

# Extreme Programming dengan Java Open Source Tools

JUnit adalah salah satu *tools* yang membantu kita untuk melakukan unit testing terhadap kode yang cocok sekali bila diimplementasikan dalam XP. Martin Fowler berkata “*Whenever you are tempted to type something into a print statement or a debugger expression, write it as a test instead.*”

Artikel edisi sebelumnya, telah dibahas apa itu *Extreme Programming* dan focus artikel ini pada *continuous integration* dan *automated testing*, yang tentu saja, karena bidang penulis adalah Java, maka penulis menerangkannya menggunakan *tools-tools* Java yang kebetulan *open source* untuk melakukan implementasi Extreme Programming. Dimana, pada edisi lalu diterangkan Ant untuk automated testing.

Artikel ini akan membahas untuk melakukan *testing* terhadap kode-kode yang telah dibuat menggunakan *framework* xunit yaitu JUnit (<http://www.junit.org>). *Framework* ini disebut xunit, karena saat ini setelah ada banyak varian dari *unit testing* yang bekerja serupa untuk setiap bahasa, diantaranya untuk Python ada PyUnit.

## Mengapa kita perlu Unit Testing?

JUnit adalah sebuah *framework* yang dikembangkan di atas Java, yang dibuat oleh masternya *Extreme Programming*, Kent Beck dan masternya Design Pattern, Erich Gamma. JUnit dapat dikatakan sebuah *simple test framework* yang didesain untuk melakukan *testing* yang bersifat berulang. Sebelumnya Kent Beck mengembangkan Sunit yaitu *unit testing* untuk SmallTalk yang sangat populer dengan MVC-nya.

Umumnya programmer malas untuk membuat unit testing karena dianggap pekerjaan tambahan. Memang *sih*, untuk pekerjaan yang simpel dan tetap, unit testing dirasakan memberatkan, tetapi kalau *programmer* yang sudah terbiasa membuat komponen pasti tahu *deh*, apa yang terjadi kalau ada kode-kode yang sudah *berjibun* dan *direfactor*, sebagai contoh misalnya semua kode dari `com.intercitra.model.*`

dipindahkan ke `org.blueoxygen.cimande.model.*`. Pekerjaan di atas memang mudah apalagi kalau menggunakan Eclipse dari `eclipse.org`, tinggal click kanan, pilih *refactor*, terus *rename*, selesai. Sadar, tidak sadar, semua code yang berhubungan dengan object didalam package `com.intercitra.model.*` harus dirubah. Kalau semua code dalam Eclipse Project, kita tidak perlu pusing, karena secara tidak langsung semua import `com.intercitra.model.*` akan di-*rename* menjadi import `org.blueoxygen.cimande.model.*`.

Sayangnya, apa dikata, bagi programmer JSP, tetap harus melakukan *unit testing* satu per satu semua kegiatan yang memanggil object didalam package `org.blueoxygen.cimande.model.*`, harus edit satu per satu, makanya sampai saat ini JSP dibidang salah satu mekanisme pemrograman yang kotor dengan kata lain membuat pekerjaan tambahan di kemudian hari. Lain halnya dengan aplikasi menggunakan pendekatan lain yang lebih OOP, kita dapat melakukan *refactoring* secara cepat terhadap seluruh kode.

Apakah dengan melakukan *refactoring*, semua programmer yakin semua kode kita akan berjalan dengan mulus dan *stable*? *Unit testing* adalah jawaban untuk merealisasikannya, karena unit testing adalah sebuah mekanisme yang meyakinkan kita bahwa setiap perubahan terhadap kode akan menghasilkan *result* yang benar. Unit testing yang sukses akan lebih meyakinkan bahwa kode kita ini benar-benar berjalan dengan baik.

## JUnit, IDE, dan Ant

Sebenarnya melakukan *unit testing* itu gampang-gampang susah, karena secara konsep yang mengacu pada JUnit, *unit*

```
public class MoneyTest extends TestCase {
    public void testSimpleAdd() {
        Money m12CHF= new Money(12,
            "CHF"); // (1)
        Money m14CHF= new Money(14,
            "CHF");
        Money expected= new Money(26,
            "CHF");
        Money result= m12CHF.add
            (m14CHF); // (2)
        Assert.assertTrue(expected.equals
            (result)); // (3)
    }
}
```



*testing* itu dipecah menjadi dua bagian, yaitu *TestCase* dan *TestSuite*, di mana *TestSuite* merupakan kumpulan *TestCase-TestCase*.

