

Einheit 06: Building Projekts mit ANT

L.Raed

Ludwig-Maximilians-Universität München
Institut für Informatik
Programmierung und Softwaretechnik
Prof. Wirsing

2. September 2013



Inhaltsverzeichnis

1 Einführung in Ant

- Motivationen
- Einrichten von Ant
- Grundgerüst von Ant Build.xml
- Build-Aufgabe

2 Haupt-Tags

- Project Tag
- Target Tag

3 Hilfs-Tags

- Property Tag
- Fileset Tag
- Classpath Tag

4 Task Tag

- Tasks : echo, copy, mkdir, tstamp, buildnumber, delete
- Tasks: javac, javadoc, junit, jar

Motivationen

- Eclipse kann folgende Aufgaben meistern
 - Kompilieren, builden, JUnit Tests ausführen.
 - Code Dokumentation mit Javadoc Wizard erstellen.
 - Klassen als JAR Files exportieren.
 - Codeaustausch (z.B. als JARs) zwischen Entwicklern.
- Komplexe Build-Aufgaben kann Eclipse ohne Build-Tool NICHT:
 - Kompilieren, Testen, JAR erstellen in einem Schritt.
 - Kompilieren UND kopieren von Dateien in beliebigen Ordner
 - Neu Ordner erstellen UND alte Builds löschen.
 - Bei Building neuer Version E-mails an die Zuständigen schicken.
 - beliebige Kombinationen von Build-Aufgaben.

Einführung in Ant

• Das Ant Building Tool

- Ant ist ein XML basiertes Building Tool von Apache Software.
- Ant ist sehr ähnlich wie make für C++, ist aber plattformunabhängig.
- Für komplexe Build Aufgaben hat Eclipse Ant integriert.
- Dokumentation und Download von Ant Project : <http://ant.apache.org>

• Erstellen von build.xml und Starten von Ant

- Ant braucht zur Ausführung eine xml Datei (build.xml)
- Im build.xml werden die ant Tasks innerhalb Targets geschrieben.
- Ant Tasks sind über 100 Befehle z.B. javac, javadoc, copy, delete
- build.xml wird normalerweise direkt unter dem Projektordner erstellt.
- In Eclipse Explorer Package: projektname → New → File → build.xml
- In Eclipse: build.xml → Rechter Mausklick → Run As → Ant Build
- In der Console: zur build.xml Ordner wechseln dann ant eintippen.

• Es wird empfohlen, Ant von der Console zu starten!

- ant.bat nutzt: ANT_HOME, JAVA_HOME und CLASSPATH
- Nur Ant_HOME und JAVA_HOME sollten gesetzt werden.

Einrichten von Ant unter Windows

1 Download Ant

- download apache-ant-1.8.0RC1-bin.zip von <http://ant.apache.org>

2 Ant Auspacken

- apache-ant-1.8.0RC1-bin.zip in C:\ant auspacken.
- Ergebnis: unter ant existieren dann die Unterverordner: bin, docs, etc, lib

3 Addiere bin zur Path Umgebungsvariable

- set PATH=C:\ant\bin;%PATH%
- set CLASSPATH = C:\ant\lib;%CLASSPATH%
- Graphisch: Umgebungsvariable → Path editieren: ;C:\ant hinzufügen.

4 ANT_HOME definieren und setzen

- set ANT_HOME = C:\ant
- set PATH = %PATH%;%ANT_HOME%\bin\ant.bat

5 JAVA_HOME und JUNIT_HOME definieren und setzen (optional)

- set JAVA_HOME = C:\Program Files\Java\jdk1.6.0_17
- set PATH = %PATH%;%JAVA_HOME%\bin\java.exe
- set JUNIT_HOME = C:\junit
- set PATH = %PATH%;%JUNIT_HOME%\junit.jar

Einrichten von Ant unter Linux/Unix (bash/csh)

1 Download Ant

- download apache-ant-1.8.0RC1-bin.zip von <http://ant.apache.org>

2 Ant Auspacken

- apache-ant-1.8.0RC1-bin.zip in /usr/local/ant auspacken.
- Ergebnis: unter ant existieren dann die Unterverzeichnisse: bin, docs, etc, lib

3 ANT_HOME und JAVA_HOME für Linux/Unix (bash) setzen.

- export ANT_HOME=/usr/local/ant
- export JAVA_HOME=/usr/local/jdk1.6.0_17
- export PATH=\$PATH:\$ANT_HOME/bin

4 ANT_HOME und JAVA_HOME für Linux/Unix (csh) setzen.

- setenv ANT_HOME /usr/local/ant
- setenv JAVA_HOME /usr/local/jdk/jdk1.6.0_17
- set path=(\$path \$ANT_HOME/bin)

