

Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre



Assignment #1
Software Engineering

[Course Code: COMP 401]

[For the partial fulfillment of 4th year/1st Semester in Computer Engineering]

Submitted by:

Sabin Thapa

Roll no. 54

CE 4th Year

Submitted to:

Rabindra Bista, PhD

Associate Professor

Department of Computer Science and Engineering

Submission date: Oct. 11, 2022

Q. 1. Briefly describe the principles of software engineering and the principles of agile development.

Ans:

Software engineering is the establishment and use of sound engineering principles in order to obtain economically developed software that is reliable and works efficiently on real machines. It is an engineering discipline that is concerned with all aspects of software production i.e, from the early stages of system specification through to maintaining the system after it has gone to use.

Irrespective to the development techniques used, some fundamental principles apply to all types of software system. These principles are discussed below.

- a. Every system should be developed using a managed and understood development process. Although different processes are used on the basis of the type of the software, the development process should be one that is managed and understood by all.
- b. For all types of system, dependability and performance are important. The dependability of the software represents the users' degree of trust in that system. Dependability covers the related system attributes like reliability, availability and security. Similarly, performance is another important factor. It is the degree to which the software meets its objectives. A software should perform as specified. So, building a dependable software and one that has a great performance is one of the software engineering principles.
- c. Another important principle is understanding and managing the software specification and requirements. Here, requirements are the descriptions of services that a software system must provide and the constraints under which it must operate and software specification is a technical document that describes the features and behavior of a software system. Simply put, specification is a technical document with the analysed requirements. Understanding and managing the software specification and requirements are software engineering principles because the entire project depends on the requirements and having a document that serves for this purpose would help ease the development process.
- d. Wherever possible, software should be reused rather than be written again. If there are softwares already developed, then they should be reused. Writing a new software that has already been developed is costly. So, focus should be given to reuse rather than remake.

Besides the principles discussed above, below are some other software engineering principles that can be applied to all software products:

a. Modularity

The principle of modularity implies separating software into components according to functionality and responsibility. With this principle, each component can be tested and debugged separately and it becomes easier to understand the system.

b. Generality

The principle of generality focuses on designing software that is free from unnatural restrictions and limitations.

c. Incremental development

Following the principle of incremental development, software is developed in small increments. It simplifies verification as we only need to deal with the increment recently added.

d. Consistency

It is the recognition of the fact that it is easier to do things in a familiar context. Design and development should focus on consistency in the UI and the overall components and features.

Hence, discussed above are some of the fundamental software engineering principles that should be applied to all software development. Next, let's discuss on the principles of agile development.

Agile development is a development methodology in which program specification, design and implementation are inter-leaved and the system is developed as a series of increments or versions with stakeholders involved in version specification and evaluation. It focuses on working code rather than tedious documentation and design. This method is based on an iterative approach to software development and is intended to deliver working software quickly and evolve this quickly to meet changing requirements.

The principles of agile development are discussed in the following section:

a. Customer involvement

In the case of agile development, customers are closely involved throughout the development process. Their role is to provide and prioritize new system requirements and to evaluate the iterations of the system. Customers are the ones that give continuous feedback

and recommendations.

b. Incremental delivery

Rather than developing the entire system at once, the software is developed in increments and the customer specifies increments to be included in each iteration.

c. People not process

Another principle of agile development is to give the development team the freedom by recognizing and exploiting their skills. The team members should be left to develop their own ways of working without prescriptive processes.

d. Embrace change

Change is inevitable. System requirements should be expected to change and the system should be designed to accommodate these changes.

e. Maintain simplicity

Both the software being developed and the development process used should be as simple as possible. Complexity should be eliminated from the system wherever possible.

f. Maintain a sustainable working pace

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

g. Promote Face-to-Face Conversations

The only way to get a quick response in order to move swiftly with the project is by talking to the team member in person. This can be done by working in the same physical space or having distributed teams.

