

## PRACTICAL 1

### **Assignments on Java Generics:**

A **generic class** in Java is a class that allows you to define methods, fields, and data types with type parameters, enabling the class to work with objects of different types. It provides a way to define **type-safe** classes without the need to specify the exact data types upfront. Instead, the class uses type parameters (often denoted as T, E, K, V, etc.) that are specified when the class is instantiated.

Detailed Explanation:

- **Type Parameters:** These are placeholders for data types. The type parameter can be any valid Java type, such as a class (Integer, String) or interface (List, Comparable).
- **Reusability:** The same generic class can be used with different types, reducing redundancy in the code and increasing flexibility.
- **Compile-time Type Safety:** Generics provide type checking at compile time, ensuring that only the specified types are used and preventing **ClassCastException** at runtime.

### **AIM 1: Write a Java Program to demonstrate a Generic Class.**

#### **Code:**

```

→ import java.util.Scanner;

class Stack<E> {
    E a[];
    int top;

    @SuppressWarnings("unchecked")
    Stack() {
        // Creates a generic array using Object casting
        a = (E[]) new Object[100];
        top = -1;
    }

    void push(E data) {
        a[++top] = data;
    }

    E pop() {
        return a[top--];
    }

    boolean hasElements() {
        return top != -1;
    }
}

class Student {

```

```

String name;
int standard;

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

@Override
public String toString() {
    return name + " " + standard;
}

class Check2 {
    public static void main(String arg[]) {
        Scanner sc = new Scanner(System.in);

        Stack<Integer> si = new Stack<>();
        Stack<Double> sd = new Stack<>();
        Stack<Student> ss = new Stack<>();

        // === Integer Stack Input ===
        System.out.print("Enter number of integers: ");
        int ni = sc.nextInt();
        System.out.println("Enter " + ni + " integers:");
        for (int i = 0; i < ni; i++) {
            si.push(sc.nextInt());
        }

        // === Double Stack Input ===
        System.out.print("\nEnter number of doubles: ");
        int nd = sc.nextInt();
        System.out.println("Enter " + nd + " doubles:");
        for (int i = 0; i < nd; i++) {
            sd.push(sc.nextDouble());
        }

        // === Student Stack Input ===
        System.out.print("\nEnter number of students: ");
        int ns = sc.nextInt();
        sc.nextLine(); // consume newline left over from nextInt()

        for (int i = 0; i < ns; i++) {
            System.out.print("Enter student name: ");
            String name = sc.nextLine();
            System.out.print("Enter standard (class): ");
            int std = sc.nextInt();
            sc.nextLine(); // consume newline
            ss.push(new Student(name, std));
        }
    }
}

```

```

    }

// === Display Results (LIFO Order) ===
System.out.println("\nintegers...");
while (si.hasElements()) {
    System.out.println(si.pop());
    System.out.println("Top index now: " + si.top);
}

System.out.println("\ndoubles...");
while (sd.hasElements()) {
    System.out.println(sd.pop());
    System.out.println("Top index now: " + sd.top);
}

System.out.println("\nstudents...");
while (ss.hasElements()) {
    System.out.println(ss.pop());
    System.out.println("Has more? " + ss.hasElements());
}

sc.close();
}
}

```

**Output**

```

Enter number of integers: 3
Enter 3 integers:
10
20
30

Enter number of doubles: 3
Enter 3 doubles:
1.5
2.9
3.9

Enter number of students: 2
Enter student name: champ
Enter standard (class): 9
Enter student name: ayush
Enter standard (class): 6

integers...
30
Top index now: 1
20
Top index now: 0
10
Top index now: -1

doubles...
3.9
Top index now: 1
2.9
Top index now: 0
1.5
Top index now: -1

students...
ayush 6
Has more? true
champ 9
Has more? false

```

**AIM 2: Write a Java Program to demonstrate Generic Methods.**

A **generic method** in Java is a method that is defined with one or more **type parameters**, allowing it to work with different data types while maintaining **compile-time type safety**. Unlike a generic class, a generic method can be declared inside **both generic and non-generic classes**, and its type parameters are limited only to that specific method.

Key Characteristics:

- Type parameters are declared **before the return type** of the method.
- The same method can process different data types without code duplication.
- Type checking is done at **compile time**, reducing runtime errors.
- Explicit type casting is not required.
- Generic methods improve **readability, maintainability, and reusability** of code.

**Code:**

```
→ class GenericMethod {
    public static void main(String[] args) {
        DemoClass demo = new DemoClass();
        demo.<String>genericsMethod("Java Programming");
        demo.<Integer>genericsMethod(25);
    }
}

class DemoClass {
    public <T> void genericsMethod(T data) {
        System.out.println("Generics Method:");
        System.out.println("Data Passed: " + data);
    }
}
```

**OUTPUT**

```
Generics Method:
Data Passed: Java Programming
Generics Method:
Data Passed: 25
```

**AIM 3: Write a Java Program to demonstrate Wildcards in Java Generics.**

**Wildcards** in Java generics are special symbols represented by the question mark (?) that allow more **flexibility** when working with generic types. They are mainly used when the exact type parameter is unknown or when you want a method to accept a **range of types** rather than a single specific type.

Why Use Wildcards?

- To make generic code **more flexible**
- To allow methods to work with different but related types
- To support **polymorphism** in generics
- To avoid unnecessary restrictions caused by exact type matching
  - i. **UpperBound Wildcards**

**CODE:**

```

→ import java.util.Arrays;
import java.util.List;

class UpperBound {
    public static void main(String[] args) {
        List<Integer> list1 = Arrays.asList(3, 5, 6, 8);
        System.out.println("Total sum is:" + sum(list1));

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

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

```

**OUTPUT**

```

Total sum is:22.0
Total sum is:15.299999999999999

```

**ii. LowerBound Wildcards****CODE:**

```

→import java.util.Arrays;
import java.util.List;
class LowerBound
{
    public static void main(String[] args)
    {
        List<Integer> list1 = Arrays.asList(2, 4, 6, 8);
        printOnlyIntegerClassorSuperClass(list1);
        List<Number> list2 = Arrays.asList(4, 5, 3, 5, 8);
        printOnlyIntegerClassorSuperClass(list2);}

    public static void printOnlyIntegerClassorSuperClass( List<? super Integer> list)
    {
        System.out.println(list);
    }
}

```

**OUTPUT**

```

[2, 4, 6, 8]
[4, 5, 3, 5, 8]

```

**iii. UnBound Wildcards****CODE:**

```
→import java.util.Arrays;
import java.util.List;
class UnBounded {
    public static void main(String[] args)
    {
        List<Integer> list1 = Arrays.asList(1, 2, 3);
        List<Double> list2 = Arrays.asList(1.1, 2.2, 3.3);
        printlist(list1);
        printlist(list2);
    }
    private static void printlist(List<?> list)
    {
        System.out.println(list);
    }
}
```

**OUTPUT**

```
[1, 2, 3]
[1.1, 2.2, 3.3]
```

**PRACTICAL 2****Assignments on List Interface**

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

**List** is an interface in Java that belongs to the `java.util` package. It represents an **ordered collection** (also known as a sequence) that allows **duplicate elements** and provides **index-based access** to elements.

Key Features of List:

- Maintains **insertion order**
- Allows **duplicate values**
- Supports **index-based** operations (get, set, add, remove)
- Can store **null** elements (depending on implementation)
- Is part of the **Java Collections Framework**

**CODE:**

```
→import java.util.*;
public class ListIteration {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Integer> al = new ArrayList<>();
        System.out.print("Enter the number of elements: ");
        int n = sc.nextInt();
        sc.nextLine();
        for (int i = 0; i < n; i++) {
            System.out.print("Enter element " + (i + 1) + ": ");
            String input = sc.nextLine();
            al.add(input);      }
        for (Integer str : al) {
            System.out.print(str + " ");
        }
        sc.close();  }}
```

**OUTPUT**

```
C:\MCA\roll114\java\practical 3>java ListIteration
Enter the number of elements: 5
Enter element 1: 10
Enter element 2: 28
Enter element 3: 309
Enter element 4: 490
Enter element 5: 49059
10 28 309 490 49059
```

**AIM 2.** Write a Java program to create List containing list of items and use ListIterator interface to print items present in the list. Also print the list in reverse / backward direction.

A List in Java is an ordered collection that allows duplicate elements. To traverse and manipulate elements of a List in both **forward and backward directions**, Java provides the **ListIterator** interface. The **ListIterator** interface is used with classes that implement the List interface (such as ArrayList or

**LinkedList).** It allows:

- Forward traversal using hasNext() and next()
- Backward traversal using hasPrevious() and previous()
- Modification of elements during iteration using add(), remove(), and set()

**CODE:**

```
→import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;
import java.util.Scanner;
public class IntegerListIterator {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Integer> numbers = new ArrayList<>();
        System.out.print("Enter the number of integers: ");
        int n = sc.nextInt();
        // Taking integer input from user
        for (int i = 0; i < n; i++) {
            System.out.print("Enter integer " + (i + 1) + ": ");
            int num = sc.nextInt();
            numbers.add(num);
        }
        // Obtaining ListIterator
        ListIterator<Integer> litr = numbers.listIterator();
        System.out.println("\nTraversing in forward direction:");
        while (litr.hasNext()) {
            System.out.print(litr.next() + " ");
        }
        System.out.println("\n\nTraversing in backward direction:");
        while (litr.hasPrevious()) {
            System.out.print(litr.previous() + " ");
        }
        sc.close(); }}
```

**OUTPUT:**

```
C:\MCA\roll14\java\practical 3>java IntegerListIterator
Enter the number of integers: 5
Enter integer 1: 40
Enter integer 2: 29
Enter integer 3: 394
Enter integer 4: 2383
Enter integer 5: 283

Traversing in forward direction:
40 29 394 2383 283

Traversing in backward direction:
283 2383 394 29 40
```

**PRACTICAL 3****Assignments on Set Interface**

**AIM 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.

A **Set** in Java is a collection that **does not allow duplicate elements** and does **not guarantee indexing**. To traverse elements of a Set, the **Iterator interface** is commonly used. The Iterator allows only **forward traversal** using `hasNext()` and `next()` methods.

