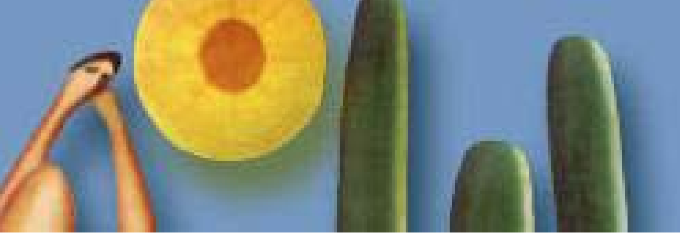


Brasil@JavaOne

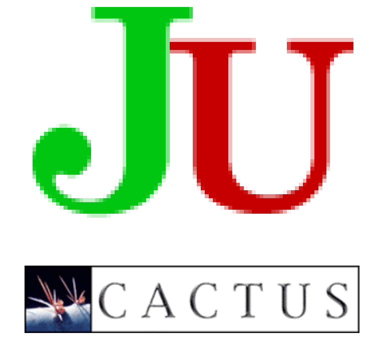
Tutorial

Ant & JUnit

Qualidade e produtividade em projetos Java

Helder da Rocha

www.argonavis.com.br



- *Como resultado deste minicurso você deverá ser capaz de*
  - *Justificar a necessidade e os benefícios de investir na criação e realização de* ***testes de unidade*** *em projetos Java*
  - *Usar o framework* ***JUnit*** *para ajudar a desenvolver código Java de melhor qualidade*
  - *Usar a ferramenta Jakarta* ***Ant*** *para automatizar a construção de seus projetos Java*
  - *Usar o Jakarta* ***Cactus*** *(extensão do JUnit) para testar a integração de aplicações J2EE*

Roteiro deste tutorial

Parte I: Automação de *testes de unidade*

- *Tutorial **JUnit***
- *Extensões do JUnit*
- *Demonstrações*

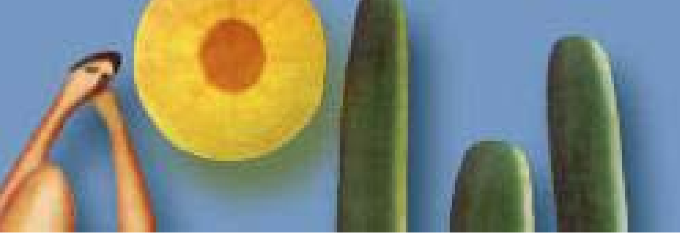
Parte II: Automação do processo de *construção* (build)

- *Tutorial **Ant***
- *Ant & JUnit integrados*
- *Demonstrações*

Parte III: automação de *testes em aplicações Web*

- ***Cactus** e HttpUnit*
- *Conclusões*
- *Fontes*

Código-fonte usado nas demonstrações estará disponível para download!



JU

Automação de

Testes de Unidade

com JUnit

www.junit.org

JUnit: o que é?

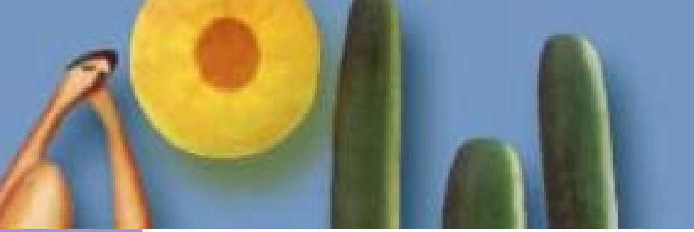
- JUnit é um framework que facilita o desenvolvimento e execução de testes de unidade em código Java
  - Uma API para construir os testes
  - Aplicações para executar testes
- A API
  - Classes Test, TestCase, TestSuite, etc. oferecem a infraestrutura necessária para criar os testes
  - Métodos assertTrue(), assertEquals(), fail(), etc. são usados para testar os resultados
- Aplicação TestRunner
  - Roda testes individuais e suites de testes
  - Versões texto, Swing e AWT.

O que são testes? Para que testar?

- O que são testes automáticos?
  - São programas que avaliam se outro programa funciona como esperado e retornam resposta tipo "sim" ou "não"
  - Ex: um main() que cria um objeto de uma classe testada, chama seus métodos e avalia os resultados
  - Validam os requisitos de um sistema
- Por que testar?
  - Coragem para mudar: é inútil isolar interface da lógica se programador tem medo de mudar a implementação!
  - Qualidade: Código testado é mais confiável
  - Saber quando projeto está pronto: Testes são requisitos 'executáveis'. Escreva-os antes. Quando todos rodarem projeto está pronto!

JUnit: para que serve?

- 'Padrão' para testes de unidade em Java
  - Desenvolvido por Kent Beck (o guru do XP) e Erich Gamma (o G do GoF "Design Patterns")
- Testar é bom mas é chato; JUnit torna as coisas mais agradáveis, facilitando
  - A criação e execução automática de testes
  - A apresentação dos resultados
- JUnit pode verificar se cada método de uma classe funciona da forma esperada
  - Permite agrupar e rodar vários testes ao mesmo tempo
  - Na falha, mostra a causa em cada teste
- Serve de base para extensões



JUnit: Como usar?

- Crie uma classe que estenda junit.framework.TestCase

```
import junit.framework.*;
class SuaClasseTest extends TestCase {...}
```

- Para cada método xxx(args) a ser testado defina um método public void testXxx() no test case
  - SuaClasse:
    - `public boolean equals(Object o) { ... }`
  - SuaClasseTest:
    - `public void testEquals() {...}`
- Sobreponha o método setUp(), se necessário
  - inicialização comum a todos os métodos.
- Sobreponha o método tearDown(), se necessário
  - para liberar recursos, apagar arquivos, etc.

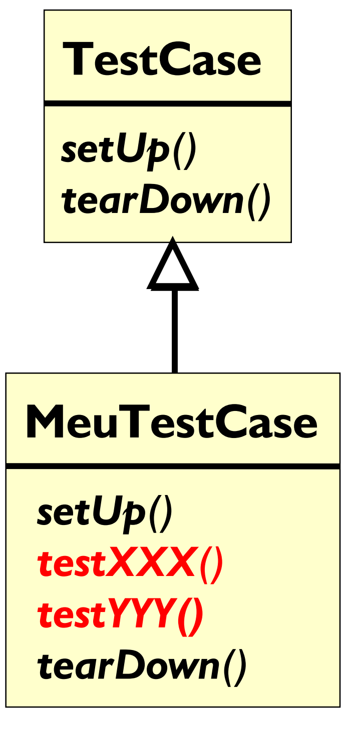
JUnit: Como usar (2)

- *Use testXXX() para testar seu método xxx().*
- *Utilize os métodos de TestCase*
  - *assertEquals(objetoEsperado, objetoRecebido),*
  - *assertTrue(valorBooleano), assertNotNull(objeto)*
  - *assertSame(objetoUm, objetoDois), fail (), ...*
- *Exemplo:*

```
public class CoisaTest extends TestCase {
   // construtor padrão omitido
   private Coisa coisa;
   public void setUp() { coisa = new Coisa("Bit"); }
   public void testToString() {
      assertEquals("<coisa>Bit</coisa>",
                   coisa.toString());
   }
}
```

JUnit: Como funciona?