To sum up, the principles of software engineering are crucial to any software product development. Failure to implement these would result in a product that is not efficient, robust and diligent. Similarly, the principles of agile development as discussed above are crucial in the development of many modern software products. It gives the customers the feeling of control and the product life cycle is reduced significantly.

Q. 2. J-Unit is a popular software tool for unit testing. Demonstrate how it works.

Ans:

JUnit is an open source unit testing framework for the Java programming language. It is used to write and run repeatable tests by the developers. As the name implies, it is used for unit testing of a small chunk of code. If we're following a test-driven methodology, we must write and execute unit test first before any code. This idea is known as "first testing then coding", which emphasizes on setting up the test data for a piece of code that can be tested first and then implemented. Once, the code is written, all the tests should be executed and all of them should pass. Every time a new code is added, all the test cases should be run to ensure that nothing is broken.

Features of JUnit

- a. JUnit is an open source framework. It is available freely to the users.
- b. It provides test runners for running tests, annotations to identify test methods and assertions for testing expected results.
- c. It allows us to write code faster, which increases quality.
- d. It is simple and takes less time.
- e. JUnit tests can be run automatically and they can check their own results and provide immediate feedback. There's no need of manual hassle.
- f. JUnit tests can be organized into test suites containing test cases, and even other test suites.

Before moving towards the installation and demonstration, we should understand what a *Unit Test Case* is. A Unit Test Case is a part of code, which ensures that another part of the code works as expected. For this, a test framework like JUnit is required. A formal written test case is characterized by a known input and an expected output - which is worked out before the test is executed. The known input tests a pre-condition and the expected output tests a post-condition. We require at least two unit test cases for each requirement - one positive and another negative. If a requirement has further sub-requirements, then each of them must have at least two test cases.

Setup

Since, JUnit is a Java framework, we should install JDK to run Java programs. In the following sections, the installation process and the entire setup process to run Java programs on my machine (Linux - Ubuntu) is shown.

Local Environment Setup

1. Java Installation

```
sabinthapa @ pop-os ~/Softwares $ sudo dpkg -i jdk-19_linux-x64_bin.deb
(Reading database ... 209842 files and directories currently installed.)
Preparing to unpack jdk-19_linux-x64_bin.deb ...
Unpacking jdk-19 (19-ga) over (19-ga) ...
Setting up jdk-19 (19-ga) ...

sabinthapa @ pop-os ~/Softwares $ sudo update-alternatives --install /usr/bin/java java /usr/lib/jvm/jdk-19/bin/java 1
update-alternatives: using /usr/lib/jvm/jdk-19/bin/java to provide /usr/bin/java (java) in auto mode

sabinthapa @ pop-os ~/Softwares $ sudo update-alternatives --install /usr/bin/javac javac /usr/lib/jvm/jdk-19/bin/javac 1
update-alternatives: using /usr/lib/jvm/jdk-19/bin/javac to provide /usr/bin/javac (javac) in auto mode

sabinthapa @ pop-os ~/Softwares $ java --version
java 19 2022-09-20
Java(TM) SE Runtime Environment (build 19+36-2238)
Java HotSpot(TM) 64-Bit Server VM (build 19+36-2238, mixed mode, sharing)
```

Fig 1: Java Installation on my linux machine

2. JAVA Environment Setup

The JAVA_HOME environment variable is then set to point to the base directory location where Java is installed on my machine.

Command:

export JAVA_HOME = /usr/local/java-current

```
sabinthapa @ pop-os ~/Softwares $ export JAVA_HOME=/usr/local/java-current
```

3. Set JUnit Environment

JUnit archive is downloaded and the path for JUnit is set.

```
sabinthapa @ pop-os ~/Softwares $ export JUNIT_HOME=~/.Softwares/JUnit
```

