



Agent and Object Technology Lab
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Parma



Ingegneria del software A



Prof. Agostino Poggi

- ◆ È un interprete di comandi che viene utilizzato principalmente per automatizzare le operazioni in ambiente Java
- ◆ Ad esempio la compilazione dei sorgenti Java, la creazione di file jar e l'esecuzione di un programma Java
- ◆ È scritto in java e legge un file di configurazione scritto in XML (build.xml) che specifica cosa Ant deve fare

- ◆ Configurazione del file di *build* in xml
 - Facile da leggere e da modificare
- ◆ Scritto *in* Java e *per* Java
 - Supporta gli strumenti Java (`javac`, `javadoc`, ecc.)
 - Estendibile con l'uso di classi Java
- ◆ Ant è più veloce di uno script di comandi
 - Script basati sulla shell: ogni comando è un nuovo processo
 - Ant viene eseguito nella JVM
 - Ogni comando viene eseguito nella stessa JVM
 - Gli strumenti (come `javac`) sono thread, non processi
- ◆ Cross-platform
 - Supporta lo sviluppo cross-platform Java
 - Ma consente anche di eseguire comandi di shell

- ◆ Ogni progetto ha un *build file*
 - Il *build file* di default si chiama `build.xml`
 - Si può specificare un file diverso con l'opzione da linea di comando `-buildfile`
- ◆ Ogni *build file* è composto da vari *target*
 - “compile”, “test”, “run”, “clean”, etc.
- ◆ Ogni *target* è composto da vari *task*
 - Invoca un comando o un altro programma

- ◆ I *target* possono avere delle *dipendenze*
 - Ad esempio il target “run” può dipendere dal target “compile”
 - Dipendenze separate da virgole
- ◆ Si possono gestire le dipendenze a cascata
- ◆ La dipendenza viene eseguita solo se necessario

```
<project name="example" default="compile" basedir=". ">  
<!-- init properties -->
```

```
<target name="compile" description="...">  
  <!-- tasks -->  
</target>
```

```
<target name="dist" description="..." depends="compile">  
  <!-- tasks -->  
</target>
```

```
<!-- more targets ... -->  
</project>
```

- ◆ Property
- ◆ Compile
- ◆ Execution
- ◆ Archive
- ◆ Documentation
- ◆ File
- ◆ ...

- ♦ Il task *property* consente di impostare una proprietà (coppia nome/valore) o di includere delle proprietà da un file esterno
- ♦ Preferibilmente all'inizio del file di build
- ♦ Le proprietà sono *case sensitive*
- ♦ Le proprietà sono *immutabili*
 - Chiunque setti per primo il valore della proprietà la “congela” per il resto dell'esecuzione del *build*

- ◆ Impostare la proprietà “hello” al valore “Ciao!”

```
<property name="hello" value="Ciao!" />
```

- ◆ Caricare varie proprietà da un file chiamato “xyz.properties” che si trova nella stessa cartella

```
<property file="xyz.properties" />
```

- ◆ Caricare varie proprietà da un indirizzo URL

```
<property url="http://www.xyz.com/xyz.properties" />
```

```
build.compiler=jikes  
deploy.server=lucky  
deploy.port=8080  
deploy.url=http://${deploy.server}:${deploy.port}/
```

<code>java.version</code>	Java version
<code>java.vendor</code>	Java vendor
<code>java.vendor.url</code>	Java vendor URL
<code>java.home</code>	Java installation directory
<code>java.class.version</code>	Java class format version number
<code>java.class.path</code>	Java class path
<code>java.ext.dirs</code>	Path of extension directories
<code>os.name</code>	Operating system name
<code>os.arch</code>	Operating system architecture
<code>os.version</code>	Operating system version
<code>file.separator</code>	File separator ("/" on UNIX)
<code>path.separator</code>	Path separator (":" on UNIX)
<code>line.separator</code>	Line separator ("\n" on UNIX)
<code>user.name</code>	User's account name
<code>user.home</code>	User's home directory
<code>user.dir</code>	User's current working directory

