**Deccan Education Society’s**

**Navinchandra Mehta Institute of**

**Technology and Development**

**C E R T I F I C A T E**

This is to certify that Mr.Siddhesh Santosh Mhatre of M.C.A. Semester I with Roll No.C23071 has completed 10 practicals of Advanced Java under my supervision in this college during the year 2023-2024.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CO** | **R1**  **(Journal)** | **R2**  **(Performance during lab session)** | **R3**  **(Implementation using different problem solving techniques)** | **R4**  **(Mock**  **Viva)** | **Attendance** |
| **CO1** |  |  |  |  |  |
| **CO2** |  |  |  |  |  |
| **CO3** |  |  |  |  |  |
| **CO4** |  |  |  |  |  |
| **CO5** |  |  |  |  |  |
| **CO6** |  |  |  |  |  |

Practical-in-charge Head of Department

MCA Department

(NMITD)

# INDEX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MCAL 12 Advance Java Lab** | | |  |  |
| **S.No.** | **Problem Statement** | **Date** | **Course Outcome** | **Faculty Signature** |
| **Category 1: Java Generics** | | |  |  |
| 1. | Write a Java Program to demonstrate Wildcards in Java Generics | **08/09/2023** | **CO1** |  |
| 2. | Create a generic stack class that supports push, pop, and peek operations for different data types. | **CO1** |
| 3. | Implement a generic class that represents a pair of values. Write a method to swap the values in the pair. | **12/09/2023** | **CO1** |  |
| 4. | Create a generic method to sort an array of any data type using a sorting algorithm like quicksort. | **CO1** |
| 5. | Create a generic method to sort an array of any data type using a sorting algorithm like bubble sort | **CO1** |
| **Category 2: List Interface** | | |  |  |
| 6. | 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. | **14/09/2023** | **CO1** |  |
| 7. | Write a Java program to create List containing list of items and use List Iterator interface to print items present in the list. Also print the list in reverse/ backword direction. | **CO1** |
| 8. | Write a Java program to implement list interface through any of the class for following task   1. iterate through all elements in an array list. 2. Insert an element into the array list at the first position**.** 3. retrieve an element (at a specified index) from a given array list. 4. To update an array element by the given element | **CO1** |
| 9. | Write a Java program to implement list interface   * iterate through all elements in a linked list starting at the specified position. * to convert a linked list to an array list. * to compare two linked lists. * to shuffle elements in a linked list. | **CO1** |
| **Category 3: Set Interface** | | |  |  |
| 10. | Write a Java program to create a Set containing list of items of type String and print the items in the list using Iterator interface. Also print the list in reverse/ backword direction. | **15/09/2023** | **CO1** |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 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 |  |  |  |
| 11. | Write a Java program to perform sort method in set interface. | **25/09/2023** | **CO1** |  |
| 12. | Write a Java program to implement set interface convert a hash set to a List/ArrayList.   * to clone a hash set to another hash set. * to compare two sets and retain elements that are the same in new set. | **CO1** |
| 13. | Write a Java program to implement set interface to add all the elements of a specified tree set to |  |
|  | another tree set.   * to create a reverse order view of the elements contained in a given tree set. * to get the first and last elements in a tree set. * to get the element in a tree set which is greater than or equal to the given element. * to retrieve and remove the last element of a tree set. |  | **CO1** |  |
| **Category 4: Map Interface** | | | | |
| 14. | 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.   1. Remove items from the map 2. Search specific key from the map 3. Get value of the specified key 4. Insert map elements of one map in to other map. 5. Print all keys and values of the map. | **9/10/2023** | **CO1** |  |
| 15. | Write a Java program to copy all mappings from the specified map to another map. | **CO1** |
| 16. | Write a Java program to test if a map contains a mapping for the specified value. | **CO1** |
| 17. | Write a Java program to associate the specified value with the specified key in a Tree Map. | **CO1** |
| 18. | Write a Java program to search for a value and key in a Tree Map. | **CO1** |
| **Category 5: Lambda Expression** | | | | |
| 19. | Write a Java program using Lambda Expression to print” Hello World”. | **12/10/2023** | **CO1** |  |
| 20. | Write a Java program using Lambda Expression with single parameters. | **CO1** |
| 21. | Write a Java program using Lambda Expression with multiple parameters to add two numbers. | **12/10/2023** | **CO1** |  |
| 22. | Write a Java program using Lambda Expression to |  |
|  | calculate the following:  a. Convert Fahrenheit to Celsius |  | **CO1** |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | b. Convert Kilometers to Miles. |  |  |  |
| 23. | Write a Java program using Lambda Expression with or without return keyword. | **CO1** |
| 24. | Write a Java program using Lambda Expression to concatenate two strings. | **CO1** |
| **Category 6: web application development using JSP** | | | | |
| 25. | 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. | **19/10/2023** | **CO2** |  |
| 26. | Write a JSP page to display the Registration form (Make your own assumptions) | **CO2** |
| 27. | Write a JSP program to add, delete and display the records from StudentMaster (RollNo, Name, Semester, Course) table. | **CO2** |
| 28. | 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: a. 1 to 7 year at 5.35%   1. 8 to 15 year at 5.5% 2. 16 to 30 year at 5.75% | **25/10/2023** | **CO2** |  |
| 29. | 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 | **CO2** |
| 30. | Write a JSP program that demonstrates the use of JSP declaration, scriptlet, directives, expression, header and footer. | **CO2** |
| **Category 7:**  **Spring Framework** | | | | |
| 31. | Write a program to print “Hello World” using spring framework. | **30/10/2023** | **CO3** |  |
| 32. | Write a program to demonstrate dependency injection via setter method. |
| 33. | Write a program to demonstrate dependency injection via Constructor. | **2/11/2023** | **CO3** |  |
| **Category 8:**  **Aspect Oriented Programming** | | | | |
| 34. | Write a program to demonstrate Spring AOP – before advice. | **24/11/2023** | **CO4** |  |
| 35. | Write a program to demonstrate Spring AOP – after advice. |
| 36. | Write a program to demonstrate Spring AOP – around advice. |
| 37. | Write a program to demonstrate Spring AOP – after returning advice. | **24/11/2023** | **CO4** |  |
| 38. | Write a program to demonstrate Spring AOP – after throwing advice. |
| 39. | Write a program to demonstrate Spring AOP – point cuts. |  |  |  |
| **Category 9:**  **Spring JDBC** | |  |  | |
| 40. | Write a program to insert, update and delete records from the given table. | **8/12/2023** | **CO5** |  |
| 41. | Write a program to demonstrate Prepared Statement in Spring JdbcTemplate | **12/12/2023** | **CO5** |  |
| 42. | Write a program in Spring JDBC to demonstrate Result Set Extractor Interface | **15/12/2023** | **CO5** |  |
| 43. | Write a program to demonstrate Row Mapper interface to fetch the records from the database. |
| **Category 10:**  **Spring Boot** | |  |  | |
| 44. | Write a program to create a simple Spring Boot application that prints a message. | **21/21/2023** | **CO6** |  |

**Siddhesh Mhatre**

**MCA-B**

**C23071**

# ADVANCED JAVA

## PRACTICAL NO. 1 JAVA GENERICS

**Practical No.1.1 : Write a Java Program to demonstrate Wildcards in Java Generics.**

**THEORY:**

Generics was added in Java 5 to provide compile-time type checking and removing risk of ClassCastException that was common while working with collection classes.

The whole collection framework was re-written to use generics for type-safety.

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.

An entity such as class, interface, or method that operates on a parameterized type is a generic entity.

Wild Cards:

The ? (question mark) symbol represents the wildcard element.

If we write <? extends Number>, it means any child class of Number, e.g., Integer, Float, and double.

Now we can call the method of Number class through any child class object.

We can use a wildcard as a type of a parameter, field, return type, or local variable. However, it is not allowed to use a wildcard as a type argument for a generic method invocation, a generic class instance creation, or a supertype.

**Types of Wild Cards:**

**Upper Bounded Wild Card.**

These wildcards can be used when you want to relax the restrictions on a variable.

To declare an upper-bounded wildcard, use the wildcard characters(?) followed by ‘extends’ keyword followed by the upper bound.

**CODE:**

**package** example;

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

**public** **class** Wildcard {

//method to find the maximum element in a list of comparable objects

**public** **static** <T **extends** Comparable<T>> T findMax(List<T> list)

{

**if**(list==**null** || list.isEmpty())

{

**return** **null**;

}

T max = list.get(0); **for**(T item: list) { **if**(item.compareTo(max)>0) {

max=item;

}

}

**return** max;

}

**public** **static** **void** main(String[] args) { List<Integer> integerList = **new** ArrayList<>();

integerList.add(3); integerList.add(1); integerList.add(5);

List<Double> doubleList = **new** ArrayList<>();

doubleList.add(2.5); doubleList.add(4.0); doubleList.add(1.0);

//finding maximum value in integerList

Integer maxInteger = *findMax*(integerList);

System.***out***.println("Max integer:" + maxInteger);

//finding maximum value in doubleList

Double maxDouble = *findMax*(doubleList);

System.***out***.println("Max double:" + maxDouble);

}

}

**Output:**



**Lower Bounded Wildcard**

We use the lower bounded wildcard to widen the use of the type of variable.

To declare an lower-bounded wildcard, use the wildcard characters(?) followed by ‘super’ keyword followed by the lower bound.

**CODE:**

**package** example;

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

**public** **class** Wildcard {

**public** **static** **void** addElements(List<? **super** Integer> list) {

**for** (**int** i=1; i <=15; i++)

{

list.add(i);

}

}

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

{

List<Object> objectList = **new** ArrayList<>(); *addElements*(objectList);

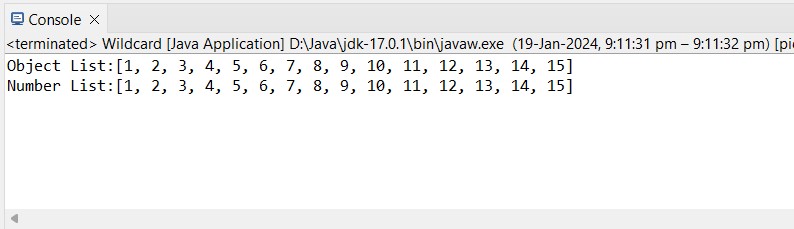
System.***out***.println("Object List:"+ objectList); List<Number> numberList = **new** ArrayList<>(); *addElements*(numberList);

System.***out***.println("Number List:" + numberList);

}

}

**Output**:



**Unbounded Wildcard**

We use the unbounded wildcard when we want to specify the type of wildcard with the wildcard character (?).

We generally use this wildcard when the code inside the method is using the object functionality and also when the code inside the method does not depend upon the parameter type.

**CODE**:

**package** example;

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

**public** **class** Wildcard {

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

{

List<?> myList = **new** ArrayList<>();

myList.add(**null**);

List<String> stringList = **new** ArrayList<>();

stringList.add("Hello"); stringList.add("world");

List<Integer>intList = **new** ArrayList<>();

intList.add(42); intList.add(56); *printListElements*(stringList);

*printListElements*(intList);

}

**public** **static** **void** printListElements(List<?> list) {

**for**(Object item : list) { System.***out***.println(item);

}

}

}

**Output:**



**Practical No.1.2 : Create a generic stack class that supports push, pop, and peek operations for different data types.**

**THEORY:**

**Generic Class:**

A generic class is implemented exactly like a non-generic class.

The only difference is that it contains a type parameter section.

There can be more than one type of parameter, separated by a comma.

The classes which accept one or more parameters are known as parameterized classes or parameterized types.

**Code:**

**package** generic; **interface** Stack<T>

{

**void** push(T item);

T pop(); T peek(); **boolean** isEmpty();

}

**class** generics<T> **implements** Stack<T>

{

**private** T[]stackArray; **private int** top; **private int** maxsize; **public** generics(**int** maxsize)

{

**this**.maxsize=maxsize;

**this**.stackArray=(T[])**new** Object[maxsize]; **this**.top=-1;

}

**public void** push(T item)

{

**if**(top<maxsize-1)

{

stackArray[++top]=item; }**else**

{

System.***out***.println("Stack is full. Cannot push"+item);

} } **public** T peek() { **if** (!isEmpty()) { **return** stackArray[top];

} **else** {

System.***out***.println("Stack is empty."); **return null**;

} } **public** T pop() {

**if**(!isEmpty())

{

**return** stackArray[top--]; }**else**

{

System.***out***.println("Stack is empty."); **return null**;

}

}

@Override **public boolean** isEmpty()

{ **return** top==-1;

} }

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

{

//create a stack of integers

Stack<Integer>intStack =**new** generics<>(5); intStack.push(10); intStack.push(20); intStack.push(30);

System.***out***.println("Popped item:"+intStack.pop());

System.***out***.println("IS stack empty?"+intStack.isEmpty());

System.***out***.println("Top element:" + intStack.peek());

//create a stack of strings

Stack<String>stringStack =**new** generics<>(3); stringStack.push("Hello"); stringStack.push("World"); stringStack.push("NMITD");

System.***out***.println("Popped item:"+stringStack.pop());

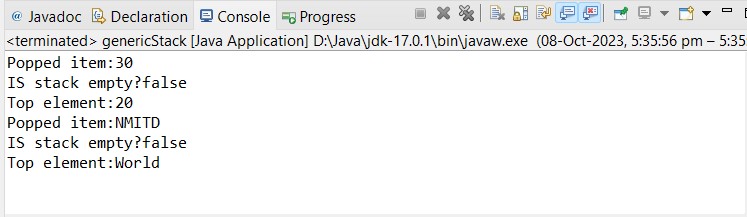
System.***out***.println("IS stack empty?"+stringStack.isEmpty());

System.***out***.println("Top element:" + stringStack.peek());

}

}

**Output:**



**Practical No.1.3 : Implement a generic class that represents a pair of values. Write a method to swap the values in the pair.**

**CODE:**

**package** example;

**import** java.util.Arrays;

**public** **class** practical {

**public** **static** <T> **void** swap(T[] array, **int** index1, **int** index2)

{

T temp = array[index1]; array[index1] = array[index2];

array[index2] = temp;

}

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

{

Integer[] intArray = {1,2,3,4,5};

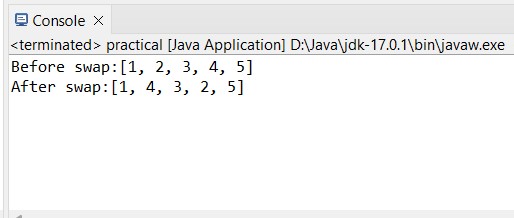
System.***out***.println("Before swap:" +Arrays.*toString*(intArray)); *swap*(intArray,1,3);

System.***out***.println("After swap:" +Arrays.*toString*(intArray));

}

}

**Output:**



**Practical No.1.4 : Create a generic method to sort an array of any data type using a sorting algorithm like quicksort.**

**Generic method:**

Like the generic class, we can create a generic method that can accept any type of arguments. Here, the scope of arguments is limited to the method where it is declared. It allows static as well as non-static methods.

**Quick Sort:**

Quicksort picks an element as pivot, and then it partitions the given array around the picked pivot element. In quick sort, a large array is divided into two arrays in which one holds values that are smaller than the specified value (Pivot), and another array holds the values that are greater than the pivot.

After that, left and right sub-arrays are also partitioned using the same approach. It will continue until the single element remains in the sub-array.