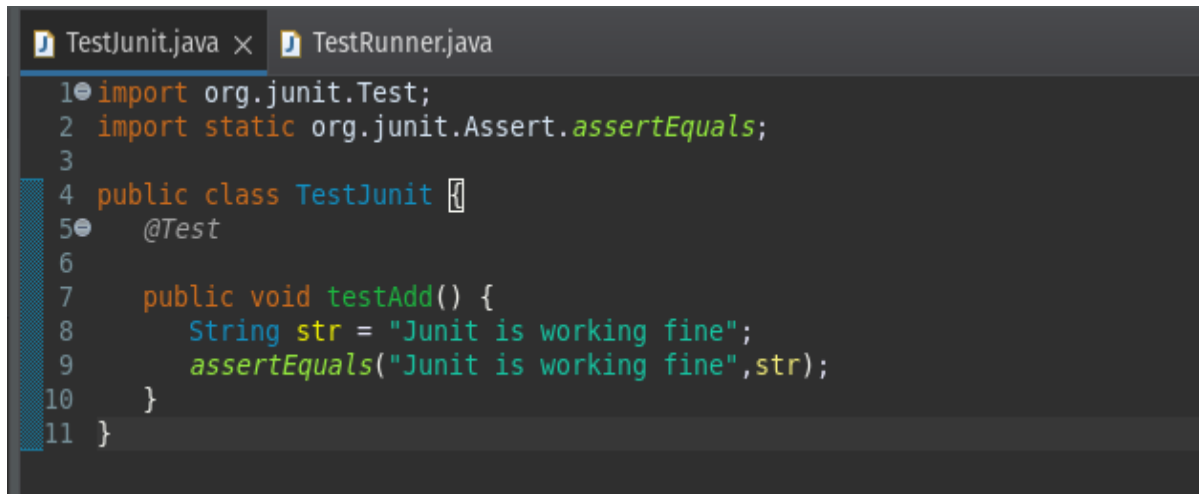
4. Set CLASSPATH Variable

The CLASSPATH environment variable is set to point to the JUNIT jar location.

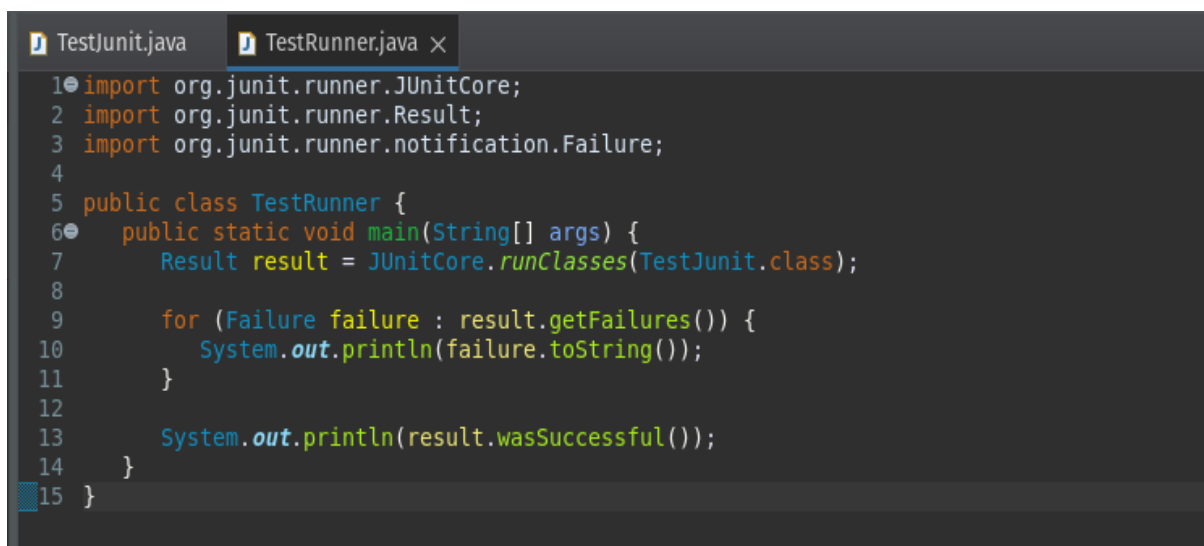
```
sabinthapa @ pop-os ~/Softwares export CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit4.10.jar:.
```

