MEKELLE UNIVERSITY

EIT-M

SCHOOL OF COMPUTING

DEPARTMENT OF SOFTWARE ENGINEERING

COURSE TITLE: Software Engineering Tools and practices

Assignment: On Test Driven Development and Git control system

Group Members

TEAM MEMBERS	ID
1.Amanuel Temesgen	EITM/UR158024/11
2.Getahun Haftu	EITM/UR158046/11
3.Natnaeal Haile	EITM/UR158028/11
4.Gebremedhin G/meskel	EITM/UR156034/11
5.Brizaf Welensea	EITM/UR156025/11

Submitted To : inst. Haftamu K.

Submission Date: 5/17/2024 Gc

Test-Driven Development (TDD)

Introduction

Test-Driven Development (TDD) is a software development methodology that emphasizes writing tests before writing the actual code. It is an iterative approach that ensures software quality and reliability by integrating testing into the development process from the outset. TDD is a fundamental practice in agile software development. This practice involves a repetitive cycle of writing a failing test (Red), implementing the code to make the test pass (Green), and then refactoring the code to improve its structure without changing its behavior (Refactor).

Key Concepts of TDD

Red-Green-Refactor Cycle: TDD is characterized by a repetitive cycle of three stages:

- Red: Write a test for a new feature or functionality. Since the feature is not yet implemented, the test will fail. This step ensures that the test is effective and can detect the absence of the feature.
- Green: Write the minimal amount of code required to pass the test. At this stage, the focus is solely on making the test pass, without worrying about code quality or optimization.
- Refactor: Improve and optimize the code while ensuring that all tests still pass. Refactoring may include cleaning up the code, improving readability, and enhancing performance.

Core Principles of Test-Driven Development (TDD)

- **Unit Tests:** TDD primarily involves writing unit tests, which test individual components or units of the software in isolation. These tests validate that each part of the software behaves as expected.
- **Continuous Feedback**: TDD provides continuous feedback to the developer. By frequently running tests, developers can quickly identify and fix defects, reducing the likelihood of bugs in the final product.
- **Design by Contract**: TDD encourages developers to think about the desired behavior and interface of the software components before implementing them. This leads to a more deliberate and thoughtful design process.

Benefits of TDD

- ☞ **Higher Quality Code**: Writing tests first ensures that the code meets the specified requirements and is less likely to contain defects.
- ☞ **Better Design**: TDD encourages modular, loosely coupled code, making the software easier to maintain and extend.
- ☞ **Increased Productivity**: Although writing tests initially takes time, TDD can increase overall productivity by reducing the time spent debugging and fixing issues later in the development process.
- ☞ **Documentation**: Tests serve as a form of documentation, providing examples of how the code is intended to be used and what its expected behavior is.

Challenges of TDD

- **Learning Curve**: TDD requires a shift in mindset and can be challenging for developers who are accustomed to writing code first and tests later.
- **Initial Time Investment**: Writing tests before code can seem slower initially, which may be a hurdle in fast-paced development environments.
- **Maintenance of Tests**: As the code base evolves, tests need to be updated, which can add to the maintenance burden.

JUnit: A Key Tool for TDD

JUnit is a widely used testing framework for Java programming that plays a crucial role in Test-Driven Development (TDD). It provides a simple and efficient way to write and run repeatable tests, ensuring that individual components of a Java application behave as expected. Developed by Kent Beck and Erich Gamma, JUnit has become an integral part of Java development, enabling developers to adopt TDD practices seamlessly.

Features of JUnit

Annotations: JUnit uses annotations to identify test methods and configure testing behavior. Key annotations include:

- ✓ @Test: Marks a method as a test method.
- ✓ @Before: Specifies a method to run before each test.
- ✓ @After: Specifies a method to run after each test.
- ✓ @BeforeClass and @AfterClass: Specify methods to run once before/after all tests in the class.

Assertions: JUnit provides a set of assertion methods to test conditions. Common assertions include:

- ✓ assertEquals(expected, actual): Checks if two values are equal.
- ✓ assertTrue(condition): Checks if a condition is true.
- ✓ assertFalse(condition): Checks if a condition is false.
- ✓ assertNull(object): Checks if an object is null.
- ✓ assertNotNull(object): Checks if an object is not null.

