**Java Library Preparation Question**

**with answer**

**Exm of Lambda expression :**

interface Sayable {

String say(String name);

}

public class LambdaExpressionExample {

public static void main(String[] args) {

// Lambda expression with a single parameter

Sayable s1 = (name) -> {

return "Hello, " + name;

};

// Shorter form (omitting return and braces)

Sayable s2 = name -> "Hello, " + name;

System.out.println(s1.say("Alice"));

System.out.println(s2.say("Bob"));

}

}

**#Object class methods :**

public class ObjectMethodsExample {

public static void main(String[] args) {

Object obj1 = new Object();

Object obj2 = new Object();

// 1. equals()

System.out.println(obj1.equals(obj2)); // Usually false for different objects

// 2. hashCode()

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

System.out.println(obj2.hashCode()); // Different hash codes for different objects

// 3. toString()

System.out.println(obj1.toString()); // Default implementation: ClassName@hashCode

// 4. clone()

// Object cloneObj = obj1.clone(); // Requires implementing Cloneable interface

// 5. finalize()

// Called by the garbage collector before object is garbage collected

// Not reliable for resource cleanup

// 6. getClass()

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

}

}

#Wapper class example:

public class WrapperClassExample {

public static void main(String[] args) {

// Creating wrapper objects

Integer intObj = new Integer(10);

Double doubleObj = new Double(3.14);

Boolean boolObj = new Boolean(true);

Character charObj = new Character('A');

// Using wrapper objects

int intValue = intObj.intValue();

double doubleValue = doubleObj.doubleValue();

boolean boolValue = boolObj.booleanValue();

char charValue = charObj.charValue();

System.out.println("Integer value: " + intValue);

System.out.println("Double value: " + doubleValue);

System.out.println("Boolean value: " + boolValue);

System.out.println("Character value: " + charValue);

// Autoboxing and unboxing

int num = 10;

Integer intObj2 = num; // Autoboxing: int to Integer

int num2 = intObj2; // Unboxing: Integer to int

}

}

#Type Casting by using wrapper class:

public class TypeCastingExample {

public static void main(String[] args) {

// Implicit Type Casting (Widening Conversion)

byte byteValue = 10;

int intValue = byteValue; // Implicit conversion from byte to int

short shortValue = 20;

float floatValue = shortValue; // Implicit conversion from short to float

char charValue = 'A';

int intValue2 = charValue; // Implicit conversion from char to int

// Explicit Type Casting (Narrowing Conversion)

double doubleValue = 10.5;

int intValue3 = (int) doubleValue; // Explicit conversion from double to int (loss of precision)

float floatValue2 = 2.5f;

short shortValue2 = (short) floatValue2; // Explicit conversion from float to short (loss of precision)

// Autoboxing and Unboxing

Integer integerObject = 10; // Autoboxing

int intValue4 = integerObject; // Unboxing

System.out.println("Implicit Conversions:");

System.out.println("int: " + intValue);

System.out.println("float: " + floatValue);

System.out.println("int: " + intValue2);

System.out.println("\nExplicit Conversions:");

System.out.println("int: " + intValue3);

System.out.println("short: " + shortValue2);

System.out.println("\nAutoboxing and Unboxing:");

System.out.println("int: " + intValue4);

}

}

**#String to primitive data conversion:**

import java.util.Scanner;

public class StringToPrimitiveConversion {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Using Integer.parseInt()

System.out.print("Enter an integer: ");

String strInt = scanner.nextLine();

int intValue = Integer.parseInt(strInt);

// Using Double.parseDouble()

System.out.print("Enter a double: ");

String strDouble = scanner.nextLine();

double doubleValue = Double.parseDouble(strDouble);

// Using Boolean.parseBoolean()

System.out.print("Enter a boolean: ");

String strBoolean = scanner.nextLine();

boolean booleanValue = Boolean.parseBoolean(strBoolean);

System.out.println("Int value: " + intValue);

System.out.println("Double value: " + doubleValue);

System.out.println("Boolean value: " + booleanValue);

scanner.close();

}

}

public class PrimitiveToStringConversion {

public static void main(String[] args) {

int number = 42;

double pi = 3.14159;

boolean isTrue = true;

char character = 'A';

// Using String.valueOf()

String numberString = String.valueOf(number);

