A Testing Framework (Part 1/2)

Alexandre Bergel Nancy Hitschfeld

28/09/2020



Goal of today

Having a brief overview of UML class diagram

Expressing requirements using unit test



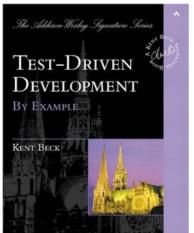
A Testing Framework

Source

JUnit 4.0 documentation (from www.junit.org)

Test-Driven Development, by Kent Beck

Vavigation	Welcome to JUnit.org!	What's New?		
Junit Releases Issue Tracker	This site is dedicated to software developers and testers using the JUnit testing framework. In addition to the latest news and download links for JUnit, you can find answers to questions about automated testing, tools and extensions for JUnit, and articles on the Divino Development and other testing topics.	 jtoc — java best oracle creator TwP 3.0 released Time files with tempus-fugit Creating abstract contract nexts in literit 4 		
Articles Announcements	Our goal is to serve you. We've redesigned the site to better meet this goal. If you have questions or suggestions please contact us.			
Mailing List Web Forums Tools		 Junit 4.8.1 is released more 		
D Submit Content		Syndicate		
		User login		
		Username: •		
		Password: *		
		Create new account		
		 Request new password 		







1.JUnit - a testing framework

1.testing practices

2.frameworks vs. libraries

3.JUnit 3.x vs. JUnit 4.x (annotations)

2.Money and MoneyBag - a testing case study





1.JUnit - a testing framework

1.testing practices

2.frameworks vs. libraries

3.JUnit 3.x vs. JUnit 4.x (annotations)

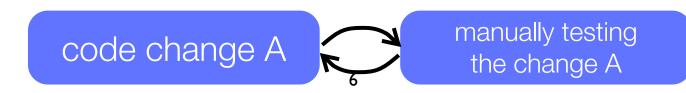
2.Money and MoneyBag - a testing case study



THE Problem

Testing is often (especially by students) done in an ad-hoc manner

With a succession of code increment, and manual testing

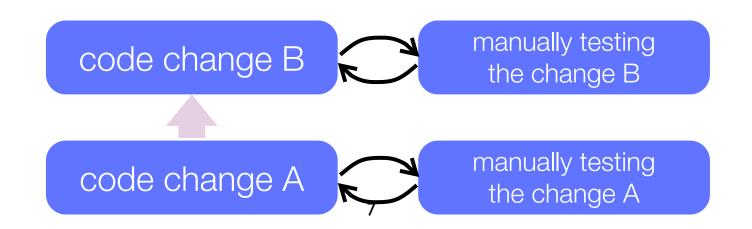




THE Problem

Testing is often (especially by students) done in an ad-hoc manner

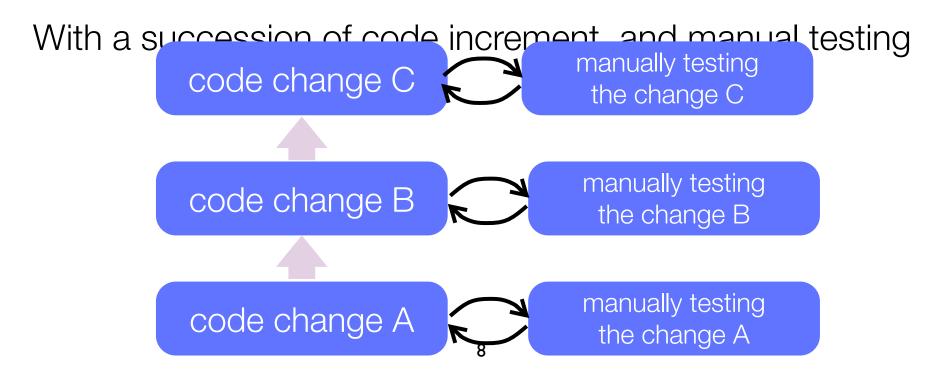
With a succession of code increment, and manual testing





THE Problem

Testing is often (especially by students) done in an ad-hoc manner





3 Testing Practices...

1 - During Development

When you need to add new functionality, write the tests first

You will be done when the test runs

2 - When you need to redesign your software to

add new features, refactor in small steps, and *run the (regression) tests* after each step

Fix what's broken before proceeding.