TDD in Practice

To illustrate how TDD works in practice we use

- ✓ Hamcrest Core 1.3.jar is a library for Java that provides a framework for writing matcher objects, allowing for more expressive and readable tests. It is commonly used alongside JUnit to enhance the readability and maintainability of test code.
- ✓ JUnit 4.13.2.jar is a version of JUnit, a popular open-source framework used for unit testing in Java. JUnit provides an easy-to-use structure for writing and running tests, enabling developers to verify that their code behaves as expected. JUnit 4.13.2 includes a variety of features and improvements that make it an essential tool for test-driven development (TDD) and other testing methodologies.

And we wrote a test case code first then the code to pass this test case code as follows.

The Test Case Code

In our practical implementation of Test-Driven Development (TDD), we have focused on thoroughly testing the ChangeNumber class. This class contains methods for converting numbers between different bases (binary, octal, hexadecimal). Below, we provide an overview of the tests we wrote and their purposes, demonstrating the principles of TDD by ensuring that the code meets the specified requirements before writing the implementation.

Overview of the Tests

Initialization and Teardown Methods

@Before method (testToHex): This method is executed before any test cases run. It tests the toHex method to ensure it converts decimal numbers to their hexadecimal equivalents correctly.

@After method (testToOctal): This method runs after all other tests have executed. It tests the toOctal method to verify its correctness in converting decimal numbers to octal.

Unit Test Methods

testToBinary: This method tests the toBinary function to ensure correct conversion of decimal numbers to binary. It uses assertions to compare expected and actual results.

testFromBinary: This method validates the fromBinary function, ensuring it correctly converts binary strings back to decimal numbers.

testFromOctal: This method checks the fromOctal function to verify correct conversion from octal strings to decimal.

testFromHex: This method tests the fromHex function to ensure correct conversion from hexadecimal strings to decimal.

Ignored Test:

testFromHexIntension: This test is intentionally ignored. It demonstrates how to skip certain tests that may be irrelevant or incomplete at the moment.

Performance Test

testWithTimeOut: This test checks the performance of the toOctal method by measuring its execution time. It ensures the method executes within an acceptable duration, highlighting potential performance issues.

Key Assertions Used

assertEquals: This assertion is used to compare the expected result with the actual result of the conversion methods.

assertTrue and **assertFalse:** These assertions check specific boolean conditions. For example, assertTrue(1001 == cNumber.toBinary(9)) ensures the binary conversion is correct.

assertTrue with a performance check ensures the method does not exceed the specified execution time.

We follow the following steps to achieve TDD Approach

1. The test methods were written before the implementation of the ChangeNumber class methods. This helps define the expected behavior of the methods.
2. Initially, running these tests would result in failures since the methods have not been implemented yet.
3. The next step is to implement the ChangeNumber class methods (toBinary, toOctal, toHex, fromBinary, fromOctal, and fromHex) to pass the tests.
4. Once the tests pass, refactor the code to improve its structure without altering its behavior, ensuring the tests still pass after each change.