String piString = String.valueOf(pi);

String booleanString = String.valueOf(isTrue);

String characterString = String.valueOf(character);

// Using concatenation with an empty string

String numberString2 = "" + number;

String piString2 = "" + pi;

String booleanString2 = "" + isTrue;

String characterString2 = "" + character;

System.out.println("Using String.valueOf():");

System.out.println("Number: " + numberString);

System.out.println("Pi: " + piString);

System.out.println("Boolean: " + booleanString);

System.out.println("Character: " + characterString);

System.out.println("\nUsing concatenation:");

System.out.println("Number: " + numberString2);

System.out.println("Pi: " + piString2);

System.out.println("Boolean: " + booleanString2);

System.out.println("Character: " + characterString2);

}

}

#Exception handling code:

import java.io.IOException;

public class ExceptionHandlingExample {

public static void main(String[] args) {

try {

// Code that might throw an exception

int result = divide(10, 0);

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

} catch (ArithmeticException e) {

System.out.println("Error: Division by zero");

} catch (IOException e) {

System.out.println("An I/O error occurred.");

} finally {

// Code that always executes, regardless of exceptions

System.out.println("This block always executes.");

}

}

public static int divide(int a, int b) throws ArithmeticException {

if (b == 0) {

throw new ArithmeticException("Division by zero");

}

return a / b;

}

}

#Unchecked Exception Propagation :

public class ExceptionPropagationExample {

public static void divide(int a, int b) throws ArithmeticException {

if (b == 0) {

throw new ArithmeticException("Division by zero");

} else {

int result = a / b;

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

}

}

public static void callerMethod() {

try {

divide(10, 0);

} catch (ArithmeticException e) {

System.out.println("Caught ArithmeticException in callerMethod: " + e.getMessage());

}

}

public static void main(String[] args) {

try {

callerMethod();

} catch (Exception e) {

System.out.println("Caught exception in main method: " + e.getMessage());

}

}

}

#Checked Exception Propagation:

import java.io.IOException;

public class ExceptionPropagationExample {

public static void method3() throws IOException {

throw new IOException("An I/O error occurred.");

}

public static void method2() throws IOException {

method3();

}

public static void main(String[] args) {

try {

method2();

} catch (IOException e) {

System.out.println("Caught IOException in main method: " + e.getMessage());

}

}

}

#Generic collection :

import java.util.ArrayList;

import java.util.List;

public class GenericCollectionsExample {

public static void main(String[] args) {

// Generic ArrayList to store Strings

List<String> names = new ArrayList<>();

names.add("Alice");

names.add("Bob");

names.add("Charlie");

// Iterating over the generic list

for (String name : names) {

System.out.println(name);

}

}

}

#Raw type collection :

import java.util.ArrayList;

public class RawTypeCollectionsExample {

public static void main(String[] args) {

// Raw ArrayList

ArrayList names = new ArrayList();

names.add("Alice");

names.add(10); // Compiles but can lead to runtime errors

// Iterating over the raw list

for (Object name : names) {

System.out.println(name);

}

}

}

#ArrayList Example :

import java.util.ArrayList;

import java.util.List;

public class ArrayListExample {

public static void main(String[] args) {

// Creating an ArrayList

List<String> fruits = new ArrayList<>();

// Adding elements

fruits.add("Apple");

fruits.add("Banana");

fruits.add("Orange");

fruits.add("Mango");

// Accessing elements

String firstFruit = fruits.get(0);

System.out.println("First fruit: " + firstFruit);

// Getting the size

int size = fruits.size();

System.out.println("Number of fruits: " + size);

// Checking if an element exists

boolean containsOrange = fruits.contains("Orange");

System.out.println("Contains Orange: " + containsOrange);

// Removing an element

fruits.remove("Banana");

// Iterating over the ArrayList

for (String fruit : fruits) {

System.out.println(fruit);

}

// Clearing the ArrayList

fruits.clear();

System.out.println("After clearing: " + fruits);

}

}

#LIkedLisat Example :

import java.util.LinkedList;

public class LinkedListExample {

public static void main(String[] args) {

LinkedList<String> linkedList = new LinkedList<>();

// Adding elements

linkedList.add("Apple");

linkedList.add("Banana");

linkedList.add("Orange");

linkedList.add("Grape");