3 Testing Practices

3 - During Debugging

When someone discovers a defect in your code, *first write a test* that demonstrates the defect

Then debug until the test succeeds

"Whenever you are tempted to type something into a print statement or a debugger expression, write it as a test instead." -- Martin Fowler



JUnit - A Testing Framework

JUnit is a simple *framework* to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks written by Kent Beck and Erich Gamma

For documentation of how to use JUnit:

http://junit.sourceforge.net/doc/cookbook/cookbook.htm







Frameworks vs. Libraries

In traditional application architectures, user code makes use of library functionality in the form of procedures or classes:

User application
 Library classes

A framework *reverses* the usual relationship between generic and application code. Frameworks provide both generic functionality and application architecture:

> Framework main() User classes



Frameworks vs. Libraries

Essentially, a framework says: "Don't call me - I'll call you."



JUnit 3.8...

JUnit is a simple "testing framework" that provides:

- classes for writing Test Cases and Test Suites
- methods for setting up and cleaning up test data ("fixtures")
- methods for making assertions
- textual and graphical tools for running tests



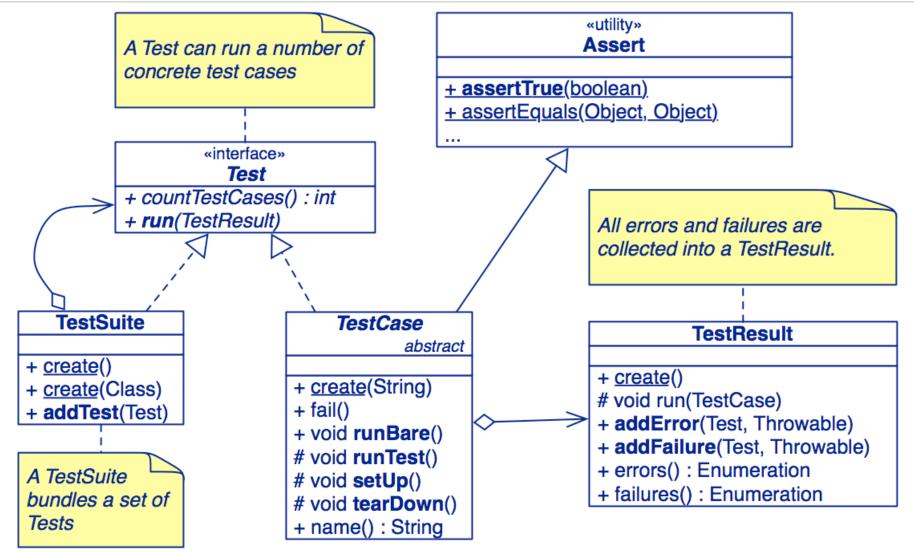
JUnit 3.8

JUnit distinguishes between failures and errors:

- A failure is a *failed assertion*, i.e., an anticipated problem that you test.
- An error is a *condition you didn't check for*, i.e., a runtime error.

