

Introduzione all'utilizzo di Java Ant

Marco Lamieri

13-03-2004

Indice

1	Java Ant	3
1.1	Vantaggi	3
1.1.1	Indipendenza dalla piattaforma	3
1.1.2	Estendibilità	3
1.1.3	Integrazione	3
1.2	Svantaggi	3
1.2.1	Necessità di installare un software aggiuntivo	3
1.2.2	Necessità di dichiarare nuove variabili d'ambiente	4
1.3	Installazione	4
1.3.1	Windows 2000 e XP	4
1.3.2	Windows 98 e ME	5
1.3.3	Linux	5
1.3.4	Cygwin	5
1.4	Utilizzo di Ant	5
1.5	Preparazione del file build.xml	6
1.5.1	Attributi di project	6
1.5.2	Attributi di target	6
1.5.3	Attributi di task	7
1.5.4	Le properties	7
1.5.5	Un semplice build file	8
1.5.6	Gestione dei classpath	9
1.5.7	Generare la documentazione con javadoc	9
1.5.8	Utilizzo di scripts	10
1.5.9	Verificare condizioni	10
1.5.10	Interazione con altre applicazioni	11
1.5.11	Gestire l'output di ant	11
1.5.12	Gestire parametri da riga di comando	12

1.6	Come estendere e personalizzare Ant	13
1.6.1	Implementare la classe BuildListener	13
1.6.2	Aggiungere nuovi oggetti task	14
1.7	Riferimenti	15
2	Appendici	16
2.1	Build file per jesframe-0.9.9.0	16
2.2	Build file per art-0.1.0	19

1 Java Ant

Java Ant¹ è un progetto open source sviluppato dall'Apache Group allo scopo di gestire la compilazione, l'esecuzione e più in generale il processo di build delle applicazioni Java. L'acronimo ANT significa Another Neat Tool ed è stata rilasciata la prima versione il 19 Luglio 2000.

1.1 Vantaggi

I principali vantaggi di Ant rispetto al tradizionale comando *make* sono:

1.1.1 Indipendenza dalla piattaforma

Ant, essendo sviluppato integralmente in Java ed utilizzando files di appoggio in formato xml, può essere utilizzato senza alcuna modifica su molte piattaforme tra cui Windows, Linux e Cigwin.

1.1.2 Estendibilità

Con Ant si possono gestire, oltre alle operazioni classiche del processo di compilazione ed esecuzione Java, operazioni definite in modo semplice dall'utente (come il comando *javaswarm* o *javacswarm*).

1.1.3 Integrazione

Le librerie di Ant sono integrate in molti editor Java IDE come Java Development Environment for Emacs (JDEE) ed Eclipse.

1.2 Svantaggi

I principali svantaggi di Ant sono:

1.2.1 Necessità di installare un software aggiuntivo

Per eseguire la compilazione con Ant è necessario installare il pacchetto Ant (30 MB) comprendente il programma vero e proprio ed un interprete xml per gestire il file *build.xml* (equivalente del vecchio *Makefile*).

¹La homepage del progetto Ant è <http://ant.apache.org/>

1.2.2 Necessità di dichiarare nuove variabili d'ambiente

Per il corretto funzionamento di Ant è necessario impostare manualmente le variabili che dichiarano il percorso in cui sono installati Ant e Java, e che dovranno essere modificate ogni qual volta si aggiorna la versione del JDK o di Ant.

1.3 Installazione

Per prima cosa è necessario scaricare una versione aggiornata di Ant dall'indirizzo: <http://www.apache.org/dist/ant/binaries/>.

Il processo di installazione consiste semplicemente nel decomprimere il file scaricato in una directory a scelta del proprio computer, ad esempio in `C:\Programmi\ant`.

A questo punto è necessario impostare alcune variabili d'ambiente. La procedura varia a seconda del sistema operativo utilizzato.

Ipotizzando che Ant sia installato nella cartella

`C:\Programmi\ant`

e JDK nella cartella

`C:\j2sdk1.4.2`

1.3.1 Windows 2000 e XP

Selezionare dal menu *Start* ⇒ *Pannello di controllo* ⇒ *Sistema*

Dalla schermata *Avanzate* selezionare il tasto *Variabili d'ambiente* e poi *Nuovo* e indicare:

Nome variabile = `JAVA_HOME`

Valore variabile = `C:\j2sdk1.4.2`

Selezionare nuovamente *Nuovo* e indicare

Nome variabile = `ANT_HOME`

Valore variabile = `C:\Programmi\ant`

Selezionare la variabile *PATH* ed il tasto *Modifica*. Aggiungere al testo esistente queste informazioni.

