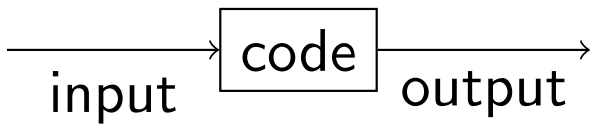
Testing on Steriods

EECS 4315

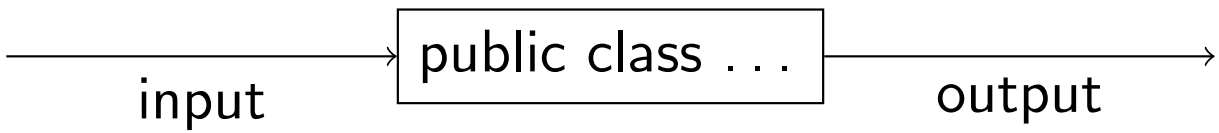
wiki.eecs.yorku.ca/course/4315/

How to test code?



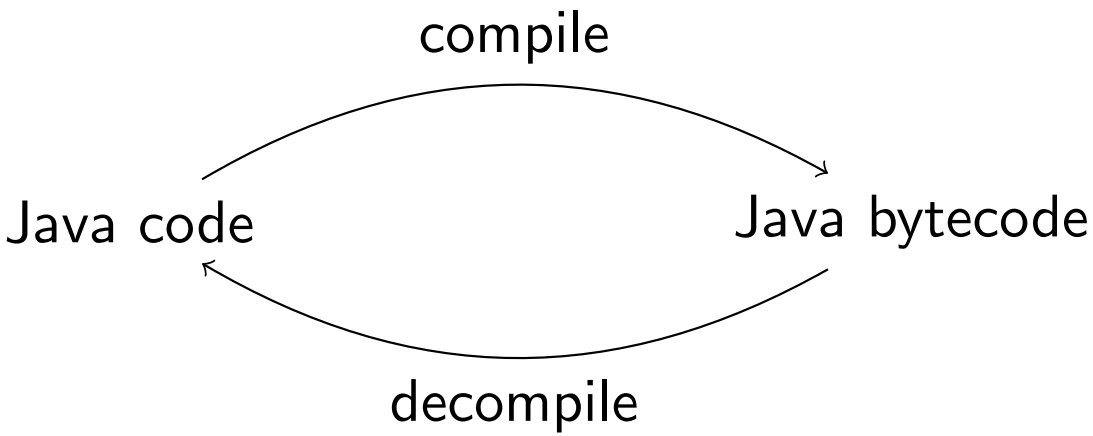
- Provide the input.
- Run the code.
- Compare the output with the expected output.

White box testing



Black box testing

Java code and Java bytecode



Why black box testing?

A Java archive (JAR) file usually only contains the bytecode and not the Java code.

Developers can obfuscate JAR files so that a user of the JAR file does not get much information regarding the original Java code.

Which test cases?

- Likely cases (black box and white box testing).
- Boundary cases (black box and white box testing).
- Cases that cover all branches (white box testing only).
- Cases that cover all execution paths (white box testing only).

Unit testing

A unit test is designed to test a single unit of code, for example, a method.

Such a test should be automated as much as possible; ideally, it should require no human interaction in order to run, should assess its own results, and notify the programmer only when it fails.

A class that contains unit tests is known as a test case.

The code to be tested is known as the unit under test.

JUnit

JUnit is a Java unit testing framework developed by Kent Beck and Erich Gamma.

JUnit is available at http://junit.org/junit5/.

Kent Beck

Kent Beck is an American software engineer and the creator of the Extreme Programming and Test Driven Development software development. He works at Facebook.

source: Three Rivers Institute

Erich Gamma

Erich Gamma is a Swiss computer scientist and member of the "Gang of Four" who wrote the influential software engineering textbook "Design Patterns: Elements of Reusable Object-Oriented Software." He works at Microsoft.

source: Pearson

Java annotations

Annotations provide data about code that is not part of the code itself. Therefore, it is also called metadata.