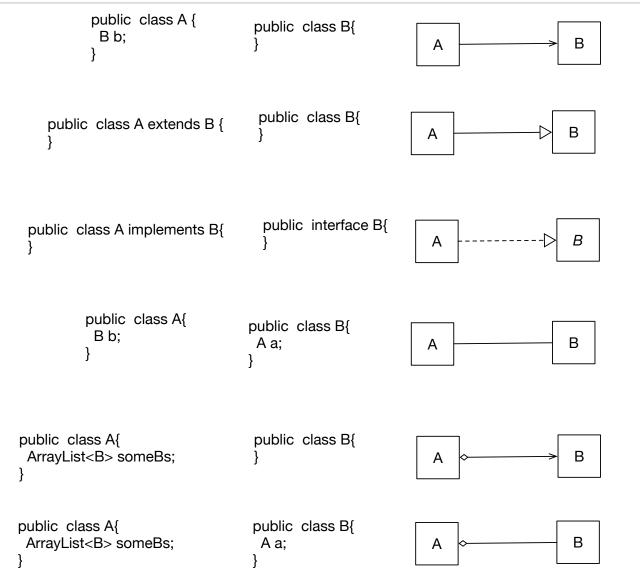


The JUnit 3.x Framework: Class



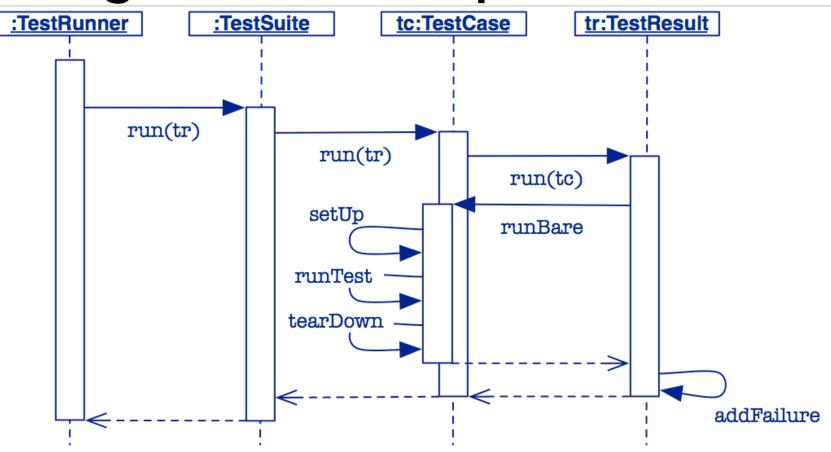


Associations in UML





A Testing Scenario: Sequence



The framework calls the test methods that you define for your test cases.



JUnit 3.x Example Code

```
import junit.framework.*;
public class MoneyTest extends TestCase {
  private Money f12CHF; // fixtures
  private Money f14CHF;
  protected void setUp() { // create the test data
     f12CHF = new Money(12, "CHF");
     f14CHF = new Money(14, "CHF");
   }
  public void testAdd() { // create the test data
     Money expected = new Money (26, "CHF");
     assertEquals("amount not equal",
                    expected, f12CHF.add(f14CHF));
```



{

In PHP

```
<?php
class MoneyTest extends PHPUnit_Framework_TestCase
{
    // ...
    public function testCanBeNegated()</pre>
```

```
// Arrange
$a = new Money(1);
```

PHPUnit is very close to JUnit 3.8

```
// Act
$b = $a->negate();
```

```
// Assert
$this->assertEquals(-1, $b->getAmount());
```

```
// ...
```

}



In Ruby

```
tc simple number2.rb
# File:
require relative "simple number"
require "test/unit"
class TestSimpleNumber < Test::Unit::TestCase</pre>
  def test simple
                                                          Same thing in Ruby
    assert equal(4, SimpleNumber.new(2).add(2) )
    assert equal(4, SimpleNumber.new(2).multiply(2) )
  end
  def test typecheck
    assert raise( RuntimeError ) { SimpleNumber.new('a') }
  end
  def test failure
    assert equal(3, SimpleNumber.new(2).add(2), "Adding doesn't work" )
  end
```

end



Annotations in J2SE 5

J2SE 5 introduces the *Metadata* feature (data about data)

Annotations allow you to add *decorations* to your code (remember javadoc tags: <code>@author</code>)

Annotations are used for code documentation, compiler processing (@Deprecated), code generation, runtime processing

http://java.sun.com/docs/books/tutorial/java/javaOO/annotations.html



JUnit 4.x

JUnit is a simple "testing framework" that provides:

Annotations for marking methods as *tests*

Annotations for marking methods that setting up and cleaning up test data ("fixtures")

methods for making assertions

textual and graphical tools for *running tests*



JUnit 4.x Example Code

```
import org.junit.*;
import static org.junit.Assert.*;
public class MoneyTest {
   private Money f12CHF;
   private Money f14CHF;
   (Before public void setUp() { // create the test data
      f12CHF = new Money(12, "CHF"); // - the fixture
      f14CHF = new Money(14, "CHF");
   @Test public void add() { // create the test data
      Money expected = new Money(26, "CHF");
      assertEquals("amount not equal",
                    expected,f12CHF.add(f14CHF));
```