**CODE:**

**package** generic; **import** java.util.Arrays; **public class** Quicksort {

**public static** <T **extends** Comparable<T>> **void** quickSort(T[] array, **int** left, **int** right) {

**if** (left < right) { **int** partitionIndex = *partition*(array, left, right); // Recursively sort elements on the left of pivot *quickSort*(array, left, partitionIndex - 1); // Recursively sort elements on the right of pivot *quickSort*(array, partitionIndex + 1, right);

} }

**private static** <T **extends** Comparable<T>> **int** partition(T[] array, **int** left, **int** right) {

// Choose the rightmost element as pivot

T pivot = array[right]; // Pointer for greater element **int** i = left - 1;

// Traverse through all elements **for** (**int** j = left; j < right; j++) { **if** (array[j].compareTo(pivot) <= 0) { i++;

// Swap elements at indices i and j T temp = array[i]; array[i] = array[j]; array[j] = temp;

}

}

T temp = array[i + 1]; array[i + 1] = array[right]; array[right] = temp; **return** i + 1;

}

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

Integer[] data = {8, 7, 2, 1, 0, 9, 6};

System.***out***.println("original array:"+ Arrays.*toString*(data)); *quickSort*(data, 0, data.length - 1);

System.***out***.println("sorted array:" + Arrays.*toString*(data));

String[] stringData = {"Orange", "Banana", "Apple", "Pineapple",

"Peach"};

System.***out***.println("original array:" +

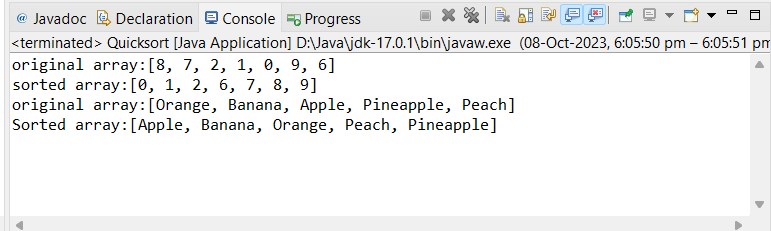
Arrays.*toString*(stringData));

*quickSort*(stringData, 0, stringData.length - 1);

System.***out***.println("Sorted array:" + Arrays.*toString*(stringData)); }

}

**Output:**



**Practical No.1.5 : Create a generic method to sort an array of any data type using a sorting algorithm like bubble sort.**

**Bubble sort:**

Bubble sort starts from the left end of the array and compare two elements in position 0 and 1. If the element positioned at 0th index is larger, we swap them. If the 1st element is larger, we don't do anything. Then we move over one position and compare the elements in positions 1 and 2.

**Code:**

**package** generic; **import** java.util.Arrays; **public class** Bubblesort {

**public static** <T **extends** Comparable<T>> **void** bubbleSort(T[] array)

{ **int** n = array.length; **for**(**int** i=0;i<n-1;i++) { **for**(**int** j=0;j<n-i-1;j++) {

**if**(array[j].compareTo(array[j+1])>0) { T temp = array[j]; array[j]=array[j+1]; array[j+1]=temp;

}

}

} }

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

{

Integer[] num = {6,2,8,5,0};

System.***out***.println("Original array:" + Arrays.*toString*(num)); *bubbleSort*(num);

System.***out***.println("sorted array:" + Arrays.*toString*(num));

String[] str =

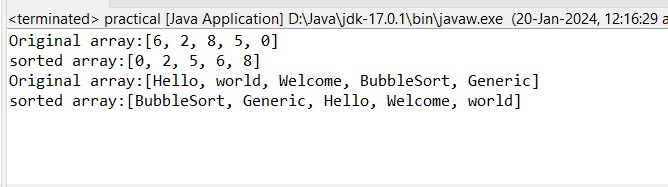
{"Hello","world","Welcome","BubbleSort","Generic"}; System.***out***.println("Original array:" + Arrays.*toString*(str)); *bubbleSort*(str);

System.***out***.println("sorted array:" + Arrays.*toString*(str));

}

}

**Output:**



## PRACTICAL NO. 2 LIST INTERFACE

**Practical 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.**

**THEORY:**

**LIST INTERFACE:**

The List Interface in java provides a way to store the ordered collection.

It is a child interface of collection.

It is an ordered collection of objects in which duplicate values can be stored.

The List Interface is found in java.util package and inherits the Collection interface.

**For-each loop:**

For-each is another array traversing technique like for loop, while loop, do-while loop introduced in Java5.

It starts with the keyword for like a normal for-loop.

Instead of declaring and initializing a loop counter variable, you declare a variable that is the same type as the base type of the array, followed by a colon, which is then followed by the array name.

In the loop body, you can use the loop variable you created rather than using an indexed array element.

It’s commonly used to iterate over an array or a Collections class (eg, ArrayList).

**Code:**

**package** example;

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

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

{

// Create an empty list to store student names

List<String> studentList = **new** ArrayList<>();

// Add students to the list studentList.add("Ahmed"); studentList.add("Babita"); studentList.add("Charu"); studentList.add("Danish"); studentList.add("mina"); // Print the list of students

System.***out***.println("List of students: " + studentList);

// Accessing elements by index

String secondStudent = studentList.get(1);

System.***out***.println("Second student: " + secondStudent);

// Check if a student String searchStudent = "Eve"; **boolean** containsEve = studentList.contains(searchStudent);

System.***out***.println("Is Eve in the list? " + containsEve);

// Remove a student from the list String removedStudent = "Charu";

**boolean** removed = studentList.remove(removedStudent);

System.***out***.println("Removed " + removedStudent + ": " + removed);

// Check the size of the list **int** size = studentList.size();

System.***out***.println("Number of students after removal: " + size);

// Iterate through the list of students System.***out***.println("Students in the list:"); **for** (String student : studentList)

{

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

}

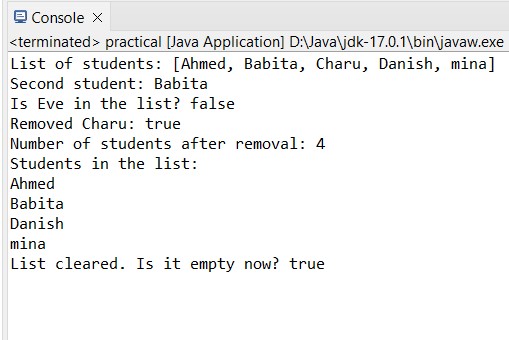
// Clear the list studentList.clear();

System.***out***.println("List cleared. Is it empty now? " + studentList.isEmpty());

}

}

**Output:**



**Practical No.2.2 : Write a Java program to create List containing list of items and use List Iterator interface to print items present in the list. Also print the list in reverse/ backward direction.**

**Theory:**

**List Iterator Interface:**

In Java, ListIterator is an interface in Collection API. It extends Iterator interface. To support Forward and Backward Direction iteration and CRUD operations. We can use this Iterator for all List implemented classes like ArrayList, CopyOnWriteArrayList, LinkedList, Stack, Vector, etc.

**Code:**

**package** example;

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

**public** **class** practical {

**public** **static** **void** main(String[] args) { // Create a list of lists containing items

List<List<String>> listOfLists = **new** ArrayList<>();

// Add inner lists

listOfLists.add(**new** ArrayList<>()); // Inner list 1 listOfLists.add(**new** ArrayList<>()); // Inner list 2

// Populate inner lists with items listOfLists.get(0).add("Apple"); listOfLists.get(0).add("Banana"); listOfLists.get(0).add("Cherry"); listOfLists.get(1).add("Dog"); listOfLists.get(1).add("Elephant"); listOfLists.get(1).add("Fox");

// Use ListIterator to print items present in the list System.***out***.println("Forward Direction:");

**for** (List<String> innerList : listOfLists)

{

ListIterator<String> iterator = innerList.listIterator();

**while** (iterator.hasNext()) {

System.***out***.println("- " + iterator.next());

}

}

// Use ListIterator to print the list in reverse direction

System.***out***.println("\nReverse Direction:"); **for** (List<String> innerList : listOfLists)

{

ListIterator<String> iterator = innerList.listIterator(innerList.size()); **while** (iterator.hasPrevious()) {

System.***out***.println("- " + iterator.previous());

}

}

}

}

**Output:**



**Practical No.2.3 : Write a Java program to implement list interface through any of the class for following task :**

**1.Iterate through all elements in an array list.**

**2.Insert an element into the array list at the first position.**

**3.Retrieve an element (at a specified index) from a given array list.**

**4.To update an array element by the given element.**

**CODE:**

**package** example;

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

**public** **class** practical {

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

List<Integer> list1 = **new** ArrayList<>();

//adding elements list1.add(10); list1.add(20); list1.add(40); list1.add(30); list1.add(50);

//iterate through elements by first method System.***out***.println("Elements in the list:");

**for** (Integer element : list1)

{

System.***out***.println(" "+ element);

}

//iterate through elements by second method

System.***out***.println("Elements in the list by second method:"); ListIterator<Integer> iterator = list1.listIterator();

**while** (iterator.hasNext()) {

System.***out***.println(" " + iterator.next());

}

//adding element at first position

list1.add(0, 35);

System.***out***.println("After addition : " + list1);

//retrieving element from a specific index

System.***out***.println("Third element is : " + list1.get(2));

//To update an array element by the given element list1.set(4, 5);

System.***out***.println("Updated list is : " + list1);

}

}

**Output:**



**Practical No.2.4 : Write a Java program to implement list interface**

* **iterate through all elements in a linked list starting at the specified position.**
* **to convert a linked list to an array list.**
* **to compare two linked lists.**
* **to shuffle elements in a linked list.**

**CODE:**

**package** generic;

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

**public** **class** ListInterface {

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

LinkedList<Integer> list1 = **new** LinkedList<>();

//adding elements list1.add(10); list1.add(20); list1.add(40); list1.add(30); list1.add(50); list1.add(60); list1.add(70);

System.***out***.println("Elements of Linkedlist1 are: " + list1);

//iterate through all elements in a linked list starting at the specified position.

ListIterator<Integer> iterator = list1.listIterator(2); System.***out***.println("\nIterated Elements of linkedlist1 are:");

**while**(iterator.hasNext()) {

System.***out***.println(" " + iterator.next());

}

//to convert a linked list to an array list. List<Integer> list2 = **new** ArrayList<>();

list2.addAll(list1); list2.add(5); list2.add(1); list2.add(2);

System.***out***.println("\nElements of ArrayList are: " + list2);

//to compare two linked lists.

LinkedList<Integer> list4 = **new** LinkedList<>();

//adding elements list4.add(10); list4.add(20); list4.add(40); list4.add(3); list4.add(5); list4.add(6); list4.add(7);

System.***out***.println("\nElements of Linkedlist2 are: " + list4);

List<Integer> list3 = **new** LinkedList<>(list1);

list3.retainAll(list4);

System.***out***.println("\nAfter comparing, same elements in both linkedlist are: "+list3);

//to shuffle elements in linked list

Collections.*shuffle*(list4);

System.***out***.println("\nShuffled Linked List2 is:"); **for** (Integer element : list4) { System.***out***.print(element + " ");

}

}

}

**Output:**



## PRACTICAL NO. 3 SET INTERFACE

**THEORY:**

**SET INTERFACE:**

The set interface is present in [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package and extends the [Collection interface.](https://www.geeksforgeeks.org/collections-in-java-2/)

It is an unordered collection of objects in which duplicate values cannot be stored.

This interface contains the methods inherited from the Collection interface and adds a feature that restricts the insertion of the duplicate elements.

**Practical 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.**

**Iterator Interface:**

Java **Iterator** Interface of java [collections](https://www.geeksforgeeks.org/collections-in-java-2/) allows us to access elements of the collection and is used to iterate over the elements in the collection [(Map,](https://www.geeksforgeeks.org/map-interface-java-examples/) [List](https://www.geeksforgeeks.org/list-interface-java-examples/) or [Set)](https://www.geeksforgeeks.org/set-in-java/).

It helps to easily retrieve the elements of a collection and perform operations on each element.

**CODE:**

**package** example;

**import** java.util.ArrayList; **import** java.util.HashSet; **import** java.util.Iterator; **import** java.util.List; **import** java.util.ListIterator; **import** java.util.Set;

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

{

// Create a set of lists containing strings

|  |  |  |
| --- | --- | --- |
|  | Set | <List<String>> setOfLists = **new** HashSet<>(); |
| // Add inner lists | | |

setOfLists.add(**new** ArrayList<>()); // Inner list 1 setOfLists.add(**new** ArrayList<>()); // Inner list 2

// Populate inner lists with items setOfLists.forEach(innerList ->

{

innerList.add("rehan"); innerList.add("shmaya"); innerList.add("anvesha"); innerList.add("adhik");

});

// Use Iterator to print items present in the list System.***out***.println("Forward Direction:"); **for** (List<String> innerList : setOfLists)

{

Iterator<String> iterator = innerList.iterator();

**while** (iterator.hasNext())

{

System.***out***.println("- " + iterator.next());

}

}

// Use Iterator to print the list in reverse direction System.***out***.println("\nReverse Direction:");

**for** (List<String> innerList : setOfLists)

{

ListIterator<String>listIterator=innerList.listIterator(innerList.size()); **while** (listIterator.hasPrevious())

{

System.***out***.println("- " + listIterator.previous());

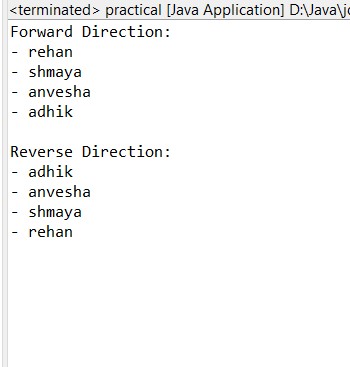
}

}

}

}

**Output:**



**Practical No.3.2 : Write a Java program using Set interface containing list of items and perform the following operations:**

**a. Add items in the set.**

1. **Insert items of one set in to other set.**
2. **Remove items from the set**
3. **Search the specified item in the set**

**CODE:**

**package** example;

**import** java.util.Arrays; **import** java.util.HashSet; **import** java.util.Set;

**public** **class** practical {

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

{

//declare a set class (HashSet)

Set<Integer> numSet = **new** HashSet<Integer>();

//add an element => add numSet.add(13);

//add a list to the set using addAll method

numSet.addAll(Arrays.*asList*(**new** Integer[] {1,6,4,7,3,9,8,2,12,11,20}));

//print the set

System.***out***.println("Original Set (numSet):" + numSet);

//size()

System.***out***.println("\nnumSet Size:" + numSet.size());

//create a new set class and initialize it with list elements

Set<Integer> oddSet = **new** HashSet<Integer>();

oddSet.addAll(Arrays.*asList*(**new** Integer[] {1, 3, 7, 5, 9}));

//print the set

System.***out***.println("\nOddSet contents:" + oddSet);

//contains ()

System.***out***.println("\nnumSet contains element 2:" + numSet.contains(3));

//containsAll ()

System.***out***.println("\nnumSet contains collection oddset:" + numSet.containsAll(oddSet));

// retainAll () => intersection

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);

// removeAll () => difference

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);

// addAll () => union

Set<Integer> set\_union = **new** HashSet<Integer>(numSet); set\_union.addAll(oddSet);

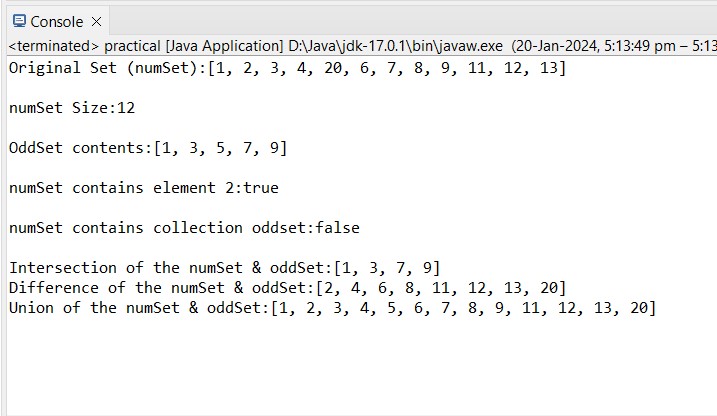
System.***out***.print("Union of the numSet & oddSet:");

System.***out***.println(set\_union);

}

}

**Output:**



**Practical No.3.3 : Write a Java program to perform sort method in set interface.**

**CODE:**

**package** generic;

**import** java.util.ArrayList; **import** java.util.Collections; **import** java.util.HashSet; **import** java.util.List; **import** java.util.Set; **import** java.util.TreeSet;

**public** **class** SetInterface {

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

//sort method in set interface

//create hash set

Set<Integer> set1 = **new** HashSet<>(); set1.add(10); set1.add(266); set1.add(35); set1.add(4);

set1.add(50);

System.***out***.println("Set elements are: " + set1);

// Convert the hash set to a array list

List<Integer> sortedList = **new** ArrayList<>(set1);

// Sort the list

Collections.*sort*(sortedList);

//create tree set to maintain order of sorted list

Set<Integer> sortedSet = **new** TreeSet<>(sortedList);

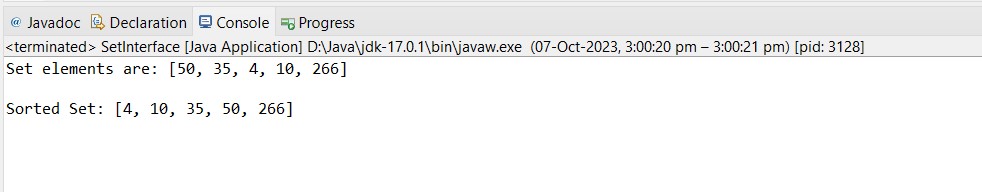
//print sorted set

System.***out***.println("\nSorted Set: " + sortedSet);

}

}

**Output:**



**Practical No.3.4 : Write a Java program to implement set interface**  **convert a hash set to a List/ArrayList.**

* **to clone a hash set to another hash set.**
* **to compare two sets and retain elements that are the same in new set.**

**HASH SET:**

* Java HashSet class is used to create a collection that uses a hash table for storage. o HashSet stores the elements by using a mechanism called **hashing.** o HashSet contains unique elements only. o HashSet allows null value.
* HashSet class is non synchronized. o HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode. o HashSet is the best approach for search operations.