Hebatnya saat ini IDE yang ada sudah memasukan JUnit secara terintegrasi, sehingga kita tidak perlu pusing, jalankan IDE, buat *TestCase*, *execute testing*, selesai. Malahan IDE yang hebat tersebut kita tidak perlu membeli, semuanya *open source*, cobalah kunjungi [eclipse.org](http://eclipse.org) atau [netbeans.org](http://netbeans.org) untuk mendapatkannya.

Apakah dengan mekanisme seperti melakukan *unit testing* secara *console based* tidak relevan lagi? Dari pengalaman penulis, didapatkan bahwa melakukan *testing* dengan IDE memang menyenangkan, kita bisa melakukan *testing* satu demi satu, tetapi apa yang terjadi kalau kita ada 1000 *TestSuite*, dan setiap *TestSuite* ada 100 *TestCase*? *Wallahualam*, ini mimpi buruk yang lain lagi.

Sayangnya mimpi buruk tersebut telah dipecahkan oleh team Ant, yang telah melakukan JUnit *integration* ke dalam Ant, sehingga kita dapat membuat sebuah kegiatan *automated testing* bersamaan dengan unit testing secara mudah, di mana pengembangan Ant untuk pemula telah dibahas diartikel *Extreme Programming* sebelumnya (Bagian 1).

Semua programmer akan merasakan kemudahan pemrograman bila menggunakan Ant yang sudah diberikan JUnit, karena dari sini *source code* bisa didistribusikan terpadu dengan skenario unit testing dalam satu eksekusi *syntax*. Ant akan secara otomatis melakukan *compile source code*, baik itu *source code* utama maupun *source code* untuk *unit testing*, dan saat yang bersamaan (berurutan tentu saja), akan melakukan *unit testing*, dan setelah itu kita bisa lihat *resultnya*. *Result* JUnit dapat digenerate

untuk ditampilkan di *console* maupun dalam bentuk HTML, malah bisa dipanggil secara *standalone* dengan AWT atau Swing.

Bagi mereka yang bekerja di bidang pengembangan komponen atau API, pasti merasakan pentingnya JUnit ini. Hal ini dikarenakan setiap penerima komponen akan merasa yakin bila lulus *unit testing* ini, komponennya lebih ok. Umumnya bila terjadi *error*, kita bisa tahu kode mana yang *error*. Walaupun terus terang saja step awalnya lumayan melelahkan, kita harus buat TestCase satu satu, setiap kasus satu TestCase.

Maklum sampai hari ini penulis belum menjumpai *unit testing* untuk TestCase, yang ada adalah TestCase untuk kode. Jadi kalau TestCase-nya yang salah, pusing juga, walaupun *asserting*-nya bisa dilihat di *console*.

## TestCase dan TestSuite di JUnit.

Seperti yang telah diutarakan sebelumnya, *unit testing* dengan JUnit hanya diperlukan dua keahlian, yaitu membuat TestCase dan TestSuite, di mana TestSuite adalah kumpulan *testing* untuk melakukan eksekusi TestCase.

