IF2281 – Java Tools

Achmad Imam Kistijantoro
Semester II 2007/2008

Overview

Tools yang umum digunakan dalam software project

- Build tools: Ant & Maven
- version control (tidak dibahas)
- continuous integration (tidak dibahas)
- Test tools: JUnit

build tools

- mengapa perlu build tools
  - kompilasi: java <nama source>
  - perlu banyak kerja..

- jika kode besar, dikelompokkan ke dalam package berbeda, dengan direktori berbeda
- hasil kompilasi (class file) dipisahkan ke dalam direktori berbeda dengan sumber
- library eksternal yang diperlukan

- tools dapat membantu proses kompilasi seluruh file dalam sebuah project hanya dengan satu perintah

- syarat: reproducible, portable, automatic

Ant

- Ant adalah alat bantu proses build pada Java, setara dengan make pada C/C++
- Dikembangkan oleh Apache (http://ant.apache.org)
- Dengan Ant, kita dapat:
  - melakukan kompilasi beberapa file/direktori sekaligus
  - mendefinisikan parameter kompilasi
  - mendefinisikan dependensi antar elemen aktivitas yang harus dilakukan (kompilasi modul A bergantung pada B, pembuatan direktori, pendefinisian variabel untuk proses build
  - semua hal di atas didefinisikan dalam sebuah file script, sehingga dapat dipanggil ulang dengan mudah hanya dengan 1 perintah saja

build file

- proses build dengan menggunakan ant dikendalikan oleh build file yang memiliki format xml (nama default: build.xml)
- build file berisi sekumpulan target-target yang dapat dipanggil dengan menggunakan perintah:
  - ant <nama target>
- sebuah target pada build file berisi perintah-perintah (ant task), misalnya untuk kompilasi, kopi file, buat/hapus direktori, melakukan packaging dsb.
- sebuah target dapat memiliki dependency terhadap target lain
- build script dapat berisi definisi properties/atribut: nama direktori sumber, lokasi tool/kompilator yang digunakan

contoh build.xml

```
<?xml version="1.0"?>
  <project name="Hello">

    <property name="src" value="src"/>

    <target name="compile">
      <javac srcdir="${src}" destdir="classes"/>
    </target>

    <target name="run" depends="compile">
      <java classname="Hello" classpath="classes" />
    </target>
  </project>
```

elemen build.xml

- project: elemen induk build file.
  - name, default, basedir
- target: satuan tugas/aktivitas yang dapat dipanggil user
  - name, depends
- task: satuan perintah yang dapat dijalankan oleh sebuah target
  - contoh task: javac, java, javadoc, copy, createdir
- property: digunakan untuk mendefinisikan atribut/variabel yang diperlukan dalam proses build

struktur direktori

- saat mengembangkan program Java, sebaiknya file source code dengan file class hasil kompilasi diletakkan pada direktori terpisah
  - memudahkan proses clean (menghapus hasil build sebelumnya)
  - memudahkan proses packaging (mengumpulkan file class ke dalam sebuah file jar)
- umumnya, source code diletakkan ke dalam direktori src, dan hasil kompilasi diletakkan ke dalam direktori classes
- jika kita menggunakan resources tambahan (gambar, teks) diletakkan pada direktori resources
- jika kita menggunakan library eksternal, letakkan dalam direktori lib
- untuk project yang berukuran besar (terdiri atas beberapa modul), setiap modul dapat diletakkan pada direktori terpisah, dan memiliki build file tersendiri

struktur direktori umum

Nama direktori	keterangan
src	source code aplikasi
test	code untuk unit test
lib	library yang digunakan project
build	hasil build
build/classes	classes hasil build
build/test-classes	test classes hasil build
dist	file siap untuk distribusi

menggunakan Ant

- dependency antar target

```
<target name="package" depends="compile"
   description="Generate JAR file">
<target name="compile" depends="init"
   description="Generate JAR file">
```

- Pendefinisian path

```
<path id="test.classpath">
  <path refid="compile.classpath"/>
  <pathelement location="lib/junit.jar"/>
  <pathelement path="build/test-classes"/>
</path>
```

task umum

- kompilasi:

```
<javac srcdir="${src}" destdir="${build}"
  classpath="xyz.jar" debug="on" source="1.4" />
```

- packaging:

```
<jar destfile="${dist}/lib/app.jar"
  basedir="${build}/classes"/>
```

- copy

```
<copy file="myfile.txt" todir="../some/other/dir"/>
```

- clean

```
<target name="clean" depends="init">
        <delete dir="${build.dir}"/>
        <delete dir="${dist.dir}"/>
</target>
```

- doc
  - hal berikut. ..

