”Hello  World”. |  |
|  | 2. Write a Java program using Lambda Expression with single  parameters. |  |
|  | 3. Write a Java program using Lambda Expression with multiple  parameters to add two numbers. |  |
|  | 1. Write a Java program using Lambda Expression to calculate the   following:   * 1. Convert Fahrenheit to Celcius   2. Convert Kilometers to Miles. |  |
|  | 5. Write a Java program using Lambda Expression with or without  return keyword. |  |
|  | 6. Write a Java program using Lambda Expression to concatenate  two strings. |  |
| **6.** | **Practical No. 6** |  |
|  | **Assignments based on web application development using JSP** |  |
|  | 1. 1. 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. |  |
|  | 2. 2. Write a JSP page to display the Registration form (Make your  own assumptions) |  |
|  | 3. 3. Write a JSP program to add, delete and display the records  from StudentMaster (RollNo, Name, Semester, Course) table. |  |
|  | 1. Design loan calculator using JSP which accepts Period of Time   (in years) and Principal Loan Amount. Display the payment amount for each loan and then list the loan balance and interest paid for each payment over the term of the loan for the following time period and interest rate:   * 1. 1 to 7 year at 5.35%   2. 8 to 15 year at 5.5%   3. 16 to 30 year at 5.75% |  |
|  | 5. Write a program using JSP that displays a webpage consisting  Application form for change of Study Center which can be filled by any student who wants to change his/her study center. Make necessary assumptions |  |
|  | 6. Write a JSP program to add, delete and display the records from  StudentMaster (RollNo, Name, Semester, Course) table. |  |

Advanced JAVA

|  |  |  |
| --- | --- | --- |
|  | 7. Write a JSP program that demonstrates the use of  JSPdeclaration, scriptlet, directives, expression, header and footer. |  |
| **7.** | **Practical No. 7** |  |
|  | **Assignment based Spring Framework** |  |
|  | 1. Write a program to print “Hello World” using spring  framework. |  |
|  | 2. Write a program to demonstrate dependency injection via setter  method. |  |
|  | 3. Write a program to demonstrate dependency injection via  Constructor. |  |
| **8.** | **Practical No. 8** |  |
|  | **Assignment based Aspect Oriented Programming** |  |
|  | 1. Write a program to demonstrate Spring AOP – before advice. |  |
|  | 2. Write a program to demonstrate Spring AOP – after advice. |  |
|  | 3. Write a program to demonstrate Spring AOP – around advice. |  |
|  | 4. Write a program to demonstrate Spring AOP – after returning  advice. |  |
|  | 5. Write a program to demonstrate Spring AOP – after throwing  advice. |  |
|  | 6. Write a program to demonstrate Spring AOP – pointcuts. |  |
| **9.** | **Practical No. 9** |  |
|  | **Assignment based Spring JDBC** |  |
|  | 1. Write a program to insert, update and delete records from the  given table. |  |
|  | 2. Write a program to demonstrate PreparedStatement in Spring  JdbcTemplate |  |
|  | 3. Write a program in Spring JDBC to demonstrate  ResultSetExtractor Interface |  |
|  | 4. Write a program to demonstrate RowMapper interface to fetch  the records from the database. |  |
| **10.** | **Practical No. 10** |  |
|  | **Assignment based Spring Boot and RESTful Web Services** |  |
|  | 1. Write a program to create a simple Spring Boot application that  prints a message. |  |
|  | 2. Write a program to demonstrate RESTful Web Services with  spring boot. |  |

**PRACTICAL NO. 1**

## AIM / TITLE: Assignments on Java Generics

1. **Write a Java Program to demonstrate a Generic Class.**
2. **Write s Java Program to demonstrate Generic Methods.**
3. **Write a Java Program to demonstrate Wildcards in Java Generics.**

**Description:**

Generics means parameterized types. The idea is to allow type (Integer, String, etc, and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types.

The Java Generics programming is introduced in J2SE 5 to deal with type-safe objects. It makes the code stable by detecting the bugs at compile time.

Before generics, we can store any type of objects in the collection, i.e., non- generic. Now generics force the java programmer to store a specific type of objects.

## CODING:

1. **Write a Java Program to demonstrate a Generic Class.**

**Generics.java**

package a;

// **TODO** Auto-generated method stub

// A Simple Java program to show working of user defined

// Generic classes

// We use < > to specify Parameter type class Test<T>

{

// An object of type T is declared T obj;

Test(T obj) { this.obj = obj; } // constructor

public T getObject() { return this.obj; }

}

// Driver class to test above class Generics

{

public static void main (String[] args)

{

// instance of Integer type

Test <Integer> iObj = new Test<Integer>(15); System.***out***.println(iObj.getObject());

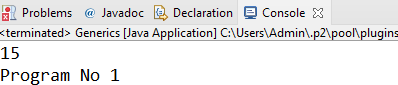
}

}

## Output:

// instance of String type

Test <String> sObj = new Test<String>("Program No 1"); System.***out***.println(sObj.getObject());



## Write s Java Program to demonstrate Generic Methods.

**GenericMethod.java**

**package** a;

**public class** GenericMethod{

**public static** < E > **void** printArray(E[] elements) {

**for** ( E element : elements){ System.***out***.println(element );

}

System.***out***.println();

}

**public static void** main( String args[] ) { Integer[] intArray = { 10, 20, 30, 40, 50 };

Character[] charArray = { 'A', 'D', 'V', 'A', 'N','C','E','D','I','T' };

System.***out***.println( "Printing Integer Array" );

*printArray*( intArray );

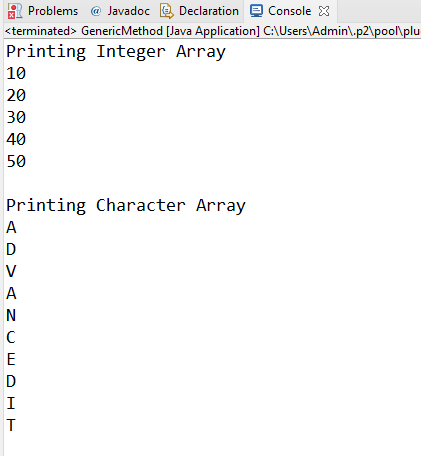
System.***out***.println( "Printing Character Array" );

*printArray*( charArray );

}

}

## Output:



1. **Write a Java Program to demonstrate Wildcards in Java Generics.**

**Wildcards.java**

**package** a;

**import** java.util.\*;

**abstract class** Shape

{

**abstract void** draw();

}

**class** Rectangle **extends** Shape{

**void** draw()

{

System.***out***.println("drawing rectangle");}

}

**class** Circle **extends** Shape

{

**void** draw(){System.***out***.println("drawing circle");}

}

**class** Wildcards

{

//creating a method that accepts only child class of Shape

**public static void** drawShapes(List<? **extends** Shape> lists)

{

**for**(Shape s:lists)

{

s.draw();

//calling method of Shape class by child class instance

}

}

**public static void** main(String args[])

{

List<Rectangle> list1=**new** ArrayList<Rectangle>(); list1.add(**new** Rectangle());

List<Circle> list2=**new** ArrayList<Circle>(); list2.add(**new** Circle());

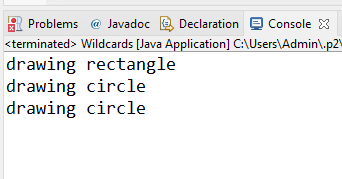
list2.add(**new** Circle());

}

## }

**Output:**

*drawShapes*(list1); *drawShapes*(list2);



**PRACTICAL NO. 2**

## AIM / TITLE: Assignments on List Interface

* 1. **Write a Java program to create List containing list of items of type String and use for---each loop to print the items of the list.**
  2. **Write a Java program to create List containing list of items and use ListIteratorinterface to print items present in the list. Also print the list in reverse/ backword direction.**

**Description:**

List in Java provides the facility to maintain the *ordered collection*. It contains the index-based methods to insert, update, delete and search the elements. It can have the duplicate elements also. We can also store the null elements in the list.

List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

## CODING:

1. **Write a Java program to create List containing list of items of type String and use for---each loop to print the items of the list.**

**ListExample.java package** b;

**import** java.util.\*;

**class** ListExample

{

**public static void** main(String args[])

{

//Creating a list of elements

ArrayList<String> list=**new** ArrayList<String>(); list.add("Cat");

list.add("Dog");

list.add("Rat");

//traversing the list of elements using for-each loop

**for**(String s:list)

{

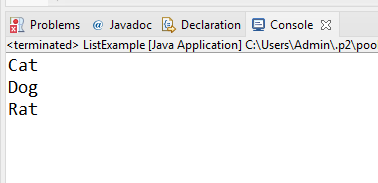
System.***out***.println(s);

}

}

## }

**Output:**



1. **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.**

**ListExample2.java package** b;

**import** java.util.ArrayList; **import** java.util.List; **import** java.util.ListIterator;

**public class** ListExample2

{

**public static void** main(String a[])

{

ListIterator<String> litr = **null**;

List<String> names = **new** ArrayList<String>(); names.add("Milk");

names.add("Egg"); names.add("Butter"); names.add("Potatoes"); names.add("Almonds");

//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:");

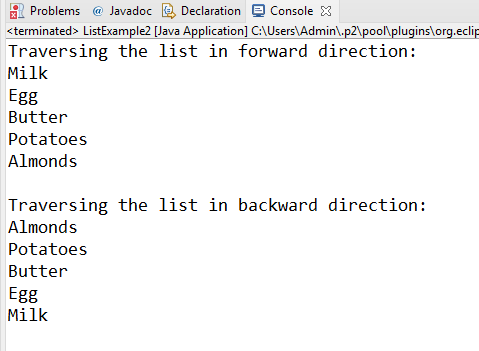
**while**(litr.hasPrevious()){ System.***out***.println(litr.previous());

}

}

## }

**Output:**



**PRACTICAL NO. 3**

**AIM / TITLE: Assignments on Set Interface**

* 1. **Write a Java program to create a Set containing list of items of type Strin and print the items in the list using Iterator interface. Also print the list in reverse/ backward direction.**
  2. **Write a Java program using Set interface containing list of items and perform the following operations:**
     1. **Add items in the set.**
     2. **Insert items of one set in to other set.**
     3. **Remove items from the set**
     4. **Search the specified item in the set**

**Description:**

A Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction.

The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.

Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ.

Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

## CODING:

1. **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/ backward direction.**

**SetExample.java package** c;

**import** java.util.ArrayList; **import** java.util.LinkedHashSet; **import** java.util.Collections;

**public class** SetExample

{

**public static void** main(String[] args)

{

// creating LinkedHashSet object of type String

LinkedHashSet<String> lhsCompanies = **new** LinkedHashSet<String>();

// adding elements to LinkedHashSet object lhsCompanies.add("LinkedIn"); lhsCompanies.add("Amazon"); lhsCompanies.add("Google"); lhsCompanies.add("Apple"); lhsCompanies.add("Facebook"); lhsCompanies.add("Oracle"); lhsCompanies.add("Microsoft");

// Iterating using enhanced for-loop

System.***out***.println("Insertion Order: Iterating LinkedHashSet\n");

**for**(String company : lhsCompanies)

{

System.***out***.println(company);

}

// convert to ArrayList

ArrayList<String> alCompanies = **new** ArrayList<String>(lhsCompanies);

// to reverse LinkedHashSet contents Collections.*reverse*(alCompanies);

// reverse order of LinkedHashSet contents System.***out***.println("\n\n\nReverse Order of LinkedHashSet\n"); **for**(String company : alCompanies) {

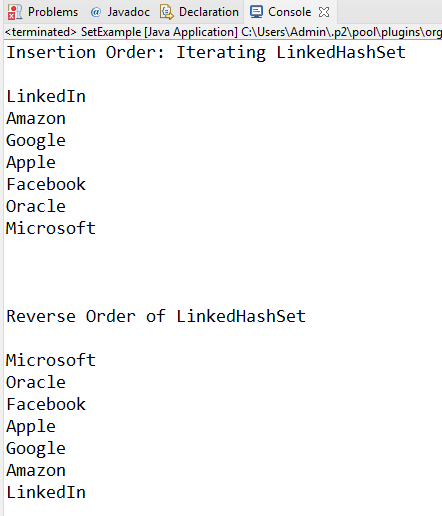
System.***out***.println(company);

}

}

## }

**Output:**



1. **Write a Java program using Set interface containing list of items and perform the following operations:**
   1. **Add items in the set.**
   2. **Insert items of one set in to other set.**
   3. **Remove items from the set**
   4. **Search the specified item in the set**

**SetExampleb.java package** c;

**import** java.util.HashSet; **import** java.util.Iterator; **class** SetExampleb

{

**public static void** main(String[] args) {

// Create a HashSet

HashSet<String> hset = **new** HashSet<String>();