Since a **Set does not support backward traversal** and the Iterator interface itself does not provide a `previous()` method, printing elements in **reverse/backward direction** requires converting the Set into a **List** and then using a **ListIterator**.

**CODE:**

```
→ import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashSet;
import java.util.LinkedHashSet;
import java.util.List;
public class SetInterface {
    public static void main(String[] args) {
        // Initializing a LinkedHashSet with even numbers
        HashSet<Integer> evenNumSet = new LinkedHashSet<>(
            Arrays.asList(4, 6, 24, 8, 30, 10, 46)
        );
        System.out.println("Unsorted Set: " + evenNumSet);
        // Converting Set to List to perform sorting
        List<Integer> numList = new ArrayList<Integer>(evenNumSet);
        Collections.sort(numList);
        // Re-assigning sorted list back to the LinkedHashSet
        evenNumSet = new LinkedHashSet<>(numList);
        System.out.println("Sorted Set:" + evenNumSet);
    }
}
```

```

    // Reversing the list for backward direction
    Collections.reverse(numList);
    System.out.println("Sorted Set In Backward Direction:" + numList);
}

```

**OUTPUT**

```

Unsorted Set: [4, 6, 24, 8, 30, 10, 46]
Sorted Set:[4, 6, 8, 10, 24, 30, 46]
Sorted Set In Backward Direction:[46, 30, 24, 10, 8, 6, 4]

```

**AIM 2.** Write a Java program using Set interface containing list of items and perform the following operations:

- a. Add items in the set.
- b. Insert items of one set in to other set.
- c. Remove items from the set
- d. Search the specified item in the set

The Set interface in Java is a part of the **Java Collections Framework** and is used to store a collection of **unique elements**, meaning duplicate values are not allowed. It does not provide index-based access.

Common implementations of the Set interface include HashSet, LinkedHashSet, and TreeSet. Using the Set interface, we can perform various operations such as **adding elements**, **inserting elements from one set into another**, **removing elements**, and **searching for a specific element**.

**Code:**

```

→ import java.util.*;

public class SetInterface {
    public static void main(String args[]) {
        Set<Integer> numSet = new HashSet<Integer>();

        numSet.add(13);
        numSet.addAll(Arrays.asList(new Integer[] {1, 6, 4, 7, 3, 9, 8, 2, 12, 11, 20}));

        System.out.println("Original Set (numSet):" + numSet);
        System.out.println("\nNumSet Size:" + numSet.size());

        Set<Integer> oddSet = new HashSet<Integer>();
        oddSet.addAll(Arrays.asList(new Integer[] {1, 3, 7, 5, 9}));

        System.out.println("\nOddSet contents:" + oddSet);
        System.out.println("\nnumSet contains element 2:" + numSet.contains(3));
    }
}

```

```

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

// 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);

// 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);

// 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:**

```

Original Set (numSet):[1, 2, 3, 4, 20, 6, 7, 8, 9, 11, 12, 13]

NumSet Size:12

OddSet contents:[1, 3, 5, 7, 9]

numSet contains element 2:true

numSet contains collection oddset:false

Intersection of the numSet & oddSet:[1, 3, 7, 9]
Difference of the numSet & oddSet:[2, 4, 20, 6, 8, 11, 12, 13]
Union of the numSet & oddSet:[1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 20]

```

**PRACTICAL 4****Assignments on Map Interface**

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

The **Map** interface in Java is part of the **Java Collections Framework** and represents a collection of **key-value pairs**, where each key is **unique** and maps to exactly one value. Common implementations include **HashMap**, **LinkedHashMap**, and **TreeMap**.

Using the Map interface, we can perform operations such as **adding entries**, **removing entries**, **searching for a key**, **retrieving values**, **merging maps**, and **printing all keys and values**.

**CODE:**

```
→ import java.util.*;

public class Program4 {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        HashMap<Integer, String> map = new HashMap<Integer, String>();
        HashMap<Integer, String> map2 = new HashMap<Integer, String>();

        // ===== INPUT FOR FIRST HASHMAP =====
        System.out.print("Enter number of entries for first HashMap: ");
        int n1 = sc.nextInt();
        sc.nextLine(); // consume newline

        for (int i = 0; i < n1; i++) {
            System.out.print("Enter key (integer): ");
            int key = sc.nextInt();
            sc.nextLine(); // consume newline
            System.out.print("Enter value (city name): ");
            String value = sc.nextLine();
            map.put(key, value);
        }

        // ===== INPUT FOR SECOND HASHMAP =====
        System.out.print("Enter number of entries for second HashMap: ");
        int n2 = sc.nextInt();
        sc.nextLine(); // consume newline
```

```
for (int i = 0; i < n2; i++) {
    System.out.print("Enter key (integer): ");
    int key = sc.nextInt();
    sc.nextLine(); // consume newline
    System.out.print("Enter value (city name): ");
    String value = sc.nextLine();
    map2.put(key, value);
}

// ===== PUT ALL FROM MAP TO MAP2 =====
map2.putAll(map);

// ===== DISPLAY RESULTS =====
System.out.println("\nHashMap 2 contains: " + map2);
System.out.println("Initial list of elements in first map: " + map);

// ===== FETCH A VALUE FROM MAP =====
System.out.print("\nEnter key to get value from first map: ");
int keyToGet = sc.nextInt();
String fetchedValue = map.get(keyToGet);
System.out.println("Value at key " + keyToGet + ": " + fetchedValue);

// ===== DISPLAY KEY/VALUE PAIRS =====
System.out.println("\nkey/value pairs: " + map.entrySet());

// ===== REMOVE OPERATIONS =====
System.out.print("\nEnter a key to remove from first map: ");
int keyToRemove = sc.nextInt();
map.remove(keyToRemove);
System.out.println("Updated list of elements: " + map);

System.out.print("\nEnter another key to remove from first map: ");
int anotherKey = sc.nextInt();
map.remove(anotherKey);
System.out.println("Updated list of elements: " + map);

sc.nextLine(); // consume newline
System.out.print("\nEnter a key and value to remove (key=value must match): ");
int keyCheck = sc.nextInt();
sc.nextLine();
String valCheck = sc.nextLine();
map.remove(keyCheck, valCheck);

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

sc.close();
}
```

**OUTPUT:**

```
C:\MCA\roll114\java\PRACTICAL 5>java Program4
Enter number of entries for first HashMap: 3
Enter key (integer): 1
Enter value (city name): chennai
Enter key (integer): 2
Enter value (city name): Mumbai
Enter key (integer): 3
Enter value (city name): Indore
Enter number of entries for second HashMap: 3
Enter key (integer): 5
Enter value (city name): Mathura
Enter key (integer): 6
Enter value (city name): Tehri
Enter key (integer): 7
Enter value (city name): Dehli

HashMap 2 contains: {1=chennai, 2=Mumbai, 3=Indore, 5=Mathura, 6=Tehri, 7=Dehli}
Initial list of elements in first map: {1=chennai, 2=Mumbai, 3=Indore}

Enter key to get value from first map: 2
Value at key 2: Mumbai

key/value pairs: [1=chennai, 2=Mumbai, 3=Indore]

Enter a key to remove from first map: 3
Updated list of elements: {1=chennai, 2=Mumbai}

Enter another key to remove from first map: 1
Updated list of elements: {2=Mumbai}

Enter a key and value to remove (key=value must match): 2
chennai
Updated list of elements: {2=Mumbai}
```

**AIM 2:Using TreeMap to store and display entries in a sorted order**

**TreeMap** in Java is an implementation of the **Map interface** that stores **key-value pairs** in a **sorted order** based on the **natural ordering of keys** (or using a custom comparator). It automatically sorts the entries by keys, unlike **HashMap** which does not guarantee any order.

**TreeMap ensures:**

- Keys are **unique**
- Entries are **sorted** in ascending order by default
- Provides methods to traverse, search, and manipulate entries efficiently

**CODE:**

```
→ import java.util.Map;
import java.util.Scanner;
import java.util.TreeMap;

public class TreeMapExample {
```

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

    // Create a TreeMap (keys will be sorted automatically)
    Map<String, Integer> treeMap = new TreeMap<>();

    // Ask the user how many entries to add
    System.out.print("Enter number of entries: ");
    int n = sc.nextInt();
    sc.nextLine(); // consume newline

    // Take user input for key-value pairs
    for (int i = 0; i < n; i++) {
        System.out.print("Enter city name (key): ");
        String city = sc.nextLine();
        System.out.print("Enter value (integer): ");
        int value = sc.nextInt();
        sc.nextLine(); // consume newline
        treeMap.put(city, value);
    }

    // Display entries sorted by keys
    System.out.println("\nTreeMap entries (sorted by key):");
    for (Map.Entry<String, Integer> entry : treeMap.entrySet()) {
        System.out.println(entry.getKey() + ": " + entry.getValue());
    }

    // Example: Get a value by key
    System.out.print("\nEnter a city name to find its value: ");
    String searchKey = sc.nextLine();

    if (treeMap.containsKey(searchKey)) {
        System.out.println("Value for " + searchKey + ": " + treeMap.get(searchKey));
    } else {
        System.out.println("City not found in TreeMap.");
    }

    sc.close();
}
```

**OUTPUT:**

```
C:\MCA\roll114\java\PRACTICAL 5>java TreeMapExample
Enter number of entries: 4
Enter city name (key): Madras
Enter value (integer): 4
Enter city name (key): Pune
Enter value (integer): 9
Enter city name (key): Panaji
Enter value (integer): 2
Enter city name (key): Patna
Enter value (integer):
3

TreeMap entries (sorted by key):
Madras: 4
Panaji: 2
Patna: 3
Pune: 9

Enter a city name to find its value: dehli
City not found in TreeMap.
```

**PRACTICAL 5****Assignments on Lambda Expression**

A **Lambda Expression** in Java is a **short block of code that takes parameters and returns a value**. It provides a clear and concise way to represent **anonymous methods** (methods without a name) and is primarily used to implement **functional interfaces** (interfaces with a single abstract method) in a simpler way.

