**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: BTech Integrated, Semester IV

**Course: Java Programming**

**Experiment No.08**

**Part B**

|  |  |
| --- | --- |
| **Roll no.: A073** | **Name: Aryan Srivastava** |
| **Class: MCA SEM1** | **Batch: B3** |
| **Date of Experiment:** | **Date of Submission:** |
| **Grade:** |  |

1. **Program scenario and Program code**

class genClass<T, U> {

    private T first;

    private U second;

    public genClass(T first, U second) {

        this.first = first;

        this.second = second;

    }

    public void print() {

        System.out.println("String value: " + first);

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

    }

}

public class GenClassExp {

    public static void main(String[] args) {

        genClass<String, Integer> obj1 = new genClass<>("AwwwRyan", 123);

        obj1.print();

    }

}

**Qn2.**

import java.util.Scanner;

public class methods {

    public static <T> void performOperation(T obj1, T obj2, Scanner scanner) {

        System.out.println("Choose an operation: 1. Addition/Concatenation 2. Subtraction");

        int choice = scanner.nextInt();

        if (choice == 1) {

            if (obj1 instanceof String && obj2 instanceof String) {

                String result = (String) obj1 + (String) obj2;

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

            } else if (obj1 instanceof Number && obj2 instanceof Number) {

                double sum = ((Number) obj1).doubleValue() + ((Number) obj2).doubleValue();

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

            }

        } else if (choice == 2) {

            if (obj1 instanceof Number && obj2 instanceof Number) {

                double difference = ((Number) obj1).doubleValue() - ((Number) obj2).doubleValue();

                System.out.println("Difference: " + difference);

            } else if (obj1 instanceof String && obj2 instanceof String) {

                String mainString = (String) obj1;

                String subString = (String) obj2;

                if (mainString.contains(subString)) {

                    System.out.println("Substring found: " + subString);

                } else {

                    System.out.println("Substring not found.");

                }

            }

        } else {

            System.out.println("Invalid choice.");

        }

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        performOperation(5, 10, scanner);

        performOperation("hello World","hello",  scanner);

        scanner.close();

    }

}

**Qn3.**

class BoundedGenericClass<T extends Number> {

    private T value;

    public BoundedGenericClass(T value) {

        this.value = value;

    }

    public void print() {

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

    }

    public double doubleValue() {

        return value.doubleValue();

    }

}

public class restrictions {

    public static void main(String[] args) {

        BoundedGenericClass<Integer> intObject = new BoundedGenericClass<>(10);

        intObject.print();

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

        BoundedGenericClass<Double> doubleObject = new BoundedGenericClass<>(5.5);

        doubleObject.print();

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

    }

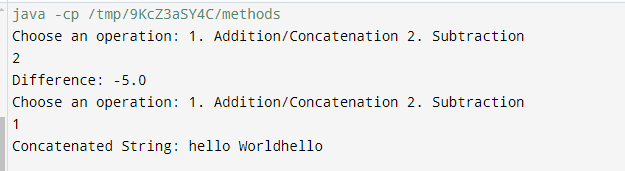
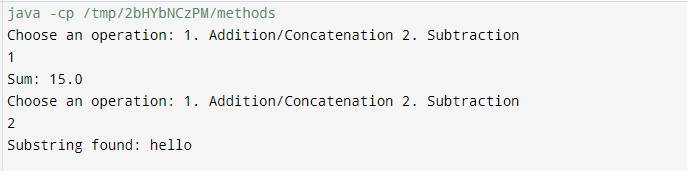
}

1. **Output**

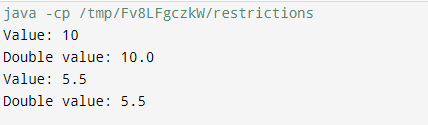
**1.**

****

**2.**

****

**3.**

****

1. **Observation learning and conclusion**

Successfully implemented the use of generics

**4. Questions of Curiosity :**

* 1. Explain Bounded types with help of a code

Bounded types in Java generics allow you to restrict the types that can be used as type parameters. This is done using the extends keyword for upper bounds and the super keyword for lower bounds.

// Bounded type example

class Box<T extends Number> {

private T item;

public void setItem(T item) {

this.item = item;

}

public T getItem() {

return item;

}

}

public class Main {

public static void main(String[] args) {

Box<Integer> integerBox = new Box<>();

integerBox.setItem(10);

System.out.println(integerBox.getItem());

// Box<String> stringBox = new Box<>(); // This will cause a compile-time error

}

}

In this example, Box<T> can only accept types that are subclasses of Number, such as Integer or Double.

* 1. Explain various Generics type parameters

1. Type Parameter <T>: Represents a generic type.

2. Multiple Type Parameters: You can use multiple type parameters.

3. Bounded Type Parameters:

Upper Bounded: T extends ClassName

Lower Bounded: T super ClassName

4. Wildcard Types: ? represents an unknown type.

* 1. Explain various advantages, limitations and applications of Java Generics.

**Advantages**:

Type Safety: Catches type errors at compile time, reducing runtime errors.

Code Reusability: Write a single class or method that works with any data type.

Elimination of Casting: Avoids explicit casting, making the code cleaner.

**Limitations**:

Type Erasure: Generic type information is erased at runtime, which can lead to type-related issues.

Cannot Create Instances of Type Parameters: You cannot instantiate a type parameter directly.

Static Context: Static fields and methods cannot use type parameters.

**Applications**:

Collections Framework: Generics are extensively used in classes like ArrayList, HashMap, etc.

Custom Data Structures: Creating type-safe data structures such as trees, lists, and queues.

APIs: Designing APIs that can work with various types without compromising type safety.