//add elements to HashSet hset.add("Rose");

hset.add("Lily");

hset.add("Tulip");

System.***out***.println("Set before addAll:"+ hset);

//Add items in the set. hset.add("Welcome");

System.***out***.println("Set after adding 1 elemrnt " + hset);

// Create another HashSet

HashSet<String> hset2 = **new** HashSet<String>(); hset2.add("Cat");

hset2.add("Dog");

// Insert items of one set in to other set. hset.addAll(hset2);

System.***out***.println("Set after add\_All: "+ hset);

// Removing elements using remove() method

hset.remove("Welcome");

// Displaying the Set after removal System.***out***.println("Set after removing elements: "

+ hset);

// Check for "Item2" in the set System.***out***.println("Does the Set contains 'Item2'? "

+ hset.contains("Item2"));

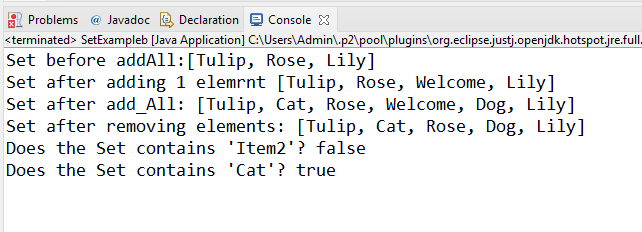
// Check for "Cat" in the set System.***out***.println("Does the Set contains 'Cat'? "

+ hset.contains("Cat"));

}

## }

**Output:**



**PRACTICAL NO. 4**

**AIM / TITLE: Assignments on Map Interface**

1. **Write a Java program using Map interface containing list of items having keys and associated values and perform the following operations:**
   1. **Add items in the map.**
   2. **Remove items from the map**
   3. **Search specific key from the map**
   4. **Get value of the specified key**
   5. **Insert map elements of one map in to other map.**
   6. **Print all keys and values of the map.**

**Description:**

The Map interface present in java.util package represents a mapping between a key and a value. The Map interface is not a subtype of the Collection interface.

Therefore it behaves a bit differently from the rest of the collection types.

A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.

A Map is useful if you have to search, update or delete elements on the basis of a

key.

## CODING:

**MapDemo.java package** d;

**import** java.util.\*;

**import** com.sun.tools.classfile.CharacterRangeTable\_attribute.Entry;

**public class** MapDemo

{

**public static void** main(String[] args)

{

// Creating an empty HashMap

HashMap<Integer, String> hash\_map = **new** HashMap<Integer, String>();

// Mapping string values to int keys hash\_map.put(10, "Apple"); hash\_map.put(15, "Banana"); hash\_map.put(20, "Berries"); hash\_map.put(25, "Melons"); hash\_map.put(30, "Oranges");

// Displaying the HashMap

System.***out***.println("Initial Mappings are: " + hash\_map);

// Creating a new hash map and copying

HashMap<Integer, String> hmap2 = **new** HashMap<Integer, String>();

// Add items in the map hmap2.put(11, "Hello");

hmap2.put(22, "Hi");

// Copying one HashMap "hash\_map" to another HashMap "hmap2" hmap2.putAll(hash\_map);

// Displaying the final HashMap

System.***out***.println("The new map looks like this: " + hmap2);

//key-based removal hmap2.remove(30);

System.***out***.println("Updated list of elements: "+hmap2);

// Checking for the key\_element '20' System.***out***.println("Is the key '20' present? " + hash\_map.containsKey(20));

// Checking for the key\_element '5' System.***out***.println("Is the key '5' present? " + hash\_map.containsKey(5));

// Getting values from HashMap

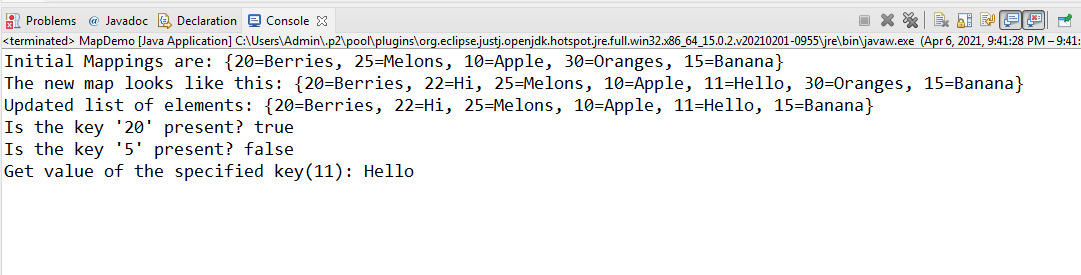
String val=hmap2.get(11);

System.***out***.println("Get value of the specified key(11): "+ val);

}

}

## Output:



**PRACTICAL NO. 5**

**AIM / TITLE: Assignments on Lambda Expression**

1. **Write a Java program using Lambda Expression to print ”Hello World”.**
2. **Write a Java program using Lambda Expression with single parameters.**
3. **Write a Java program using Lambda Expression with multiple parameters to add two numbers.**
4. **Write a Java program using Lambda Expression to calculate the following:**
   1. **Convert Fahrenheit to Celcius**
   2. **Convert Kilometers to Miles.**
5. **Write a Java program using Lambda Expression with or without return keyword.**
6. **Write a Java program using Lambda Expression to concatenate two strings.**

**Description:**

Lambda expression is a new and important feature of Java which was included in Java SE 8. It provides a clear and concise way to represent one method interface using an expression. It is very useful in collection library. It helps to iterate, filter and extract data from collection.

The Lambda expression is used to provide the implementation of an interface which has functional interface. It saves a lot of code. In case of lambda expression, we don't need to define the method again for providing the implementation. Here, we just write the implementation code.

Java lambda expression is treated as a function, so compiler does not create .class

file.

## CODING:

1. **Write a Java program using Lambda Expression to print ”Hello World”.**

**Lam1.java package** e;

@FunctionalInterface

**interface** MyFunctionalInterface

{

//A method with no parameter

**public** String sayHello();

}

**public class** Lam1

{

**public static void** main(String args[])

{

// lambda expression MyFunctionalInterface msg = () ->

{

**return** "Hello World";

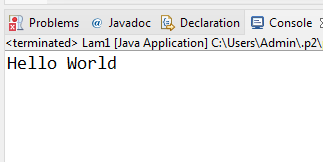
};

System.***out***.println(msg.sayHello());

}

}

## Output:



1. **Write a Java program using Lambda Expression with single parameters.**

**Lam2.java package** e;

@FunctionalInterface

**interface** MyFun

{

//A method with single parameter

**public int** incrementByFive(**int** a);

}

**public class** Lam2

{

**public static void** main(String args[])

{

// lambda expression with single parameter num MyFun f = (num) -> num+5;

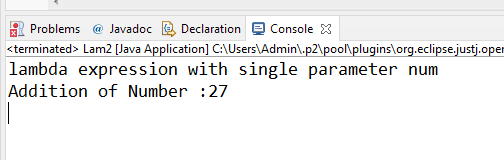
System.***out***.println("lambda expression with single parameter num");

System.***out***.println("Addition of Number :"+f.incrementByFive(22));

}

## }

**Output:**



1. **Write a Java program using Lambda Expression with multiple parameters to add two numbers.**

**Lam3.java package** e;

**interface** Addable

{ // Multiple parameters in lambda expression

**int** add(**int** a,**int** b);

}

**public class** Lam3

{

**public static void** main(String[] args)

{

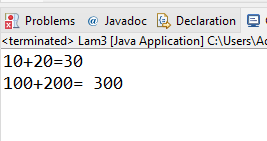
Addable ad1=(a,b)->(a+b); System.***out***.println("10+20=" +ad1.add(10,20));

// Multiple parameters with data type in lambda expression Addable ad2=(**int** a,**int** b)->(a+b); System.***out***.println("100+200= "+ad2.add(100,200));

}

}

## Output:



1. **Write a Java program using Lambda Expression to calculate the following:**
   1. **Convert Fahrenheit to Celcius**
   2. **Convert Kilometers to Miles.**

**Lam4.java**

**package** e; @FunctionalInterface **interface** Converter

{

**double** convert(**double** input);

}

**public class** Lam4

{

**public static void** main(String[] args)

{

// Convert Fahrenheit to Celsius

System.***out***.println("Fahrenheit to Celsius: "+*convert*(input -> (input-32)\*5.0/9.0,

98.6));

// Convert Kilometers to Miles

System.***out***.println("Kilometers to Miles: "+*convert*(input -> input/1.609344, 8));

}

**static double** convert(Converter converter, **double** input)

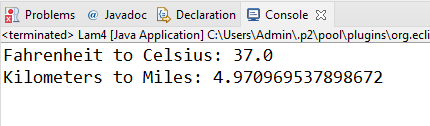
{

**return** converter.convert(input);

}

## }

**Output:**



1. **Write a Java program using Lambda Expression with or without return keyword.**

**Lam5.java package** e;

**interface** Sub

{

**int** sub(**int** a,**int** b);

}

**public class** Lam5 {

**public static void** main(String[] args) {

// Lambda expression without return keyword. Sub ad=(a,b)->(a-b); System.***out***.println(ad.sub(40,20));

// Lambda expression with return keyword. Sub ad2=(**int** a,**int** b)->

{

**return** (a-b);

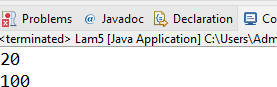
};

System.***out***.println(ad2.sub(300,200));

}

## }

**Output:**



1. **Write a Java program using Lambda Expression to concatenate two strings.**

**Lam6.java package** e;

@FunctionalInterface

**interface** Sayable{

String say(String message);

}

**public class** Lam6

{

**public static void** main(String[] args)

{

// You can pass multiple statements in lambda expression Sayable person = (message)-> {

String str1 = "When you fully commit, ";

//Lambda Expression to concatenate two strings String str2 = str1 + message;

**return** str2;

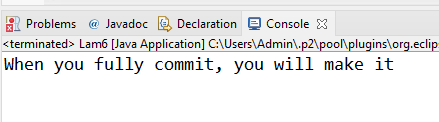
};

System.***out***.println(person.say("you will make it"));

}

}

## Output:



**PRACTICAL NO. 6**

**AIM / TITLE: Assignments based on web application development using JSP**

**Description:-**

**Java Server Pages** :- JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc. A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

# Advantages of JSP :-

There are many advantages of JSP over the Servlet. They are as follows :-

1. **Extension to Servlet** :- JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.
2. **Easy to maintain** :- JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.
3. **Fast Development**: No need to recompile and redeploy :- If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application
4. **Less code than Servlet** :- In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

**Syntax available in JSP are following :-**

1. **Declaration Tag** :- It is used to declare variables. Syntax:- <%! Dec var %>
2. **Java sScriplet** :- It allows us to add any number of JAVA code, variables and expression Syntax:- <% java code %>
3. **JSP Expression** :- It evaluates and convert the expression to a string. Syntax:- <%= expression %>
4. **JAVA Comments** :- It contains the text that is added for information which has Syntax:- <% -- JSP Comments %>

## CODING:

* 1. **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.**

<%@ page import="java.io.,java.util.,java.sql.\*"%>

<%@ page import="javax.servlet.http.,javax.servlet." %>

<%@ taglib uri="<http://java.sun.com/jsp/jstl/core>" prefix="c"%>

<%@ taglib uri="<http://java.sun.com/jsp/jstl/sql>" prefix="sql"%>

<%@ 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>Employee Table</title>

</head>

<sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"

url="jdbc:mysql://localhost:3306/assignment?allowPublicKeyRetrieval=true&useSSL=false" user="root" password="1234"/>

<%--"jdbc:mysql://localhost:3306/students?allowPublicKeyRetrieval=true&useSSL=false//--%>

<sql:update dataSource="${snapshot}" var="result">

INSERT INTO TELEPHONE VALUES(4 , 'Sandra Bullock' , '9184567418','California');

</sql:update>

<sql:query dataSource="${snapshot}" var="result"> SELECT \* from telephone ;

</sql:query>

<table border="1" width="100%">

<tr>

<th>PersonId</th>

<th>Name</th>

<th>Telephone Number</th>

<th> City</th>

</tr>

<c:forEach var="row" items="${result.rows}">

<tr>

<td><c:out value="${row.PersonID}"/></td>

<td><c:out value="${row.Name}"/></td>

<td><c:out value="${row.Phone}"/></td>

<td><c:out value="${row.City}"/></td>

</tr>

</c:forEach>

</table>

</html>

## Output:



* 1. **Write a JSP page to display the Registration form (Make your own assumptions)**

**newjsp.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"](http://www.w3.org/TR/html4/loose.dtd)>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>JSP Registration Form</title>