- ◆ Un *fileset* rappresenta un gruppo di file
- ◆ Per individuarli, si possono usare dei caratteri jolly:
 - *?* ogni carattere
 - *** zero o più caratteri
 - **** zero o più directory
- ◆ Ad esempio, per selezionare i file in una cartella che
 - Abbiano l'estensione “.java” (*include*)
 - Non contengano nel nome “Test” (*exclude*)

```
<fileset dir="${src}">  
  <include name="**/*.java"/>  
  <exclude name="**/*Test*" />  
</fileset>
```

- ◆ Selezione attraverso diversi *Selector*, oppure *Patternset*

- Possibile assegnare un *id*, in modo tale da poterlo riutilizzare

```
<fileset dir="${server.src}" casesensitive="yes">  
  <patternset id="non.test.sources">  
    <include name="**/*.java"/>  
    <exclude name="**/*Test*" />  
  </patternset>  
</fileset>
```

- ◆ Stesso *patternset* applicato su un'altra cartella

```
<fileset dir="${client.src}" casesensitive="yes">  
  <patternset refid="non.test.sources" />  
</fileset>
```

- ◆ Due classpath: base, e con classi per i test

```
<path id="base.path">  
  <pathelement path="{classpath}" />  
  <fileset dir="lib">  
    <include name="**/*.jar" />  
  </fileset>  
  <pathelement location="classes" />  
</path>
```

```
<path id="test.path">  
  <path refid="base.path" />  
  <pathelement location="testclasses" />  
</path>
```

- ♦ Il seguente task *javac* compila tutte le classi della directory `${src}`
- ♦ I file compilati vengono messi in `${build}`
- ♦ Il classpath usato include la libreria `log4j.jar`
- ♦ Le informazioni di debug sono attivate

```
<javac srcdir="${src}"  
      destdir="${build}"  
      classpath="3rdparty\log4j.jar"  
      debug="on" />
```

- ◆ Simile al precedente esempio, ma include solo le classi nei package con nome `p*1` e `p*3`, tranne i file denominati `Test.java` e usa un classpath definito in precedenza

```
<javac srcdir="${src}"  
      destdir="${build}"  
      includes="**/p*1/*.java,**/p*3/*.java"  
      excludes="**/Test.java"  
      classpathref="${base.path}"  
      debug="on" />
```


- ♦ Il task *java* esegue una applicazione
- ♦ Si può forzare l'uso di un'altra JVM con `fork="true"`
- ♦ È possibile specificare un classpath

```
<java classname="test.Main" fork="true">  
  <arg value="-h" />  
  <classpath>  
    <pathelement location="dist/test.jar" />  
    <pathelement path="{java.class.path}" />  
  </classpath>  
</java>
```

- ♦ È possibile eseguire un file jar
- ♦ Ma il *manifest* del jar deve specificare la classe di avvio

```
<java jar="test.jar">  
  <arg value="something" />  
  <arg value="to" />  
  <arg value="test" />  
</java>
```

- ♦ Il seguente task *jar* crea il file `app.jar` in `${dist}`, contenente i file di `${build}`

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

- ♦ Stavolta vengono esclusi i file con nome `Test.class`

```
<jar destfile="${dist}/appNoTest.jar"  
    basedir="${build}"  
    excludes="**/Test.class" />
```

- ◆ Si possono impostare le proprietà del *manifest*, ad esempio per indicare la classe di avvio

Nota: qui il target jar dipende da compile

```
<target name="jar" depends="compile">
  <mkdir dir="${lib}" />
  <jar destfile="${lib}/${project.name}.jar"
      basedir="${build}">
    <manifest>
      <attribute name="Main-Class"
                  value="HelloWorldApp" />
    </manifest>
  </jar>
</target>
```

- ♦ Il task *javadoc* genera documentazione da sorgenti java

```
<javadoc destdir="docs/api"
        author="true" version="true"
        use="true" access="protected">

    <fileset dir="src">
        <include name="com/dummy/test/**/*.*.java"/>
        <exclude name="com/dummy/test/tmp/**" />
    </fileset>

    <bottom><![CDATA[<i>Copyright   2000 Dummy
        Corp. All Rights Reserved.</i>]]></bottom>
</javadoc>
```

- ♦ Il task *copy* copia file e cartelle
- ♦ Ad esempio, copiare una cartella in un'altra

```
<copy todir="${dst}">  
  <fileset dir="${src}" />  
</copy>
```

- ♦ Ad esempio, copiare tutti i file non html

```
<copy todir="${dst}">  
  <fileset dir="${src}">  
    <exclude name="**/*.html" />  
  </fileset>  
</copy>
```

- ♦ Ad esempio, copiare un set di file in una directory, appendendo al volo l'estensione .bak

```
<copy todir="${backup}">
  <fileset dir="${src}">
    <include name="**/*.java" />
    <include name="**/*.jar" />
  </fileset>
  <globmapper from="*" to="*.bak" />
</copy>
```

- ♦ Il task *delete* cancella file e/o cartelle
- ♦ Es. cancellare il file `log4j.jar` e poi la cartella `libs` (con file e sottocartelle)

```
<delete file="${dist}/libs/log4j1.jar" />  
<delete dir="${dist}/libs" />
```

- ♦ Es. cancellare tutti i file `html` da una cartella (e dalle sottocartelle)

```
<delete>  
  <fileset dir="${dist}" includes="**/*.html" />  
</delete>
```


- ♦ Il task **exec** esegue un comando di shell
- ♦ Se viene specificato l'attributo `os`, il task viene eseguito solo se ant sta girando su uno dei sistemi specificati nel parametro
- ♦ Se il parametro `spawn` è abilitato, allora l'applicazione lanciata resterà aperta al termine del build

- ♦ Avvia il `${browser}` con il file specificato `${file}` per un computer con WinXP e lo mantiene aperto anche dopo l'esecuzione del build

```
<property name="browser"
          location="C:/.../iexplore.exe" />
<property name="file"
          location="ant/docs/manual/index.html" />

<exec executable="${browser}"
      os="Windows XP"
      spawn="true">
  <arg value="${file}" />
</exec>
```

- ♦ Il task *dependset* confronta due insiemi di file e rimuove i file oggetto se almeno uno dei file sorgente è più recente
- ♦ Ad esempio, rimuovere i file html in `${output.dir}` se almeno uno dei seguenti file è più recente: `paper.dtd`, `common.dtd`, `common.xsl`, `build.xml`

```
<dependset>
  <srcfilelist dir="${dtd.dir}"
    files="paper.dtd,common.dtd" />
  <srcfilelist dir="${xsl.dir}" files="common.xsl" />
  <srcfilelist dir="${basedir}" files="build.xml" />
  <targetfileset dir="${output.dir}"
    includes="**/*.html" />
</dependset>
```

- ♦ Il task **echo** stampa un messaggio sul logger corrente
 - Può essere specificato un livello di importanza per il messaggio
 - Il doppio simbolo \$ serve da escape per il carattere \$
- ♦ Qualche esempio:

```
<echo message="Hello, world!" />  
<echo message="OS: ${os.name}" level="debug" />  
<echo file="runner.csh" append="false">  
  #\!/bin/tcsh java-1.3.1 -mx1024m  
  ${project.entrypoint} $$*  
</echo>
```

- ♦ Il task *input* abilita l'interazione con l'utente durante building (input da console)
- ♦ Es. attesa e richiesta di conferma

```
<input>Press Return key to continue...</input>
```

```
<input addproperty="do.delete"  
      message="Delete the DB (y/n)?"  
      validargs="y,n"  
      defaultvalue="n" />
```