Lambda expressions were introduced in **Java 8** to support **functional programming** and to make code more readable and concise, especially when working with collections, streams, or event-handling code.

**AIM 1:WAP using Lambda Expression to print “Hello World”.****CODE:**

```
→ interface Message {
    void print();
}

class Lamda {
    public static void main(String[] args) {
        // Implementing the interface using a Lambda Expression
        Message msg = () -> System.out.println("Hello World");

        // Calling the method
        msg.print();
    }
}
```

**OUTPUT:**

```
C:\Users\sujit\.jdks\openjdk-25\bin\java.exe
Hello World

Process finished with exit code 0
```

**AIM 2:Write a Java program using Lambda Expression to concatenate two strings.****CODE:**

```
→interface Concatenate {
    String concat(String s1, String s2);
}

public class LambdaConcat {
```

```
public static void main(String[] args) {
    Concatenate c = (str1, str2) -> str1 + str2;
    String result = c.concat("Hello ", "World!");
    System.out.println("Concatenated String: " + result);
}
```

**OUTPUT:**

```
C:\Users\sujit\.jdks\openjdk-25\bin\java.exe
Concatenated String: Hello World!
```

**AIM 3: WAP using Lambda Expression with single parameters.****CODE:**

```
→@FunctionalInterface
interface Square {
    int calculate(int x);
}
public class SingleParamLambda {
    public static void main(String[] args) {
        Square sq = num -> num * num;
        int result = sq.calculate(5);
        System.out.println("Square: " + result);
    }
}
```

**OUTPUT:**

```
C:\Users\sujit\.jdks\openjdk-25\bin\java.exe
Square: 25
```

**AIM 4: Write a Java program using Lambda Expression with multiple parameters to add two numbers.****CODE:**

```
→@FunctionalInterface
interface AddNumbers {
    int add(int a, int b);
}
```

```
public class MultiParamLambda {
    public static void main(String[] args) {
        AddNumbers add = (x, y) -> x + y;
        int result = add.add(10, 20);
        System.out.println("Sum: " + result); }}
```

**OUTPUT:**

```
C:\Users\sujit\.jdks\openjdk-25\bin\java.exe
Sum: 30
```

**AIM 5:** Write a Java program using Lambda Expression to calculate the following:

- a.Convert Fahrenheit to Celsius
- b.Convert Kilometers to Miles.

**CODE:**

```
→@FunctionalInterface
interface TemperatureConverter {
    double convert(double f);
}

@interface FunctionalInterface
interface DistanceConverter {
    double convert(double km);
}
public class LambdaConversion {
    public static void main(String[] args) {
        TemperatureConverter fahrenheitToCelsius =
            f -> (f - 32) * 5.0 / 9.0;
        DistanceConverter kilometerToMiles =
            km -> km * 0.621371;
        double f = 98.6;
        double km = 10;
        System.out.println("Fahrenheit to Celsius: " +
            fahrenheitToCelsius.convert(f));
        System.out.println("Kilometers to Miles: " +
            kilometerToMiles.convert(km));
    }
}
```

**OUTPUT:**

```
C:\Users\sujit\.jdks\openjdk-25\bin\java.exe
Fahrenheit to Celsius: 37.0
Kilometers to Miles: 6.21371
```

**AIM 6:** Write a Java program using Lambda Expression with or without return keyword.

**CODE:**

Lambda Expression without return keyword.

```
→@FunctionalInterface
interface Square {
    int calculate(int x);
}
public class LambdaWithoutReturn {
    public static void main(String[] args) {
        Square sq = n -> n * n;
        System.out.println("Square: " + sq.calculate(5));
    }
}
```

Lambda Expression with return keyword.

```
→@FunctionalInterface
interface Square {
    int calculate(int x);
}
public class LambdaWithReturn {
    public static void main(String[] args) {
        Square sq = n -> {
            return n * n;
        };
        System.out.println("Square: " + sq.calculate(6));
    }
}
```

**OUTPUT:**

```
C:\Users\sujit\.jdks\openjdk-25\bin\java.exe
Square: 25
```

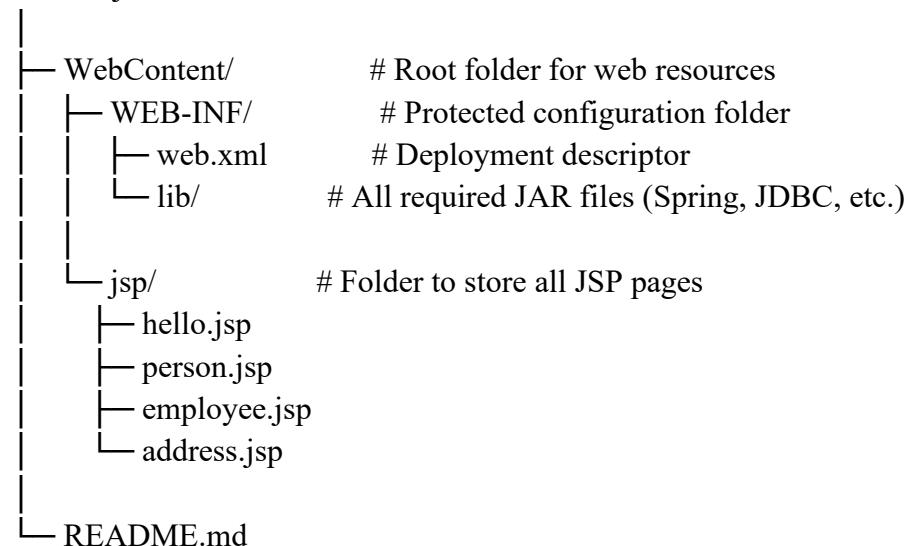
## Practical 6

### Assignments based on web application development using JSP

1. **JSP (JavaServer Pages)** is a server-side technology used to create **dynamic web pages** using Java.
2. It allows embedding **Java Code in HTML** to interact with backend components like Spring beans.
3. JSP works with **Spring MVC** to display data from **controllers and models** to the user.
4. It separates **presentation (JSP) from business logic (Spring beans/Java classes)** for maintainable web applications.

### JSP File Structure

WebProject/



### Explanation:

1. **WebContent/** → Root folder for all web resources (HTML, CSS, JS, JSP).
2. **WEB-INF/** → Contains files that are **not directly accessible** via browser:
  - o **web.xml** → Configures servlets and Spring Dispatcher
  - o **lib/** → All required JAR files (Spring Core, JDBC, etc.)
3. **jsp/** → Folder for all JSP pages, e.g., hello.jsp, person.jsp, etc.

JSP pages should **never go directly under WEB-INF** if you want to access them via browser, but placing them under a subfolder like **jsp/** helps **organize the project**.

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

**Code: -**

```
<!DOCTYPE html>
```

```
<!--
```

Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

Click nbfs://nbhost/SystemFileSystem/Templates/JSP\_Servlet/Html.html to edit this template

```
-->
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

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

```
    <title>Registration Form</title>
```

```
    <style>
```

```
        body {
```

```
            font-family: Arial, sans-serif;
```

```
        }
```

```
        .form-container {
```

```
            width: 400px;
```

```
            margin: 50px auto;
```

```
            padding: 20px;
```

```
            border: 1px solid #ccc;
```

```
            border-radius: 10px;
```

```
            background-color: #f9f9f9;
```

```
        }
```

```
        .form-container h2 {
```

```
            text-align: center;
```

```
            margin-bottom: 20px;
```

```
        }
```

```
        .form-container label {
```

```
            display: block;
```

```
            margin-bottom: 8px;
```

```
        }
```

```
        .form-container input,
```

```
        .form-container select,
```

```
        .form-container textarea {
```

```
            width: 100%;
```

```
            padding: 8px;
```

```
            margin-bottom: 12px;
```

```
border: 1px solid #ccc;
border-radius: 5px;
}

.form-container input[type="submit"] {
    width: auto;
    background-color: #4CAF50;
    color: white;
    border: none;
    cursor: pointer;
    padding: 10px 20px;
}
.form-container input[type="submit"]:hover {
    background-color: #45a049;
}

```

</style>

</head>

<body>

<div class="form-container">

<h2>Registration Form</h2>

<form action="process\_registration.jsp" method="post">

<label for="name">Full Name:</label>

<input type="text" id="name" name="name" placeholder="Enter your full name" required>

<label for="email">Email Address:</label>

<input type="email" id="email" name="email" placeholder="Enter your email address" required>

<label for="password">Password:</label>

<input type="password" id="password" name="password" placeholder="Enter your password" required>

<label for="gender">Gender:</label>

<select id="gender" name="gender" required>

<option value="">Select Gender</option>

<option value="male">Male</option>

<option value="female">Female</option>

<option value="other">Other</option>

</select>

<label for="interests">Interests:</label>

<textarea id="interests" name="interests" placeholder="Enter your interests" rows="4"></textarea>