Nome variabile = `PATH`

Valore variabile: `... ;%ANT_HOME%\bin`

1.3.2 Windows 98 e ME

Aggiungere al file *autoexec.bat* seguenti comandi:

```
set ANT_HOME=c:\ant
set JAVA_HOME=c:\jdk1.4.2
set PATH=%PATH%;%ANT_HOME%\bin
```

1.3.3 Linux

Eeguire i seguenti comandi:

```
export ANT_HOME=/usr/local/ant
export JAVA_HOME=/usr/local/jdk-1.2.2
export PATH=${PATH}:${ANT_HOME}/bin
```

1.3.4 Cygwin

Per poter eseguire correntamente ant dalla shell Cygwin è sufficiente seguire la procedura di installazione indicata per i sistemi Windows. Non si consiglia di indicare le variabili necessarie (JAVA_HOME e ANT_HOME) all'interno di *.bashrc* poiché, utilizzando direttamente quelle del sistema operativo Windows in il programma funzionerà correttamente sia se eseguito da Cigwin sia da Dos/Windows.

1.4 Utilizzo di Ant

Una volta costruito il file *build.xml* nella cartella principale è sufficiente eseguire dalla bash Cygwin il comando *ant* utilizzando questa sintassi:

```
ant [options] [targets...]
```

Alcune opzioni utili sono:

- ? per ottenere l'help;
- v per ottenere l'output verbose che descrive nel dettaglio tutto ciò che viene eseguito;
- f <filename> per specificare il nome del file di build (default build.xml).

Alcuni esempi sono:

```
ant compileBasic
```

per eseguire l'operazione *compileBasic* oppure:

ant

Questo comando è l'equivalente di *make* ed esegue l'operazione di default (compile).

Da riga di comando è anche possibile passare dei parametri da utilizzare all'interno del buildfile. La sintassi da utilizzare è:

```
ant -Dname=value
```

Le variabili potranno essere utilizzate normalmente nei build file con `${name}`.

Per visualizzare i target disponibili ed una breve descrizione di questi:

```
ant -projecthelp
```

1.5 Preparazione del file build.xml

Il file build.xml è l'equivalente del Makefile, ma molto più moderno (infatti è scritto in xml) e leggibile anche senza conoscenze specifiche.

Una guida completa all'uso di Ant ed alla sintassi del file build si trova in "`ANT_HOME/docs/index.html`"

A titolo esemplificativo (e senza pretesa di esaustività) viene riportato di seguito una breve guida alla costruzione di un build file.

Ogni build file contiene un *project* e molti *target* dei quali al massimo un *default target*. Ogni *target* contiene delle operazioni definite *task*. Ogni *task* può avere un *id* univoco.

1.5.1 Attributi di project

Gli attributi di ogni project sono:

name nome progetto;

default target di default;

basedir punto di partenza dal quale calcolare i path delle directory.

1.5.2 Attributi di target

I target possono essere concatenati da dipendenze.

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="B"/>
<target name="D" depends="C,B,A"/>
```

L'esecuzione di un target può essere condizionata alla presenza di una variabile.

```
<target name="build-module-A" if="module-A-present"/>
<target name="build-own-fake-module-A"
unless="module-A-present"/>
```

Gli attributi dei targets sono:

name nome del target;

depends elenco dipendenze; item[description] descrizione delle funzioni del target visualizzata con l'opzione `-projecthelp`.

1.5.3 Attributi di task

Il task è la parte di codice che viene eseguita. La struttura standard dei task è:

```
<name attribute1="value1" attribute2="value2" ... />
```

Dove *name* è il nome del target (come per esempio 'javac').

I tasks, come se fossero dei metodi java, possono essere richiamati da altri task.