5 ANT Einrichten testen unter Windows/Linux:

- ant in der Kommando-Zeile Console eintippen: z.B. C:\ ant
- Im Erfolgsfall erscheint: build.xml does not exist! Build failed

Grundgerüst von Build.xml

1 XML Version Tag

- `<?xml version="1.0"?>`

2 Projekt Tag

- `<project name="projektName" default="RunTarget">`

3 Properties Tag (optional)

- `<property name="propertyName" value="literalWert"/>`
- `<property name="propertyName" location="path"/>`

4 Target und Tasks Tag

- `<target name=targetName depends=target1,...,n>`
- `<anttask></anttask> // Mehr als 100 anttask`
- `</target>`
- `</project>`

Build-Aufgabe

- **Gegeben sei folgendes Dummy-Projekt testant**

- Projekt testant mit 3 Packages: math, person, probieren
- Alle nötigen JARs sind in dem Ordner testant/lib zu finden.
- Alle Dokumente/Bilder sind in dem Ordner testant/assert
- Klassen: math.Fib, math.Fakultaet, person.Mensch, person.Adresse
- Main-Klasse: probieren.ProbiereMath.java
- Erzeugen Sie das Projekt und Tippen Sie den Code ein

- **Aufgabe: Ausführung von ant für das Dummy-Projekt**

- 1 Erzeugen Sie build.xml mit default-Target:RunTarget"
- 2 Überlegen Sie eine geeignete Struktur für den Build-Prozess.
- 3 Schreiben Sie Initialize und Clean Target für Ihr Build-Prozess.
- 4 Wenden sie die Tasks: javac, javadoc, junit und ant an.
- 5 Senden Sie bei Building eine Nachricht mit BuildNr an Ihre E-mail.

- **Lösung: Schrittweise nach der geeigneten Theorie im Kurs**

Project Tag

- **<project> Funktionsweise**

- Wurzelement von build.xml.
- Spezifiziert den Projektname und das Haupt-Target.
- Ant Ausführung beginnt bei dem Haupt-Target.

- **<project> Attribute: E:erforderlich, NE:nicht erforderlich**

- 1 **name**: Namen des Projekts z.B.testant"(NE).
- 2 **default**: Name des Haupt-Target z.B RunTarget"(E).
- 3 **basedir**: Wurzelverzeichnis (Ant benutzt es: Relativ → Voll-Paths).
- 4 **description**: einzeilige Projektbeschreibung.(NE).

- **<project> geschachtelte Elemente**

- 1 **<description>**: Projektbeschreibung über mehrere Zeilen.
- 2 **<property>**:definiert name/value symbolische Konstante
- 3 **<target>**: definiert ein Target, das Ant-Tasks enthalten kann.

Beispiel: <project> Tag

- **<project name=testant"default=RunTarget"basedir=". ">**
 - name =testantName des Projekts.
 - default=RunTargetName des Haupt-Target
 - basedir=". "Verezeichnis von build.xml (C:\workspace\testant)
 - Ant benutzt basedir um relative Paths zu vervollständigen.
 - Relativer Path `src` wird dann zu `C:\workspace\testant\src`
- **Grundgerüst mit <project> und seine geschachtelte Elemente**
 - `<project name=testant"default=RunTarget"basedir=". ">`
 - `<description>Ant: Dummy-Projekt</description>`
 - `<property name="src" location="src"/>`
 - `<target name="RunTarget" depends="Init,Compile">`
 - Hier kommen Init und Compile Targets
 - `</project>`

Targets

- **Ein Target ist:**

- Ein Container für ein oder zusammengehörige Tasks.
- Vergleichbar zu einer Methode, die eine/mehrere Anweisungen hat.
- Haupt-Target(Ausführungswurzel) wird in project Tag spezifiziert.
- Haupt-Target ist vergleichbar mit der Main-Methode.

- **<Target> Attribute**

- 1 **name**: ist der Name des Targets (erforderlich)
- 2 **depends**: Abhängigkeitsliste (durch komma separierter Targets)
- 3 **if**: ausgeführt nur wenn die spezifizierte Property gesetzt ist.
- 4 **unless**: ausgeführt nur wenn die spezifizierte Property nicht gesetzt ist.
- 5 **description**: was macht/wie funktioniert das Target.

- **Target Abhängigkeitsarten: <target name="x" depends="?">**

- 1 **Sammelabhängigkeit**: (Haupt-)Target hängt von mehreren Targets ab.
 - 2 **Kettenabhängigkeit**: (Haupt-)Target hängt von einem Target ab.
- Schlägt ein Task in der depends Liste fehl, hört die Ausführung auf.