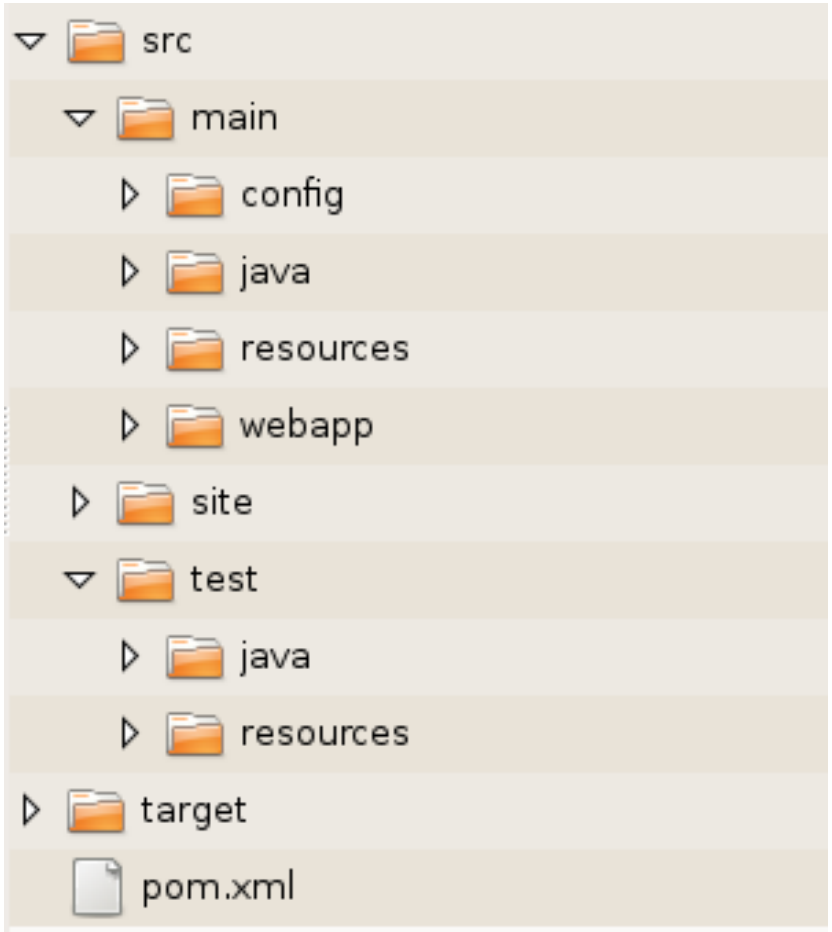
```
<target name="javadoc" depends="compile,init"
   description="Generate JavaDocs.">
  <javadoc sourcepath="${src.dir}"
    destdir="${reports.javadoc}" author="true"
    version="true" use="true" access="private"
    linksource="true"
    windowtitle="${ant.project.name} API">
      <classpath>
        <path refid="compile.classpath" />
        <pathelement path="${build.classes.dir}" />
      </classpath>
<doctitle><![CDATA[<h1>${ant.project.name}</h1>]]></doctitle><bot
   tom><![CDATA[<i>Copyright &#169; 2007 All Rights Reserved.
   </i>]]></bottom>
  </javadoc>
</target>
```

Maven

- Maven dapat disebut sebagai alat bantu proses build pada Java, namun menyediakan fitur lebih dari sekedar build tool. Oleh pengembangnya, maven disebut sebagai project management framework
- Maven dikembangkan oleh Apache (http://maven.apache.org)
- Saat mengembangkan aplikasi multi-developer, sering terdapat masalah:
  - mengatur ketergantungan antar berbagai versi library yang digunakan pada setiap modul
- Maven menyediakan repository untuk mengatur ketergantungan antar berbagai library yang digunakan