```
<input type="submit" value="Register">
</form>

</div>

</body>

</html>
```

**Outputs: -**

## Registration Form

**Full Name:**

**Email Address:**

**Password:**

**Gender:**

Select Gender
 ▼

**Interests:**

Enter your interests
 ✖

**Register**

**AIM 2:** - 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% b) 8 to 15 year at 5.5% c) 16 to 30 year at 5.75%

**Code:** -

### Index.jsp

```
<%--  
 Document : index  
 Created on : 9 Dec 2025, 8:46:39 pm  
 Author : sujit  
--%>
```

```
<%@page contentType="text/html"%>  
<%@page pageEncoding="UTF-8"%>  
<%@page errorPage="error.jsp" %>  
  
<html>  
  <head>  
    <title>Loan Application</title>  
  </head>  
  <%@include file="header.html"%>  
  <body>  
    <center>  
      <form name="form1" method="get" action="controller.jsp">  
        <table>  
          <tr>  
            <td>  
              Amount:  
            </td>  
            <td><input type="text" name="amt" id="amt">  
            </td>  
          </tr>  
          <tr>  
            <td>  
              Period:  
            </td>  
            <td>  
              <input type="text" name="n" id="n">  
            </td>  
          </tr>  
          <tr>
```

```

<td>
    Interest Rate:
</td>
<td>
    <input type="text" name="r" id="r">
</td>
</tr>
<tr>
<td>
    <input type="radio" name="type" value="S">Simple
</td>
<td>
    <input type="radio" name="type" value="C">Compound
</td>
</tr>
<td colspan="2" align="center">
<br>
    <input type="submit" name="submit" value="Calculate">
</td>
</tr>
</table>
</form>
</center>
</body>
<jsp:include page="footer.jsp"/>
</html>

```

### Controller.jsp

```

<%-->
Document : controller
Created on : 9 Dec 2025, 8:43:37 pm
Author   : sujit
--%>
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<html>
    <head>

    </head>

    <%@page errorPage="error.jsp" %>

```

```
<%! String s;%>

<body>
<%
s=request.getParameter("type");

if(s.equals("S"))
{
%>
<jsp:forward page="simple.jsp"/>
<%
}
else
{
%
%>
<jsp:forward page="compound.jsp"/>
<%
}
%
%>
</body>
</html>
```

**Error.jsp**

```
<%--  
 Document : error  
 Created on : 9 Dec 2025, 8:44:14 pm  
 Author : sujit  
--%>
```

```
<%@page contentType="text/html" import="java.io.*"%>
<%@page isErrorPage="true"%>
<%@page pageEncoding="UTF-8"%>
<%@include file="header.html"%>

<html>
<head>
<title>Loan Application Error Page</title>
</head>
<body>
```

```
<h1>ERROR</h1>
</body>
</html>
<%@page isErrorPage="true"%>
<jsp:include page="footer.jsp"/>
```

### Footer.jsp

```
<%-- Document : footer
   Created on : 9 Dec 2025, 8:44:46 pm
   Author    : sujit
--%>

<%@page contentType="text/html" import="java.util.*"%>
<%@page pageEncoding="UTF-8"%>

<html>
  <head>
    </head>
  <body>
    <hr>
    <%=new Date()%>
  </body>
</html>
```

### Compound.jsp

```
<%-- Document : compound
   Created on : 9 Dec 2025, 8:41:33 pm
   Author    : sujit
--%>

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<html>
```

```

<head>
</head>

<%!
public double cal(int a, int n,int r) {
    return (double)(a*(1+n)*r)/100;
}
%>
<%@include file="header.html"%>

<body>

<%
double ans =
cal(Integer.parseInt(request.getParameter("amt")),Integer.parseInt(request.getParameter("n")),Integer.parseInt(
request.getParameter("r")));
%>

Compound Interest = <%=ans%>

</body>

<jsp:include page="footer.jsp"/>

</html>

```

**Simple.jsp**

```

<%-- 
Document : simple
Created on : 9 Dec 2025, 8:43:08 pm
Author   : sujit
--%>

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<html>
<head>
</head>

<%!
public double cal(int a, int n,int r) {
    return (double)(a*n*r)/100;
}
%>

```

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

<body>

<%
double ans = cal(Integer.parseInt(request.getParameter("amt")),
    Integer.parseInt(request.getParameter("n")),
    Integer.parseInt(request.getParameter("r")));
%>

Simple Interest = <%=ans%>

</body>

<jsp:include page="footer.jsp"/>

</html>
```

**Outputs:** -

2. 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%  
b) 8 to 15 year at 5.5%  
c) 16 to 30 year at 5.75%

Amount:	<input type="text" value="1253"/>
Period:	<input type="text" value="5"/>
Interest Rate:	<input type="text" value="12"/>
<input checked="" type="radio"/> Simple <input type="radio"/> Compound	
<input type="button" value="Calculate"/>	

Wed Dec 17 21:25:41 IST 2025

2. 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%  
b) 8 to 15 year at 5.5%  
c) 16 to 30 year at 5.75%

Simple Interest = 751.8

Wed Dec 17 21:26:38 IST 2025

Loan Application

localhost:8080/Loan\_Calculator1/

2. 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%  
b) 8 to 15 year at 5.5%  
c) 16 to 30 year at 5.75%

Amount:	1253
Period:	5
Interest Rate:	12

Simple     Compound

Calculate

Wed Dec 17 21:25:41 IST 2025

Calculator

localhost:8080/Loan\_Calculator1/controller.jsp?amt=1253&... Chat

2. 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%  
b) 8 to 15 year at 5.5%  
c) 16 to 30 year at 5.75%

Compound Interest = 902.16

Wed Dec 17 21:26:01 IST 2025

**Aim:** - Write a JSP program that demonstrates the use of JSP declaration, scriptlet, directives, expression, header and footer.

**Code:** -

### Index.jsp

```
<%--  
 Document : Index  
 Created on : 9 Dec 2025, 9:14:47 pm  
 Author : sujit  
--%>  
  
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  
<%@ include file="header.jsp" %>  
  
<!DOCTYPE html>  
  
<html>  
  
<head>  
  
 <meta charset="UTF-8">  
  
 <title>JSP Example with Declarations, Scriptlets, and Expressions</title>  
  
</head>  
  
<body>  
 <h1>JSP Demonstration</h1>  
  
 <%-- JSP Declaration --%>  
 <%!  
 // Declaration: Declares class-level variables and methods  
 int counter = 0;  
 public String greetUser(String name) {  
     return "Hello, " + name + "!";  
 }  
%>  
  
 <%  
 counter++;  
 String user = "Surjit Kargal!";  
%>  
  
 <p>  
 <%-- JSP Expression --%>
```

```

Greeting message: <%= greetUser(user) %>
</p>
<p>
    Page has been accessed <%= counter %> times.
</p>
</body>
</html>

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

```

**Header.jsp**

```

<%--
    Document : header
    Created on : 9 Dec 2025, 9:16:05 pm
    Author   : sujit
--%>

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

<html>
<head>
    <meta charset="UTF-8">
    <title>Header</title>
</head>
<body>

<div style="background-color: lightgray; padding: 10px;">
    <h2>Welcome to the JSP Example Site!</h2>

</div>
</body>
</html>

```

**Footer.jsp**

```

<%--
    Document : footer
    Created on : 9 Dec 2025, 9:15:38 pm
    Author   : sujit
--%>

```

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

<!DOCTYPE html>
<html>
<head>

<meta charset="UTF-8">
<title>Footer</title>

</head>
<body>

<div style="background-color: lightgray; padding: 10px; text-align: center; margin-top: 20px;">
<p>&copy; 2025 JSP Example Site</p>

</div>
</body>
</html>
```

**Outputs: -**

## JSP Demonstration

Greeting message: Hello, Surjit Kargal!!

Page has been accessed 1 times.

© 2025 JSP Example Site

**Aim 4: - Database Application****Documentation: Adding MySQL JDBC Driver to Tomcat****Purpose**

To allow JSP applications to connect to a MySQL database, the **MySQL JDBC driver (Connector/J)** must be available to the Tomcat server. This document explains how to add the driver to Tomcat on a Windows system.

---

**Steps****Step 1: Download MySQL Connector/J**

1. Go to: <https://dev.mysql.com/downloads/connector/j/>
2. Download the **Platform Independent ZIP** version.
3. Extract the ZIP to a folder.
4. Locate the JAR file inside, for example:

**mysql-connector-j-8.1.0.jar**

---

**Step 2: Copy JAR to Tomcat Library**

1. Open your Tomcat installation directory. Example:  
**C:\apache-tomcat-10.1.50**
  2. Navigate to the lib folder:  
**C:\apache-tomcat-10.1.50\lib**
  3. Copy the extracted JAR file (mysql-connector-j-8.1.0.jar) into this folder.
- 

**Step 3: Restart Tomcat**

1. Stop Tomcat if it is running.
2. Start Tomcat again using startup.bat (Windows) or from your service manager.
3. Tomcat now **recognizes the MySQL driver**, and JSP pages can use it to connect to MySQL databases.

**Code: -****Index.jsp**

