MCA Department

MCAL12 Advanced Java Lab Journal

Assignment No - 1

1.1 WRITE A JAVA PROGRAM TO DEMONSTRATE A GENERIC CLASS

JAVA GENERICS

Generics add stability to your code by making more of your bugs detectable at compile time.
Generics enable types (classes and interfaces) to be parameters when defining classes, interfaces and methods.
A Generic Type is a generic class or interface that is parameterized over types.
The most commonly used type parameter names are:
E – Element
K - Key
N - Number
T - Type
V – Value

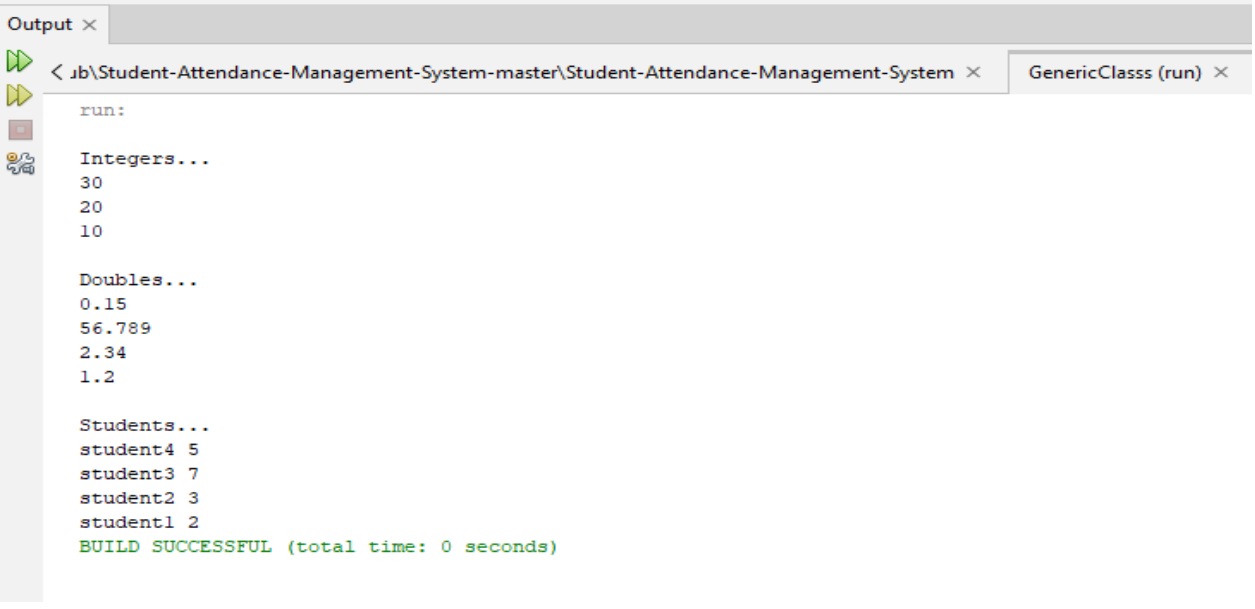
CODE:

```
/**
 GenericClass demonstrates a generic Stack implementation.
  @param <E> The type of elements held in this collection.
 */

package genericclasss;

public class GenericClass<E> {
   E[] a;
   int top;

   // Constructor
   GenericClass() {
      a = (E[]) new Object[100]; // Create a generic array
      top = -1;
   }

   // Push method to add an element
   void push(E data) {
      a[++top] = data;
   }

   // Pop method to remove and return an element
   E pop() {
      return a[top--];
```

```
}

// Check if the stack has elements
boolean hasElements() {
   return top != -1;
}

public static void main(String[] args) {
   // Test the GenericClass with different data types
   GenericClass<Integer> si = new GenericClass<>(); // Stack of Integer objects
   GenericClass<Double> sd = new GenericClass<>();  // Stack of Double objects
   GenericClass<Student> ss = new GenericClass<>();  // Stack of Student objects

   // Pushing elements into Integer stack
   si.push(10);
   si.push(20);
   si.push(30);

   // Pushing elements into Double stack
   sd.push(1.2);
   sd.push(2.34);
   sd.push(56.789);
   sd.push(0.15);

   // Pushing Student objects into the Student stack
   ss.push(new Student("student1", 2));
   ss.push(new Student("student2", 3));
   ss.push(new Student("student3", 7));
   ss.push(new Student("student4", 5));

   // Displaying Integer stack elements
   System.out.println("\nIntegers...");
   while (si.hasElements()) {
      System.out.println(si.pop());
   }

   // Displaying Double stack elements
   System.out.println("\nDoubles...");
   while (sd.hasElements()) {
      System.out.println(sd.pop());
   }

   // Displaying Student stack elements
   System.out.println("\nStudents...");
   while (ss.hasElements()) {
```

```
            System.out.println(ss.pop());
        }
    }

    // Student class for holding student information
    static class Student {
        String name;
        int standard;

        // Constructor
        Student(String n, int s) {
            name = n;
            standard = s;
        }

        // Override toString method to display student information
        public String toString() {
            return name + " " + standard;
        }
    }
}
```

OUTPUT :



1.2. WRITE A JAVA PROGRAM TO DEMONSTRATE GENERIC METHODS.

GENERICS METHOD

MCA Department

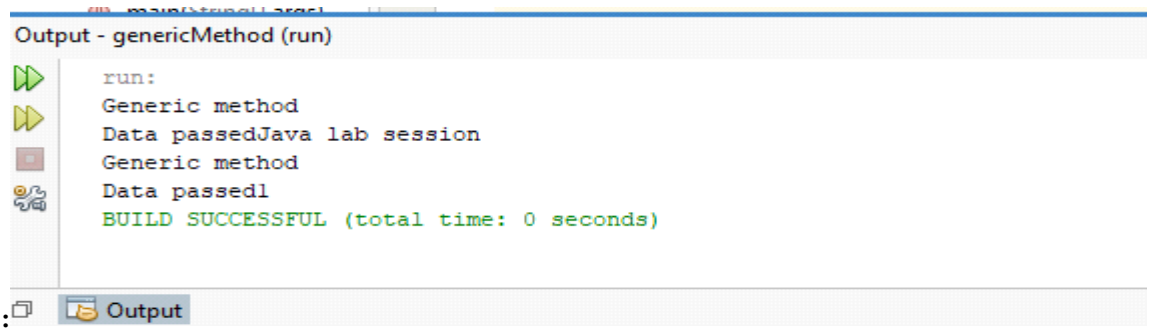
MCAL12 Advanced Java Lab Journal

Generic Methods are methods that introduce their own type parameters. This is similar to declaring a generic type, but the type parameter's scope is limited to the method where it is declared.

CODE :

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
 */
package genericmethod;

public class GenericMethod {

    public static void main(String args []) {
        // TODO code application logic here
        Democlass objDemoclass = new Democlass();
        objDemoclass.<String>genericMethod("Java lab session");
        objDemoclass.<Integer>genericMethod(1);
    }

}
class Democlass
{
    public <T> void genericMethod(T data)
    {
        System.out.println("Generic method");
        System.out.println("Data passed"+data);
    }
}
```

OUTPUT



:

1.3. WRITE A JAVA PROGRAM TO DEMONSTRATE WILDCARDS IN JAVA GENERICS.

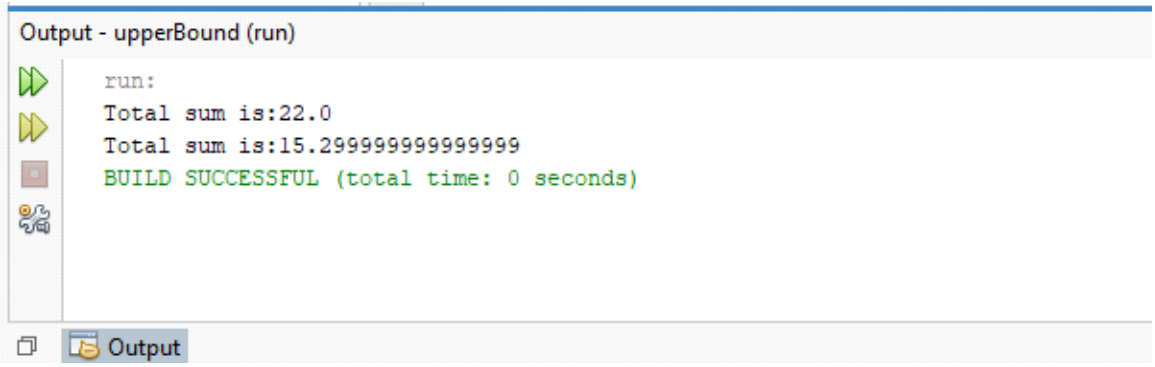
Upper Bounded Wildcard

You can use an upper bounded wildcard to relax the restrictions on a variable. For example, say you want to write a method that works on List, List, and List; you can achieve this by using an upper bounded wildcard.

CODE :

```
package upperbound;

import java.util.*;
// import java.util.Arrays;
// import java.util.List;

/***/
 public class UpperBound{


   public static void main(String[] args) {
      // TODO code application logic here

      List<Integer>list1= Arrays.asList(3,5,6,8);

      System.out.println("Total sum is:" + sum(list1));
      List<Double> list2 = Arrays.asList(3.1,5.1,7.1);

      System.out.println("Total sum is:"+sum(list2));