```
<script ... >
    task1.setFoo("bar");
</script>
```

1.5.4 Le properties

Le properties possono essere viste come variabili locali globali o di sistema. La sintassi per definirle è:

```
<property name="propName" location="propDesc"/>
```

Generalmente vengono utilizzate per individuare cartelle dove eseguire operazioni specifiche.

Le properties predefinite sono:

basedir the absolute path of the project's basedir (as set with the basedir attribute of `project`).

ant.file the absolute path of the buildfile.

ant.version the version of Ant

ant.project.name the name of the project that is currently executing; it is set in the name attribute of `<project>`.

ant.java.version the JVM version Ant detected; currently it can hold the values 1.1, 1.2, 1.3 and 1.4.

1.5.5 Un semplice build file

```
<project name="MyProject" default="dist" basedir=".">
  <description>
    simple example build file
  </description>
  <!-- set global properties for this build -->
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="dist" location="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init"
    description="compile the source " >
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile"
    description="generate the distribution" >
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>

    <!-- Put everything in ${build} into the MyProject-${DSTAMP}
    }.jar file -->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="
    ${build}"/>
  </target>

  <target name="clean"
    description="clean up" >
```

```

    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>

```

1.5.6 Gestione dei classpath

I classpath possono essere indicati in modo semplice per ogni target.

```

<classpath>
  <pathelement path="${classpath}"/>
  <pathelement location="lib/helper.jar"/>
</classpath>

```

Oppure definendo la variabile globale 'base.path' valida per tutti i target.

```

<path id="base.path" path="${classpath}"/>

```

In alternativa esistono strutture più sofisticate che utilizzano i comandi *dirSet*, *fileSet* e *fileList*.

```

<classpath>
  <pathelement path="${classpath}"/>
  <fileset dir="lib">
    <include name="**/*.jar"/>
  </fileset>
  <pathelement location="classes"/>
  <dirset dir="${build.dir}">
    <include name="apps/**/classes"/>
    <exclude name="apps/**/*Test*"/>
  </dirset>
  <filelist refid="third-party_jars"/>
</classpath>

```

1.5.7 Generare la documentazione con javadoc

Per generare la la documentazione automatica del codice sorgente utilizzando lo strumento javadoc esiste un task apposito chiamato *javadoc*, di seguito un esempio di utilizzo.

```

<target name="generateJavadoc" depends="init"
  description="Generate API documentation using javadoc">

  <javadoc packagenames="*"
    sourcepath="src"
    classpath="{jar.classpath}"
    defaultexcludes="no"
    destdir="doc"
    author="true"
    version="true"
    use="true"
    package="true"
    windowtitle="ART - Artificial Reasoning Toolkit"
    Overview="./src/overview.html"
    Verbose="false"
  >
  <doctitle>
  <![CDATA[<h1>ART - Artificial Reasoning Toolkit</h1>]]>
  </doctitle>
</javadoc>

```

1.5.8 Utilizzo di scripts

E' possibile eseguire script windows e linux utilizzando <exec>. Un esempio per windows è riportato di seguito.

```

<exec dir="." executable="cmd" os="Windows NT">
  <arg line="/c test.bat"/>
</exec>

```

1.5.9 Verificare condizioni

Si possono eseguire specifiche operazioni solo se si verificano condizioni descritte da <condition>, paragonabili ad un blocco if.

```

<target name="check-cond">
  <condition property="cond-is-true">
    <and>
      <isset property="prop1"/>
      <isset property="prop2"/>
    </and>
    <not>
      <isset property="prop3"/>
    </not>
  </condition>

```

```
    </not>
  </and>
</condition>
</target>
```

Questo target viene eseguito solo se tutte le condizioni sono rispettate.

1.5.10 Interazione con altre applicazioni

All'interno del file build è possibile richiamare variabili esterne ed utilizzarle all'interno del processo di build attraverso, ad esempio, sintassi javascript.

```
<script language="javascript">
  propname = project.getProperty("anotherprop");
  project.setNewProperty("prop", propname);
</script>
```

1.5.11 Gestire l'output di ant

Inviare email con il risultato dell'esecuzione è sufficiente indicare da riga di comando:

```
ant -logger org.apache.tools.ant.listener.MailLogger
```

Indicando le seguenti variabili:

```
MailLogger.mailhost [default: localhost]
Mail server to use
```

```
MailLogger.port [default: 25]
Default port for SMTP
```

```
MailLogger.from [required]
Mail "from" address
```

```
MailLogger.failure.notify [default: true]
Send build failure e-mails?
```

```
MailLogger.success.notify [default: true]
Send build success e-mails?
```

```
MailLogger.failure.to [required if failure mail to be sent]
```

Address to send failure messages to

MailLogger.success.to [required if success mail to be sent]
Address to send success messages to

MailLogger.failure.subject [default: "Build Failure"]
Subject of failed build

MailLogger.success.subject [default: "Build Success"]
Subject of successful build

Le proprietà possono essere indicate sia nel buildfile con `<property>` si attraverso riga di comando con `-D` oppure in un file richiamato da:

```
ant -propertyfile <name>
```

Per salvare il risultato del processo su file:

```
ant -logfile <name>
```

1.5.12 Gestire parametri da riga di comando

Quando si compila o si esegue un applicazione è possibile passare dei parametri attraverso la riga di comando sia all'applicazione sia alla *jvm*. Per farlo vengono utilizzati i task `input`, che chiede il parametro all'utente, e `arg`, che passa il parametro specificato all'applicazione. Di seguito un semplice esempio.