```
<%--  
 Document : index.jsp  
 Author  : sujit  
--%>  
<%@ page language="java" contentType="text/html; charset=UTF-8" %>  
<%@ page import="java.sql.*" %>  
<!DOCTYPE html>  
<html>  
<head>  
    <title>Student Database Application</title>  
    <style>  
        body {  
            font-family: Arial, sans-serif;  
            background-color: #f2f2f2;  
        }  
        h2 {  
            color: #2c3e50;  
        }  
        form {  
            background-color: white;  
            width: 400px;  
            padding: 20px;  
            border-radius: 5px;  
            box-shadow: 0 0 10px gray;  
        }  
        input[type=text],  
        input[type=email] {  
            width: 100%;  
            padding: 8px;  
            box-sizing: border-box; /* Added for better alignment */  
        }  
        input[type=submit] {  
            background-color: #27ae60;  
            color: white;  
            padding: 8px 15px;  
            border: none;  
            cursor: pointer;  
        }  
    </style>  
</head>  
<body>  
    <h2>Student Database Application</h2>  
    <form>  
        <input type="text" placeholder="Enter Student ID" name="id" />  
        <input type="text" placeholder="Enter Student Name" name="name" />  
        <input type="email" placeholder="Enter Email Address" name="email" />  
        <input type="password" placeholder="Enter Password" name="password" />  
        <input type="submit" value="Login" />  
    </form>  
</body>  
</html>
```

```

input[type=submit]:hover {
    background-color: #219150;
}
table {
    border-collapse: collapse;
    margin-top: 20px;
    background-color: white;
    width: 80%;
}
table th {
    background-color: #2980b9;
    color: white;
    padding: 8px;
}
table td {
    padding: 8px;
    text-align: center;
}
hr {
    margin: 30px 0;
}
</style>

</head>
<body>
<center>

<h2>Student Registration</h2>

<form method="post">

    Name: <input type="text" name="name" required><br><br>
    Email: <input type="email" name="email" required><br><br>
    Course: <input type="text" name="course" required><br><br>
    <input type="submit" name="submit" value="Save">
</form>

<hr>

<%
// Retrieve form data

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

```

```

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

/* MySQL connection details */

String url = "jdbc:mysql://localhost:3306/college";
String user = "root";
String password = "root";
Connection con = null;
PreparedStatement ps = null;
Statement st = null;
ResultSet rs = null;

try {

    Class.forName("com.mysql.cj.jdbc.Driver");
    con = DriverManager.getConnection(url, user, password);

    // Insert record if form is submitted

    if (submit != null) {
        ps = con.prepareStatement("INSERT INTO student(name, email, course) VALUES (?, ?, ?)");
        ps.setString(1, name);
        ps.setString(2, email);
        ps.setString(3, course);
        ps.executeUpdate();
        out.println("<p style='color:green'>Record Saved Successfully!</p>");
    }

    // Fetch all records to display in the table

    st = con.createStatement();
    rs = st.executeQuery("SELECT * FROM student");

    %>

<h2>Student List</h2>
<table border='1'>
    <tr>
        <th>ID</th>

```

```

<th>Name</th>
<th>Email</th>
<th>Course</th>
</tr>

<%
while (rs.next()) {
%>

<tr>
<td><%= rs.getInt("id") %></td>
<td><%= rs.getString("name") %></td>
<td><%= rs.getString("email") %></td>
<td><%= rs.getString("course") %></td>
</tr>
<%
}

} catch (Exception e) {
    out.println("<p style='color:red'>Error: " + e.getMessage() + "</p>");
} finally {
    if (rs != null) rs.close();
    if (st != null) st.close();
    if (ps != null) ps.close();
    if (con != null) con.close();
}
%>

</table>
</center>
</body>
</html>

```

**Db.Sql :-**

```

CREATE DATABASE college;

use college;

CREATE TABLE student (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50),
    email VARCHAR(50),
    course VARCHAR(30)
);

```

**Outputs: -**

	id	name	email	course
▶	1	sujit	sujit@gmail.com	MCA
	2	Lovekesh	lovekesh@gmail.com	MCA
	3	Shalini	shalini@gmail.com	MCA
	4	Anirudh	anirudh@gmail.com	MCA
	5	Sumrit	sumrit@gmail.com	MCA
	6	Mayuresh	mayuresh@gmail.com	MCA
	NULL	NULL	NULL	NULL

## Student Registration

Name:

Email:

Course:

**Save**

Record Saved Successfully!

## Student List

ID	Name	Email	Course
1	sujit	sujit@gmail.com	MCA
2	Lovekesh	lovekesh@gmail.com	MCA
3	Shalini	shalini@gmail.com	MCA
4	Anirudh	anirudh@gmail.com	MCA
5	Sumrit	sumrit@gmail.com	MCA
6	Mayuresh	mayuresh@gmail.com	MCA

## Practical 7

### Assignment based Spring Framework

#### Spring Framework

1. **Spring Framework** is a **lightweight Java framework** that simplifies enterprise application development by providing features like **Dependency Injection (DI)** and **Inversion of Control (IoC)**.
2. **Dependency Injection (DI)** allows objects to receive their dependencies from the Spring container instead of creating them manually, which reduces coupling and improves maintainability.
3. Spring supports **constructor-based, setter-based, and autowiring-based DI**, enabling flexible configuration of beans.
4. Spring beans are defined either in **XML configuration files** (like beans.xml) or using **annotations** (@Component, @Autowired, etc.).
5. The framework promotes **modular, reusable, and testable Code**, separating **business logic** from configuration.

#### Spring Framework Project File Structure

```

SpringProject/
  └── src/
    └── main/
      └── java/          # Java source files (beans, services)
        └── com/example/
          ├── HelloWorld.java
          ├── Person.java
          ├── Employee.java
          └── Address.java

  └── src/
    └── main/resources/
      └── beans.xml      # Spring configuration (beans, DI setup)

  └── lib/            # Required JARs (spring-core, spring-beans, etc.)

  └── README.md
  └── pom.xml         # Maven dependencies (if using Maven)

```

**Explanation of Folders:**

- src/main/java/ → Contains all **Java classes** (beans, services).
- src/main/resources/ → Contains **Spring configuration** (beans.xml or annotation-based config).
- lib/ → Contains **Spring framework JARs** if not using Maven.
- pom.xml → Maven file listing **Spring dependencies** (optional).

**Aim 1:** - Write a program to print “Hello World” using spring framework.

**Code:** -

**HelloWorldApplication.java**

```
package SujitKargal.HelloWorld;

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

public class HelloWorldApplication {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
        HelloWorld hw = (HelloWorld) context.getBean("helloBean");
        hw.sayHello();
    }
}
```

**HelloWorld.java**

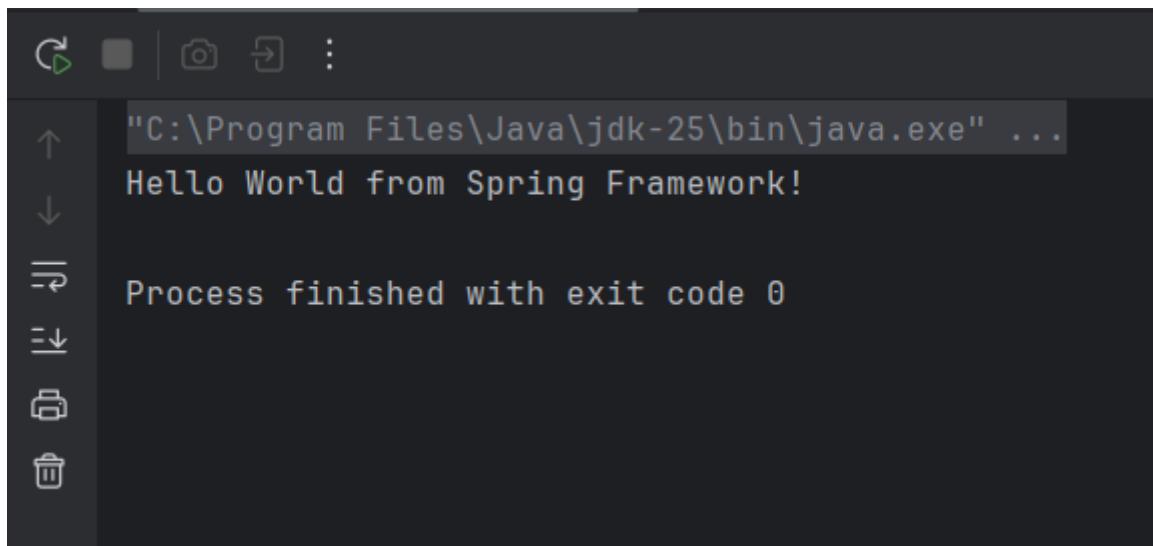
```
package SujitKargal.HelloWorld;

public class HelloWorld {
    public void sayHello() {
        System.out.println("Hello World from Spring Framework!");
    }
}
```

**bean.xml**

```
<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">
```

```
<bean id="helloBean" class="SujitKargal.HelloWorld.HelloWorld"/>
</beans>
```

**Outputs: -**


```
"C:\Program Files\Java\jdk-25\bin\java.exe" ...
Hello World from Spring Framework!
Process finished with exit code 0
```

**Aim 2:** - Write a program to demonstrate dependency injection via setter method.

**Code: -****DependencyInjectionApplication.java**

```
package SujitKargal.Dependency.Injection;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