**CODE:**

**package** generic;

**import** java.util.ArrayList; **import** java.util.Collections; **import** java.util.HashSet; **import** java.util.List; **import** java.util.Set; **import** java.util.TreeSet;

**public** **class** SetInterface {

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

//sort method in set interface

//create hash set

Set<Integer> set1 = **new** HashSet<>(); set1.add(10); set1.add(266); set1.add(35); set1.add(4); set1.add(50);

System.***out***.println("Set elements are: " + set1);

// Convert the hash set to a array list

List<Integer> list1 = **new** ArrayList<>(set1);

System.***out***.println("Arraylist elements are: " + list1);

//to clone a hash set to another hash set

Set<Integer> set2 = **new** HashSet<>(set1);

System.***out***.println("Clone set is: " + set2);

set2.add(5); set2.add(100); set2.add(2);

//to compare two sets and retain elements that are the same in new set. Set<Integer> set3 = **new** HashSet<Integer>(set1); set3.retainAll(set2);

System.***out***.println("Set1 elements are: " + set1);

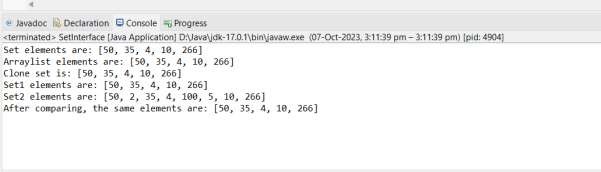
System.***out***.println("Set2 elements are: " + set2);

System.***out***.println("After comparing, the same elements are: " + set3 );

}

}

**Output:**



**Practical No.3.5 : Write a Java program to implement set interface**  **to add all the elements of a specified tree set to another tree set.**

* **to create a reverse order view of the elements contained in a given tree set.**
* **to get the first and last elements in a tree set.**
* **to get the element in a tree set which is greater than or equal to the given element.**
* **to retrieve and remove the last element of a tree set.**

**TREE SET:** o Java TreeSet class implements the Set interface that uses a tree for storage.

* The objects of the TreeSet class are stored in ascending order. o Java TreeSet class contains unique elements only like HashSet. o Java TreeSet class access and retrieval times are quiet fast. o Java TreeSet class doesn't allow null element.
* Java TreeSet class is non synchronized. o Java TreeSet class maintains ascending order.

**CODE:**

**package** generic;

**import** java.util.Iterator; **import** java.util.Set; **import** java.util.SortedSet; **import** java.util.TreeSet;

**public** **class** SetInterface {

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

//to add all the elements of a specified tree set to another tree set. Set<Integer> set1 = **new** TreeSet<>(); set1.add(10); set1.add(20); set1.add(30); set1.add(40); set1.add(50);

System.***out***.println("Treeset1 elements are: " + set1);

Set<Integer> set2 = **new** TreeSet<>();

set2.addAll(set1);

System.***out***.println("TreeSet2 elements are: " + set2);

//to create a reverse order view of the elements contained in a given tree set.

Iterator<Integer> reverseIterator = ((TreeSet<Integer>) set1).descendingIterator();

System.***out***.println("Reverse Order TreeSet:"); **while** (reverseIterator.hasNext()) { Integer element = reverseIterator.next();

System.***out***.print(element + " ");

}

//to get the first and last elements in a tree set.

System.***out***.println("\nFirst element of set1 is : " + ((TreeSet<Integer>) set1).first());

System.***out***.println("Last element of set1 is : " + ((TreeSet<Integer>) set1).last());

//to get the element in a tree set which is greater than or equal to the given element.

// Element to search for **int** target = 25;

// Find the least element greater than or equal to the target

Integer result = **null**;

**for** (Integer element : set1) { **if** (element >= target) { result = element;

**break**; // Exit the loop once we find the first match

}

}

**if** (result != **null**) {

System.***out***.println("Element greater than or equal to " + target + ": " + result); } **else** {

System.***out***.println("No such element found.");

}

//if we have to find more than one element

SortedSet<Integer> greaterOrEqualElements = ((TreeSet<Integer>) set1).tailSet(target);

// Print the elements

System.***out***.println("Elements greater than or equal to " + target + ":");

**for** (Integer element : greaterOrEqualElements) {

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

}

//to retrieve and remove the last element of a tree set

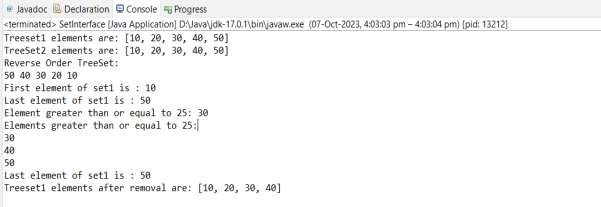
System.***out***.println("Last element of set1 is : " + ((TreeSet<Integer>) set1).last());

set1.remove(50);

System.***out***.println("Treeset1 elements after removal are: " + set1);

}

**Output:**



## PRACTICAL NO. 4 MAP INTERFACE

**Practical 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.**

1. **Remove items from the map**
2. **Search specific key from the map**
3. **Get value of the specified key**
4. **Insert map elements of one map in to other map.**
5. **Print all keys and values of the map.**

**THEORY:**

**MAP INTERFACE:**

In Java, Map Interface is present in [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package represents a mapping between a key and a value.

Java Map interface is not a subtype of the [Collection interface.](https://www.geeksforgeeks.org/collections-in-java-2/)

Therefore it behaves a bit differently from the rest of the collection types. A map contains unique keys.

**CODE:**

**package** example;

**import** java.util.HashMap;

**import** java.util.Map;

**public** **class** practical {

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

// Create two maps

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

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

// Add items to mapA mapA.put("Apple", 10); mapA.put("Banana", 20); mapA.put("Cherry", 30);

// Print mapA

System.***out***.println("Map A: " + mapA); // Remove an item from mapA

mapA.remove("Banana");

// Print mapA

System.***out***.println("Map A (after removing 'Banana'): " + mapA);

// Search for a specific key in mapA

String searchKey = "Cherry"; **if** (mapA.containsKey(searchKey)) {

System.***out***.println("'" + searchKey + "' found in mapA.");

} **else** {

System.***out***.println("'" + searchKey + "' not found in mapA.");

}

// Get value of the specified key in mapA

String getKey = "Apple"; **if** (mapA.containsKey(getKey)) { **int** value = mapA.get(getKey);

System.***out***.println("Value associated with '" + getKey + "': " + value);

}

// Insert map elements of mapA into mapB

mapB.putAll(mapA);

// Print mapB

System.***out***.println("Map B (after inserting mapA): " + mapB);

// Print all keys and values of mapA

System.***out***.println("\nAll keys and values of mapA:");

**for** (Map.Entry<String, Integer> entry: mapA.entrySet())

{

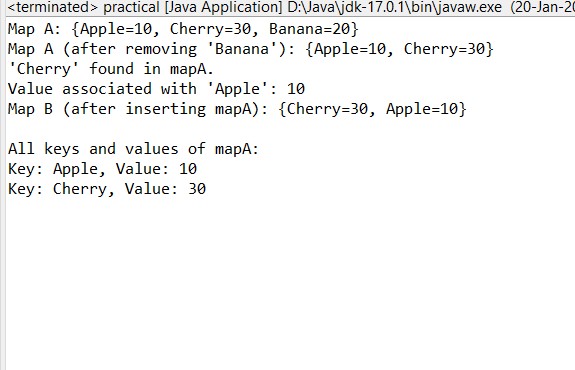
System.***out***.println("Key: " + entry.getKey() + ", Value: " + entry.getValue());

}

}

}

**Output:**



**Practical No.4.2 : Write a Java program to copy all mappings from the specified map to another map.**

**CODE:**

**package** generic;

**import** java.util.HashMap; **import** java.util.Map;

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

//to copy all mappings from the specified map to another map. Map<Integer, String> map1 = **new** HashMap<>();

map1.put(1,"ball"); map1.put(2, "bat"); map1.put(3, "chair"); map1.put(4, "table");

map1.put(5, "tv");

System.***out***.println("Elements of hash map are:" + map1);

Map<Integer,String> map2 = **new** HashMap<>();

map2.putAll(map1);

System.***out***.println("Map2 after copying from map1: " + map2);

}

}

**Output:**



**Practical No. 4.3 : Write a Java program to test if a map contains a mapping for the specified value.**

**CODE:**

**package** generic;

**import** java.util.HashMap; **import** java.util.Map;

**public** **class** MapInterface {

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

//to test if a map contains a mapping for the specified value

Map<Integer, String> map1 = **new** HashMap<>();

map1.put(1,"ball"); map1.put(2, "bat"); map1.put(3, "chair"); map1.put(4, "table");

map1.put(5, "tv");

System.***out***.println("Elements of hash map are:" + map1);

String target = "chair";

**boolean** contains = map1.containsValue(target);

**if** (contains) {

System.***out***.println("The map contains a mapping for value : " + target); } **else** {

System.***out***.println("The map does not contain a mapping for value : " + target);

}

}

}

**Output:**



**Practical No.4.4 : Write a Java program to associate the specified value with the specified key in a Tree Map.**

**CODE:**

**package** generic;

**import** java.util.Map; **import** java.util.TreeMap;

**public** **class** MapInterface {

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

Map<Integer, String> map1 = **new** TreeMap<>();

map1.put(1,"ball"); map1.put(2, "bat"); map1.put(3, "chair"); map1.put(4, "table");

map1.put(5, "tv");

System.***out***.println("Elements of Tree map are:" + map1);

//to associate the specified value with the specified key in a Tree Map

Integer update = 3;

String value = "Butter";

map1.put(update,value);

System.***out***.println("New treemap elements are:" + map1);

}

}

**Output:**



**Practical No.4.5 : Write a Java program to search for a value and key in a Tree Map.**

**CODE:**

**package** generic;

**import** java.util.Map; **import** java.util.TreeMap;

**public** **class** MapInterface {

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

Map<Integer, String> map1 = **new** TreeMap<>();

map1.put(1,"ball"); map1.put(2, "bat"); map1.put(3, "chair"); map1.put(4, "table");

map1.put(5, "tv");

System.***out***.println("Elements of Tree map are:" + map1);

//to search for a value and key in a Tree Map.

Integer searchKey = 3; String searchValue = "ball"; **if**(map1.containsKey(searchKey)) {

System.***out***.println(searchKey + " : Key found!!");

}**else** {

System.***out***.println(searchKey + " : Not found");

}

**if**(map1.containsValue(searchValue)) {

System.***out***.println(searchValue + " : Value found!!");

}**else** {

System.***out***.println(searchValue + " : Not found");

}

}

}

**Output:**



## PRACTICAL NO. 5 LAMBDA EXPRESSION

**THEORY:**

**LAMBDA EXPRESSION:**

Lambda Expressions were added in Java 8.

A lambda expression is a short block of code which takes in parameters and returns a value. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method.

SYNTAX:

The simplest lambda expression contains a single parameter and an expression:

“ parameter -> expression “

**Practical No. 5.1 : Write a Java program using Lambda Expression to print ”Hello World”.**

**CODE:**

**package** example;

**interface** Greeting

{

**void** greet();

}

**public** **class** practical {

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

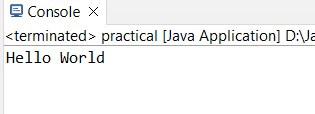
{

Greeting hello = () -> System.***out***.println("Hello World"); hello.greet();

}

}

**Output:**



**Practical No.5.2 : Write a Java program using Lambda Expression with single parameters.**

**CODE:**

**package** example;

**interface** StringLength

{

**int** getLength(String s);

}

**public** **class** practical

{

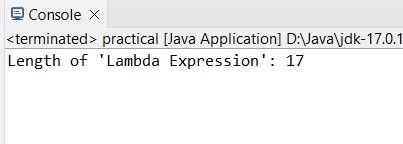
**public** **static** **void** main(String[] args) { StringLength lengthFunc = s -> s.length(); String input = "Lambda Expression"; **int** length = lengthFunc.getLength(input);

System.***out***.println("Length of '" + input + "': " + length);

}

}

**Output:**



**Practical No.5.3 : Write a Java program using Lambda Expression with multiple parameters to add two numbers.**

**CODE:**

**package** example;

**interface** MathOperation

{

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

}

**public** **class** practical

{

**public** **static** **void** main(String[] args) { MathOperation addition = (a, b) -> a + b; **int** num1 = 5;

**int** num2 = 10;

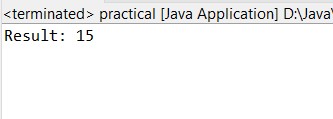
**int** result = addition.perform(num1, num2);

System.***out***.println("Result: " + result);

}

}

**Output:**



**Practical No. 5.4 : Write a Java program using Lambda Expression to calculate the following:**

1. **Convert Fahrenheit to Celsius**

**CODE:**

**package** example;

**interface** TemperatureConverter

{

**double** convert(**double** fahrenheit);

}

**public** **class** practical {

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

TemperatureConverter fahrenheitToCelsius = f -> (f - 32.0) \* 5.0/9.0; **double** fahrenheit = 98.6;

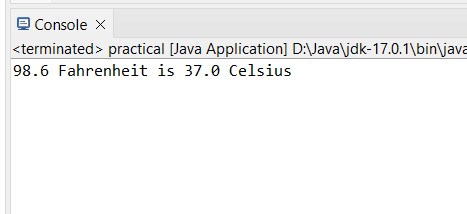
**double** celsius = fahrenheitToCelsius.convert(fahrenheit);

System.***out***.println(fahrenheit + " Fahrenheit is " + celsius + " Celsius");

}

}

**Output:**



**b. Convert Kilometers to Miles.**

**CODE:**

**package** example;

**interface** DistanceConverter

{

**double** convert(**double** kilometers);

}

**public** **class** practical {

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

DistanceConverter kilometersToMiles = km -> km \* 0.621371; **double** kilometers = 100;

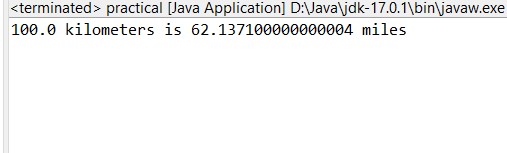
**double** miles = kilometersToMiles.convert(kilometers);

System.***out***.println(kilometers + " kilometers is " + miles + " miles");

}

}

**Output:**



**Practical No.5.5 : Write a Java program using Lambda Expression with or without return keyword.**

**CODE:**

**Using Lambda Expression with Return Keyword:**

**package** example; **interface** MathOperation

{

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

}

**public** **class** practical { **public** **static** **void** main(String[] args) { // Lambda expression with return keyword

MathOperation addition = (a, b) ->

{

**return** a + b;

};

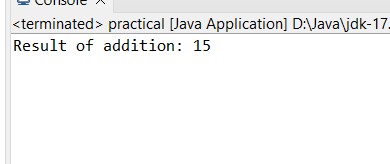
**int** result = addition.perform(5, 10);

System.***out***.println("Result of addition: " + result);

}

}

**Output:**



**Using Lambda Expression without Return Keyword:**

**CODE:**

**package** example; **interface** MathOp

{

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

}

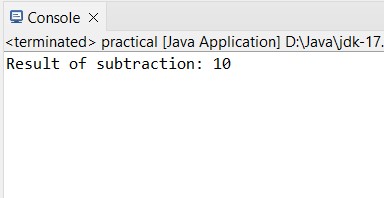
**public** **class** practical { **public** **static** **void** main(String[] args) { // Lambda expression without return keyword MathOp subtraction = (a, b) -> a - b; **int** result = subtraction.perform(15, 5);

System.***out***.println("Result of subtraction: " + result);

}

}

**Output:**



**Practical No.5.6 : Write a Java program using Lambda Expression to concatenate two strings.**

**CODE:**

**package** example;

**interface** StringConcatenator

{

String concatenate(String str1, String str2);

}

**public** **class** practical {

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

// Lambda expression to concatenate two strings

StringConcatenator concatenator = (str1, str2) -> str1 + str2;

String firstString = "Hello, ";

String secondString = "Lambda Expression!";

System.***out***.println("Strigs are:");

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

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

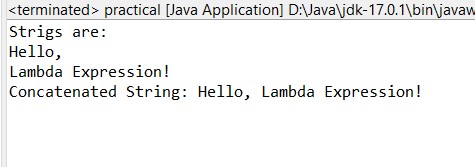
String result = concatenator.concatenate(firstString, secondString);

System.***out***.println("Concatenated String: " + result);

}

}

**Output:**



## PRACTICAL NO. 6 WEB APPLICATION DEVELOPMENT USING JSP

**THEORY:**

**JSP:**

* In Java, JSP stands for Java Server Pages.
* It is a server-side technology which is used for creating web applications.
* It is used to create dynamic web content.
* JSP consists of both HTML tags and JSP tags.
* In this, JSP tags are used to insert JAVA code into HTML pages.
* It is an advanced version of Servlet Technology i.e. a web-based technology that helps us to create dynamic and platform-independent web pages.
* In this, Java code can be inserted in HTML/ XML pages or both.
* JSP is first converted into a servlet by the JSP container before processing the client’s request.
* JSP has various features like JSP Expressions, JSP tags, JSP Expression Language, etc.