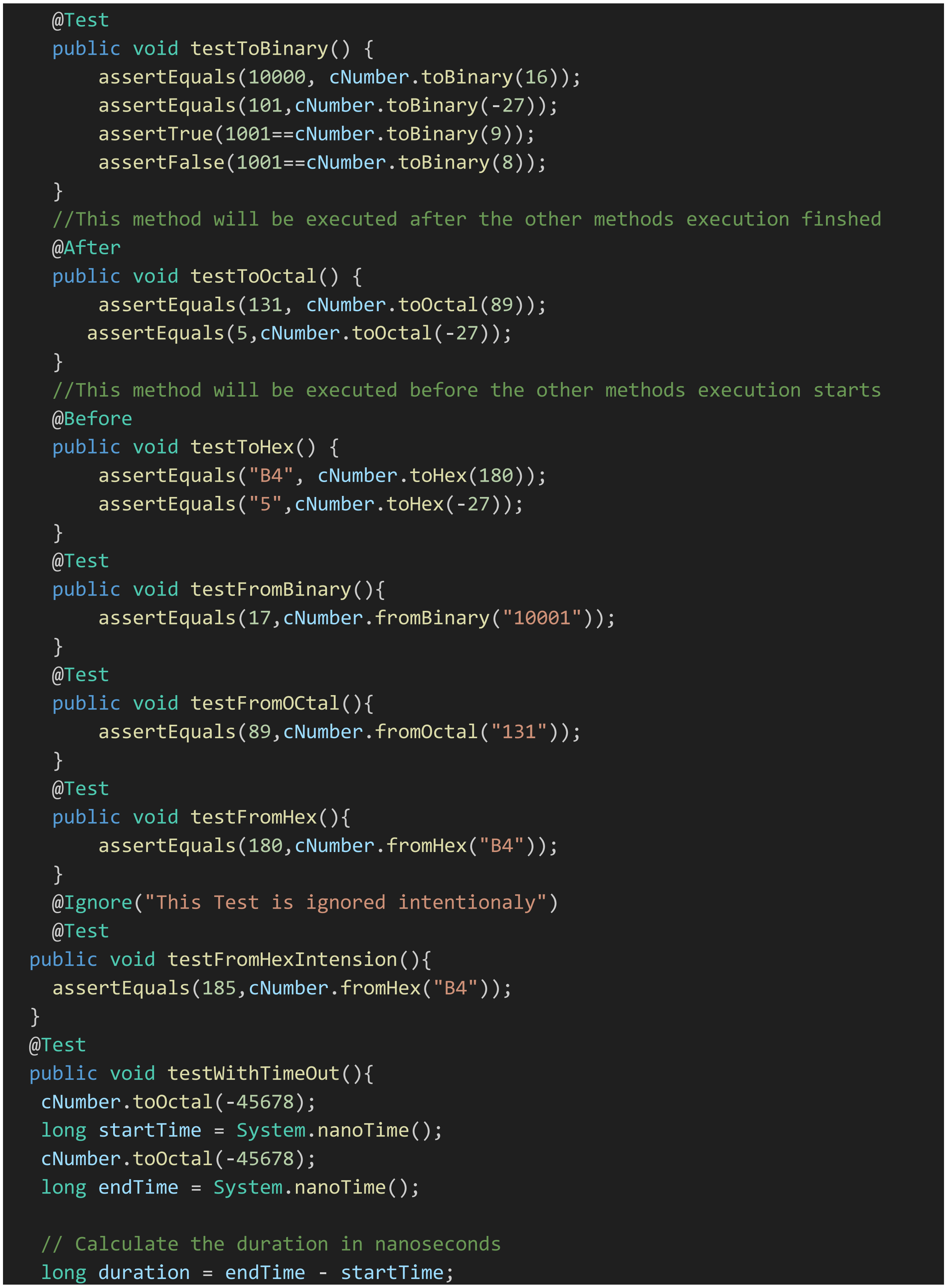
By adhering to TDD principles, we have ensured that the ChangeNumber class methods are correctly implemented and validated against a comprehensive set of test cases. This process helps in identifying and fixing bugs early, improving code quality, and ensuring the software meets its requirements.

Here’s the Test case ChangeNumberTest.java that we wrote before the implementation ChangeNumber.java

```java
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import org.junit.After;
import org.junit.Before;
import org.junit.Ignore;
import org.junit.Test;

public class ChangeNumberTest {
    private ChangeNumber cNumber = new ChangeNumber();
```



```java
    @Test
    public void testToBinary() {
        assertEquals(10000, cNumber.toBinary(16));
        assertEquals(101,cNumber.toBinary(-27));
        assertTrue(1001==cNumber.toBinary(9));
        assertFalse(1001==cNumber.toBinary(8));
    }
    //This method will be executed after the other methods execution finshed
    @After
    public void testToOctal() {
        assertEquals(131, cNumber.toOctal(89));
       assertEquals(5,cNumber.toOctal(-27));
    }
    //This method will be executed before the other methods execution starts
    @Before
    public void testToHex() {
        assertEquals("B4", cNumber.toHex(180));
        assertEquals("5",cNumber.toHex(-27));
    }
    @Test
    public void testFromBinary(){
        assertEquals(17,cNumber.fromBinary("10001"));
    }
    @Test
    public void testFromOCtal(){
        assertEquals(89,cNumber.fromOctal("131"));
    }
    @Test
    public void testFromHex(){
        assertEquals(180,cNumber.fromHex("B4"));
    }
    @Ignore("This Test is ignored intentionaly")
    @Test
  public void testFromHexIntension(){
    assertEquals(185,cNumber.fromHex("B4"));
  }
  @Test
  public void testWithTimeOut(){
   cNumber.toOctal(-45678);
   long startTime = System.nanoTime();
   cNumber.toOctal(-45678);
   long endTime = System.nanoTime();

   // Calculate the duration in nanoseconds
   long duration = endTime - startTime;
   // Check if the duration is less than 1 millisecond (1,000 nanoseconds)
   assertTrue("Method took too long to execute", duration < 1_000);
  }
}
```

The functions To Pass The Test Case:

We've implemented the ChangeNumber class, which provides methods for converting decimal numbers to binary, octal, and hexadecimal representations, as well as converting from these bases back to decimal.