@SpringBootApplication
public class DependencyInjectionApplication {

    public static void main(String[] args) {
        // Load Spring XML configuration
        ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");

        // Get the bean
        Person person = (Person) context.getBean("personBean");

        // Display data
        person.display();
    }
}
```

**Person.java**

```
package SujitKargal.Dependency.Injection;

public class Person {
    private String name;
    private int age;

    // Setter methods for dependency injection
    public void setName(String name) {
        this.name = name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public void display() {
        System.out.println("Person Name: " + name);
        System.out.println("Person Age: " + age);
    }
}
```

**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">

    <!-- Bean with setter injection -->
    <!-- <bean id="personBean" class="SujitKargal.Person">-->
    <bean id="personBean" class="SujitKargal.Dependency.Injection.Person">
        <property name="name" value="Sujit"/>
        <property name="age" value="23"/>
    </bean>

</beans>
```

**Output: -**

```
"C:\Program Files\Java\jdk-25\bin\java.exe" ...
Person Name: Sujit
Person Age: 23

Process finished with exit code 0
```

**Aim 3: - Write a program to demonstrate dependency injection via Constructor.****Code: -****ConstructorInjectionApplication.java**

```
package SujitKargal.ConstructorInjection;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

@SpringBootApplication
public class ConstructorInjectionApplication {

    public static void main(String[] args) {
        // Load Spring XML configuration
        ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");

        // Get the bean
        Person person = (Person) context.getBean("personBean");

        // Display person info
        person.display();
    }
}
```

**Person.java**

```
package SujitKargal.ConstructorInjection;
```

```

public class Person {
    private String name;
    private int age;

    // Constructor for dependency injection
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void display() {
        System.out.println("Person Name: " + name);
        System.out.println("Person Age: " + age);
    }
}

```

**beans.java**

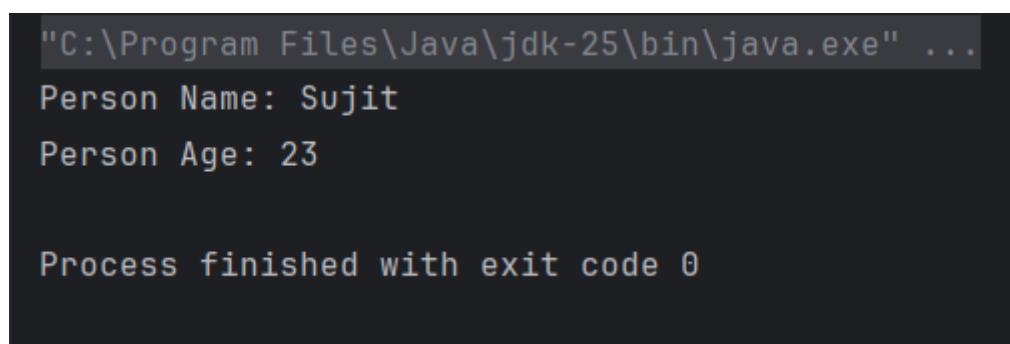
```

<?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">

    <!-- Bean with constructor injection -->
    <bean id="personBean" class="SujitKargal.ConstructorInjection.Person">
        <constructor-arg name="name" value="Sujit"/>
        <constructor-arg name="age" value="23"/>
    </bean>

</beans>

```

**Outputs: -**


```

"C:\Program Files\Java\jdk-25\bin\java.exe" ...
Person Name: Sujit
Person Age: 23

Process finished with exit code 0

```

**Aim 4: - Write a program to demonstrate Autowiring.****AutowiringApplication.java**

```
package SujitKargal.Autowiring;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

@SpringBootApplication
public class AutowiringApplication {

    public static void main(String[] args) {
        // Load Spring XML configuration
        ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");

        // Get the bean
        Person person = (Person) context.getBean("personBean");

        // Display data
        person.display();
    }
}
```

**Address.java**

```
package SujitKargal.Autowiring;

public class Address {
    private String city;
    private String state;

    // Default constructor
    public Address() {
    }

    // Setters for Spring dependency injection
    public void setCity(String city) {
        this.city = city;
    }

    public void setState(String state) {
        this.state = state;
    }
}
```

```

    }

    public void display() {
        System.out.println("City: " + city);
        System.out.println("State: " + state);
    }

}

```

**Person.java**

```

package SujitKargal.Autowiring;

public class Person {
    private String name;
    private int age;
    private Address address; // Dependency

    public Person() {}

    // Setters
    public void setName(String name) {
        this.name = name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public void setAddress(Address address) {
        this.address = address;
    }

    public void display() {
        System.out.println("Person Name: " + name);
        System.out.println("Person Age: " + age);
        System.out.println("Address Details:");
        address.display();
    }
}

```

**beans: -**

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"

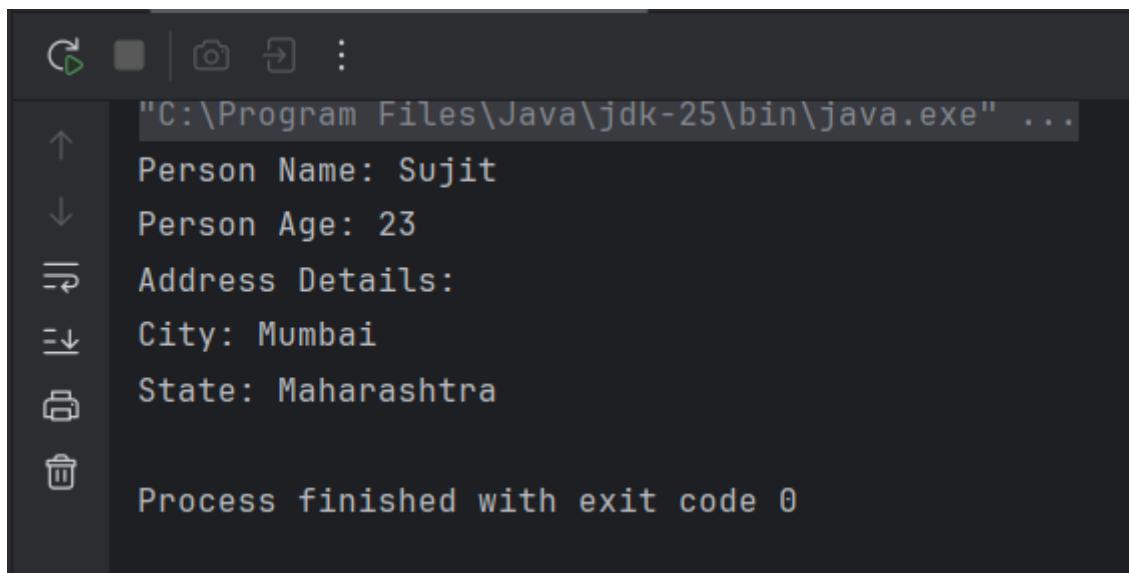
```

```
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">

<!-- Address bean -->
<bean id="addressBean" class="SujitKargal.Autowiring.Address">
    <property name="city" value="Mumbai"/>
    <property name="state" value="Maharashtra"/>
</bean>

<!-- Person bean with autowiring by type -->
<bean id="personBean" class="SujitKargal.Autowiring.Person" autowire="byType">
    <property name="name" value="Sujit"/>
    <property name="age" value="23"/>
</bean>

</beans>
```

**Outputs: -**

```
"C:\Program Files\Java\jdk-25\bin\java.exe" ...
Person Name: Sujit
Person Age: 23
Address Details:
City: Mumbai
State: Maharashtra

Process finished with exit code 0
```

## Practical 8

### Assignment based Aspect Oriented Programming

#### What is Spring AOP?

**Spring AOP** is a programming technique that allows you to "inject" extra behavior (like logging or security) into your existing code without modifying the actual methods. It works by creating a proxy around your service.

#### The 5 Types of Advice in your Code

Your LoggingAspect.java uses all the primary AOP advice types:

1. **@Before**: Runs **before** the method starts. Useful for logging inputs or security checks.
2. **@After**: Runs **after** the method finishes, regardless of whether it succeeded or threw an error (like a finally block).
3. **@Around**: The most powerful advice. It wraps the method and can control whether the method even runs using pjp.proceed().
4. **@AfterReturning**: Runs only if the method **finishes successfully**. It allows you to see the returned value.
5. **@AfterThrowing**: Runs only if the method **throws an exception**. It is used for error logging or handling.

#### Key AOP Terms

- **Aspect**: The class (LoggingAspect) that contains the extra logic.
- **Pointcut**: The expression (execution(\* com.example.aop.DemoService.\*(..))) that defines **where** the advice should be applied.
- **Join Point**: The specific point during program execution (like calling the success() method) where the aspect is plugged in.

#### Summary Table

Advice	When it runs?
<b>Before</b>	Before method execution.
<b>After</b>	After method execution (always).
<b>Around</b>	Before and after execution.
<b>AfterReturning</b>	Only after successful completion.
<b>AfterThrowing</b>	Only if an exception occurs.

**1. Write a program to demonstrate Spring AOP – before advice, After advice, Around advice, after returning advice, after throwing, pointcuts.**

**Code:-**

**LoggingAspect.java:-**

```
package com.example.aop;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.stereotype.Component;

@Aspect
@Component

public class LoggingAspect {

    // pointcut for all methods in DemoService
    @Pointcut("execution(* com.example.aop.DemoService.*(..))")

    public void demoMethods() {}

    @Before("demoMethods()")
    public void beforeAdvice() {
        System.out.println("[AOP] Before advice executed");
    }

    @After("demoMethods()")
    public void afterAdvice() {
        System.out.println("[AOP] After advice executed");
    }