**Practical No.6.1 : 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.**

**CODE:**

**Indexfile.jsp**

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

<!DOCTYPE html>

<html>

<head>

<title>Telephone Directory</title>

</head>

<body>

<h1>Telephone Directory</h1>

<a href=*"add\_contact.jsp"*>Add Contact</a>

<a href=*"view\_contact.jsp"*>View Contacts</a>

</body>

</html>

**Add\_contact.jsp**

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

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Add contact</title>

</head>

<body>

<body>

<h1>Add Contact</h1>

<form action=*"add\_contact\_process.jsp"* method=*"post"*>

Name: <input type=*"text"* name=*"name"*><br>

Mobile-no: <input type=*"number"* name=*"mobileno"*><br>

Email: <input type=*"email"* name=*"email"*><br>

<input type=*"submit"* value=*"Add Contact"*>

</form>

</body>

</html>

**Add\_contact\_processs.jsp**

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

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Insert title here</title>

</head>

<body>

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

<%

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

String Mobileno = request.getParameter("mobileno");

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

Connection conn = **null**;

PreparedStatement ps = **null**;

String dbURL = "jdbc:postgres://localhost/telephonedirectory";

String dbUser = "postgres"; String dbPass = "root";

**try** {

Class.forName("com.postgres.cj.jdbc.Driver"); conn = DriverManager.getConnection(dbURL, dbUser, dbPass);

ps = conn.prepareStatement("INSERT INTO contactdetails (name, mobileno, email) VALUES (?, ?, ?)"); ps.setString(1, name); ps.setString(2, Mobileno); ps.setString(3, email); ps.executeUpdate();

response.sendRedirect("indexfile.jsp");

} **catch** (Exception e)

{

e.printStackTrace();

} **finally**

{

**if** (ps != **null**) ps.close(); **if** (conn != **null**) conn.close();

}

%>

</body>

</html>

**View\_contact.jsp**

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%> <%@ page import="java.sql.\*" %>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Insert title here</title>

</head>

<body>

<h1>View Contacts</h1>

<table border=*"1"*>

<tr>

<th>Name</th>

<th>Phone</th>

<th>Email</th>

</tr>

<%

Connection conn = **null**;

Statement stmt = **null**; String dbURL = "jdbc:postgres://localhost:5432/telephonedirectory";

String dbUser = "postgres"; String dbPass = "root";

**try** {

Class.forName("com.postgres.cj.jdbc.Driver"); conn = DriverManager.getConnection(dbURL, dbUser, dbPass); stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM contactdetails"); **while** (rs.next()) {

%>

<tr>

<td><%= rs.getString("name") %></td>

<td><%= rs.getString("Mobileno") %></td>

<td><%= rs.getString("email") %></td>

</tr>

<%

}

} **catch** (Exception e) {

e.printStackTrace(); } **finally** {

**if** (stmt != **null**) stmt.close(); **if** (conn != **null**) conn.close();

}

%>

</table>

</body>

</html>

**Practical No.6.2 : Write a JSP page to display the Registration form (Make your own assumptions) CODE:**

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

<!DOCTYPE html>

<html>

<head>

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

<title>Registration form </title>

</head>

<body>

<div align=*"center"*>

<h1> Student Registration Form</h1>

<form action=*"register2.jsp"* method=*"post"*>

<table style="with: *20%*">

<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>Email Id</td>

<td><input type=*"text"* name=*"Email ID"* />

</td>

</tr>

</table>

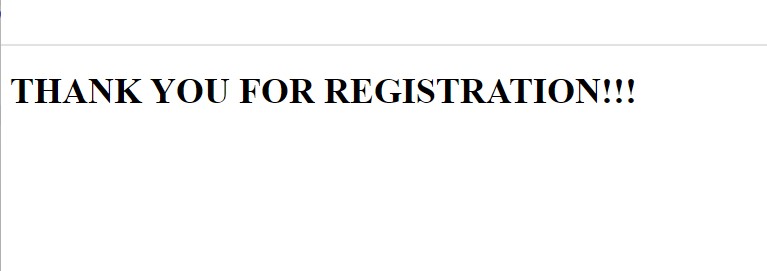
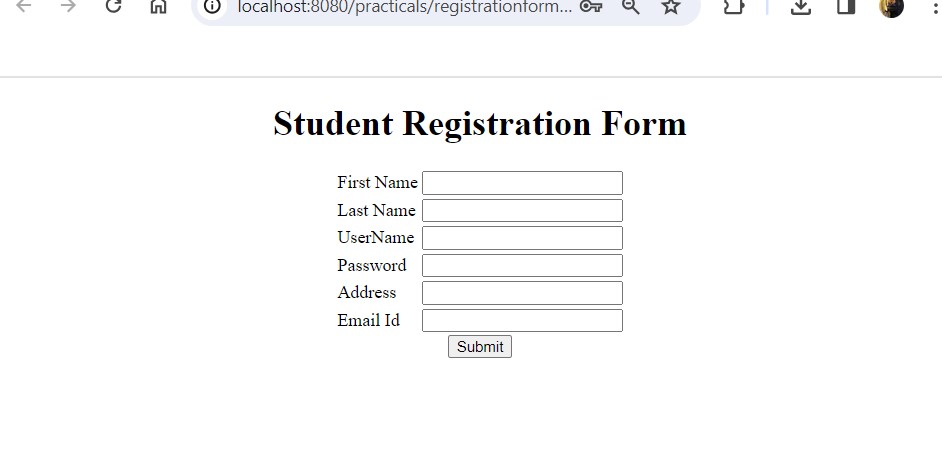
<input type=*"submit"* value=*"Submit"* />

</form> </div>

</body>

</html>

**After submitting.**



**Practical No.6.3 : Write a JSP program to add, delete and display the records from StudentMaster (RollNo, Name, Semester, Course) table.**

**CODE:**

**StudentMaster.js**

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

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

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

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>JSP List Students Records</title>

</head> <body>

<sql:setDataSource var="myDS" driver="com.mysql.jdbc.Driver" url="jdbc:mysql://localhost:3306/mydb" user="root" password="root" />

<sql:query var="listStudents" dataSource="${myDS}">SELECT \* FROM students; </sql:query>

<div align="center">

<table border="1" cellpadding="5">

<caption>

<h2>List of Students</h2>

</caption>

<tr>

<th>Roll No.</th>

<th>Name</th>

<th>Semester</th>

<th>Course</th>

</tr>

<c:forEach var="students" items="${listStudents.rows}"> <tr>

<td><c:out value="${students.roll\_no}"/></td>

<td><c:out value="${students.name}" /></td>

<td><c:out value="${students.semester}"/></td>

<td><c:out value="${students.course}" /></td>

</tr>

</c:forEach>

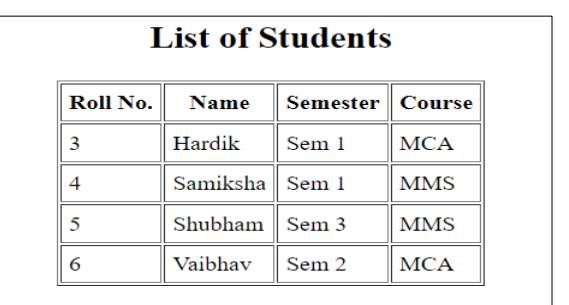
</table>

</div>

</body>

</html>

**Output:**



**Practical No.6.4 : 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:**

**a. 1 to 7 year at 5.35%**

1. **8 to 15 year at 5.5%**
2. **16 to 30 year at 5.75%**

**CODE:**

**Loancal.jsp**

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

<!DOCTYPE html>

<html>

<head>

<title>Loan Calculator</title>

</head>

<body>

<h1>Loan Calculator</h1>

<form action=*"calculateloan.jsp"* method=*"post"*>

<label for=*"loanAmount"*>Principal Loan Amount:</label>

<input type=*"number"* name=*"loanAmount"* id=*"loanAmount"* required><br>

<label for=*"interestRate"*>Annual Interest Rate (%):</label>

<input type=*"number"* name=*"interestRate"* id=*"interestRate"* required><br>

<label for=*"loanTerm"*>Loan Term (in years):</label>

<input type=*"number"* name=*"loanTerm"* id=*"loanTerm"* required><br>

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

</form>

</body> </html>

**Calculateloan.jsp**

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

<html>

<head>

<title>Loan Amortization Schedule</title>

</head>

<body>

<%

// Get input values from the form

**double** loanAmount = Double.parseDouble(request.getParameter("loanAmount")); **double** interestRate = Double.parseDouble(request.getParameter("interestRate")); **int** loanTerm = Integer.parseInt(request.getParameter("loanTerm")); // Calculate monthly interest rate and number of payments **double** monthlyInterestRate = (interestRate / 100) / 12; **int** numberOfPayments = loanTerm \* 12; // Calculate monthly payment

**double** monthlyPayment = (loanAmount \* monthlyInterestRate) / (1 - Math.pow(1 + monthlyInterestRate, -numberOfPayments));

// Initialize variables for loan balance and total interest paid

**double** loanBalance = loanAmount;

**double** totalInterestPaid = 0;

// Display loan details and amortization schedule out.println("<h1>Loan Amortization Schedule</h1>"); out.println("<p>Loan Amount: $" + loanAmount + "</p>"); out.println("<p>Annual Interest Rate: " + interestRate + "%</p>"); out.println("<p>Loan Term: " + loanTerm + " years</p>"); out.println("<p>Monthly Payment: $" + monthlyPayment + "</p>");

out.println("<table border='1'>");

out.println("<tr><th>Payment #</th><th>Payment Amount</th><th>Principal

Paid</th><th>Interest Paid</th><th>Loan Balance</th></tr>");

**for** (**int** paymentNumber = 1; paymentNumber <= numberOfPayments; paymentNumber++) { **double** interestPaid = loanBalance \* monthlyInterestRate; **double** principalPaid = monthlyPayment - interestPaid;

loanBalance -= principalPaid;

totalInterestPaid += interestPaid; out.println("<tr>");

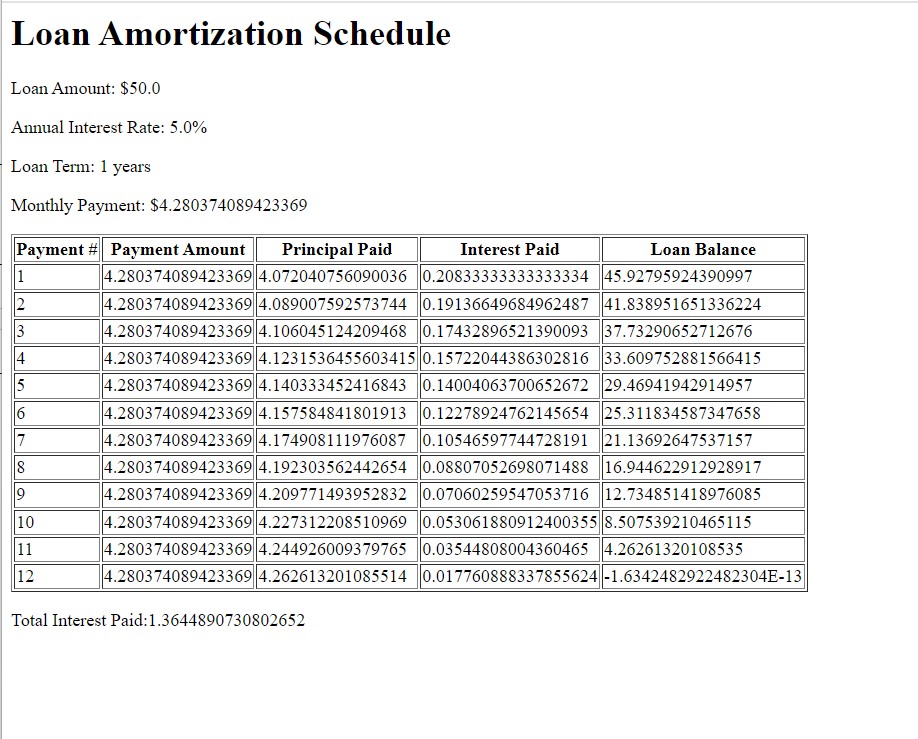
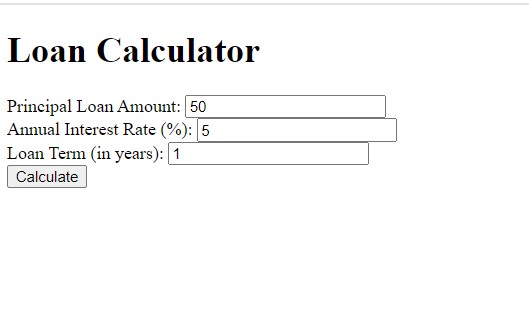
out.println("<td>" + paymentNumber + "</td>"); out.println("<td>" + monthlyPayment + "</td>"); out.println("<td>" + principalPaid + "</td>"); out.println("<td>" + interestPaid + "</td>"); out.println("<td>" + loanBalance + "</td>");

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

out.println("<p>Total Interest Paid:" + totalInterestPaid + "</p>"); %>

</body> </html>

**Output:**



**Practical No.6.5 : 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.**

**CODE:**

**Studycenter.jsp**

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

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Change Study Center Application Form</title>

</head>

<body>

<h2>Change Study Center Application Form</h2>

<form action=*"processChangeStudyCenter.jsp"* method=*"post"*>

<!-- Student Name -->

<label for=*"name"*>Student Name:</label>

<input type=*"text"* id=*"name"* name=*"name"* required><br>

<!-- Current Study Center -->

<label for=*"currentCenter"*>Current Study Center:</label>

<input type=*"text"* id=*"currentCenter"* name=*"currentCenter"* required><br>

<!-- New Study Center Choice -->

<label for=*"newCenter"*>New Study Center Choice:</label>

<input type=*"text"* id=*"newCenter"* name=*"newCenter"* required><br>

<!-- Submit Button -->

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

</form>

</body>

</html>

**Processchangestudycenter.jsp**

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

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Change Study Center Application - Confirmation</title>

</head>

<body>

<h2>Change Study Center Application - Confirmation</h2>

<p>Thank you for submitting your Change of Study Center Application. The details you provided are:</p>

<ul>

<li><strong>Student Name:</strong> <%= request.getParameter("name") %></li> <li><strong>Current Study Center:</strong> <%= request.getParameter("currentCenter") %></li>

<li><strong>New Study Center Choice:</strong> <%= request.getParameter("newCenter") %></li>

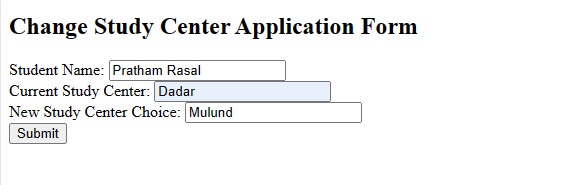
</ul>

<p>Your request will be processed soon. If you have any further questions, please contact the administration.</p>

</body>

</html>

**Output:**



**Practical No.6.6 : Write a JSP program that demonstrates the use of JSP declaration, scriptlet, directives, expression, header and footer.**

**CODE:**

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

<!DOCTYPE html>

<html>

<head>

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

<title>JSP example</title>

</head>

<body>

<header>

<h1>Welcome to My JSP Page</h1>

</header>

<main>

<%-- JSP Declaration: Define a variable --%> <%!

