



Tutorial Using JUnit

In this tutorial, we will be developing a student object with the following requirements:

- The student object will have a name field that will be a String.
- The student object will have an address that will be a String.
- The student object will have a student ID that is a String and cannot be null, and the String shall be less than five characters.

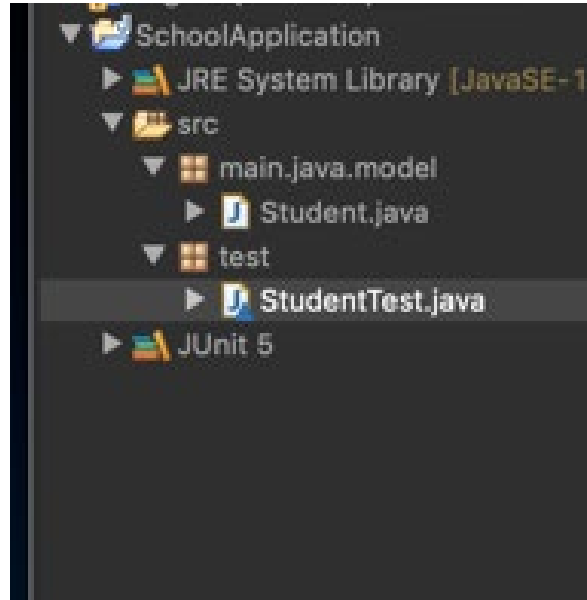
Let's Look at the Student Class

The student class takes in three arguments in the constructor. It takes the name, the address, and the student ID. Please also note that there is a conditional check in the constructor to make sure that the student ID is not null or is no longer than five characters; otherwise, we throw an exception that the input is invalid. This check is done to ensure that the following requirement is being met: "The student object will have a student ID that is a String and cannot be null, and shall be less than five characters."

```
Student.java x
1 package main.java.model;
2
3 public class Student {
4
5     private String name;
6     private String address;
7     private String studentId;
8
9     //The requirement for student ID is that it cannot be null and it cannot be longer than 5 characters or through an exception
10    public Student(String name, String address, String studentId) {
11        if(studentId == null || studentId.length() > 5) {
12            //We will want to test for the exception
13            throw new IllegalArgumentException("Invalid input");
14        }
15        this.name = name;
16        this.address = address;
17        this.studentId = studentId;
18    }
19
20    public String getName() {
21        return name;
22    }
23
24    public String getAddress() {
25        return address;
26    }
27
28    public String getStudentId() {
29        return studentId;
30    }
31
32 }
33
```

Next, Let's Look at Writing the Test in JUnit

Before we begin, we need to create a new package in which our tests can exist. Therefore, first, under the "src" folder in our Java project, we create a new package named "test." Second, we right-click into the new package and select "New" and then select "JUnit test case."



Now we are ready to start to develop our tests. In this tutorial, we will be creating two test points. The first one will cover the success case. In the success case, we want to ensure that we can create the student object successfully and have confidence that the software meets the following requirements:

- The student object will have a name field that will be a String.
- The student object will have an address that will be a String.
- The student object will have a student ID that is a String and cannot be null, and the String shall be less than five characters.

Typically, for JUnit test cases, we want to start each test point with the word “test”:

- In the first test point, we have chosen the name “testStudent.” In this test point, we are creating a student object with a name, address, and student ID. The next step is to use assertions to test that the data we provided in the constructor is being returned from the get methods.
- In the second test point named, “testStudentIDTooLong,” we are testing that the student constructor will throw an exception if the ID is too long. We test this by using the following pattern:

```
Assertions.assertThrows(IllegalArgumentException.class, () -> {
    new Student("Jeff", "1111 E Road", "123456");
});
```