// Accessing elements

String firstElement = linkedList.getFirst();

String lastElement = linkedList.getLast();

System.out.println("First element: " + firstElement);

System.out.println("Last element: " + lastElement);

// Removing elements

linkedList.removeFirst(); // Removes "Apple"

linkedList.removeLast(); // Removes "Grape"

// Iterating over elements

for (String fruit : linkedList) {

System.out.println(fruit);

}

// Checking size

int size = linkedList.size();

System.out.println("Size of the list: " + size);

// Checking if the list is empty

boolean isEmpty = linkedList.isEmpty();

System.out.println("Is the list empty? " + isEmpty);

// Clearing the list

linkedList.clear();

System.out.println("List after clearing: " + linkedList);

}

}

#Vector Example :

import java.util.Vector;

public class VectorExample {

public static void main(String[] args) {

Vector<String> vector = new Vector<>();

// Adding elements

vector.add("apple");

vector.add("banana");

vector.add("orange");

// Accessing elements

String firstElement = vector.get(0);

System.out.println("First element: " + firstElement);

// Removing elements

vector.remove(1); // Removes the element at index 1

// Iterating over elements

for (String fruit : vector) {

System.out.println(fruit);

}

// Checking size

int size = vector.size();

System.out.println("Vector size: " + size);

// Checking if empty

boolean isEmpty = vector.isEmpty();

System.out.println("Is vector empty? " + isEmpty);

}

}

**Basic Operations**

import java.util.Vector;

public class VectorExample {

public static void main(String[] args) {

Vector<String> vector = new Vector<>();

// Adding elements

vector.add("apple");

vector.add("banana");

vector.add("orange");

// Accessing elements

String firstElement = vector.get(0);

System.out.println("First element: " + firstElement);

// Removing elements

vector.remove(1); // Removes the element at index 1

// Iterating over elements

for (String fruit : vector) {

System.out.println(fruit);

}

// Checking size

int size = vector.size();

System.out.println("Vector size: " + size);

// Checking if empty

boolean isEmpty = vector.isEmpty();

System.out.println("Is vector empty? " + isEmpty);

}

}

**More Advanced Operations**

import java.util.Vector;

public class VectorAdvancedExample {

public static void main(String[] args) {

Vector<Integer> numbers = new Vector<>();

numbers.add(10);

numbers.add(20);

numbers.add(30);

// Inserting an element at a specific index

numbers.add(1, 15);

// Removing the first occurrence of an element

numbers.removeElement(20);

// Removing all elements

numbers.clear();

// Checking if an element exists

boolean contains20 = numbers.contains(20);

System.out.println("Contains 20: " + contains20);

// Getting the index of an element

int index = numbers.indexOf(30);

System.out.println("Index of 30: " + index);

// Cloning the vector

Vector<Integer> clonedNumbers = (Vector<Integer>) numbers.clone();

}

}

# PriorityQueue Example :

import java.util.PriorityQueue;

public class PriorityQueueExample {

public static void main(String[] args) {

// Create a PriorityQueue of integers

PriorityQueue<Integer> pq = new PriorityQueue<>();

// Add elements to the queue

pq.add(10);

pq.add(20);

pq.add(5);

pq.add(15);

// Print the elements in priority order

while (!pq.isEmpty()) {

System.out.print(pq.poll() + " ");

}

System.out.println();

// Create a PriorityQueue of custom objects (e.g., Persons)

PriorityQueue<Person> people = new PriorityQueue<>();

people.add(new Person("Alice", 25));

people.add(new Person("Bob", 30));

people.add(new Person("Charlie", 20));

// Print the people in order of their age

while (!people.isEmpty()) {

System.out.println(people.poll().name);

}

}

static class Person implements Comparable<Person> {

String name;

int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

@Override

public int compareTo(Person other) {

return Integer.compare(this.age, other.age);

}

}

}

#HashSet Example:

import java.util.HashSet;

import java.util.Iterator;

public class HashSetExample {

public static void main(String[] args) {

// Create a HashSet to store strings

HashSet<String> names = new HashSet<>();

// Add elements to the HashSet

names.add("Alice");

names.add("Bob");

names.add("Charlie");

names.add("Alice"); // Duplicate element will be ignored

// Check if an element exists

boolean containsAlice = names.contains("Alice");