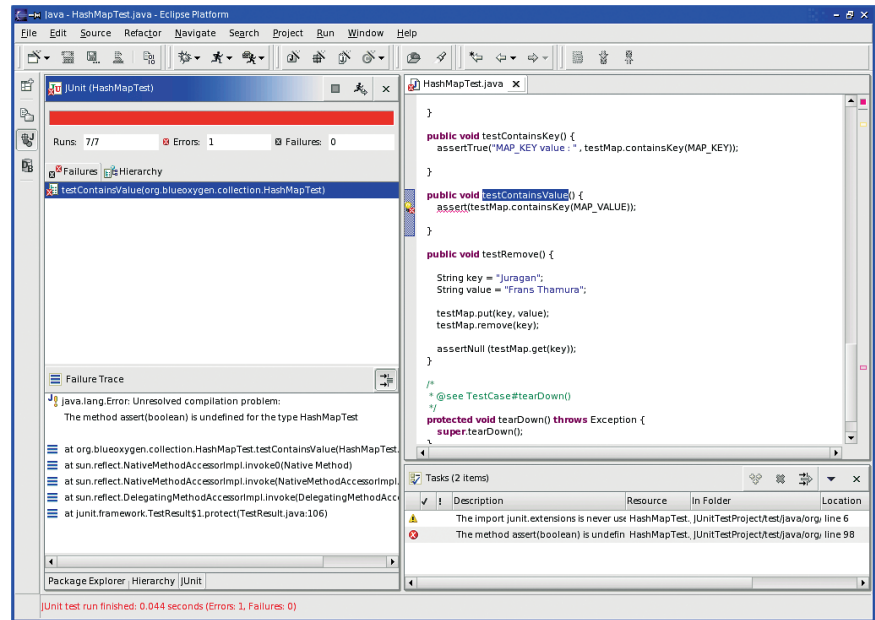
Struktur pengembangan TestCase itu simple sekali, dan untuk lebih jelasnya lihatlah kode berikut ini:

```
public class MoneyTest extends TestCase {

    public void testSimpleAdd() {
        Money m12CHF = new Money(12, "CHF"); // (1)
        Money m14CHF = new Money(14, "CHF");
        Money expected = new Money(26, "CHF");
        Money result = m12CHF.add(m14CHF); //(2)
        Assert.assertTrue(expected.equals(result)); //(3)
    }
}
```

diasar adalah kode yang sederhana, sedangkan yang lebih lengkap adalah:

```
public class MoneyTest extends TestCase {
    private Money f12CHF;
    private Money f14CHF;
```



### Failures Test on Eclipse.

```
protected void setUp() {
    f12CHF = new Money(12, "CHF");
    f14CHF = new Money(14, "CHF");
}

public void testEquals() {
    Assert.assertTrue(!f12CHF.equals(null));
    Assert.assertEquals(f12CHF, f12CHF);
    Assert.assertEquals(f12CHF, new Money(12, "CHF"));
    Assert.assertTrue(!f12CHF.equals(f14CHF));
}

public void testSimpleAdd() {
    Money expected = new Money(26, "CHF");
    Money result = f12CHF.add(f14CHF);
    Assert.assertTrue(expected.equals(result));
}
```

Code di atas adalah sebuah TestCase simple untuk melakukan test terhadap code Money, di mana testingnya adalah testEquals() untuk test method equals() dan testSimpleAdd() untuk test method simpleAdd(). Setiap assert akan membandingkan *result* yang dihasilkan, bila lolos, berarti testing lolos skenario tersebut.

## Bekerja dengan JUnit

Untuk memulai pengembangan dengan JUnit, buatlah sebuah *project*, yang termudah adalah menggunakan Eclipse. Artikel mengenai Eclipse ada di *InfoLINUX* edisi Januari 2004 atau kunjungi <http://www.eclipse.org>.

Sesuai yang disarankan oleh *team* JUnit, sebaiknya kita membuat 2 *folder* pada proyek tersebut yaitu *src* untuk menyimpan *source code*, dan *test* untuk menyimpan *unit testing code*, sehingga bila kita akan melakukan *packaging binary* dari *source code*, *unit testing*nya tidak terbawa. Untuk memudahkan biasanya *source code* ditaruh di subfolder *java* di bawah *src*, karena umumnya para *programmer* menyimpan *source code* lainnya juga di dalam *folder* *src*, seperti *sql*, *template*, *config*, malah tidak jarang memasukkan *WAR* container ke dalamnya.

Jadi struktur yang disarankan penulis adalah sebagai berikut:

```
src
  java
    org
      blueoxygen
        SomeClass1.java
        SomeClass2.java
  sql
    smiletown.sql
  conf
    cimande.properties
```

```
test
java
org
blueoxygen
SomeClass1Test.java
SomeClass2Test.java
SomeClassSuite.java
```

Kasus di atas dapat diartikan bahwa ada code java dengan nama org.blueoxygen. SomeClass1 yang akan dilakukan unit testing, di mana code unit testingnya adalah org.blueoxygen.SomeClass1Test, sedangkan org.blueoxygen.SomeClass2, kode *unit testing*nya adalah SomeClass2Test, serta sebuah TestSuite org.blueoxygen. SomeClassSuite.

Bila suatu hari dirasakan bahwa menggunakan package org.blueoxygen untuk test merasa terganggu, dimungkinkan untuk menggunakan package org.blueoxygen.test atau test.org.blueoxygen.