```
<target name="runChange" depends="init"
  description="Run the minimize change model" >
  <input message="Input amount:" addproperty="amount"/>
  <java classname="change.Change">
    <classpath>
      <pathelement path="."/>
      <pathelement path="${build}"/>
      <pathelement path="${jar.classpath}"/>
    </classpath>
    <arg value="${amount}"/>
  </java>
</target>
```

1.6 Come estendere e personalizzare Ant

Java Ant, essendo scritto integralmente in java e utilizzando standard xml può essere facilmente esteso aggiungendo nuove funzionalità o personalizzando e modificando quelle esistenti.

Le tecniche utilizzate sono due:

1. attraverso la gestione degli eventi implementando BuildListener;
2. aggiungendo nuovi oggetti 'task'.

1.6.1 Implementare la classe BuildListener

Si possono estendere le funzionalità di ant implementando l'interfaccia BuildListener presente nell'archivio ant.jar org.apache.tools.ant

Il seguente esempio mostra come costruire un listener personalizzato che stampa quando il processo di build inizia e finisce².

Gli eventi che debbono essere gestiti sono:

- Build started
- Build finished
- Target started
- Target finished
- Task started
- Task finished
- Message logged

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

public class MyBuildListener implements BuildListener
{
    public void buildStarted( BuildEvent event )
    {
    }

    public void buildFinished( BuildEvent event )
    {
    }
}
```

²Non utilizzare System.out e System.err poiché creerebbe errori nell'esecuzione.

```

    }

    public void targetStarted( BuildEvent event )
    {
    }

    public void targetFinished( BuildEvent event )
    {
    }

    public void taskStarted( BuildEvent event )
    {
        event.getTask().log(event.getTask().getTaskName
            () + ": Task started...");
    }

    public void taskFinished( BuildEvent event )
    {
        event.getTask().log(event.getTask().getTaskName
            () + ": Task finished..." );
    }

    public void messageLogged( BuildEvent event )
    {
    }
}

```

La classe deve essere compilata indicando nel classpath *ANT_HOME/lib/ant.jar*.

Per eseguire il listener appena creato, dopo aver compilato la classe e averla posizionata nella directory principale del progetto eseguire:

```
ant -lib ./ -listener MyBuildListener
```

Esistono alcuni listener predefiniti che sono:

AnsiColorLogger Usa colori predefiniti per i messaggi a schermo;

Log4jListener Listener che manda eventi a Log4j (progetto del gruppo Apache);

MailLogger Crea messaggi con DefaultLogger, e invia e-mail con il risultato.

1.6.2 Aggiungere nuovi oggetti task

Per aggiungere nuovi task è sufficiente dichiarare quale classe contiene il codice del nuovo task attraverso *< taskdef >* nel build file:

```
<taskdef name="newtask" classname="NewTask"/>
```

Un esempio di classe java che definisce un nuovo task è il seguente(NewTask.java)

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

public class NewTask extends Task
{
    private String msg;

    public void execute() throws BuildException
    {
        System.out.println(msg);
    }
    public void setMessage(String str)
    {
        msg=str;
    }
}
```

Per compilarlo è necessario includere ant.jar nel classpath.

```
javac -classpath "./;%ANT_HOME%/lib/ant.jar" NewTask.java
```

Per usare il nuovo task semplicemente indicarlo all'interno di un target come per qualsiasi task standard:

```
<target name="newTask"
    description="ANT DEMO - New task example" >
    <newtask message="Hello World! NewTask works!"/>
</target>
```

Eeguire da riga di comando ricordandosi di indicare il classpath dove trovare la classe.