5. JUnit Setup Test

The JUnit installation is then tested by writing the test program as follows.



```
TestJUnit.java x TestRunner.java
1 import org.junit.Test;
2 import static org.junit.Assert.assertEquals;
3
4 public class TestJUnit {
5     @Test
6
7     public void testAdd() {
8         String str = "JUnit is working fine";
9         assertEquals("JUnit is working fine",str);
10    }
11 }
```



```
TestJUnit.java TestRunner.java x
1 import org.junit.runner.JUnitCore;
2 import org.junit.runner.Result;
3 import org.junit.runner.notification.Failure;
4
5 public class TestRunner {
6     public static void main(String[] args) {
7         Result result = JUnitCore.runClasses(TestJUnit.class);
8
9         for (Failure failure : result.getFailures()) {
10             System.out.println(failure.toString());
11         }
12
13         System.out.println(result.wasSuccessful());
14     }
15 }
```

Fig 2: Simple Java programs to test JUnit installation

We've created a Test class called TestJUnit and a runner class called TestRunner. Upon successful testing, 'true' is printed to the console. This proves that JUnit is working fine. The test class consists of a simple class that matches a string with the expected string. Since, both the strings are the same to the assertEquals API, the output should be true.

Output

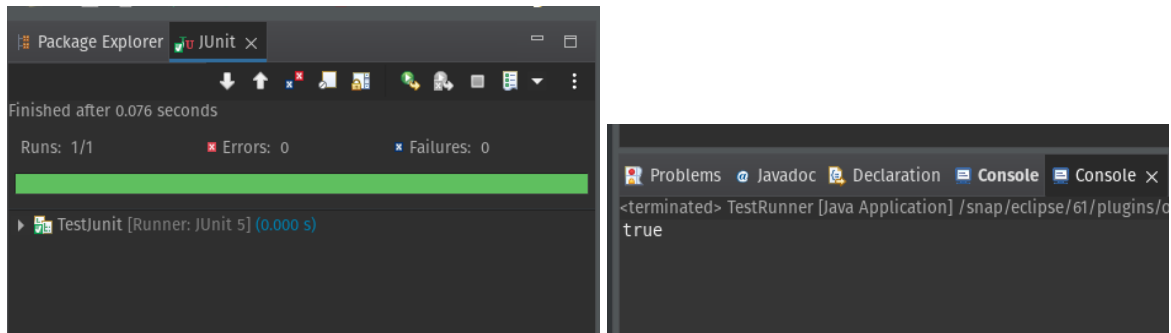
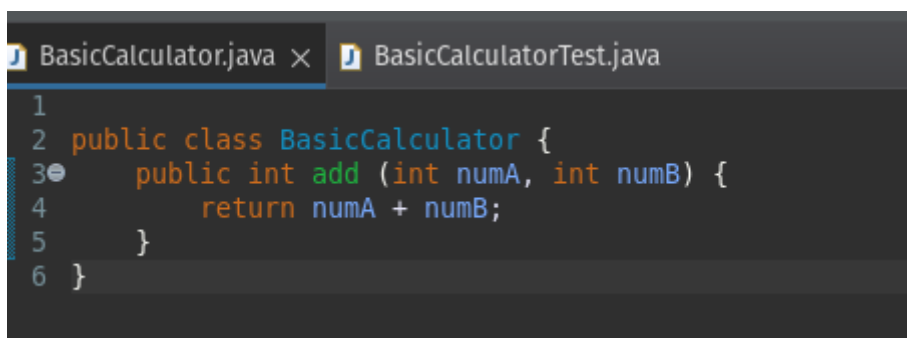