In C#

```
[TestMethod]
public void Withdraw_ValidAmount_ChangesBalance()
  // arrange
  double currentBalance = 10.0;
  double withdrawal = 1.0;
  double expected = 9.0;
  var account = new CheckingAccount("JohnDoe", currentBalance);
  // act
  account.Withdraw(withdrawal);
  double actual = account.Balance;
  // assert
  Assert.AreEqual(expected, actual);
}
[TestMethod]
[ExpectedException(typeof(ArgumentException))]
public void Withdraw_AmountMoreThanBalance_Throws()
  // arrange
  var account = new CheckingAccount("John Doe", 10.0);
  // act
  account.Withdraw(1.0);
  // assert is handled by the ExpectedException
```

Unit testing in C# is similar

to JUnit 4.X



Testing Style

"The style here is to write a few lines of code, then a test that should run, or even better, to write a test that won't run, then write the code that will make it run."

write unit tests that thoroughly test a single class

write tests as you develop (even before you implement)

write tests for every new piece of functionality

"Developers should spend 25-50% of their time developing tests."





1.JUnit - a testing framework

1.testing practices

2.frameworks vs. libraries

3.JUnit 3.x vs. JUnit 4.x (annotations)

2.Money and MoneyBag - a testing case study



Representing multiple currencies

The problem ...

"The program we write will solve the problem of *representing arithmetic with multiple currencies*. Arithmetic between single currencies is trivial, you can just add the two amounts. ... Things get more interesting once multiple currencies are involved."



MoneyTest

We start by defining a TestCase that exercises the interface we would like our Money class to support:

```
import org.junit.*;
import static org.junit.Assert.*;
public class MoneyTest {
    private Money f12CHF;
    private Money f14CHF;
    @Before public void setUp() { // create the test data
      f12CHF = new Money(12, "CHF");
      f14CHF = new Money(14, "CHF");
    }
....
}
```



Some basic tests...

We define methods to test what we expect to be true ...

```
@Test public void testEquals() {
    assertEquals(f12CHF, f12CHF);
    assertEquals(f12CHF, new Money(12, "CHF"));
    assertFalse(f12CHF.equals(f14CHF));
}
@Test public void testSimpleAdd() {
    Money expected = new Money(26, "CHF");
    Money result = f12CHF.add(f14CHF);
    assertEquals(expected, result);
}
```



Some basic tests

NB: assertTrue, etc. are static imported methods of the Assert class of the JUnit 4.x Framework and raise an AssertionError if they fail.

Junit 3.x raises a JUnit AssertionFailedError (!)



Money

We now implement a Money class that fills our first few requirements:

nublic close Menou (Money
public class Money {	- fAmount : int
• • •	- fCurrency : String
<pre>public Money add(Money m) {</pre>	+ create(int, String)
return new Money();	+ amount() : int
	+ currency() : String
}	+ add(Money) : Money
• • •	+ equals(Object) : boolean
}	+ toString() : String
-	

Note how the test case drives the design!

NB: The first version does not consider how to add different currencies!



Money

We now implement a Money class that fills our first few requirements:

```
public class Money {
    ...
    public Money add(Money m) {
        return new Money(...);
    }
    ...
}
Money
-fAmount:int
-fCurrency:String
+ create(int, String)
+ amount():int
+ currency():String
+ add(Money):Money
+ equals(Object):boolean
+ toString():String
```

What should the class invariant be? (i.e., what are the conditions to have an object Money well formed?)



Running tests from eclipse / IntelliJ

Right-click on the		Java - Money/si			
0] [] ▼ [] @]] 黎 ▼ ③ ▼ [] # @ @ ▼] @ @ A]] Ø [] ♥] ♥] ♥ ♥ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓				
class	Image: Finished after 0.013 seconds Image: Transmission of the second sec	package money.v1;			
(or package) to run		<pre>import static org.junit.Assert.*; import org.junit.*;</pre>			
the tests	money.v1.MoneyTest [Runner: JUn EstEquals testSimpleAdd	<pre>public class MoneyTest { private Money f12CHF; private Money f14CHF;</pre>			
Money.java 16436 2/27/08 private Money f12CHF; MoneyTest.java 164 New v2 Open v3 Open v4 Delete is/P2/Examples/TicTacToe Paste P2/Examples/TicTacToe Paste P2/Examples/TicTacToe Paste P2/Examples/TicTacToe Paste P2/Examples/TicTacToe Import P2/Examples/TicTacToe Paste P2/Examples/TicTacToe Paste		<pre>@Before public void setUp() { f12CHF = new Money(12, "CHF"); f14CHF = new Money(14, "CHF"); } @ @Test public void testEquals() { assertEquals(f12CHF); assertEquals(f12CHF, f12CHF); assertFalse(f12CHF, new Money(12, "CHF")); assertFalse(f12CHF.equals(f14CHF)); } @ @Test public void testSimpleAdd() { Money expected = new Money(26, "CHF"); Money result = f12CHF.add(f14CHF); assertEquals(expected, result); }</pre>			