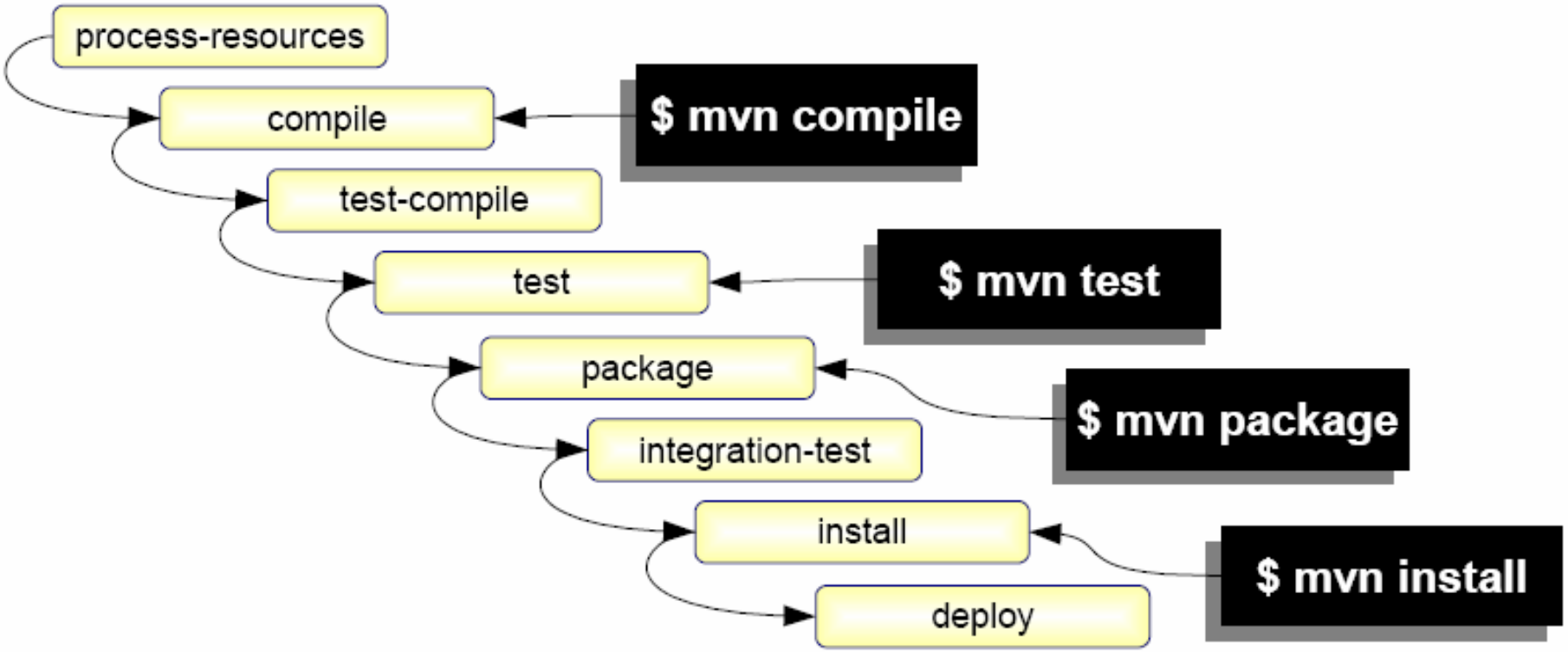
Maven

- menyediakan standar proses: code generation, kompilasi, testing, packaging, deploy, documentation
- struktur project maven:



Maven

- standard process



Maven

- pada project maven, kita cukup mendefinisikan:
  - deskripsi project (nama, versi, developer)
  - external library & dependency
  - kustomisasi proses

- seluruh proses standar telah tersedia: kompilasi, packaging, deploy, testing, cleanup.

maven

- deskripsi project

```
<project...>
<!-- PROJECT DESCRIPTION -->
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.mycompany</groupId>
   <artifactId>myapp</artifactId>
   <packaging>jar</packaging>
    <name>Killer application</name>
    <version>1.0</version>
    <description>My new killer app</description> ...
</project>
```

maven

- definisi library & dependency

```
<project...> ...

  <!-- PROJECT DEPENDENCIES -->
  <dependencies>
  <!-- Hibernate -->
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate</artifactId>
      <version>3.2.4.</version>
    </dependency>
   <!-- Log4j -->
     <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.14</version>
     </dependency>
... </dependencies> </project>
```