- O TestRunner recebe uma subclasse de junit.framework.TestCase
  - Usa reflection para descobrir seus métodos
- Para cada método testXXX(), executa:
  - 1. o método setUp()
  - 2. o próprio método testXXX()
  - 3. o método tearDown()
- O test case é instanciado para executar um método testXXX() de cada vez.
  - As alterações que ele fizer ao estado do objeto não afetarão os demais testes
- Método pode **terminar**, **falhar** ou provocar **exceção**



Exemplo: uma classe

```
package junitdemo;
import java.io.*;

public class TextUtils {

  public static String removeWhiteSpaces(String text)
                                        throws IOException {
    StringReader reader = new StringReader(text);
    StringBuffer buffer = new StringBuffer(text.length());
    int c;
    while( (c = reader.read()) != -1) {
      if (c ==' '||c =='\n'||c =='\r'|| c =='\f'||c =='\t') {
         ; /* do nothing */
      } else {
         buffer.append((char)c);
      }
    }
    return buffer.toString();
  }
}
```

veja demonstração

junitdemo.zip

Exemplo: um test case para a classe

```java
package junitdemo;

import junit.framework.*;
import java.io.IOException;

public class TextUtilsTest extends TestCase {

    public TextUtilsTest(String name) {
        super(name);
    }

    public void testRemoveWhiteSpaces() throws IOException {
        String testString = "one, ( two | three+) ,       "+
                            "(((four+ |\t  five)?\n \n,  six?";
        String expectedString = "one,(two|three+)"+
                                ",(((four+|five)?,six?";
        String results = TextUtils.removeWhiteSpaces(testString);
        assertEquals(expectedString, results);
    }
}
```

Construtor precisa ser publico, receber String name e chamar `super(String name)` *

*Método começa com "**test**" e é sempre **public void***

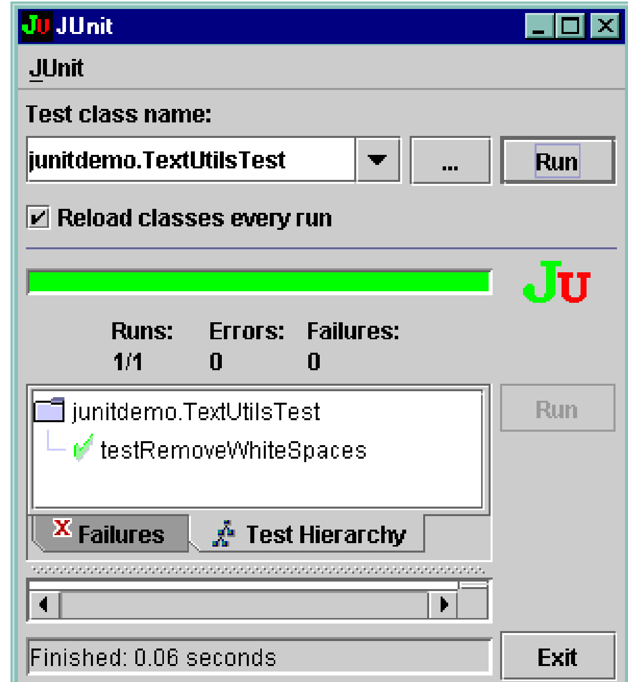
** Pode ser dispensado usando patch (veja www.junit.org)*

Exemplo: como executar

veja demonstração **junitdemo.zip**

- Use a interface de texto
  - *java -cp junit.jar junit.textui.TestRunner junitdemo.TextUtilsTest*
- Ou use a interface gráfica
  - *java -cp junit.jar junit.swingui.TestRunner junitdemo.TextUtilsTest*
- Use Ant <junit>
  - *tarefa do Apache Ant*
- Ou forneça um main():

```
public static void main (String[] args) {
    TestSuite suite =
      new TestSuite(TextUtilsTest.class);
    junit.textui.TestRunner.run(suite);
}
```



- Permite executar uma coleção de testes
  - Método addTest(TestSuite) adiciona um teste na lista
- Padrão de codificação (usando reflection):
  - retornar um TestSuite em cada test-case:

```
public static TestSuite suite() {
    return new TestSuite(SuaClasseTest.class);
}
```

  - criar uma classe AllTests que combina as suites:

```
public class AllTests {
    public static Test suite() {
        TestSuite testSuite =
                         new TestSuite("Roda tudo");
        testSuite.addTest(pacote.AllTests.suite());
        testSuite.addTest(MinhaClasseTest.suite());
        testSuite.addTest(SuaClasseTest.suite());
        return testSuite;
    }
}
```

Pode incluir outras suites

- São os dados reutilizados por vários testes
  - Inicializados no setUp() e destruídos no tearDown() (se necessário)

```
public class AttributeEnumerationTest extends TestCase {
    String   testString;
    String[] testArray;
    AttributeEnumeration testEnum;
    public void setUp() {
        testString  = "(alpha|beta|gamma)";
        testArray = new String[]{"alpha", "beta", "gamma"};
        testEnum = new AttributeEnumeration(testArray);
    }
    public void testGetNames() {
        assertEquals(testEnum.getNames(), testArray);
    }
    public void testToString() {
        assertEquals(testEnum.toString(), testString);
    }
(...)
```

Fixture

- Extensão **JXUnit** (jxunit.sourceforge.net) permite manter dados de teste em arquivo XML (*.jxu) separado do código
  - Mais flexibilidade. Permite escrever testes mais rigorosos, com muitos dados

Teste situações de falha

- *É tão importante testar o cenário de falha do seu codigo quanto o sucesso*
- *Método fail() provoca uma falha*
  - *Use para verificar se exceções ocorrem quando se espera que elas ocorram*
- *Exemplo*

```
public void testEntityNotFoundException() {
   resetEntityTable(); // no entities to resolve!
   try {
      // Following method call must cause exception!
      ParameterEntityTag tag = parser.resolveEntity("bogus");
      fail("Should have caused EntityNotFoundException!");
   } catch (EntityNotFoundException e) {
       // success: exception occurred as expected
   }
}
```

Afirmações do J2SDK1.4 (assertions)

- *São expressões booleanas que o programador define para afirmar uma condição que ele acredita ser verdade*
  - *Afirmações são usadas para validar código (ter a certeza que um vetor tem determinado tamanho, ter a certeza que o programa não passou por determinado lugar)*
  - *Melhoram a qualidade do código: tipo de teste*
  - *Devem ser usadas durante o desenvolvimento e desligadas na produção (afeta a performance)*
  - *Não devem ser usadas como parte da lógica do código*
- *Afirmações são um recurso novo do JSDK1.4.0*
  - *Nova palavra-chave:* `assert`
  - *É preciso compilar usando a opção -source 1.4:*
    ```
    > javac -source 1.4 Classe.java
    ```
  - *Para executar, é preciso habilitar afirmações (enable assertions):*
    ```
    > java -ea Classe
    ```

JUnit vs. afirmações

- *Afirmações do J2SDK 1.4 são usadas dentro do código*
  - *Podem incluir testes dentro da lógica procedural de um programa*

```
if (i%3 == 0) {
    doThis();
} else if (i%3 == 1) {
    doThat();
} else {
    assert i%3 == 2: "Erro interno!";
}
```

  - *Provocam um AssertionError quando falham (que pode ser encapsulado pelas exceções do JUnit)*
- *Afirmações do JUnit são usadas em classe separada (TestCase)*
  - *Não têm acesso ao interior dos métodos (verificam se a interface dos métodos funciona como esperado)*
- *Afirmações do J2SDK1.4 e JUnit são complementares*
  - *JUnit testa a interface dos métodos*
  - *assert testa trechos de lógica dentro dos métodos*

Boas práticas: metodologia "test-first"

- Testes geralmente ...
  - ... são mais simples que código a ser testado
  - ... refletem com clareza o que se espera do código,

portanto, devem ser escritos **antes** do código!

- Testes definem com precisão o que precisa ser feito
  - Estabelece uma meta clara para cada unidade de código
  - Evita que se perca tempo desenvolvendo o que é desnecessário
- Desenvolvimento usando metodologia "test-first"
  1. Escreva o esqueleto da sua classe (métodos vazios)
  2. Escreva a sua classe de teste e implemente todos os testes (um para cada método ou condição importante)
  3. Rode os testes. Todos os testes devem falhar
  4. Implemente uma unidade de código e rode os testes

➡ Quando todos os testes rodarem com sucesso a sua classe está pronta!

Como escrever bons testes

- JUnit facilita bastante a criação e execução de testes, mas elaborar bons testes exige mais
  - O que testar? Como saber se testes estão completos?
- "Teste tudo o que pode falhar" [2]
  - Métodos triviais (get/set) não precisam ser testados.
  - Será? E se houver uma rotina de validação no método set?
  - Métodos get/set bem feitos não falham (não devem conter lógica)!
- É melhor ter testes a mais que testes a menos
  - Use assertNotNull() sempre que puder (reduz drasticamente erros de NullPointerException difíceis de encontrar)
  - Reescreva seu código para que fique mais fácil de testar
  - Escreva um teste para cada afirmação assertXXX()
- Bugs revelam testes
  - Achou um bug? Não conserte sem antes escrever um teste que o pegue (se você não o fizer, ele volta)!