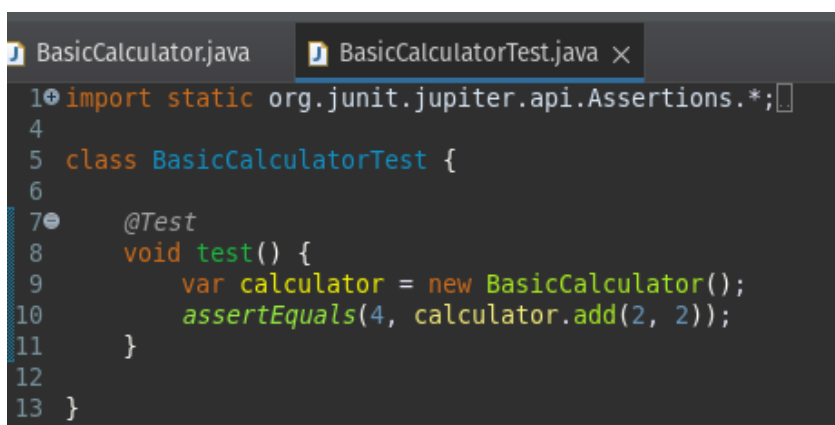
Fig 3: Output of the test program

Demonstration

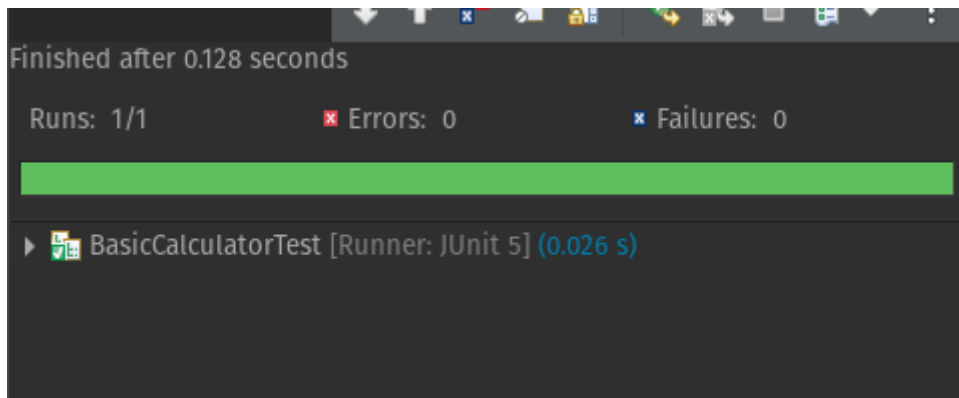
For the demonstration of JUnit, let's create a basic calculator class that adds the two given numbers.



Now, let's write the test case for this class in a new file named BasicCalculatorTest.java. To create a test class, we first create the file and add a test method to the test class. Here, the method is test(). The annotation **@Test** should be added to the method. The test condition is then implemented and checked using assertEquals API of JUnit.



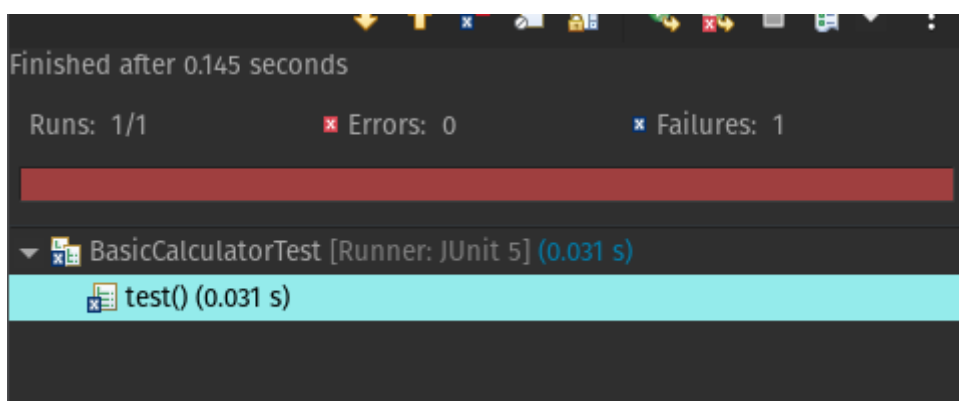
Here, we've used the **assertEquals** API of JUnit to check whether the expected output and the computed output from the class is the same. The output of running this test is shown below:



Here, the test passes successfully because two plus two is indeed four. Let us give a faulty expected output this time.

```
class BasicCalculatorTest {  
    @Test  
    void test() {  
        var calculator = new BasicCalculator();  
        assertEquals(5, calculator.add(2, 2));  
        assertEquals(4, calculator.add(20, 2));  
    }  
}
```

The output shows failures as the sum of 2 and 2 is not 5.