```
ant -lib ./ newTask
```

1.7 Riferimenti

Le risorse principali sono:

- Ant website <http://ant.apache.org>

- jGuru website <http://www.jguru.com/forums/home.jsp?topic=Ant>
- Extreme Programming with Ant (Glenn Niemeyer and Jeremy Poteet).

2 Appendici

A titolo di esempio ho preparato i file `buil.xml` per `jesframe-0.9.9.0` e `Art-0.1.0` allo scopo di eseguire alcune delle operazioni che vengono normalmente gestite con il comando `make`.

Per poter utilizzare `ant` con `jesframe` è sufficiente copiare il file `jes.xml` nella cartella principale `\jesframe-0.9.9.0` e rinominarlo come `build.xml`. Il build file per `Art-0.1.0` è già compreso nella distribuzione.

2.1 Build file per `jesframe-0.9.9.0`

```
<project name="jES-Frame" default="compile" basedir=".">
  <description>
    Java Enterprise Simulator
  </description>
  <!-- set global properties for this build -->
  <property name="src" location="src"/>
  <property name="build" location="classes"/>
  <property name="jar" location="lib"/>
  <property environment="env"/>
  <!--
  <echo message="Swarm home is set to = ${env.SWARMHOME}"/>
  <echo message="Ant home is set to = ${env.ANT_HOME}"/>
  -->

  <property name="swarm.classpath" location="${env.SWARMHOME}/
    share/swarm/swarm.jar;lib;/lib/plot.jar;lib/gui.jar;lib/
    xldr.jar"/>
  <property name="jar.classpath" location="lib/jesframe.jar"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
  </target>

  <target name="compile" depends="init"
    description="compile the source in the base directory" >
    <!-- Compile the java code in the basedir -->
    <javac srcdir="." destdir=".">
    <patternset id="java.pattern">
```

```

    <include name="*.java" />
  </patternset>
  <classpath>
    <pathelement path="${swarm.classpath}" />
    <pathelement path="${jar.classpath}" />
  </classpath>
</javac>
</target>

<target name="compileBasic" depends="init"
  description="compile the source in /src" >
  <!-- Compile the java code from ${src} into ${build} -->
  <javac srcdir="${src}" destdir="${build}">
    <patternset id="java.pattern">
      <include name="*.java" />
    </patternset>
    <classpath>
      <pathelement path="${swarm.classpath}" />
      <pathelement path="${jar.classpath}" />
    </classpath>
  </javac>
</target>

<target name="jar" depends="compileBasic"
  description="generate the jar file" >
  <!-- Put everything in ${build} into the jesframe.jar file
  -->
  <jar jarfile="${jar}/jesframe.jar" basedir="${build}" />
</target>

<target name="run" depends="init"
  description="run the simulation" >
  <java classname="StartESFrame">
    <arg value="Xmx200m" />
    <classpath>
      <pathelement path="." />
      <pathelement path="${swarm.classpath}" />
      <pathelement path="${jar.classpath}" />
    </classpath>
  </java>
</target>

<target name="runBig" depends="init"
  description="to be used to run big simulations" >
  <java classname="StartESFrame">
    <!-- Memory to be used by the java virtual machine is set
    to 400 MB -->
    <arg value="Xmx400m" />
    <classpath>

```

```

        <pathelement path="." />
        <pathelement path="${swarm.classpath}" />
        <pathelement path="${jar.classpath}" />
    </classpath>
</java>
</target>

<target name="runFromClasses" depends="init"
    description="run the simulation from classes in
    directory ./classes" >
    <java classname="StartESFrame">
        <arg value="Xmx200m" />
        <classpath>
            <pathelement path="./classes" />
            <pathelement path="${swarm.classpath}" />
        </classpath>
    </java>
</target>

<target name="runJar" depends="init"
    description="run the simulation from jesframe.jar" >
    <java classname="StartESFrame">
        <arg value="Xmx200m" />
        <classpath>
            <pathelement path="${swarm.classpath}" />
            <pathelement path="${jar.classpath}" />
        </classpath>
    </java>
</target>

<target name="clean"
    description="clean up classes" >
    <delete>
        <fileset dir="${build}" includes="*.class" />
        <fileset dir="." includes="*.class" />
        <fileset dir="." includes="*stackdump" />
        <fileset dir="./lib" includes="*stackdump" />
    </delete>
</target>

<target name="cleanAll"
    description="clean up classes and log" >
    <delete>
        <fileset dir="${build}" includes="*.class" />
        <fileset dir="${build}" includes="*" />
        <fileset dir="." includes="*.class" />
        <fileset dir="." includes="*stackdump" />
        <fileset dir="." includes="*" />
        <fileset dir="./lib" includes="*stackdump" />
    </delete>
</target>

```

```

    <fileset dir="{src}" includes="*" />
    <fileset dir="./unitData" includes="*" />
    <fileset dir="./log" includes="*.txt" />
    <fileset dir="./HeldOrders" includes="*.txt" />
  </delete>
</target>