String greeting = "Hello, NMITD Welcome you all!";

%>

<%-- JSP Scriptlet: Use the declared variable --%>

<%

String username = "Gaurav";

%>

<%-- JSP Expression: Display the variable value --%>

<p><%= greeting %></p>

<p>Hello, <%= username %></p>

</main>

<footer>

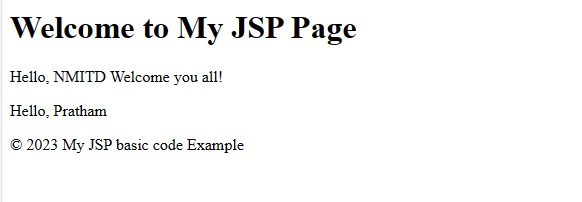
<p>&copy; 2023 My JSP basic code Example</p>

</footer>

</body>

</html>

**Output:**



## PRACTICAL NO. 7 SPRING FRAMEWORK

**Practical No.7.1 : Write a program to print “Hello World” using spring framework.**

**THEORY:**

**Spring Framework:**

* Spring is a lightweight and popular open-source Java-based framework developed by Rod Johnson in 2003.
* It is used to develop enterprise-level applications.
* It provides support to many other frameworks such as Hibernate, Tapestry, EJB, JSF, Struts, etc, so it is also called a framework of frameworks.
* It’s an application framework and IOC (Inversion of Control) container for the Java platform.
* The spring contains several modules like IOC, AOP, DAO, Context, WEB MVC, etc.

**CODE:**

**Person.java**

**package** springid;

**public** **class** Person { **private** String firstName; **private** String lastName;

**private** **int** age;

**public** Person() { }

**public** Person(String firstName, String lastName, **int** age) {

**this**.firstName = firstName; **this**.lastName = lastName;

**this**.age = age; }

**public** String getFirstName() { **return** firstName; }

**public** String getLastName() { **return** lastName; } **public** **int** getAge() {

**return** age; }

**public** **void** setFirstName(String firstName) { **this**.firstName = firstName; }

**public** **void** setLastName(String lastName) { **this**.lastName = lastName; }

**public** **void** setAge(**int** age) { **this**.age = age; } @Override **public** String toString() { **return** "Person{" +

"firstName='" + firstName + '\'' +

", lastName='" + lastName + '\'' +

", age=" + age +

'}';

}

}

**Metafile.xml**

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

<beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"* xsi:schemaLocation=*"http://www.springframework.org/schema/beans* *http://www.springframework.org/schema/beans/spring-beans.xsd* *http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- this is our beans -->

<bean id=*"obj1"* class=*"springid.Person"* name=*"Person"*>

<property name=*"firstName"* value=*"nmitd"*/>

<property name=*"lastName"* value=*"dadar"*/>

<property name=*"age"* value=*"2008"*/>

</bean> </beans>

**App.java**

**package** springid;

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

**import** org.springframework.context.support.ClassPathXmlApplicationContext; **public** **class** App

{

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

{

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

@SuppressWarnings("resource")

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

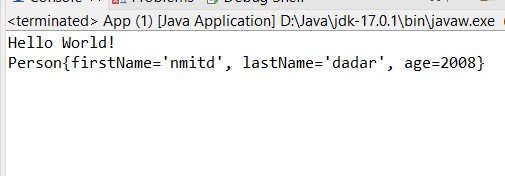
Person obj1=(Person) context.getBean("obj1");

System.***out***.println(obj1);

}

}

**Output:**



**Practical No.7.2 : Write a program to demonstrate dependency injection via setter method**.

**CODE:**

**Person.java**

**package** springid;

**public** **class** Person { **private** String studId; **private** String studName;

**private** String studAddress;

@Override

**public** String toString() {

**return** "student [studId=" + studId + ", studName=" + studName + ", studAddress=" + studAddress + "]"; }

**public** String getStudId() { **return** studId; }

**public** **void** setStudId(String studId) { **this**.studId = studId; } **public** String getStudName() { **return** studName; }

**public** **void** setStudName(String studName) { **this**.studName = studName; } **public** String getStudAddress() {

**return** studAddress; }

**public** **void** setStudAddress(String studAddress) { **this**.studAddress = studAddress;

} **public** Person(String studId, String studName, String studAddress) {

**super**();

**this**.studId = studId; **this**.studName = studName; **this**.studAddress = studAddress; } **public** Person() { **super**();

// **TODO** Auto-generated constructor stub

}

}

**Metafile.xml**

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

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

xmlns:p=*"http://www.springframework.org/schema/p"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd"*>

<bean class=*"springid.Person"* name=*"Person"* id=*"studBean"*>

<property name=*"studId"*> <value>01</value> </property>

<property name=*"studName"*> <value>Ritesh</value> </property>

<property name=*"studAddress"*> <value>dadar</value> </property>

</bean>

</beans>

**App.java**

**package** springid;

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

**import** org.springframework.context.support.ClassPathXmlApplicationContext; **public** **class** App

{

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

{

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

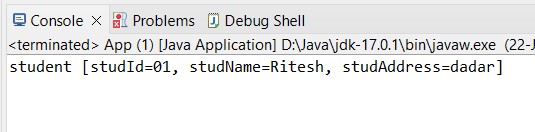
Person studobj= (Person) context.getBean("studBean");

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

}

}

**Output:**



**Practical No.7.3 : Write a program to demonstrate dependency injection via Constructor.**

**CODE:**

**package** constructor;

**public** **class** Engine { **private** String type; // Constructor **public** Engine(String type) { **this**.type = type;

}

// Getter

**public** String getType() { **return** type; }

@Override

**public** String toString() {

**return** "Engine{" + "type='" + type + '\'' + '}';

}

}

**Car.java**

**package** constructor;

**public** **class** Car { **private** String model; **private** Engine engine;

**public** Car(String model, Engine engine) { **this**.model = model; **this**.engine = engine; } **public** String getModel() { **return** model; }

**public** Engine getEngine() { **return** engine; } @Override

**public** String toString() {

**return** "Car{" +"model='" + model + '\'' +", engine=" + engine +'}'; }

}

**Testdata.java**

**package** constructor;

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

**import** org.springframework.context.support.ClassPathXmlApplicationContext; **public** **class** Testdata { **public** **static** **void** main(String[] args) {

// Load the Spring configuration file

@SuppressWarnings("resource") ApplicationContext context = **new**

ClassPathXmlApplicationContext("Beans.xml");

// Retrieve the Car bean from the Spring container

Car car = (Car) context.getBean("carBean");

// Print the Car details

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

}

**Beans.xml**

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

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

<!-- Define beans for Engine and Car -->

<bean id=*"engineBean"* class=*"constructor.Engine"*>

<constructor-arg value=*"V6"*/>

</bean>

<bean id=*"carBean"* class=*"constructor.Car"*>

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

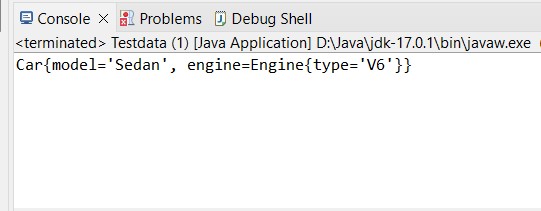
<!-- Inject the Engine dependency via the constructor -->

<constructor-arg ref=*"engineBean"*/>

</bean>

</beans>

**Output:**



## PRACTICAL NO.8 : Aspect Oriented Programming

**THEORY:**

**Aspect Oriented programming:**

* Aspect-Oriented Programming (AOP) is one of the key elements of the Spring Framework. AOP praises [Object-Oriented Programming](https://www.simplilearn.com/tutorials/java-tutorial/oops-interview-questions) in such a way that it also provides modularity.
* But the key point of modularity is the aspect than the class.
* AOP breaks the program logic into separate parts called concerns.
* The functions that span multiple points of a web application are named cross-cutting concerns and these cross-cutting concerns are conceptually separate from the application's business logic.
* AOP aims to increase modularity by allowing the separation of cross-cutting concerns.

**Practical No. 8.1 : Write a program to demonstrate Spring AOP – before advice.**

**Before Advice:**

* Before advice is used in Aspect-Oriented Programming to achieve the cross-cutting.
* It is an advice type which ensures that an advice runs before the method execution.
* We use **@Before** annotation to implement the before advice.

**CODE:**

**Through XML Configuration**

**Business Logic Class Calculator.java**

**package** com.Aspect.Before;

**public** **class** Calculator { **public** **int** add(**int** num1, **int** num2)

{

**return** num1 + num2;

}

}

**Aspect class BeforeDemo.java**

**package** com.Aspect.Before;

**public** **class** BeforeDemo { **public** **void** logBefore() {

System.***out***.println("Welcome to before world...");

System.***out***.println("Executing before the method...");

}

}

**XML Configuration File Metafile.xml**

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

<beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"* xsi:schemaLocation=*"http://www.springframework.org/schema/beans* *http://www.springframework.org/schema/beans/spring-beans.xsd* *http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- this is our beans -->

<bean id=*"obj1"* class=*"springid.Person"* name=*"Person"*>

<property name=*"firstName"* value=*"nmitd"*/>

<property name=*"lastName"* value=*"dadar"*/>

<property name=*"age"* value=*"2008"*/>

</bean>

</beans>

**Main class**

**BeforeMain.java**

**package com.Aspect.Before;**

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

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

**public class BeforeMain {**

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

**// Load the Spring XML configuration**

**@SuppressWarnings("resource")**

**ApplicationContext context = new**

**ClassPathXmlApplicationContext("BeforeConf.xml");**

**// Get the Calculator bean from the context**

**Calculator calculator = (Calculator) context.getBean("calculator"); // Call the add method to trigger AOP with before advice int result = calculator.add(5, 10);**

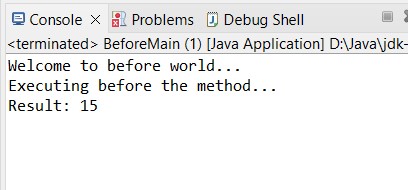
**// Print the result**

**System.out.println("Result: " + result); }**

**}**

**Output:**

**Annotation implementation**



**We Can Use Same Business Logic Class for implementation**

**BeforeAnnotation Aspect Class**

**BeforeAnnClass.java**

**package** com.Aspect.Before;

**import** org.aspectj.lang.annotation.Aspect; **import** org.aspectj.lang.annotation.Before; **import** org.springframework.stereotype.Component;

@Aspect @Component

**public** **class** BeforeAnnClass {

// Before advice method

@Before("execution(\* com.Aspect.Before.Calculator.add(..))") **public** **void** logBefore() {

System.***out***.println("Executing before the method...");

}

}

**BeforeAnnotation Configuration Class**

**BeforeAnnConf.java**

**package** com.Aspect.Before;

**import** org.springframework.context.annotation.Bean; **import** org.springframework.context.annotation.ComponentScan; **import** org.springframework.context.annotation.Configuration; **import** org.springframework.context.annotation.EnableAspectJAutoProxy;

@Configuration

@ComponentScan(basePackages = "com.Aspect.Before")

@EnableAspectJAutoProxy **public** **class** BeforeAnnConf { @Bean

**public**

Calcul

ator

calculator() {

**return**

**new**

Calculator();

}

}

**Main Class**

**BeforeAnnMain.java**

**package** com.Aspect.Before;

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

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

// Load the Spring configuration

AnnotationConfigApplicationContext context = **new**

AnnotationConfigApplicationContext(BeforeAnnConf.**class**);

// Get the Calculator bean from the context

Calculator calculator = context.getBean(Calculator.**class**); // Call the add method to trigger AOP with before advice

**int** result = calculator.add(5, 10);

// Print the result

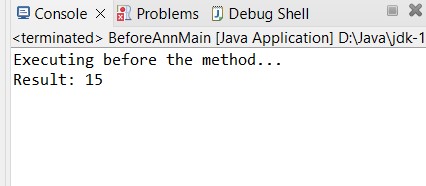
System.***out***.println("Result: " + result);

// Close the context context.close();

}

}

**Output:**



**Practical No.8.2 : Write a program to demonstrate Spring AOP – after advice.**

**CODE:**

**Student Class**

**package** com.Aspect.After; **public class** Student { **private** String studentId; **private** String name; **public** String getStudentId() { **return** studentId;

}

**public void** setStudentId(String studentId) { **this**.studentId = studentId;

}

**public** String getName() { **return** name;

}

**public void** setName(String name) { **this**.name = name;

}

}

**Another Business Logic Class**

**package** com.Aspect.After; **public class** StudentService {

**public void** enrollStudent(Student student) { // Logic for enrolling a student

System.***out***.println("Enrolling student: " + student.getName());

}

**public void** processPayment(Student student, **double** amount) {

// Logic for processing payment

System.***out***.println("Processing payment for student: " + student.getName() + ", Amount: $" + amount);

}

**public void** generateTranscript(Student student) {

// Logic for generating a transcript

System.***out***.println("Generating transcript for student: " + student.getName()); } }

**AfterXmlAspect Class**

**package** com.Aspect.After; **public class** AfterAspect { // After advice method **public void** logAfter() {

System.***out***.println("Executing after advice. Operation completed.");

}

}

**After Configuration File**

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

<beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xsi:schemaLocation=*"*

[*http://www.springframework.org/schema/beans*](http://www.springframework.org/schema/beans)

*http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop.xsd"*>

<!-- Define the StudentService bean -->

<bean id=*"studentService"* class=*"com.Aspect.After.StudentService"*/>

<!-- Define the LoggingAspect bean -->

<bean id=*"loggingAspect"* class=*"com.Aspect.After.AfterAspect"*/>

<!-- Configure AOP with After, After-Returning, and After-Throwing advice -->

<aop:config>

<!-- Define the pointcut for the methods in StudentService --> <aop:pointcut id=*"studentServiceMethods"* expression=*"execution(\**

*com.Aspect.After.StudentService.\*(..))"*/> <!-- Define the after advice using the pointcut -->

<aop:aspect ref=*"loggingAspect"*>

<aop:after method=*"logAfter"* pointcut-ref=*"studentServiceMethods"*/>

</aop:aspect>

</aop:config>

</beans>

**AfterMain Class**

**package** com.Aspect.After;

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

**import** org.springframework.context.support.ClassPathXmlApplicationContext; **import** com.Aspect.After.Student; **import** com.Aspect.After.StudentService; **public class** AfterXmlMain

{

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

{

// Load the Spring XML configuration

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

// Get the StudentService bean from the context

StudentService studentService = (StudentService) context.getBean("studentService");

// Create a sample student Student student = **new** Student(); student.setStudentId("123"); student.setName("NMITD");

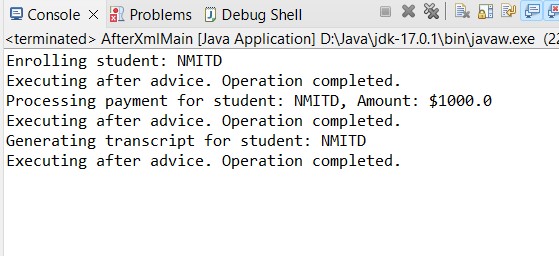
// Call methods to trigger AOP with after, after-returning, and after-throwing advice studentService.enrollStudent(student); studentService.processPayment(student, 1000); studentService.generateTranscript(student);

// Close the context

((ClassPathXmlApplicationContext) context).close();

}}

**Output:**



**AfterAnnotation Implmenation AfterAnnClass**

**package** com.Aspect.After; **import** org.aspectj.lang.annotation.After; **import** org.aspectj.lang.annotation.AfterReturning; **import** org.aspectj.lang.annotation.AfterThrowing; **import** org.aspectj.lang.annotation.Aspect; **import** org.springframework.stereotype.Component;

@Aspect

@Component

**public class** AfterAnnAspect {

// After advice method

@After("execution(\* com.Aspect.After.StudentService.\*(..))") **public void** logAfter() {

System.***out***.println("Executing after advice. Operation completed.");

}

}

**AppConfiguration File**

**package** com.Aspect.After;

**import** org.springframework.context.annotation.Bean; **import** org.springframework.context.annotation.ComponentScan; **import** org.springframework.context.annotation.Configuration; **import** org.springframework.context.annotation.EnableAspectJAutoProxy; **import** com.Aspect.After.StudentService;

@Configuration

@ComponentScan(basePackages = "com.Aspect.After")

@EnableAspectJAutoProxy **public class** AfterAppConfig {

@Bean

**public** StudentService studentServices()

{

**return new** StudentService();

}

}

**AnnotationMain Class**

**package** com.Aspect.After;

**import** org.springframework.context.annotation.AnnotationConfigApplicationContext; **public class** AfterAnnMain { **public static void** main(String[] args) {