Como lidar com testes difíceis

- Testes devem ser simples e suficientes
  - **XP**: design mais simples que resolva o problema; sempre pode-se escrever novos testes, quando necessário
- Não complique
  - Não teste o que é responsabilidade de outra classe/método
  - Assuma que outras classes e métodos funcionam
- Testes difíceis (ou que parecem difíceis)
  - Aplicações gráficas: eventos, layouts, threads
  - Objetos inaccessíveis, métodos privativos, Singletons
  - Objetos que dependem de outros objetos
  - Objetos cujo estado varia devido a fatores imprevisíveis
- Soluções
  - Alterar o design da aplicação para facilitar os testes
  - Simular dependências usando proxies e stubs

Testes de interface

- Caso específico: resposta de servidores Web
  - Verificar se uma página HTML ou XML contém determinado texto ou determinado elemento
  - Verificar se resposta está de acordo com dados passados na requisição: testes funcionais tipo "caixa-preta"
- Soluções (extensões do JUnit)
  - **HttpUnit** e **ServletUnit**:
    - permite testar dados de árvore DOM HTML gerada
  - **JXWeb** (combinação do JXUnit com HttpUnit)
    - permite especificar os dados de teste em arquivos XML
    - arquivos de teste Java são gerados a partir do XML
  - **XMLUnit**
    - extensão simples para testar árvores XML
  - Onde encontrar: (httpunit|jxunit|xmlunit).sourceforge.net

Testes de performance

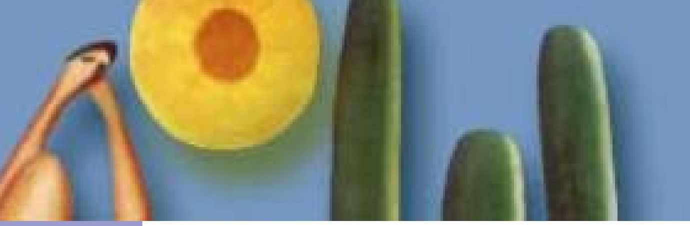
- **JUnitPerf** (www.clarkware.com)
  - Coleção de decoradores para medir performance e escalabilidade em testes JUnit existentes
- TimedTest
  - Executa um teste e mede o tempo transcorrido
  - Define um tempo máximo para a execução. Teste falha se execução durar mais que o tempo estabelecido
- LoadTest
  - Executa um teste com uma carga simulada
  - Utiliza timers para distribuir as cargas usando distribuições randômicas
  - Combinado com TimerTest para medir tempo com carga
- ThreadedTest
  - Executa o teste em um thread separado

Testes: conclusões

- *É possível desenvolver software de qualidade com um investimento mínimo em ferramentas*
  - *Há ótimas opções de ferramentas open-source*
  - *Ferramentas tornam mais fácil a adoção de práticas como "test-first" e "refactoring" que aumentam a qualidade do software*
- *JUnit é muito simples, é de graça, mas vale muito!*
  - *"Never in the field of software development was so much owed by so many to so few lines of code" Martin Fowler, sobre o JUnit*
  - *Principais ferramentas comerciais incluem o JUnit: Together Control Center, Sun Forté for Java, IBM Websphere Studio, etc.*
- *Vale a pena investir tempo para desenvolver e aperfeiçoar a prática constante de escrever testes com o JUnit*
  - *mais produtividade, maior integração de equipes*
  - *produtos de melhor qualidade, com prazo previsível*
  - *menos stress, mais organização*

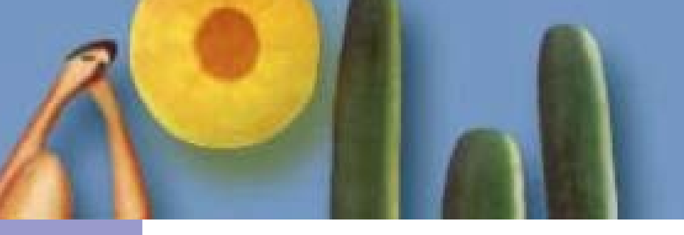
Construção de aplicações com Apache Ant

jakarta.apache.org/ant/



Ant: o que é?

- *Ferramenta para construção de aplicações*
  - *Implementada em Java*
  - *Baseada em roteiros XML*
  - *Extensível (via scripts ou classes)*
  - *'padrão' do mercado*
  - *Open Source (Grupo Apache, Projeto Jakarta)*
- *Semelhante a make, porém*
  - *Mais simples e estruturada (XML)*
  - *Mais adequada a tarefas comuns em projetos Java*
  - *Independente de plataforma*



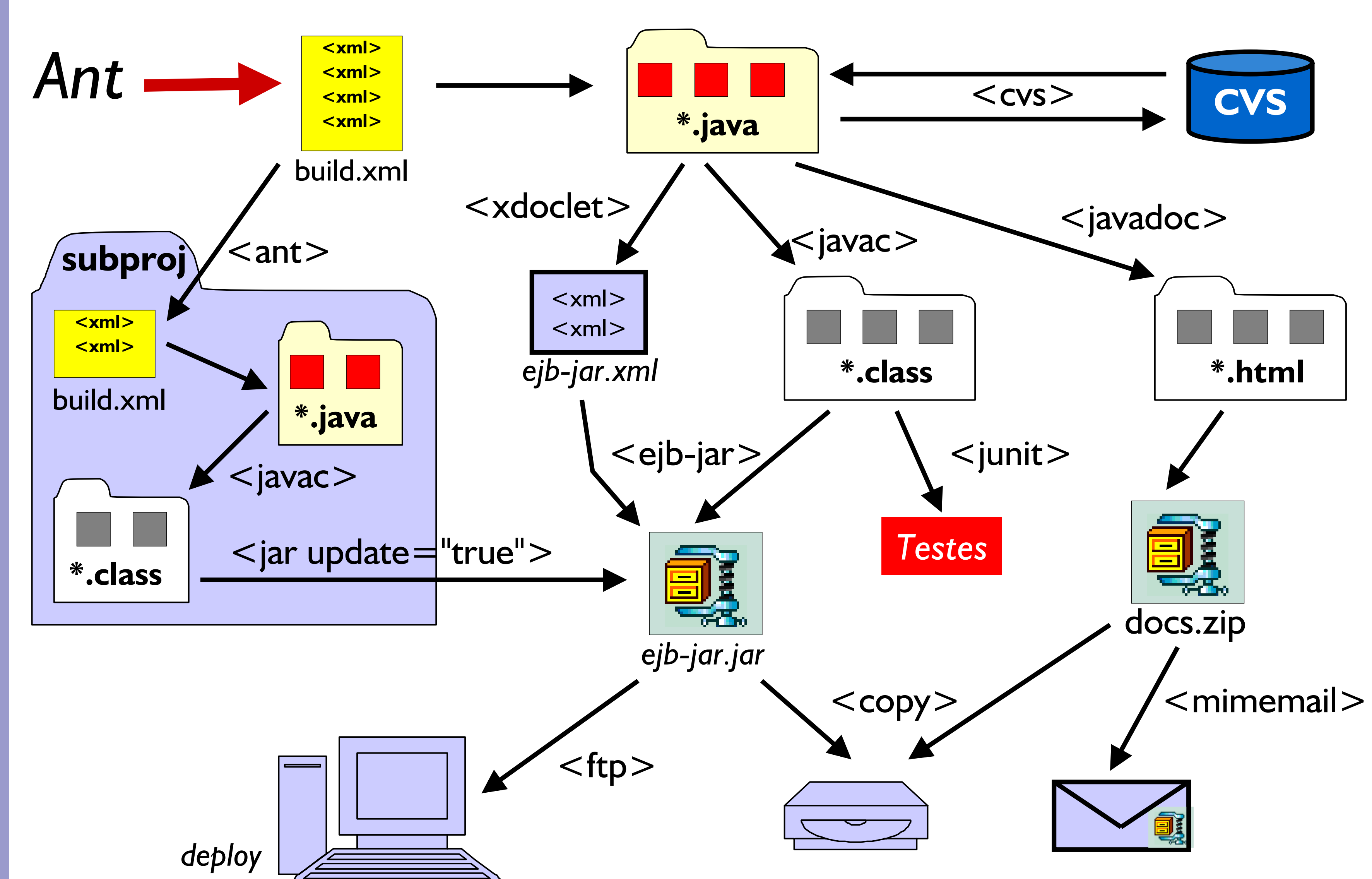
Para que serve?