Sammelabhängigkeit vs. Kettenabhängigkeit

- **Sammelabhängigkeit**

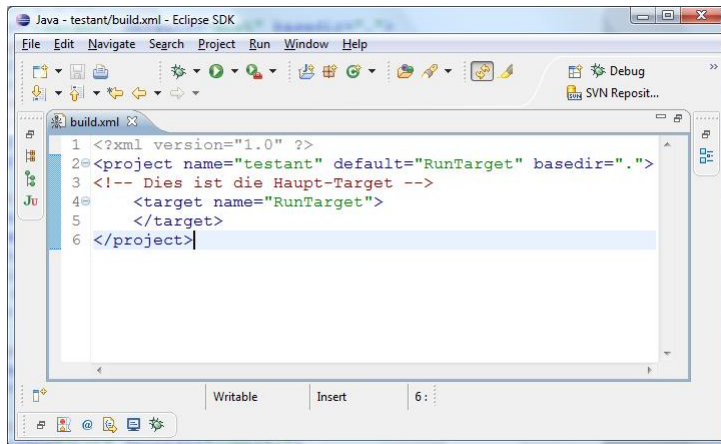
```
<?xml version="1.0"?>
<project name="testant" default="BuildMain" basedir=".">
  <target name="BuildMain" depends="init, Compile,
    Junit, Javadoc, Jar" />....
</project>
```

- **Kettenabhängigkeit**

```
<?xml version="1.0"?>
<project name="testant" default="BuildMain" basedir=".">
  <target name="BuildMain" depends="dist" />
  <target name="dist" depends="Compile">...</target>
  <target name="Compile" depends="init">...</target>
  <target name="init" depends="clean">...</target>
  <target name="clean"> <delete dir="${build}" /> </target>
</project>
```

Build-Aufgabe Teil-1 Lösung

- Erzeuge build.xml mit dem Projekttag und Haupt-Target:
- Eclipse → File → New → File → File name: build.xml
- Haupt-Target RunTarget"hängt noch von keinem Target ab.



```
1 <?xml version="1.0" ?>
2 <project name="testant" default="RunTarget" basedir=".">
3   <!-- Dies ist die Haupt-Target -->
4     <target name="RunTarget">
5     </target>
6 </project>
```

Properties und das <property> Tag

• Einführung in Properties

- Definieren die von Ant-Tasks benötigte JAR-PATH/Literale/Ordner.
- Realisieren also den Build-Prozess: welche Ordner/Path/Literale?
- Definieren eine symbolische Konstante: <name-value> Paare
- Liefern mit \${name} den Path/Literale von "value" zurück
- Beispiel: <property name="junit_homelocation=C:\junit"/>
- Property-Name \${junit_home} liefert also den value: C:\junit

• Properties definieren

- 1 Vordefiniert bei Ant z.B. java.class.path, os.name, os.version, ... usw.
- 2 In der Ant-Command-Zeile: ant -Djunit_home=C:\junit
- 3 In build.xml: <property name="name" value="path"/>

• <property> Tag und seine Attribute

- 1 name/location: definieren einen absoluten oder einen relativen Path.
- 2 name/value: definieren einen literalen Wert.
- 3 file: lädt Properties von einer Properties-File.
- 4 environment: ermöglicht Zugriff auf die Umgebungsvariablen.

<property> Beispiele

- ❶ **Path definieren:** `<property name=namelocation=path>`
 - Aboluter: `<property name="junit_homelocation=C:\junit"/>`
 - Relativer: `<property name=βrclocation=βrc"/>`
- ❷ **Literal definieren:** `<property name=name"value=literal">`
 - `<property name=Company"value=Äppache Software"/>`
 - `<property name=mathapi"value="${lib}\mathapp.jar"/>`
- ❸ **File Properties Laden:** `<property file="build.properties"/>`
 - build.properties File erzeugen
 - Properties in build.properties schreiben:: `junit home=C:\junit`
 - build.properties laden: `<property file="build.properties"/>`
- ❹ **Zugriff auf Umgebungsvariable:** `<property environment=ënv"/>`
 - Umgebungsvariable erhalten: `<property environment=ënv"/>`
 - Zugriff auf Paths aus der Umgebungsvariable: `${env.JAVA_HOME}`

