Modulo IIa
Extra: Logging

Professor
Ismael H F Santos – ismael@tecgraf.puc-rio.br

Bibliografia

- *Linguagem de Programação JAVA*
  - *Ismael H. F. Santos, Apostila UniverCidade, 2002*
- *The Java Tutorial: A practical guide for programmers*
  - *Tutorial on-line: http://java.sun.com/docs/books/tutorial*
- *Java in a Nutshell*
  - *David Flanagan, O´Reilly & Associates*
- *Just Java 2*
  - *Mark C. Chan, Steven W. Griffith e Anthony F. Iasi, Makron Books.*
- *Java 1.2*
  - *Laura Lemay & Rogers Cadenhead, Editora Campos*

Bibliografia

[1] Richard Hightower e Nicholas Lesiecki. *Java Tools for eXtreme Programming.* Wiley, 2002. Explora Ant e outras ferramentas em ambiente XP.

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

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

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

[5] 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.

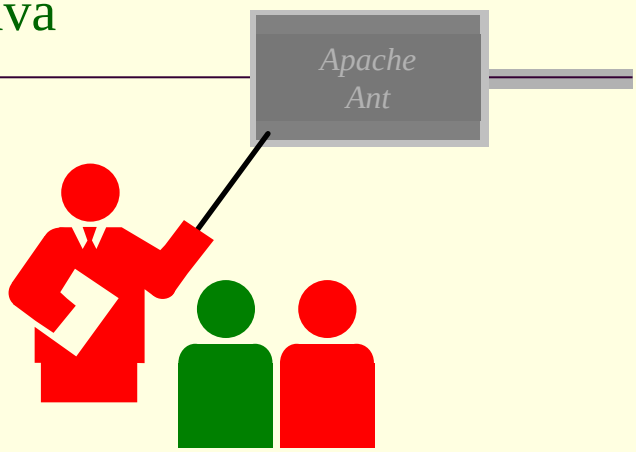
[6] Jesse Tilly e Erik Burke. *Ant: The Definitive Guide.* O'Reilly and Associates. May 2002. Contém referência completa e ótimo tutorial sobre recursos avançados como controle dos eventos do Ant e criação de novas tarefas.

[7] Karl Fogel. *Open Source Development with CVS.* Coriolis Press. http://cvsbook.red-bean.com/.

Ementa

- Princípios e Padrões de Projeto de SW
  - Ferramenta para construção de Aplicações - Ant
  - Logging
  - Classe Object
  - Documentação de código com JavaDOC

POO-Java



O que é o Ant?

- Uma 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
- Onde encontrar: http://ant.apache.org

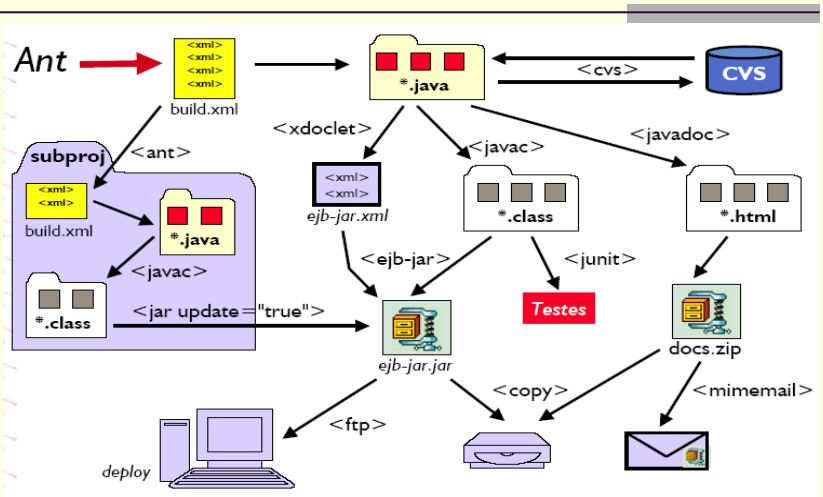
Para que serve ?

- 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



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*
    - *Pode-se estabelecer dependências entre alvos*
    - *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) - usadas dentro dos alvos.*
    `<javac>`, `<jar>`, `<java>`, `<copy>`, `<mkdir>`, ...

Buildfile

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!-- Compila diversos arquivos .java -->
<project default="compile" basedir=".">
  <property name="src.dir" value="src" />
  <property name="build.dir" value="classes" />
  <target name="init">
    <mkdir dir="${build.dir}" />
  </target>
  <target name="clean">
    <delete dir="${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

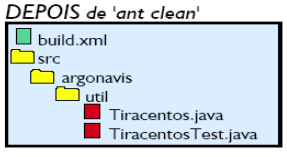
Elementos embutidos nas tarefas

Exemplo

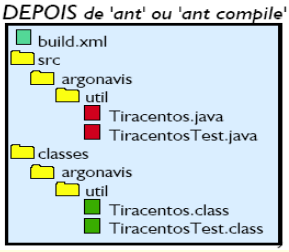
- *Executando buildfile da página anterior*

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

init:
    [mkdir] Created dir:
    C:\usr\palestra\antdemo\classes

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

BUILD SUCCESSFUL
Total time: 4 seconds
C:\usr\palestra\antdemo> ant clean
Buildfile: build.xml

clean:
    [delete] Deleting dir:
    C:\usr\palestra\antdemo\classes

BUILD SUCCESSFUL
Total time: 2 seconds
C:\usr\palestra\antdemo>
```

ANTES de 'ant'
DEPOIS de 'ant clean'



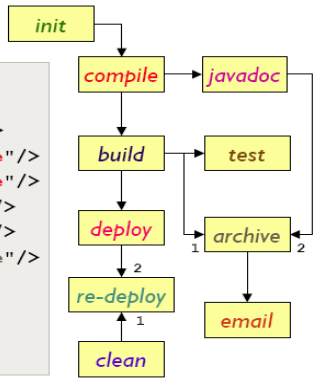
DEPOIS de 'ant' ou 'ant compile'



Dependências

- *Fazem com que a chamada de um alvo cause a chamada de outros alvos, em determinada ordem*
  - *Promovem reuso de código*

```
<target name="init"  />
<target name="clean" />
<target name="compile" depends="init"/>
<target name="javadoc" depends="compile"/>
<target name="build"   depends="compile"/>
<target name="test"    depends="build"/>
<target name="deploy"  depends="build"/>
<target name="email"   depends="archive"/>
<target name="archive"
        depends="build, javadoc"/>
<target name="re-deploy"
        depends="clean, deploy"/>
```



Tarefas Condicionadas

- *Algumas tarefas só são executadas dentro de determinadas condições*
  - *<mkdir> só cria o diretório se este não existir*
  - *<delete> só apaga o que existe (não faz nada se arquivo ou diretório não existir)*
  - *<javac> compila apenas os arquivos *.java que foram modificados desde a última compilação*
- *Comportamento condicional do <javac> depende da estrutura de pacotes*
  - *É preciso que a estrutura de diretórios dos fontes (diretório src/) reflita a estrutura de pacotes*
  - *Ex: se Conta.java declara pertencer a pacote banco, deve estar em diretório banco dentro de src/*

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>
- Executar roteiros e sons
  <script>, <sound>
- Criar novas tarefas
  <taskdef>

Tarefas úteis

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

Tarefas úteis

- **<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"/>
```


Tipos de Dados

- **<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>
```

Árvore a ser copiada para ${build.dir}/META-INF consiste de

- O arquivo ejb-jar.xml localizado em ${xml.dir}
- Todos os arquivos .xml de ${xml.dir}/jboss com exceção dos arquivos terminados em -orig.xml

Mais tarefas úteis

- **<javadoc>**: Gera documentação do código-fonte.
  - Exemplo: alvo generate-docs abaixo gera documentação excluindo classes que terminam em '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>
```

Copiar de ${src.dir}

Procurar em todos os subdiretórios

Onde achar as fontes

Propriedades

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

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

- Podem ser carregadas de um arquivo

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

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

arquivo.properties

- 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>
```


Mais tipos de dados

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

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

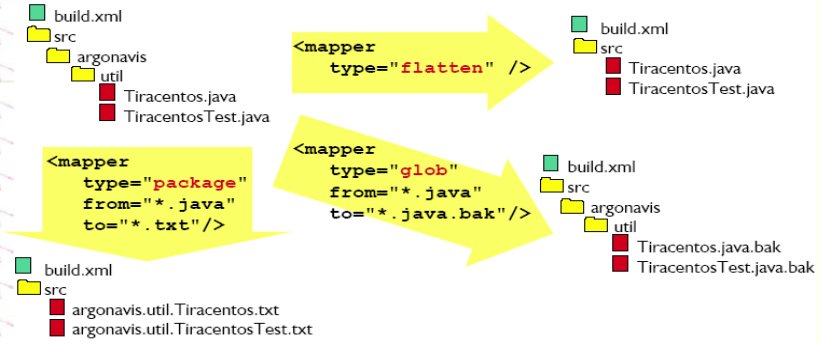
Padrões podem ser reusados e são identificados pelo ID

- **<path>**: 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>
```

Ainda mais tipos de dados ....

- **<mapper>**: *altera nomes de arquivos durante cópias ou transformações (use dentro de <copy>, por exemplo)*
  - *Seis tipos: identity, flatten, merge, regexp, glob, package*



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 cópia abaixo irá substituir todas as ocorrências de `@javahome@` por `c:\j2sdk1.4` nos arquivos copiados

```
<copy todir="${dest.dir}">
   <fileset dir="${src.dir}"/>
   <filterset>
      <filter token="javahome" value="c:\j2sdk1.4"/>
   </filterset>
</copy>
```

- Pares token=valor podem ser carregados de arquivo:

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

➡ Substituição sem tokens pode ser feita com <replace>

Tarefas úteis para Execução

- **<java>**: executa 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 úteis para J2EE

```
<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 de aplicação

```
<war destfile="bookstore.war" webxml="meta/metainf.xml">
   <fileset dir="${build}/${bookstore2}" >
      <include name="*.jsp" />
      <exclude name="*.txt" />
   </fileset>
   <classes dir="${build}" >
      <include name="database/*.class" />
   </classes>
   <lib dir="bibliotecas" />
   <webinf dir="etc" />
</war>
```

WEB-INF/web.xml

Fileset para raiz do WAR

Fileset para WEB-INF/classes

Fileset para WEB-INF/lib

Fileset para WEB-INF/

Tarefas úteis para extensão XDoclet

<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>
```

Detalhes da configuração do componente estão nos comentários de JavaDocs do código-fonte dos arquivos envolvidos

Tarefas úteis para 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>
```


Tarefas úteis para XSLT

- **<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" />
```

  - 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

argonavis.com.br

Tarefas úteis para JDBC

- **<sql>**: Comunica-se com banco de dados através de um 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>
```


Tarefas úteis para chamadas externas

- **<ant>**: chama alvo de subprojeto (buildfile externo)

```
<target name="run-sub">
   <ant dir="subproj" />
</target>
```

Chama alvo default de build.xml localizado no subdiretório subproj/

```
<target name="run-sub">
 <ant dir="subproj" >
   <property name="versao"
             value="1.0" />
 </ant>
</target>
```

Define propriedade que será lida no outro build.xml

- **<antcall>**: chama alvo local

```
<target name="fazer-isto">
   <antcall target="fazer">
      <param name="oque"
             value="isto" />
   </antcall>
</target>
```

```
<target name="fazer-aquilo">
   <antcall target="fazer">
      <param name="oque"
             value="aquilo" />
   </antcall>
</target>
```

```
<target name="fazer">
   <tarefa atributo="${oque}" />
</target>
```

Template!

Efeitos Sonoros

- **<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* `festa.wav` *será tocado quando o build terminar sem erros fatais.* `vaia.wav` *tocará se houver algum erro que interrompa o processo:*

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


usando extensão XML

- *Como o buildfile é um arquivo XML, pode-se incluir trechos de XML externos através do uso de entidades externas*

sound.xml

```
<property file="sound.properties" />
<sound>
    <success source="${success.sound}"/>
    <fail source="${fail.sound}"/>
</sound>
```

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE project [
    <!ENTITY sound SYSTEM "sound.xml">
]>
<project default="dtd">
    <description>Gera um DTD para o Ant</description>
    <target name="init">
        &sound;
    </target>
    <target name="dtd" depends="init">
        <antstructure output="ant.dtd" />
    </target>
</project>
```

Gerenciando projetos com Ant

- Crie um diretório para armazenar seu projeto. Guarde na sua raiz o seu build.xml
  - Use um arquivo build.properties para definir propriedades exclusivas do seu projeto (assim você consegue reutilizar o mesmo build.xml em outros projetos). Importe-o com
    `<property file="build.properties" />`
- Dentro desse diretório, crie alguns subdiretórios
  - `src/` Para armazenar o código-fonte
  - `lib/` Opcional. Para guardar os JARs de APIs usadas
  - `doc/` Opcional. Para guardar a documentação gerada
  - `etc/` Opcional. Para arquivos de configuração se houver
  - `web/` Em projetos Web, para raiz de documentos do site
- O seu Ant script deve ainda criar durante a execução
  - `build/` Ou `classes/`. Onde estará o código compilado
  - `dist/` Ou `jars/` ou `release/`. Onde estarão JARs criados

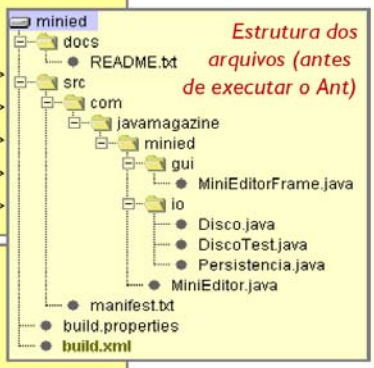
Targets Básicos do build.xml

- Você também deve padronizar os nomes dos alvos dos seus build.xml. Alguns alvos típicos são
  - init Para criar dietórios, inicializar o ambiente, etc.
  - clean Para fazer a faxina, remover diretórios gerados, etc.
  - compile Para compilar
  - build Para construir a aplicação, integrar, criar JARs
  - run Para executar um cliente da aplicação
  - test Para executar os testes da aplicação
  - deploy Para implantar componentes Web e EJB
- Você pode usar outros nomes, mas mantenha um padrão
- Também pode criar uma nomenclatura que destaque alvos principais, usando maiúsculas. Ex:
  - CLEAN, chamando clean-isto, clean-aquilo, undeploy, etc.
  - BUILD, que chama build-depend, build-client, build-server

Exemplo de Projeto

build.xml

```
<project default="compile" name="MiniEd">
  <property file="build.properties"/>
  <target name="init">
    <mkdir dir="${build.dir}"/>
    <mkdir dir="${dist.dir}"/>
  </target>
  <target name="clean">   ...  </target>
  <target name="compile"
          depends="init"> ...  </target>
  <target name="build"
          depends="compile">...</target>
  <target name="javadoc"
          depends="build"> ... </target>
  <target name="run"
          depends="build"> ... </target>
</project>
```



Estrutura dos arquivos (antes de executar o Ant)

```
# Nome da aplicação
app.name=minied
# Nomes dos diretórios
src.dir=src
docs.dir=docs
build.dir=classes
dist.dir=jars
# Nome da classe executável
app.main.class=com.javamagazine.minied.MiniEditor
root.package=com
```

build.properties

Construindo aplicações com Ant

- Exemplos de projetos usando o Ant
  - Aplicação gráfica com JAR executável
  - Aplicação RMI-IIOP
  - Aplicação Web com geração de WAR e deployment no servidor Tomcat
  - Aplicação EJB com geração de EJB-JAR e deployment no servidor JBoss
  - Aplicação com transformação XSL
- As listagens são resumidas e incompletas
  - Faça download do código completo

Estes exemplos não serão explorados na palestra devido ao tempo, mas foram mantidos como referência para consultas

Aplicação Gráfica Executável com Ant

```
<project default="compile" name="MiniEd">

    <property file="build.properties"/>

    <target name="compile" depends="init">
        <javac destdir="classes" srcdir="src">
            <classpath>
                <pathelement location="classes"/>
            </classpath>
        </javac>
    </target>

    <target name="build" depends="compile">
        <jar destfile="release/${app.name}.jar">
            <manifest>
                <attribute name="Main-class" value="${app.main.class}" />
            </manifest>
            <fileset dir="classes"/>
        </jar>
    </target>

    <target name="run" depends="build">
        <java jar="release/${app.name}.jar" fork="true" />
    </target>

</project>
```

Definindo o JAR com atributo Main-class para torná-lo executável

```
# Nome da aplicação - este nome será usado para criar o JAR
app.name=minied
# Nome da classe executável
app.main.class=com.javamagazine.minied.MiniEditor
```


Buildfile para aplicação Web

build.xml

```
<project default="deploy" name="Aplicação Web">
    <property file="build.properties" /> <!-- init e clean omitidos -->

    <target name="compile" depends="init">
        <javac srcdir="src" destdir="classes">
            <classpath path="${servlet.jar}" />
        </javac>
    </target>

    <target name="war" depends="compile">
        <war warfile="release/${context}.war" webxml="etc/web.xml">
            <fileset dir="web" />
            <classes dir="classes" />
        </war>
    </target>

    <target name="deploy" depends="war">
        <copy todir="${deploy.dir}">
            <fileset dir="release">
                <include name="*.war" />
            </fileset>
        </copy>
    </target>

</project>
```

build.properties

```
# Localizacao do Servidor
tomcat.home=/tomcat-4.0

# Altere para informar dir de instalacao
deploy.dir=${tomcat.home}/webapps

# Coloque aqui nome do contexto
context=forum

# JAR com Servlet API
servlet.jar=${tomcat.home}/common/lib/servlet.jar
```

argonavis.com.br

Buildfile para RMI-IIOP

```
<project name="Aplicação RMI" default="compile">
    <target name="compile" depends="init"> <!-- Vários <target> omitidos -->
        <javac destdir="classes" srcdir="src" >
            <classpath refid="app.path" />
        </javac>
    </target>
    <target name="buildrmi" depends="compile">
        <rmic idl="true" iiop="true" base="classes">
            <include name="**/rmiop/**Impl.class" />
            <include name="**/portable/**Impl.class" />
        </rmic>
    </target>
    <target name="runserver" depends="buildrmi">
        <java classname="hello.rmiop.HelloServer" fork="true">
            <jvmarg value="-Djava.rmi.server.codebase=${codebase}"/>
            <jvmarg value="-Djava.security.policy=${lib.dir}/rmi.policy"/>
            <jvmarg value="-Djava.naming.factory.initial=..."/>
            <jvmarg value="-Djava.naming.provider.url=iiop://${host}:1900"/>
            <classpath refid="app.path" />
        </java>
    </target>
    <target name="orbd">
        <exec executable="${java.home}\bin\orbd">
            <arg line="-ORBInitialPort 1900 -ORBInitialHost ${host}"/>
        </exec>
    </target>
</project>
```


Buildfile para aplicação EJB

build.xml

```
<project name="Aplicação EJB" default="deploy">

    <property file="build.properties" />

    <!-- elementos <path> e <target> init, compile, clean omitidos -->

    <target name="build" depends="compile">
        <copy todir="classes/META-INF">
            <fileset dir="etc" includes="ejb-jar.xml"/>
        </copy>
        <jar jarfile="release/${app.name}.jar">
            <fileset dir="classes" />
        </jar>
    </target>

    <target name="deploy" depends="build">
        <copy todir="${deploy.dir}" file="release/${app.name}.jar" />
    </target>

    <target name="undeploy" depends="build">
        <delete file="${deploy.dir}/${app.name}.jar" />
    </target>

</project>
```

build.properties

```
# Localizacao do Servidor
jboss.home=/jboss-3.0.0

# Altere para informar dir de instalacao
deploy.dir=${jboss.home}/server/default/deploy

# Coloque aqui nome da aplicação
app.name=forumejb
```

Buildfile para transformação XSL

```
<project name="foptask-example" default="pdf">

    <target name="setup" depends="check">
      <taskdef name="fop" classname="argonavis.pdf.FopTask">
        <classpath> ... </classpath>
      </taskdef>
    </target>

    <target name="many2fo" depends="init">
        <style in="template.xml" out="all.xml" style="many2one.xsl">
            <param name="docsdir" expression="dados"/>
        </style>
        <style in="all.xml" out="all.fo"
               extension=".fo" style="many2fo.xsl"/>
    </target>

    <target name="many2pdf" depends="many2fo">
        <fop in="all.fo" out="all.pdf" />
    </target>

    <target name="html" depends="init">
        <style basedir="dados" destdir="html"
               extension=".html" style="toHtml.xsl" />
    </target>

</project>
```

Mescla vários XML em um único XML maior e converte em XSL-FO

Converte XSL-FO em PDF

Converte vários XML em HTML

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()

Ant Programável – Exemplo

- *Cria 10 diretórios pasta1, pasta2, etc. em pastas/*

```
<project name="scriptdemo" default="makedirs" >
  <property name="result.dir" value="pastas" />
  <target name="setup-makedirs">
    <mkdir dir="${result.dir}" />
    <script language="javascript"><![CDATA[
      for (i = 0; i < 10; i++) {
        criadir = scriptdemo.createTask("mkdir");
        // Obter propriedade ${result.dir} deste projeto
        root = scriptdemo.getProperty("result.dir");
        // Definir diretorio a criar
        criadir.setDir(new
                  Packages.java.io.File(root+"/pasta"+(i+1)));
        // Executa tarefa mkdir (todo Task tem um metodo execute)
        criadir.execute();
      }
    ]]></script>
  </target>
  <target name="makedirs" depends="setup-makedirs" />
</project>
```

Obtém referência para objeto que implementa tarefa mkdir

Ant Programável – Exemplo

```
import org.apache.tools.ant.*;

public class ChangeCaseTask extends Task {

    public static final int UPPERCASE = 1;
    private String message;
    private int strCase = 0;

    public void execute() {
        log( getMessage() );
    }
    public void setMessage(String message) {
        this.message = message;
    }
    public void setCase(String strCase) {
        if (strCase.toLowerCase().equals("uppercase"))
           this.strCase = UPPERCASE;
        else if (strCase.toLowerCase().equals("lowercase"))
           this.strCase = -UPPERCASE;
    }
    public String getMessage() {
        switch(strCase) {
            case UPPERCASE:  return message.toUpperCase();
            case -UPPERCASE: return message.toLowerCase();
            default: return message;
        }
    }
}
```

```
<target name="define-task" depends="init">
  <taskdef name="changecase"
           classname="ChangeCaseTask">
    <classpath>
      <pathelement location="tasks.jar" />
    </classpath>
  </taskdef>
</target>

<target name="run-task" depends="define-task">
  <changecase message="Mensagem simples" />
  <changecase message="Mensagem em caixa-alta"
              case="uppercase" />
  <changecase message="Mensagem em caixa-baixa"
              case="lowercase" />
</target>
```

Trecho do build.xml usando tarefa <changecase>

Cada atributo é definido em um método setAtributo(String)

Método execute() chama log(), que imprime resultado na saída do Ant

(Exceções foram ignoradas por falta de espaço)

12

Integração com outras aplicações

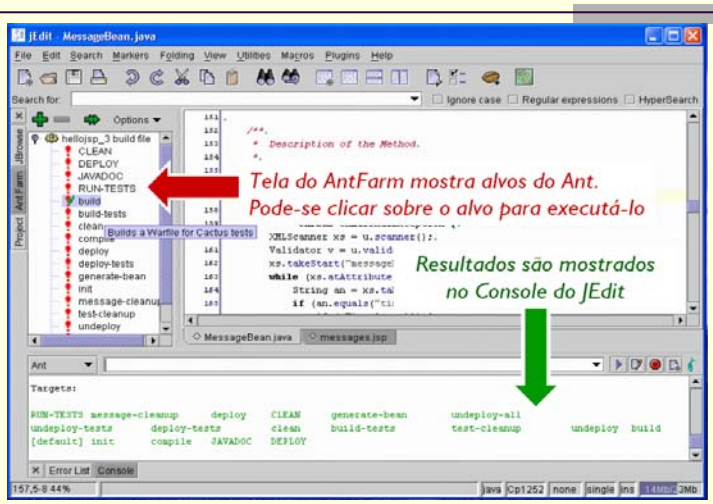
- Ant provoca vários eventos que podem ser capturados por outras aplicações
  - Útil para implementar integração, enviar notificações por email, gravar logs, etc.
- Eventos
  - Build iniciou/terminou
  - Alvo iniciou/terminou
  - Tarefa iniciou/terminou
  - Mensagens logadas
- Vários listeners e loggers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

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


Integração com outros 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/
  - Eclipse
    - http://eclipse.org
  - JEdit (AntFarm plug-in)
    - http://www.jedit.org
  - Jext (AntWork plug-in)
    - ftp://jext.sourceforge.net/pub/jext/plugins/AntWork.zip

Integração com JEdit



Conclusões

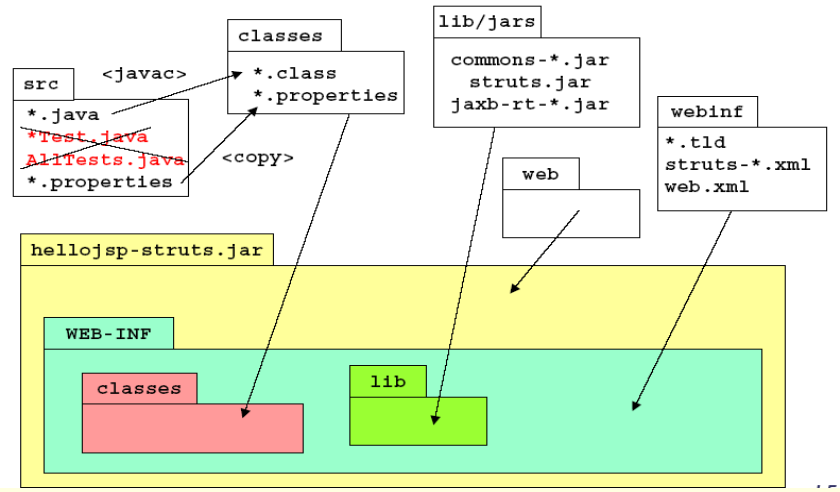
- Ant é uma ferramenta indispensável em qualquer projeto de desenvolvimento Java
  - Permite automatizar todo o desenvolvimento
  - Facilita a montagem da aplicação por outras pessoas
  - Ajuda em diversas tarefas essenciais do desenvolvimento como compilar, rodar, testar, gerar JavaDocs, etc.
  - Independe de um IDE comercial (mas pode ser facilmente integrado a um)
- Use o Ant em todos os seus projetos
  - Crie sempre um projeto e um buildfile, por mais simples que seja a sua aplicação
  - Escreva buildfiles que possam ser reutilizados
  - Desenvolva o hábito de sempre usar o Ant

Exercício

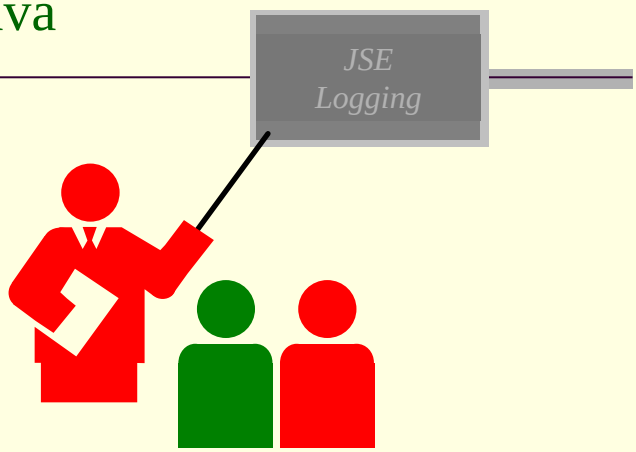
- *1. Monte um buildfile simples para a aplicação MiniEd.jar. Use o build.xml e build.properties fornecidos (com comentários). Implemente alvos para*
  - *Compilar a aplicação*
  - *Montar a aplicação como um JAR executável*
  - *Gerar JavaDocs da aplicação e colocá-los em um ZIP*
  - *Executar a aplicação*

  *Observações: (a) use <exclude> patternsets para não compilar as classes que terminarem em *Test.java.*
  (b) Inclua a pasta icons/ no CLASSPATH (raiz do JAR)
- *2. (opcional) Monte um build.xml para construir um WAR para a aplicação Web fornecida*
  - *Veja desenho na próxima página (observe os excludes)*

Exercício - Diagrama



POO-Java



Logging

- Recurso introduzido no J2SDK 1.4
- Oferece um serviço que gera relatórios durante a execução de uma aplicação. Os relatórios são formados por eventos escolhidos pelo programador e podem ter como destino:
  - A tela (console), um arquivo, uma conexão de rede, etc.
- Os dados também podem ser formatados de diversas formas
  - Texto, XML e filtros
- As mensagens são classificadas de acordo com a severidade, em sete níveis diferentes. O usuário da aplicação pode configurá-la para exibir mais ou menos mensagens de acordo com o nível desejado
- Principais classes
  - java.util.logging.Logger e java.util.logging.Level

Logging

- **Logging**: manutenção do registro do comportamento de uma aplicação.
- Por que realizar logging?
  - Mensagens de log podem ajudar na depuração da aplicação.
  - Mensagens de log fornecem o contexto da execução de uma aplicação

Logging

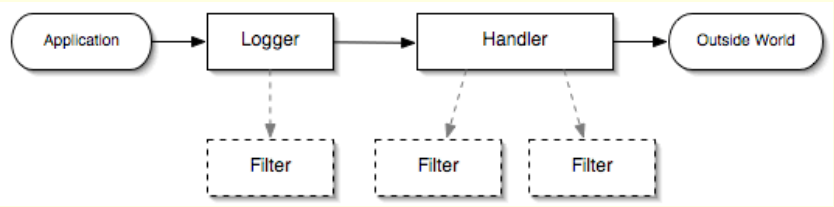
- Quando utilizar logging?
  - Na fase de desenvolvimento: para depurar a aplicação.
    - O que houver de errado? Onde ocorreu? Por que ocorreu?
  - Na fase de produção: ajuda a resolver problemas de execução.
- (Desvantagem) instruções de log têm o potencial de aumentar o tamanho do executável e de reduzir a velocidade da aplicação.

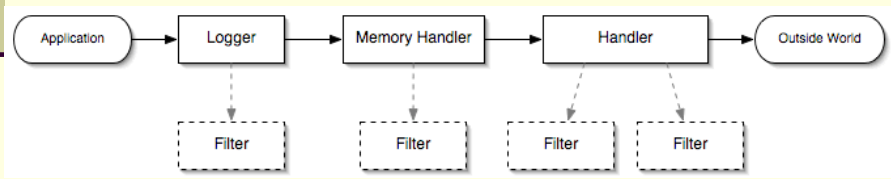
O componente Logger

- O ponto de entrada para o logging é um objeto Logger.
- Criamos um objeto dessa classe e requisitamos a ele que faça o log de mensagens.
- Há diversos **métodos de log** (i.e., métodos que aceitam como argumento uma mensagem de log) existentes em Logger.
- Além disso, todo objeto da classe Logger possui duas propriedades importantes:
  - Seu **nível**
  - Seu **nome**

- The JavaTM Logging APIs - package java.util.logging,
  - software servicing and maintenance at customer sites producing log reports suitable for analysis by end users, system administrators, field service engineers, and software development teams
- The Logging APIs capture information such as
  - security failures,
  - configuration errors,
  - performance bottlenecks,
  - and/or bugs in the application or platform

Logging – Control Flow





Logging architecture

- Applications make logging calls on Logger objects. Logger objects allocate LogRecord objects which are passed to Handler objects for publication
- Handlers may use logging Levels and (optionally) Filters to decide if they are interested in a particular LogRecord
- Handler can (optionally) use a Formatter to localize and format the message before publishing it to an I/O stream

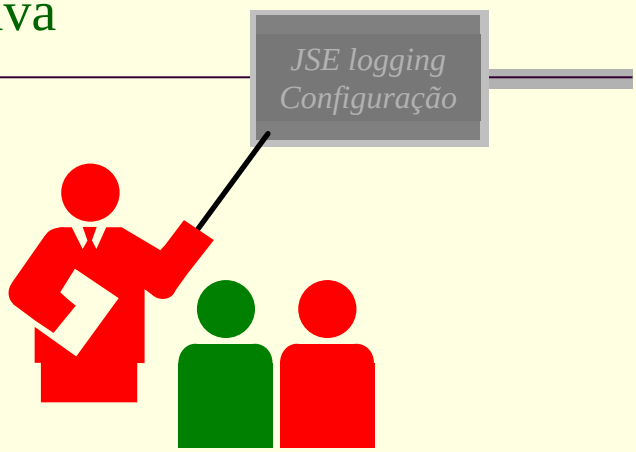
Logging

- Loggers are named entities, using dot-separated names, namespace is hierarchical and is managed by the LogManager
- The namespace should typically be aligned with the Java packaging namespace
- Loggers keep track of their parent loggers in the logging namespace,
- Loggers may inherit various attributes from their parents in the logger namespace
  - Logging level. If a Logger's level is set to be null then the Logger will use an effective Level that will be obtained by walking up the parent tree and using the first non-null Level.
  - Handlers. By default a Logger will log any output messages to its parent's handlers, and so on recursively up the tree. But Loggers may also be configured to ignore Handlers higher up the tree. (useParentHandlers=false)
  - Resource bundle names. If a logger has a null resource bundle name, then it will inherit any resource bundle name defined for its parent, and so on recursively up the tree.

Níveis de Severidade

- *As seguintes constantes da classe Level devem ser usadas para indicar o nível das mensagens gravadas*
  - *Level.OFF (não imprime nada no log)*
  - *Level.SEVERE (maior valor - imprime mensagens graves)*
  - *Level.WARNING*
  - *Level.INFO*
  - *Level.CONFIG*
  - *Level.FINE*
  - *Level.FINER*
  - *Level.FINEST (menor valor - imprime detalhamento)*
  - *Level.ALL (imprime tudo no log)*
- *Quanto maior o valor escolhido pelo usuário, menos mensagens são gravadas.*

POO-Java



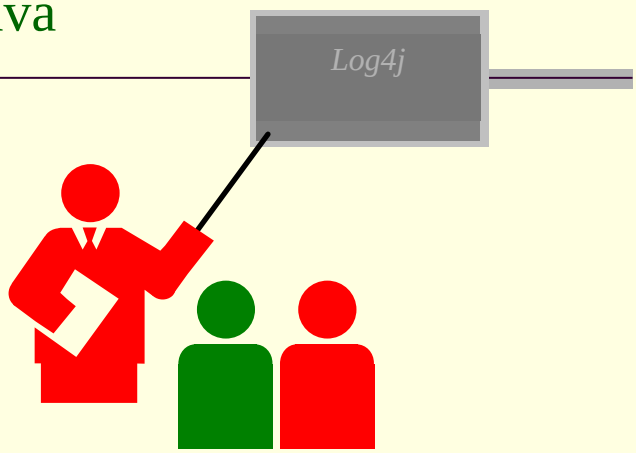
Configuracao

- **java.util.logging.manager** system property - for all JVMs instances
  - Default
    - -D **java.util.logging.config.class** ==null
      - -D **java.util.logging.config.file** !=null
        - The initial logging configuration will be read from this file
      - -D **java.util.logging.config.file** ==null
        - %JRE%lib/logging.properties
    - **-D java.util.logging.config.class** !=null
      - The given class will be loaded, an object will be instantiated, and that object's constructor is responsible for reading in the initial configuration.
  - Custom
    - The given class will be loaded, an object will be instantiated, and that object's constructor is responsible for reading in the initial configuration.

Configuracao Declarative

- **java.util.logging.manager** system property - for all JVMs instances
  - Default
    - -D **java.util.logging.config.class** ==null
      - -D **java.util.logging.config.file** !=null
        - The initial logging configuration will be read from this file
      - -D **java.util.logging.config.file** ==null
        - %JRE%lib/logging.properties
    - **-D java.util.logging.config.class** !=null
      - The given class will be loaded, an object will be instantiated, and that object's constructor is responsible for reading in the initial configuration.
  - Custom
    - The given class will be loaded, an object will be instantiated, and that object's constructor is responsible for reading in the initial configuration.

POO-Java



Log4j

- Log4j é um projeto de código aberto (*open source*) que permite ao desenvolvedor incluir logging em sua aplicação.
  - http://logging.apache.org
- Fornece serviços de logging para auditoria e depuração de aplicações.
- Com o Log4j, podemos ativar e desativar o logging sem a necessidade de modificar os binários da aplicação.
  - Isso pode ser controlado apenas editando um arquivo de configuração (detalhes mais adiante).

Log4j

- Permite o controle dinâmico de:
  - Destino da saída de log (email, console, SGBD, arquivo do SO, servidores de log, etc.)
  - Que informação deve ser registrada
  - Formatação da saída (texto, HTML, XML, etc)
- Foi implementado para outras linguagens além de Java:
  - C, C++, C#, Perl, Python, Ruby e Eiffel

Instalação do Log4j

- A versão atual do Log4J pode ser encontrada em
  - http://logging.apache.org/log4j/docs/download.html
  - disponíveis o código fonte e as bibliotecas (.jar)
- Faça a descompactação dos arquivos para alguma pasta.
- Adicione a biblioteca (.jar) do log4J no classpath da aplicação.
  - Nome do arquivo da biblioteca: log4j-xxx.jar (onde xxx é a versão).

Componentes do Log4J

- A API do Log4j tem os seguintes componentes principais:
  - Logger (classe)
  - Appender (interface)
  - Layout (classe)
- Há também componentes acessórios (auxiliares):
  - Level
  - PropertyConfigurator, DOMConfigurator
- Vamos agora detalhar esses componentes...

Logger

- *Para criar um Logger, é preciso usar seu construtor estático:*

```
Logger logger = Logger.getLogger("com.meupacote");
```

- *Os principais métodos de Logger encapsulam os diferentes níveis de detalhamento (severidade) ou tipos de informação. Métodos log() são genéricos e aceitam qualquer nível*
  - `config(String msg)`
  - `entering(String class, String method)`
  - `exiting(String class, String method)`
  - `fine(String msg)`
  - `finer(String msg)`
  - `finest(String msg)`
  - `info(String msg)`
  - `log(Level level, String msg)`
  - `severe(String msg)`
  - `throwing(String class, String method, Throwable e)`
  - `warning(String msg)`

Nível de um Logger

- O nível de um objeto logger determina que mensagens de log são realmente consideradas por este objeto.
  - Mensagens com nível igual ou mais alto que o nível definido para o logger são enviadas para a saída.
- Há 8 níveis, definidos como constantes (inteiras) na classe Level (também do log4j):
  - DEBUG < INFO < WARN < ERROR < FATAL
  - ALL (nível mais baixo) e OFF (nível mais alto)
- Um logger pode ser definido em um desses níveis
  - O método **setLevel** permite definir o nível do logger.

Método setLevel - exemplo

- O Log4j registra uma mensagem sse o método de log utilizado corresponde a um nível que é igual ou maior que o nível do logger.
- Exemplo:

```
Logger logger = Logger.getLogger(MinhaClasse.class.getName());
...
logger.setLevel(Level.ERROR);
...
```


Métodos de log

- Método de log = método chamado quando desejamos registrar uma mensagem no log na aplicação.
- São os métodos de log existentes na classes Logger são: debug, info, warn, error e fatal.
- Cada um desses métodos está associado a um nível, correspondente ao seu nome.
  - e.g., ***logger.info("Servidor Levantado")*** é uma requisição de log no nível INFO.

Métodos de log

- Habilitação versus desabilitação
  - Diz-se que uma mensagem de log está habilitada se o nível do método de log utilizado for maior ou igual ao nível do logger.
  - Do contrário, diz-se que a mensagem está desabilitada.

public void debug(Object message [, Throwable t]);
public void error(Object message [, Throwable t]);
public void fatal(Object message [, Throwable t]);
public void info(Object message [, Throwable t]);
public void warn(Object message [, Throwable t]);

Métodos de log - exemplo

```
Logger logger = Logger.getLogger(MinhaClasse.class.getName());
logger.setLevel(Level.ERROR);
...
logger.info("Início...");                    // Mensagem desabilitada
...
logger.error("Entrada inválida!");           // Mensagem habilitada
...
logger.fatal("Erro fatal.  Abortando!");     // Mensagem habilitada
```

```
...
catch (Exception ex) {
   logger.error("Exceção não esperada.", ex);
   return false;
}
…
```

Exemplo Logging

- *Exemplo da documentação da Sun [J2SDK14]*

```
package com.wombat;
public class Nose{
   // Obtain a suitable logger.
   private static Logger logger =
      Logger.getLogger("com.wombat.nose");

   public static void main(String argv[]){
     // Log a FINE tracing message
     logger.fine("doing stuff");
     try{
        Wombat.sneeze();
     } catch (Error ex){ // Log the error
       logger.log(Level.WARNING,"trouble sneezing",ex);
     }
     logger.fine("done");
   }
}
```


Uso dos métodos de log - Melhores Práticas

- Use **debug** para mensagens de depuração, que não devem ser gravadas quando a aplicação estiver em produção.
- Use **info** para mensagens similares ao modo "verbose".
- Use **warn** para avisos, que são gravados em algum lugar. Avisos não devem corresponder a situações em que a aplicação deve parar de executar.
- Use **error** para mensagens que são registradas por conta de algum erro recuperável da aplicação.
- Use **fatal** para mensagens críticas; após a gravação das mesmas, a aplicação deve ser abortada.

Nome de um Logger

- O padrão de nomenclatura para objetos da classe Logger é semelhante ao encontrado para classes em Java.
  - (i.e. com.pdxinc é “pai” de com.pdxinc.nhinar)
- Prática normalmente utilizada: aproveitar o nome de uma classe como o nome do seu logger.

```
public class MinhaClasse {
  private static Logger log= Logger.getLogger(MinhaClasse.class.getName());
            ...
}
```

Appenders

- No log4j, Appender é uma interface que representa o destinatário da saída do log.
- Um logger pode ter vários appenders associados a ele.
  - Ou seja, o Log4j permite que uma requisição de log seja enviada para mais de um objeto que implemente Appender.

Appenders

- O método **addAppender** da classe Logger adiciona um objeto Appender a um certo logger.
- Há diversas classes que implementam a interface Appender...

Classes que implementam Appender

- **ConsoleAppender**: envia requisições de log para a saída padrão (System.out ou System.err).
- **FileAppender**: envia requisições de log para um arquivo
- **RollingFileAppender**
  - subclasse de FileAppender,
  - pode fazer um backup do arquivo sempre que ele atingir um determinado tamanho.

Classes que implementam Appender (cont)

- **DailyRollingFileAppender**
  - subclasse de FileAppender,
  - pode fazer um backup de tempos em tempos (e.g, a cada semana).
- **SMTPAppender** - appender para enviar o log para um destinatário de e-mail
- **SocketAppender** - envia os eventos de log para um servidor de logs remoto através do protocolo TCP.

Classes que implementam Appender (cont)

- **NTEventLogAppender** - envia para o sistema de log de uma máquina com plataforma Windows.
- **SyslogAppender** - envia os logs para um daemon (monitor de logs) remoto
- **JMSAppender** - envia os logs como uma mensagem JMS
- **AsyncAppender** - possibilita que os logs sejam coletados em um buffer e depois enviados para um ou mais appender anexados a ele.

ConsoleAppender - exemplo

```
public class ExemploConsoleAppender {
   static Logger logger =
            Logger.getLogger(ExemploConsoleAppender.class.getName());
   public static void main(String args[]) {
            SimpleLayout layout = new SimpleLayout();
            ConsoleAppender appender = new ConsoleAppender(layout);
            logger.addAppender(appender);
            logger.setLevel(Level.DEBUG);
            logger.debug("Mensagem de depuração.");
   }
}
```


FileAppender - exemplo

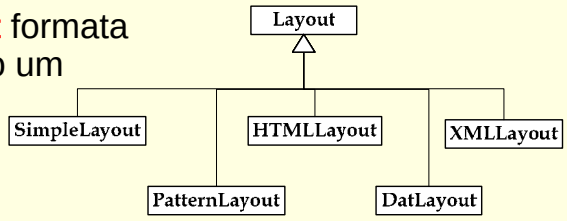
```
public class ExemploFileAppender {
  static Logger logger = Logger.getLogger(simpandfile.class);
  public static void main(String args[]) {
    SimpleLayout layout = new SimpleLayout();
    FileAppender appender = null;
    try {
        appender = new FileAppender(layout, "output1.txt", false);
    } catch(Exception e) {
        …
    }
    logger.addAppender(appender);
    logger.setLevel((Level) Level.DEBUG);
    logger.debug("Mensagem de DEBUG");  logger.info(" Mensagem de INFO");
    logger.warn(" Mensagem de WARN");  logger.error(" Mensagem de ERROR");
    logger.fatal(" Mensagem de FATAL");
  }
}
```

Layouts

- Além de registrar mensagens de log, o Log4j pode também registrar:
  - Instante (data e hora) em que o log foi requisitado,
  - Prioridade da mensagem (DEBUG, WARN, FATAL etc.),
  - Nome da classe, nome do método, número da linha no código fonte, onde o log foi requisitado,
  - etc
- O formato do que é registrado em log é especificado no **layout** ao qual o appender é associado.
- Um appender deve ter um Layout associado.
- Layout é a classe base do Log4j responsável pela formatação da saída para um certo objeto Appender.
  - Classes do Log4j que especificam a formação estendem Layout

Subclasses de Layout

- Há no log4j diversas subclasses de Layout predefinidas:
  - **SimpleLayout**: imprime o nível, o símbolo '-' e a mensagem de log.
  - **HTMLLayout** formata a saída como HTML.
  - **XMLLayout** formata a saída como XML.
  - **PatternLayout** formata a saída usando um *padrão de conversão*.
  - **DateLayout**



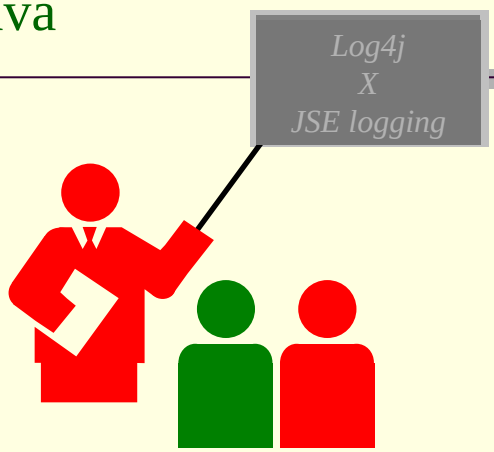
SimpleLayout - exemplo

```
public class ExemploConsoleAppender {
   static Logger logger =
            Logger.getLogger(ExemploConsoleAppender.class.getName());
   public static void main(String args[]) {
            SimpleLayout layout = new SimpleLayout();
            ConsoleAppender appender = new ConsoleAppender(layout);
            logger.addAppender(appender);
            logger.setLevel(Level.DEBUG);
            logger.debug("Mensagem de depuração.");
   }
}
```


PatternLayout - exemplo

```
public class ExemploPatternLayout {
   static Logger logger =
      Logger.getLogger(ExemploPatternLayout.class.getName());
   public static void main(String args[]) {
      String pattern =  "Duração do programa em milisegundos: %r %n";
      pattern += "Nome da classe: %C %n";
      pattern += "Data no formato ISO8601: %d{ISO8601} %n";
      pattern += "Local do evento de log: %l %n";
      pattern += "Mensagem: %m %n %n";
      PatternLayout layout = new PatternLayout(pattern);
      ConsoleAppender appender = new ConsoleAppender(layout);
      logger.addAppender(appender);
      logger.setLevel(Level.DEBUG);
      logger.debug("Mensagem de depuração.");
   }
}
```

POO-Java



Hello Log4j


```
import org.apache.log4j.*;
public class HelloLog4j
{
    private static Logger logger =
       Logger.getLogger(HelloLog4j.class);
 public static void main(String[] args)
 {
    BasicConfigurator.configure();
    logger.debug("In the main method");
    logger.info("What a beautiful day.");
    logger.error("This is an error message.");
 }
}
```

Output from HelloLog4j

```
0 [main] DEBUG HelloLog4j  - In the main method
0 [main] INFO HelloLog4j  - What a beautiful day.
10 [main] ERROR HelloLog4j  - This is an error message.
```


Hello java.util.logging

```
import java.util.logging.*;
public class HelloJDKLogging
{
   private static Logger logger =
     Logger.getLogger("com. foo.HelloJDKLogging");
   public static void main(String argv[])
   {
      logger.setLevel(Level.ALL);
     logger.fine(“Program started");
     logger.info(“This app uses java.util.logging”);
     logger.warning(“Oops, I did it again");
    }
}
```

Output from HelloJDKLogging

```
Jul 23, 2002 12:12:43 PM HelloJDKLogging main
INFO: This app uses java.util.logging
Jul 23, 2002 12:12:43 PM HelloJDKLogging main
WARNING: Oops, I did it again
```


Comparison: log4j & java.util.logging

org.apache.log4j	java.util.logging
LogManager class	LogManager class
Logger class	Logger class
Named loggers	Named loggers
Hierarchical namespaces	Hierarchical namespaces
Appender's	Handler's
Layout's	Formatter's
Level class	Level class
Filter class	Filter interface

Level

java.util.Logging.Level

Order: FINEST < FINER < FINE < CONFIG < INFO < WARNING < SEVERE
Other levels: Level.ALL
Level.OFF

org.apache.log4j.Level

Order: DEBUG < INFO < WARN < ERROR < FATAL
Other levels: Level.ALL
Level.OFF

Log4j Appender's

- AsyncAppender
- ConsoleAppender
- DailyRollingFileAppender
- JMSAppender
- NTEventLogAppender
- NullAppender
- RollingFileAppender
- SMTPAppender
- SocketAppender
- SyslogAppender

java.util.logging Handlers & Formaters

- Handlers
  - StreamHandler
  - ConsoleHandler
  - FileHandler
  - SocketHandler
  - MemoryHandler
- Formaters
  - SimpleFormatter
  - XMLFormatter

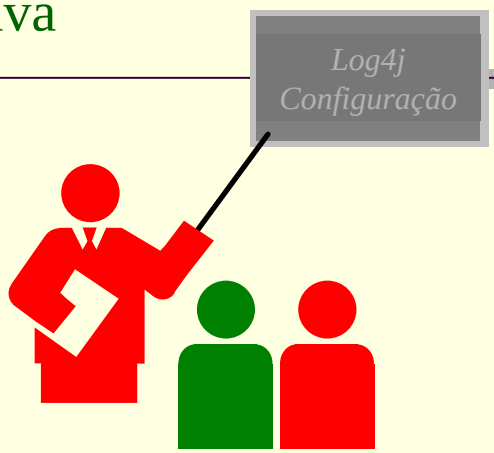
Logging: Best and Worst practices

- Best Practices
  - use the appropriate message level
  - roll your log files daily / weekly
  - review your error log on a daily basis
- Worst Practices
  - System.out.println / System.err.println
  - logging passwords to a log file
  - logging informational messages to STDERR
  - logging a message for every single HTTP request
  - multiple applications sending log messages to a single log file
  - ignoring error messages that appear in your application error log
  - misleading log message

JSE Logging x Log4j

- The primary weakness of JDK Logging over Log4J is that the JDKs configuration file isn't as robust, but it does provide a mechanism to punt down in to classes to do arbitrary work rather than having a more rich configuration file format
- Property "config". This property is intended to allow arbitrary configuration code to be run. The property defines a whitespace separated list of class names. A new instance will be created for each named class. The default constructor of each class may execute arbitrary code to update the logging configuration, such as setting logger levels, adding handlers, adding filters, etc

POO-Java



Configuração do log4j por arquivo

- As configurações do log4j **não** devem ser feitas no código fonte.
  - Isso porque é desejável mudar as opções de logging ser ter que recompilar a aplicação.
- Modo adequado: através de um arquivo de configuração.
- Há dois modos de especificar o arquivo de configuração:
  - Arquivo de propriedades (PropertyConfigurator)
  - Arquivo XML (DOMConfigurator)

PropertyConfigurator - exemplo

- Considere o seguinte arquivo de propriedades:

```
#define o nível do logger raiz e o nome de seu appender (htmlAppender)
log4j.rootLogger=DEBUG, htmlAppender

# define a classe do appender
log4j.appender.htmlAppender=org.apache.log4j.RollingFileAppender

#nome do arquivo de log
log4j.appender.htmlAppender.File=logging/HTMLLayout.html

# define a classe para formação (layout)
log4j.appender.htmlAppender.layout=org.apache.log4j.HTMLLayout

# define que deve ser registrada informação de contexto
log4j.appender.htmlAppender.layout.LocationInfo=true

# título do arquivo de log
log4j.appender.htmlAppender.layout.Title=Log gerado pelo Log4j
```

PropertyConfigurator – exemplo (cont.)

- Considere que o arquivo de propriedades anterior foi passado como argumento para o programa a seguir:

```
public class ExemploPropertyConfigurator {
        static Logger logger =
           Logger.getLogger(ExemploPropertyConfigurator.class.getName());
        public static void main(String[] args) {
           PropertyConfigurator.configure(args[0]);
           logger.info("Aplicação iniciada.");
           doSomething();
           logger.info("Aplicação terminada.");
        }
}
```


PropertyConfigurator – exemplo (cont.)

- O arquivo de log criado a partir do programa anterior:
  - Terá o nome HTMLLayout.html,
  - Terá como conteúdo uma tabela HTML com as mensagens de log em cada linha da tabela.
  - Para cada mensagem, terá:
    - o tempo (em milisegundos) decorrido desde quando o programa foi iniciado,
    - a thread que disparou o log,
    - o nível de prioridade,
    - (se LocationInfo = true) a classe, o nome do arquivo *.java desta classe e o número da linha,
    - a mensagem de log propriamente dita.

Customizações com PatternLayout

Pattern: "%r [%t] %-5p %c{2} - %m%n"
176 [main] INFO examples.Sort - Populating an array of 2 elements in reverse order

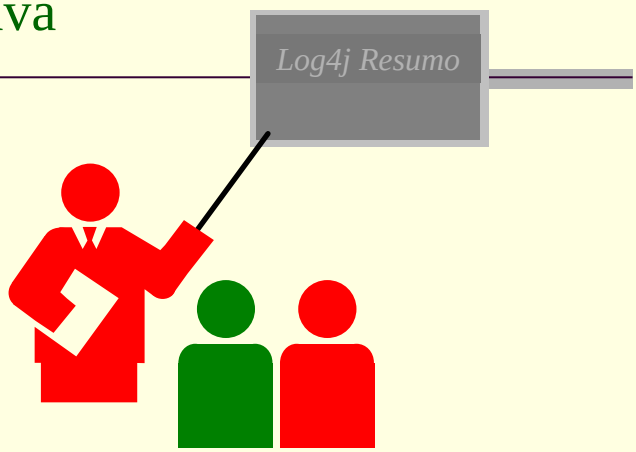
- r - número de milissegundos transcorridos desde o início do programa
- t - nome da thread que gerou o evento de log
- p - prioridade (o -5 indica que deve alinhar à direita se o número de caracteres for menor que cinco)

Customizações com PatternLayout

Pattern: "%r [%t] %-5p %c{2} - %m%n"
176 [main] INFO examples.Sort - Populating an array of 2 elements in reverse order

- c - nome da classe (2 indica que se o nome completo da classe for "a.b.c" por exemplo, deverá ser mostrado apenas "b.c")
- m - é a mensagem (não pode faltar !)
- n - é o separador de linhas do SO ("\n" ou "\r\n“)

POO-Java



Log4j - Resumo

- Principais componentes:
  - Logger: permite realizar requisições de log.
  - Appender: corresponde ao destinatário de uma mensagem de log.
  - Layout: especifica o formato de uma mensagem de log
- Propriedades de um objeto Logger
  - nome
  - nível
- Hierarquia de loggers oferece um maior controle;
- Diversas saídas: Console, Arquivo, Banco de Dados, XML, HTML, e-mail, ...
- API de código aberto;

Log4j - Resumo

- Elimina a necessidade de uso de “System.out” para depuração do código fonte;
- Permite controlar de maneira flexível as saídas de log;
- Configuração do log em tempo de execução sem alteração na codificação e sim em um arquivo de configuração;
- Com o uso de logging, há a possibilidade de diminuição do desempenho da aplicação.

Exercício

- Experimente utilizar os arquivos de configuração a seguir:

```
log4j.rootLogger=DEBUG, htmlAppender
# htmlAppender is set to be a ConsoleAppender which outputs to System.out.
log4j.appender.htmlAppender=org.apache.log4j.ConsoleAppender
# htmlAppender uses PatternLayout.
log4j.appender.htmlAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.htmlAppender.layout.ConversionPattern=%-5p [%t]: %m%n
#log4j.logger.package.MyApp =WARN
```

```
log4j.rootLogger=DEBUG, htmlAppender
log4j.appender.htmlAppender=org.apache.log4j.ConsoleAppender
log4j.appender.htmlAppender.layout=org.apache.log4j.PattenLayout
log4j.appender.htmlAppender.layout. ConversionPattern=%-4r [%t] %-5p %c - %m%n
```

Parágrafo **@since**

- Especifica a versão onde o identificador foi introduzido
- Recebe o identificador da versão
- Exemplo:

```
@since 1.0β
```