In its simplest form, an annotation looks like

```
@Deprecated
```

(The annotation type `Deprecated` is part of `java.lang` and,therefore, need not be imported.)

JUnit contains annotations such as

```
@Test
```

(The annotation type `Test` is part of `org.junit.jupiter.api` and, therefore, needs to be imported.)

An annotation can include elements and their values:

```
@EnabledIfSystemProperty(named="os.arch", matches=".*64.*")
```

(The annotation type `EnabledIfSystemProperty` is part of `org.junit.jupiter.api.condition`.)

A test case

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class ... {
   @Test
   public void ...() {
      ...
   }

   @Test
   public void ...() {
      ...
   }
}
```

Names of test methods

It is good practice to use descriptive names for the test methods.
This makes tests more readable when they are looked at later.

Assertions in test methods

Each test method should contain (at least) one assertion: an invocation of a method of the Assertions class of the `org.junit.jupiter.api` package.

Do not confuse these assertions with Java's `assert` statement.

Methods of the Assert class

`assertEquals(long, long)`

assert that the two are the same.

`assertEquals(long, long, String)`

assert that the two are the same; if not, the message is used.

Methods of the Assert class

```
assertEquals(double, double, double)
assertEquals(double, double, double, String)
```

The method invocation

```
Assert.assertEquals(expectedValue, actualValue, delta)
```

asserts

$$|\texttt{expectedValue} - \texttt{actualValue}| < \texttt{delta}$$

Methods of the Assert class

`assertEquals(Object, Object)`
`assertEquals(Object, Object, String)`

asserts that the objects are equal according to the `equals` method.

`assertSame(Object, Object)`
`assertSame(Object, Object, String)`

asserts that the objects are equal according to the `==` operator.

Methods of the Assert class

`assertTrue(boolean)`
`assertTrue(boolean, String)`

asserts that the condition is true.

`assertFalse(boolean)`
`assertFalse(boolean, String)`

asserts that the condition is false.

Methods of the Assert class

`assertNull(Object)`
`assertNull(Object, String)`

asserts that the object is null.

`assertNotNull(Object)`
`assertNotNull(Object, String)`

asserts that the object is not null.

Timeout

Cause a test to fail if it takes longer than a specified time in milliseconds:

```
@Test
public void ...() {
  Assertions.assertTimeout(ofMillis(1000), () -> {
    ...
  });
}
```

Exceptions

Cause a test to fail if a specified exception is not thrown:

```
@Test
public void ... () {
  Assertions.assertThrows(NumberFormatException.class, () -
    ...
  });
}
```

Body of unit test method

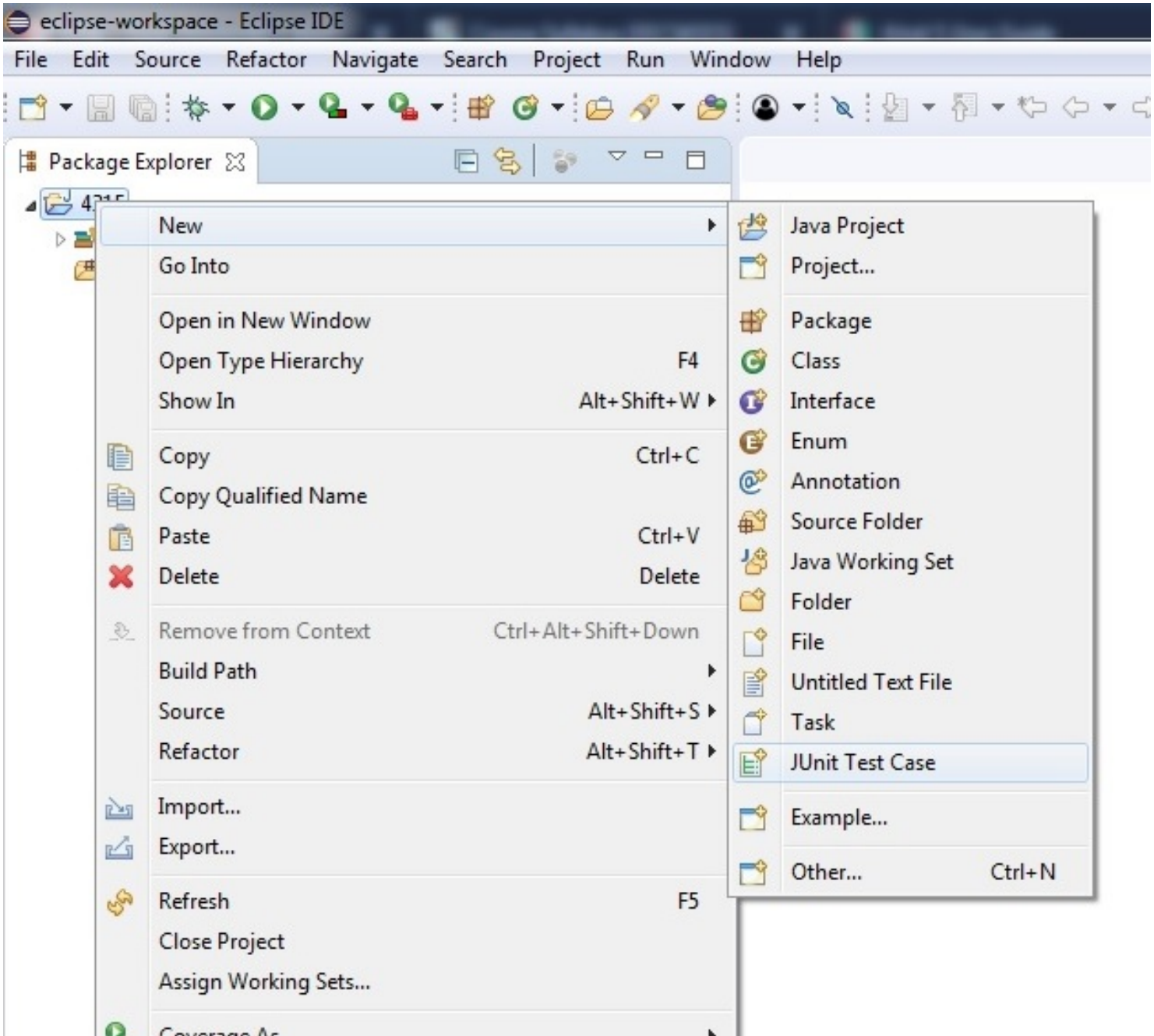
1. Create some objects.
2. Invoke methods on them.