System.out.println("Contains Alice: " + containsAlice);

// Remove an element

names.remove("Bob");

// Get the size of the HashSet

int size = names.size();

System.out.println("Size of the HashSet: " + size);

// Iterate over the HashSet using an iterator

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

while (iterator.hasNext()) {

System.out.println(iterator.next());

}

// Clear the HashSet

names.clear();

System.out.println("HashSet is now empty: " + names.isEmpty());

}

}

#LinkedHashSet Example:

import java.util.LinkedHashSet;

import java.util.Iterator;

public class LinkedHashSetExample {

public static void main(String[] args) {

// Create a LinkedHashSet to store Strings

LinkedHashSet<String> set = new LinkedHashSet<>();

// Add elements to the set

set.add("Alice");

set.add("Bob");

set.add("Charlie");

set.add("Alice"); // Duplicate, won't be added

// Print the elements in insertion order

System.out.println("Elements in the LinkedHashSet:");

for (String name : set) {

System.out.println(name);

}

// Check if an element exists

boolean containsAlice = set.contains("Alice");

System.out.println("Contains Alice: " + containsAlice);

// Remove an element

set.remove("Bob");

// Get the size of the set

int size = set.size();

System.out.println("Size of the set: " + size);

// Clear the set

set.clear();

System.out.println("Set after clearing: " + set);

// Iterate using an iterator

LinkedHashSet<Integer> numbers = new LinkedHashSet<>();

numbers.add(10);

numbers.add(5);

numbers.add(15);

Iterator<Integer> iterator = numbers.iterator();

while (iterator.hasNext()) {

System.out.print(iterator.next() + " ");

}

}

}

# TreeSet Example:

import java.util.TreeSet;

public class TreeSetExample {

public static void main(String[] args) {

// Create a TreeSet of integers

TreeSet<Integer> numbers = new TreeSet<>();

// Add elements to the set

numbers.add(10);

numbers.add(5);

numbers.add(15);

numbers.add(10); // Duplicate elements are ignored

// Print the elements in sorted order

for (int num : numbers) {

System.out.print(num + " ");

}

System.out.println();

// Create a TreeSet of custom objects (e.g., Persons)

TreeSet<Person> people = new TreeSet<>();

people.add(new Person("Alice", 25));

people.add(new Person("Bob", 30));

people.add(new Person("Charlie", 20));

// Print the people in sorted order based on age

for (Person person : people) {

System.out.println(person.name + " (" + person.age + ")");

}

}

static class Person implements Comparable<Person> {

String name;

int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

@Override

public int compareTo(Person other) {

return Integer.compare(this.age, other.age);

}

}

}

#Iterator Example :

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

public class IteratorExample {

public static void main(String[] args) {

List<String> fruits = new ArrayList<>();

fruits.add("Apple");

fruits.add("Banana");

fruits.add("Orange");

fruits.add("Grape");

// Using Iterator

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

while (iterator.hasNext()) {

String fruit = iterator.next();

System.out.println(fruit);

}

// Using enhanced for loop (simplified iteration)

for (String fruit : fruits) {

System.out.println(fruit);

}

}

}

#ListIterator Example:

import java.util.ArrayList;

import java.util.ListIterator;

public class ListIteratorExample {

public static void main(String[] args) {

ArrayList<String> list = new ArrayList<>();

list.add("Apple");

list.add("Banana");

list.add("Cherry");

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

// Forward iteration

while (iterator.hasNext()) {

String fruit = iterator.next();

System.out.println("Forward: " + fruit);

}

// Backward iteration

while (iterator.hasPrevious()) {

String fruit = iterator.previous();

System.out.println("Backward: " + fruit);

}

// Inserting an element

iterator.add("Date");

// Replacing an element

iterator.set("Dragon Fruit");

// Removing an element

iterator.remove();

// Printing the modified list

for (String fruit : list) {

System.out.println(fruit);

}

}

}

#Enumaration Example:

public enum Color {

RED, GREEN, BLUE

}

public class EnumExample {

public static void main(String[] args) {

Color color1 = Color.RED;

Color color2 = Color.GREEN;

System.out.println(color1);

System.out.println(color2);

// Iterating over enum values

for (Color color : Color.values()) {

System.out.println(color);