- *Para montar praticamente qualquer aplicação Java que consista de mais que meia dúzia de classes; Aplicações*
  - *distribuídas em pacotes*
  - *que requerem a definição de classpaths locais, e precisam vincular código a bibliotecas (JARs)*
  - *cuja criação/instalação depende de mais que uma simples chamada ao javac. Ex: RMI, CORBA, EJB, servlets, JSP,...*
- *Para automatizar processos frequentes*
  - *Javadoc, XSLT, implantação de serviços Web e J2EE (deployment), CVS, criação de JARs, testes, FTP, email*

Como funciona?

- Ant executa roteiros escritos em XML: 'buildfiles'
- Cada projeto do Ant possui um buildfile
  - subprojetos podem ter, opcionalmente, buildfiles adicionais chamados durante a execução do primeiro
- Cada projeto possui uma coleção de alvos
- Cada alvo consiste de uma seqüência de tarefas
- Exemplos de execução
  - **`ant`**
    - procura build.xml no diretório atual e roda alvo default
  - **`ant -buildfile outro.xml`**
    - executa alvo default de arquivo outro.xml
  - **`ant compilar`**
    - roda alvo 'compilar' e possíveis dependências em build.xml

Como funciona (2)



Buildfile

- O buildfile é um arquivo XML: **build.xml** (default)
- Principais elementos

```
<project default="alvo_default">
```

  - Elemento raiz (obrigatório): define o projeto.

```
<target name="nome_do_alvo">
```

  - Coleção de tarefas a serem executadas em seqüência
  - Deve haver pelo menos um <target>

```
<property name="nome" value="valor">
```

  - pares nome/valor usados em atributos dos elementos do build.xml da forma `${nome}`
  - propriedades também podem ser definidas em linha de comando (`-Dnome=valor`) ou lidas de arquivos externos (atributo `file`)
- tarefas (mais de 130) - dentro dos alvos.
  - `<javac>`, `<jar>`, `<java>`, `<copy>`, `<mkdir>`, ...

Buildfile (2)

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!-- Compila diversos arquivos .java -->

<project default="compile" basedir=".">
  <property name="src.dir" value="${basedir}/src" />
  <property name="build.dir" value="build" />
  <target name="init">
    <echo> Criando diretório </echo>
    <mkdir dir="${build.dir}" />
  </target>
  <target name="compile" depends="init"
          description="Compila os arquivos-fonte">
    <javac srcdir="${src.dir}" destdir="${build.dir}">
      <classpath>
        <pathelement location="${build.dir}" />
      </classpath>
    </javac>
  </target>
</project>
```

Propriedades

Alvos

Tarefas

- *Executando buildfile da página anterior*

```
C:\usr\palestra\antdemo> ant
Buildfile: build.xml

init:
     [echo]  Criando diretório
     [mkdir] Created dir:
     C:\usr\palestra\antdemo\build

compile:
     [javac] Compiling 2 source files to
     C:\usr\palestra\antdemo\build

BUILD SUCCESSFUL
Total time: 4 seconds
C:\usr\palestra\antdemo>
```

ANTES

```
build.xml
src
  argonavis
    util
      Tiracentos.java
      TiracentosTest.java
```

DEPOIS

```
build.xml
src
  argonavis
    util
      Tiracentos.java
      TiracentosTest.java
build
  argonavis
    util
      Tiracentos.class
      TiracentosTest.class
```

veja demonstração

antdemo.zip

O que se pode fazer com Ant?

- Compilar.
  - <javac>, <csc>
- Gerar documentação
  - <javadoc>, <junitreport>, <style>, <stylebook>
- Gerar código (XDoclet)
  - <ejbdoclet>, <webdoclet>
- Executar programas
  - <java>, <apply>, <exec> <ant>, <sql>
- Empacotar e comprimir
  - <jar>, <zip>, <tar>, <war>, <ear>, <cab>
- Expandir, copiar, instalar
  - <copy>, <delete>, <mkdir>, <unjar>, <unwar>, <unzip>
- Acesso remoto
  - <ftp>, <telnet>, <cvs>, <mail>, <mimemail>
- Montar componentes
  - <ejbc>, <ejb-jar>, <rmic>
- Testar unidades de código
  - <junit>
- Criar novas tarefas
  - <taskdef>
- Executar roteiros e sons
  - <script>, <sound>

- **<javac>**: *Chama o compilador Java*

```
<javac srcdir="dirfontes" destdir="dirbuild" >
    <classpath>
        <pathelement path="arquivo.jar" />
        <pathelement path="/arquivos" />
    </classpath>
    <classpath idref="extra" />
</javac>
```

- **<jar>**: *Monta um JAR*

```
<jar destfile="bin/programa.jar">
   <manifest>
      <attribute name="Main-class"
                 value="exemplo.main.Exec">
   </manifest>
   <fileset dir="${build.dir}"/>
</jar>
```

Tarefas do sistema de arquivos

- **<mkdir>**: *cria diretórios*

```
<mkdir dir="diretorio" />
```

- **<copy>**: *copia arquivos*

```
<copy todir="dir" file="arquivo" />
<copy todir="dir">
    <fileset dir="fonte"
             includes="*.txt" />
</copy>
```

- **<delete>**: *apaga arquivos*

```
<delete file="arquivo />
<delete dir="diretorio"/>
```

Geração de documentação

- **<javadoc>**: *Gera documentação do código-fonte.*
  - *Alvo abaixo gera documentação e exclui classes que contém 'Test.java'*

```
<target name="generate-docs">
  <mkdir dir="docs/api"/>
  <copy todir="tmp">
    <fileset dir="${src.dir}">
      <include name="**/*.java" />
      <exclude name="**/**Test.java" />
    </fileset>
  </copy>
  <javadoc destdir="docs/api"
           packagenames="argonavis.*"
           sourcepath="tmp" />
  <delete dir="tmp" />
</target>
```

Propriedades

- *Podem ser definidas com* **<property>**

```
<property name="app.nome" value="jmovie" />
```

- *Podem ser carregadas de um arquivo*

```
<property file="c:/conf/arquivo.conf" />
```

```
app.ver=1.0
docs.dir=c:\docs\
codigo=15323
```

arquivo.conf

- *Podem ser passadas na linha de comando*

```
c:\> ant -Dautor=Wilde
```

- *Para recuperar o valor, usa-se* ${nome}

```
<jar destfile="${app.nome}-${app.ver}.jar"/>
<echo message="O autor é ${autor}" />
<mkdir dir="build${codigo}" />
```

Propriedades especiais

- **<tstamp>**: Grava um instante
  - A hora e data podem ser recuperados como propriedades
    - `${TSTAMP}` hhmm 1345
    - `${DSTAMP}` aaaammdd 20020525
    - `${TODAY}` dia mes ano 25 May 2002
  - Novas propriedades podem ser definidas, locale, etc.
  - Uso típico: `<tstamp/>`
- **<property** environment="env"**>**: Propriedade de onde se pode ler variáveis de ambiente do sistema
  - Dependende de plataforma

```
<target name="init">
  <property environment="env"/>
  <property name="j2ee.home"
            value="env.J2EE_HOME" />
</target>
```

Tipos de dados: arquivos e diretórios

- **<fileset>**: *árvore de arquivos e diretórios*
  - *Conteúdo do conjunto pode ser reduzido utilizando elementos <include> e <exclude>*
  - *Usando dentro de tarefas que manipulam com arquivos e diretórios como <copy>, <zip>, etc.*

```
<copy todir="${build.dir}/META-INF">
   <fileset dir="${xml.dir}" includes="ejb-jar.xml"/>
   <fileset dir="${xml.dir}/jboss">
      <include name="*.xml" />
      <exclude name="*-orig.xml" />
   </fileset>
</copy>
```

- **<dirset>**: *árvore de diretórios*
  - *Não inclui arquivos individuais*

Tipos de dados: coleções

- **<patternset>**: representa coleção de padrões