   }
   private static double sum(List<? extends Number> list)
   {
      double sum=0.0;
      for(Number i : list)
      {
         sum+=i.doubleValue();
      }
      return sum;
   } }
```

OUTPUT :



Ø Lower Bounded

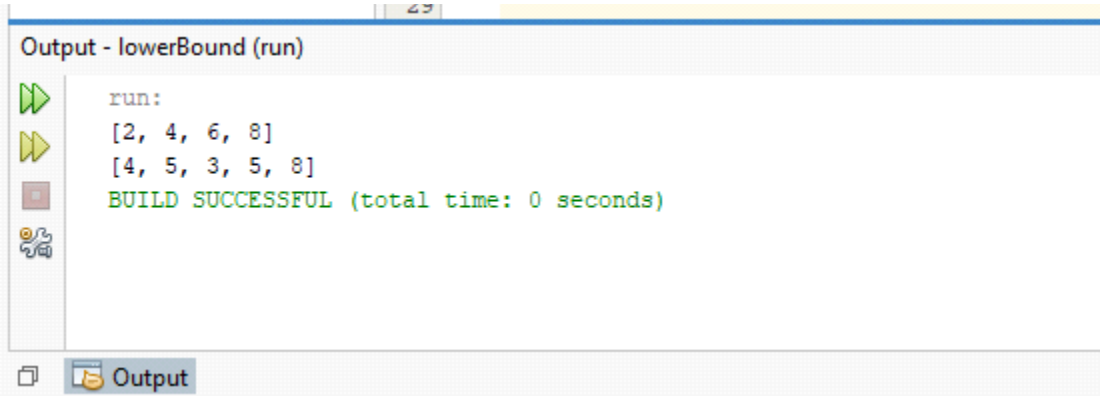
Wildcard Lower bounded wildcard restricts the unknown type to be a specific type or a super type of that type. A lower bounded wildcard is expressed using the wildcard character ('?'), following by the super keyword, followed by its lower bound:

CODE :

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
 */
package lowerbound;
import java.util.*;
// import java.util.List;
// import java.util.Arrays;

public class LowerBound {

   /** */
   public static void main(String[] args) {
      // TODO code application logic here
      List<Integer> list1 = Arrays.asList(2, 4, 6, 8);

      printOnlyIntegerClassorSuperClass(list1);

      List<Number> list2 = Arrays.asList(4, 5, 3, 5,8);

      printOnlyIntegerClassorSuperClass(list2);
   }
   public static void printOnlyIntegerClassorSuperClass(List<? super Integer>list)
```

```
  {
     System.out.println(list);
  }

}
```

OUTPUT :



Ø Unbounded Wildcard

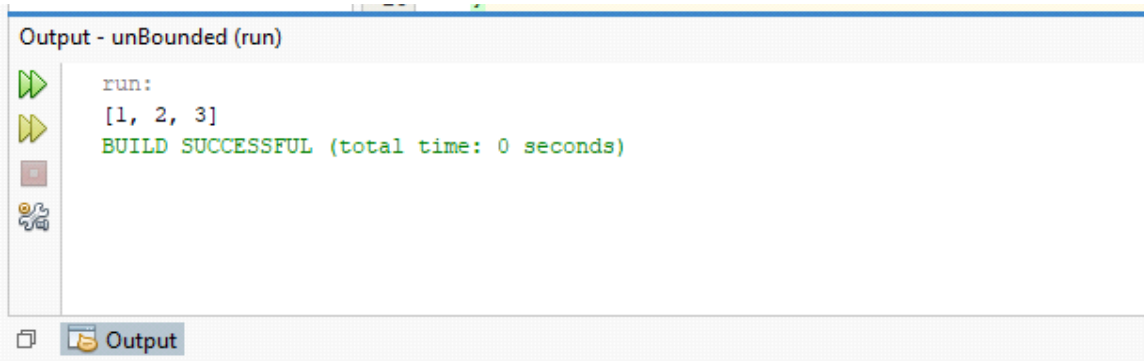
The unbounded wildcard type is specified using the wildcard character (?), for example, List. This is called a list of unknown type. When the code is using methods in the generic class that don't depend on the type parameter. For example, List.size or List.clear. In fact, Class is so often used because most of the methods in Class do not depend on T.

CODE:

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

public class UnBounded {
   public static void main(String[] args) {
      List<Integer>list1=Arrays.asList(1,2,3);
      List<Double>list=Arrays.asList(1.1,2.2,3.3);

      printlist(list1);
      // printlist(list2);

   }
   private static void printlist(List<?>list)
```

```
  {
  System.out.println(list);
}
}
```

OUTPUT :



MCA Department

MCAL12 Advanced Java Lab Journal

Assignment No - 2

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**

LIST INTERFACE

An ordered collection (also known as a sequence). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.

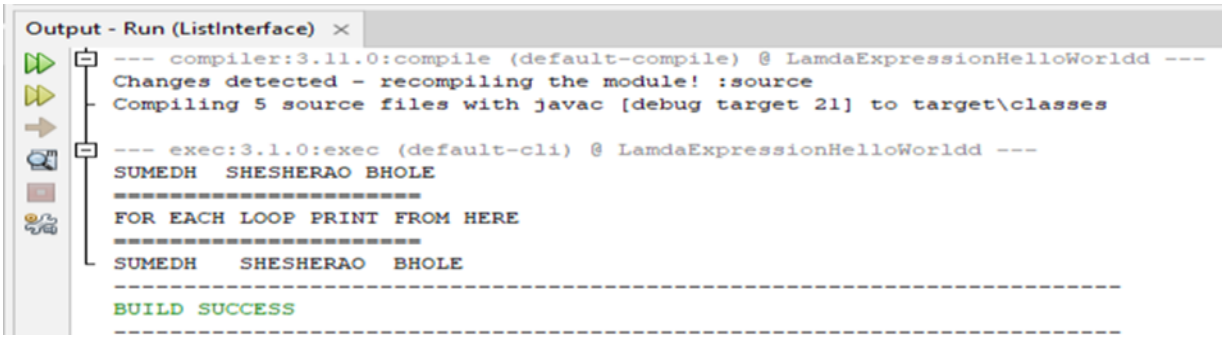
Unlike sets, lists typically allow duplicate elements. More formally, lists typically allow pairs of elements e1 and e2 such that e1.equals(e2), and they typically allow multiple null elements if they allow null elements at all. It is not inconceivable that someone might wish to implement a list that prohibits duplicates, by throwing runtime exceptions when the user attempts to insert them, but we expect this usage to be rare.

The List interface places additional stipulations, beyond those specified in the Collection interface, on the contracts of the iterator, add, remove, equals, and Hash Code methods. Declarations for other inherited methods are also included here for convenience.

CODE:

```
package AssingmentThree;
import java.util.*;
public class ListInterface {
   public static void main(String args[])
   {
      List<String> name = new ArrayList<>()
      name.add("SUMEDH  ");
      name.add("BHOLE  ");
      name.add(1, "SHESHERAO ");
      for (int i = 0; i < name.size(); i++)
      {
         System.out.print(name.get(i) + "");
      }
      System.out.println();
      System.out.println("======================");
      System.out.println("FOR EACH LOOP PRINT FROM HERE ");
      System.out.println("======================");
      for (String str : name)
      System.out.print(str + " ");
   }
}
```

OUTPUT



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 / BACKWARD DIRECTION.

List Iteration

Java provides an interface Iterator to iterate over the Collections, such as List, Map, etc. It contains two

key methods next() and hasNaxt() that allows us to perform an iteration over the List. next(): The next() method perform the iteration in forward order. It returns the next element in the List. It throws NoSuchElementException if the iteration does not contain the next element in the List. This method may be called repeatedly to iterate through the list, or intermixed with calls to previous() to go back and forth.

hasNext(): The hasNext() method helps us to find the last element of the List. It checks if the List has the next element or not. If the hasNext() method gets the element during traversing in the forward direction, returns true, else returns false and terminate the execution.

CODE :

```
package AssingmentThree;
import java.util.*;
public class ListInterfacePrintListForwardReverse {
   public static void main(String a[]){
   ListIterator<String> litr = null;
   List<String> names = new ArrayList<String>();
   names.add("TATA");
   names.add("TOYOTA");
   names.add("MAHINDRA");
```

```
    names.add("BMW");
    names.add("AUDI");
    names.add("KIA");

    //Obtaining list iterator
    litr=names.listIterator();

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

OUTPUT:

```
Traversing in forward direction:
================================
TATA
TOYOTA
MAHINDRA
BMW
AUDI
KIA

Traversing in backward direction:
================================
KIA
AUDI
BMW
MAHINDRA
TOYOTA
TATA
```

MCA Department

MCAL12 Advanced Java Lab Journal

Assignment No – 3

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.**

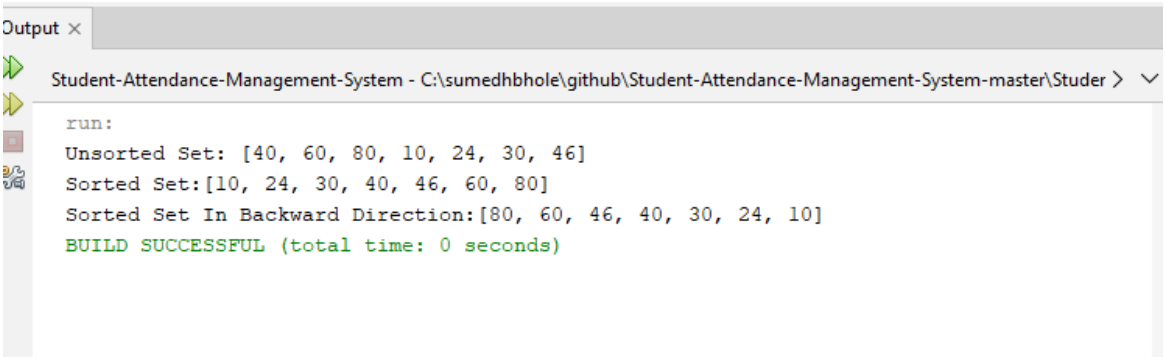
Set Interface

The Set interface places additional stipulations, beyond those inherited from the Collection interface, on the contracts of all constructors and on the contracts of the add, equals and hashCode methods. Declarations for other inherited methods are also included here for convenience. E - the type of elements maintained by this set.