3. Check the results using a method of the `Assertions` class.

Test case

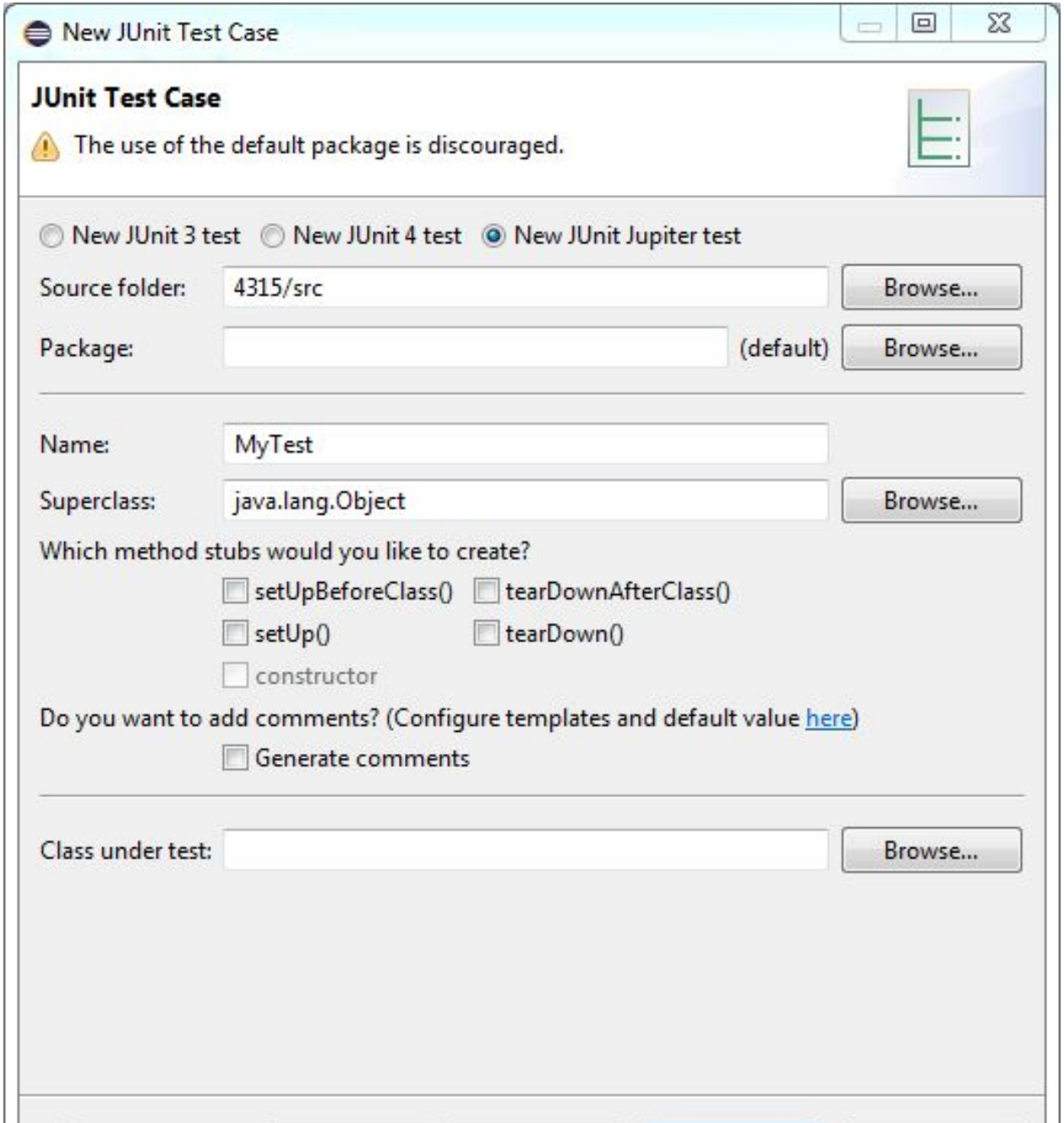
For each method and constructor (from simplest to most complex)

1. Study its API.
2. Create unit tests.

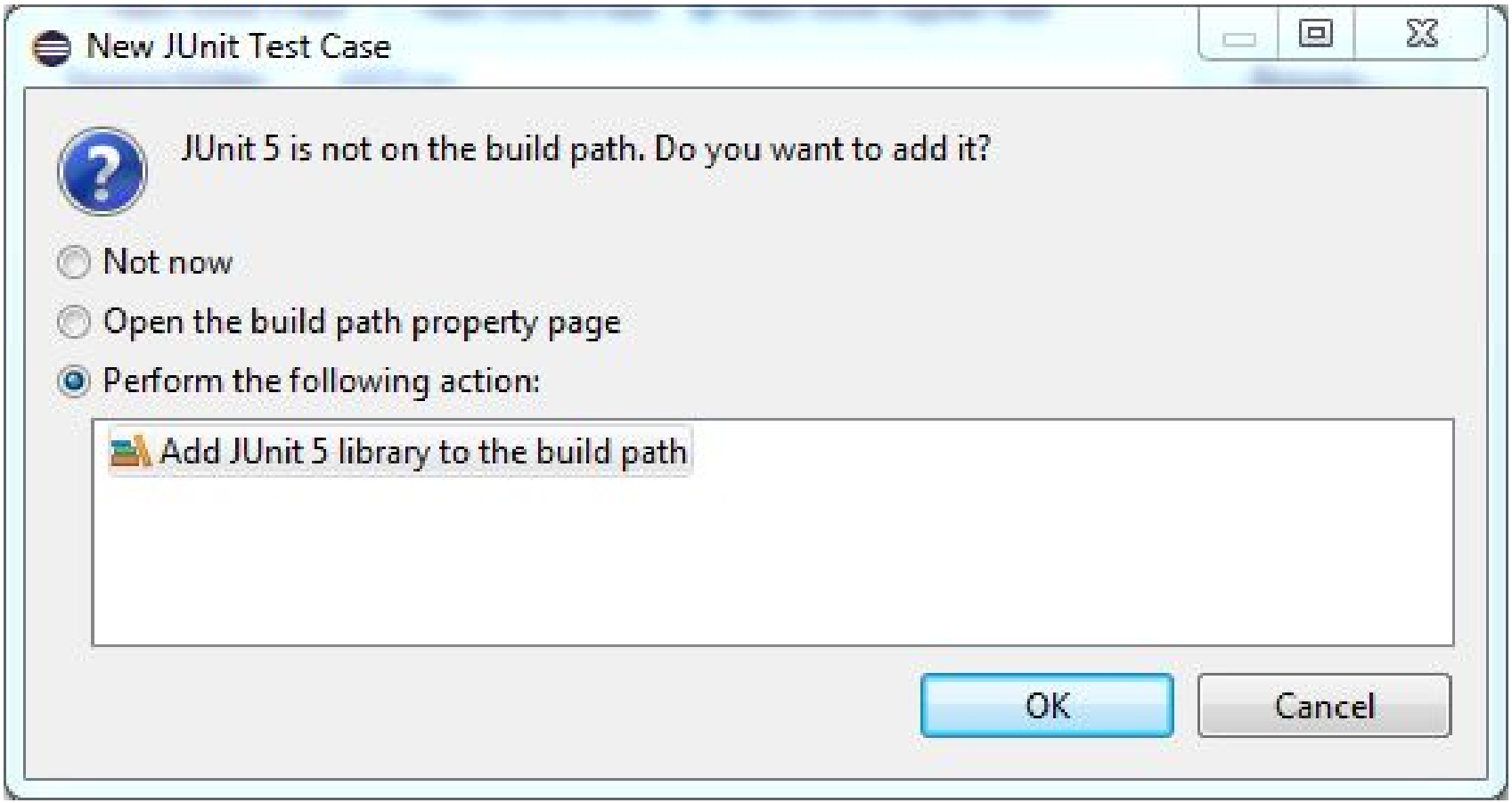
Creating a JUnit test case in eclipse



Creating a JUnit test case in eclipse



Creating a JUnit test case in eclipse

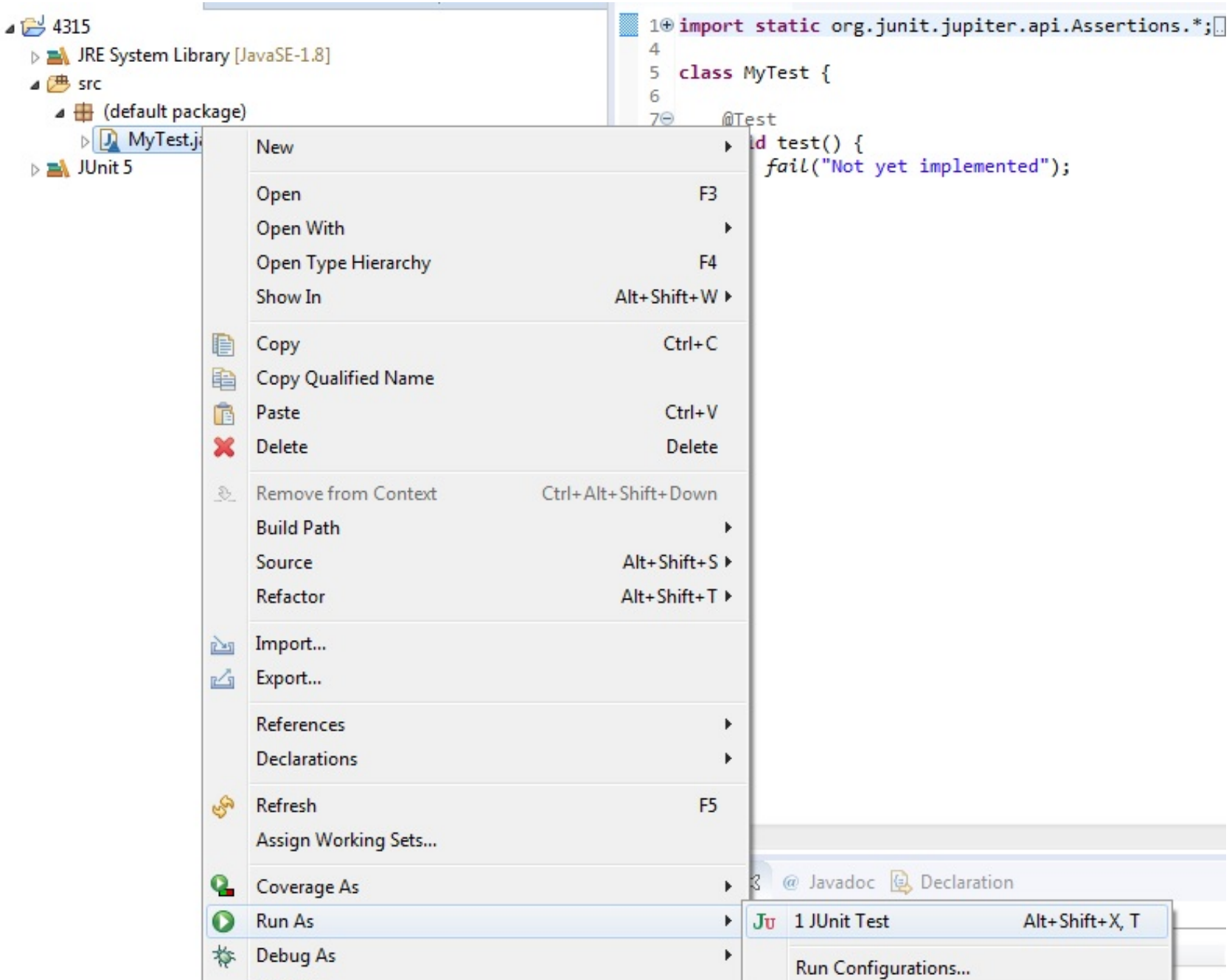


Creating a JUnit test case in eclipse

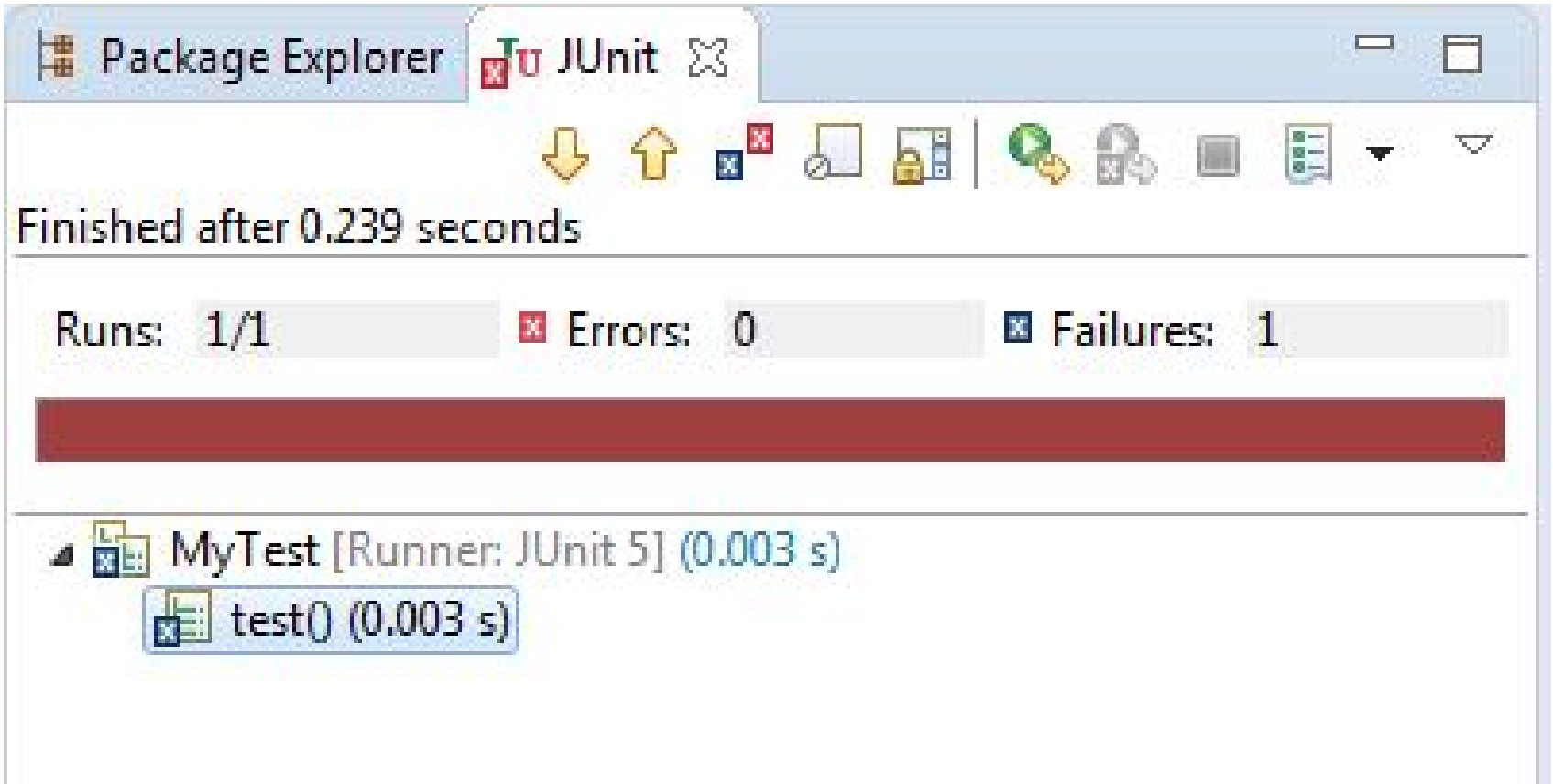
MyTest.java

```
1⊕ import static org.junit.jupiter.api.Assertions.*;
4
5  class MyTest {
6
7⊖     @Test
8      void test() {
9          fail("Not yet implemented");
10     }
11
12 }
13
```

Running a JUnit test case in eclipse



Running a JUnit test case in eclipse



Running a JUnit test case in eclipse

*MyTest.java

```
1  import static org.junit.jupiter.api.Assertions.*;
4
5  class MyTest {
6
7      @Test
8      void test() {
9          //fail("Not yet implemented");
10     }
11
12 }
13
```

Running a JUnit test case in eclipse