```
<patternset id="project.jars" >
  <include name="**/*.jar"/>
  <exclude name="**/*-test.jar"/>
</patternset>
```

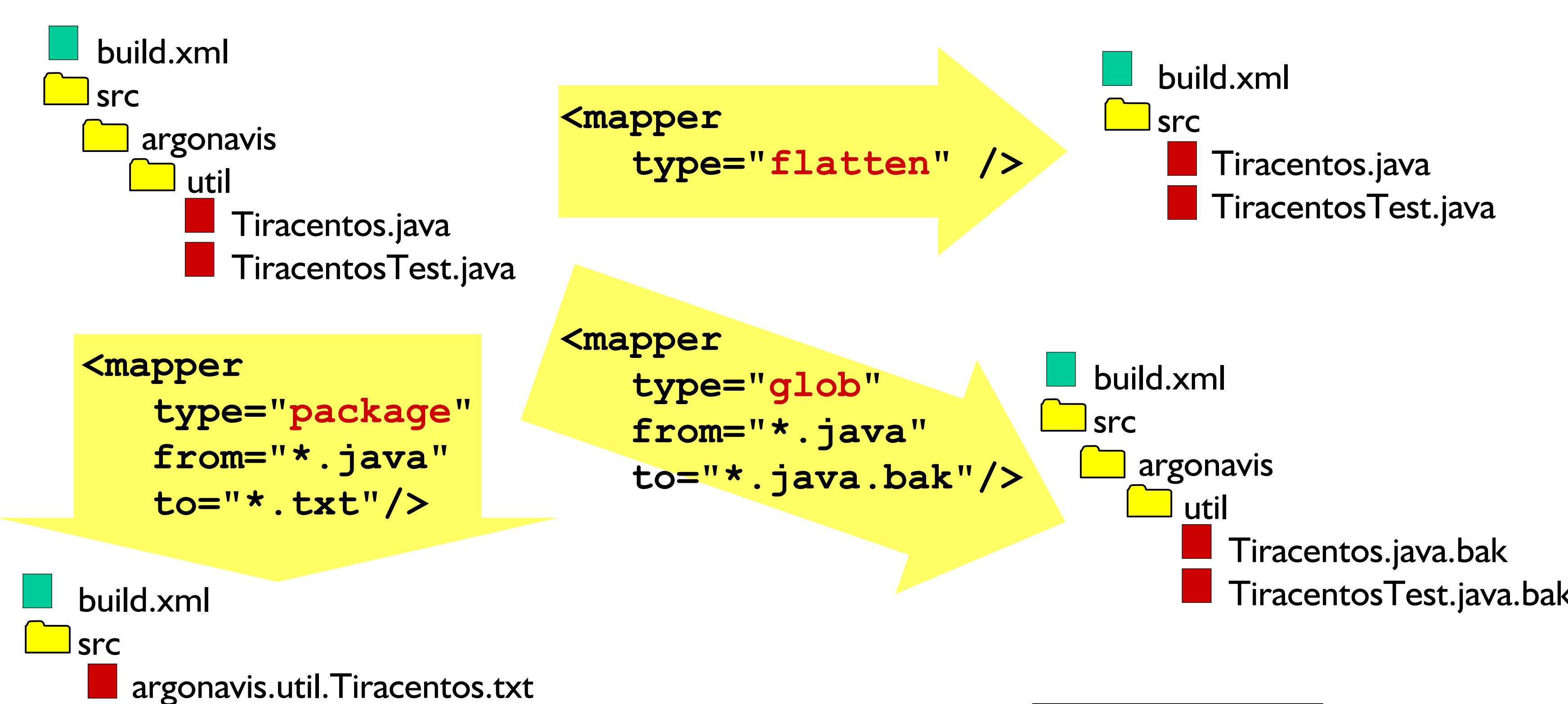
- **<path>**: representa uma coleção de caminhos
  - Associa um ID a grupo de arquivos ou caminhos

```
<path id="server.path">
  <pathelement path="${j2ee.home}/lib/locale" />
  <fileset dir="${j2ee.home}/lib">
    <patternset refid="project.jars" />
  </fileset>
</path>

<target name="compile" depends="init">
   <javac destdir="${build.dir}" srcdir="${src.dir}">
      <classpath refid="server.path" />
   </javac>
</target>
```

Tipos de dados: File Mapper

- **<mapper>**: *altera nomes de arquivos durante cópias ou transformações*
  - *Seis tipos: identity, flatten, merge, regexp, glob, package*



veja exemplos *mappertest.zip*

Tipos de dados: seletores

- Permitem a seleção dos elementos de um fileset usando critérios além dos definidos por <include> e <exclude>
- Sete seletores básicos (pode-se criar novos)
  - **<contains>** - Seleciona arquivos que contém determinado texto
  - **<date>** - Arquivos modificados antes ou depois de certa data
  - **<depend>** - Seleciona arquivos cuja data de modificação seja posterior a arquivos localizados em outro lugar
  - **<depth>** - Seleciona arquivos encontrados até certa profundidade de uma árvore de diretórios
  - **<filename>** - Equivalente ao include e exclude
  - **<present>** - Seleciona arquivo com base na sua (in)existência
  - **<size>** - Seleciona com base no tamanho em bytes

Exemplo: *Seleciona arquivos do diretório "fonte" que também estão presentes em "destino"*

```
<fileset dir="fonte">
  <present targetdir="destino"/>
</fileset>
```

Tipos de dados: filtros

- **<filter>** e **<filterset>**: *Permite a substituição de padrões em arquivos durante a execução de uma tarefa*
  - *Caractere default: @*
- *Exemplo: a tarefa abaixo irá substituir todas as ocorrências de* `@javahome@` *por* `c:\j2sdk1.4.0` *nos arquivos copiados*

```
<copy todir="${dest.dir}">
   <fileset dir="${src.dir}"/>
   <filterset>
      <filter token="javahome" value="c:\j2sdk1.4.0"/>
   </filterset>
</copy>
```

- *Pares token=valor podem ser carregados de arquivo:*

```
<filterset>
   <filtersfile file="build.properties" />
</filterset>
```

```
<ear destfile="app.ear" appxml="application.xml">
  <fileset dir="${build}" includes="*.jar,*.war"/>
</ear>

<ejbjar srcdir="${build}" descriptordir="${xml.dir}" ... >
  <jboss destdir="${deployjars.dir}" />
</ejbjar>
```

Há suporte aos principais servidores

```
<war warfile="bookstore.war" webxml="meta/metainf.xml">
   <fileset dir="${build}/${bookstore2}" >
      <include name="*.tld" />
      <exclude name="web.xml" />
   </fileset>
   <classes dir="${build}" >
      <include name="database/*.class" />
   </classes>
   <lib dir="bibliotecas" />
   <webinf dir="jboss-web.xml" />
</war>
```

WEB-INF/web.xml

Raiz do WAR

WEB-INF/classes

WEB-INF/lib

WEB-INF/

<ejbdoclet> e <webdoclet>: Geram código

- *Requer JAR de* xdoclet.sourceforge.net
- *Ideal para* *geração automática de arquivos de configuração (web.xml, ejb-jar.xml, application.xml, taglibs, struts-config, etc.) e código-fonte (beans, value-objects)*

```
<ejbdoclet sourcepath="src" destdir="${build.dir}"
           classpathref="xdoclet.path" ejbspec="2.0">
   <fileset dir="src">
      <include name="**/*Bean.java" />
   </fileset>
   <remoteinterface/>
   <homeinterface/>
   <utilobject/>
   <entitypk/>
   <entitycmp/>
   <deploymentdescriptor destdir="${dd.dir}"/>
   <jboss datasource="java:/OracleDS" />
</ejbdoclet>
```

XDoclet faz muito mais que isto!

Detalhes da configuração do componente estão em arquivos de template externos

- **<java>**: *roda o interpretador Java*

```
<target name="runrmiclient">
  <java classname="hello.rmi.HelloClient" fork="true">
    <jvmarg value="-Djava.security.policy=rmi.policy"/>
    <arg name="host" value="${remote.host}" />
    <classpath refid="app.path" />
  </java>
</target>
```

- **<exec>**: *executa um comando do sistema*

```
<target name="orbd">
  <exec executable="${java.home}\bin\orbd">
    <arg line="-ORBInitialHost ${nameserver.host}"/>
  </exec>
</target>
```

- **<apply>**: *semelhante a <exec> mas usado em executáveis que operam sobre outros arquivos*

Tarefas de rede

- ***<ftp>**: Realiza a comunicação com um servidor FTP remoto para upload ou download de arquivos*
  - *Tarefa opcional que requer NetComponents.jar (http://www.savarese.org)*