CODE :

```
/* 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. */

package javaassingmentfour;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashSet;
import java.util.LinkedHashSet;
import java.util.List;

public class SetInterfaceListItemsTypeString {
   public static void main(String[] args)
{
   HashSet<Integer> evenNumSet = new LinkedHashSet<>(
   Arrays.asList(40,60,80,10,24,30,46) );
   System.out.println("Unsorted Set: " + evenNumSet);

   List<Integer> numList = new ArrayList<Integer>(evenNumSet);
   Collections.sort(numList);
   evenNumSet = new LinkedHashSet<>(numList);
   System.out.println("Sorted Set:" + evenNumSet);
   Collections.reverse(numList);
   System.out.println("Sorted Set In Backward Direction:"+numList);
   }
}
```

OUTPUT



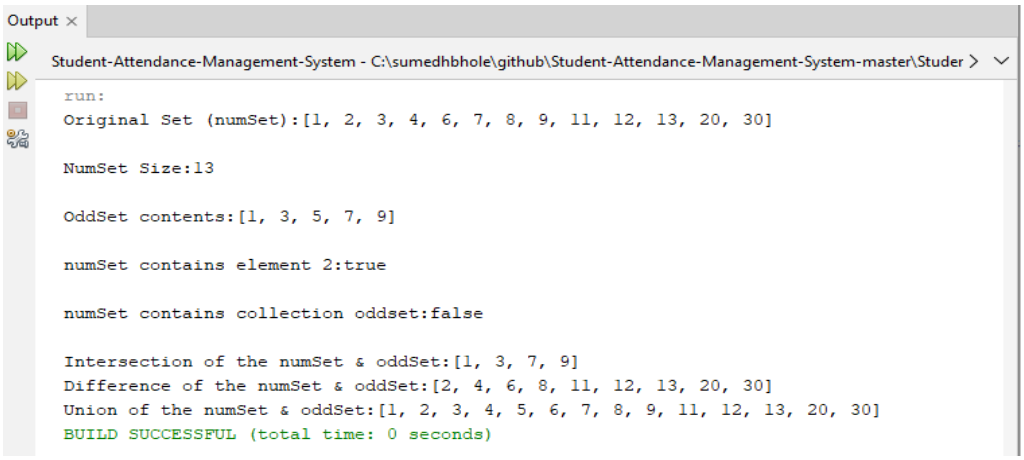
2. WRITE A JAVA PROGRAM USING SET INTERFACE CONTAINING LIST OF ITEMS AND PERFORM THE FOLLOWING OPERATIONS:
A. ADD ITEMS IN THE SET.
B. INSERT ITEMS OF ONE SET IN TO OTHER SET.
C. REMOVE ITEMS FROM THE SET
D. SEARCH THE SPECIFIED ITEM IN THE SET

CODE :

```
/*  Write a Java program using Set interface containing list of items and perform the        operations:*/
package javaassingmentfour;
import java.util.*;
public class SetInterfacePerformOperations {
   public static void main(String args[])
   {
      Set<Integer> numSet = new HashSet<Integer>();
      numSet.add(13);
      numSet.addAll(Arrays.asList(new Integer[] { 1,6,4,7,3,9,8,2,12,11,20,30 }))
      System.out.println("Original Set (numSet):" + numSet);
      System.out.println("\nNumSet Size:" + numSet.size());
      Set<Integer> oddSet = new HashSet<Integer>();
      oddSet.addAll(Arrays.asList(new Integer[] {1, 3, 7, 5, 9}));
      System.out.println("\nOddSet contents:" + oddSet);
      System.out.println("\nnumSet contains element 2:" + numSet.contains(3));
      System.out.println("\nnumSet contains collection oddset:" +
      numSet.containsAll(oddSet));
      Set<Integer> set_intersection = new HashSet<Integer>(numSet);
      set_intersection.retainAll(oddSet);
      System.out.print("\nIntersection of the numSet & oddSet:");
      System.out.println(set_intersection);
      Set<Integer> set_difference = new HashSet<Integer>(numSet);
      set_difference.removeAll(oddSet);
      System.out.print("Difference of the numSet & oddSet:");
      System.out.println(set_difference);
      Set<Integer> set_union = new HashSet<Integer>(numSet);
```

```
    set_union.addAll(oddSet);
    System.out.print("Union of the numSet & oddSet:");
    System.out.println(set_union);
  }
```

OUTPUT::



MCA Department

MCAL12 Advanced Java Lab Journal

Assignment No – 4

1. **Write a Java program using Map interface containing list of items having keys and associated values and perform the following operations:**

a. Add items in the map.

b. Remove items from the map

c. Search specific key from the map

d. Get value of the specified key

e. Insert map elements of one map in to other map.

f. Print all keys and values of the map.

Java Map Interface

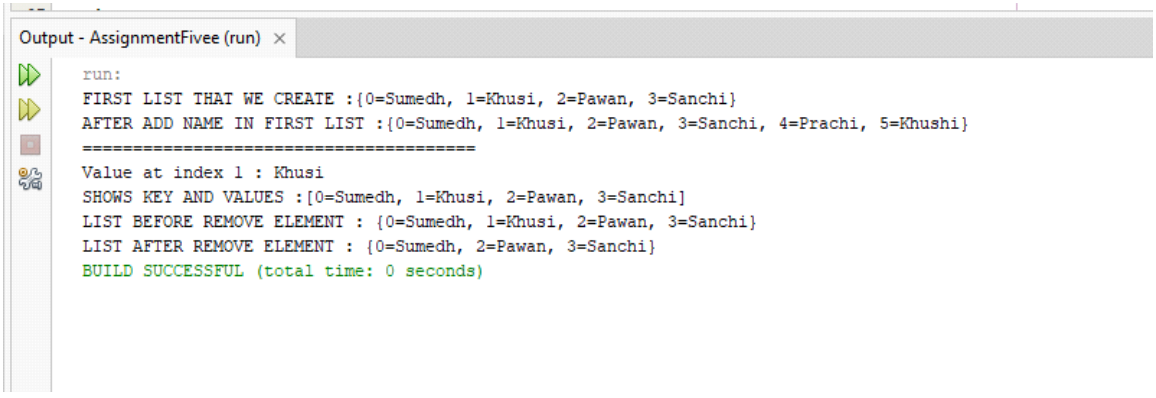
An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value. This interface takes the place of the Dictionary class, which was a totally abstract class rather than an interface. The Map interface provides three collection views, which allow a map's contents to be viewed as a set of keys, collection of values, or set of key-value mappings. The order of a map is defined as the order in which the iterators on the map's collection views return their elements. Some map implementations, like the TreeMap class, make specific guarantees as to their order; others, like the HashMap class, do not

CODE :

```
package assignmentfivee;

import java.util.*;

public class ListMapInterfaceCurd {

   public static void main(String args []){

      HashMap<Integer,String>idName=new HashMap<Integer,String>();

      idName.put(0,"Sumedh");

      idName.put(01,"Khusi");

      idName.put(02,"Pawan");

      idName.put(03,"Sanchi");

      // Here we create another name which we want to and in existing map list

      HashMap<Integer,String>idNametwo=new HashMap<Integer,String>();
```

```
idNametwo.put(04,"Prachi");

idNametwo.put(05,"Khushi")

// here we add new id and name into maplist

idNametwo.putAll(idName);

// Display the map list

System.out.println("FIRST LIST THAT WE CREATE :"+idName);

System.out.println("AFTER ADD NAME IN FIRST LIST :"+idNametwo);

System.out.println("=====================================");

String value=idName.get(1)

System.out.println("Value at index 1 : "+value);

System.out.println("SHOWS KEY AND VALUES :"+idName.entrySet());

System.out.println("LIST BEFORE REMOVE ELEMENT : "+idName);

// REMOVE KEY VALUE PAIR FROM LIST

idName.remove(1);

System.out.println("LIST AFTER REMOVE ELEMENT : "+idName)

}}
```

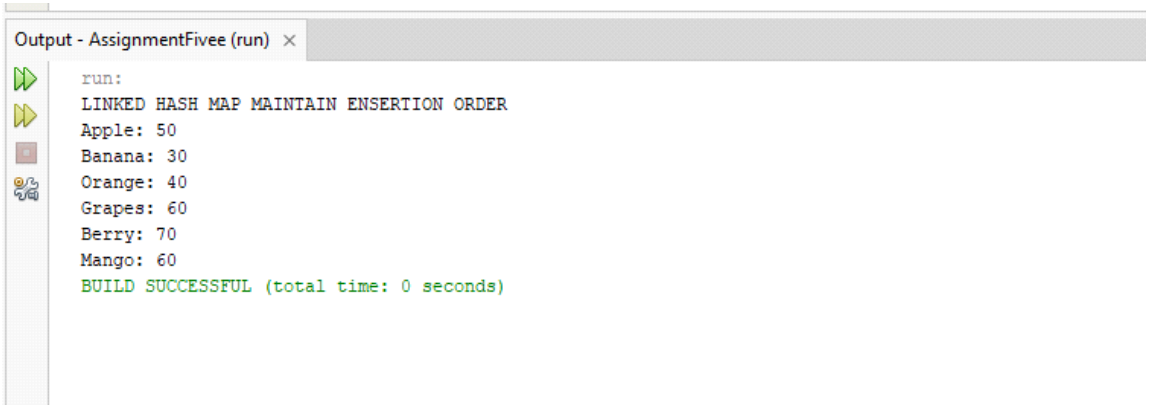
OUTPUT:



/* Using LinkedHashMap to maintain insertion order */

```
package assignmentfivee;

import java.util.*;

import java.util.LinkedHashMap;

import java.util.Map;

public class LinkedHasMapExample {

public static void main(String[] args) {

   // Create a LinkedHashMap

   Map<String, Integer> keyValuesLinkedHashMap = new LinkedHashMap<>();

   // Add key-value pairs to the LinkedHashMap

   keyValuesLinkedHashMap.put("Apple", 50);

   keyValuesLinkedHashMap.put("Banana", 30);

   keyValuesLinkedHashMap.put("Orange", 40);

   keyValuesLinkedHashMap.put("Grapes", 60);

   keyValuesLinkedHashMap.put("Berry", 70);

   keyValuesLinkedHashMap.put("Mango", 60);

   // LinkedHashMap maintains insertion order

   System.out.println("LINKED HASH MAP MAINTAIN ENSERTION ORDER");

   for (Map.Entry<String, Integer> entry : keyValuesLinkedHashMap.entrySet()) {

   System.out.println(entry.getKey() + ": " + entry.getValue());

 }

 }

}
```

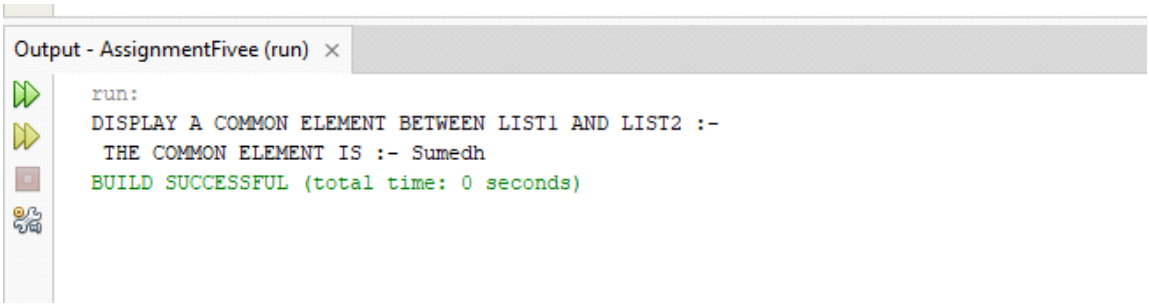
OUTPUT:



/* Finding common elements between two maps */

```
package assignmentfivee;

import java.util.*;

import java.util.Map;

public class FindCommonElementOfTwoMaps {

    public static void main(String[] args){

        // Create first HashMap

        Map<String, Integer> list1 = new HashMap<>();

        list1.put("Sumedh", 01);

        list1.put("Khusi", 02);

        list1.put("Sanchi", 04);

        // Create second HashMap

        Map<String, Integer> list2 = new HashMap<>();

        list2.put("Pawan", 88);

        list2.put("Sumedh", 92);

        list2.put("Pankaj", 80);
```

```
// Find and display common keys between the two maps
        System.out.println("DISPLAY A COMMON ELEMENT BETWEEN LIST1 AND LIST2 :- ");
        for (String key : list1.keySet()) {
        if (list2.containsKey(key)) {
        System.out.println(" THE COMMON ELEMENT IS :- " +key);
    }
    }
}
}
```

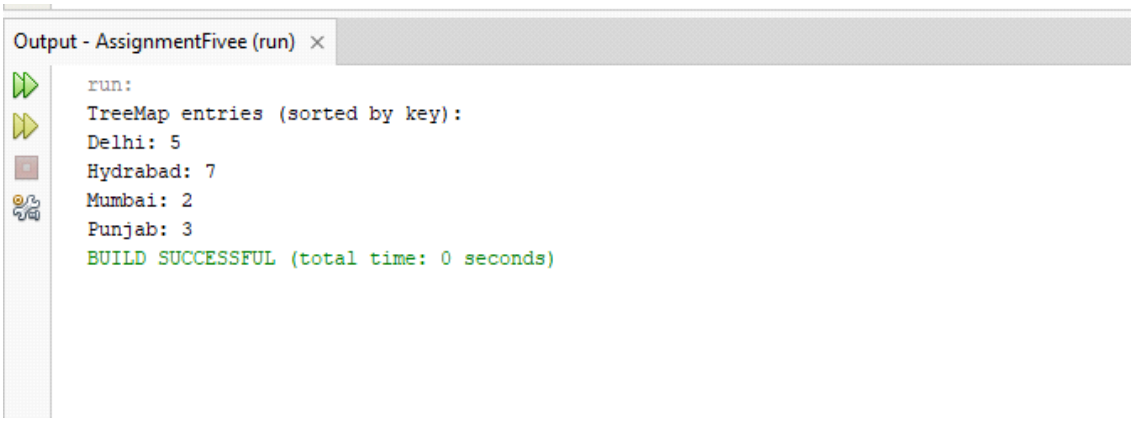


/* Using TreeMap to store and display entries in a sorted order */

```
package assignmentfivee;
import java.util.*;
import java.util.TreeMap;
public class TreeMapExample{
    public static void main(String []args){
        // Create a TreeMap
// Create a TreeMap
 Map<String, Integer> treeMap = new TreeMap<>();
 // Add key-value pairs to the TreeMap
```

```
treeMap.put("Mumbai", 2);

treeMap.put("Delhi", 5);

treeMap.put("Punjab", 3);

treeMap.put("Hydrabad", 7);

// TreeMap automatically sorts entries by keys

System.out.println("TreeMap entries (sorted by key):");

for (Map.Entry<String, Integer> entry : treeMap.entrySet()) {

System.out.println(entry.getKey() + ": " + entry.getValue());

}

}

}
```

OUTPUT:



MCA Department

MCAL12 Advanced Java Lab Journal

Assignment No – 5

1. WRITE A PROGRAM USING LAMBDA EXPRESSION TO PRINT " HELLO WORLD ".

Java Lambda Expressions

One issue with anonymous classes is that if the implementation of your anonymous class is very simple, such as an interface that contains only one method, then the syntax of anonymous classes may seem unwieldy and unclear. In these cases, you're usually trying to pass functionality as an argument to another method, such as what action should be taken when someone clicks a button. Lambda expressions enable you to do this, to treat functionality as method argument, or code as data.

The previous section, Anonymous Classes, shows you how to implement a base class without giving it a name. Although this is often more concise than a named class, for classes with only one method,even an anonymous class seems a bit excessive and cumbersome. Lambda expressions let you express instances of single-method classes more compactly.

CODE:

```
/** Write a program using Lambda Expression to print "Hello World". */
package com.JavaAssinmentSecond;
public class LamdaExpressionHelloWorldd {
    interface HelloWorld
    {
       String hello (String name);
    }
    //public class Program
    // {
       public static void main(String[] args)
       {
          HelloWorld print = (String name) ->(name);
          System.out.println("======================");
          System.out.println(" HELLO IM SUMEDH BHOLE ");
          System.out.println("======================");
       }
}
```

OUTPUT :

```
--- exec:3.1.0:exec (default-cli) @ LamdaExpressionHelloWorldd ---
=======================
 HELLO IM SUMEDH BHOLE
=======================
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  5.318 s
Finished at: 2024-10-16T23:18:12+05:30
```

2 WRITE A JAVA PROGRAM USING LAMBDA EXPRESSION TO CONCATENATE TWO STRINGS.

CODE :

```
/** Write a Java program using Lambda Expression to concatenate two strings. */
package com.JavaAssinmentSecond;
import java.util.*;
public class LamdaExConcatenateTwoString {
        interface Concatenate {
       String add(String a, String b);
    }
    public static void main(String[] args) {
       Concatenate objConcatenate = (String a, String b)->(a+""+b);
       System.out.println("============================");
       System.out.println("WELCOME TO ADVANCE "+ objConcatenate.add("JAVA ","PROGRAMMING"));
    }

}
```

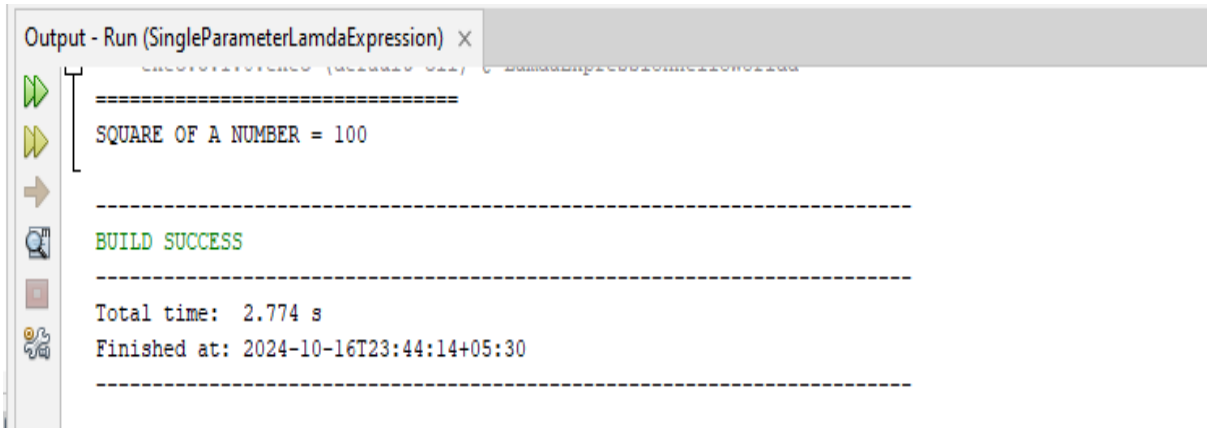
OUTPUT :

2 WRITE A PROGRAM USING LAMBDA EXPRESSION WITH SINGLE PARAMETERS.

CODE:

```
/** WAP using Lambda Expression with single parameters. */
package com.JavaAssinmentSecond;
import java.util.*;
   interface Square{
      int squre(int num);
   }
   public class SingleParameterLamdaExpression {
   public static void main(String[] args)
   {
      Square squr = (num)->(num*num);
      System.out.println("================================");
      System.out.println("SQUARE OF A NUMBER = "+squr.squre(10)+"\n");
   }
}
```

OUTPUT :



2 WRITE A JAVA PROGRAM USING LAMBDA EXPRESSION WITH MULTIPLE PARAMETERS TO ADD TWO NUMBERS.

CODE :

```
/*  Write a Java program using Lambda Expression with multiple parameters to add two numbers.*/
package com.JavaAssinmentSecond;
import java.util.*;
interface Addition{
   int addition(int num1, int num2);
}
interface Multiplication{
   int multiply (int num1, int num2);
}
interface Subtraction {
   int subtraction(int num1, int num2);
}
interface Division{
   int division(int num1, int num2);
}
public class LambdaExMultiParaAddTwoNum {
   public static void main(String[]args){
      Addition add=(int num1, int num2)->(num1+num2);
      Multiplication multy = (int num1, int num2)->(num1*num2);
      Subtraction sub=(int num1, int num2)->(num1-num2);
      Division div=(int num1, int num2)->(num1/num2);

      System.out.println("=========================================");
      System.out.println("ADDITION OF TWO NUMBER = "+add.addition(10,20));
      System.out.println("MULTIPLICATION OF TWO NUMBER = "+multy.multiply(10,20));
      System.out.println("SUBTRACTION OF TWO NUMBER = "+sub.subtraction(20,10));
      System.out.println("DIVISION OF TWO NUMBER = "+div.division(2,2));

   } }
```

OUTPUT

```
--- exec:3.1.0:exec (default-cli) @ LamdaExpressionHelloWorldd ---
=========================================
ADDITION OF TWO NUMBER = 30
MULTIPLICATION OF TWO NUMBER = 200
SUBTRACTION OF TWO NUMBER = 10
DIVISION OF TWO NUMBER = 1
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  3.450 s
Finished at: 2024-10-17T00:17:00+05:30
------------------------------------------------------------------------
```

2 WRITE A JAVA PROGRAM USING LAMBDA EXPRESSION TO CALCULATE THE FOLLOWING.

A) CONVERT FAHRENHEIT TO CELSIUS.