    @Around("demoMethods()")
    public Object aroundAdvice(ProceedingJoinPoint pjp) throws Throwable {
```

```
System.out.println("[AOP] Around - before proceed");

Object result = pjp.proceed();

System.out.println("[AOP] Around - after proceed");

return result;

}

{@AfterReturning(pointcut = "demoMethods()", returning = "ret")}

public void afterReturningAdvice(Object ret) {

    System.out.println("[AOP] After returning advice. Return value: " + ret);

}

{@AfterThrowing(pointcut = "demoMethods()", throwing = "ex")}

public void afterThrowingAdvice(Throwable ex) {

    System.out.println("[AOP] After throwing advice. Exception: " + ex.getMessage());

}

}
```

**DemoService.java: -**

```
package com.example.aop;

import org.springframework.stereotype.Service;

@Service

public class DemoService {

    public String success(String name) {

        System.out.println("DemoService.success executing for " + name);

        return "Hello " + name;

    }

    public void error() {

        System.out.println("DemoService.error will throw");

        throw new RuntimeException("Simulated exception");

    }

}
```

```
}
```

**Application.java: -**

```
package com.example.aop;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class Application {

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

    @Bean
    CommandLineRunner run(DemoService demoService) {
        return args -> {
            System.out.println("--- Calling success method (triggers before) ---");
            demoService.success("Alice");
            System.out.println("\n--- Calling error method (triggers afterThrowing) ---");
            try {
                demoService.error();
            } catch (RuntimeException e) {
                System.out.println("Caught exception in main: " + e.getMessage());
            }
        };
    }
}
```

**Pom.xml:-**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
```

```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                     http://maven.apache.org/xsd/maven-4.0.0.xsd">
<parent>
  <groupId>com.example</groupId>
  <artifactId>spring-teaching-project</artifactId>
  <version>1.0.0</version>
</parent>
<modelVersion>4.0.0</modelVersion>
<artifactId>spring-aop</artifactId>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
  </dependency>
</dependencies>
</project>

```

**OUTPUT: -**

```

--- Calling success method (triggers before) ---
[AOP] Around - before proceed
[AOP] Before advice executed
DemoService.success executing for Alice
[AOP] After returning advice. Return value: Hello Alice
[AOP] After advice executed
[AOP] Around - after proceed

--- Calling error method (triggers afterThrowing) ---
[AOP] Around - before proceed
[AOP] Before advice executed
DemoService.error will throw
[AOP] After throwing advice. Exception: Simulated exception
[AOP] After advice executed
Caught exception in main: Simulated exception

```

## Practical 9

### Assignment based Spring JDBC

**Spring JDBC** is a simplified way to connect Java apps to databases. It removes "boilerplate" code (like manually opening/closing connections or handling errors), letting you focus purely on writing SQL.

#### The 3 Main JdbcTemplate Methods

##### 1. update() (Change Data)

- **Use for:** INSERT, UPDATE, and DELETE.
- **Result:** Returns a number (how many rows were changed).
- **Benefit:** Uses prepared statements to stop SQL injection.

##### 2. queryForObject() (Get One Item)

- **Use for:** SELECT queries where you expect exactly **one** result (e.g., searching by ID).
- **Requirement:** Needs a RowMapper to turn the database row into a Java object.
- **Note:** Fails if zero or multiple items are found.

##### 3. query() (Get Many Items)

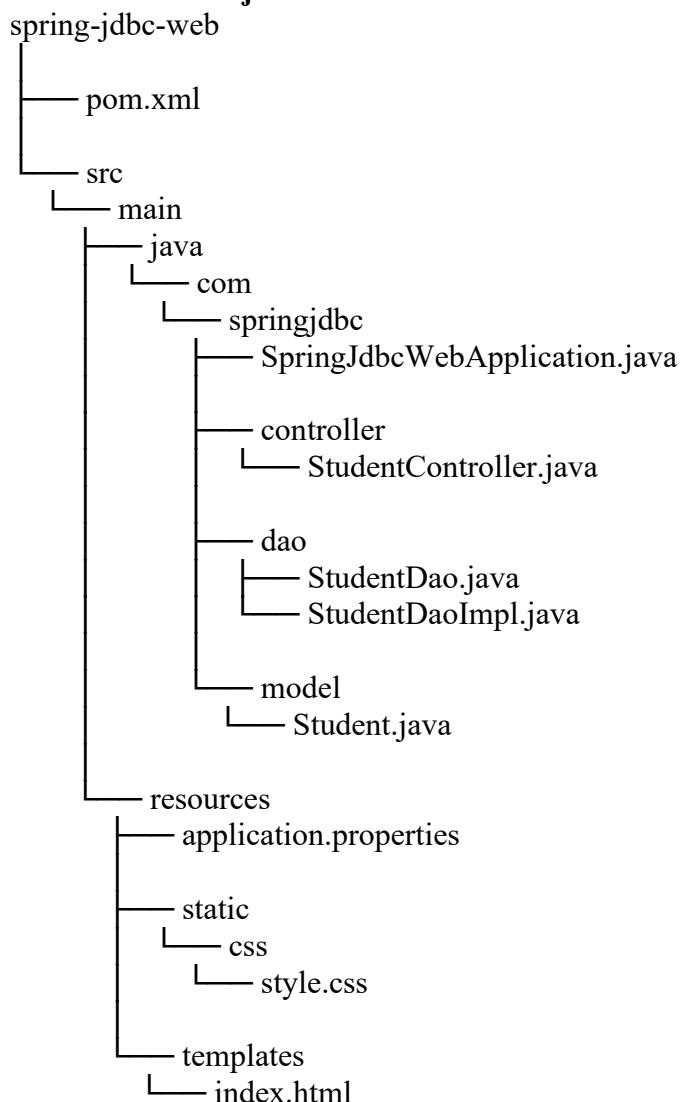
- **Use for:** SELECT queries that return a **list** of data (e.g., getting all students).
- **Result:** Automatically loops through all rows and returns a List of objects.

#### Quick Selection Table

Goal	Method	Output
Modify Data	update()	int (rows affected)
Get 1 Record	queryForObject()	Single Object
Get List	query()	List of Objects

#### Summary of Annotations Used

Annotation	Purpose
@Repository	Tells Spring that StudentDaoImpl is a data access component and enables automatic translation of database exceptions.
@Controller	Marks the class as a web controller that returns "Views" (HTML files) rather than raw data.
@ModelAttribute	(Implicit in your code) Used to bind form data from the HTML page back to the Student object.
@PathVariable	Extracts the id directly from the URL (e.g., /delete/5) to use as a parameter in the method.

**Structure Of Project****Aim: - Assignment based Spring JDBC****SpringjdbcApplication.java**

```
package sujitkargal.springjdbc;
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
public class SpringjdbcApplication {
```

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

**StudentController.java**

```
package sujitzkargal.springjdbc.controller;

import sujitzkargal.springjdbc.dao.StudentDao;
import sujitzkargal.springjdbc.model.Student;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller
public class StudentController {

    private final StudentDao studentDao;

    public StudentController(StudentDao studentDao) {
        this.studentDao = studentDao;
    }

    @GetMapping("/")
    public String home(Model model) {
        model.addAttribute("student", new Student());
        model.addAttribute("students", studentDao.getAll());
        return "index";
    }

    @PostMapping("/save")
    public String save(Student student) {
        studentDao.save(student);
        return "redirect:/";
    }

    @GetMapping("/edit/{id}")
    public String edit(@PathVariable int id, Model model) {
        model.addAttribute("student", studentDao.getById(id));
        model.addAttribute("students", studentDao.getAll());
        return "index";
    }

    @PostMapping("/update")
    public String update(Student student) {
        studentDao.update(student);
        return "redirect:/";
    }

    @GetMapping("/delete/{id}")
    public String delete(@PathVariable int id) {
        studentDao.delete(id);
        return "redirect:/";
    }
}
```

**StudentDao.java**

```
package sujitkargal.springjdbc.dao;

import sujitkargal.springjdbc.model.Student;
import java.util.List;

public interface StudentDao {

    void save(Student student);

    void update(Student student);

    void delete(int id);

    Student getById(int id);

    List<Student> getAll();
}
```

**StudentDaoImpl.java**

```
package sujitkargal.springjdbc.dao;

import sujitkargal.springjdbc.model.Student;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public class StudentDaoImpl implements StudentDao {

    private final JdbcTemplate jdbcTemplate;

    public StudentDaoImpl(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    public void save(Student student) {
        jdbcTemplate.update(
            "INSERT INTO student(name,email,course) VALUES(?, ?, ?)",
            student.getName(), student.getEmail(), student.getCourse()
        );
    }

    public void update(Student student) {
        jdbcTemplate.update(

```

```

        "UPDATE student SET name=?, email=?, course=? WHERE id=?",
        student.getName(), student.getEmail(),
        student.getCourse(), student.getId()
    );
}

public void delete(int id) {
    jdbcTemplate.update("DELETE FROM student WHERE id=?", id);
}

public Student getById(int id) {
    return jdbcTemplate.queryForObject(
        "SELECT * FROM student WHERE id=?",
        (rs, rowNum) -> {
            Student s = new Student();
            s.setId(rs.getInt("id"));
            s.setName(rs.getString("name"));
            s.setEmail(rs.getString("email"));
            s.setCourse(rs.getString("course"));
            return s;
        }, id
    );
}

public List<Student> getAll() {
    return jdbcTemplate.query(
        "SELECT * FROM student",
        (rs, rowNum) -> {
            Student s = new Student();
            s.setId(rs.getInt("id"));
            s.setName(rs.getString("name"));
            s.setEmail(rs.getString("email"));
            s.setCourse(rs.getString("course"));
            return s;
        }
    );
}
}