Now, let's understand how JUnit works with a more comprehensive example. Consider the class 'Grade' as shown below.



```
1 public class Grade {
2     private final int points;
3
4     public int getPoints() {
5         return points;
6     }
7
8     public Grade(int p) throws IllegalArgumentException {
9         if(p<1 || p>20)
10             throw new IllegalArgumentException();
11         points = p;
12     }
13 }
14
```

The constructor of the class 'Grade' throws an IllegalArgumentException when the value for points is out of range(1-20). Creating an instance of the class Grade is tedious this way, so we create a method to create an instance of Grade from percentage. This method takes in a percentage value and returns an instance of the Grade class.

```
Grade.java x GradeTestCoverage.java
1 public class Grade {
2     private final int points;
3
4     public int getPoints() {
5         return points;
6     }
7
8     public Grade(int p) throws IllegalArgumentException {
9         if(p<1 || p>20)
10            throw new IllegalArgumentException();
11         points = p;
12     }
13
14     public static Grade fromPercentage(int g) throws IllegalArgumentException {
15         if(g >= 0 && g <= 100) {
16             if(g >= 79) return new Grade(1);
17             else if(g >= 76 && g <= 78) return new Grade(2);
18             else if (g >= 73 && g <= 75) return new Grade(3);
19             else if (g >= 70 && g <= 72) return new Grade(4);
20             else if (g >= 67 && g <= 69) return new Grade(5);
21             else if (g >= 65 && g <= 66) return new Grade(6);
22             else if (g >= 62 && g <= 64) return new Grade(7);
23             else if (g >= 60 && g <= 61) return new Grade(8);
24             else if (g >= 57 && g <= 59) return new Grade(9);
25             else if (g >= 55 && g <= 56) return new Grade(10);
26             else if (g >= 52 && g <= 54) return new Grade(11);
27             else if (g >= 50 && g <= 51) return new Grade(12);
28             else if (g >= 47 && g <= 49) return new Grade(13);
29             else if (g >= 45 && g <= 46) return new Grade(14);
30             else if (g >= 42 && g <= 44) return new Grade(15);
31             else if (g >= 40 && g <= 41) return new Grade(16);
32             else if (g >= 35 && g <= 39) return new Grade(17);
33             else if (g >= 30 && g <= 34) return new Grade(18);
34             else return new Grade(19);
35         } else if (g == -1) return new Grade(20);
36         throw new IllegalArgumentException();
37     }
38 }
39
```

We've created a class GradeTestCoverage to test the two methods we created on the class Grade. We created a test method to test the constructor first. To denote a method is a test method, we annotate it with “@Test” before defining the function.