</head>

<body>

<h1>JSP Register Form</h1>

<form action="jsp\_register" method="post">

<table style="with: 50%">

<tr>

<td>First Name</td>

<td><input type="text" name="first\_name" /></td>

</tr>

<tr>

<td>Last Name</td>

<td><input type="text" name="last\_name" /></td>

</tr>

<tr>

<td>UserName</td>

<td><input type="text" name="username" /></td>

</tr>

<tr>

<td>Password</td>

<td><input type="password" name="password" /></td>

</tr>

<tr>

<td>Address</td>

<td><input type="text" name="address" /></td>

</tr>

<tr>

<td>Contact No</td>

<td><input type="text" name="contact" /></td>

</tr></table>

<input type="submit" value="Submit" /></form>

</body>

## Jspregister.java

package com.test;

import java.io.IOException;

import javax.servlet.RequestDispatcher; import javax.servlet.ServletException; import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse; public class jsp\_register extends HttpServlet {

private static final long serialVersionUID = 1L;

protected void doPost(HttpServletRequest request, HttpServletResponse

response) throws ServletException, IOException {

// TODO Auto-generated method stub

String first\_name = request.getParameter("first\_name"); String last\_name = request.getParameter("last\_name"); String username = request.getParameter("username"); String password = request.getParameter("password"); String address = request.getParameter("address"); String contact = request.getParameter("contact");

if(first\_name.isEmpty() || last\_name.isEmpty() || username.isEmpty()

||

password.isEmpty() || address.isEmpty() ||

contact.isEmpty())

{

RequestDispatcher req =

request.getRequestDispatcher("newjsp.jsp");

req.include(request, response);

}

else

{

RequestDispatcher req =

request.getRequestDispatcher("Register\_2.jsp");

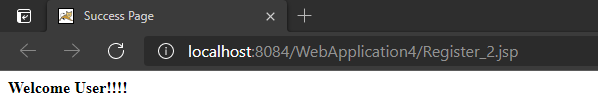
req.forward(request, response);

}

}

}

## Output:



**Q3:**. **Write a JSP program to add, delete and display the records from StudentMaster (RollNo, Name, Semester, Course) table.**

* + 1. First create database in mysql Create Database **StudentMaster;**
    2. Create Table in StudentMaster Database

CREATE TABLE students (

id int NOT NULL AUTO\_INCREMENT,

RollNo varchar(50) NOT NULL, Name varchar(50) NOT NULL , Semester varchar(50) NOT NULL, Course varchar(50) NOT NULL, PRIMARY KEY (id)

);

* + 1. Add Jar **mysql- Connecter**

## form.html

<!DOCTYPE html>

<html>

<body>

<form method="post" action="insert.jsp"> Roll No:<br>

<input type="text" name="rollno">

<br> Name:<br>

<input type="text" name="name">

<br> Semester:<br>

<input type="text" name="semester">

<br> Course:<br>

<input type="text" name="course">

<br><br>

<input type="submit" value="submit">

</form>

</body>

</html>

## Insert.jsp

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

<%@page import="java.sql.\*,java.util.\*"%>

<%

String RollNo=request.getParameter("rollno"); String Name=request.getParameter("name"); String Semester=request.getParameter("semester"); String Course=request.getParameter("course");

try

{

Class.forName("com.mysql.jdbc.Driver"); Connection conn =

DriverManager.getConnection("jdbc:mysql://localhost:3306/StudentMaster", "root", "1234");

Statement st=conn.createStatement();

int i=st.executeUpdate("insert into students(RollNo,Name,Semester,Course)values('"+RollNo+"','"+Name+"','"+Semester+ "','"+Course+"')");

out.println("Data is successfully inserted!");

}

catch(Exception e)

{

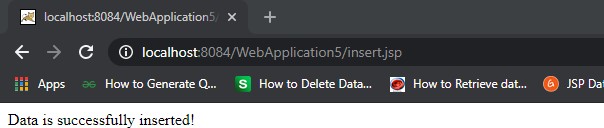
System.out.print(e); e.printStackTrace();

}

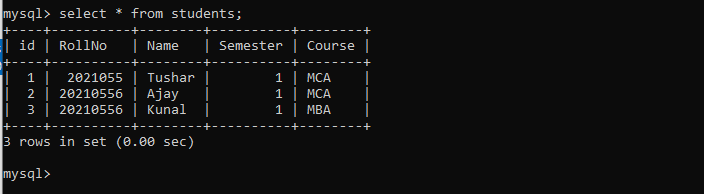
%>

## Output:

## 



Advanced JAVA

****

**For Display and Delete ;**

**Delete.jsp**

<%@page import="java.sql.DriverManager"%>

<%@page import="java.sql.ResultSet"%>

<%@page import="java.sql.Statement"%>

<%@page import="java.sql.Connection"%>

<%

String driver = "com.mysql.jdbc.Driver";

String connectionUrl = "jdbc:mysql://localhost:3306/"; String database = "StudentMaster";

String userid = "root"; String password = "1234"; try { Class.forName(driver);

} catch (ClassNotFoundException e) { e.printStackTrace();

}

Connection connection = null; Statement statement = null; ResultSet resultSet = null;

%>

<!DOCTYPE html>

<html>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<body>

<h1>Retrieve data from database in jsp</h1>

<table border="1">

<tr>

<td>Roll No</td>

<td>Name</td>

<td>Semester</td>

<td>Course</td>

</tr>

<%

try{

connection = DriverManager.getConnection(connectionUrl+database, userid, password); statement=connection.createStatement();

String sql ="select \* from students";

resultSet = statement.executeQuery(sql); int i=0;

while(resultSet.next()){

%>

<tr>

<td><%=resultSet.getString("Rollno") %></td>

<td><%=resultSet.getString("Name") %></td>

<td><%=resultSet.getString("Semester") %></td>

<td><%=resultSet.getString("Course") %></td>

<td><a href="fordelete.jsp?id=<%=resultSet.getString("id") %>"><button type="button" class="delete">Delete</button></a></td>

</tr>

<% i++;

}

connection.close();

} catch (Exception e) { e.printStackTrace();

}

%>

</table>

</body>

</html>

## Fordelete.jsp

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

<%@page import="java.sql.\*,java.util.\*"%>

<%

String id=request.getParameter("id"); try

{

Class.forName("com.mysql.jdbc.Driver"); Connection conn =

DriverManager.getConnection("jdbc:mysql://localhost:3306/StudentMaster", "root", "1234");

Statement st=conn.createStatement();

int i=st.executeUpdate("DELETE FROM students WHERE id="+id); out.println("Data Deleted Successfully!");

}

catch(Exception e)

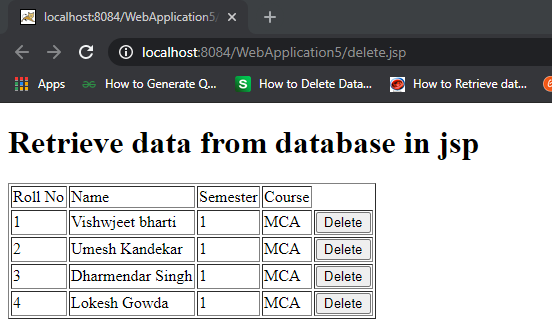
{

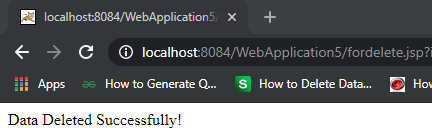
System.out.print(e); e.printStackTrace();

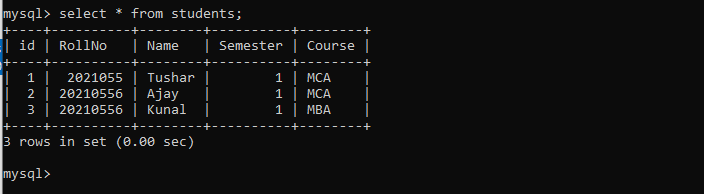
}

%>

## Output:







**Q4. Design loan calculator using JSP which accepts Period of Time (in years) and Principal Loan Amount. Display the payment amount for each loan and then list the loan balance and interest paid for each payment over the term of the loan for the following time period and interest rate:**

* + - 1. **1 to 7 year at 5.35%**
      2. **8 to 15 year at 5.5%**
      3. **16 to 30 year at 5.75%**

**Code:-**

**Q3Page1.jsp**

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

pageEncoding=*"ISO-8859-1"*%>

<!DOCTYPE html>

<html> <head>

<meta charset=*"ISO-8859-1"*>

<title>Loan Calculator</title>

</head> <body>

<form action=*"Prac2Q3 B.jsp"*>

<p>Enter Period of Time in years :- <input type=*"text"* name=*"time"* required></p>

<p>Enter Principal Loan Amount :- <input type=*"text"* name=*"amt"* required></p>

<p><input type=*"submit"* value=*"Submit"*></p>

</form> </body> </html>

**Q3Page2.jsp**

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

pageEncoding=*"ISO-8859-1"*%>

<!DOCTYPE html>

<html> <head>

<meta charset=*"ISO-8859-1"*>

<title>Loan Calculator</title>

</head> <body>

<%!

String StrFormat(**double** a){

String res = String.format("%.2f", a);

**return** res;

}

%>

<%

**double** P = Double.parseDouble(request.getParameter("amt")); **double** Year = Double.parseDouble(request.getParameter("time")); **double** N= Year \*12;

**double** tamt=0, intrest=0, emi=0, R=0;

**if**( Year > 15 && Year < 31){ R=0.0575/12;

}**else if**( Year > 7 && Year < 16){

R=0.055/(12);

}**else if**( Year > 0 && Year < 8){ R=0.0535/12;

}

emi = (P \* R \* Math.pow((1+R),N))/(Math.pow(1+R, N)-1); out.print("Monthly Payment (EMI) :-"+StrFormat(emi)+"<br>"); out.print("Total Payment :-"+ StrFormat((emi\*12)\*Year)+"<br>");

**double** balance = P;

**double** principal, interest; out.print("<br><table border="+1+">");

out.print("<tr><td> Payment# </td> <td> Interest Paid </td> <td> Principal Paid </td> <td>

Loan Balance </td></tr>"); **for** (**int** i = 1; i <= N; i++) { interest = R \* balance;

principal = emi - interest; balance = balance - principal;

out.print("<tr><td>"+ i +"</td> <td>"+ StrFormat(interest) +"</td>

<td>"+StrFormat(principal)+"</td> <td>"+ StrFormat(balance)+"</td></tr>");

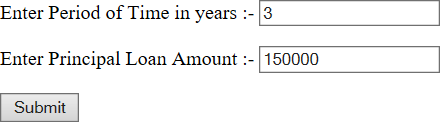
}

out.print("</table>");

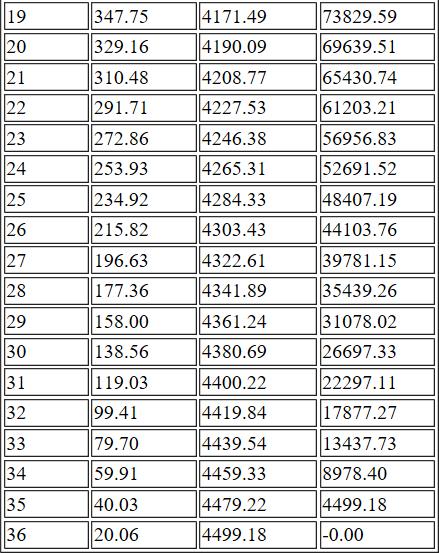
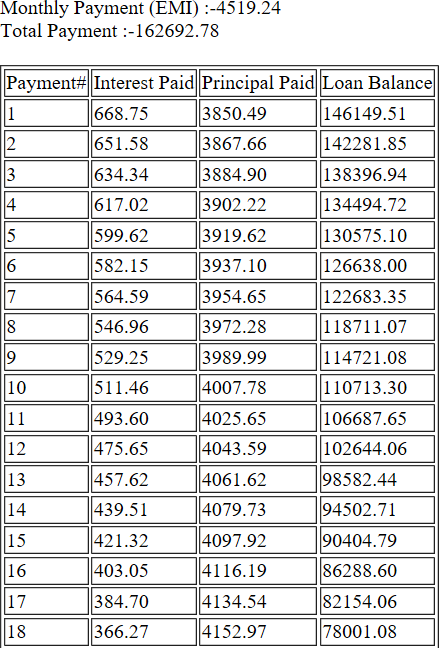
%>

</body>

</html>

**Output :- Page1 :-**

**Page2 :-**



1. **Write a program using JSP that displays a webpage consisting Application form for change of Study Center which can be filled by any student who wants to change his/her study center. Make necessary assumptions**

**Code:-**

**Q4Page1.jsp**

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

pageEncoding=*"ISO-8859-1"*%>

<!DOCTYPE html>

<html> <head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head> <body>

<form action=*"Prac2Q4B.jsp"* >

<table>

<tr><td colspan=*"2"* align=*"center"*>Application for Change of Study Center</td></tr>