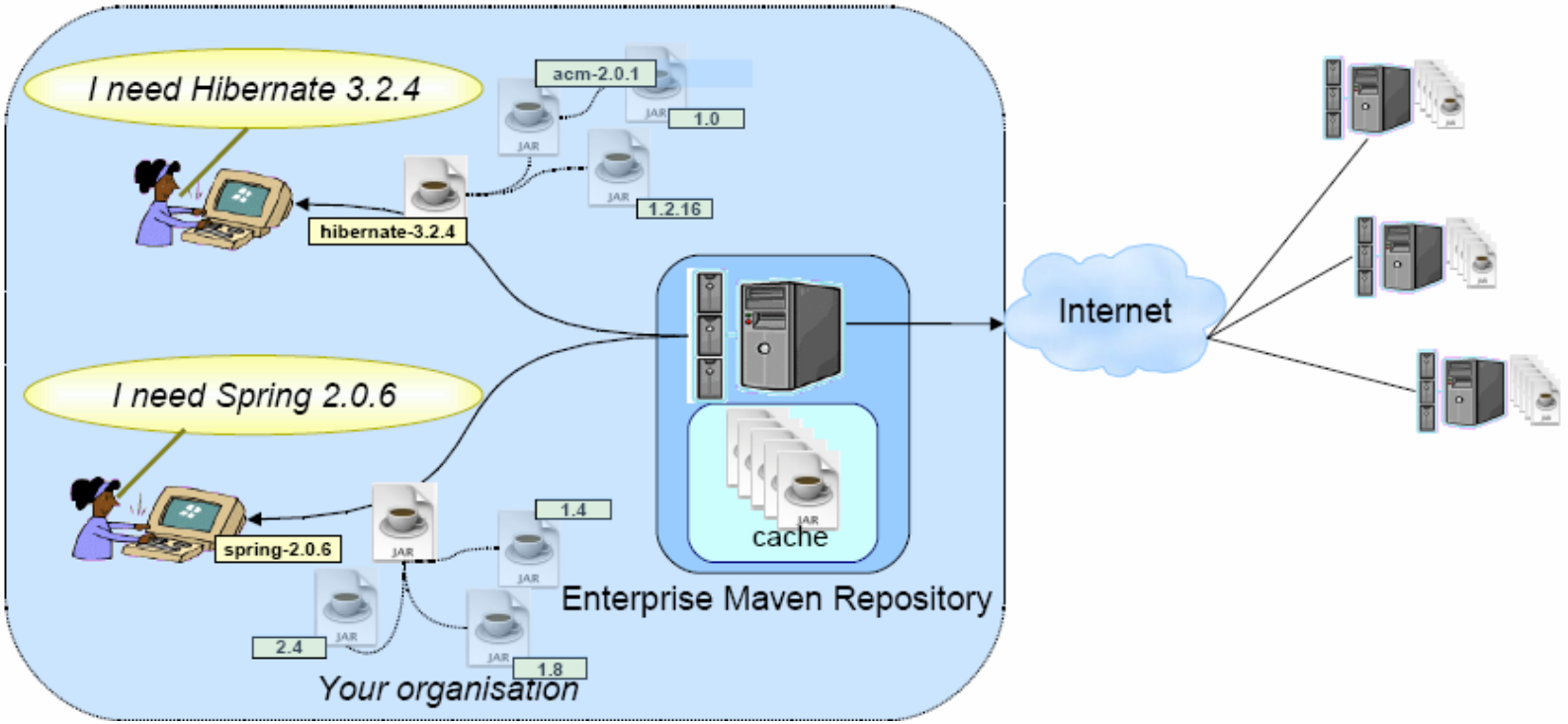
maven

- kustomisasi proses

```
<project...> ...
  <build>
    <plug-ins>
    <!-- Using Java 5 -->
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
           <source>1.5</source>
           <target>1.5</target>
        </configuration>
      </plugin>
    </plug-ins>
  </build> ...
</project>
```

maven

- developer cukup mendefinisikan dependency terhadap library yang langsung digunakan pada project tersebut
- maven akan mencek dependency secara transitif
- library dikelola oleh repository
- maven otomatis mendownload semua library yang diperlukan sebuah project