```
<target name="remote.jboss.deploy" depends="dist">
    <ftp server="${ftp.host}" port="${ftp.port}"
         remotedir="/jboss/server/default/deploy"
         userid="admin" password="jboss"
         depends="yes" binary="yes">
        <fileset dir="${basedir}">
            <include name="*.war"/>
            <include name="*.ear"/>
            <include name="*.jar"/>
        </fileset>
    </ftp>
</target>
```

- *<style>: Transforma documentos XML em outros formatos usando folha de estilos XSLT (nativa)*
  - *Usa Trax (default), Xalan ou outro transformador XSL*

```
<style basedir="xmldocs"
       destdir="htmldocs"
       style="xmltohtml.xsl" />
```

veja demonstração

styledemo.zip

  - *Elemento <param> passa valores para elementos <xsl:param> da folha de estilos*

```
<style in="cartao.xml"
       out="cartao.html"
       style="cartao2html.xsl">
    <param name="docsdir"
           expression="/cartoes"/>
</style>
```

build.xml

```
(...)
<xsl:param name="docsdir"/>
(...)
<xsl:valueof select="$docsdir"/>
(...)
```

cartao2html.xsl

- *<sound>: define um par de arquivos de som para soar no sucesso ou falha de um projeto*
  - *Tarefa opcional que requer Java Media Framework*
- *Exemplo:*
  - *No exemplo abaixo, o som* frog.wav *será tocado quando o build terminar sem erros fatais.* Bark.wav *tocará se houver algum erro que interrompa o processo:*

```
<target name="init">
  <sound>
    <success source="C:/Media/frog.wav"/>
    <fail source="C:/Media/Bark.wav"/>
  </sound>
</target>
```

- **<sql>**: *Comunica-se com banco de dados através de driver JDBC*

```
<property name="jdbc.url"
     value="jdbc:cloudscape:rmi://server:1099/Cloud" />
<target name="populate.table">
    <sql driver="COM.cloudscape.core.RmiJdbcDriver"
         url="${jdbc.url}"
         userid="helder"
         password="helder"
         onerror="continue">
        <transaction src="droptable.sql" />
        <transaction src="create.sql" />
        <transaction src="populate.sql" />
        <classpath refid="jdbc.driver.path" />
    </sql>
</target>
```

- *Viabiliza a integração contínua:*
  - *Pode-se executar todos os testes após a integração com um único comando:*
  - **`ant roda-testes`**
- *Com as tarefas <junit> e <junitreport> é possível*
  - *executar todos os testes*
  - *gerar um relatório simples ou detalhado, em diversos formatos (XML, HTML, etc.)*
  - *executar testes de integração*
- *São tarefas opcionais. É preciso ter no* **`$ANT_HOME/lib`**
  - *optional.jar (distribuído com Ant)*
  - *junit.jar (distribuído com JUnit)*

```
<target name="test" depends="build">
    <junit printsummary="true" dir="${build.dir}"
           fork="true">
       <formatter type="plain" usefile="false" />
       <classpath path="${build.dir}" /
       <test name="argonavis.dtd.AllTests" />
    </junit>
</target>
```

Formata os dados na tela (plain)
Roda apenas arquivo AllTests

```
<target name="batchtest" depends="build" >
    <junit dir="${build.dir}" fork="true">
       <formatter type="xml" usefile="true" />
       <classpath path="${build.dir}" />
       <batchtest todir="${test.report.dir}">
          <fileset dir="${src.dir}">
             <include name="**/*Test.java" />
             <exclude name="**/AllTests.java" />
          </fileset>
       </batchtest>
    </junit>
</target>
```

Gera arquivo XML
Inclui todos os arquivos que terminam em TEST.java

Brasil@JavaOne

<junitreport>

- *Gera um relatório detalhado (estilo JavaDoc) de todos os testes, sucessos, falhas, exceções, tempo, ...*

```
<target name="test-report" depends="batchtest" >
  <junitreport todir="${test.report.dir}">
    <fileset dir="${test.report.dir}">
       <include name="TEST-*.xml" />
    </fileset>
    <report todir="${test.report.dir}/html"
            format="frames" />
  </junitreport>
</target>
```

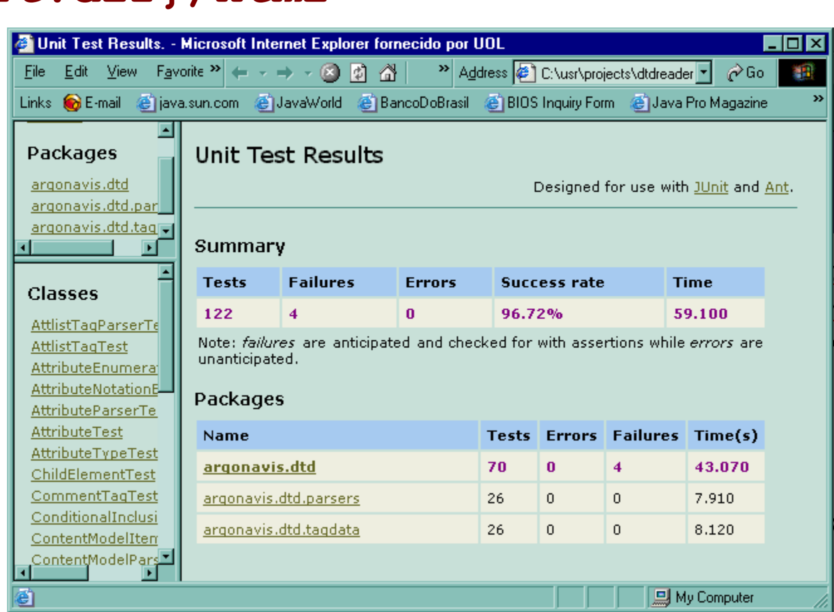
Usa arquivos XML gerados por <formatter>

veja demonstração

junitdemo.zip

veja demonstração

dtdreader.zip



Ant programável

- *Há duas formas de estender o Ant com novas funções*
  - *Implementar roteiros usando JavaScript*
  - *Criar novas tarefas reutilizáveis*
- *A tarefa <script> permite embutir JavaScript em um buildfile. Pode-se*
  - *realizar operações aritméticas e booleanas*
  - *utilizar estruturas como if/else, for, foreach e while*
  - *manipular com os elementos do buildfile usando DOM*
- *A tarefa <taskdef> permite definir novas tarefas*
  - *tarefa deve ser implementada em Java e estender Task*
  - *método execute() contém código de ação da tarefa*
  - *cada atributo corresponde a um método setXXX()*

veja demonstração **scriptdemo.zip**

veja exemplos **foptask.zip**

veja demonstração **taskdemo.zip**

Integração com outras aplicações

- Ant provoca vários eventos que podem ser capturados por outras aplicações
  - Útil para implementar integração, enviar notificações por email, gravar logs, etc.
- Eventos
  - Build iniciou/terminou
  - Alvo iniciou/terminou
  - Tarefa iniciou/terminou
  - Mensagens logadas
- Vários listeners e loggers pré-definidos
  - Pode-se usar ou estender classe existente.
  - Para gravar processo (build) em XML:

```
ant -listener org.apache.tools.ant.XmlLogger
```

Integração com editores e IDEs

- Produtos que integram com Ant e oferecem interface gráfica e eventos para buildfiles:
  - Antidote: GUI para Ant (do projeto Jakarta)
    - http://cvs.apache.org/viewcvs/jakarta-ant-antidote/
  - JBuilder (AntRunner plug-in)
    - http://www.dieter-bogdoll.de/java/AntRunner/
  - NetBeans e Forté for Java
    - http://ant.netbeans.org/
  - Visual Age for Java (integração direta)
  - JEdit (AntFarm plug-in)
    - http://www.jedit.org
  - Jext (AntWork plug-in)
    - ftp://jext.sourceforge.net/pub/jext/plugins/AntWork.zip

Ant: conclusões

- Vale a pena aprender e ganhar experiência com o Ant
  - É bom, é de graça e todo mundo usa!
  - Facilita a compilação, depuração, execução, montagem, instalação, documentação e utilização de qualquer aplicação Java
  - Faz ainda transformação XSLT, geração de código e qualquer outra tarefa que o programador desejar
  - Você pode integrá-lo em sua aplicação. O código é aberto!
  - É mais fácil que make. É melhor que usar arquivos .bat e .sh
  - É independente de IDE e plataforma
- Use Ant mesmo que seu IDE já possua um "build"
  - Ant oferece muito mais recursos que qualquer comando "build" dos IDEs existentes hoje, é extensível e deixa seu projeto independente de um IDE específico
  - Os principais fabricantes de IDEs Java suportam Ant ou possuem plug-ins para integração com Ant

Testes em aplicações J2EE com Apache Cactus

jakarta.apache.org/cactus/

Cactus: o que é?

- É um framework que oferece facilidades para testar
  - servlets
  - JSP custom tags
  - filtros de servlets
- Produto Open Source do projeto Jakarta
  - Metas de curto prazo: testar componentes acima + EJB
  - Metas de longo prazo: oferecer facilidades para testar todos os componentes J2EE; ser o framework de referência para testes in-container.
- Cactus estende o JUnit framework
  - Execução dos testes é realizada de forma idêntica
  - TestCases são construídos sobre uma subclasse de junit.framework.TestCase

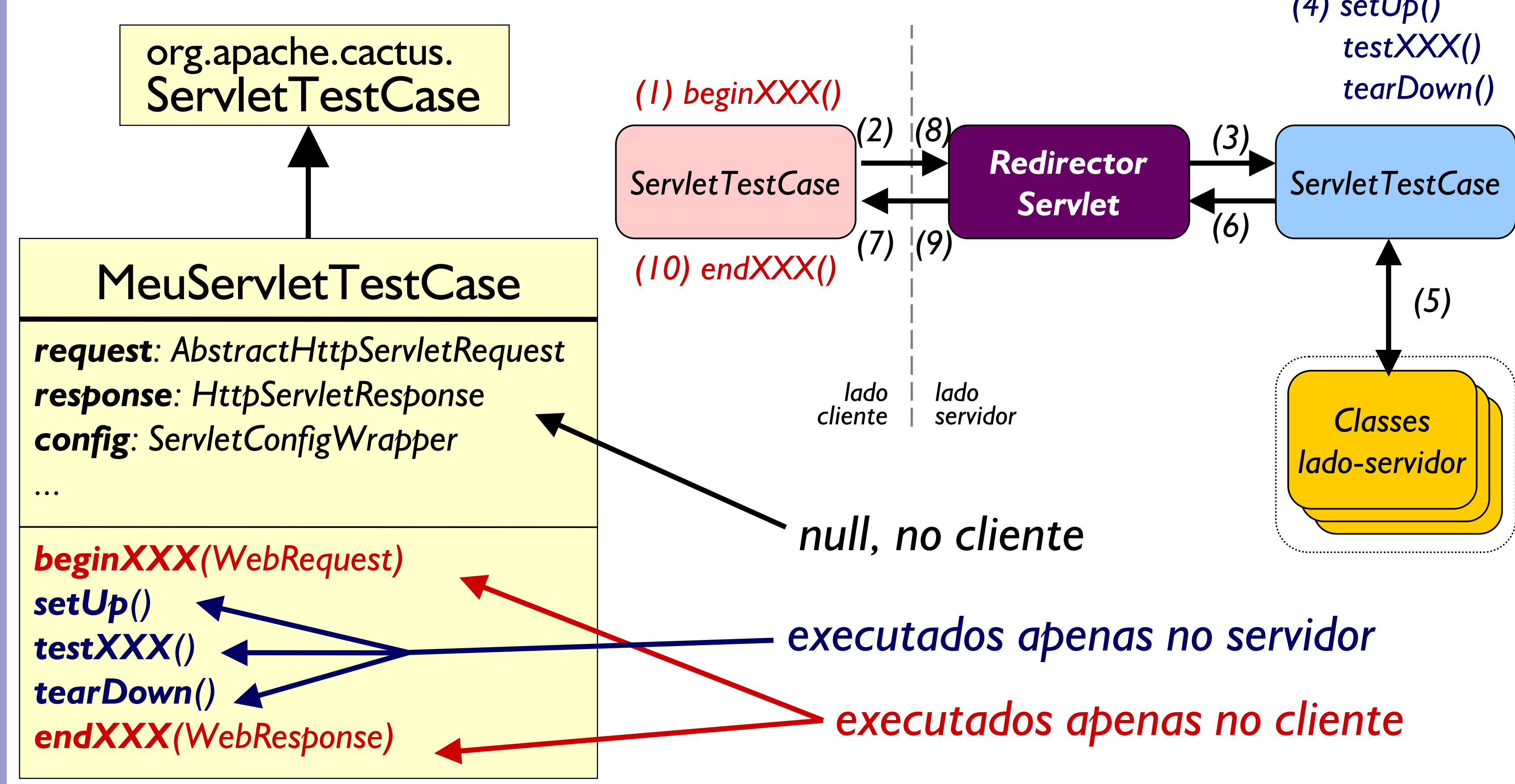
Cactus: para que serve?

- Cactus pode ser usado para testar aplicações que usam componentes
  - servlets
  - JSPs (indiretamente, na versão. 1.3)
  - filtros
  - custom tags (elementos JSP definidas pelo usuário)
  - EJB (indiretamente, v. 1.3)
- Cactus testa a integração desses componentes com seus containers
  - não usa stubs - usa o próprio container como servidor e usa JUnit como cliente
  - comunicação é intermediada por um proxy

Cactus: como funciona?

- *Cactus utiliza os test cases simultaneamente no cliente e no servidor: duas cópias*
  - *Uma cópia é instanciada pelo servlet container*
  - *Outra cópia é instanciada pelo JUnit*
- *Comunicação com o servlet container é feita através de um proxy (XXXRedirector)*
  - *JUnit envia requisições via HTTP para proxy*
  - *Proxy devolve resultado via HTTP e JUnit os mostra*
- *Há, atualmente (Cactus 1.3) três tipos de proxies:*
  - *ServletRedirector: para testar servlets*
  - *JSPRedirector: para testar JSP custom tags*
  - *FilterRedirector: para testar filtros de servlets*

- *Parte da mesma classe (ServletTestCase) é executada no cliente, parte no servidor*



ServletTestCase (ou similar)

- Para cada método XXX() a ser testado, pode haver:
  - Um `beginXXX()`, para inicializar a requisição do cliente
    - encapsulada em um objeto WebRequest a ser enviado ao servidor
  - Um `testXXX()`, para testar o funcionamento do método no servidor (deve haver ao menos um)
  - Um `endXXX()`, para verificar a resposta do servidor
    - devolvida em um objeto WebResponse retornada pelo servidor
- Além desses três métodos, cada TestCase pode conter
  - `setUp()`, opcional, para inicializar objetos no servidor
  - `tearDown()`, opcional, para liberar recursos no servidor
- Os métodos do lado do servidor têm acesso aos mesmos objetos implícitos disponíveis em um servlet ou página JSP: request, response, etc.

Cactus: exemplo

- Veja cactusdemo.zip (distribuído com esta palestra)
  - Usa duas classes: um servlet (MapperServlet) e uma classe (SessionMapper) que guarda cada parâmetro como atributo da sessão e em um HashMap - veja fontes em src/xptoolkit/cactus
- Para rodar, configure o seu ambiente:
  - build.properties - localização dos JARs usados pelo servidor Web (CLASSPATH do servidor)
  - runtests.bat (para Windows) e runtests.sh (para Unix) - localização dos JARs usados pelo JUnit (CLASSPATH do cliente)
  - lib/client.properties (se desejar rodar cliente e servidor em máquinas separadas, troque as ocorrências de localhost pelo nome do servidor)
- Para montar, execute:
  - 1. `ant test-deploy` instala cactus-tests.war no tomcat
  - 2. o servidor (Tomcat 4.0 startup)
  - 3. `runtests.bat` roda os testes no JUnit

veja demonstração

cactusdemo.zip

CactusDemo: testes

MapperServletTest.java