<target name="cleanBackup"
  description="clean up backup" >
  <delete>
    <fileset dir="{build}" includes="*" />
    <fileset dir="." includes="*stackdump" />
    <fileset dir="." includes="*" />
    <fileset dir="./lib" includes="*stackdump" />
    <fileset dir="{src}" includes="*" />
    <fileset dir="./unitData" includes="*" />
  </delete>
</target>

<target name="cleanResults"
  description="clean up results of previous run" >
  <delete>
    <fileset dir="./log" includes="*.txt" />
    <fileset dir="./HeldOrders" includes="*.txt" />
    <fileset dir="./Benefit" includes="*.txt" />
    <fileset dir="./Costs" includes="*.txt" />
    <fileset dir="./Revenues" includes="*.txt" />
    <fileset dir="." includes="data.ratio*" />
  </delete>
</target>
</project>

```

2.2 Build file per art-0.1.0

```

<project name="ART" default="compile" basedir=".">
  <description>
    Artificial Resoing Toolkit library
  </description>
  <!-- set global properties for this build -->
  <property name="src" location="src" />
  <property name="build" location="classes" />
  <property environment="env" />

```

```

<property name="jar.classpath" location="lib/xlrd.jar"/>
<property name="swarm.classpath" location="${env.SWARMHOME}/
  share/swarm/swarm.jar"/>

<target name="init">
  <!-- Create the time stamp -->
  <tstamp/>
</target>

<target name="compile" depends="init"
  description="Compile the source files" >
  <!-- Compile the java code from ${src} into ${build} -->
  <javac srcdir="${src}" destdir="${build}">
    <classpath>
      <pathelement path="${jar.classpath}"/>
    </classpath>
  </javac>
</target>

<target name="runEasyTestCase" depends="init"
  description="Run the EasyTestCase example" >
  <java classname="basicExamples.EasyTestCase">
    <classpath>
      <pathelement path="."/>
      <pathelement path="${build}"/>
      <pathelement path="${jar.classpath}"/>
    </classpath>
  </java>
</target>

<target name="runInteractiveTestCase" depends="init"
  description="Run the InteractiveTestCase example" >
  <java classname="basicExamples.InteractiveTestCase">
    <classpath>
      <pathelement path="."/>
      <pathelement path="${build}"/>
      <pathelement path="${jar.classpath}"/>
    </classpath>
  </java>
</target>

<target name="runAgentBasedTestCase" depends="init"
  description="Run the AgentBasedTestCase example" >
  <java classname="basicExamples.AgentBasedTestCase">
    <classpath>
      <pathelement path="."/>
      <pathelement path="${build}"/>
      <pathelement path="${jar.classpath}"/>
    </classpath>
  </java>
</target>

```

```

    </java>
</target>

<target name="runJavaCournot" depends="init"
    description="Run the Cournot example model" >
    <java classname="javaCournot.Cournot">
        <classpath>
            <pathelement path="." />
            <pathelement path="${build}" />
            <pathelement path="${jar.classpath}" />
        </classpath>
    </java>
</target>

<target name="runChange" depends="init"
    description="Run the minimize change model" >
    <input message="Input amount:" addproperty="amount" />
    <java classname="change.Change">
        <classpath>
            <pathelement path="." />
            <pathelement path="${build}" />
            <pathelement path="${jar.classpath}" />
        </classpath>
        <arg value="${amount}" />
    </java>
</target>

<target name="generateJavadoc" depends="init"
    description="Generate API documentation using
    javadoc">

    <javadoc packagenames="*"
        sourcepath="src"
        classpath="${jar.classpath}"
        defaultexcludes="no"
        destdir="doc"
        author="true"
        version="true"
        use="true"
        package="true"
        windowtitle="ART - Artificial Reasoning Toolkit"
        Overview="./src/overview.html"
        Verbose="false"
    >
    <doctitle><<![CDATA[<h1>ART - Artificial Reasoning Toolkit</
        h1>]]>></doctitle>
</javadoc>

</target>

```

```
<target name="clean"
  description="Clean up classes" >
  <delete dir="${build}" />
  <mkdir dir="${build}" />
  <delete>
    <fileset dir="./lib" includes="*stackdump" />
    <fileset dir="${src}" includes="*~" />
  </delete>
</target>

<target name="cleanLogs"
  description="Clean up log files" >
  <delete>
    <fileset dir="./log" includes="*.txt" />
  </delete>
</target>

</project>
```