// Load the Spring configuration

AnnotationConfigApplicationContext context = **new**

AnnotationConfigApplicationContext(AfterAppConfig.**class**);

// Get the StudentService bean from the context

StudentService studentService = context.getBean(StudentService.**class**);

// Create a sample student Student student = **new** Student(); student.setStudentId("123"); student.setName("NMITD");

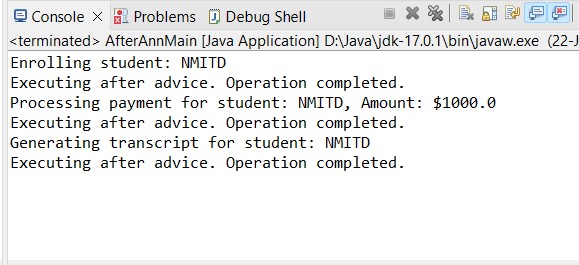
// Call methods to trigger AOP with after, after-returning, and after-throwing advice studentService.enrollStudent(student); studentService.processPayment(student, 1000); studentService.generateTranscript(student);

// Close the context context.close();

}

}

**Output:**



**Practical No.8.3 : Write a program to demonstrate Spring AOP –** **returning advice.**

**CODE:**

**Student Class**

**package** com.Aspect.After; **public class** Student { **private** String studentId; **private** String name; **public** String getStudentId() { **return** studentId;

}

**public void** setStudentId(String studentId) { **this**.studentId = studentId;

}

**public** String getName() { **return** name;

}

**public void** setName(String name) { **this**.name = name;

}

}

**Another Business Logic Class**

**package** com.Aspect.After; **public class** StudentService {

**public void** enrollStudent(Student student) {

// Logic for enrolling a student

System.***out***.println("Enrolling student: " + student.getName());

}

**public void** processPayment(Student student, **double** amount) {

// Logic for processing payment

System.***out***.println("Processing payment for student: " + student.getName() + ", Amount: $" + amount);

}

**public void** generateTranscript(Student student) {

// Logic for generating a transcript

System.***out***.println("Generating transcript for student: " + student.getName()); }

}

**AfterXmlAspect Class**

**package** com.Aspect.After; **public class** AfterAspect { // After-returning advice method **public void** logAfterReturning(Object result) {

System.***out***.println("Executing after-returning advice. Result: " + result); }}

**After Configuration File**

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

<beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop.xsd"*>

<!-- Define the StudentService bean -->

<bean id=*"studentService"* class=*"com.Aspect.After.StudentService"*/>

<!-- Define the LoggingAspect bean -->

<bean id=*"loggingAspect"* class=*"com.Aspect.After.AfterAspect"*/>

<!-- Configure AOP with After, After-Returning, and After-Throwing advice -->

<aop:config>

<!-- Define the pointcut for the methods in StudentService --> <aop:pointcut id=*"studentServiceMethods"* expression=*"execution(\**

*com.Aspect.After.StudentService.\*(..))"*/>

<!-- Define the after-returning advice using the pointcut -->

<aop:aspect ref=*"loggingAspect"*>

<aop:after-returning method=*"logAfterReturning"* pointcut-ref=*"studentServiceMethods"* returning=*"result"*/>

</aop:aspect>

</aop:config>

</beans>

**AfterMain Class**

**package** com.Aspect.After;

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

**import** org.springframework.context.support.ClassPathXmlApplicationContext; **import** com.Aspect.After.Student; **import** com.Aspect.After.StudentService; **public class** AfterXmlMain

{

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

{

// Load the Spring XML configuration

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

// Get the StudentService bean from the context

StudentService studentService = (StudentService) context.getBean("studentService");

// Create a sample student Student student = **new** Student(); student.setStudentId("123"); student.setName("NMITD");

// Call methods to trigger AOP with after, after-returning, and after-throwing advice studentService.enrollStudent(student);

studentService.processPayment(student, 1000); studentService.generateTranscript(student);

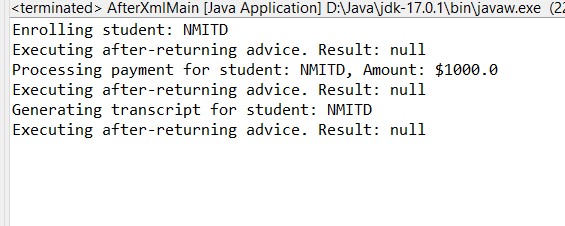
// Close the context

((ClassPathXmlApplicationContext) context).close();

}

}

**Output:**



**AfterAnnotation Implmenation**

**AfterAnnClass**

**package** com.Aspect.After;

**import** org.aspectj.lang.annotation.After; **import** org.aspectj.lang.annotation.AfterReturning; **import** org.aspectj.lang.annotation.AfterThrowing; **import** org.aspectj.lang.annotation.Aspect; **import** org.springframework.stereotype.Component;

@Aspect

@Component

**public class** AfterAnnAspect { // After-returning advice method

@AfterReturning(pointcut = "execution(\* com.Aspect.After.StudentService.\*(..))", returning = "result") **public void** logAfterReturning(Object result) {

System.***out***.println("Executing after-returning advice. Result: " + result); }}

**AppConfiguration File**

**package** com.Aspect.After;

**import** org.springframework.context.annotation.Bean; **import** org.springframework.context.annotation.ComponentScan; **import** org.springframework.context.annotation.Configuration; **import** org.springframework.context.annotation.EnableAspectJAutoProxy; **import** com.Aspect.After.StudentService;

@Configuration

@ComponentScan(basePackages = "com.Aspect.After")

@EnableAspectJAutoProxy

**public class** AfterAppConfig {

@Bean

**public** StudentService studentServices()

{

**return new** StudentService();

}

}

**AnnotationMain Class**

**package** com.Aspect.After;

**import** org.springframework.context.annotation.AnnotationConfigApplicationContext; **public class** AfterAnnMain { **public static void** main(String[] args) {

// Load the Spring configuration

AnnotationConfigApplicationContext context = **new**

AnnotationConfigApplicationContext(AfterAppConfig.**class**);

// Get the StudentService bean from the context

StudentService studentService = context.getBean(StudentService.**class**);

// Create a sample student Student student = **new** Student(); student.setStudentId("123"); student.setName("NMITD");

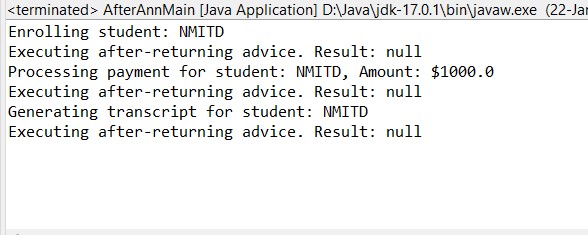
// Call methods to trigger AOP with after, after-returning, and after-throwing advice studentService.enrollStudent(student); studentService.processPayment(student, 1000); studentService.generateTranscript(student);

// Close the context context.close();

}

}

**Output:**



**Practical No.8.4 : Write a program to demonstrate Spring AOP – after throwing advice**.

**CODE:**

**Student Class**

**package** com.Aspect.After; **public class** Student { **private** String studentId; **private** String name; **public** String getStudentId() { **return** studentId;

}

**public void** setStudentId(String studentId) { **this**.studentId = studentId;

}

**public** String getName() { **return** name;

}

**public void** setName(String name) { **this**.name = name;

} }

**Another Business Logic Class**

**package** com.Aspect.After; **public class** StudentService {

**public void** enrollStudent(Student student) {

// Logic for enrolling a student

System.***out***.println("Enrolling student: " + student.getName());

}

**public void** processPayment(Student student, **double** amount) {

// Logic for processing payment

System.***out***.println("Processing payment for student: " + student.getName() + ", Amount: $" + amount);

}

**public void** generateTranscript(Student student) {

// Logic for generating a transcript

System.***out***.println("Generating transcript for student: " + student.getName()); }}

**AfterXmlAspect Class**

**package** com.Aspect.After; **public class** AfterAspect { // After-throwing advice method

**public void** logAfterThrowing(Exception ex) {

System.***out***.println("Executing after-throwing advice. Exception: " + ex.getMessage()); }

}

**After Configuration File**

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

<beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xsi:schemaLocation=*"*

[*http://www.springframework.org/schema/beans*](http://www.springframework.org/schema/beans)

*http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop.xsd"*>

<!-- Define the StudentService bean -->

<bean id=*"studentService"* class=*"com.Aspect.After.StudentService"*/>

<!-- Define the LoggingAspect bean -->

<bean id=*"loggingAspect"* class=*"com.Aspect.After.AfterAspect"*/>

<!-- Configure AOP with After, After-Returning, and After-Throwing advice --> <aop:config>

<!-- Define the pointcut for the methods in StudentService --> <aop:pointcut id=*"studentServiceMethods"* expression=*"execution(\**

*com.Aspect.After.StudentService.\*(..))"*/>

<!-- Define the after-throwing advice using the pointcut -->

<aop:aspect ref=*"loggingAspect"*>

<aop:after-throwing method=*"logAfterThrowing"* pointcut-ref=*"studentServiceMethods"* throwing=*"ex"*/> </aop:aspect>

</aop:config>

</beans>

**AfterMain Class**

**package** com.Aspect.After;

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

**import** org.springframework.context.support.ClassPathXmlApplicationContext; **import** com.Aspect.After.Student; **import** com.Aspect.After.StudentService; **public class** AfterXmlMain

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

{

// Load the Spring XML configuration

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

// Get the StudentService bean from the context

StudentService studentService = (StudentService) context.getBean("studentService"); // Create a sample student Student student = **new** Student(); student.setStudentId("123"); student.setName("NMITD");

// Call methods to trigger AOP with after, after-returning, and after-throwing advice studentService.enrollStudent(student); studentService.processPayment(student, 1000); studentService.generateTranscript(student);

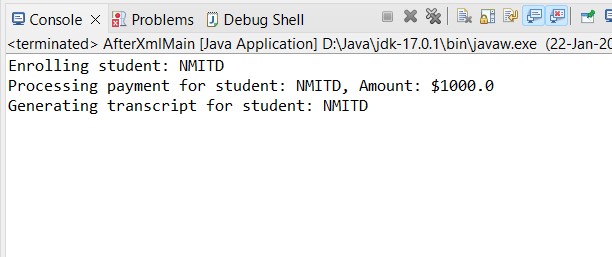
// Close the context

((ClassPathXmlApplicationContext) context).close();

}

}

**Output:**



**AfterAnnotation Implmenation**

**AfterAnnClass**

**package** com.Aspect.After; **import** org.aspectj.lang.annotation.After; **import** org.aspectj.lang.annotation.AfterReturning; **import** org.aspectj.lang.annotation.AfterThrowing; **import** org.aspectj.lang.annotation.Aspect; **import** org.springframework.stereotype.Component;

@Aspect

@Component

**public class** AfterAnnAspect { // After-throwing advice method

@AfterThrowing(pointcut = "execution(\* com.Aspect.After.StudentService.\*(..))", throwing = "ex")

**public void** logAfterThrowing(Exception ex)

{

System.***out***.println("Executing after-throwing advice. Exception: " + ex.getMessage()); }

}

**AppConfiguration File**

**package** com.Aspect.After;

**import** org.springframework.context.annotation.Bean; **import** org.springframework.context.annotation.ComponentScan; **import** org.springframework.context.annotation.Configuration; **import** org.springframework.context.annotation.EnableAspectJAutoProxy; **import** com.Aspect.After.StudentService;

@Configuration

@ComponentScan(basePackages = "com.Aspect.After")

@EnableAspectJAutoProxy

**public class** AfterAppConfig {

@Bean

**public** StudentService studentServices()

{

**return new** StudentService();

}

}

**AnnotationMain Class**

**package** com.Aspect.After;

**import** org.springframework.context.annotation.AnnotationConfigApplicationContext; **public class** AfterAnnMain { **public static void** main(String[] args) {

// Load the Spring configuration

AnnotationConfigApplicationContext context = **new**

AnnotationConfigApplicationContext(AfterAppConfig.**class**);

// Get the StudentService bean from the context

StudentService studentService = context.getBean(StudentService.**class**);

// Create a sample student Student student = **new** Student(); student.setStudentId("123"); student.setName("NMITD");

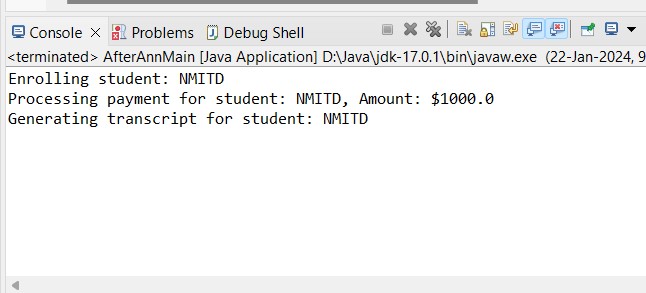
// Call methods to trigger AOP with after, after-returning, and after-throwing advice studentService.enrollStudent(student); studentService.processPayment(student, 1000); studentService.generateTranscript(student);

// Close the context context.close();

}

}

**Output:**



**Practical No.8.5 : Write a program to demonstrate Spring AOP – around advice.**

**CODE:**

**PaymentClass**

**package** com.Aspect.Around; **public class** Payment { **private** String transactionId; **private double** amount; **public** String getTransactionId() { **return** transactionId;

}

**public void** setTransactionId(String transactionId) { **this**.transactionId = transactionId;

}

**public double** getAmount() { **return** amount;

}

**public void** setAmount(**double** amount) { **this**.amount = amount;

}

// Getters and setters

}

**otherBusnisslogic Class**

**package** com.Aspect.Around; **public class** PaymentService {

**public void** processPayment(Payment payment)

{

// Simulate processing payment

System.***out***.println("Processing payment. Transaction ID: " + payment.getTransactionId() + ", Amount: $" + payment.getAmount());

}

}

**AroundAspect Class**

**package** com.Aspect.Around;

**import** org.aspectj.lang.ProceedingJoinPoint;

**public class** AroundXml

{

**public** Object logPayment(ProceedingJoinPoint joinPoint) **throws** Throwable {

// Log before processing payment

System.***out***.println("Audit Log: Before processing payment.");

// Proceed with the payment processing

Object result = joinPoint.proceed();

// Log after processing payment

System.***out***.println("Audit Log: After processing payment."); **return** result;}}

**AroundXMLConfiguration file**

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

<beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop.xsd"*>

<!-- Define the PaymentService bean -->

<bean id=*"paymentService"* class=*"com.Aspect.Around.PaymentService"*/>

<!-- Define the AuditLogAspect bean -->

<bean id=*"auditLogAspect"* class=*"com.Aspect.Around.AroundXml"*/>

<!-- Configure AOP with Around advice -->

<aop:config>

<!-- Define the pointcut for the methods in PaymentService --> <aop:pointcut id=*"paymentServiceMethods"* expression=*"execution(\* com.Aspect.Around.PaymentService.\*(..))"*/> <!-- Define the around advice using the pointcut -->

<aop:aspect ref=*"auditLogAspect"*>

<aop:around method=*"logPayment"* pointcut-ref=*"paymentServiceMethods"*/>

</aop:aspect>

</aop:config>

</beans>

**MainClass**

**package** com.Aspect.Around;

// Main.java

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

**import** org.springframework.context.support.ClassPathXmlApplicationContext; **public class** AroundMain { **public static void** main(String[] args) {

// Load the Spring XML configuration

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

// Get the PaymentService bean from the context

PaymentService paymentService = context.getBean(PaymentService.**class**);

// Create a sample payment Payment payment = **new** Payment(); payment.setTransactionId("TX123"); payment.setAmount(500.00);

// Call the method to trigger AOP with around advice paymentService.processPayment(payment);

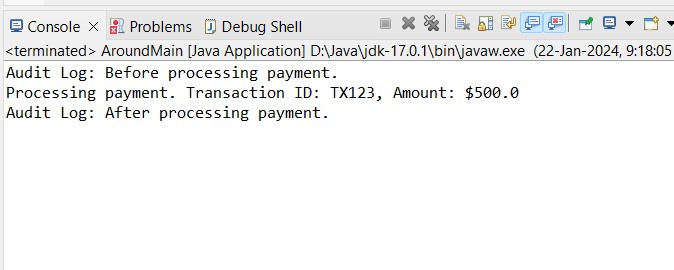
// Close the context

((ClassPathXmlApplicationContext) context).close();

}

}

**Output:**



**Around Annotation Implementation**

**AroundAspect Class**

**package** com.Aspect.Around; **import** org.aspectj.lang.ProceedingJoinPoint; **import** org.aspectj.lang.annotation.Around; **import** org.aspectj.lang.annotation.Aspect; **import** org.springframework.stereotype.Component;