```

**Student.java**

```

package sujatkargal.springjdbc.model;

public class Student {

    private int id;
    private String name;
    private String email;
    private String course;

```

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

public String getName() { return name; }
public void setName(String name) { this.name = name; }

public String getEmail() { return email; }
public void setEmail(String email) { this.email = email; }

public String getCourse() { return course; }
public void setCourse(String course) { this.course = course; }
}
```

### style.css

```
body {
    font-family: Arial;
    background: #f2f2f2;
}

.container {
    width: 600px;
    margin: auto;
    background: white;
    padding: 20px;
    box-shadow: 0 0 10px gray;
}

h2 {
    text-align: center;
    color: #2c3e50;
}

input {
    width: 100%;
    padding: 8px;
    margin: 6px 0;
}

button {
    width: 100%;
    padding: 10px;
    background: #27ae60;
    color: white;
    border: none;
    cursor: pointer;
}
```

```

button:hover {
    background: #219150;
}

table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
}

table, th, td {
    border: 1px solid #ccc;
}

th {
    background: #2980b9;
    color: white;
}

td {
    text-align: center;
    padding: 8px;
}

a {
    margin: 0 5px;
    color: #2980b9;
    text-decoration: none;
}

```

**index.java**

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Student Management</title>
    <link rel="stylesheet" href="/css/style.css">
</head>
<body>

<div class="container">
    <h2>Student Registration</h2>

    <form th:action="${student.id == 0} ? @{/save} : @{/update}"
          th:object="${student}" method="post">

        <input type="hidden" th:field="*{id}">

        <input type="text" th:field="*{name}" placeholder="Name" required>

```

```

<input type="email" th:field="*{email}" placeholder="Email" required>
<input type="text" th:field="*{course}" placeholder="Course" required>

<button type="submit"
        th:text="${student.id == 0} ? 'Save' : 'Update'"></button>
</form>

<h2>Student List</h2>

<table>
  <tr>
    <th>ID</th><th>Name</th><th>Email</th><th>Course</th><th>Action</th>
  </tr>

  <tr th:each="s : ${students}">
    <td th:text="${s.id}"></td>
    <td th:text="${s.name}"></td>
    <td th:text="${s.email}"></td>
    <td th:text="${s.course}"></td>
    <td>
      <a th:href="@{/edit/{id}(id=${s.id})}">Edit</a>
      <a th:href="@{/delete/{id}(id=${s.id})}"
         onclick="return confirm('Delete?')">Delete</a>
    </td>
  </tr>
</table>
</div>

</body>
</html>

```

## Application.Properties

```

spring.application.name=springjdbc
spring.datasource.url=jdbc:mysql://localhost:3306/college // database path
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

```

```
spring.thymeleaf.cache=false
```

## pom.xml

```

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

```

```
<modelVersion>4.0.0</modelVersion>

<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.5</version>
    <relativePath/>
</parent>

<groupId>sujitkargal</groupId>
<artifactId>springjdbc</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>springjdbc</name>
<description>Spring JDBC Web Application</description>

<properties>
    <java.version>17</java.version>
</properties>

<dependencies>

    <!-- Spring MVC + Embedded Tomcat -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Thymeleaf (HTML, no JSP) -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>

    <!-- Spring JDBC -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jdbc</artifactId>
    </dependency>

    <!-- MySQL Driver -->
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <version>9.5.0</version>
        <scope>runtime</scope>
    </dependency>

    <!-- Testing -->
    <dependency>
```

```

<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>

</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>

</project>

```

**Outputs: -**

The screenshot shows a web application interface. At the top, there is a header bar with icons for refresh, search, and other navigation. Below the header, the URL 'localhost:8080' is visible.

**Student Registration:** This section contains three input fields: 'Name', 'Email', and 'Course'. Below these fields is a green 'Save' button.

**Student List:** This section displays a table with the following data:

ID	Name	Email	Course	Action
1	sujit	sujit@gmail.com	MCA	Edit Delete
2	Lovekesh	lovekesh@gmail.com	MCA	Edit Delete
3	Shalini	shalini@gmail.com	MCA	Edit Delete
4	Anirudh	anirudh@gmail.com	MCA	Edit Delete
5	Sumrit	sumrit@gmail.com	MCA	Edit Delete
6	Mayuresh	mayuresh@gmail.com	MCA	Edit Delete

localhost:8080/edit/1

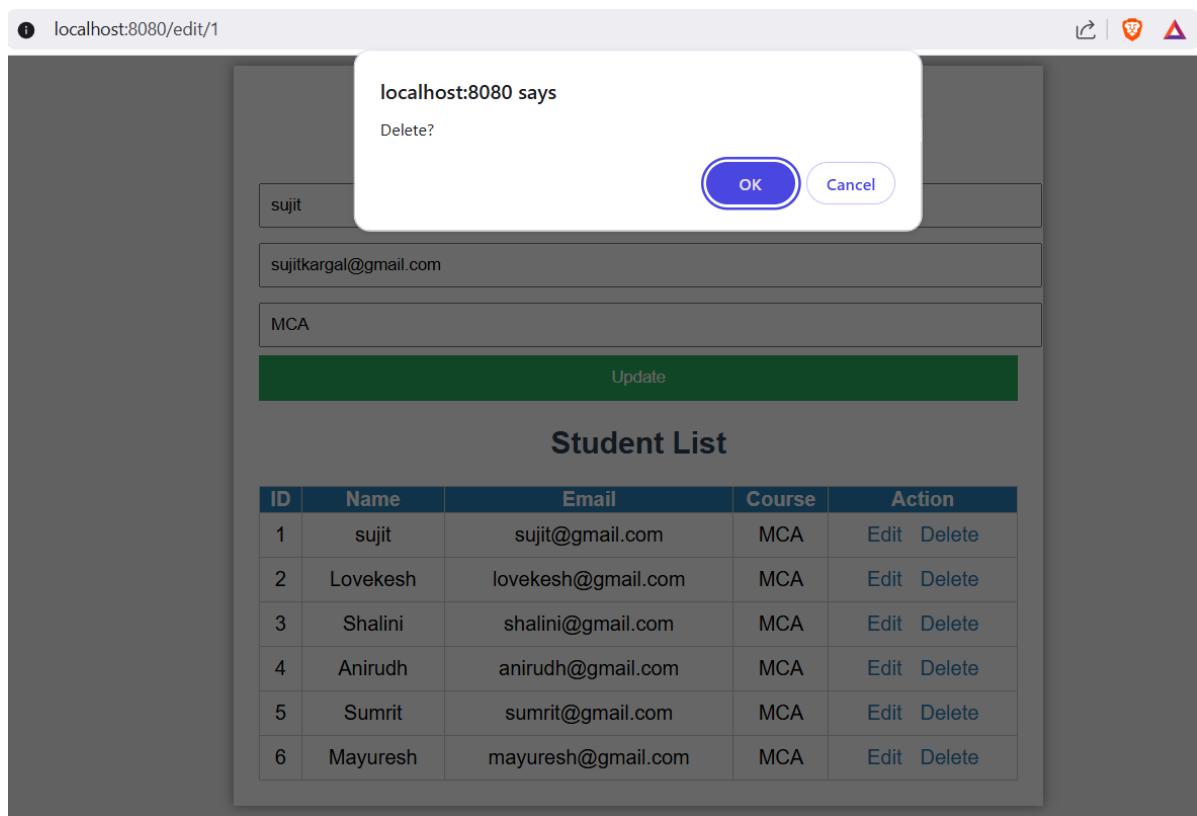
### Student Registration




[Update](#)

### Student List

ID	Name	Email	Course	Action
1	sujit	sujit@gmail.com	MCA	<a href="#">Edit</a> <a href="#">Delete</a>
2	Lovekesh	lovekesh@gmail.com	MCA	<a href="#">Edit</a> <a href="#">Delete</a>
3	Shalini	shalini@gmail.com	MCA	<a href="#">Edit</a> <a href="#">Delete</a>
4	Anirudh	anirudh@gmail.com	MCA	<a href="#">Edit</a> <a href="#">Delete</a>
5	Sumrit	sumrit@gmail.com	MCA	<a href="#">Edit</a> <a href="#">Delete</a>
6	Mayuresh	mayuresh@gmail.com	MCA	<a href="#">Edit</a> <a href="#">Delete</a>



**Practical 10****Assignment based Spring Boot and RESTful Web Services**

- An **API (Application Programming Interface)** is a set of rules that allows two different software applications to communicate and exchange data.
- **@SpringBootApplication** is the main annotation used to launch the application, enabling auto-configuration and component scanning.
- **@RestController** marks a class as a web handler that returns data directly to the user instead of rendering an HTML page.
- **@GetMapping**, **@PostMapping**, and **@Autowired** are used to map web requests to methods and automatically inject dependencies.

**Aim:** - Write a program to create a simple Spring Boot application that prints a message.

**Code:** -

**ApiApplication.java**

```
package prints_a_message.API;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ApiApplication {

    public static void main(String[] args) {
        SpringApplication.run(ApiApplication.class, args);
        System.out.println("Spring Boot Application Started!");
    }
}
```

**HelloController.java**

```
package prints_a_message.API;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {
    @GetMapping("/hello")
    public String hello(){
        return "Hello, Welcome to Spring Boot!";
    }
}
```

**Outputs:** -