```
public class MapperServletTest extends ServletTestCase { (...)
  private MapperServlet servlet;
  public void beginDoGet(WebRequest cSideReq) {
     cSideReq.addParameter("user", "Jabberwock");
  }
  public void setUp() throws ServletException {
    this.config.setInitParameter("ALL_CAPS", "true");
    servlet = new MapperServlet();
    servlet.init(this.config);
  }
  public void testDoGet() throws IOException {
    servlet.doGet(this.request, this.response);
    String value = (String) session.getAttribute("user");
    assertEquals("Jabberwock", value);
  }
  public void tearDown() { /* ... */ }
  public void endDoGet(WebResponse cSideResponse) {
    String str = cSideResponse.getText();
    assertTrue(str.indexOf("USER</b></td><td>JABBERWOCK") > -1);
  }
}
```

Simula DD <init-param>

Simula servlet container

Verifica se parâmetro foi mapeado à sessão

Verifica se parâmetro aparece na tabela HTML

Exemplo: funcionamento

Cliente (JUnit)

beginDoGet(WebRequest req)

- Grava parâmetro:

nome=**user**

value=**Jabberwock**

SUCCESS!!

JUnit
Test class name: junitdemo.TextUtilsTest
Reload classes every run
Runs: 1/1 Errors: 0 Failures: 0
junitdemo.TextUtilsTest
testRemoveWhiteSpaces
Failures | Test Hierarchy
Finished: 0.06 seconds

FAIL!

JUnit
Test class name: junitdemo.TextUtilsTest
Reload classes every run
Runs: 1/1 Errors: 0 Failures: 0
junitdemo.TextUtilsTest
testRemoveWhiteSpaces
Failures | Test Hierarchy
Finished: 0.06 seconds

falha local

falha remota

&

endDoGet(WebResponse res)

- Verifica se resposta contém

USER</td><td>JABBERWOCK

ReqInfo

2 conexões HTTP:

- *Uma p/ rodar os testes e obter saida do servlet*
- *Outra para esperar resultados de testes (info sobre exceções)*

TestInfo

Output

Servidor (Tomcat)

setUp()

- *Define init-params no objeto* **config**
- *Roda* **init(config)**

testDoGet()

- *Roda* **doGet()**
- *Verifica se parâmetro (no response) foi mapeado à sessão*

tearDown()

- *Onde encontrar*
  - `http://httpunit.sourceforge.net`
- *Framework para testes funcionais de interface (teste tipo "caixa-preta")*
  - *Verifica a resposta de uma aplicação Web ou página HTML*
  - *É teste funcional caixa-preta (não é "unit")*
  - *Oferece métodos para "navegar" na resposta*
    - *links, tabelas, imagens*
    - *objetos DOM (Node, Element, Attribute)*
- *Pode ser combinado com Cactus no endXXX()*
  - *Argumento* `com.meterware.httpunit.WebResponse`
- *Acompanha ServletUnit*
  - *stub que simula o servlet container*

HttpUnit com Cactus

Veja MapperServletTest2.java

- *Troque o* **WebResponse** *em cada endXXX() por* **com.meterware.httpunit.WebResponse**

```
public void endDoGet(com.meterware.httpunit.WebResponse resp)
                                  throws org.xml.sax.SAXException {
   WebTable[] tables = resp.getTables();
   assertNotNull(tables);
   assertEquals(tables.length, 1);  // só há uma tabela
   WebTable table = tables[0];
   int rows = table.getRowCount();
   boolean keyDefined = false;
   for (int i = 0; i < rows; i++) {
        String key   = table.getCellAsText(i, 0); // col 1
        String value = table.getCellAsText(i, 1); // col 2
        if (key.equals("USER")) {
             keyDefined = true;
             assertEquals("JABBERWOCK", value);
        }
   }
   if (!keyDefined) {
        fail("No key named USER was found!");
   }
}
```

Outros testes com Cactus

- *Testes em taglibs*
  - *Veja exemplos em cactusdemo/taglib/src*
- *Testes em filtros*
  - *Usa proxy FilterRedirector*
  - *Teste básico é verificar se método doFilter() foi chamado*
  - *Veja exemplos em cactusdemo/src/xptoolkit/AuthFilter*
- *Testes em páginas JSP*
  - *Ideal é JSP não ter código Java*
  - *Principais testes são sobre a interface: HttpUnit!*
- *Testes em EJB*
  - *Indireto, através dos redirectors*
  - *Redirectors permitem testar beans com interface local chamados por código no servidor*

Testes em aplicações Web: conclusões

- Aplicações Web são difíceis de testar porque dependem da comunicação com servlet containers
  - Stubs, proxies e APIs, que estendem ou cooperam com o JUnit, tornam o trabalho mais fácil
  - Neste bloco, conhecemos três soluções que facilitam testes de unidade, de integração e de caixa-preta em aplicações Web
- Stubs como ServletUnit permitem testar as unidades de código mesmo que um servidor não esteja presente
- Proxies como os "redirectors" do Cactus permitem testar a integração da aplicação com o container
- Uma API, como a fornecida pelo HttpUnit ajuda a testar o funcionamento da aplicação do ponto de vista do usuário

Resumo

- *Neste minicurso, você conheceu*
  - ***JUnit*** *- framework criado para facilitar a criação e execução de testes para medir a qualidade do seu software*
  - ***Ant*** *- ferramenta indispensável que ajuda a automatizar diversos processos comuns em ambientes de desenvolvimento em Java*
  - ***Cactus*** *- coleção de redirecionadores para facilitar testes de integração de aplicações Web*

Conclusões

- *As ferramentas apresentadas neste tutorial podem*
  - *melhorar a qualidade do seu software*
  - *aumentar o reuso de seus componentes*
  - *melhorar suas estimativas de prazos*
  - *melhorar a produtividade de sua equipe*
  - *melhorar a comunicação*
  - *reduzir custos*
  - *tornar o desenvolvimento mais ágil e eficiente*
  - *reduzir drasticamente o tempo gasto na depuração*
- *O único investimento necessário para obter os benefícios é aprender a usá-las*

Fontes

[1] Richard Hightower e Nicholas Lesiecki. *Java Tools for eXtreme Programming.* Wiley, 2002. Explora as ferramentas Ant, JUnit, Cactus, JUnitPerf, JMeter, HttpUnit usando estudo de caso com processo XP.

[2] Jeffries, Anderson, Hendrickson. *eXtreme Programming Installed*, Addison-Wesley, 2001. Contém exemplos de estratégias para testes.

[3] *Apache Ant User's Manual*. Ótima documentação repleta de exemplos.

[4] *Apache Cactus User's Manual*. Contém tutorial para instalação passo-a-passo.

[5] Steve Lougran. *Ant In Anger - Using Apache Ant in a Production Development System*. (Ant docs) Ótimo artigo com boas dicas para organizar um projeto mantido com Ant.

[6] Kent Beck, Erich Gamma. *JUnit Test Infected: programmers love writing tests*. (JUnit docs). Aprenda a usar JUnit em uma hora.

[7] Andy Schneider. *JUnit Best Practices*. JavaWorld, Dec. 2000. Dicas do que fazer ou não fazer para construir bons testes.

[8] Martin Fowler, Matthew Foemmel. *Continuous Integration*. http://www.martinfowler.com/articles/continuousIntegration.html. Ótimo artigo sobre integração contínua e o CruiseControl.

Fontes (2)

[9] *Robert Koss, Testing Things that are Hard to Test. Object Mentor, 2002*
http://www.objectmentor.com/resources/articles/TestingThingsThatAreHa~9740.ppt. Mostra estratégias para testar GUIs e código com dependências usando stubs.

[10] *Mackinnon, Freeman, Craig. Endo-testing with Mock Objects.*
http://mockobjects.sourceforge.net/misc/mockobjects.pdf. O autor apresenta técnicas para testes usando uma variação da técnica de stubs chamada de "mock objects".

[11] *William Wake. Test/Code Cycle in XP. Part 1: Model, Part II: GUI.*
http://users.vnet.net/wwake/xp/xp0001/. Ótimo tutorial em duas partes sobre a metodologia "test-first" mostrando estratégias para testar GUIs na segunda parte.

[12] *Steve Freeman, Developing JDBC Applications Test First. 2001. Tutorial sobre metodolodia test-first com mock objects para JDBC.*

[13] *Martin Fowler. Refactoring: improving the design of existing code. Addison-Wesley, 2000. Cap 4 (building tests) é um tutorial usando JUnit.*

[14] *Erik Hatcher. Java Development with Ant. Manning Publications. August 2002. Explora os recursos básicos e avançados do Ant, sua integração com JUnit e uso com ferramentas de integração contínua como AntHill e CruiseControl.*

[15] *Jesse Tilly e Erik Burke. Ant: The Definitive Guide. O'Reilly and Associates. May 2002. Contém referência completa e ótimo tutorial sobre recursos avançados como controle dos eventos do Ant e criação de novas tarefas.*

helder@argonavis.com.br

www.argonavis.com.br/palestras/abaporu

© 2002, Helder da Rocha