<tr><td>Enter Student Name:- </td>

<td><input type=*"text"* name=*"StdName"* ></td></tr>

<tr><td >Enter Student Application Id:-</td>

<td><input type=*"text"* name=*"StdId"*></td></tr>

<tr><td>Enter Branch Name:-</td>

<td><input type=*"text"* name=*"Branch"*></td></tr>

<tr><td valign=*"top"*>Select Current Study Center:-</td>

<td><input type=*"radio"* name=*"CurrentCenter"* value=*"Mumbai"* required>Mumbai<br>

<input type=*"radio"* name=*"CurrentCenter"* value=*"Pune"* required>Pune<br>

<input type=*"radio"* name=*"CurrentCenter"* value=*"Delhi"* required>Delhi<br>

<input type=*"radio"* name=*"CurrentCenter"* value=*"Kolkata"* required>Kolkata<br>

<input type=*"radio"* name=*"CurrentCenter"* value=*"Chennai"* required>Chennai<br>

<input type=*"radio"* name=*"CurrentCenter"* value=*"Hydrabad"* required>Hydrabad<br>

</td></tr>

<tr><td valign=*"top"* >Select New Study Center:-</td>

<td><input type=*"radio"* name=*"NewCenter"* value=*"Mumbai"* required>Mumbai<br>

<input type=*"radio"* name=*"NewCenter"* value=*"Pune"* required>Pune<br>

<input type=*"radio"* name=*"NewCenter"* value=*"Delhi"* required>Delhi<br>

<input type=*"radio"* name=*"NewCenter"* value=*"Kolkata"* required>Kolkata<br>

<input type=*"radio"* name=*"NewCenter"* value=*"Chennai"* required>Chennai<br>

<input type=*"radio"* name=*"NewCenter"* value=*"Hydrabad"* required>Hydrabad<br>

</td></tr>

<tr><td>Enter Reason for Change of Study Center:- </td>

<td><input type=*"text"* name=*"Reason"*></td></tr>

<tr><td colspan=*"2"* align=*"center"* ><input type=*"submit"* value=*"Submit"*></td></tr>

</table>

</form>

</body>

</html>

**Q4Page2.jsp**

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

pageEncoding=*"ISO-8859-1"*%>

<%@ page import=*"java.util.Date"* %>

<!DOCTYPE html>

<html> <head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head> <body>

<%

Date date= **new** Date();

String Date= date.getDate()+"/"+(date.getMonth()+1)+"/"+date.getYear(); String Name= request.getParameter("StdName");

String Id= request.getParameter("StdId");

String Branch= request.getParameter("Branch");

String CurrentCenter= request.getParameter("CurrentCenter"); String NewCenter= request.getParameter("NewCenter"); String Reason= request.getParameter("Reason");

%>

<table>

<tr><td><%= Date %></td></tr>

<tr><td>The Principle,</td></tr>

<tr><td>University Of Mumbai</td></tr>

<tr><td>Vidya Nagari, Kala Ghoda,</td></tr>

<tr><td>Fort, Mumbai</td></tr>

<tr><td >Maharashtra 400032</td></tr>

<tr><td>&emsp;&emsp;Sub: Request for Change Examination Center</td></tr>

<tr><td>Respected Sir,</td></tr>

<tr><td>&emsp;I <%= Name%> am writing this to you so that I can share my problem, Sir you have allotted me</td></tr>

<tr><td><%= CurrentCenter %> as study center and the center allotted to me is very far from

my house and as I have a </td></tr>

<tr><td>serious transportation issue it will not be possible for me to go there due to <%= Reason %></td></tr>

<tr><td>So I want you to change my examination center from <%= CurrentCenter %> center to

<%= NewCenter %> center</td></tr>

<tr><td>It would be so much helpful for me.</td></tr>

<tr><td>Thanks,</td></tr>

<tr><td>Sincerely,</td></tr>

<tr></tr>

<table border=*"1"*>

<tr><td>Name of Student :-</td>

<td><%= Name%></td></tr>

<tr><td>Application ID of Student :-</td>

<td><%= Id%></td></tr>

<tr><td>Branch of Student :-</td>

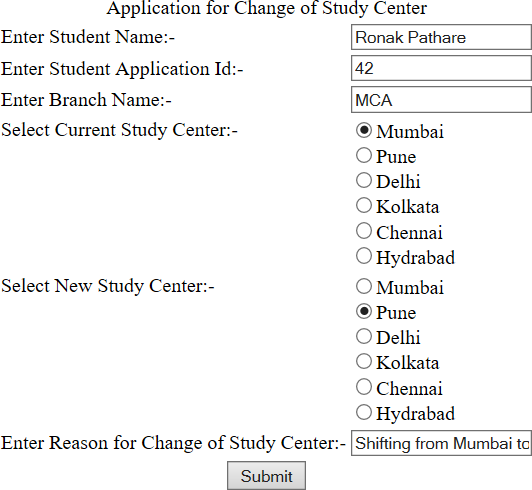
<td><%= Branch%></td></tr>

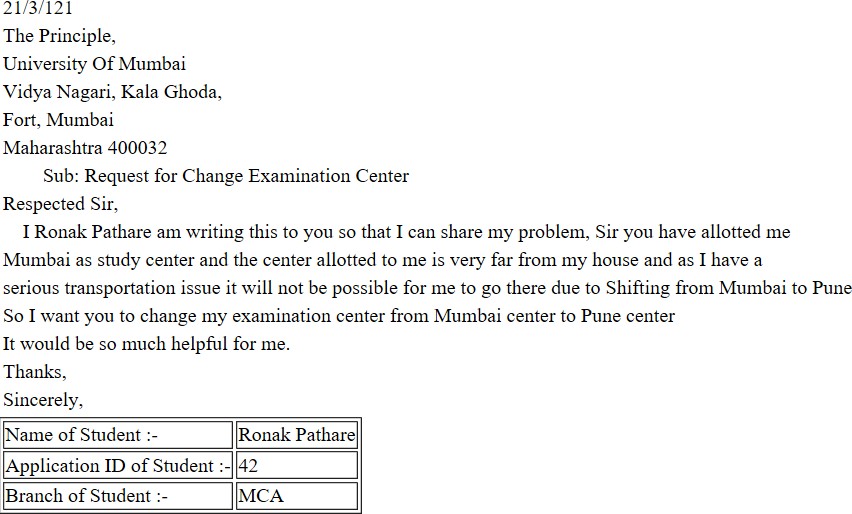
</table> </table>

</body> </html>

**Output:-**

**Page1 :-**



**Page2:**

## Write a JSP program to add, delete and display the records from StudentMaster (RollNo, Name, Semester, Course) table

* 1. First create database in mysql Create Database **StudentMaster;**
  2. Create Table in StudentMaster Database

CREATE TABLE students (

id int NOT NULL AUTO\_INCREMENT,

RollNo varchar(50) NOT NULL, Name varchar(50) NOT NULL , Semester varchar(50) NOT NULL, Course varchar(50) NOT NULL, PRIMARY KEY (id)

);

* 1. Add Jar **mysql- Connecter**

## form.html

<!DOCTYPE html>

<html>

<body>

<form method="post" action="insert.jsp"> Roll No:<br>

<input type="text" name="rollno">

<br> Name:<br>

<input type="text" name="name">

<br> Semester:<br>

<input type="text" name="semester">

<br> Course:<br>

<input type="text" name="course">

<br><br>

<input type="submit" value="submit">

</form>

</body>

</html>

## Insert.jsp

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

<%@page import="java.sql.\*,java.util.\*"%>

<%

String RollNo=request.getParameter("rollno"); String Name=request.getParameter("name"); String Semester=request.getParameter("semester"); String Course=request.getParameter("course");

try

{

Class.forName("com.mysql.jdbc.Driver"); Connection conn =

DriverManager.getConnection("jdbc:mysql://localhost:3306/StudentMaster", "root", "1234");

Statement st=conn.createStatement();

int i=st.executeUpdate("insert into students(RollNo,Name,Semester,Course)values('"+RollNo+"','"+Name+"','"+Semester+ "','"+Course+"')");

out.println("Data is successfully inserted!");

}

catch(Exception e)

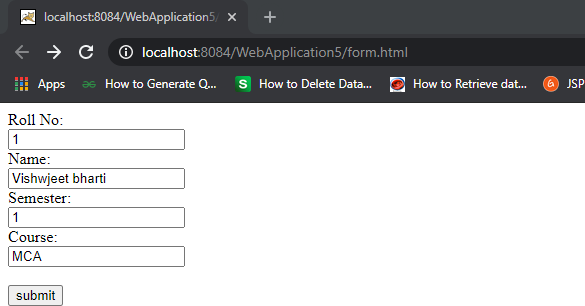
{

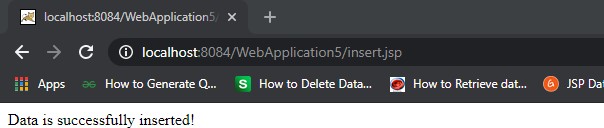
System.out.print(e); e.printStackTrace();

}

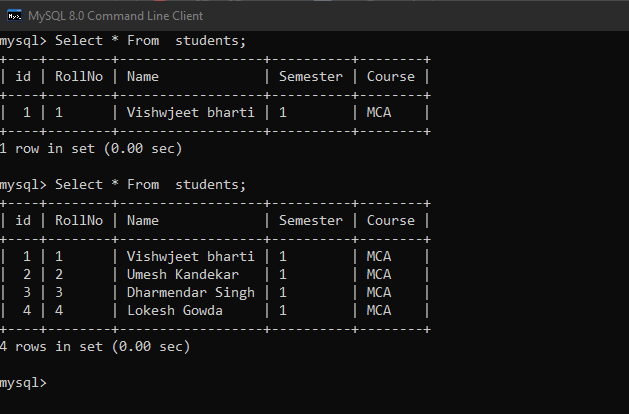
%>

## Output:





Advanced JAVA



**For Display and Delete ;**

**Delete.jsp**

<%@page import="java.sql.DriverManager"%>

<%@page import="java.sql.ResultSet"%>

<%@page import="java.sql.Statement"%>

<%@page import="java.sql.Connection"%>

<%

String driver = "com.mysql.jdbc.Driver";

String connectionUrl = "jdbc:mysql://localhost:3306/"; String database = "StudentMaster";

String userid = "root"; String password = "1234"; try { Class.forName(driver);

} catch (ClassNotFoundException e) { e.printStackTrace();

}

Connection connection = null; Statement statement = null; ResultSet resultSet = null;

%>

<!DOCTYPE html>

<html>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<body>

<h1>Retrieve data from database in jsp</h1>

<table border="1">

<tr>

<td>Roll No</td>

<td>Name</td>

<td>Semester</td>

<td>Course</td>

</tr>

<%

try{

connection = DriverManager.getConnection(connectionUrl+database, userid, password); statement=connection.createStatement();

String sql ="select \* from students";

resultSet = statement.executeQuery(sql); int i=0;

while(resultSet.next()){

%>

<tr>

<td><%=resultSet.getString("Rollno") %></td>

<td><%=resultSet.getString("Name") %></td>

<td><%=resultSet.getString("Semester") %></td>

<td><%=resultSet.getString("Course") %></td>

<td><a href="fordelete.jsp?id=<%=resultSet.getString("id") %>"><button type="button" class="delete">Delete</button></a></td>

</tr>

<% i++;

}

connection.close();

} catch (Exception e) { e.printStackTrace();

}

%>

</table>

</body>

</html>

## Fordelete.jsp

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

<%@page import="java.sql.\*,java.util.\*"%>

<%

String id=request.getParameter("id"); try

{

Class.forName("com.mysql.jdbc.Driver"); Connection conn =

DriverManager.getConnection("jdbc:mysql://localhost:3306/StudentMaster", "root", "1234");

Statement st=conn.createStatement();

int i=st.executeUpdate("DELETE FROM students WHERE id="+id); out.println("Data Deleted Successfully!");

}

catch(Exception e)

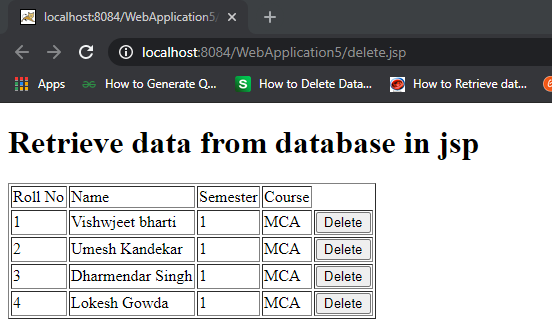
{

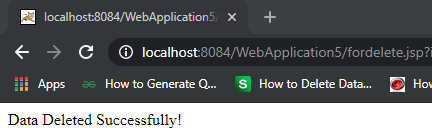
System.out.print(e); e.printStackTrace();