}

}

}

#HashMap example:

import java.util.HashMap;

import java.util.Map;

public class HashMapExample {

public static void main(String[] args) {

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

// Adding key-value pairs

map.put("Alice", 25);

map.put("Bob", 30);

map.put("Charlie", 28);

// Retrieving values

int aliceAge = map.get("Alice");

System.out.println("Alice's age: " + aliceAge);

// Removing a key-value pair

map.remove("Bob");

// Checking if a key exists

boolean containsCharlie = map.containsKey("Charlie");

System.out.println("Does the map contain Charlie? " + containsCharlie);

// Getting all keys and values

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

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

}

// Clearing the map

map.clear();

System.out.println("Map is now empty: " + map.isEmpty());

}

}

#LikedHashMap Example:

import java.util.LinkedHashMap;

import java.util.Map;

public class LinkedHashMapExample {

public static void main(String[] args) {

// Create a LinkedHashMap

LinkedHashMap<String, String> linkedHashMap = new LinkedHashMap<>();

// Add key-value pairs

linkedHashMap.put("India", "Delhi");

linkedHashMap.put("USA", "Washington D.C.");

linkedHashMap.put("UK", "London");

linkedHashMap.put("France", "Paris");

// Print the LinkedHashMap

System.out.println("LinkedHashMap: " + linkedHashMap);

// Get value for a key

String capitalOfIndia = linkedHashMap.get("India");

System.out.println("Capital of India: " + capitalOfIndia);

// Check if a key exists

boolean containsKey = linkedHashMap.containsKey("Germany");

System.out.println("Does LinkedHashMap contain Germany? " + containsKey);

// Remove a key-value pair

String removedValue = linkedHashMap.remove("UK");

System.out.println("Removed value for UK: " + removedValue);

// Get all keys

System.out.println("Keys: " + linkedHashMap.keySet());

// Get all values

System.out.println("Values: " + linkedHashMap.values());

// Get size of LinkedHashMap

int size = linkedHashMap.size();

System.out.println("Size of LinkedHashMap: " + size);

// Check if LinkedHashMap is empty

boolean isEmpty = linkedHashMap.isEmpty();

System.out.println("Is LinkedHashMap empty? " + isEmpty);

// Clear the LinkedHashMap

linkedHashMap.clear();

System.out.println("After clearing: " + linkedHashMap);

}

}

#TreeMap Example :

import java.util.\*;

public class TreeMapExample {

public static void main(String[] args) {

// Create a TreeMap to store key-value pairs (where keys are sorted)

TreeMap<String, Integer> treeMap = new TreeMap<>();

// Insert key-value pairs

treeMap.put("Apple", 10);

treeMap.put("Banana", 20);

treeMap.put("Cherry", 30);

treeMap.put("Date", 40);

treeMap.put("Fig", 50);

// Get value associated with a key

Integer value = treeMap.get("Banana");

System.out.println("Value for Banana: " + value);

// Check if a key exists

boolean containsKey = treeMap.containsKey("Orange");

System.out.println("Contains Orange: " + containsKey);

// Check if a value exists

boolean containsValue = treeMap.containsValue(30);

System.out.println("Contains value 30: " + containsValue);

// Get a set of all keys

Set<String> keys = treeMap.keySet();

System.out.println("Keys: " + keys);

// Get a collection of all values

Collection<Integer> values = treeMap.values();

System.out.println("Values: " + values);

// Get a set of all key-value pairs

Set<Map.Entry<String, Integer>> entrySet = treeMap.entrySet();

System.out.println("Entry Set: " + entrySet);

// Remove a key-value pair

treeMap.remove("Date");

System.out.println("After removing Date: " + treeMap);

// Get the first key (lowest key)

String firstKey = treeMap.firstKey();

System.out.println("First Key: " + firstKey);

// Get the last key (highest key)

String lastKey = treeMap.lastKey();

System.out.println("Last Key: " + lastKey);

// Get the key-value pair lower than the given key

Map.Entry<String, Integer> lowerEntry = treeMap.lowerEntry("Cherry");

System.out.println("Lower Entry for Cherry: " + lowerEntry);

// Get the key-value pair higher than the given key

Map.Entry<String, Integer> higherEntry = treeMap.higherEntry("Cherry");