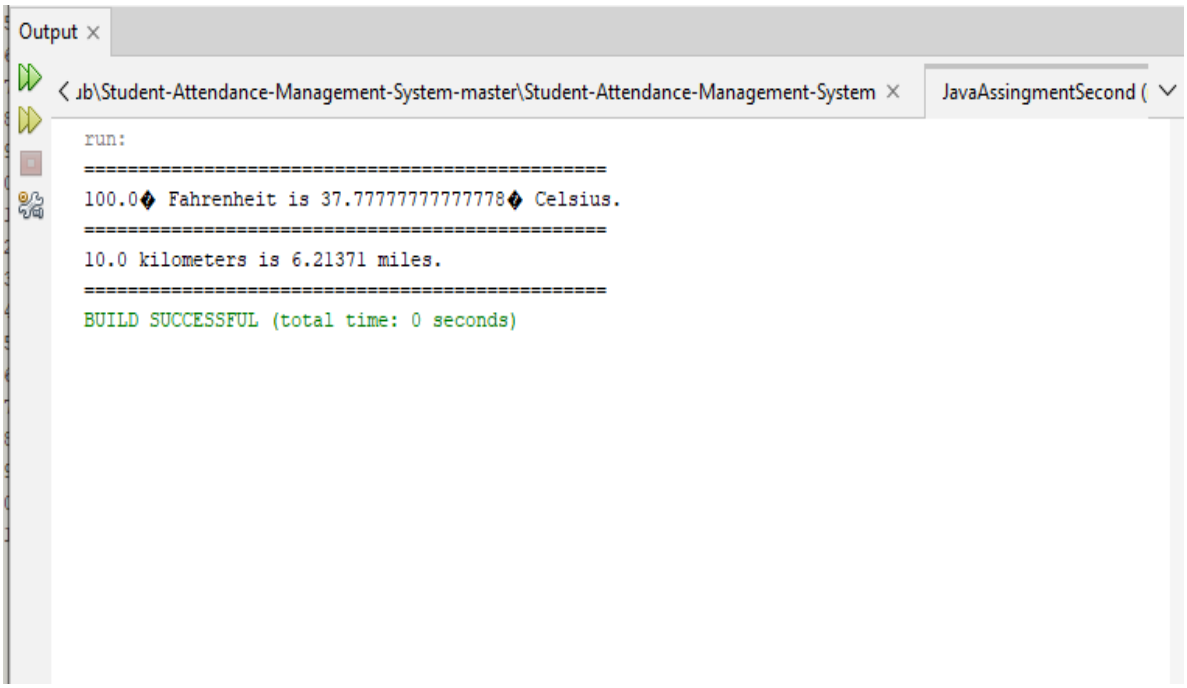
B) CONVERT KILOMETERS TO MILES.

CODE :

```
/*
   Write a Java program using Lambda Expression to calculate the following:
   1) Convert Fahrenheit to Celsius. 2)Convert Kilometers to Miles
 */
package LambdaExpression;

public class ConvertFahrenheitToCelsiusConvertKmToMiles {

   @FunctionalInterface
   interface FahrenheitToCelsius {
      double convert(double fahrenheit);
   }

   // Functional Interface for converting Kilometers to Miles
   @FunctionalInterface
   interface KilometersToMiles {
      double convert(double kilometers);
   }

   public static void main(String[] args) {
      // Lambda expression to convert Fahrenheit to Celsius
      FahrenheitToCelsius fahrenheitToCelsius = (fahrenheit) -> (fahrenheit - 32) * 5 / 9;
      // Lambda expression to convert Kilometers to Miles
      KilometersToMiles kilometersToMiles = (kilometers) -> kilometers * 0.621371;
      // Example Usage:
      double fahrenheit = 100;
```

```
    double celsius = fahrenheitToCelsius.convert(fahrenheit);
    System.out.println("=================================================");
    System.out.println(fahrenheit + "° Fahrenheit is " + celsius + "° Celsius.");
    System.out.println("=================================================");
    double kilometers = 10;
    double miles = kilometersToMiles.convert(kilometers);
    System.out.println(kilometers + " kilometers is " + miles + " miles.");
    System.out.println("=================================================");
}}
```

OUTPUT:



2 WRITE A JAVA PROGRAM USING LAMBDA EXPRESSION WITH OR WITHOUT RETURN KEYWORD.

A) Lambda Expression With Return Keyword

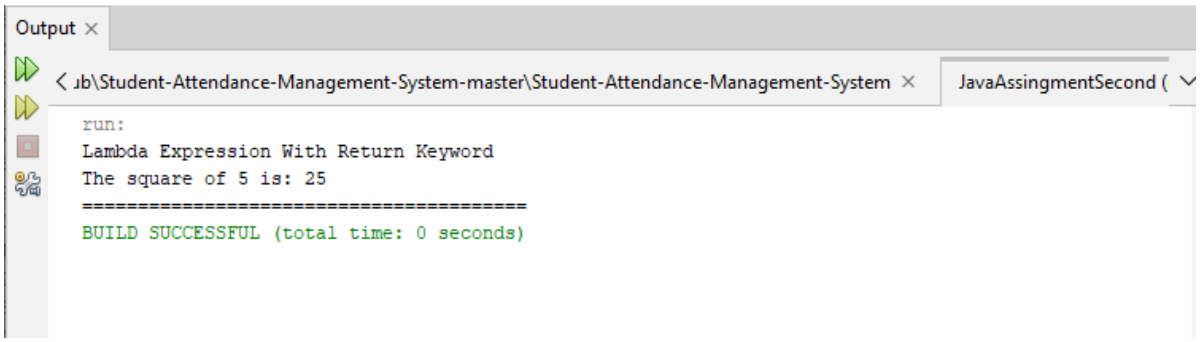
CODE :

/* WRITE A JAVA PROGRAM USING LAMBDA EXPRESSION WITH OR WITHOUT RETURN KEYWORD. */

```
package LambdaExpression;
public class LambdaExpressionWithReturnKeyword {
    // Define a functional interface with a method that returns a value
```

```
    @FunctionalInterface
    interface Calculator {
        // Method to calculate the square of a number
        int square(int number);
    }

    public static void main(String[] args) {
        // Lambda expression with a return keyword to calculate the square of a number
        Calculator calculator = (number) -> {
            return number * number; // Using 'return' to return the square of the number
        };

        // Testing the lambda expression
        int num = 5;
        int result = calculator.square(num); // Call the square method of the functional interface
        System.out.println("Lambda Expression With Return Keyword");
        System.out.println("The square of " + num + " is: " + result);
        System.out.println("========================================");
    }

}
```

OUTPUT:



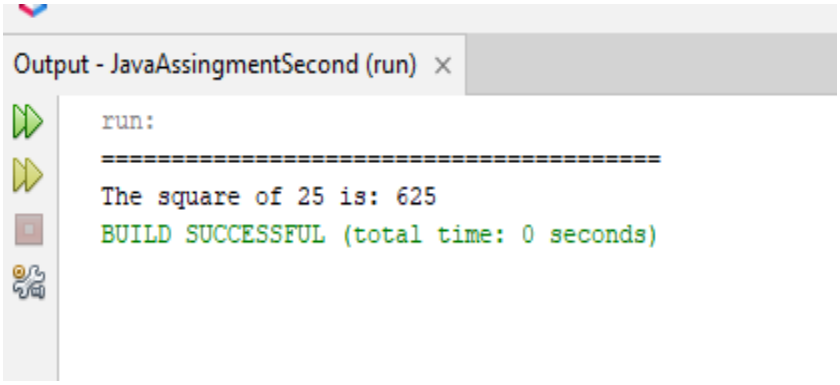
B) LAMBDA EXPRESSION WITHOUT RETURN KEYWORD.

CODE:

```
/* Lambda Expression With Return Keyword. */
package LambdaExpression;
public class LambdaExpressionWithoutReturnKyeword {
    // Define a functional interface with a method that returns a value
    @FunctionalInterface
    interface Calculate {
        // Method to calculate the square of a number
        int square(int number);
```

```
    }
    public static void main(String[] args) {
        // Lambda expression without the return keyword (single expression)
        Calculate calculate = (number) -> number * number;
        // Single expression, no need for 'return'
        // Testing the lambda expression
        int num = 25;
            int result = calculate.square(num); // Call the square method of the functional interface
        System.out.println("=========================================");
        System.out.println("The square of " + num + " is: " + result); // Output: The square      of 5 is: 25
    }
}
```

OUTPUT:



MCA Department

MCAL12 Advanced Java Lab Journal

Assignment No – 6

1. **Write a JSP to page to display registration from (make your own assumption) .**

JSP

JSP (JavaServer Pages) is a server-side technology developed by Sun Microsystems (now owned by Oracle) that allows the creation of dynamic, platform-independent web applications. It is an extension of Java Servlets and is used to build web pages with dynamic content, typically in HTML, XML, or other document types.

CODE:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>

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

<html>

<head>

  <title>User Registration</title>

  <style>

    body {

      font-family: Arial, sans-serif;

      background-color: #f4f4f4;

      margin: 0;

      padding: 20px;

    }

    .container {

      max-width: 400px;

      margin: auto;

      background: white;

      padding: 20px;

      border-radius: 5px;

      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
```

```
}
h2 {
    text-align: center;
}
.form-group {
    margin-bottom: 15px;
}
.form-group label {
    display: block;
    margin-bottom: 5px;
}
.form-group input {
    width: 100%;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 4px;
}
.form-group input[type="submit"] {
    background-color: #5cb85c;
    color: white;
    border: none;
    cursor: pointer;
}
.form-group input[type="submit"]:hover {
    background-color: #4cae4c;
}
```

```
    </style>
</head>
<body>
<div class="container">
    <h2>User Registration</h2>
    <form action="RegisterServlet" method="post">
        <div class="form-group">
            <label for="name">Full Name:</label>
            <input type="text" id="name" name="name" required>
        </div>
        <div class="form-group">
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" required>
        </div>
        <div class="form-group">
            <label for="password">Password:</label>
            <input type="password" id="password" name="password" required>
        </div>
        <div class="form-group">
            <label for="confirmPassword">Confirm Password:</label>
            <input type="password" id="confirmPassword" name="confirmPassword" required>
        </div>
        <div class="form-group">
            <input type="submit" value="Register">
        </div>
    </form>
```

</body>

</html>

OUTPUT:

2. **Design loan calculator using JSP which accepted 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 loan for the following time period and interest rate (A) 1 to 7 years at 5.35% (B) 8 to 15 years, at 5. 5 % (C) 16 to 30 years at 5.75% .**

CODE:

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<%@ page import="java.text.DecimalFormat" %>

<!DOCTYPE html>

<html>

<head>

<title>Loan Calculator</title>

</head>

<body>

<h2>Loan Calculator</h2>

<form method="post">

