**SE Tools Exercises**

**9조 김민준, 김재원, 이동섭, 허윤영**

Our group made a calculator source code and used the tool(Git, JavaDoc, JUnit).

**1. Git**

Git URL: <https://github.com/J1-coding/Calculator>

텍스트, 모니터, 스크린샷, 화면이(가) 표시된 사진

자동 생성된 설명

텍스트, 모니터, 스크린샷, 검은색이(가) 표시된 사진

자동 생성된 설명

텍스트, 모니터, 스크린샷, 검은색이(가) 표시된 사진

자동 생성된 설명

**pros**:

1. Collaboration is the biggest advantage of using GitHub.

2. There is branching capabilities as a function for collaboration. For people who do not remain in the same physical location.

3. Git refers to an easy solution needing no setup for new users.

4. Unlike centralized version control systems, Git branches are cheap and easy to merge.

5. Git is a distributed version control system. Each developer gets their own local repository.

6. Having a full local history makes Git fast, since you don’t need a network connection to create commits.

**cons**:

1. Git is not built for security so, there are many security issues that can occur.

2. Self-hosting a Git server is a big security problem. Users probably not qualified to maintain a self-hosted Git solution hosting sensitive data. When self-hosting a Git server, it is important to secure the “.git” directory.

3. A publicly accessible Git directory can allow malicious things to clone the repository. Then they can scan it for secrets in the code or within historical records.

4. With many companies relying on Git for code management, Git has become a target for hackers.

**2. JavaDoc**

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

**pros**:

1. It is in HTML format, that other related APIs can be accessed through hyperlinks

2. It is easy to see at a glance by organizing information on Java code in documents, so it can help others understand.

**cons**:

1. In public APIs, javadoc can help users understand, but in private code, it is unnecessary because annotations can cause distraction.

**3. JUnit**

텍스트이(가) 표시된 사진

자동 생성된 설명

**pros**:

1. It makes development easier because fast and accurate testing is possible by comparing expected behavior with actual behavior.

2. Even if the application is changed (extension of function, refactoring, etc.), it can be checked whether the application works correctly.

3. Because the test can be divided into small units, it is possible to test each function as well as the integrated test.

**cons**:

1. It takes a long time to develop because you have to write additional test code

**Source Code**

**Calculator.class**

|  |
| --- |
| // |
|  | // Source code recreated from a .class file by IntelliJ IDEA |
|  | // (powered by FernFlower decompiler) |
|  | // |
|  |  |
|  | public class Calculator { |
|  | public Calculator() { |
|  | } |
|  |  |
|  | public static double add(double a, double b) { |
|  | return a + b; |
|  | } |
|  |  |
|  | public static double subtract(double a, double b) { |
|  | return a - b; |
|  | } |
|  |  |
|  | public static double multiply(double a, double b) { |
|  | return a \* b; |
|  | } |
|  |  |
|  | public static double divide(double a, double b) { |
|  | return a / b; |
|  | } |
|  | } |

**CalculatorTest.class**

|  |
| --- |
| import org.junit.jupiter.api.Assertions; |
|  | import org.junit.jupiter.api.DisplayName; |
|  | import org.junit.jupiter.api.Test; |
|  |  |
|  | import java.util.Scanner; |
|  |  |
|  |  |
|  | class CalculatorTest { |
|  |  |
|  | @Test |
|  | @DisplayName("더하기\_테스트") |
|  | public void add() { |
|  | //given |
|  | double a = 10; |
|  | double b = 30; |
|  |  |
|  | //then |
|  | Assertions.assertEquals(40,Calculator.add(a,b)); |
|  | } |
|  |  |
|  | @Test |
|  | @DisplayName("빼기\_테스트") |
|  | public void subtract() { |
|  | //given |
|  | double a = 30; |
|  | double b = 10; |
|  |  |
|  | //then |
|  | Assertions.assertEquals(20,Calculator.subtract(a,b)); |
|  | } |
|  |  |
|  | @Test |
|  | @DisplayName("곱하기\_테스트") |
|  | public void multiply() { |
|  | //given |
|  | double a = 10; |
|  | double b = 30; |
|  |  |
|  | //then |
|  | Assertions.assertEquals(300,Calculator.multiply(a,b)); |
|  | } |
|  |  |
|  | @Test |
|  | @DisplayName("나누기\_테스트") |
|  | public void divide() { |
|  | //given |
|  | double a = 30; |
|  | double b = 10; |
|  |  |
|  | //then |
|  | Assertions.assertEquals(3,Calculator.divide(a,b)); |
|  | } |
|  | } |

**Main.class**

|  |
| --- |
| import java.util.Scanner; |
|  |  |
|  | public class Main { |
|  |  |
|  | public static void main(String[] args) { |
|  |  |
|  | System.out.println("[계산기 프로그램 만들기 실시]"); |
|  |  |
|  | //첫번째와 두번째 정수 값을 담을 변수 선언 |
|  | double value1; |
|  | double value2; |
|  |  |
|  | //사용자로부터 키보드 입력을 받기 위한 scanner 객체 선언 실시 |
|  | Scanner scan = new Scanner(System.in); |
|  |  |
|  | //사용자로부터 정수값과 연산자 부호값을 입력받습니다 |
|  | System.out.print("첫번째 값 : "); |
|  | value1 = scan.nextInt(); |
|  |  |
|  | System.out.print("두번째 값 : "); |
|  | value2 = scan.nextInt(); |
|  |  |
|  | System.out.print("사칙연산부호 (+,-,\*,/) : "); |
|  | String sign = scan.next(); |
|  |  |
|  | double result = 0; |
|  |  |
|  |  |
|  | // 연산자 로직 처리 |
|  | if(sign.equals("+")) { |
|  | result = Calculator.add(value1,value2); |
|  | } |
|  | else if(sign.equals("-")) { |
|  | result= Calculator.subtract(value1,value2); |
|  | } |
|  | else if(sign.equals("\*")) { |
|  | result =Calculator.multiply(value1,value2); |
|  | } |
|  | else if(sign.equals("/")) { |
|  | result = Calculator.divide(value1,value2); |
|  | } |
|  | else { |
|  | System.out.println("알수없는 연산자입니다 … "); |
|  | } |
|  |  |
|  | System.out.println("result = " + result); |
|  |  |
|  | } |
|  | } |