Here's the implementation of ChangeNumber.java

```
public class ChangeNumber {

    public int toBinary(int decimal) {
        if(decimal<0){
            int result=positiveDecimalTobinary(decimal)*-1;
            String resultDecimal=result+"";
            char[] array=resultDecimal.toCharArray();
            String resulString="";
            for(int i=0;i<array.length;i++){
                if(array[i]=='0') array[i]='1';
                else array[i]='0';
                resulString+=array[i];
            }
            result=fromBinary(resulString);
            result++;
            return positiveDecimalTobinary(result);
        }
      return positiveDecimalTobinary(decimal);
    }

    public int toOctal(int decimal) {
        if(decimal<0){
            int result=toBinary(decimal);
            result=fromBinary(result+"");
            return positiveDecimalToOctal(result);
        }
        return positiveDecimalToOctal(decimal);
    }
    public String toHex(int decimal) {
        if(decimal<0){
            int result=toBinary(decimal);
            result=fromBinary(result+"");
            return positiveDecimalToHex(result);
        }
       return positiveDecimalToHex(decimal);
    }


    public int fromBinary(String binaryString){
        char[] array=binaryString.toCharArray();
        int result=0;
        int tempResult=0;
        for(int i=0;i<array.length;i++){
             tempResult=getNumFromChar(array[i])*(int)Math.pow(2, array.length-i-
1);
             result+=tempResult;
        }
        return result;
    }
    public int fromOctal(String octalString){
        char[] array=octalString.toCharArray();
        int result=0;
        int tempResult=0;
        for(int i=0;i<array.length;i++){
             tempResult=Integer.parseInt(array[i]+"")*(int)Math.pow(8,
array.length-i-1);
             result+=tempResult;
        }
        return result;
    }
    public int fromHex(String hexString){
        char[] array=hexString.toCharArray();
        int result=0;
        int tempResult=0;
        for(int i=0;i<array.length;i++){
             tempResult=getNumFromChar(array[i])*(int)Math.pow(16, array.length-
i-1);
             result+=tempResult;
        }
        return result;
    }
    private int positiveDecimalTobinary(int decimal){
        int counter = 0;
        int remainder = 0;
        int result = 0;
        while (decimal != 0) {
            remainder = decimal % 2;
            result += (int) Math.pow(10, counter) * remainder;
            decimal = decimal / 2;
            counter++;
        }
        return result;
    }

    private int positiveDecimalToOctal(int decimal){
        int counter = 0;
        int remainder = 0;
        int result = 0;
        while (decimal != 0) {
            remainder = decimal % 8;
            result += (int) Math.pow(10, counter) * remainder;
            decimal = decimal / 8;
            counter++;
        }
        return result;
    }
    private String positiveDecimalToHex(int decimal){
        int remainder = 0;
        String hexString = "";
        String hexCopy = "";
        while (decimal != 0) {
            remainder = decimal % 16;
            hexCopy = getCharNum(remainder);
            hexCopy += hexString;
            hexString = hexCopy;
            decimal = decimal / 16;
        }
        return hexString;
    }
    private String getCharNum(int num) {
        if (num == 10)
            return "A";
        else if (num == 11)
            return "B";
        else if (num == 12)
            return "C";
        else if (num == 13)
            return "D";
        else if (num == 14)
```

```
            return "E";
        else if (num == 15)
            return "F";
        else
            return num + "";
    }
    private int getNumFromChar(char c){
    if (c=='A') return 10;
    else if(c=='B') return 11;
    else if(c=='C') return 12;
    else if(c=='D') return 13;
    else if(c=='E') return 14;
    else if(c=='F') return 15;
    else  return Integer.parseInt(c+"");
    }
}
```