## Membuat TestCase

Untuk membuat sebuah TestCase sebenarnya mudah sekali, Eclipse telah menyediakan *Wizard*nya, sehingga kita hanya perlu memasukkan kode *testing*nya seperti Assert.assertTrue, Assert.assertEquals atau Assert.assertNull.

Contohnya adalah seperti berikut:

```
assertEquals(3, testMap.size());
assertEquals("Intercitra", testMap.get("Company"));
```

Baiklah, untuk memulai project *unit testing*, kita akan membuat sebuah *unit*

*testing* untuk HashMap, sebuah kode yang sudah disediakan secara otomatis oleh Java SDK, HashMap adalah salah satu dari *Java Collection Library*, selain HashMap, ada beberapa Java Collection seperti Hashtable, ArrayList, atau List.

Didalam HashMap, terdapat banyak method seperti put, putAll, size, get, ataupun containsKey, maka untuk melakukan unit testing sebaiknya semua testing tersebut dilakukan satu persatu, seperti untuk testing put kita membuat sebuah method dengan nama testPut() didalam TestCase, testing method get menggunakan method testGet() di dalam TestCase. Untuk memudahkannya cobalah buat sebuah *file* HashMapTest. java dibawah *directory* yang telah diterangkan diatas, dan ketikkan baris di bawah ini:

```
package org.blueoxygen.collection;

import junit.framework.TestCase;
import java.util.Map;
import java.util.HashMap;

/*
 * Created on Jan 18, 2004
 *
 * To change the template for this
 * generated file go to
 * Window>Preferences>
 * Java>Code Generation>Code and
 * Comments
 */

/**
 * @author Frans Thamura
 * (frans@intercitra.com)
 *
 * To change the template for this
 * generated type comment go to
 * Window>Preferences>Java>
 * Code Generation>Code and
 * Comments
 */

public class HashMapTest extends
    TestCase {

    private Map testMap;
    private Map testMap2;

    private static final String MAP_KEY =
```

```
"Open Source";
    private static final String MAP_VALUE
        = "BlueOxygen";

    /**
     * Constructor for HashMapTest.
     * @param arg0
     */
    public HashMapTest(String arg0) {
        super(arg0);
    }

    // You can run the TestCase as a
    // standalone Swing
    // If you are using Eclipse, you dont
    // need this
    public static void main(String[] args) {

        // You can use swingui, awtui or
        // text ui. But, I love Swing UI
        junit.swingui.TestRunner.run
            (HashMapTest.class);
    }

    /**
     * @see TestCase#setUp()
     */
    protected void setUp() throws
        Exception {

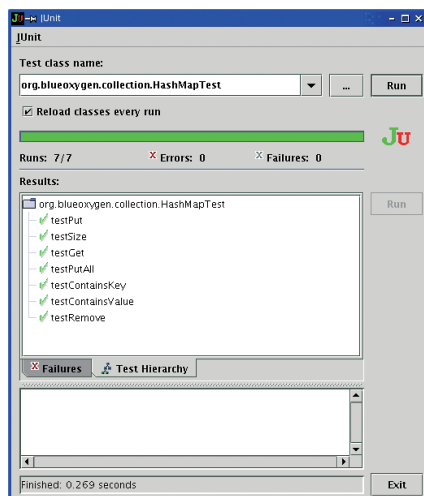
        // First Collection
        testMap = new HashMap();
        testMap.put(MAP_KEY,
            MAP_VALUE);
        testMap.put("Open Source",
            "BlueOxygen");

        testMap2 = new HashMap();
        testMap2.put("Company",
            "Intercitra");
        testMap2.put("Community", "Java
            User Group");
    }

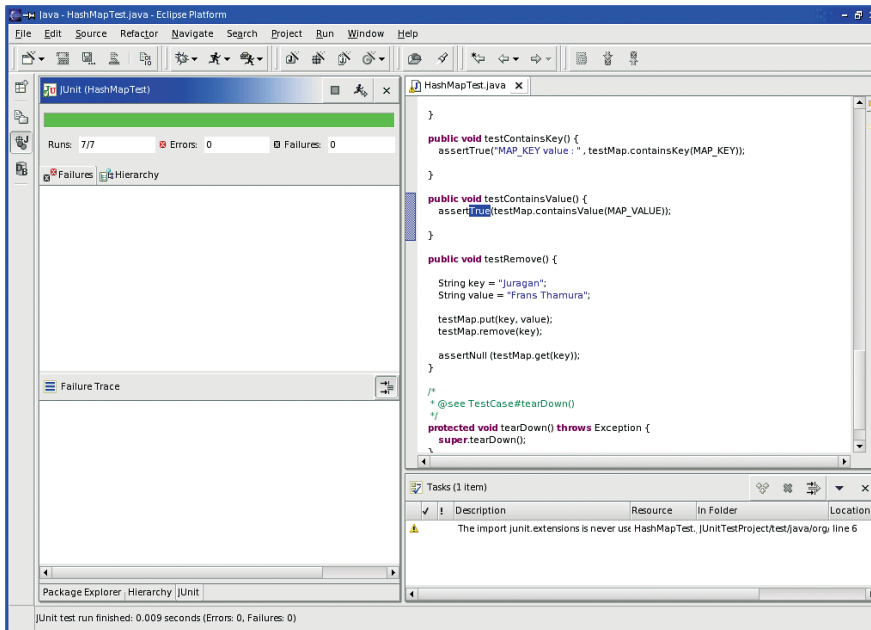
    // the real test

    public void testPut() {
        String key = "Juragan";
        String value = "Frans Thamura";

        // second Put
        testMap.put(key, value);
```