We mostly use assertEquals() and assertThrows() for testing with J-Unit.

```

Grade.java  GradeTestCoverage.java X
1 import static org.junit.jupiter.api.Assertions.*;
4
5 class GradeTestCoverage {
6
7     @Test
8     void testConstructor() {
9         assertThrows(IllegalArgumentException.class, () -> new Grade(-1));
10        assertThrows(IllegalArgumentException.class, () -> new Grade(21));
11    }
12
13    @Test
14    void testFromPercentage() {
15        assertThrows(IllegalArgumentException.class, () -> Grade.fromPercentage(-2));
16        assertThrows(IllegalArgumentException.class, () -> Grade.fromPercentage(101));
17
18
19
20        assertEquals(1, Grade.fromPercentage(79).getPoints());
21        assertEquals(2, Grade.fromPercentage(76).getPoints());
22        assertEquals(3, Grade.fromPercentage(73).getPoints());
23        assertEquals(4, Grade.fromPercentage(70).getPoints());
24        assertEquals(5, Grade.fromPercentage(67).getPoints());
25        assertEquals(6, Grade.fromPercentage(65).getPoints());
26        assertEquals(7, Grade.fromPercentage(62).getPoints());
27        assertEquals(8, Grade.fromPercentage(60).getPoints());
28        assertEquals(9, Grade.fromPercentage(57).getPoints());
29        assertEquals(10, Grade.fromPercentage(55).getPoints());
30        assertEquals(11, Grade.fromPercentage(52).getPoints());
31        assertEquals(12, Grade.fromPercentage(50).getPoints());
32        assertEquals(13, Grade.fromPercentage(47).getPoints());
33        assertEquals(14, Grade.fromPercentage(45).getPoints());
34        assertEquals(15, Grade.fromPercentage(42).getPoints());
35        assertEquals(16, Grade.fromPercentage(40).getPoints());
36        assertEquals(17, Grade.fromPercentage(35).getPoints());
37        assertEquals(18, Grade.fromPercentage(30).getPoints());
38        assertEquals(19, Grade.fromPercentage(0).getPoints());
39    }
40 }
41

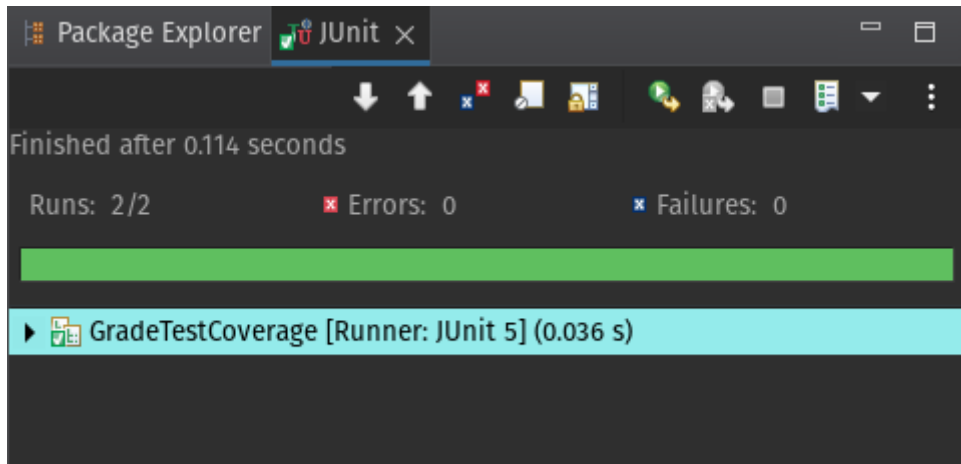
```

The method `assertThrows()` takes 2 parameters as shown in the examples above. The first parameter is the type of an exception. The second parameter is the code that is supposed to throw such an exception. The code that is supposed to throw exceptions must be wrapped within lambda expressions. Lambda expressions are basically like arrow functions in JavaScript.

The method `assertEquals()` also takes 2 parameters as shown in the examples above. The second parameter is the code that returns some value, and the first parameter is the value that is supposed to be returned by the second parameter.

On completion of all test cases, J-Unit shows a colored bar. Green colored bar means all the

tests have passed and the code worked as expected. Red bar means some or all of the tests have failed. On failure of tests, the descriptions of tests that failed are provided just below the bar. When we ran the GradeTest, JUnit showed a green bar, meaning all of the test cases that we wrote have passed.



Conclusion

In this way, JUnit is used for unit testing for the Java programming language. It has been very important in the development of the test-driven development. With the idea of “first testing then coding”, emphasis is given on setting up the test data for a piece of code that can be tested first then implemented.

To sum up, JUnit provides static methods to test for certain conditions via the Assert class and these assert statements typically start with assert. They allow us to specify the expected and the actual result along with the error message. It provides a standard way to write, organize and run tests.