```
    <label for="principal">Principal Loan Amount:</label>
    <input type="number" id="principal" name="principal" required><br><br>
    <label for="years">Period of Time (in years):</label>
    <input type="number" id="years" name="years" required><br><br>
    <input type="submit" value="Calculate">
</form>
<%
    if (request.getMethod().equalsIgnoreCase("post")) {
        double principal = Double.parseDouble(request.getParameter("principal"));
        int years = Integer.parseInt(request.getParameter("years"));
        double interestRate = 0.0;
        // Determine the interest rate based on the number of years
        if (years >= 1 && years <= 7) {
            interestRate = 5.35;
        } else if (years >= 8 && years <= 15) {
            interestRate = 5.5;
        } else if (years >= 16 && years <= 30) {
            interestRate = 5.75;
        } else {
            out.println("<p>Invalid loan period. Please enter a value between 1 and 30 years.</p>");
            return;
        }
        // Monthly interest rate
        double monthlyRate = interestRate / 100 / 12;
        // Total number of payments
        int numberOfPayments = years * 12;
```

```
        // Monthly payment calculation using the formula

        double monthlyPayment = (principal * monthlyRate) / (1 - Math.pow(1 + monthlyRate, -numberOfPayments));

        // Display the monthly payment

        DecimalFormat df = new DecimalFormat("#.##");

        out.println("<h3>Monthly Payment: $" + df.format(monthlyPayment) + "</h3>");

        // Display loan balance and interest paid for each payment

        out.println("<h3>Loan Amortization Schedule:</h3>");

        out.println("<table border='1'><tr><th>Payment Number</th><th>Payment Amount</th><th>Interest
Paid</th><th>Principal Paid</th><th>Remaining Balance</th></tr>");

        double remainingBalance = principal;

        for (int paymentNumber = 1; paymentNumber <= numberOfPayments; paymentNumber++) {

            double interestPaid = remainingBalance * monthlyRate;

            double principalPaid = monthlyPayment - interestPaid;

            remainingBalance -= principalPaid;

            out.println("<tr><td>" + paymentNumber + "</td>");

            out.println("<td>$" + df.format(monthlyPayment) + "</td>");

            out.println("<td>$" + df.format(interestPaid) + "</td>");

            out.println("<td>$" + df.format(principalPaid) + "</td>");

            out.println("<td>$" + df.format(remainingBalance) + "</td></tr>");

        }

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

    }

%>

</body>

</html>
```

OUTPUT:

```
Monthly Payment: $188.71

Loan Amortization Schedule:
-------------------------------------------------
| Payment Number | Payment Amount | Interest Paid | Principal Paid |  Remaining Balance |
|----------------|----------------|---------------|----------------| -------------------|
| 1              | $188.71        | $44.58        | $144.13        |  $9855.87          |
| 2              | $188.71        | $43.87        | $144.84        |  $9841.03          |
| 3              | $188.71        | $43.75        | $144.96        |  $9696.07          |
| ...            | ...            | ...           | ...            |  ...               |
| 60             | $188.71        | $0.84         | $187.87        |  $0.00             |
-------------------------------------------------
```

3. **Write a JSP program that demonstrate the use of JSP declaration ,scriptlet, directivity ,expression, header and footer.**

JSP Scripting Tags

JSP scripting tags are used to embed Java code directly into JSP pages. There are three types:

1. Declarations (<%! ... %>): Used to declare variables and methods at the page level.

<%! int counter = 0; %>

2. Scriptlets (<% ... %>): Allows Java code to be inserted and executed during the request.

<% counter++; %>

3. Expressions (<%= ... %>): Evaluates an expression and outputs the result as part of the

HTML.

CODE:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

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

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<%! // JSP Declaration

   public String getCurrentDate() {

      return new Date().toString();

   }
```

```
%>
<%
   // JSP Scriptlet
   String userName = request.getParameter("username");
   if (userName == null || userName.isEmpty()) {
      userName = "Guest";
   }
%>
<jsp:include page="header.jsp" />
<html>
<head>
   <title>JSP Example</title>
</head>
<body>
   <h1>Welcome, <%= userName %>!</h1> <!-- JSP Expression -->
   <p>Today's date is: <%= getCurrentDate() %></p> <!-- JSP Expression -->
   <p>This is a simple demonstration of JSP features including:</p>
   <ul>
      <li>Declarations</li>
      <li>Scriptlets</li>
      <li>Expressions</li>
   </ul>
</body>
</html>
<jsp:include page="footer.jsp" />
```

header.jsp:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>My JSP Application</title>
</head>
<body>
    <header>
        <h1>My JSP Application Header</h1>
    </header>
```

footer.jsp:

```
    <footer>
        <p>&copy; 2023 My JSP Application. All rights reserved.</p>
    </footer>
</body>
</html>
```

OUTPUT:

No Username Provided :

```
http://localhost:8080/yourapp/example.jsp
```

Username Provided:

```
http://localhost:8080/yourapp/example.jsp?username=John
```

MCA Department

MCAL12 Advanced Java Lab Journal

Assignment No – 7

1. **Write a program to print "Hello World" using spring framework.**

Spring Framework is a comprehensive and versatile platform for enterprise Java development. It is known for its **Inversion of Control (IoC)** and **Dependency Injection (DI)** capabilities that simplify creating modular and testable applications. Key features include **Spring MVC** for web development, **Spring Boot** for rapid application setup, and **Spring Security** for robust authentication and authorization. With a rich ecosystem covering **Spring Data** for database interactions and **Spring Cloud** for building microservices, **Spring** supports scalable and resilient enterprise solutions, making it an essential framework for developers of all experience levels.

CODE :

```
package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class HelloWorldApplication {

        public static void main(String[] args) {

                SpringApplication.run(HelloWorldApplication.class, args);

                System.out.println("Hello World");

} }
```

OUTPUT :

```
 :: Spring Boot ::                (v3.4.0)

2024-12-11T00:31:33.677+05:30  INFO 9528 --- [HelloWorld] [  restartedMain]
2024-12-11T00:31:33.680+05:30  INFO 9528 --- [HelloWorld] [  restartedMain]
2024-12-11T00:31:33.717+05:30  INFO 9528 --- [HelloWorld] [  restartedMain]
2024-12-11T00:31:33.717+05:30  INFO 9528 --- [HelloWorld] [  restartedMain]
2024-12-11T00:31:34.614+05:30  INFO 9528 --- [HelloWorld] [  restartedMain]
2024-12-11T00:31:34.629+05:30  INFO 9528 --- [HelloWorld] [  restartedMain]
2024-12-11T00:31:34.629+05:30  INFO 9528 --- [HelloWorld] [  restartedMain]
2024-12-11T00:31:34.664+05:30  INFO 9528 --- [HelloWorld] [  restartedMain]
2024-12-11T00:31:34.664+05:30  INFO 9528 --- [HelloWorld] [  restartedMain]
2024-12-11T00:31:34.917+05:30  WARN 9528 --- [HelloWorld] [  restartedMain]
2024-12-11T00:31:34.989+05:30  INFO 9528 --- [HelloWorld] [  restartedMain]
2024-12-11T00:31:35.016+05:30  INFO 9528 --- [HelloWorld] [  restartedMain]
2024-12-11T00:31:35.023+05:30  INFO 9528 --- [HelloWorld] [  restartedMain]
Hello World
```

2. Write a program to demonstrate dependency injection via setter method.

Dependency Injection

Dependency Injection is the main functionality provided by Spring IOC(Inversion of Control). The Spring-Core module is responsible for injecting dependencies through either Constructor or Setter methods. The design principle of Inversion of Control emphasizes keeping the Java classes independent of each other and the container frees them from object creation and maintenance. These classes, managed by Spring, must adhere to the standard definition of Java-Bean. Dependency Injection in Spring also ensures loose coupling between the classes.

Setter-based dependency injection :

Setter-based DI is accomplished by the container invoking setter properties on your objects after invoking a no-argument constructor or no-argument static factory method to instantiate your object.

CODE :

ConstructorInjectionApplication :

```
package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication

public class ConstructorInjectionApplication {

	public static void main(String[] args) {

		ConfigurableApplicationContext context =
SpringApplication.run(ConstructorInjectionApplication.class, args);

		Car c = context.getBean(Car.class);

		c.show();

		} }
```

Car.java :

```
package com.example.demo;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;
```

```
@Component
public class Car {
	private int cid;
	private String name = "Saurabh";
	private String tech;
	public int getCid() {
		return cid;
	}
	public void setCid(int cid) {
		this.cid = cid;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getTech() {
		return tech;
	}
	public void setTech(String tech) {
		this.tech = tech;
	}
	public void show()
	{
		System.out.println("Show is Called!!"+"\n"+name);
```

```
        laptop.Compile();

    } }
```

OUTPUT :

```
ConstructorInjection - ConstructorInjectionApplication [Spring Boot App] C:\Saurabh\spri

 :: Spring Boot ::                (v3.4.0)

2024-12-11T10:27:04.852+05:30  INFO 9808 --- [ConstructorInjection]
2024-12-11T10:27:04.857+05:30  INFO 9808 --- [ConstructorInjection]
2024-12-11T10:27:05.763+05:30  INFO 9808 --- [ConstructorInjection]
2024-12-11T10:27:05.788+05:30  INFO 9808 --- [ConstructorInjection]
2024-12-11T10:27:05.788+05:30  INFO 9808 --- [ConstructorInjection]
2024-12-11T10:27:05.848+05:30  INFO 9808 --- [ConstructorInjection]
2024-12-11T10:27:05.848+05:30  INFO 9808 --- [ConstructorInjection]
2024-12-11T10:27:06.140+05:30  WARN 9808 --- [ConstructorInjection]
2024-12-11T10:27:06.253+05:30  INFO 9808 --- [ConstructorInjection]
2024-12-11T10:27:06.260+05:30  INFO 9808 --- [ConstructorInjection]
Show is Called!!
Saurabh
```

3. Write a program to demonstrate dependency injection via Constructor.

Constructor-based dependency injection:

Constructor-based DI is accomplished by the container invoking a constructor with a number of arguments, each representing a dependency.

CODE :

ConstructorInjectionApplication :