Intro JUnit

- JUnit adalah opensource framework untuk unit testing pada Java
- Dikembangkan oleh Erich Gamma (Design Pattern) dan Kent Beck (Extreme Programming)
- Target: mempercepat coding & meningkatkan kualitas software

Teknik umum untuk menguji kode

- Setiap programmer pernah melakukan unit test
- Membuat kelas Driver
- Membuat instance kelas yang akan diuji
- Memanggil method yang akan diuji
- Membandingkan hasilnya (mencetak hasil ke layar, dan memeriksa manual)
- Menggunakan debugger

Kelemahan

- Pembandingan hasil dilakukan secara manual
- Tidak ada konsep tes berhasil atau gagal, karena tergantung interpretasi penguji
- Tidak ada repeatability: setiap pengujian, sukses tidaknya sebuah pengujian bergantung pada interpretasi penguji
- Hasil pengujian tidak dapat dikumpulkan secara otomatis

Pro unit test

- Identifikasi defect pada tahap awal implementasi
- Testing mempengaruhi desain
- Tes yang berhasil akan meningkatkan kepercayaan diri
- Testing membuat kita membaca ulang kode yang kita tulis
- Testing dapat digunakan sebagai dokumentasi awal cara memanggil/menggunakan sebuah modul

Against unit test

- Koding unit test memerlukan waktu
- Membosankan
- Terlalu percaya diri, kode saya sudah benar
- Testing hanya dilakukan oleh tester
- Menunda testing hingga kode selesai dibuat

Urutan membuat unit test

- Mengembangkan sekumpulan test cases (test suite)
- Membuat fixture (sekumpulan data yang diperlukan oleh beberapa test cases)
- Menjalankan tes, mengumpulkan hasil
- Membuat report dan analisis hasil tes

Filosofi

- Unit Test dibuat oleh developer
- Tes harus dapat dijalankan ulang semudah mungkin
- Tes dibuat seawal mungkin
- Tes dikembangkan secara incremental

Fitur JUnit

- Menyediakan asersi untuk pembandingan hasil tes
- Menyediakan fixtures untuk resource & data yang digunakan dalam tes
- Menguji exception
- Suites & TestRunner
- Integrasi dengan tools lainnya (Netbeans, Eclipse, Ant)

Tahapan testing dengan JUnit (3)

- Membuat kelas untuk test sebagai turunan dari TestCase
- Membuat method dengan nama testxxx() untuk setiap tes yang akan dilakukan
- Membuat fixtures untuk data dan resource yang digunakan dalam tes
- Menulis kode tes
- Menjalankan tes

Contoh tes sederhana

```
import java.util.*;
import junit.framework.*;

public class SimpleTest extends TestCase {

  public void testEmptyCollection() {
    Collection testCollection = new ArrayList();
    assertTrue( testCollection.isEmpty() );
  }

  public static void main(String args[] ) {
    junit.textui.TestRunner.run(SimpleTest.class);
  }
}
```

Test Fixtures

- Sebuah Test case dapat berisi lebih dari satu method tes. Untuk menyederhanakan koding, kita dapat melakukan inisiasi data & alokasi resource yang digunakan pada satu tempat, dengan mendefinisikan method setUp, dan melakukan dealokasi resource pada method tearDown

- Setiap method test yang dipanggil, akan melakukan setUp dan tearDown secara terpisah, sehingga tes pada satu method tidak mengganggu tes method lainnya.

Contoh

```
import java.util.Vector;

import junit.framework.*;

public class VectorTest extends TestCase {
   protected Vector fEmpty;
   protected Vector fFull;

   public static void main (String[] args)
   {
     junit.textui.TestRunner.run (
              VectorTest.class );
   }
   protected void setUp() {
     fEmpty= new Vector();
     fFull= new Vector();
     fFull.addElement(new Integer(1));
     fFull.addElement(new Integer(2));
     fFull.addElement(new Integer(3));
   }

    protected void tearDown() { }
}
```

```
public void testCapacity() {
  int size= fFull.size();
  for (int i= 0; i < 100; i++)
    fFull.addElement(new
Integer(i));
    assertTrue(
         fFull.size() ==
100+size);
}


public void testRemoveAll() {
     fFull.removeAllElements();
     fEmpty.removeAllElements();
     assertTrue(fFull.isEmpty());

     assertTrue(fEmpty.isEmpty())
;
}
```