System.out.println("Higher Entry for Cherry: " + higherEntry);

// Get the key-value pair lower or equal to the given key

Map.Entry<String, Integer> floorEntry = treeMap.floorEntry("Cherry");

System.out.println("Floor Entry for Cherry: " + floorEntry);

// Get the key-value pair higher or equal to the given key

Map.Entry<String, Integer> ceilingEntry = treeMap.ceilingEntry("Cherry");

System.out.println("Ceiling Entry for Cherry: " + ceilingEntry);

// Clear the TreeMap

treeMap.clear();

System.out.println("After clearing: " + treeMap);

}

}

#Thread example :

public class Example {

// Method to calculate the sum of two integers

public static int sum(int a, int b) {

return a + b;

}

// Method to find the maximum of two numbers

public static int max(int a, int b) {

if (a > b) {

return a;

} else {

return b;

}

}

// Method to check if a number is even

public static boolean isEven(int num) {

return num % 2 == 0;

}

// Method to print a string n times

public static void printNTimes(String str, int n) {

for (int i = 0; i < n; i++) {

System.out.println(str);

}

}

// Method to calculate the factorial of a number

public static int factorial(int num) {

if (num == 0) {

return 1;

} else {

return num \* factorial(num - 1);

}

}

public static void main(String[] args) {

// Example usage of the methods

int sumResult = sum(5, 3);

System.out.println("Sum: " + sumResult);

int maxResult = max(10, 7);

System.out.println("Maximum: " + maxResult);

boolean isEvenResult = isEven(4);

System.out.println("Is Even: " + isEvenResult);

printNTimes("Hello", 3);

int factorialResult = factorial(5);

System.out.println("Factorial: " + factorialResult);

}

}

#Thread Creation by implementing Runnable interface:

public class MyThread implements Runnable {

@Override

public void run() {

for (int i = 0; i < 5; i++) {

System.out.println("Thread Name : " + Thread.currentThread().getName());

try {

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public static void main(String[] args) {

// Create two threads

Thread thread1 = new Thread(new MyThread(), "Thread-1");

Thread thread2 = new Thread(new MyThread(), "Thread-2");

// Start the threads

thread1.start();

thread2.start();

}

}

#By Inheriting thread class making multithreading :

class MyThread extends Thread {

public void run() {

for (int i = 0; i < 5; i++) {

System.out.println("Child Thread");

try {

Thread.sleep(1000); // Sleep for 1 second

} catch (InterruptedException e) {

System.out.println("Child thread interrupted.");

}

}

}

public static void main(String[] args) {

MyThread t = new MyThread();

t.start(); // Start the thread

for (int i = 0; i < 5; i++) {

System.out.println("Main Thread");

try {

Thread.sleep(1000); // Sleep for 1 second

} catch (InterruptedException e) {

System.out.println("Main thread interrupted.");

}

}

}

}

#Example for bean class:

import java.util.ArrayList;

import java.util.List;

public class Bin {

private double lowerBound;

private double upperBound;

private List<Double> values;

public Bin(double lowerBound, double upperBound) {

this.lowerBound = lowerBound;

this.upperBound = upperBound;

this.values = new ArrayList<>();

}

public double getLowerBound() {

return lowerBound;

}

public double getUpperBound() {

return upperBound;

}

public List<Double> getValues() {

return values;

}

public void addValue(double value) {

if (value >= lowerBound && value <= upperBound) {

values.add(value);

} else {

throw new IllegalArgumentException("Value " + value + " is outside the bin range.");

}

}

public int getSize() {

return values.size();

}

public double getAverage() {

if (values.isEmpty()) {

return 0.0; // Or handle this case differently

}

double sum = 0.0;

for (double value : values) {

sum += value;

}

return sum / values.size();

}

@Override

public String toString() {

return "[" + lowerBound + ", " + upperBound + "] - " + values;

}

}

#Default Method :

interface MyInterface {

void method1(); // Abstract method

default void method2() {

System.out.println("This is a default method.");

}

}

class MyClass implements MyInterface {

@Override

public void method1() {

System.out.println("Implementing method1.");

}

// No need to implement method2 as it's already provided by the interface

}

public class Main {

public static void main(String[] args) {

MyClass obj = new MyClass();

obj.method1();

obj.method2(); // Calling the default method

}

}