▲ Standalone JUnitTest.



#### Success Test.

```
String value2 = (String)testMap.get(key);
assertEquals("The value back from the map ", value, value2);
}
```

```
public void testSize() {
    assertEquals(1, testMap.size());
}
```

```
public void testGet() {
    assertEquals("BlueOxygen", testMap.get(MAP_KEY));
    assertNull(testMap.get("JUNK_KEY"));
}
```

```
public void testPutAll() {
    // put testMap2 to testMap
    testMap.putAll(testMap2);
    // check the size, must be 3, because the "Juragan" in the testMap now.
    assertEquals(3, testMap.size());
    assertEquals("Intercitra", testMap.get("Company"));
}
```

```
public void testContainsKey() {
    assertTrue("MAP_KEY value : " , testMap.containsKey(MAP_KEY));
}
```

```
}

public void testContainsValue() {
    assertTrue(testMap.containsValue(MAP_VALUE));
}

public void testRemove() {

    String key = "Juragan";
    String value = "Frans Thamura";

    testMap.put(key, value);
    testMap.remove(key);

    assertNull (testMap.get(key));
}

/*
 * @see TestCase#tearDown()
 */
protected void tearDown() throws Exception {
    super.tearDown();
}
```

Kemudian *executelah* sebagai JUnit, lihat hasilnya, bila warnanya hijau, berarti pembaca telah berhasil membuat satu TestCase.

## Membuat TestSuite

Untuk memudahkan konsep pengembangan TestSuite, buatlah sebuah *script* seperti di bawah ini dan tentu saja di bawah *folder* test/org/blueoxygen/collection.

```
/*
 * Created on Jan 18, 2004
 *
 */
package org.blueoxygen.collection;

import junit.framework.Test;
import junit.framework.TestSuite;

/**
 * @author Frans Thamura
 * (frans@intercitra.com)
 */
public class AllTests {

    public static void main(String[] args) {
        junit.swingui.TestRunner.run
            (AllTests.class);
    }

    public static Test suite() {
        TestSuite suite = new TestSuite
            ("Test for org.blueoxygen.
            collection");
        //$JUnit-BEGIN$
        suite.addTest(new TestSuite
            (HashMapTest.class));
        //$JUnit-END$
        return suite;
    }
}
```

TestSuite di atas akan mengeksekusi TestCase, yang dalam kasus di atas adalah HashMapTest.class.

Frans Thamura (frans@blueoxygen.org)

## Tip

Lakukan testing sesering mungkin. *Unit testing* yang baik adalah *unit testing* yang dibuat sebelum kode dibuat. Kita tidak perlu membuat semua *unit testing*, buatlah *unit testing* yang akan membuat terputusnya rantai *testing*, karena suka tidak suka membuat kode testing sama dengan investasi, jadi tentu saja memerlukan biaya dan waktu, jadi lakukanlah secara efektif.