@Aspect

@Component

**public class** AroundAnnClass {

@Around("execution(\* com.Aspect.Around.PaymentService.processPayment(..))") **public** Object logPayment(ProceedingJoinPoint joinPoint) **throws** Throwable {

// Log before processing payment

System.***out***.println("Audit Log: Before processing payment.");

// Proceed with the payment processing

Object result = joinPoint.proceed();

// Log after processing payment

System.***out***.println("Audit Log: After processing payment."); **return** result;

}

}

**AroundAnn Configuration Class**

**package** com.Aspect.Around;

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

// AppConfig.java **import** org.springframework.context.annotation.ComponentScan; **import** org.springframework.context.annotation.Configuration; **import** org.springframework.context.annotation.EnableAspectJAutoProxy;

@Configuration

@ComponentScan(basePackages = "com.Aspect.Around")

@EnableAspectJAutoProxy **public class** AroundAnnConfig {

@Bean

**public** PaymentService paymentService()

{

**return new** PaymentService();

}}

**AroundAnnMain Class**

**package** com.Aspect.Around;

**import** org.springframework.context.annotation.AnnotationConfigApplicationContext; **public class** AoundAnnMain { **public static void** main(String[] args) {

// Load the Spring configuration

AnnotationConfigApplicationContext context = **new**

AnnotationConfigApplicationContext(AroundAnnConfig.**class**);

// Get the PaymentService bean from the context

PaymentService paymentService = context.getBean(PaymentService.**class**);

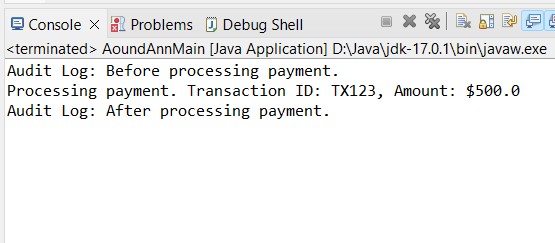
// Create a sample payment Payment payment = **new** Payment(); payment.setTransactionId("TX123"); payment.setAmount(500.00);

// Call the method to trigger AOP with around advice paymentService.processPayment(payment);

// Close the context context.close();

} }

**Output:**



**Practical No.8.6 : Write a program to demonstrate Spring AOP – point cuts.**

**CODE:**

**UserService.java**

**package** com.Aspect.pointcutAspect;

**public** **class** UserService { **public** **void** createUser(String username, String password) { System.***out***.println("Creating user: " + username); } **public** **void** updateUser(String username) {

System.***out***.println("Updating user: " + username); } **public** **void** deleteUser(String username) {

System.***out***.println("Deleting user: " + username); } **public** **void** nonOperation() { // This method will not be logged

}

}

**PointcutAnn.java**

**package** com.Aspect.pointcutAspect;

**import** org.aspectj.lang.JoinPoint; **import** org.aspectj.lang.annotation.Aspect; **import** org.aspectj.lang.annotation.Before; **import** org.springframework.stereotype.Component;

@Component @Aspect

**public** **class** PointcutAnn {

@Before("execution(\* com.Aspect.pointcutAspect..\*(..))") **public** **void** logAllServiceMethods(JoinPoint joinPoint) { System.***out***.println("Logging all service methods: " + joinPoint.getSignature().toShortString()); }

@Before("execution(\* com.Aspect.pointcutAspect..create\*(..))") **public** **void** logCreateMethods(JoinPoint joinPoint) { System.***out***.println("Logging create methods: " +

joinPoint.getSignature().toShortString());

}

@Before("execution(\* com.Aspect.pointcutAspect..update\*(..))") **public** **void** logUpdateMethods(JoinPoint joinPoint) { System.***out***.println("Logging update methods: " +

joinPoint.getSignature().toShortString());

}

}

**Conf.xml**

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

<beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:aop=*"http://www.springframework.org/schema/aop"* xsi:schemaLocation=*"http://www.springframework.org/schema/beans*   *http://www.springframework.org/schema/beans/spring-beans.xsd*   *http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop.xsd"*>

<!-- Define the UserService bean -->

<bean id=*"userService"* class=*"com.Aspect.pointcutAspect.UserService"*/>

<!-- Define the LoggingAspect bean -->

<bean id=*"loggingAspect"* class=*"com.Aspect.pointcutAspect.PointcutAnn"*/>

<!-- Configure AOP -->

<aop:config>

<aop:aspect ref=*"loggingAspect"*>

<!-- Pointcut for all methods in the service package --> <aop:pointcut id=*"allServiceMethods"* expression=*"execution(\* com.Aspect.pointcutAspect.UserService.\*(..))"*/>

<aop:before method=*"logAllServiceMethods"* pointcut-ref=*"allServiceMethods"*/>

<!-- Pointcut for methods starting with 'create' in the service package -->

<aop:pointcut id=*"createMethods"* expression=*"execution(\* com.Aspect.pointcutAspect.UserService.create\*(..))"*/>

<aop:before method=*"logCreateMethods"* pointcut-ref=*"createMethods"*/>

<!-- Pointcut for methods starting with 'update' in the service package -->

<aop:pointcut id=*"updateMethods"* expression=*"execution(\* com.Aspect.pointcutAspect.UserService.update\*(..))"*/>

<aop:before method=*"logUpdateMethods"* pointcut-ref=*"updateMethods"*/>

</aop:aspect>

</aop:config>

</beans>

**Pointcutmain.java**

**package** com.Aspect.pointcutAspect;

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

**public** **class** Pointcutmain { **public** **static** **void** main(String[] args) { // Load Spring configuration from XML **try** (ClassPathXmlApplicationContext context = **new**

ClassPathXmlApplicationContext("Conf.xml")) {

// Get the UserService bean from the context

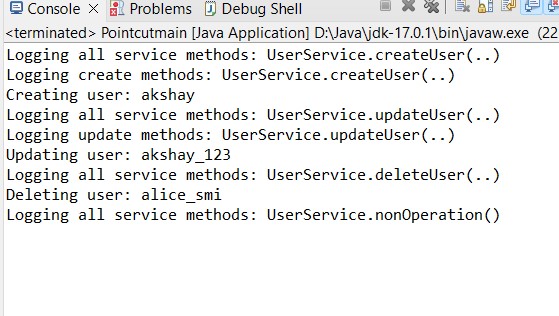
UserService userService = context.getBean(UserService.**class**);

// Use the UserService methods userService.createUser("akshay", "password123"); userService.updateUser("akshay\_123"); userService.deleteUser("alice\_smi"); userService.nonOperation(); // This method will not be logged

} }

}

**Output:**



## PRACTICAL NO.9 Spring JDBC

**THEORY:**

**JDBC TEMPLATE:**

* Spring **JdbcTemplate** is a powerful mechanism to connect to the database and execute SQL queries.
* It internally uses JDBC api, but eliminates a lot of problems of JDBC API.
* [JDBC](https://www.geeksforgeeks.org/introduction-to-jdbc/) (Java Database Connectivity) is an application programming interface **(API)** that defines how a client may access a database.
* It is a data access technology used for Java database connectivity.
* It provides methods to query and update data in a database and is oriented toward relational databases.
* JDBC offers a natural Java interface for working with **SQL**.
* JDBC is needed to provide a “Pure Java” solution for application development.
* JDBC API uses JDBC drivers to connect with the database.

**Practical No.9.1 : Write a program to insert, update and delete records from the given table.**

**CODE:**

**POJO CLASS**

**package** com.jdbc.jdbcdemo; **public class** logindata

{

**private int** id; **private** String username; **private** String password; **public** logindata()

{ **super**(); }

**public** logindata(**int** id, String username, String password)

{ **super**();

**this**.id = id;

**this**.username = username;

**this**.password = password;

}

**public int** getId() { **return** id;

}

**public void** setId(**int** id) { **this**.id = id;

}

**public** String getUsername() { **return** username;

}

**public void** setUsername(String username) { **this**.username = username;

}

**public** String getPassword() { **return** password;

}

**public void** setPassword(String password) { **this**.password = password;

}

@Override

**public** String toString() {

**return** "logindata [id=" + id + ", username=" + username + ", password=" + password + "]";

} }

**LOGINDAO INTERFACE**

**package** com.jdbc.jdbcdao; **import** com.jdbc.jdbcdemo.logindata;

**public interface** logindaointerface

{

**public int** insert(logindata logindata); **public int** modify(logindata logindata);

**public int** delete(**int** id); }

**IMPLMENTATION CLASS OF DAO INTERFACE**

package com.jdbc.jdbcdao;

import org.springframework.jdbc.core.JdbcTemplate; import com.jdbc.jdbcdemo.logindata;

public class logindatadaoimp implements logindaointerface

{

private JdbcTemplate jdbcTemplate; public int insert(logindata logindata)

{

String query="insert into logindata(id,username,password)values(?,?,?)"; int rest=

this.jdbcTemplate.update(query,logindata.getId(),logindata.getUsername(),logindata.getPassword()); return rest; }

@Override

public int modify(logindata logindata) {

String query="update logindata set username=?, password=? where id=?";

int re= this.jdbcTemplate.update(query,logindata.getUsername(),logindata.getPassword(),logindata.getId()); return re;

}

public JdbcTemplate getJdbcTemplate()

{

return jdbcTemplate;

} public void setJdbcTemplate(JdbcTemplate jdbcTemplate)

{

this.jdbcTemplate = jdbcTemplate;

}

@Override

public int delete(int id)

{

String query= "delete from logindata where id=?"; int res= this.jdbcTemplate.update(query,id); return res;

} }

**XMLCONFIGURATION**

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

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

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd"*>

<bean class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"* name=*"Driv"*>

<property name=*"driverClassName"* value=*"com.mysql.cj.jdbc.Driver"*/>

<property name=*"url"* value=*"jdbc:mysql://localhost:3306/logindb"*/>

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

<property name=*"password"* value=*"root"*/>

</bean>

<bean class=*"org.springframework.jdbc.core.JdbcTemplate"* name=*"jdbctemplate"* p:dataSourceref=*"Driv"*/>

<bean class=*"com.jdbc.jdbcdao.logindatadaoimp"* name=*"logindao"*>

<property name=*"jdbcTemplate"* ref=*"jdbctemplate"*/>

</bean>

</beans>

**MAINCLASS**

package com.jdbc.jdbcdemo; import java.util.Scanner;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext; import org.springframework.jdbc.core.JdbcTemplate; import com.jdbc.jdbcdao.logindaointerface; public class loginMain {

public static void main(String[] args)

{

System.out.println("welcome to JDBC template");

ApplicationContext context=new ClassPathXmlApplicationContext("Appconf.xml"); logindaointerface ldb=context.getBean("logindao",logindaointerface.class);

/\*Insert data in database logindata lodb= new logindata();

lodb.setId(17); lodb.setUsername("ruhi"); lodb.setPassword("ruchi"); int result=ldb.insert(lodb);

System.out.println("record inserted successfully......"+result);

System.out.printf("record inserted printed"+lodb);

\*/

//update data in database logindata log= new logindata(); log.setId(3); log.setUsername("database"); log.setPassword("hello"); int resu=ldb.modify(log);

System.out.println("record updated successfully......"+resu);

System.out.printf("after update record printed"+log);

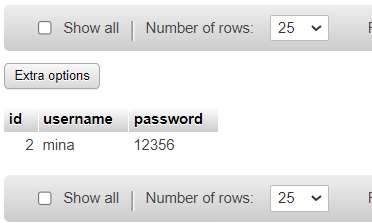
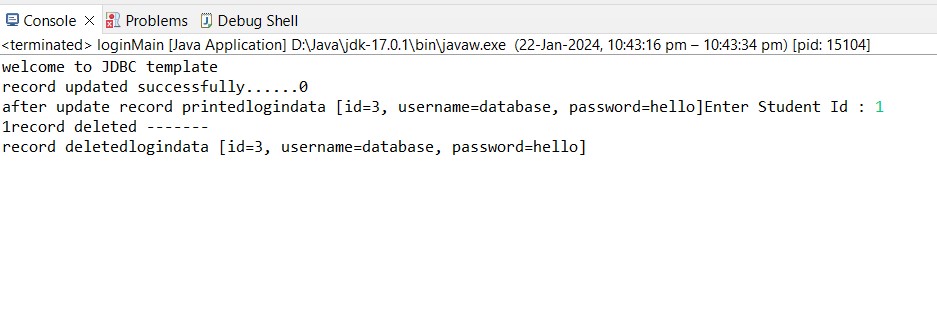
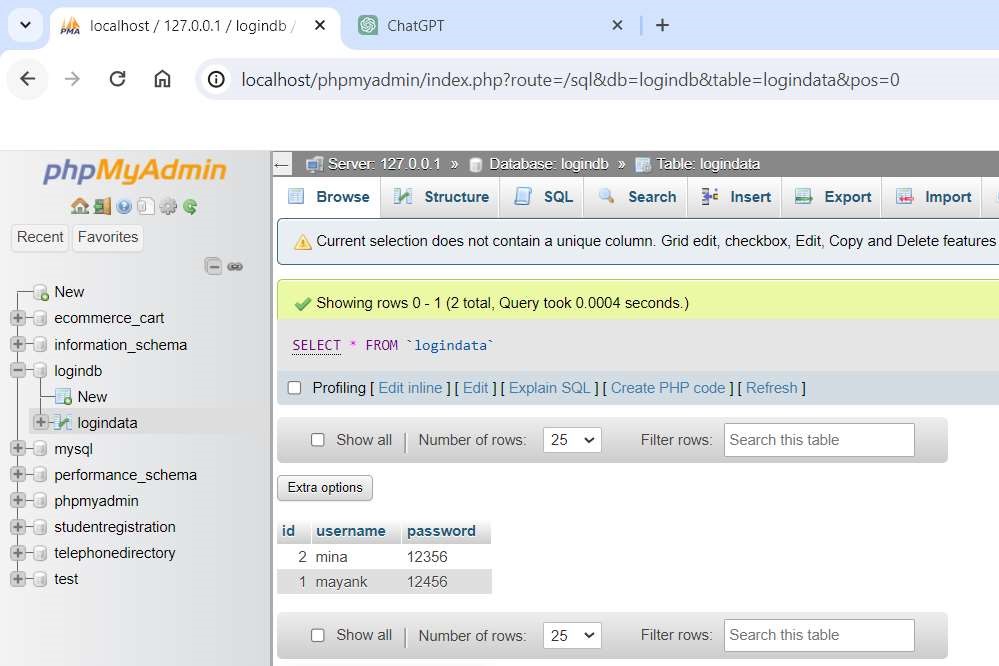
//Delete data in database

Scanner input=new Scanner(System.in); System.out.print("Enter Student Id : "); int id=input.nextInt(); int result=ldb.delete(id);

System.out.println(result+"record deleted -------");

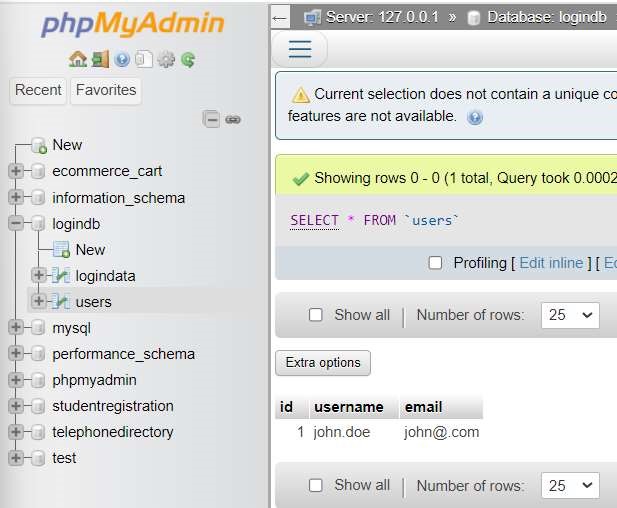
System.out.printf("record deleted"+log); } }

**Output:**



**Practical No. 9.2 : Write a program to demonstrate Prepared Statement in Spring JdbcTemplate.**

**CODE:**



**JdbcTemplateExample.java**

**package** springid;

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

**import** org.springframework.context.support.ClassPathXmlApplicationContext; **import** org.springframework.jdbc.core.JdbcTemplate;

**public** **class** JdbcTemplateExample {

**public** **static** **void** main(String[] args) { // Load Spring configuration

ApplicationContext context = **new**

ClassPathXmlApplicationContext("applicationContext.xml"); // Get JdbcTemplate bean

JdbcTemplate jdbcTemplate = context.getBean("jdbcTemplate", JdbcTemplate.**class**);