```
package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication

public class ConstructorInjectionApplication {

        public static void main(String[] args) {

                ConfigurableApplicationContext context =
SpringApplication.run(ConstructorInjectionApplication.class, args);

                Car c = context.getBean(Car.class);
```

```
		c.show();
		} }
```

#Constructor :

```
package com.example.demo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
@Component
public class Car {
	private int cid;
	private String name = "Saurabh";
	private String tech;

	public void setLaptop(Laptop laptop) {
		this.laptop = laptop;
	}
//Constructor
	public Car()
	{
		super();
		System.out.println("Object");
	}
	public int getCid() {
		return cid;
	}
	public void setCid(int cid) {
		this.cid = cid;
```

```
}
public String getName() {
        return name;
}
public void setName(String name) {
        this.name = name;
}
public String getTech() {
        return tech;
}
public void setTech(String tech) {
        this.tech = tech;
}
public void show()
{
        System.out.println("Show is Called!!"+"\n"+name);
}}
```

OUTPUT :

```
<terminated> ConstructorInjection - Co

2024-12-11T10:33:18.245+05:30
2024-12-11T10:33:18.248+05:30
2024-12-11T10:33:19.068+05:30
2024-12-11T10:33:19.094+05:30
2024-12-11T10:33:19.094+05:30
2024-12-11T10:33:19.161+05:30
2024-12-11T10:33:19.162+05:30
Object
2024-12-11T10:33:19.452+05:30
2024-12-11T10:33:19.545+05:30
2024-12-11T10:33:19.562+05:30
```

4. **Write a program to demonstrate Autowiring.**

Auto Wiring:

The Spring container can autowire relationships between collaborating beans without using <constructor-arg> and <property> elements, which helps cut down on the amount of XML configuration you write for a big Spring-based application.

CODE:

```
package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication

public class ConstructorInjectionApplication {

	public static void main(String[] args) {

		ConfigurableApplicationContext context =
SpringApplication.run(ConstructorInjectionApplication.class, args);

		Car c = context.getBean(Car.class);

		c.show();

	} }
```

AUTOWIRING CLASS :

```
package com.example.demo;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

@Component

public class Car {

	private int cid;
```

```
private String name = "Saurabh";

private String tech;

@Autowired

private Laptop laptop;

public Laptop getLaptop() {

	return laptop;

}

public void setLaptop(Laptop laptop) {

	this.laptop = laptop;

}

public int getCid() {

	return cid;

}

public void setCid(int cid) {

	this.cid = cid;

}

public String getName() {

	return name;

}

public void setName(String name) {

	this.name = name;

}

public String getTech() {

	return tech;

}

public void setTech(String tech) {
```

```
		this.tech = tech;
	}
	public void show()
	{
		System.out.println("Show is Called!!"+"\n"+name);
		laptop.Compile();
	}}
```

LAPTOP.JAVA :

```
package com.example.demo;
import org.springframework.stereotype.Component;
@Component
public class Laptop {
	private int id;
	private String brand;
	@Override
	public String toString() {
		return "Laptop [id=" + id + ", brand=" + brand + "]";
	}
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getBrand() {
```

```
        return brand;

    }

    public void setBrand(String brand) {

        this.brand = brand;

    }

    public void Compile()

    {

        System.out.println("Laptop Compiling");

    }       }
```

OUTPUT :

```
<terminated> ConstructorInjection - ConstructorInjectionApplication [Spring Boot App] C
2024-12-11T10:27:04.857+05:30  INFO 9808 --- [ConstructorInjection]
2024-12-11T10:27:05.763+05:30  INFO 9808 --- [ConstructorInjection]
2024-12-11T10:27:05.788+05:30  INFO 9808 --- [ConstructorInjection]
2024-12-11T10:27:05.788+05:30  INFO 9808 --- [ConstructorInjection]
2024-12-11T10:27:05.848+05:30  INFO 9808 --- [ConstructorInjection]
2024-12-11T10:27:05.848+05:30  INFO 9808 --- [ConstructorInjection]
2024-12-11T10:27:06.140+05:30  WARN 9808 --- [ConstructorInjection]
2024-12-11T10:27:06.253+05:30  INFO 9808 --- [ConstructorInjection]
2024-12-11T10:27:06.260+05:30  INFO 9808 --- [ConstructorInjection]
Show is Called!!
Saurabh
Laptop Compiling
```

MCA Department

MCAL12 Advanced Java Lab Journal

Assignment No – 8

1. **Assignment based Aspect Oriented Programming**

AOP

Aspect-oriented programming (AOP) is one of the major components of the SpringFramework.

The Spring AOP helps in breaking down the logic of the program into

several distinct parts called as concerns. Cross-cutting concerns is the functions

which span multiple points of an application.

The cross-cutting concerns help in increasing the modularity and separate it from

the business logic of an application. Also, a cross-cutting is a concern that affects

the whole application and it should be centralized in one location in code as

possible such as authentication, transaction management, logging etc.

CODE :

----------APPRUNNER----------

```
package com.study.SpringSession.AOP;
import javax.management.RuntimeErrorException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
@Component
public class AppRunner implements CommandLineRunner{
   @Autowired
   MyService myService;
   @Override
   public void run(String... args){
         System.out.println("Demonstrate spring AOP");
```

```
            System.out.println("Performing Task");
        myService.performTask();
        System.out.println("Fetching Data");
        myService.fetchData();
}}
```

----------LOGING ASPECTS----------

```
    package com.study.SpringSession.AOP;
    import org.aspectj.lang.ProceedingJoinPoint;
    import org.aspectj.lang.annotation.After;
    import org.aspectj.lang.annotation.AfterReturning;
    import org.aspectj.lang.annotation.Around;
    import org.aspectj.lang.annotation.Aspect;
    import org.aspectj.lang.annotation.Before;
    import org.aspectj.lang.annotation.Pointcut;
    import org.springframework.stereotype.Component;
    @Aspect
    @Component
    public class LoggingAspect {
        // pointcut
        @Pointcut("execution(* com.study.SpringSession.AOP.MyService.*(..))")
        private void serviceMethods(){
        }
        //BeforeAdvice
        @Before("servicMethod()")
        public void beforeAdvice(){
            System.out.println("Before Advice Method ia about to excute");
```

```
    }
    //After Advice
    @After("serviceMethods()")
    public void afterAdvice(){
        System.out.println("After ADvice method is about to excute");
    }
    //AroundADvice
    @Around("serviceMethods()")
    public Object aroundAdvice(ProceedingJoinPoint joinPoint) throws Throwable{
        System.out.println("Before mehtod execute");
        Object result = joinPoint.proceed();
        System.out.println("After mehtod execute");
        return result;
    }
    // After Returning Advice
    //@AfterReturning
(pointcut = "execution(* com.study.SpringSession.AOP.MyService.fetchData(..))"
  }
```

----------MYSERVICE----------

```
package com.study.SpringSession.AOP;
import org.springframework.stereotype.Service;
@Service
public class MyService {
  public void performTask(){
    System.out.println("Executing performTask");
  }
```

```
    public String fetchData(){
        System.out.println("Fetching data");
        return"Sample Data";
    }
    public void errorProneTask(){
        System.out.println("Executing error-prone task...");
        throw new RuntimeException("something went wrong !");
    }
}
```

----------SIMPLE SPRING PROGRA HELO----------

```
package com.study.SpringSession;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
@SpringBootApplication
public class SpringSessionApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringSessionApplication.class, args);
                    System.out.println("Helo");
    }
}
```

OUTPUT :

```
2024-12-10T16:26:12.061+05:30  INFO 14544 --- [SpringSession] [  restartedMain] c.s.S.SpringSessionApplication
   : Started SpringSessionApplication in 3.253 seconds (process running for 3.984)
Demonstrate spring AOP
Performing Task
Before mehtod execute
Executing performTask
After ADvice method is about to excute
After mehtod execute
Fetching Data
Before mehtod execute
Fetching data
After ADvice method is about to excute
After mehtod execute
Helo
```

MCA Department

MCAL12 Advanced Java Lab Journal

Assignment No – 9

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

JDBC

JDBC stands for Java Database Connectivity. Driver play role like to move an object from one place to another. Vehicle drivers are playing role to move vehicle as well objects whose included inside the vehicles from one place to another. JDBC APIs are used to access virtually any kind of data source from anywhere. JDBC is one type of API which connect and execute the query with the database. JDBC is part of JAVA SE (Java Standard Edition). JDBC API uses JDBC drivers to connect with different types of databases.

JDBC Drivers are used to manipulate data from database with the help of java platform. JDBC perform all types of SQL operations with java.

CODE :

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;
public class DatabaseOperations {
    // Database connection details
    private static final String DB_URL = "jdbc:mysql://localhost:3306/your_database";
    private static final String DB_USER = "your_username";
    private static final String DB_PASSWORD = "your_password";
    public static void main(String[] args) {
```

```
Scanner scanner = new Scanner(System.in);


try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
    System.out.println("Connected to the database.");
    while (true) {
        System.out.println("\nChoose an operation:");
        System.out.println("1. Insert record");
        System.out.println("2. Update record");
        System.out.println("3. Delete record");
        System.out.println("4. Exit");
        int choice = scanner.nextInt();
        scanner.nextLine(); // consume newline
        switch (choice) {
            case 1:
                insertRecord(connection, scanner);
                break;
            case 2:
                updateRecord(connection, scanner);
                break;
            case 3:
                deleteRecord(connection, scanner);
                break;
            case 4:
                System.out.println("Exiting program.");
                return;
            default:
```

```
                System.out.println("Invalid choice. Please try again.");
            }
        }
    } catch (SQLException e) {
        System.err.println("Database error: " + e.getMessage());
    }
}
private static void insertRecord(Connection connection, Scanner scanner) {
    System.out.print("Enter name: ");
    String name = scanner.nextLine();
    System.out.print("Enter email: ");
    String email = scanner.nextLine();
    String insertQuery = "INSERT INTO users (name, email) VALUES (?, ?)";
    try (PreparedStatement preparedStatement = connection.prepareStatement(insertQuery)) {
        preparedStatement.setString(1, name);
        preparedStatement.setString(2, email);
        int rowsInserted = preparedStatement.executeUpdate();
        System.out.println(rowsInserted + " record(s) inserted.");
    } catch (SQLException e) {
        System.err.println("Error inserting record: " + e.getMessage());
    }
}
private static void updateRecord(Connection connection, Scanner scanner) {
    System.out.print("Enter user ID to update: ");
    int id = scanner.nextInt();
    scanner.nextLine(); // consume newline
```

```
        System.out.print("Enter new name: ");
        String name = scanner.nextLine();
        System.out.print("Enter new email: ");
        String email = scanner.nextLine();
        String updateQuery = "UPDATE users SET name = ?, email = ? WHERE id = ?";
        try (PreparedStatement preparedStatement = connection.prepareStatement(updateQuery)) {
            preparedStatement.setString(1, name);
            preparedStatement.setString(2, email);
            preparedStatement.setInt(3, id);
            int rowsUpdated = preparedStatement.executeUpdate();
            System.out.println(rowsUpdated + " record(s) updated.");
        } catch (SQLException e) {
            System.err.println("Error updating record: " + e.getMessage());
        }
    }
    private static void deleteRecord(Connection connection, Scanner scanner) {
        System.out.print("Enter user ID to delete: ");
        int id = scanner.nextInt();
        String deleteQuery = "DELETE FROM users WHERE id = ?";
        try (PreparedStatement preparedStatement = connection.prepareStatement(deleteQuery)) {
            preparedStatement.setInt(1, id);
            int rowsDeleted = preparedStatement.executeUpdate();
            System.out.println(rowsDeleted + " record(s) deleted.");
        } catch (SQLException e) {
            System.err.println("Error deleting record: " + e.getMessage());
        }
```