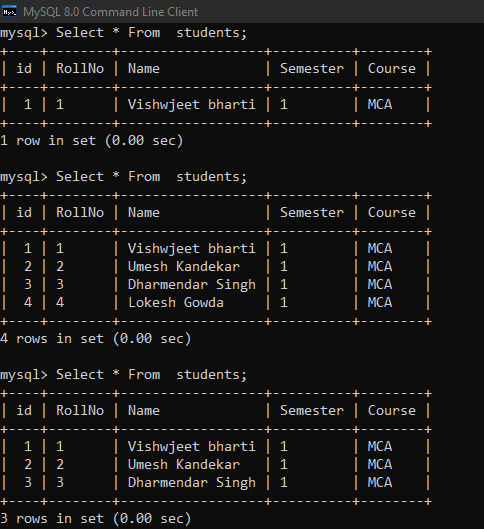
}

%>

## Output:







Advanced JAVA

1. **Write a JSP program that demonstrates the use of JSPdeclaration, scriptlet, directives, expression, header and footer.**

**Newjsp.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"](http://www.w3.org/TR/html4/loose.dtd)>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title> JSP Example</title>

</head>

<body>

<%@include file="header.html" %>

<%-- This is a JSP example with scriplets, comments , expressions --%>

<% out.println("This is JSP Example"); %>

<% out.println("The number is "); %>

<%! int num12 = 12; int num32 = 12; %>

<%= num12\*num32 %>

Today's date: <%= (new java.util.Date()).toLocaleString()%>

<%@include file="footer.html" %>

</body>

</html>

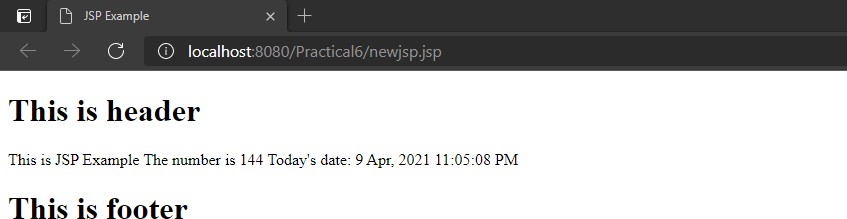
## header.html

<h1>This is header</h1>

**footer.html**

<h1>This is footer</h1>

**Output:**



**PRACTICAL NO. 7**

**AIM / TITLE: Assignment based Spring Framework**

1. **Write a program to print “Hello World” using spring framework.**
2. **Write a program to demonstrate dependency injection via setter method.**
3. **Write a program to demonstrate dependency injection via Constructor. Description:**

The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform.

A key element of Spring is infrastructural support at the application level: Spring focuses on the "plumbing" of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.

It is a lightweight, loosely coupled and integrated framework. Spring Framework is built on top of two design concepts ? Dependency Injection and Aspect Oriented Programming.

## Write a program to print “Hello World” using spring framework.

**HelloWorld.java**

**package** com.first;

**public class** HelloWorld

{

**public** String message;

**public void** setMessage(String message){

**this**.message = message;

}

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

}

}

**MainApp.java package** com.first;

**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");

HelloWorld obj = (HelloWorld) context.getBean("helloWorld"); obj.getMessage();

}

}

## Beans.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns = *"*[*http://www.springframework.org/schema/beans"*](http://www.springframework.org/schema/beans)xmlns:xsi = *"*[*http://www.w3.org/2001/XMLSchema-instance"*](http://www.w3.org/2001/XMLSchema-instance)xsi:schemaLocation = *"*[*http://www.springframework.org/schema/beans*](http://www.springframework.org/schema/beans)[*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"*](http://www.springframework.org/schema/beans/spring-beans-3.0.xsd)>

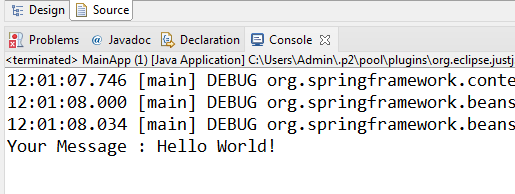
<bean id = *"helloWorld"* class = *"com.first.HelloWorld"*>

<property name = *"message"* value = *"Hello World!"*/>

</bean>

</beans>

## Output:



1. **Write a program to demonstrate dependency injection via setter method.**

**Employee.java package** com.two;

**public class** Employee

{

**public int** id; **public** String name; **public** 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(id+" "+name+" "+city);

}

}

## Test.java

**package** com.two;

**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);

Employee e=(Employee)factory.getBean("obj"); e.display();

}

## }

**applicationContext.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans

xmlns=*"*[*http://www.springframework.org/schema/beans"*](http://www.springframework.org/schema/beans)xmlns:xsi=*"*[*http://www.w3.org/2001/XMLSchema-instance*](http://www.w3.org/2001/XMLSchema-instance)*"* xmlns:p=*"*[*http://www.springframework.org/schema/p*](http://www.springframework.org/schema/p)*"* xsi:schemaLocation=*"*[*http://www.springframework.org/schema/beans*](http://www.springframework.org/schema/beans)

[*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"*](http://www.springframework.org/schema/beans/spring-beans-3.0.xsd)>

<bean id=*"obj"* class=*"com.two.Employee"*>

<property name=*"id"*>

<value>20</value>

</property>

<property name=*"name"*>

<value>Cat</value>

</property>

<property name=*"city"*>

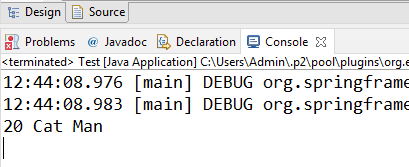
<value>Man</value>

</property>

</bean>

</beans>

## Output:



1. **Write a program to demonstrate dependency injection via Constructor.**

**Employee.java package** com.three;

**public class** Employee

{

## private int id;

**private** String name;

**public** Employee() {System.***out***.println("def cons");

}

**public** Employee(**int** id)

{

**this**.id = id;

}

**public** Employee(String name)

{

**this**.name = name;

}

**public** Employee(**int** id, String name)

{

**this**.id = id;

**this**.name = name;

}

**void** show()

{

System.***out***.println(id+" "+name);

}

}

## application.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans

xmlns=*"*[*http://www.springframework.org/schema/beans"*](http://www.springframework.org/schema/beans)xmlns:xsi=*"*[*http://www.w3.org/2001/XMLSchema-instance*](http://www.w3.org/2001/XMLSchema-instance)*"* xmlns:p=*"*[*http://www.springframework.org/schema/p*](http://www.springframework.org/schema/p)*"* xsi:schemaLocation=*"*[*http://www.springframework.org/schema/beans*](http://www.springframework.org/schema/beans)

[*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"*](http://www.springframework.org/schema/beans/spring-beans-3.0.xsd)>

<bean id=*"e"* class=*"com.three.Employee"*>

<constructor-arg value=*"10"* type=*"int"*></constructor-arg>

<constructor-arg value=*"Moon"*></constructor-arg>

</bean>

</beans>

## Test.java

**package** com.three;

**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)

{

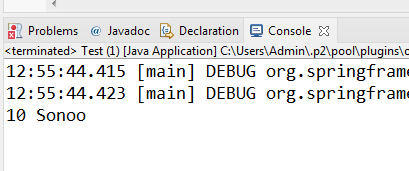
Resource r=**new** ClassPathResource("application.xml"); BeanFactory factory=**new** ~~XmlBeanFactory~~(r);

Employee s=(Employee)factory.getBean("e"); s.show();

}

}

## Output:



**PRACTICAL NO. 8**

**AIM / TITLE: Assignment based Aspect Oriented Programming**

* 1. **Write a program to demonstrate Spring AOP – before advice.**
  2. **Write a program to demonstrate Spring AOP – after advice.**
  3. **Write a program to demonstrate Spring AOP – around advice.**
  4. **Write a program to demonstrate Spring AOP – after returning advice.**
  5. **Write a program to demonstrate Spring AOP – after throwing advice.**
  6. **Write a program to demonstrate Spring AOP – pointcuts. Description:**

An aspect is a common feature that's typically scattered across methods, classes,

object hierarchies, or even entire object models. It is behavior that looks and smells like it should have structure, but you can't find a way to express this structure in code with traditional object-oriented techniques.

**Aspect oriented programming(AOP)** as the name suggests uses aspects in programming. It can be defined as the breaking of code into different modules, also known as [modularisation](https://www.geeksforgeeks.org/modular-approach-in-programming/), where the aspect is the key unit of modularity. Aspects enable the implementation of crosscutting concerns such as- transaction, logging not central to business logic without cluttering the code core to its functionality. It does so by adding additional behaviour that is the advice to the existing code. For example- Security is a crosscutting concern, in many methods in an application security rules can be applied, therefore repeating the code at every method, define the functionality in a common class and control were to apply that functionality in the whole application.

## Write a program to demonstrate Spring AOP – before advice.

**App.java package** com.ram;

**import** org.springframework.context.ApplicationContext;

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

**import** com.ram.EmployeeService;

## public class App

{

**public static void** main(String[] args)

{

ApplicationContext context = **new** ClassPathXmlApplicationContext( "applicationContext.xml");

System.***out***.println(" "); EmployeeService employeeService =

context.getBean("employeeServiceProxy",

EmployeeService.**class**);

employeeService.displayEmployeeInfo();

}

}

**LoggingAdvice.java package** com.ram;

**import** java.lang.reflect.Method;

**import** org.springframework.aop.MethodBeforeAdvice;

**public class** LoggingAdvice **implements** MethodBeforeAdvice

{

**public void** before(Method method, Object[] args, Object target)

**throws** Throwable

{

System.***out***.println("Logging advice is applied before the method "

+ method.getName() + " in the target Object "

+ target.getClass().getName());

}

}

**EmployeeService.java package** com.ram;

**public class** EmployeeService

{

**public void** displayEmployeeInfo()

{

System.***out***.println("Display Employee Information");

}

## }

**applicationContext.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"*[*http://www.springframework.org/schema/beans"*](http://www.springframework.org/schema/beans)xmlns:xsi=*"*[*http://www.w3.org/2001/XMLSchema-instance"*](http://www.w3.org/2001/XMLSchema-instance)xmlns:context=*"*[*http://www.springframework.org/schema/context"*](http://www.springframework.org/schema/context)xsi:schemaLocation=*"*[*http://www.springframework.org/schema/beans*](http://www.springframework.org/schema/beans)

[*http://www.springframework.org/schema/beans/spring-beans-4.3.xsd*](http://www.springframework.org/schema/beans/spring-beans-4.3.xsd)[*http://www.springframework.org/schema/context*](http://www.springframework.org/schema/context)[*http://www.springframework.org/schema/context/spring-context-4.3.xsd"*](http://www.springframework.org/schema/context/spring-context-4.3.xsd)>

<bean id=*"employeeService"* class=*"com.ram.EmployeeService"*></bean>

<bean id=*"loggingAdvice"* class=*"com.ram.LoggingAdvice"*></bean>

<bean id=*"employeeServiceProxy"* class=*"org.springframework.aop.framework.ProxyFactoryBean"*>

<property name=*"target"* ref=*"employeeService"*></property>

<property name=*"interceptorNames"*>

<list>

<value>loggingAdvice</value>

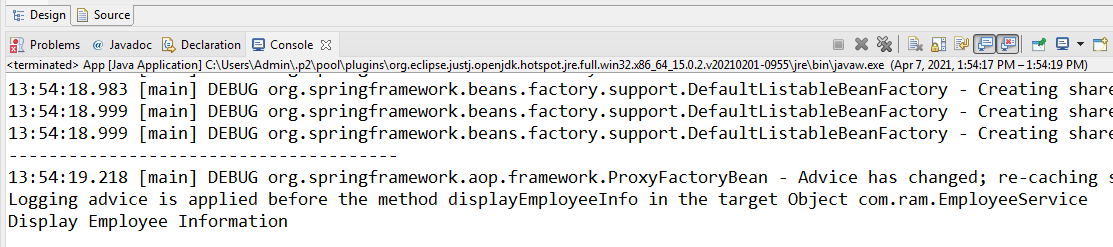
</list>

</property>

</bean>

</beans>

## Output:



1. **Write a program to demonstrate Spring AOP – after advice.**

**Logging.java package** com.two;

**import** org.aspectj.lang.annotation.Aspect; **import** org.aspectj.lang.annotation.Pointcut; **import** org.aspectj.lang.annotation.After;

@Aspect

**public class** Logging {

/\*\* Following is the definition for a PointCut to select

* all the methods available. So advice will be called
* for all the methods.

\*/

@Pointcut("execution(\* com.two.Student.getAge(..))")

**private void** selectGetAge(){}

/\*\*

* This is the method which I would like to execute
* after a selected method execution.

\*/ @After("selectGetAge()") **public void** afterAdvice(){

System.***out***.println("Student profile setup completed.");

}

## }

**Student.java package** com.two;

**public class** Student {

**private** Integer age;

**private** String name;

**public void** setAge(Integer age) {

**this**.age = age;

}

**public** Integer getAge() { System.***out***.println("Age : " + age ); **return** age;

}

**public void** setName(String name) {

**this**.name = name;

}

**public** String getName() { System.***out***.println("Name : " + name ); **return** name;

}

**public void** printThrowException(){ System.***out***.println("Exception raised"); **thrownew** IllegalArgumentException();

}

## }

**MainApp.java**

**package** com.two;

**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();

}

## }

**Beans.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns = "[http://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans) xmlns:xsi = "<http://www.w3.org/2001/XMLSchema-instance>" xmlns:aop = "[http://www.springframework.org/schema/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 student bean -->

<bean id = "student" class = "com.two.Student">

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

<property name = "age" value = "22"/>

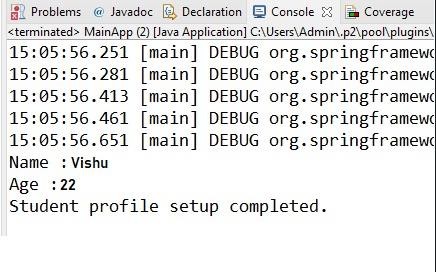
</bean>

<!-- Definition for logging aspect -->

<bean id = "logging" class = "com.two.Logging" />

## </beans>

**Output:**



1. **Write a program to demonstrate Spring AOP – around advice.**

**A.java package** three;

public class A

{

public void m()

{

System.out.println("actual business logic");

}

}

## three.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans

xmlns="[http://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans) xmlns:xsi="[http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance) xmlns:p="[http://www.springframework.org/schema/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"](http://www.springframework.org/schema/beans/spring-beans-3.0.xsd)>

<bean id="obj" class="three.A"></bean>

<bean id="ba" class="three.AroundAdvisor"></bean>

<bean id="proxy" class="org.springframework.aop.framework.ProxyFactoryBean">

<property name="target" ref="obj"></property>

<property name="interceptorNames">

<list>

<value>ba</value>

</list>

</property>

</bean>

</beans>

## Test.java

**package** three;

**import** org.springframework.beans.factory.BeanFactory;

**import** org.springframework.beans.factory.xml.~~XmlBeanFactory~~;

**import** org.springframework.core.io.ClassPathResource;

**import** org.springframework.core.io.Resource;

**public class** Test

{

**public static void** main(String[] args)

{

Resource r=**new** ClassPathResource("three.xml"); BeanFactory factory=**new** ~~XmlBeanFactory~~(r);

A a=factory.getBean("proxy",A.**class**);

System.***out***.println("proxy class name: "+a.getClass().getName());

a.m();

}

}

## AroundAdvisor.java

**package** three;

**import** org.aopalliance.intercept.MethodInterceptor;

**import** org.aopalliance.intercept.MethodInvocation;

**public class** AroundAdvisor **implements** MethodInterceptor{

@Override

**public** Object invoke(MethodInvocation mi) **throws** Throwable {

Object obj;

System.***out***.println("Additional concern before actual logic");

obj=mi.proceed();

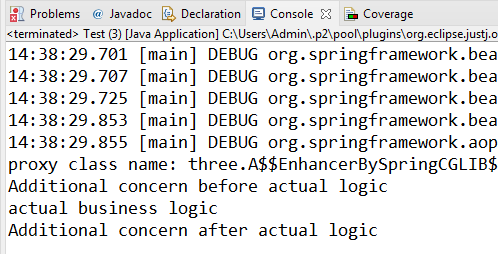
System.***out***.println("Additional concern after actual logic");

**return** obj;

}

}

## Output:



1. **Write a program to demonstrate Spring AOP – after returning advice.**

**A.java package** two;

public class A

{

public void m()

{

System.out.println("actual business logic");

}

}

## AfterAdvisor.java

package two;

import java.lang.reflect.Method;

import org.springframework.aop.AfterReturningAdvice;

public class AfterAdvisor implements AfterReturningAdvice

{

@Override

public void afterReturning(Object returnValue, Method method, Object[] args, Object target) throws Throwable

{

System.out.println("additional concern after returning advice");

}

}

## applicationContext.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans

xmlns="[http://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans) xmlns:xsi="[http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance) xmlns:p="[http://www.springframework.org/schema/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"](http://www.springframework.org/schema/beans/spring-beans-3.0.xsd)>

<bean id="obj" class="two.A"></bean>

<bean id="ba" class="two.AfterAdvisor"></bean>

<bean id="proxy" class="org.springframework.aop.framework.ProxyFactoryBean">

<property name="target" ref="obj"></property>

<property name="interceptorNames">

<list>

<value>ba</value>

</list>

</property>

</bean>

</beans>

## Test.java

package two;

import org.springframework.beans.factory.BeanFactory;

import org.springframework.beans.factory.xml.~~XmlBeanFactory~~; import org.springframework.core.io.ClassPathResource;

import org.springframework.core.io.Resource;

public class Test

{

public static void main(String[] args)

{

Resource r=new ClassPathResource("application.xml"); BeanFactory factory=new XmlBeanFactory(r);

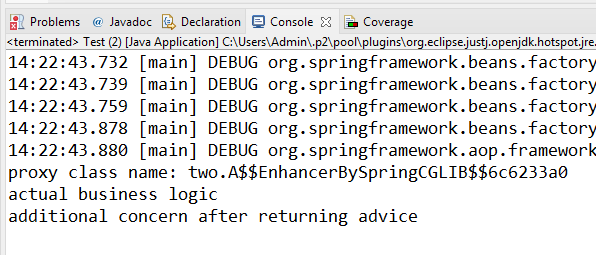
A a=factory.getBean("proxy",A.class);

System.*out*.println("proxy class name: "+a.getClass().getName()); a.m();

}

}

## Output:



1. **Write a program to demonstrate Spring AOP – after throwing advice.**

**Logging.java**

**package** com.five;

**import** org.aspectj.lang.annotation.\*; @Aspect

**public class** Logging

{

//Following is the definition for a pointcut to select all the methods available. So advice will

//be called for all the methods. @Pointcut("execution(\* com.test.\*.\*(..))") **private void** selectAll()

{

}

//This is the method which I would like to execute before a selected method execution. @Before("selectAll()")

**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. @After("selectAll()")

**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. @AfterReturning(pointcut = "selectAll()", returning = "retVal")

**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 by any

//method.

@AfterThrowing(pointcut = "selectAll()", throwing = "ex")

**public void** AfterThrowingAdvice(IllegalArgumentException ex){ System.***out***.println("There has been an exception: " + ex.toString());

}

}

## Student.java

**package** com.five;

**public class** Student { **private** Integer age; **private** String name;

**public void** setAge(Integer age) {

**this**.age = age;

}

**public** Integer getAge() { System.***out***.println("Age : " + age ); **return** age;

}

**public void** setName(String name) {

**this**.name = name;

}

**public** String getName() { System.***out***.println("Name : " + name ); **return** name;

}

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

}

}

## MainApp.java

**package** com.five;

**import** org.springframework.context.ApplicationContext;

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

**public class** MainApp {

**public static void** main(String[] args) {

ApplicationContext context = **new** ClassPathXmlApplicationContext("Bean.xml"); Student student = (Student) context.getBean("student");

student.getName(); student.getAge(); student.printThrowException();

}

}

## Bean.xml

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "[http://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans) xmlns:xsi = "[http://www.w3.org/2001/XMLSchema-instance"](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 student bean -->

<bean id = "student" class = "com.five.Student">

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

<property name = "age" value = "11"/>

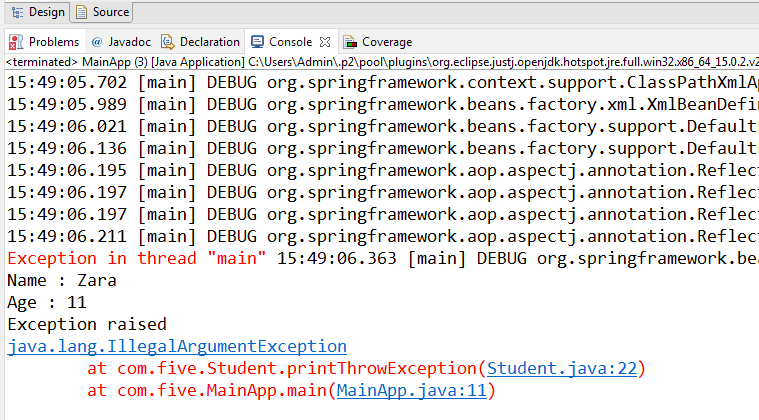
</bean>

<!-- Definition for logging aspect -->

<bean id = "logging" class = "com.five.Logging"/>

</beans>

## Output:



1. **Write a program to demonstrate Spring AOP – pointcut.**

**Logging.java**

**package** six;

**import** org.aspectj.lang.annotation.Aspect; **import** org.aspectj.lang.annotation.Pointcut; **import** org.aspectj.lang.annotation.Before;

@Aspect

**public class** Logging {

/\*\* Following is the definition for a PointCut to select

* all the methods available. So advice will be called
* for all the methods.

\*/

//@PointCut("execution(\* six.\*.\*(..))") @Pointcut("within(six.\*)")

**private void** selectAll()

{}

/\*\*

* This is the method which I would like to execute
* before a selected method execution.

\*/ @Before("selectAll()")

**public void** beforeAdvice(){ System.***out***.println("Going to setup student profile.");

}

}

## Student.java

**package** six;

**public class** Student {

**private** Integer age;

**private** String name;

**public void** setAge(Integer age) {

**this**.age = age;

}

**public** Integer getAge() { System.***out***.println("Age : " + age ); **return** age;

}

**public void** setName(String name) {

**this**.name = name;

}

**public** String getName() { System.***out***.println("Name : " + name ); **return** name;

}

**public void** printThrowException(){ System.***out***.println("Exception raised"); **thrownew** IllegalArgumentException();

}

}

## MainApp.java

**package** six;

**import** org.springframework.context.ApplicationContext;

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

**public class** MainApp {

**public static void** main(String[] args)

{

ApplicationContext context =

**new** ClassPathXmlApplicationContext("Bean.xml");

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

student.getName(); student.getAge();

}

}

## Bean.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns = "[http://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans) xmlns:xsi = "<http://www.w3.org/2001/XMLSchema-instance>" xmlns:aop = "[http://www.springframework.org/schema/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 student bean -->

<bean id = "student" class = "six.Student">

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

<property name = "age" value = "11"/>

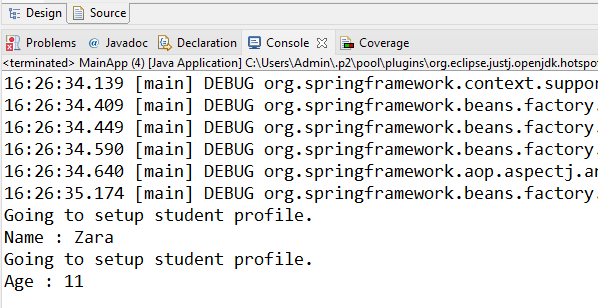
</bean>

<!-- Definition for logging aspect -->

<bean id = "logging" class = "six.Logging"/>

</beans>

## Output:



**PRACTICAL NO. 9**

**AIM / TITLE: Assignment based Spring JDBC**

1. **Write a program to insert, update and delete records from the given table.**
2. **Write a program to demonstrate PreparedStatement in Spring JdbcTemplate**
3. **Write a program in Spring JDBC to demonstrate ResultSetExtractor Interface**
4. **Write a program to demonstrate RowMapper interface to fetch the records from the database.**

**Description:**

While working with the database using plain old JDBC, it becomes cumbersome to write unnecessary code to handle exceptions, opening and closing database connections, etc. However, Spring JDBC Framework takes care of all the low-level details starting from opening the connection, prepare and execute the SQL statement, process exceptions, handle transactions and finally close the connection.

So what you have to do is just define the connection parameters and specify the SQL statement to be executed and do the required work for each iteration while fetching data from the database.

Spring JDBC provides several approaches and correspondingly different classes to interface with the database. I'm going to take classic and the most popular approach which makes use of **JdbcTemplate** class of the framework. This is the central framework class that manages all the database communication and exception handling.

