Adavanced Java Lab

Name: Sayyed Sabi Ahsan

Roll No: 24MCA50

Q.A) Write a Java Program to create generic arithmetic calculator to perform addition, subtraction, multiplication and division operation

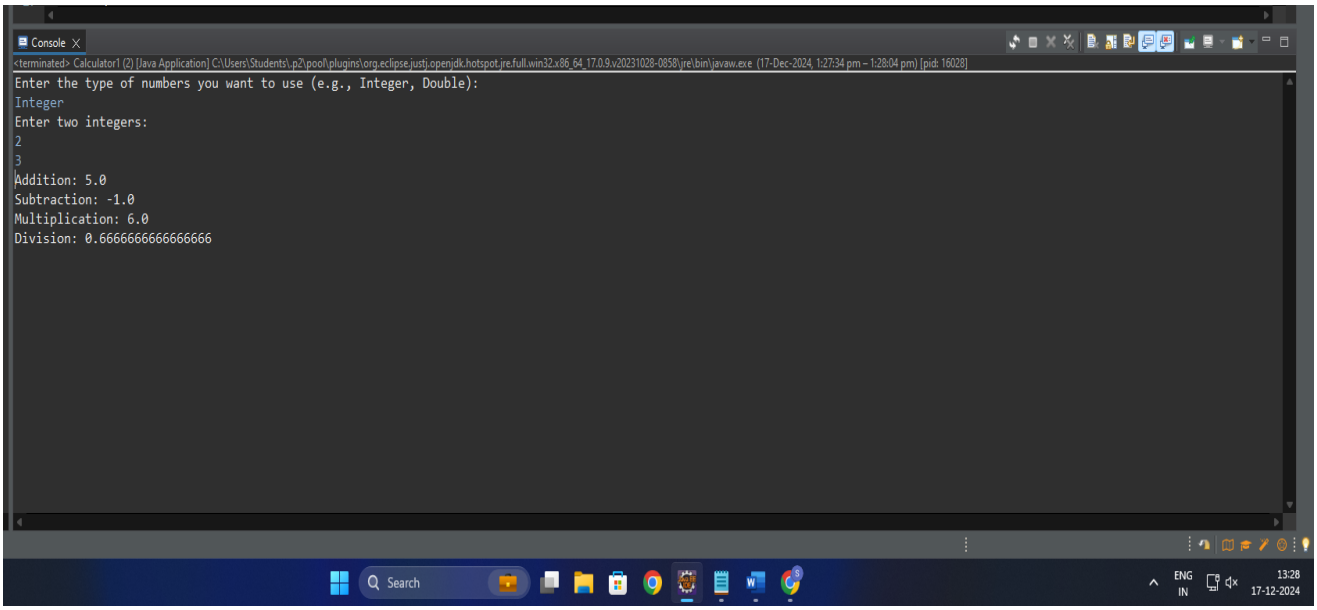
Code:

```
package genericCalculator1Demo;
import java.util.Scanner;
class ArithmeticCalculator<T extends Number> {

    public T add(T num1, T num2) {
        return (T) Double.valueOf(num1.doubleValue() + num2.doubleValue());
    }

    public T subtract(T num1, T num2) {
        return (T) Double.valueOf(num1.doubleValue() - num2.doubleValue());
    }

    public T multiply(T num1, T num2) {
        return (T) Double.valueOf(num1.doubleValue() * num2.doubleValue());
    }

    public T divide(T num1, T num2) {
        if (num2.doubleValue() == 0) {
            throw new ArithmeticException("Division by zero is not allowed.");
```

```
        }
        return (T) Double.valueOf(num1.doubleValue() / num2.doubleValue());
    }
}

public class Calculator1 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the type of numbers you want to use (e.g., Integer, Double): ");
        String type = scanner.nextLine();

        ArithmeticCalculator<Number> calculator = new ArithmeticCalculator<>();

        try {
            if (type.equalsIgnoreCase("Integer")) {
                System.out.println("Enter two integers:");
                int num1 = scanner.nextInt();
                int num2 = scanner.nextInt();

                System.out.println("Addition: " + calculator.add(num1, num2));
                System.out.println("Subtraction: " + calculator.subtract(num1, num2));
                System.out.println("Multiplication: " + calculator.multiply(num1, num2));
                System.out.println("Division: " + calculator.divide(num1, num2));
            } else if (type.equalsIgnoreCase("Double")) {
                System.out.println("Enter two double numbers:");
                double num1 = scanner.nextDouble();
                double num2 = scanner.nextDouble();
```

```
            System.out.println("Addition: " + calculator.add(num1, num2));
            System.out.println("Subtraction: " + calculator.subtract(num1, num2));
            System.out.println("Multiplication: " + calculator.multiply(num1, num2));
            System.out.println("Division: " + calculator.divide(num1, num2));
        } else {
            System.out.println("Invalid type entered. Please use Integer or Double.");
        }
    } catch (Exception e) {
        System.out.println("Error: " + e.getMessage());
    } finally {
        scanner.close();
    }
  }
}
```

Output:



Q.B) Build Spring application to demonstrate Spring AOP - Before, After, around and after-throwing advices using AspectJ XML configuration style. Maintain log details using Logger Factory class for at least one of the advices.

Code:

```
package com.example.aop;
public class AccountService {

    public void addAccount(String accountName) {
        System.out.println("Account added: " + accountName);
    }

    public void updateAccount(String accountName) {
        System.out.println("Account updated: " + accountName);
    }
    public void throwException() {
        throw new RuntimeException("Something went wrong!");
    }
}
```

```
package com.example.aop;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class LoggingFactory {
    private static final Logger logger = LoggerFactory.getLogger(LoggingFactory.class);
```

```
    public static Logger getLogger() {
        return logger;
    }
}
```

```
package com.example.aop;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.AfterThrowing;
import org.slf4j.Logger;

@Aspect
public class LoggingAspect {

    private Logger logger = LoggingFactory.getLogger();

    @Before("execution(* com.example.aop.AccountService.addAccount(..))")
    public void logBeforeAddAccount(JoinPoint joinPoint) {
        logger.info("Before adding account: " + joinPoint.getSignature());
    }

    @After("execution(* com.example.aop.AccountService.updateAccount(..))")
```

```
    public void logAfterUpdateAccount(JoinPoint joinPoint) {
        logger.info("After updating account: " + joinPoint.getSignature());
    }

    @Around("execution(* com.example.aop.AccountService.throwException(..))")
    public Object logAroundException(JoinPoint joinPoint) throws Throwable {
        logger.info("Around method: " + joinPoint.getSignature());
        try {
            return joinPoint.proceed();
        } catch (Throwable ex) {
            logger.error("Exception in method: " + joinPoint.getSignature(), ex);
            throw ex;
        }
    }

    @AfterThrowing(pointcut = "execution(*
com.example.aop.AccountService.throwException(..))", throwing = "ex")
    public void logAfterThrowing(JoinPoint joinPoint, Throwable ex) {
        logger.error("Exception thrown in method: " + joinPoint.getSignature() + " with
message: " + ex.getMessage());
    }
}
```