} }

SQL:

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100)
);
```

OUTPUT :

```
Connected to the database.

Choose an operation:
1. Insert record
2. Update record
3. Delete record
4. Exit
1
Enter name: John Doe
Enter email: john.doe@example.com
1 record(s) inserted.
```

2.Write a program to demonstrate PreparedStatement in Spring JdbcTemplate

CODE :

```
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.jdbc.core.RowMapper;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
public class EmployeeJdbcTemplateExample {
    private JdbcTemplate jdbcTemplate;
    public EmployeeJdbcTemplateExample() {
        // Setting up the DataSource
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
        dataSource.setUrl("jdbc:mysql://localhost:3306/your_database");
        dataSource.setUsername("your_username");
        dataSource.setPassword("your_password");
        jdbcTemplate = new JdbcTemplate(dataSource);
    }
    public void insertEmployee(String name, String position, double salary) {
        String sql = "INSERT INTO employees (name, position, salary) VALUES (?, ?, ?)";
        jdbcTemplate.update(sql, name, position, salary);
        System.out.println("Employee inserted: " + name);
    }
    public void updateEmployee(int id, String name, String position, double salary) {
        String sql = "UPDATE employees SET name = ?, position = ?, salary = ? WHERE id = ?";
        jdbcTemplate.update(sql, name, position, salary, id);
        System.out.println("Employee updated: " + name);
    }
    public void deleteEmployee(int id) {
        String sql = "DELETE FROM employees WHERE id = ?";
```

```
        jdbcTemplate.update(sql, id);
        System.out.println("Employee deleted with ID: " + id);
    }
    public List<Employee> listEmployees() {
        String sql = "SELECT * FROM employees";
        return jdbcTemplate.query(sql, new EmployeeRowMapper());
    }
    public static void main(String[] args) {
        EmployeeJdbcTemplateExample example = new EmployeeJdbcTemplateExample();
        // Insert employees
        example.insertEmployee("Raj", "Developer", 60000);
        example.insertEmployee("Mayur", "Manager", 80000);
        // Update an employee
        example.updateEmployee(1, "Alice Johnson", "Senior Developer", 70000);
        // List all employees
        List<Employee> employees = example.listEmployees();
        System.out.println("Employee List:");
        for (Employee emp : employees) {
            System.out.println(emp);
        }
        // Delete an employee
        example.deleteEmployee(2);
        // List all employees again
        employees = example.listEmployees();
        System.out.println("Employee List after deletion:");
        for (Employee emp : employees) {
```

```
        System.out.println(emp);
    }
}
// Employee class
public static class Employee {
    private int id;
    private String name;
    private String position;
    private double salary;
    public Employee(int id, String name, String position, double salary) {
        this.id = id;
        this.name = name;
        this.position = position;
        this.salary = salary;
    }
    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Position: " + position + ", Salary: " + salary;
    }
}
public static class EmployeeRowMapper implements RowMapper<Employee> {
    @Override
    public Employee mapRow(ResultSet rs, int rowNum) throws SQLException {
        return new Employee(rs.getInt("id"), rs.getString("name"), rs.getString("position"), rs.getDouble("salary"));
    }
}
```

}

SQL :

```
CREATE TABLE employees (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    position VARCHAR(100) NOT NULL,
    salary DECIMAL(10, 2) NOT NULL
);
```

OUTPUT :

```
Employee inserted: Raj
Employee inserted: Mayur
Employee updated: Raj
Employee List:
ID: 1, Name: Raj, Position: Senior Developer, Salary: 70000.0
ID: 2, Name: Mayur, Position: Manager, Salary: 80000.0
Employee deleted with ID: 2
Employee List after deletion:
ID: 1, Name: RAj, Position: Senior Developer, Salary: 70000.0
```

3. Write a program in Spring JDBC to demonstrate ResultSetExtractor Interface.

SQL :

```
CREATE TABLE employees (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    position VARCHAR(100) NOT NULL,
    salary DECIMAL(10, 2) NOT NULL
);
```

CODE :

```
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.jdbc.core.ResultSetExtractor;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
public class EmployeeJdbcTemplateExample {
    private JdbcTemplate jdbcTemplate;
    public EmployeeJdbcTemplateExample() {
        // Setting up the DataSource
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
        dataSource.setUrl("jdbc:mysql://localhost:3306/your_database");
        dataSource.setUsername("your_username");
        dataSource.setPassword("your_password");
        jdbcTemplate = new JdbcTemplate(dataSource);
    }
    public void insertEmployee(String name, String position, double salary) {
        String sql = "INSERT INTO employees (name, position, salary) VALUES (?, ?, ?)";
        jdbcTemplate.update(sql, name, position, salary);
        System.out.println("Employee inserted: " + name);
    }
    public List<Employee> listEmployees() {
        String sql = "SELECT * FROM employees";
```

```
        return jdbcTemplate.query(sql, new EmployeeResultSetExtractor());
    }
    public static void main(String[] args) {
        EmployeeJdbcTemplateExample example = new EmployeeJdbcTemplateExample();
        // Insert employees
        example.insertEmployee("Raj", "Developer", 60000);
        example.insertEmployee("Mayur", "Manager", 80000);
        // List all employees
        List<Employee> employees = example.listEmployees();
        System.out.println("Employee List:");
        for (Employee emp : employees) {
            System.out.println(emp);
        }
    }
    // Employee class
    public static class Employee {
        private int id;
        private String name;
        private String position;
        private double salary;
        public Employee(int id, String name, String position, double salary) {
            this.id = id;
            this.name = name;
            this.position = position;
            this.salary = salary;
        }
```

```
    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Position: " + position + ", Salary: " + salary;
    }
}
// ResultSetExtractor implementation
public static class EmployeeResultSetExtractor implements ResultSetExtractor<List<Employee>> {
    @Override
    public List<Employee> extractData(ResultSet rs) throws SQLException {
        List<Employee> employees = new ArrayList<>();
        while (rs.next()) {
            Employee employee = new Employee(
                    rs.getInt("id"),
                    rs.getString("name"),
                    rs.getString("position"),
                    rs.getDouble("salary")
            );
            employees.add(employee);
        }
        return employees;
    }
}
}
```

OUTPUT :

```
Employee inserted: Raj
Employee inserted: Mayur
Employee List:
ID: 1, Name: Raj, Position: Developer, Salary: 60000.0
ID: 2, Name: Mayur, Position: Manager, Salary: 80000.0
```

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

CODE :

```
iimport org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
// User class (POJO)
class User {
    private int id;
    private String name;
    private String email;
    // Getters and Setters
    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 getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    @Override
    public String toString() {
        return "User{id=" + id + ", name='" + name + "', email='" + email + "'}";
    }
}
public class RowMapperExample {
    private static JdbcTemplate jdbcTemplate;
    public static void main(String[] args) {
        // Initialize DataSource and JdbcTemplate
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
```

```
        dataSource.setUrl("jdbc:mysql://localhost:3306/your_database");
        dataSource.setUsername("your_username");
        dataSource.setPassword("your_password");
        jdbcTemplate = new JdbcTemplate(dataSource);

        // Fetch all users and print them
        List<User> users = fetchAllUsers();
        System.out.println("Fetched Users:");
        users.forEach(System.out::println);
    }
    // Method to fetch all users from the database
    private static List<User> fetchAllUsers() {
        String query = "SELECT * FROM users";
        return jdbcTemplate.query(query, new RowMapper<User>() {
            @Override
            public User mapRow(ResultSet rs, int rowNum) throws SQLException {
                User user = new User();
                user.setId(rs.getInt("id"));
                user.setName(rs.getString("name"));
                user.setEmail(rs.getString("email"));
                return user;
            }
        });
    }
}
```

SQL :

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100)
);
INSERT INTO users (name, email) VALUES
    ('Alice', 'alice@example.com'),
    ('Bob', 'bob@example.com');
```

OUTPUT :

id	name	email
1	Alice	alice@example.com
2	Bob	bob@example.com

```
Fetched Users:
User{id=1, name='Alice', email='alice@example.com'}
User{id=2, name='Bob', email='bob@example.com'}
```

MCA Department

MCAL12 Advanced Java Lab Journal

Assignment No – 10

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

Spring Boot

Spring Boot is just extension of the already existing and expansive Spring frameworks, but it has some specific features that make the application easier for working within the developer ecosystem.

That extension includes pre-configurable web starter kits that help facilitate the responsibilities of an application server that are required for other Spring projects.

Restful Web services

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).

CODE:

```
package com.example.demo;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
@SpringBootApplication
public class DemoApplication {
```

```
	@GetMapping("/")
	public String Hello()
	{
		return "<h1>Hello World!!!</h1>";
	}
	public static void main(String[] args) {
		SpringApplication.run(DemoApplication.class, args);
	}
}
```

OUTPUT: