

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА  
(САМАРСКИЙ УНИВЕРСИТЕТ)»

Институт Информатики и кибернетики   
Кафедра Программных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  
  
к лабораторной работе №1 по дисциплине «Автоматизация тестирования»

Обучающийся группы 6231-020302D \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гижевская В.Д.

Обучающийся группы 6231-020302D \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Алкеев М.Г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лобанков А.А.

Самара 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146727791)

[2 Результаты работы 4](#_Toc146727792)

[2.1 Результаты задания 1 4](#_Toc146727793)

[2.2 Результаты задания 2 5](#_Toc146727794)

[2.3 Результаты задания 3 7](#_Toc146727795)

[ПРИЛОЖЕНИЕ А Листинг программы 8](#_Toc146727796)

1. Постановка задачи

В процессе выполнения заданий ознакомиться с созданием параметризованных и динамических тестов с использованием JUnit5.

Задание 1

Реализовать простое консольное приложение калькулятор, позволяющее производить следующие арифметические операции в 4х системах счисления: сложение, вычитание, умножение, деление. В процессе выполнения задания, рекомендуется использовать шаблон проектирования Фабрика.

Задание 2

Реализовать несколько параметризованных тестов для проверки каждой арифметической операции по каждой системе счисления. Должно быть, как минимум, по одному тесту, которые принимают в качестве параметров csv и источник данных. Отдельно проверить выбрасывание исключения (assertThrows) при попытке деления на 0

Задание 3

Подготовить yaml или csv файл с данными для тестирования. Реализовать динамические тесты на основе данных из подготовленного файла для одной из арифметической операции в каждой системе счисления. У каждого теста должно быть читаемое имя.

1. Результаты работы
   1. Результаты задания 1

Мы создали консольное приложение калькулятора, которое позволяет выполнять арифметические операции в четырех разных системах счисления: двоичной, восьмеричной, десятичной и шестнадцатеричной. Проект использует шаблон проектирования "Фабрика" для создания калькуляторов в соответствии с выбранной системой счисления.

Main класс:

* этот класс является точкой входа в программу и содержит основную логику;
* в бесконечном цикле пользователю предлагается выбрать систему счисления и операцию;
* затем пользователю предлагается ввести два числа в выбранной системе счисления;
* используется CalculatorFactory для создания соответствующего калькулятора на основе выбранной системы и операции;
* выполняется операция, и результат выводится на экран;
* пользователю предлагается выполнить еще одну операцию или завершить программу.

Calculator интерфейс:

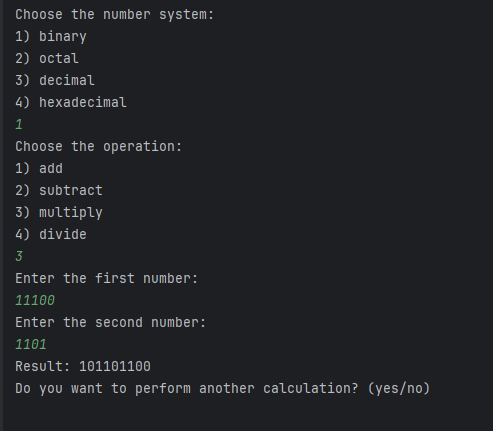
* этот интерфейс определяет метод calculate, который каждый калькулятор должен реализовать.

CalculatorFactory класс:

* этот класс создает конкретные калькуляторы в зависимости от выбранной системы счисления и операции;
* использует оператор switch для определения, какой калькулятор следует создать.

Калькуляторы для различных систем счисления:

* BinaryCalculator, OctalCalculator, DecimalCalculator, и HexadecimalCalculator - каждый из этих классов реализует интерфейс Calculator и предоставляет реализацию для соответствующей системы счисления;
* в этих классах переопределены методы convertNumbersToDecimal и convertResultToBase для преобразования чисел и результатов между системами счисления;
* в конструкторе каждого калькулятора передается выбранная операция.



* 1. Результаты задания 2

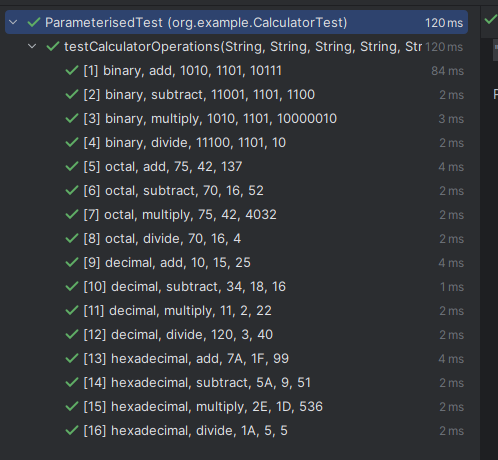
ParameterisedTest класс:

* в этом классе мы использовали аннотацию @ParameterizedTest для создания параметризованных тестов, используя @CsvSource для указания параметров;
* каждый параметризованный тест принимает систему счисления, операцию, два числа и ожидаемый результат;
* в тесте создается фабрика калькуляторов, создается калькулятор для указанной системы и операции, и затем проверяется, что результат соответствует ожидаемому.

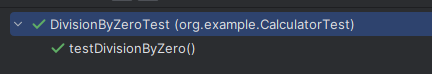
DivisionByZeroTest класс:

* в этом классе мы тестируем выбрасывание исключения при попытке деления на 0.
* создается фабрика калькуляторов, создается калькулятор для деления в десятичной системе, и затем используется assertThrows для проверки, что при попытке деления на 0 выбрасывается исключение ArithmeticException.

Результат работы параметризованного теста:



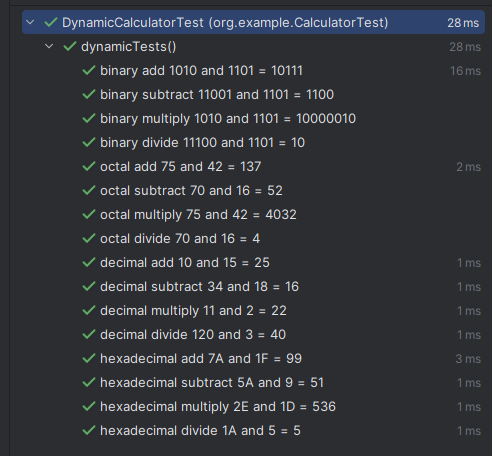
Результат работы проверки выброса исключения при делении на 0:



* 1. Результаты задания 3

DynamicCalculatorTest класс:

* в этом классе мы реализовали динамические тесты на основе данных из CSV-файла.
* используя @TestFactory, создаем поток динамических тестов, которые будут читать данные из CSV-файла.
* в цикле читаются строки из файла, и данные о системе счисления, операции, числах и ожидаемом результате извлекаются из каждой строки.
* для каждой записи создается динамический тест с читаемым именем, где выполняется операция и сравнивается результат с ожидаемым.
* динамические тесты создаются на основе данных из файла, что делает тесты более гибкими и позволяет легко добавлять новые тесты, изменив только файл с данными.



ПРИЛОЖЕНИЕ А  
Листинг программы

package org.example.Calculator;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("Choose the number system:");

String[] bases = {"binary", "octal", "decimal", "hexadecimal"};

int baseChoice = getUserChoice(scanner, bases);

String base = bases[baseChoice - 1];

System.out.println("Choose the operation:");

String[] operations = {"add", "subtract", "multiply", "divide"};

int operationChoice = getUserChoice(scanner, operations);

String operation = operations[operationChoice - 1];

System.out.println("Enter the first number:");

scanner.nextLine();

String num1 = scanner.next();

System.out.println("Enter the second number:");

scanner.nextLine();

String num2 = scanner.next();

CalculatorFactory calculatorFactory = new CalculatorFactory();

Calculator calculator = calculatorFactory.createCalculator(base, operation);

String result = calculator.calculate(operation, base, num1, num2);

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

System.out.println("Do you want to perform another calculation? (yes/no)");

scanner.nextLine();

String choice = scanner.nextLine().toLowerCase();

if (!choice.equals("yes")) {

break;

}

}

}

private static int getUserChoice(Scanner scanner, String[] options) {

for (int i = 0; i < options.length; i++) {

System.out.println((i + 1) + ") " + options[i]);

}

return scanner.nextInt();

}

}

package org.example.Calculator;

public interface Calculator {

String calculate(String operation, String base, String num1, String num2);

}

package org.example.Calculator;

public class CalculatorFactory {

public Calculator createCalculator(String base, String operation) {

return switch (base) {

case "binary" -> new BinaryCalculator(operation);

case "octal" -> new OctalCalculator(operation);

case "decimal" -> new DecimalCalculator(operation);

case "hexadecimal" -> new HexadecimalCalculator(operation);

default -> throw new IllegalArgumentException("Wrong choice of number system");

};

}

}

package org.example.Calculator;

public class CalculatorFactory {

public Calculator createCalculator(String base, String operation) {

return switch (base) {

case "binary" -> new BinaryCalculator(operation);

case "octal" -> new OctalCalculator(operation);

case "decimal" -> new DecimalCalculator(operation);

case "hexadecimal" -> new HexadecimalCalculator(operation);

default -> throw new IllegalArgumentException("Wrong choice of number system");

};

}

}

package org.example.Calculator;

public class BinaryCalculator extends BaseCalculator {

public BinaryCalculator(String operation) {

super(operation);

}

@Override

protected int[] convertNumbersToDecimal(String num1, String num2) {

int[] desNumbers = new int[2];

desNumbers[0] = Integer.parseInt(num1, 2);

desNumbers[1] = Integer.parseInt(num2, 2);

return desNumbers;

}

@Override

protected String convertResultToBase(int result){

return Integer.toBinaryString(result);

}

}

package org.example.Calculator;

public class OctalCalculator extends BaseCalculator{

public OctalCalculator(String operation) {

super(operation);

}

@Override

protected int[] convertNumbersToDecimal(String num1, String num2) {

int[] desNumbers = new int[2];

desNumbers[0] = Integer.parseInt(num1, 8);

desNumbers[1] = Integer.parseInt(num2, 8);

return desNumbers;

}

@Override

protected String convertResultToBase(int result){

return Integer.toOctalString(result);

}

}

package org.example.Calculator;

public class DecimalCalculator extends BaseCalculator{

public DecimalCalculator(String operation) {

super(operation);

}

@Override

protected int[] convertNumbersToDecimal(String num1, String num2) {

int[] desNumbers = new int[2];

desNumbers[0] = Integer.parseInt(num1);

desNumbers[1] = Integer.parseInt(num2);

return desNumbers;

}

@Override

protected String convertResultToBase(int result){

return Integer.toString(result);

}

}

package org.example.Calculator;

public class HexadecimalCalculator extends BaseCalculator{

public HexadecimalCalculator(String operation) {

super(operation);

}

@Override

protected int[] convertNumbersToDecimal(String num1, String num2) {

int[] desNumbers = new int[2];

desNumbers[0] = Integer.parseInt(num1, 16);

desNumbers[1] = Integer.parseInt(num2, 16);

return desNumbers;

}

@Override

protected String convertResultToBase(int result){

return Integer.toHexString(result);

}

}

package org.example.CalculatorTest;

import static org.junit.jupiter.api.Assertions.assertEquals;

import static org.junit.jupiter.api.Assertions.assertThrows;

import org.example.Calculator.Calculator;

import org.example.Calculator.CalculatorFactory;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.CsvSource;

public class ParameterisedTest {

@ParameterizedTest

@CsvSource({

"binary, add, 1010, 1101, 10111",

"binary, subtract, 11001, 1101, 1100",

"binary, multiply, 1010, 1101, 10000010",

"binary, divide, 11100, 1101, 10",

"octal, add, 75, 42, 137",

"octal, subtract, 70, 16, 52",

"octal, multiply, 75, 42, 4032",

"octal, divide, 70, 16, 4",

"decimal, add, 10, 15, 25",

"decimal, subtract, 34, 18, 16",

"decimal, multiply, 11, 2, 22",

"decimal, divide, 120, 3, 40",

"hexadecimal, add, 7A, 1F, 99",

"hexadecimal, subtract, 5A, 9, 51",

"hexadecimal, multiply, 2E, 1D, 536",

"hexadecimal, divide, 1A, 5, 5"

})

public void testCalculatorOperations(String base, String operation, String num1, String num2, String expectedResult) {

CalculatorFactory calculatorFactory = new CalculatorFactory();

Calculator calculator = calculatorFactory.createCalculator(base, operation);

String result = calculator.calculate(operation, base, num1, num2);

assertEquals(expectedResult, result);

}

}

package org.example.CalculatorTest;

import org.example.Calculator.Calculator;

import org.example.Calculator.CalculatorFactory;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertThrows;

public class DivisionByZeroTest {

@Test

public void testDivisionByZero() {

CalculatorFactory calculatorFactory = new CalculatorFactory();

Calculator calculator = calculatorFactory.createCalculator("decimal", "divide");

assertThrows(ArithmeticException.class, () -> {

calculator.calculate("divide", "decimal", "10", "0");

});

}

}

package org.example.CalculatorTest;

import org.example.Calculator.Calculator;

import org.example.Calculator.CalculatorFactory;

import org.junit.jupiter.api.DynamicTest;

import org.junit.jupiter.api.TestFactory;

import org.junit.jupiter.api.Assertions;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Stream;

public class DynamicCalculatorTest {

@TestFactory

Stream<DynamicTest> dynamicTests() throws IOException {

List<DynamicTest> dynamicTests = new ArrayList<>();

String csvFile = "src/test/resources/test\_data.csv";

String line = "";

String delimiter = ",";

try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {

while ((line = br.readLine()) != null) {

String[] data = line.split(delimiter);

String base = data[0];

String operation = data[1];

String num1 = data[2];

String num2 = data[3];

String expectedResult = data[4];

String testName = String.format("%s %s %s and %s = %s", base, operation, num1, num2, expectedResult);

DynamicTest dynamicTest = DynamicTest.dynamicTest(testName, () -> {

CalculatorFactory calculatorFactory = new CalculatorFactory();

Calculator calculator = calculatorFactory.createCalculator(base, operation);

String result = calculator.calculate(operation, base, num1, num2);

Assertions.assertEquals(expectedResult, result);

});

dynamicTests.add(dynamicTest);

}

}

return dynamicTests.stream();

}

}