// Create and execute a Prepared Statement

String username = "mina.doe";

String email = "mina@.com";

String sql = "INSERT INTO users (id, username, email) VALUES (?, ?, ?)"; **int** id = 2; // Provide a valid value for 'id' jdbcTemplate.update(sql, id, username, email);

System.***out***.println("User inserted successfully.");

// Close the application context

((ClassPathXmlApplicationContext) context).close();

}

}

**ApplicationContext.xml**

<!-- applicationContext.xml -->

<beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:context=*"http://www.springframework.org/schema/context"* xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- Data Source Configuration -->

<bean id=*"dataSource"*

class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*> <property name=*"driverClassName"* value=*"com.mysql.cj.jdbc.Driver"* />

<property name=*"url"* value=*"jdbc:mysql://localhost:3306/logindb"* />

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

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

</bean>

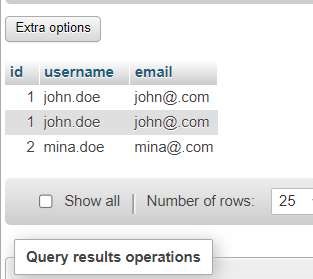
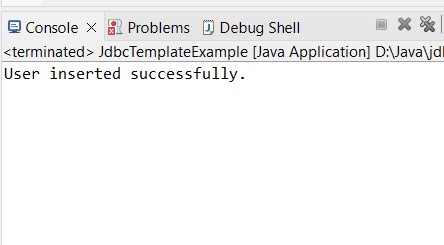
<!-- JdbcTemplate Configuration -->

<bean id=*"jdbcTemplate"* class=*"org.springframework.jdbc.core.JdbcTemplate"*>

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

</bean> </beans>

**Output:**



**Practical No.9.3 : Write a program in Spring JDBC to demonstrate Result Set Extractor Interface.**

**CODE:**

**ResultSetExtractorExample.java**

**package** springid;

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

**import** org.springframework.context.support.ClassPathXmlApplicationContext; **import** org.springframework.jdbc.core.JdbcTemplate; **import** org.springframework.jdbc.core.ResultSetExtractor;

**public** **class** ResultSetExtractorExample {

**public** **static** **void** main(String[] args) { // Load Spring configuration

ApplicationContext context = **new**

ClassPathXmlApplicationContext("applicationContext.xml");

// Get JdbcTemplate bean

JdbcTemplate jdbcTemplate = context.getBean("jdbcTemplate", JdbcTemplate.**class**);

// Execute query with ResultSetExtractor

String sql = "SELECT \* FROM employees";

EmployeeListExtractor employeeListExtractor = **new** EmployeeListExtractor(); jdbcTemplate.query(sql, employeeListExtractor);

// Print the result

System.***out***.println("Employees: " + employeeListExtractor.getEmployees());

// Close the application context

((ClassPathXmlApplicationContext) context).close();

}

}

**Employee.java**

**package** springid;

**public** **class** Employee { **private** **int** id;

**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** **double** getSalary() {

**return** salary;

}

**public** **void** setSalary(**double** salary) { **this**.salary = salary;

}

**private** String name; **private** **double** salary;

// getters and setters

@Override

**public** String toString() {

**return** "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "]"; } }

**EmployeeListExtrator.java package springid;**

**import java.sql.ResultSet; import java.sql.SQLException; import java.util.ArrayList; import java.util.List;**

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

**public class EmployeeListExtractor implements ResultSetExtractor<List<Employee>> {**

**private List<Employee> employees = new ArrayList<>();**

**@Override**

**public List<Employee> extractData(ResultSet rs) throws SQLException,**

**DataAccessException {**

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

**Employee employee = new Employee(); employee.setId(rs.getInt("id")); employee.setName(rs.getString("name")); employee.setSalary(rs.getDouble("salary")); employees.add(employee);**

**}**

**return employees;**

**}**

**public List<Employee> getEmployees() { return employees;**

**}**

**}**

**ApplicatioContext.xml**

<!-- applicationContext.xml -->

<beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:context=*"http://www.springframework.org/schema/context"* xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- Data Source Configuration -->

<bean id=*"dataSource"*

class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*> <property name=*"driverClassName"* value=*"com.mysql.cj.jdbc.Driver"* /> <property name=*"url"* value=*"jdbc:mysql://localhost:3306/logindb"* />

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

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

</bean>

<!-- JdbcTemplate Configuration -->

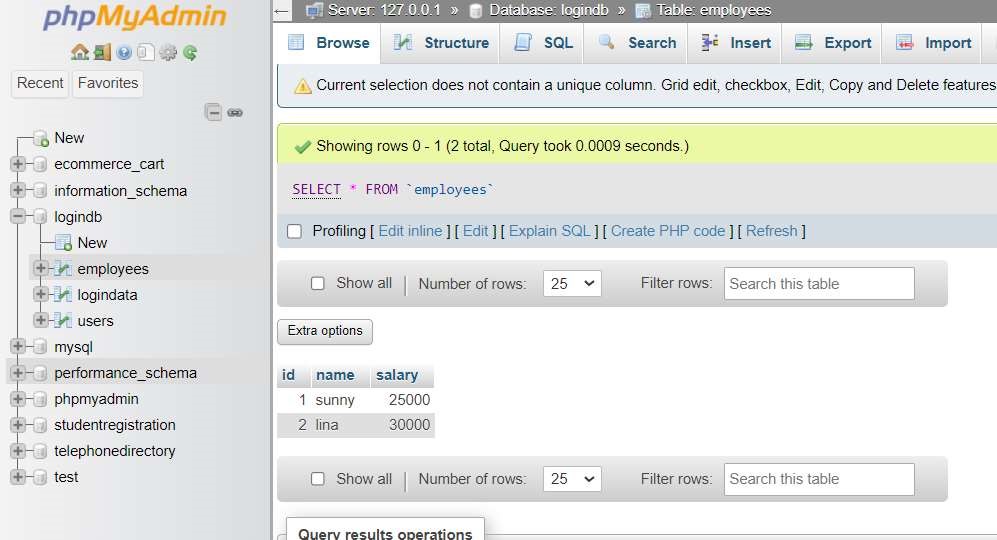
<bean id=*"jdbcTemplate"* class=*"org.springframework.jdbc.core.JdbcTemplate"*>

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

</bean>

</beans>

**Output:**



**Practical No.9.4 : Write a program to demonstrate Row Mapper interface to fetch the records from the database.**

**CODE:**

**Logindata.java**

**package** com.jdbc.jdbcdemo;

**public** **class** logindata { **private** **int** id; **private** String username; **private** String password; **public** logindata() { **super**(); // **TODO** Auto-generated constructor stub

}

**public** logindata(**int** id, String username, String password) {

**super**();

**this**.id = id; **this**.username = username; **this**.password = password;

}

**public** **int** getId() {

**return** id;

}

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

**this**.id = id;

}

**public** String getUsername() { **return** username;

}

**public** **void** setUsername(String username) { **this**.username = username;

}

**public** String getPassword() { **return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

@Override

**public** String toString() {

**return** "logindata [id=" + id + ", username=" + username + ", password=" + password + "]";

}

}

**Logindaointerface.java**

**package** com.jdbc.jdbcdao; **import** java.util.List; **import** com.jdbc.jdbcdemo.logindata; **public** **interface** logindaointerface { **public** **int** insert(logindata logindata); **public** **int** modify(logindata logindata); **public** **int** delete(**int** id); **public** logindata getlogindata(**int** id); **public** List<logindata> getallobject();

}

**Logindatadaoimp.java**

**package** com.jdbc.jdbcdao; **import** java.util.List;

**import** org.springframework.jdbc.core.JdbcTemplate; **import** org.springframework.jdbc.core.RowMapper; **import** com.jdbc.jdbcdemo.logindata;

**public** **class** logindatadaoimp **implements** logindaointerface{

**private** JdbcTemplate jdbcTemplate; **public** **int** insert(logindata logindata) {

String query="insert into logindata(id,username,password)values(?,?,?)"; **int** rest=

**this**.jdbcTemplate.update(query,logindata.getId(),logindata.getUsername(),logindata.getPass word());

**return** rest; }

**public** JdbcTemplate getJdbcTemplate() { **return** jdbcTemplate;

}

**public** **void** setJdbcTemplate(JdbcTemplate jdbcTemplate) {

**this**.jdbcTemplate = jdbcTemplate; }

@Override

**public** **int** modify(logindata logindata) {

String query="update logindata set username=?, password=? where id=?";

**int** re=

**this**.jdbcTemplate.update(query,logindata.getUsername(),logindata.getPassword(),logindata.g etId());

**return** re; } @Override

**public** **int** delete(**int** id) {

String query= "delete from logindata where id=?";

**int** res= **this**.jdbcTemplate.update(query,id); **return** res; }

@Override

**public** logindata getlogindata(**int** id) {

String query= "select \* from logindata where id=?"; RowMapper<logindata>rowMapper= **new** RowMapperimp();

logindata logindata=**this**.jdbcTemplate.queryForObject(query,rowMapper,id); **return** logindata; }

@Override

**public** List<logindata> getallobject() { String query= "select \* from logindata";

List<logindata> ldata=**this**.jdbcTemplate.query(query,**new** RowMapperimp()); **return** ldata; }

}

**RowMapperimp.java**

**package** com.jdbc.jdbcdao; **import** java.sql.ResultSet; **import** java.sql.SQLException;

**import** org.springframework.jdbc.core.RowMapper; **import** com.jdbc.jdbcdemo.logindata;

**public** **class** RowMapperimp **implements** RowMapper<logindata> {

@Override

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

// **TODO** Auto-generated method stub logindata logindata=**new** logindata(); logindata.setId(rs.getInt(1)); logindata.setUsername(rs.getString(2));

logindata.setPassword(rs.getString(3));

**return** logindata;

}

}

**loginMain.java**

**package** com.jdbc.jdbcdemo; **import** java.util.List; **import** java.util.Scanner;

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

**import** org.springframework.context.support.ClassPathXmlApplicationContext; **import** com.jdbc.jdbcdao.logindaointerface; **public** **class** loginMain {

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

{

System.***out***.println("welcome to JDBC template");

ApplicationContext context=**new** ClassPathXmlApplicationContext("Appconf.xml"); logindaointerface ldb=context.getBean("logindao",logindaointerface.**class**);

// Insert data in database logindata lodb= **new** logindata(); lodb.setId(3); lodb.setUsername("ruhi"); lodb.setPassword("ruchi"); **int** result=ldb.insert(lodb);

System.***out***.println("record inserted successfully......"+result);

System.***out***.printf("record inserted printed"+lodb);

//update data in database logindata log= **new** logindata(); Scanner inp=**new** Scanner(System.***in***);

System.***out***.print("\n"+"Enter Student Id which you want to update : "); **int** in=inp.nextInt(); log.setId(in); log.setUsername("database"); log.setPassword("hello"); **int** resu=ldb.modify(log);

System.***out***.println("\n"+"record updated successfully......"+"/"+resu);

System.***out***.printf("\n"+"after update record printed"+log);

//Delete data in database

Scanner input=**new** Scanner(System.***in***);

System.***out***.print("\n"+"Enter Student Id which you want to delete : "); **int** id=input.nextInt(); **int** reslt=ldb.delete(id);

System.***out***.println("\n"+result+"record deleted -------"+"/");

System.***out***.printf("record deleted"+reslt+"\n");

//select record from database

Scanner inpu=**new** Scanner(System.***in***);

System.***out***.print("\n"+"Enter Student Id which you want to findout : "); **int** i=inpu.nextInt();

logindata login=ldb.getlogindata(i);

System.***out***.println("\n"+"record accesss-------"+"/");

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

//select all record from database

List<logindata> lon=ldb.getallobject();

System.***out***.println("\n"+"all record accesss-------"+"/");

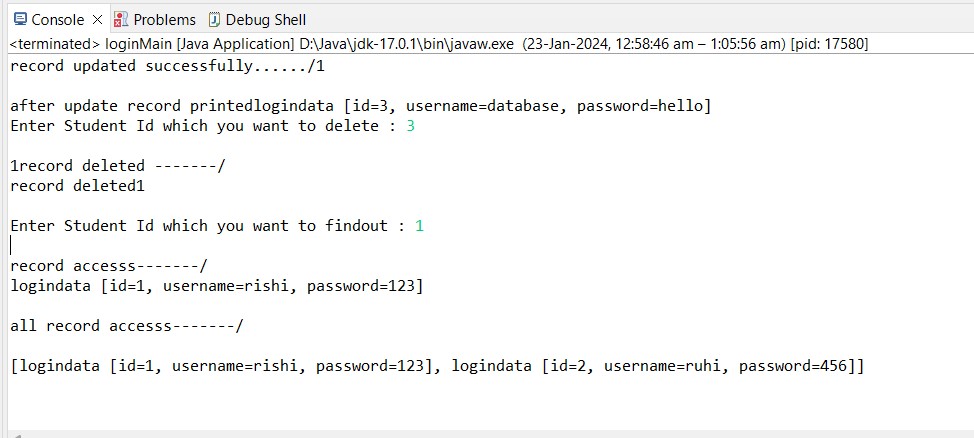
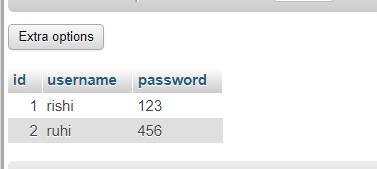
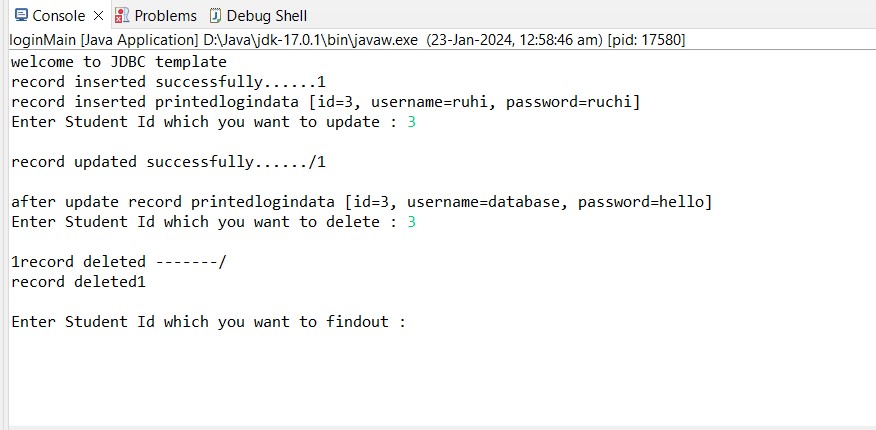
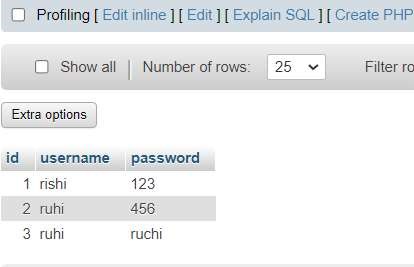
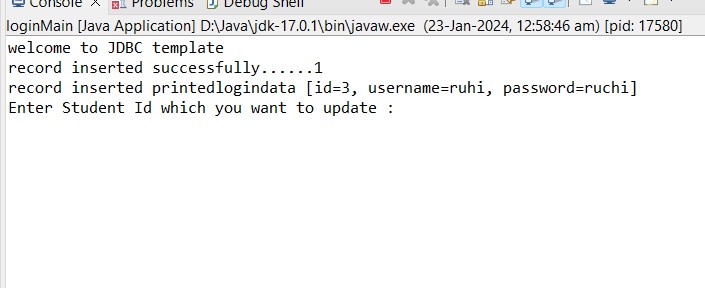
System.***out***.printf("\n");

System.***out***.println(lon+"\n");

}

}

**Output:**



## PRACTICAL NO.10 : SPRING BOOT

**THEORY:**

**SPRING BOOT:**

* Spring Boot is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based applications.
* It is a Spring module that provides the **RAD (***Rapid Application Development***)** feature to the Spring Framework. It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.
* In short, Spring Boot is the combination of **Spring Framework** and **Embedded Servers**.  In Spring Boot, there is no requirement for XML configuration (deployment descriptor).
* It uses convention over configuration software design paradigm that means it decreases the effort of the developer.

**Practical No.10.1 : Write a program to create a simple Spring Boot application that prints a message.**

**CODE:**

**SpringBootDemoApplication.java**

**package** springboot;

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

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

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

SpringApplication.*run*(SpringBootDemoApplication.**class**, args);

System.***out***.println("Hello, Spring Boot!");

}}

**Output:**