## CODING:

1. **Write a program to insert, update and delete records from the given table.**

**SQL Code for creating Table**

CREATE TABLE Student(

ID INT NOT NULL AUTO\_INCREMENT, NAME VARCHAR(20) NOT NULL,

AGE INT NOT NULL, PRIMARY KEY (ID)

);

## MySpringJDBC.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns = "[http://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans) xmlns:xsi = "[http://www.w3.org/2001/XMLSchema-instance"](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/imcost?useSSL=false"/>

<property name = "username" value = "root"/>

<property name = "password" value = "1234"/>

</bean>

<!-- Definition for studentJDBCTemplate bean -->

<bean id = "studentJDBCTemplate"

class = "com.test.StudentJDBCTemplate">

<property name = "dataSource" ref = "dataSource" />

</bean>

**</**beans>

**StudentDAO.java package** com.test;

**import** java.util.List;

**import** javax.sql.DataSource;

**public interface** StudentDAO

{

// 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 Student table.

**public void** create(String name, Integer age);

// This is the method to be used to list down a record from the Student table

// corresponding to a passed student id.

**public** Student getStudent(Integer id);

//This is the method to be used to list down all the records from the Student table.

**public** List<Student> listStudents();

//This is the method to be used to delete a record from the Student table

//corresponding to a passed student id.

**public void** delete(Integer id);

//This is the method to be used to update a record into the Student table.

**public void** update(Integer id, Integer age);

}

}

**Student.java package** com.test;

**public class** Student {

**public static void** main(String[] args) {

// **TODO** Auto-generated method stub

**public class** Student {

**private** Integer id;

**private** String name; **private** Integer age; **public** Integer getId() { **return** id;

}

**public void** setId(Integer id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public void** setName(String name) {

**this**.name = name;

}

**public** Integer getAge() {

**return** age;

}

**public void** setAge(Integer age) {

**this**.age = age;

}

}

}

}

**StudentJDBCTemplate.java package** com.test;

**import** java.util.List;

**import** javax.sql.DataSource;

**import** org.springframework.jdbc.core.JdbcTemplate;

**public class** StudentJDBCTemplate **implements** StudentDAO

{

**private** DataSource dataSource;

**private** JdbcTemplate jdbcTemplateObject;

@Override

**public void** setDataSource(DataSource ds) {

// **TODO** Auto-generated method stub

**this**.dataSource=ds;

**this**.jdbcTemplateObject = **new** JdbcTemplate(dataSource);

}

@Override

**public void** create(String name, Integer age) {

// **TODO** Auto-generated method stub

String SQL = "insert into Student (name, age) values (?, ?)"; jdbcTemplateObject.update( SQL, name, age); System.***out***.println("Created Record Name = " + name + " Age = " + age); **return**;

}

@Override

**public** Student getStudent(Integer id) {

// **TODO** Auto-generated method stub

## return null;

}

@Override

**public** List<Student> listStudents() {

// **TODO** Auto-generated method stub

## return null;

}

@Override

**public void** delete(Integer id) {

// **TODO** Auto-generated method stub

String SQL = "delete from Student where id = ?"; jdbcTemplateObject.update(SQL, id); System.***out***.println("Deleted Record with ID = " + id ); **return**;

}

@Override

**public void** update(Integer id, Integer age) {

// **TODO** Auto-generated method stub

String SQL = "update Student set age = ? where id = ?"; jdbcTemplateObject.update(SQL, age, id); System.***out***.println("Updated Record with ID = " + id ); **return**;

}

}

## MyMain.java

**package** com.test;

**import** java.util.List;

**import** org.springframework.context.ApplicationContext;

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

**public class** MyMain

{

**public static void** main(String[] args)

{

ApplicationContext context = **new**

ClassPathXmlApplicationContext("MySpringJDBC.xml"); StudentJDBCTemplate studentJDBCTemplate

=(StudentJDBCTemplate)context.getBean("studentJDBCTemplate");

studentJDBCTemplate.create("King", 45);

studentJDBCTemplate.create("Queen", 42);

studentJDBCTemplate.create("Worker", 23); System.***out***.println("------Records Creation " );

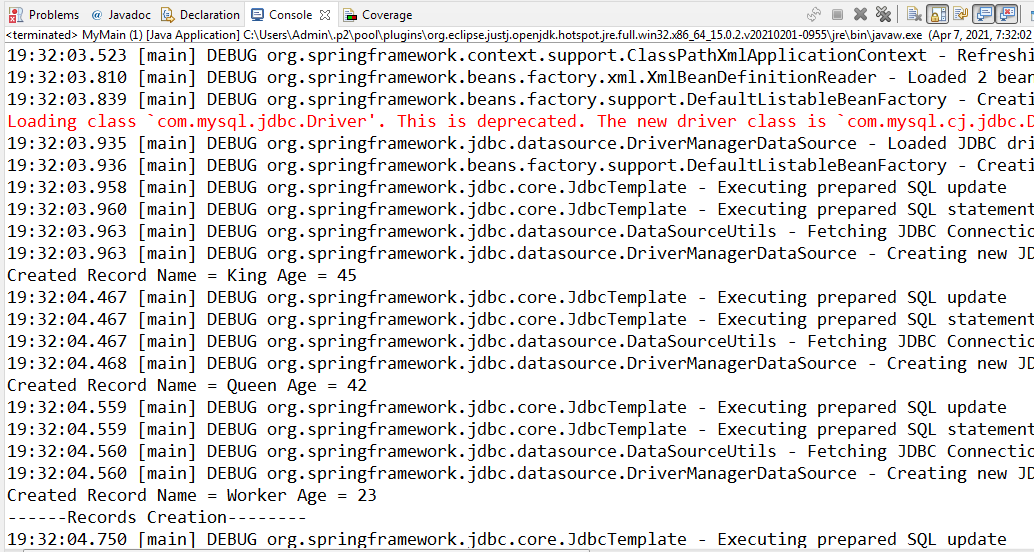
studentJDBCTemplate.delete(2); System.***out***.println("------Records Deleted " );

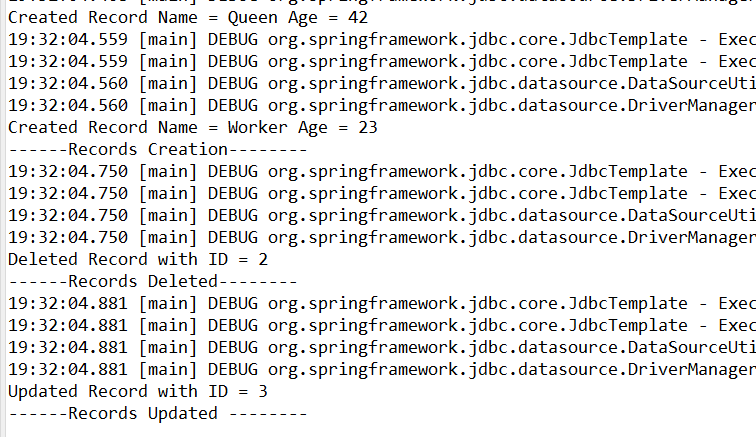
studentJDBCTemplate.update(3, 23); System.***out***.println("------Records Updated " );

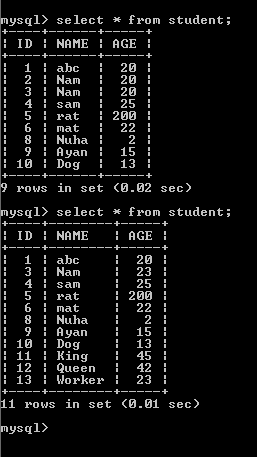
}

}

## Output:







Advanced JAVA

1. **Write a program to demonstrate PreparedStatement in Spring JdbcTemplate**

**SQL Code for creating Table**

CREATE TABLE Student(

ID INT NOT NULL AUTO\_INCREMENT, NAME VARCHAR(20) NOT NULL,

AGE INT NOT NULL, PRIMARY KEY (ID)

);

## MySpringJDBC.xml

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "[http://www.springframework.org/schema/beans"](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/imcost?useSSL=false"/>

<property name = "username" value = "root"/>

<property name = "password" value = "1234"/>

</bean>

<!-- Definition for studentJDBCTemplate bean -->

<bean id = "studentJDBCTemplate"

class = "com.test.StudentJDBCTemplate">

<property name = "dataSource" ref = "dataSource" />

</bean>

</beans>

## StudentDAO.java

package com.test; import java.util.List;

import javax.sql.DataSource;

import java.util.List;

import javax.sql.DataSource;

public interface StudentDAO

{

/\*\*

* 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 list down
* a record from the Student table corresponding
* to a passed student id.

\*/

public Student getStudent(Integer id);

}

## Student.java

**package** com.test;

**public class** Student

{

**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;

}

}

## StudentJDBCTemplate.java

**package** com.test;

**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** StudentJDBCTemplate **implements** StudentDAO {

**private** DataSource dataSource;

**private** JdbcTemplate jdbcTemplateObject;

**public void** setDataSource(DataSource dataSource) {

**this**.dataSource = dataSource;

**this**.jdbcTemplateObject = **new** JdbcTemplate(dataSource);

}

**public** Student getStudent(**final** Integer id) {

**final** String SQL = "select \* from Student where id = ? "; List <Student> students = jdbcTemplateObject.query(

SQL, **new** PreparedStatementSetter() {

**public void** setValues(PreparedStatement preparedStatement) **throws**

SQLException {

preparedStatement.setInt(1, id);

}

},

**new** StudentMapper());

**return** students.get(0);

}

}

## MyMain.java

**package** com.test;

**import** java.util.List;

**import** org.springframework.context.ApplicationContext;

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

**public class** MyMain {

**public static void** main(String[] args) { ApplicationContext context = **new**

ClassPathXmlApplicationContext("MySpringJDBC.xml"); StudentJDBCTemplate studentJDBCTemplate =

(StudentJDBCTemplate)context.getBean("studentJDBCTemplate");

Student student = studentJDBCTemplate.getStudent(5); System.***out***.print("ID : " + student.getId() ); System.***out***.println(", Age : " + student.getAge());

}

}

## StudentMapper.java

**package** com.test;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** org.springframework.jdbc.core.RowMapper;

**public class** StudentMapper **implements** RowMapper<Student> {

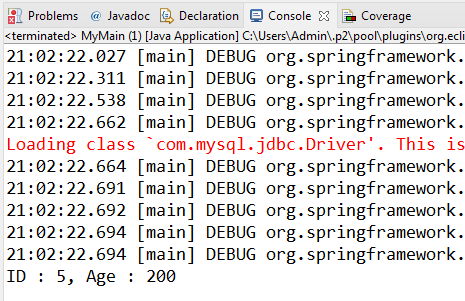
**public** Student mapRow(ResultSet rs, **int** rowNum) **throws** SQLException { Student student = **new** Student();

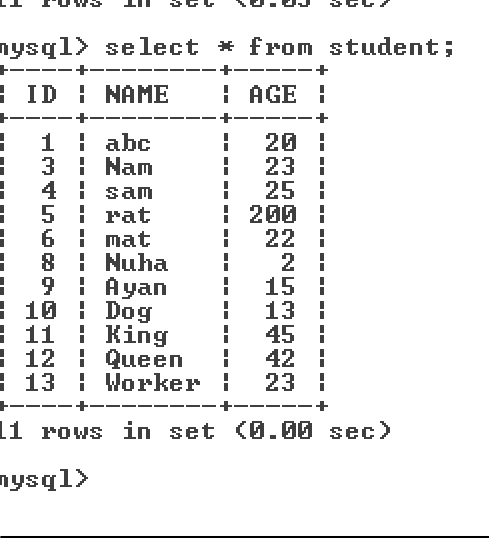
student.setId(rs.getInt("id")); student.setName(rs.getString("name")); student.setAge(rs.getInt("age")); **return** student;

}

}

## Output:





Advanced JAVA **2021**

1. **Write a program in Spring JDBC to demonstrate ResultSetExtractor Interface**

**SQL Code for creating Table**

CREATE TABLE Student(

ID INT NOT NULL AUTO\_INCREMENT, NAME VARCHAR(20) NOT NULL,

AGE INT NOT NULL, PRIMARY KEY (ID)

);

## MySpringJDBC.xml

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "[http://www.springframework.org/schema/beans"](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/imcost?useSSL=false"/>

<property name = "username" value = "root"/>

<property name = "password" value = "1234"/>

</bean>

<!-- Definition for studentJDBCTemplate bean -->

<bean id = "studentJDBCTemplate"

class = "com.test.StudentJDBCTemplate">

<property name = "dataSource" ref = "dataSource" />

</bean>

## </beans>

**StudentDAO.java**

**package** com.test;

**import** java.util.List;

**import** javax.sql.DataSource;

**public interface** StudentDAO {

/\*\*

* 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 list down
* all the records from the Student table.

