Testing on Steriods

EECS 4315

www.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 written by Kent Beck and Erich Gamma.

JUnit is available at http://junit.org/junit4/.

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.)

An annotation can include elements and their values:

```
@Test(timeout=1000)
```

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

A test case

```
import org.junit.Assert;
import org.junit.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 Assert class of the `org.junit` 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 `Assert` class.

Test case

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

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

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 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 smaller 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.

Exceptions

```
@Test
public void test() {
  try {
    call of constructor or method;
  } catch (Exception e) {
    Assert.fail("Exception was thrown");
  }
}
```

Question

Do we have to test whether each constructor and method does not throw any exceptions?

Exceptions

```
@Test
public void test() {
  try {
    call of constructor or method;
  } catch (Exception e) {
    Assert.fail("Exception was thrown");
  }
}
```

Question

Do we have to test whether each constructor and method does not throw any exceptions?

Answer

No. If a constructor or method throws an exception, the test case will fail and the exception will be reported.

Exceptions

```
@Test
public void test() {
  Boolean value = new Boolean(true);
  Assert.assertNotNull("...", value);
  value = false;
  Assert.assertFalse("...", value);
  value = true;
  Assert.assertTrue("...", value);
}
```

Question

Which class is tested, `java.lang.Boolean` or `lab.Boolean`?

Exceptions

```
@Test
public void test() {
  Boolean value = new Boolean(true);
  Assert.assertNotNull("...", value);
  value = false;
  Assert.assertFalse("...", value);
  value = true;
  Assert.assertTrue("...", value);
}
```

Question

Which class is tested, `java.lang.Boolean` or `lab.Boolean`?

Answer

`java.lang.Boolean`.

Descriptive variable names

```
Boolean bool = new Boolean(true);
Boolean bool2 = new Boolean(true);
Boolean bool3 = new Boolean(false);
Boolean bool4 = new Boolean(false);
```

Question

Are these variable names descriptive?

Descriptive variable names

```
Boolean bool = new Boolean(true);
Boolean bool2 = new Boolean(true);
Boolean bool3 = new Boolean(false);
Boolean bool4 = new Boolean(false);
```

Question

Are these variable names descriptive?

Answer

No.

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? No.

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? No.

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).

Test the Boolean class

Question

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

Test the Boolean class

Question

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

Answer

No.

Test the Boolean class

Question

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

Answer

No.

Question

Why not?

Test the Boolean class

Question

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

Answer

No.

Question

Why not?

Answer

Run the JUnit test case `BooleanTest` several times.

Test the Boolean class

Question

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

Test the Boolean class

Question

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

Answer

Lets have a look at the code of `compareTo`.

Test the Boolean class

Question

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

Answer

Lets have a look at the code of `compareTo`.

Answer

Because the code of `compareTo` 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 12 during the lab
- Topic: testing