Testing MoneyBags (I)

To handle multiple currencies, we introduce a MoneyBag class that can hold several instances of Money:

```
import static org.junit.Assert.*;
public class MoneyTest {
    ...
    @Before public void setUp() {
      f12CHF = new Money(12, "CHF");
      f14CHF = new Money(14, "CHF");
      f7USD = new Money( 7, "USD");
      f21USD = new Money(21, "USD");
      fMB1 = new MoneyBag(f12CHF, f7USD);
      fMB2 = new MoneyBag(f14CHF, f21USD);
}
```



Testing MoneyBags (II)

... and define some new (obvious) tests ...

```
@Test public void testBagEquals() {
   assertEquals(fMB1, fMB1);
   assertFalse(fMB1.equals(f12CHF));
   assertFalse(f12CHF.equals(fMB1));
   assertFalse(fMB1.equals(fMB2));
}
```



MoneyBags

We can use a HashTable to keep track of multiple Monies:

```
public class MoneyBag {
    private Hashtable<String, Money> monies = new Hashtable<>();
    public MoneyBag(Money m1, Money m2) {
         this(new Money[]{m1, m2});
    }
                                                                      MoneyBag
                                                                  - fMonies : HashTable
    public MoneyBag(Money[] bag) {
                                                                  + <u>create</u>(Money, Money)
         for(Money m : bag)
                                                                  + create(Money [])
             this.appendMoney(m);
                                                                  - appendMoney(Money)
                                                                  + toString() : String
    }
    private void appendMoney(Money aMoney) {
        Money m = monies.get(aMoney.getCurrency());
         if(m != null) { m = m.add(aMoney); }
        else { m = aMoney; }
        monies.put(aMoney.getCurrency(), m);
    }
}
```



Testing MoneyBags (III)

and we run the tests.

📱 Package Explorer 😤 Navigator 🔂 JUnit 🖾							
Finished	after 0.016 s	econds 🕂	⊕ ∎	a: 🗞 🔈		•	⊳
Runs:	3/3	Errors:	0	E Failures:	0		
	money.v2.Mor	s eAdd	nner: JUn	it 4]			



Adding MoneyBags

We would like to freely add together arbitrary Monies and MoneyBags, and be sure that *equals behave as equals*:

@Test public void mixedSimpleAdd() {
 // [12 CHF] + [7 USD] == {[12 CHF][7 USD]}
 Money[] bag = { f12CHF, f7USD };
 MoneyBag expected = new MoneyBag(bag);
 assertEquals(expected, f12CHF.add(f7USD));
}

This test *fails*. Next time we will see how to fix it!



What you should know!

- How does a *framework* differ from a library?
- What is a *unit test*?
- What is an *annotation*?
- How does *JUnit 3.x* differ from *JUnit 4.x*?
- What is a test "*fixture*"?
- What should you test in a *TestCase*?
- How can testing *drive* design?



Can you answer these questions?

How does implementing toString() help in debugging?

How does the MoneyTest suite know which test methods to run?

How does the TestRunner invoke the right suite() method?

Why doesn't the Java compiler complain that MoneyBag.equals() is used without being declared?



Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

You are free to:

-Share: copy and redistribute the material in any medium or format

-Adapt: remix, transform, and build upon the material for any purpose, even commercially

The licensor cannot revoke these freedoms as long as you follow the license terms

Attribution: you must give appropriate credit

ShareAlike: if you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original

Complete license: https://creativecommons.org/licenses/by-sa/4.0/