\*/

**public** List<Student> listStudents();

}

**Student.java package** com.test;

**public class** Student

{

**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;

}

## }

**StudentJDBCTemplate.java package** com.test;

**import** java.util.List;

**import** java.sql.ResultSet; **import** java.sql.SQLException; **import** java.util.ArrayList; **import** javax.sql.DataSource;

**import** org.springframework.dao.DataAccessException; **import** org.springframework.jdbc.core.JdbcTemplate; **import** org.springframework.jdbc.core.ResultSetExtractor;

**public class** StudentJDBCTemplate **implements** StudentDAO {

**private** DataSource dataSource;

**private** JdbcTemplate jdbcTemplateObject;

**public void** setDataSource(DataSource dataSource) {

**this**.dataSource = dataSource;

**this**.jdbcTemplateObject = **new** JdbcTemplate(dataSource);

}

**public** List<Student> listStudents() { String SQL = "select \* from Student";

List <Student> students = jdbcTemplateObject.query(SQL,

**new** ResultSetExtractor<List<Student>>(){

**public** List<Student> extractData(

ResultSet rs) **throws** SQLException, DataAccessException {

List<Student> list = **new** ArrayList<Student>();

**while**(rs.next()){

Student student = **new** Student(); student.setId(rs.getInt("id")); student.setName(rs.getString("name")); student.setAge(rs.getInt("age"));

list.add(student);

}

**return** list;

}

});

**return** students;

}

## }

**MyMain.java**

**package** com.test;

**import** java.util.List;

**import** org.springframework.context.ApplicationContext;

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

**public class** MyMain

{

**public static void** main(String[] args)

{

ApplicationContext context = **new**

ClassPathXmlApplicationContext("MySpringJDBC.xml");

StudentJDBCTemplate studentJDBCTemplate = (StudentJDBCTemplate)context.getBean("studentJDBCTemplate");

List<Student> students = studentJDBCTemplate.listStudents();

**for**(Student student: students)

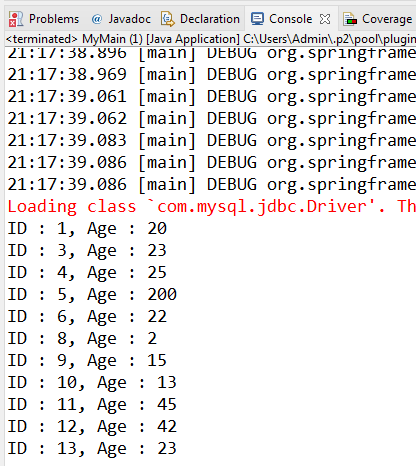
{

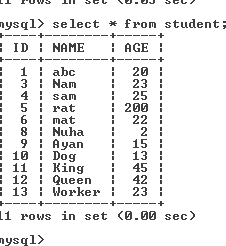
System.***out***.print("ID : " + student.getId() ); System.***out***.println(", Age : " + student.getAge());

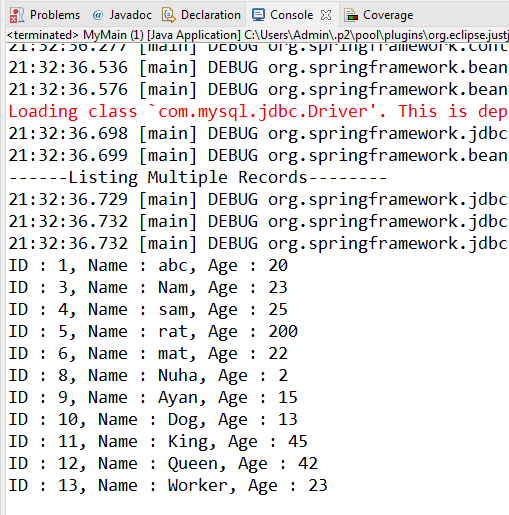
}

}

## Output:







Advanced JAVA **2021**

1. **Write a program to demonstrate RowMapper interface to fetch the records from the database.**

**SQL Code for creating Table**

CREATE TABLE Student(

ID INT NOT NULL AUTO\_INCREMENT, NAME VARCHAR(20) NOT NULL,

AGE INT NOT NULL, PRIMARY KEY (ID)

);

## MySpringJDBC.xml

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "[http://www.springframework.org/schema/beans"](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/imcost?useSSL=false"/>

<property name = "username" value = "root"/>

<property name = "password" value = "1234"/>

</bean>

<!-- Definition for studentJDBCTemplate bean -->

<bean id = "studentJDBCTemplate"

class = "com.test.StudentJDBCTemplate">

<property name = "dataSource" ref = "dataSource" />

</bean>

</beans>

## StudentDAO.java

**package** com.test;

**import** java.util.List;

**import** javax.sql.DataSource;

**public interface** StudentDAO {

/\*\*

* 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 list down
* all the records from the Student table.

\*/

**public** List<Student> listStudents();

## }

**Student.java package** com.test;

**public class** Student

{

**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;

}

## }

**StudentJDBCTemplate.java package** com.test;

**import** java.util.List;

**import** javax.sql.DataSource;

**import** org.springframework.jdbc.core.JdbcTemplate;

**public class** StudentJDBCTemplate **implements** StudentDAO

{

**private** DataSource dataSource;

**private** JdbcTemplate jdbcTemplateObject;

**public void** setDataSource(DataSource dataSource)

{

**this**.dataSource = dataSource;

**this**.jdbcTemplateObject = **new** JdbcTemplate(dataSource);

}

**public** List<Student> listStudents()

{

String SQL = "select \* from Student";

List <Student> students = jdbcTemplateObject.query(SQL, **new** StudentMapper());

**return** students;

}

## }

**MyMain.java package** com.test;

**import** java.util.List;

**import** org.springframework.context.ApplicationContext;

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

**public class** MyMain

{

**public static void** main(String[] args)

{

ApplicationContext context = **new**

ClassPathXmlApplicationContext("MySpringJDBC.xml"); StudentJDBCTemplate studentJDBCTemplate =

(StudentJDBCTemplate)context.getBean("studentJDBCTemplate");

System.***out***.println("------Listing Multiple Records " );

List<Student> students = studentJDBCTemplate.listStudents();

**for** (Student record : students)

{

System.***out***.print("ID : " + record.getId() ); System.***out***.print(", Name : " + record.getName() ); System.***out***.println(", Age : " + record.getAge());

}

}

}

**StudentMapper.java package** com.test;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** org.springframework.jdbc.core.RowMapper;

**public class** StudentMapper **implements** RowMapper<Student>

{

**public** Student mapRow(ResultSet rs, **int** rowNum) **throws** SQLException

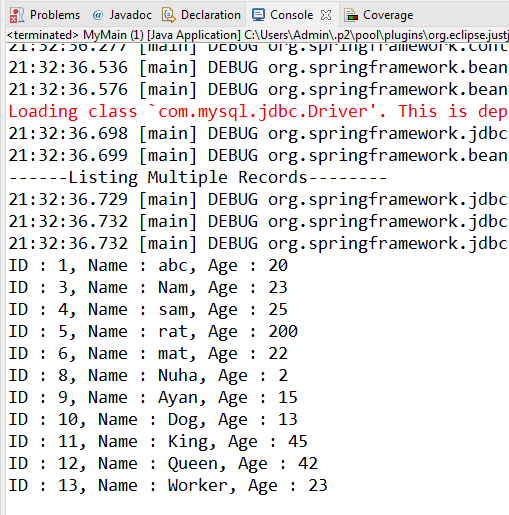
{

Student student = **new** Student(); student.setId(rs.getInt("id")); student.setName(rs.getString("name")); student.setAge(rs.getInt("age")); **return** student;

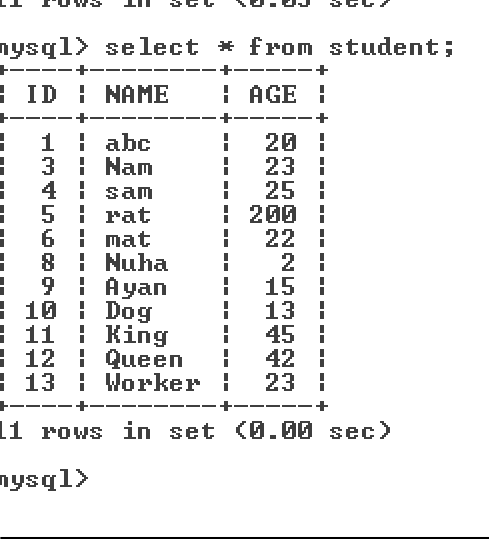
}

## }

**Output:**



Advanced JAVA



**PRACTICAL NO. 10**

**AIM / TITLE: Assignment based Spring Boot and RESTful Web Services**

* 1. **Write a program to create a simple Spring Boot application that prints a message.**
  2. **Write a program to demonstrate RESTful Web Services with spring boot. Description:**

REST stands for **REpresentational State Transfer**. It is developed by **Roy Thomas Fielding**, who also developed HTTP. The main goal of RESTful web services is to make web services **more effective**. RESTful web services try to define services using the different concepts that are already present in HTTP. REST is an **architectural approach**, not a protocol.

It does not define the standard message exchange format. We can build REST services with both XML and JSON. JSON is more popular format with REST. The **key abstraction** is a resource in REST. A resource can be anything. It can be accessed through a **Uniform Resource Identifier (URI)**. For example:

## CODING:

1. **Write a program to create a simple Spring Boot application that prints a message.**

**MondayBoot/pom.xml**

<project xmlns=*"*[*http://maven.apache.org/POM/4.0.0"*](http://maven.apache.org/POM/4.0.0)xmlns:xsi=*"*[*http://www.w3.org/2001/XMLSchema-instance"*](http://www.w3.org/2001/XMLSchema-instance)xsi:schemaLocation=*"*[*http://maven.apache.org/POM/4.0.0*](http://maven.apache.org/POM/4.0.0) *https://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>com.test</groupId>

<artifactId>MondayBoot</artifactId>

<version>0.0.1-SNAPSHOT</version>

<description>spring boot</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.3.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<version>2.2.2.RELEASE</version>

</dependency>

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

## MyMain.java

**package** com.abc;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication; **import** org.springframework.context.ApplicationContext; @SpringBootApplication

**public class** MyMain

{

**public static void** main(String[] args)

{

// **TODO** Auto-generated method stub

ApplicationContext con= SpringApplication.*run*(MyMain.**class**, args);

}

}

## MyController.java

**package** com.abc;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController; @RestController

**public class** MyController

{

@RequestMapping("/Home")

**public** String welcome()

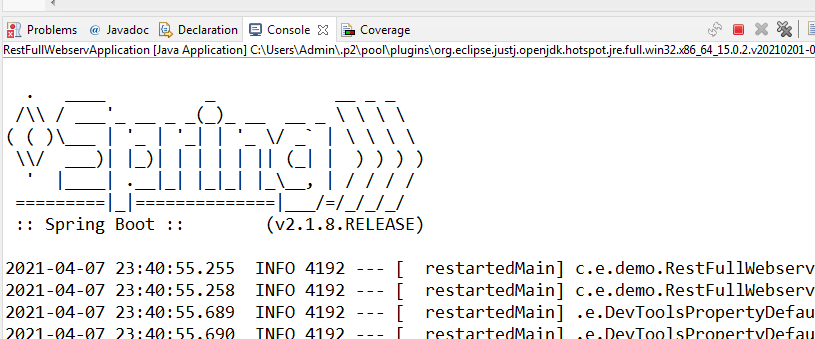
{

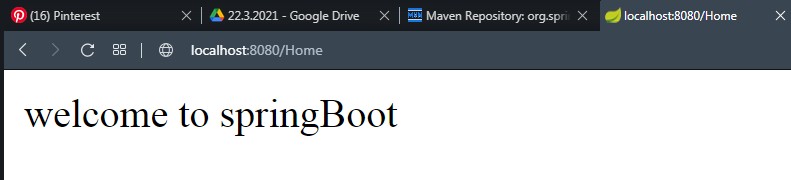
}

## Output:

**return** "welcome to springBoot";

}





Advanced JAVA **2021**

## Write a program to demonstrate RESTful Web Services with spring boot.

**HelloRESET/pom.xml**

<?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>2.4.4</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>net.codejava</groupId>

<artifactId>HelloREST</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>HelloREST</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> HelloRestApplication.java package net.codejava.ws;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class HelloRestApplication {

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

}

}

## HelloController.java

**package** net.codejava.ws;

**import** org.springframework.web.bind.annotation.RequestMapping; **import** org.springframework.web.bind.annotation.RequestParam; **import** org.springframework.web.bind.annotation.RestController;

@RestController

**public class** HelloController

{

@RequestMapping("/hello")

**public** String hello(@RequestParam(name = "name", defaultValue = "Earth") String name)

{

**return** "Hello World RESTFull with Spring boot Name:" + name ;

}

}

## Output:

