Testing on Steriods

EECS 4315

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

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/`.

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.

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.

Lab 1

Write a JUnit test case to test the class `Color`, whose API can be found here.

Test the constructor

Question

What can we test about the constructor?

Test the constructor

Question

What can we test about the constructor?

Answer

That the created object is not null.

Test the constructor

Question

What can we test about the constructor?

Answer

That the created object is not null.

Question

How many “inputs” does the constructor have?

Test the constructor

Question

What can we test about the constructor?

Answer

That the created object is not null.

Question

How many “inputs” does the constructor have?

Answer

Three.

Test the constructor

Question

How many combinations of “inputs” for the constructor are there?

Test the constructor

Question

How many combinations of “inputs” for the constructor are there?

Answer

$256 \times 256 \times 256 = 16777216 \approx 10^7$.

Test the constructor

Question

How many combinations of “inputs” for the constructor are there?

Answer

$256 \times 256 \times 256 = 16777216 \approx 10^7$.

Question

Can we check all these combinations of “inputs”?

Test the constructor

Question

How many combinations of “inputs” for the constructor are there?

Answer

$256 \times 256 \times 256 = 16777216 \approx 10^7$.

Question

Can we check all these combinations of “inputs”?

Answer

Yes.

Test the acccessors

Question

What can we test about the accessors?

Test the acccessors

Question

What can we test about the accessors?

Answer

That they return the correct values.

Test the constant BLACK

Question

What can we test about the constant `Color.BLACK`?

Test the constant BLACK

Question

What can we test about the constant `Color.BLACK`?

Answer

That it is not null.

Test the constant BLACK

Question

What can we test about the constant `Color.BLACK`?

Answer

That it is not null.

Question

Should we test that the three accessors return 0 for the constant `Color.BLACK`?

Test the constant BLACK

Question

What can we test about the constant `Color.BLACK`?

Answer

That it is not null.

Question

Should we test that the three accessors return 0 for the constant `Color.BLACK`?

Answer

No. This has not been specified in the API.

Test the equals method

Question

What can we test about the `equals` method?

Test the equals method

Question

What can we test about the `equals` method?

Answer

- a `Color` object is equal to itself,
- a `Color` object is equal to a `Color` object with the same RGB values,
- a `Color` object is not equal to a `Color` object with the different RGB values,
- a `Color` object is not equal to `null`, and
- a `Color` object is not equal to an object of another type.

Test the equals method

Question

Can we test that a `Color` object is not equal to a `Color` object with the different RGB values for all possible combinations?

Test the equals method

Question

Can we test that a `Color` object is not equal to a `Color` object with the different RGB values for all possible combinations?

Answer

There are $256 \times 256 \times 256 \times 256 \times 256 \times 256 \approx 10^{14}$ and, hence, no.

Test the equals method

Question

Can we test that a `Color` object is not equal to a `Color` object with the different RGB values for all possible combinations?

Answer

There are $256 \times 256 \times 256 \times 256 \times 256 \times 256 \approx 10^{14}$ and, hence, no.

Question

Which combinations do we check?

Test the equals method

Question

Can we test that a `Color` object is not equal to a `Color` object with the different RGB values for all possible combinations?

Answer

There are $256 \times 256 \times 256 \times 256 \times 256 \times 256 \approx 10^{14}$ and, hence, no.

Question

Which combinations do we check?

Question

Random combinations.

Avoid break statements in loops

```
@Test
public void testConstructor() {
  for (byte i = -128; i < 128; i++) {
    ...
    if (i == 127) {
      break;
    }
  }
}
```

Descriptive variable names

```
Color c = new Color(0, 0, 0);
Color c2 = new Color(0, 0, 0);
```

Question

Are these variable names descriptive?

Descriptive variable names

```
Color c = new Color(0, 0, 0);
Color c2 = new Color(0, 0, 0);
```

Question

Are these variable names descriptive?

Answer

No. They are cryptic variable names. As a result, their meaning might not be clear to others. They are also very similar, which makes it easy to mix them up.

Use constants

Instead of

```
@Test
public void testConstructor() {
  for (byte i = -128; i < 128; i++) {
    ...
  }
}
```

Use constants

Instead of

```
@Test
public void testConstructor() {
  for (byte i = -128; i < 128; i++) {
    ...
  }
}
```

use

```
@Test
public void testConstructor() {
  for (int i = Byte.MIN_VALUE; i <= Byte.MAX_VALUE; i++) {
    ...
  }
}
```

Global variables

Question

What is the scope of the attribute in the following code snippet?

```
public class ColorTest {
  private Color color;

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

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

  ...
}
```

Global variables

Question

What is the scope of the attribute in the following code snippet?

```
public class ColorTest {
  private Color color;

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

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

  ...
}
```

Answer

The whole class. If possible, try to limit the scope.

Document your code

```
/**
 * Tests the constructor for all combinations
 * of RGB values.
 */
public void testConstructor() {
  ..
}
```

Exercise

Write a JUnit test case to test the class `Boolean`, whose API can be found here.

Test the constructor

Question

What can we test about the constructor?

Test the constructor

Question

What can we test about the constructor?

Answer

That the created object is not null.

Test the booleanValue method

Question

What can we test about the booleanValue method?

Test the booleanValue method

Question

What can we test about the booleanValue method?

Answer

Check if it returns the correct value.

Test the constant TRUE

Question

What can we test about the constant TRUE?

Test the constant TRUE

Question

What can we test about the constant TRUE?

Answer

Check if it is not null and has the correct value.

Test the compareTo method

Question

What can we test about the compareTo method?

Test the compareTo method

Question

What can we test about the compareTo method?

Answer

1. Check if it returns a correct value.
2. Check if it throws an `IllegalArgumentException` if the argument is `null`.

Test the compareTo method

Question

How many “inputs” does the `compareTo` method have?

Test the compareTo method

Question

How many “inputs” does the `compareTo` method have?

Answer

Two: `one.compareTo(two)`

Test the compareTo method

Question

How many “inputs” does the `compareTo` method have?

Answer

Two: `one.compareTo(two)`

Question

How many combinations of “inputs” for the `compareTo` method do we have to check?

Test the compareTo method

Question

How many “inputs” does the `compareTo` method have?

Answer

Two: `one.compareTo(two)`

Question

How many combinations of “inputs” for the `compareTo` method do we have to check?

Answer

Four.

Test the compareTo method

```
@Test
public void testCompareTo() {
  Boolean FALSE = new Boolean(false);
  ...
  ... Boolean.TRUE.compareTo(FALSE) ...
```

Question

Should we check if the result is 1?

Test the compareTo method

```
@Test
public void testCompareTo() {
  Boolean FALSE = new Boolean(false);
  ...
  ... Boolean.TRUE.compareTo(FALSE) ...
```

Question

Should we check if the result is 1?

Answer

No, we should check if the result is greater than zero.

Test the compareTo method

Question

How many “inputs” does `compareTo(null)` have?

Test the compareTo method

Question

How many “inputs” does `compareTo(null)` have?

Answer

One.

Test the compareTo method

Question

How many “inputs” does `compareTo(null)` have?

Answer

One.

Question

How many combinations of “inputs” for `compareTo(null)` do we have to check?

Test the compareTo method

Question

How many “inputs” does `compareTo(null)` have?

Answer

One.

Question

How many combinations of “inputs” for `compareTo(null)` do we have to check?

Answer

Two.

Test the equals method

Question

Do we have to test the `equals` method?

Test the equals method

Question

Do we have to test the `equals` method?

Answer

No, since it is not part of the API of the `Boolean` class.

Correctness of the JUnit test cases

Question

Should we test the JUnit test cases?

Correctness of the JUnit test cases

Question

Should we test the JUnit test cases?

Answer

Should we test the tests that test the JUnit test cases?

Correctness of the JUnit test cases

Question

Should we test the JUnit test cases?

Answer

Should we test the tests that test the JUnit test cases?

We may find bugs in our tests when a test case fails and we inspect our code and the test case. When evaluating test cases, we are often interested in coverage (code, path).

Software Engineering Testing (EECS 4313)

Test the Color class

Question

If we run the JUnit test case `ColorTest` and all tests pass, can we conclude that the class `Color` correctly implements the API?

Test the Color class

Question

If we run the JUnit test case `ColorTest` and all tests pass, can we conclude that the class `Color` correctly implements the API?

Answer

No.

Test the Color class

Question

If we run the JUnit test case `ColorTest` and all tests pass, can we conclude that the class `Color` correctly implements the API?

Answer

No.

Question

Why not?

Test the Color class

Question

If we run the JUnit test case `ColorTest` and all tests pass, can we conclude that the class `Color` correctly implements the API?

Answer

No.

Question

Why not?

Answer

Run the JUnit test case `ColorTest` several times.

Test the Color class

Question

How is it possible that the JUnit test case `ColorTest` passes all tests pass in some runs and fails the method `testConstructorAndAccessors` in other runs?

Test the Color class

Question

How is it possible that the JUnit test case `ColorTest` passes all tests pass in some runs and fails the method `testConstructorAndAccessors` in other runs?

Answer

Let's have a look at the code of `testConstructorAndAccessors`.

Test the Color class

Question

How is it possible that the JUnit test case `ColorTest` passes all tests pass in some runs and fails the method `testConstructorAndAccessors` in other runs?

Answer

Let's have a look at the code of `testConstructorAndAccessors`.

Answer

Because the code of `testConstructorAndAccessors` uses randomization.

Randomization

Question

Why are we interested in randomization in our code?

Randomization

Question

Why are we interested in randomization in our code?

Answer

The source code of most computer and video games contains some sort of randomization. This provides games with the ability to surprise players, which is a key factor to their long-term appeal.

Katie Salen and Eric Zimmerman. *Rules of Play: Game Design Fundamentals*. The MIT Press. 2004.

Randomization

Question

Why are we interested in randomization in our code?

Randomization

Question

Why are we interested in randomization in our code?

Answer

Randomization may reduce the expected running time or memory usage.

Randomization

Question

Why are we interested in randomization in our code?

Answer

Randomization may reduce the expected running time or memory usage.

Question

Which algorithms exploit randomization this way?

Randomization

Question

Why are we interested in randomization in our code?

Answer

Randomization may reduce the expected running time or memory usage.

Question

Which algorithms exploit randomization this way?

Answer

- Randomized quicksort.
- Skiplist.
- ...

Randomization

Question

Why are we interested in randomization in our code?

Randomization

Question

Why are we interested in randomization in our code?

Answer

Randomization may allow us to solve problems.

Randomization

Question

Why are we interested in randomization in our code?

Answer

Randomization may allow us to solve problems.

Question

Which algorithms exploit randomization this way?

Randomization

Question

Why are we interested in randomization in our code?

Answer

Randomization may allow us to solve problems.

Question

Which algorithms exploit randomization this way?

Answer

- Consensus problem (in an asynchronous distributed system in which processes may fail).
- . . .

Nondeterminism

Nondeterministic code is code that, even for the same input, can exhibit different behaviors on different runs, as opposed to deterministic code.

Randomization gives rise to nondeterminism.

Nondeterminism

Nondeterministic code is code that, even for the same input, can exhibit different behaviors on different runs, as opposed to deterministic code.

Randomization gives rise to nondeterminism.

Question

Besides randomization, are there other programming concept that give rise to nondeterminism?

Nondeterminism

Nondeterministic code is code that, even for the same input, can exhibit different behaviors on different runs, as opposed to deterministic code.

Randomization gives rise to nondeterminism.

Question

Besides randomization, are there other programming concept that give rise to nondeterminism?

Answer

Concurrency.

Quiz 1

- When: Friday January 11 during the lab
- Topic: testing