Hirarki tes

- Kita dapat membuat hirarki tes dengan mendefinisikan 1 atau lebih test suites
- Test Suite A
  - Test case 1
  - Test case 2
  - Test Suite B
    - Test case 3
- Test Suite C
  - Test case 4

Contoh

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

/**
 * TestSuite that runs all the sample tests
 */
public class AllTests {

   public static void main (String[] args) {
       junit.textui.TestRunner.run (suite());
   }
   public static Test suite ( ) {
       TestSuite suite= new TestSuite("All JUnit Tests");
       suite.addTest(new TestSuite( VectorTest.class ));
       suite.addTest(new TestSuite( SimpleTest.class ));
       return suite;
   }
}
```

Assertion

- assertEquals
- assertFalse
- assertNotNull
- assertNotSame
- assertNull
- assertSame
- assertTrue
- fail

Menentukan tes

- Fungsionalitas unit
- Code coverage
- Decision coverage

Contoh

- Buatlah kelas IntegerSet, yang menyediakan fungsi:
  - void add(int x): menambahkan sebuah integer ke dalam set. Jika telah ada sebelumnya, instance IntegerSet hanya berisi sebuah x saja
  - void remove(int x): menghapus x dari IntegerSet, dan melemparkan exception ElementNotFound jika tidak ada x
  - Int findClosest(int x): mengembalikan elemen anggota IntegerSet yang paling dekat dengan x. Jika x adalah anggota IntegerSet, maka fungsi akan mengembalikan x. Jika ada 2 buah element yang memiliki jarak/perbedaan yang sama dengan x, maka elemen yang lebih kecil yang akan dikembalikan. Contoh: jika anggota IntegerSet adalah {1, 2, 6}, maka findClosest(5) akan mengembalikan 6, dan findClosest(4) akan mengembalikan 2.
- Tentukan test case yang akan dibuat.

Integrasi dengan Ant

- Ant: alat bantu untuk melakukan build pada java
- Integrasi Ant-JUnit: melakukan unit test sebagai bagian dari proses build

Contoh ant build script

```
<project name="p1">

<target name="compile">
  <javac destdir="classes">
    <src path="src"/>
    <classpath path="lib/junit-3.8.1.jar"/>
  </javac>
</target>

<target name="test" depends="compile">
  <junit>
    <classpath path="classes"/>
    <test name="SimpleTest"/>
  </junit>
</target>

</project>
```

<junit>

- printsummary: mengatur tampilan hasil tes pada layar
- haltonfailure: berhenti menjalankan tes sat failure
- <formatter>: mengatur report hasil tes dalam bentuk file
  - type: xml, plain, brief

JUnit 4

- JUnit 4 memanfaatkan fitur annotation yang disediakan oleh JDK 5
- Test case tidak perlu lagi diturunkan dari kelas TestCase, namun sebagai kelas biasa yang memiliki method dengan anotasi @Test
- Fixtures diidentifikasikan dengan anotasi @Before dan @After
- Exception diperiksa dengan menggunakan parameter anotasi @Test(expected=XXX)
- Method untuk asersi digunakan langsung dengan static import

Contoh JUnit 4

```
import java.util.*;
import org.junit.Test;
import static org.junit.Assert.*;

public class SimpleTest {

  @Test
  public void emptyCollection() {
    Collection testCollection = new ArrayList();
    assertTrue( testCollection.isEmpty() );
  }

  public static junit.framework.Test suite() {
    return new JUnit4TestAdapter(SimpleTest.class);
  }

}
```

Contoh JUnit 4

```
public class VectorTest {
  protected Vector fEmpty;
  protected Vector fFull;

  @Before
  void initData() {
    fEmpty= new Vector();
    fFull= new Vector();
    fFull.addElement(new Integer(1));
    fFull.addElement(new Integer(2));
    fFull.addElement(new Integer(3));
  }
```

Contoh JUnit 4

```
@Test(expected= IndexOutOfBoundsException.class)
  public void empty() {
    new ArrayList<Object>().get(0);
  }
```

Sumber

- Ant Manual (http://ant.apache.org/manual/index.html)
- V. Massol, J van Zyl et. al. Better Builds with Maven. Mergere Library Press, 2006
- John F. Smart. Java Power Tools. O'Reilly, 2008