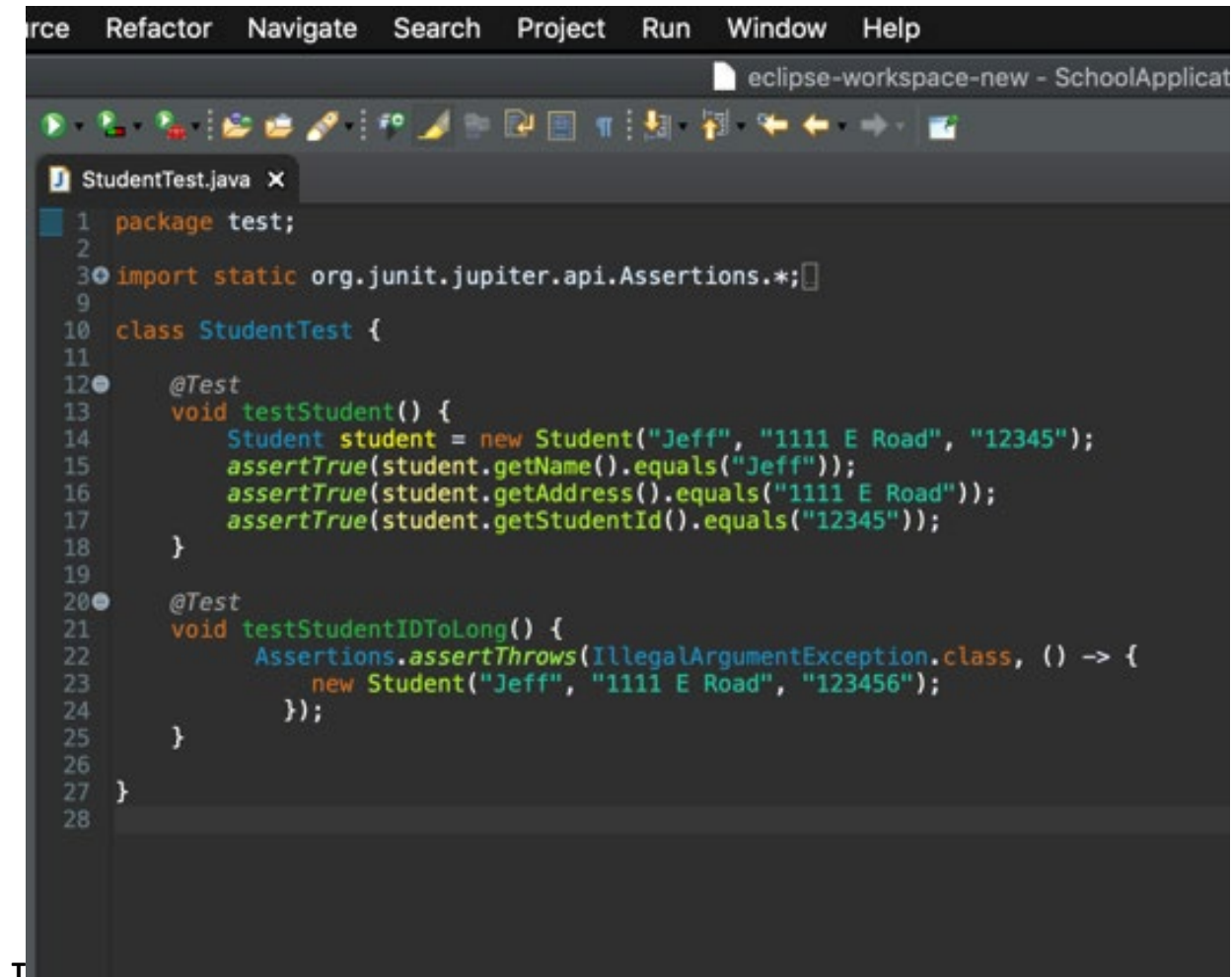
This assertion checks to see if an illegal exception is being thrown. There are two parts to this assertion:

1. First, we need to identify the exception that is expected to be thrown

```
Assertions.assertThrows(IllegalArgumentException.class, () ->
```

2. Second, we need to provide the behavior that is expected to cause the exception. In this example, we expect that an exception will be thrown when we create a new student object with an ID that is longer than five characters:

```
new Student("Jeff", "1111 E Road", "123456");
```



```

1 package test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class StudentTest {
6
7     @Test
8     void testStudent() {
9         Student student = new Student("Jeff", "1111 E Road", "12345");
10        assertTrue(student.getName().equals("Jeff"));
11        assertTrue(student.getAddress().equals("1111 E Road"));
12        assertTrue(student.getStudentId().equals("12345"));
13    }
14
15    @Test
16    void testStudentIDToLong() {
17        Assertions.assertThrows(IllegalArgumentException.class, () -> {
18            new Student("Jeff", "1111 E Road", "123456");
19        });
20    }
21
22 }

```

Note: We can reuse this approach when creating our mobile application to ensure that requirements are being met:

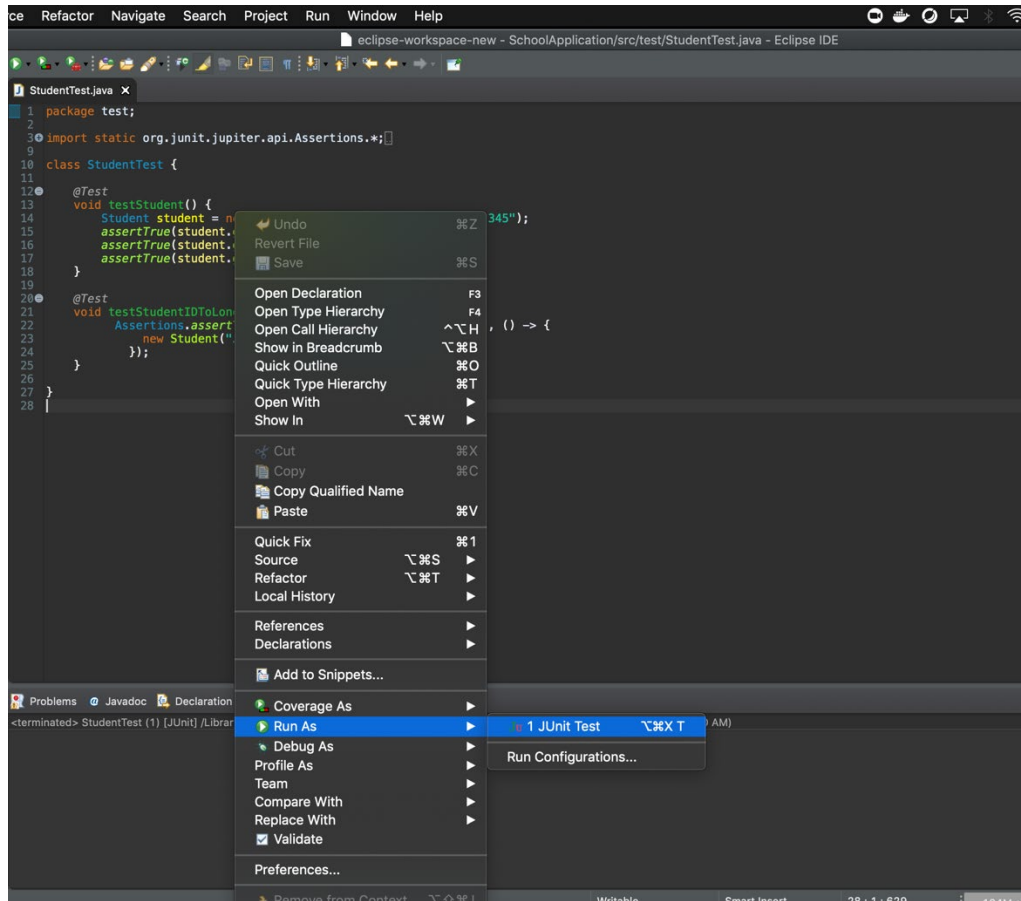
```

Assertions.assertThrows("The expected Exception", () -> {
    "The class constructor creating the exception";
});

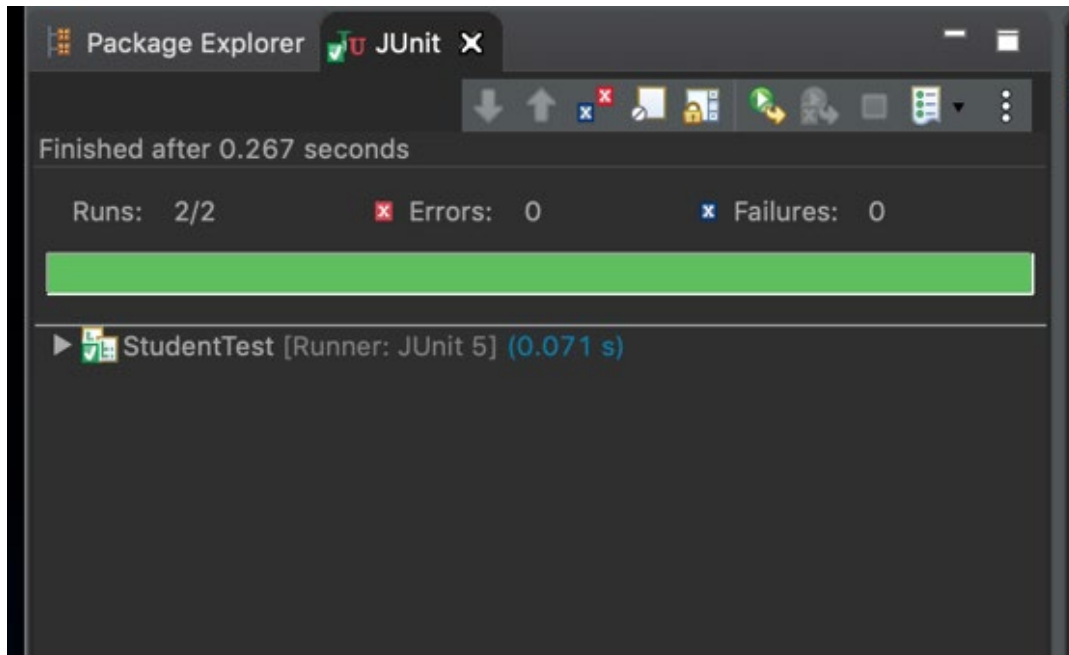
```

How Do I Run My Tests?

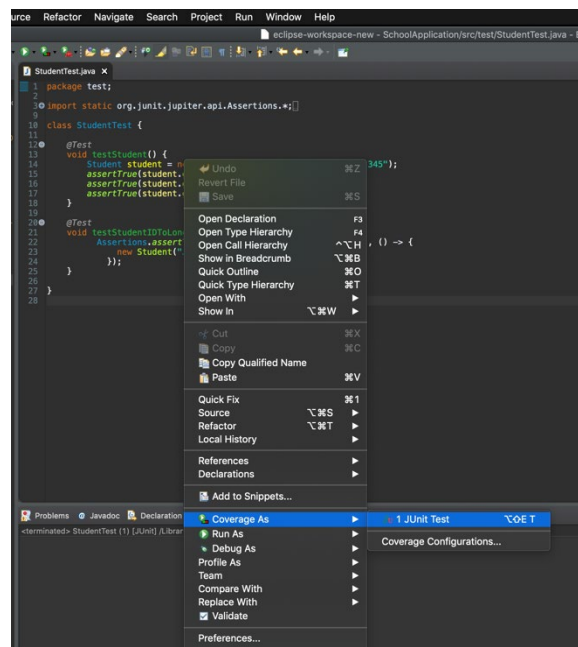
The next step is to run the tests to make sure our tests are passing. To run the JUnit tests, right-click and select “Run As.” Then, select “JUnit Test” from the menu.



Below, we can see the results of the JUnit tests being executed. The green bar means that all test points executed successfully. “Runs:2/2” describes the number of tests pointed, executed, and successfully passed.



We can also run the tests with “Coverage As.” This option will show us the test coverage percentage. We should strive for the highest percentage we can get to ensure that the tests provide sufficient coverage for the object they are testing.



In the next example, we should make note of two things. First, the coverage option also runs the JUnit test as in the previous example. Second, we have a new tab below named “Coverage.” If we expand it, we can see that the StudentTest class has tested the Student object at 100% coverage. This means that our tests have sufficiently tested the Student object. Test coverage is based on the following: all the



public application programming interfaces in the software we are testing and on conditional flows. This means, Do our tests all pass through conditional statements?

The screenshot shows an IDE with a JUnit test run and a coverage report. The test run summary indicates that the tests passed successfully.

Test Run Summary:

- Finished after 0.267 seconds
- Runs: 2/2
- Errors: 0
- Failures: 0

Test Results:

- StudentTest [Runner: JUnit 5] (0.071 s)

Code Snippet (StudentTest.java):

```
1 package test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class StudentTest {
6
7     @Test
8     void testStudent() {
9         Student student = new Student("Jeff", "1111 E Road", "12345");
10        assertTrue(student.getName().equals("Jeff"));
11        assertTrue(student.getAddress().equals("1111 E Road"));
12        assertTrue(student.getStudentId().equals("12345"));
13    }
14
15    @Test
16    void testStudentIDTooLong() {
17        Assertions.assertThrows(IllegalArgumentException.class, () -> {
18            new Student("Jeff", "1111 E Road", "123456");
19        });
20    }
21 }
22
```

Coverage Report:

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
SchoolApplication	91.3 %	63	6	69
src	91.3 %	63	6	69
test	83.8 %	31	6	37
main.java.model	100.0 %	32	0	32
Student.java	100.0 %	32	0	32