Lösung: 2 Struktur von dem Build-Prozess

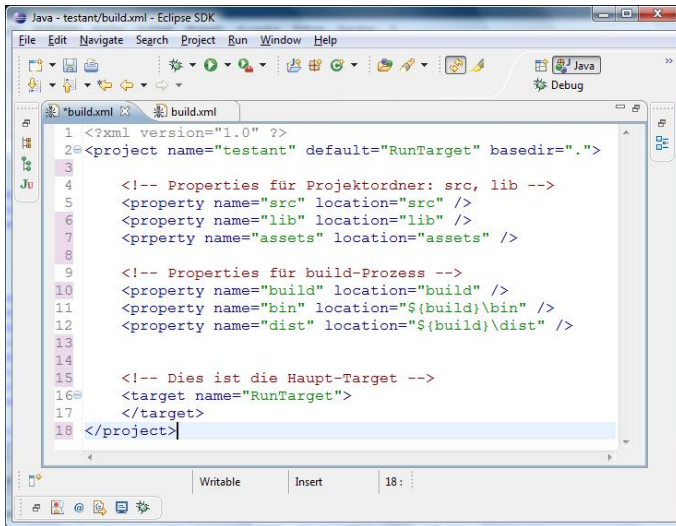
1 Lege die Building-Struktur für den Building-Prozess fest

- Vorhandene Projektordner: src, lib, assets werden für javac gebraucht.
- Build-Prozess Struktur: build, build/bin, build/dist Ordner als Property.
- Kompilierte Klassen kommen in build/bin alles anders unter dist.
- dist Unterorder doc,lib,assets werden direkt in Tasks erzeugt.
- Faustregel: Build-Prozess Struktur: klein, übersichtlich!

2 Definiere geeignete Properties für den Building-Prozess

- Für Projektordner: src, lib, assets
- Für Building-Prozess z.B: build, bin, dist.

Lösung: Teilaufgabe 2 Build-Prozess Struktur



The screenshot shows the Eclipse IDE with the 'Java - testant/build.xml - Eclipse SDK' window open. The menu bar includes File, Edit, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations, running, and debugging. The main editor displays the content of 'build.xml' with line numbers 1 through 18. The XML code defines a project named 'testant' with a default target of 'RunTarget'. It includes properties for source, library, and asset locations, as well as build and bin directories. A single target named 'RunTarget' is defined.

```
1 <?xml version="1.0" ?>
2 <project name="testant" default="RunTarget" basedir=".">
3
4     <!-- Properties für Projektordner: src, lib -->
5     <property name="src" location="src" />
6     <property name="lib" location="lib" />
7     <prperty name="assets" location="assets" />
8
9     <!-- Properties für build-Prozess -->
10    <property name="build" location="build" />
11    <property name="bin" location="${build}\bin" />
12    <property name="dist" location="${build}\dist" />
13
14
15    <!-- Dies ist die Haupt-Target -->
16    <target name="RunTarget">
17        </target>
18 </project>
```

Das <fileset> Tag

- **<fileset> Funktionsweise**

- Erlaubt das Selektieren einer/mehrere Datei(n).
- Wird oft in javac/javadoc/junit und andere Tasks benutzt.

- **<fileset> Attribute** E:=erforderlich, NE:=nicht erforderlich,default

- 1 **dir**: Wurzelverzeichnis kann src/package sein (E)
- 2 **defaultexclude**: exclude temporäre und Tools Dateien(NE,true).
- 3 **includes**: selektiert files gemäß eines Suchmusters (NE,alle)
- 4 **excludes**: ignoriert files gemäß eines Suchmusters(NE,keine)
- 5 **followsymlinke**: files, die symbolische links benutzen (NE)

- **<fileset> Geschachtelte Elemente**

- 1 **<include name=files-Suchmuster"/>**
- 2 **<exclude name=files-Suchmuster"/>**

- **Suchmuster von includes/excludes Pattern**

- 1 **?** Finde jedes Zeichen.
- 2 ***** Finde kein oder mehrere Zeichen.
- 3 ****** Finde kein oder mehrere Verzeichnisse.

<fileset> Beispiele

1 **Selektiere ein bestimmtes Package z.B. src/math**

- `<fileset dir=src\math"/>`

2 **Selektiere durch Attribute .java und ignoriere Testklassen.**

- `<fileset dir=src\math includes="**/*.java"`
- `excludes="**/*Test*"/>`

3 **Selektiere durch Elemente .java und ignoriere Testklassen.**

- `<target name="Javadoc">`
- `<javadoc destdir="${doc}" author=true" version=true use=true package=true">`
- `<fileset dir="${src}">`
- `<include name="**/*.java"/>`
- `<exclude name="**/*Test*"/>`
- `</fileset>`
-
- `</javadoc>`

Classpath Tags

- **<classpath>: Funktionsweise**

- Spezifiziert Paths von JARs/Verzeichnisse zur Ausführung von Tasks.
- Wird oft in javac/javadoc/junit und andere Tasks benutzt.

- **<classpath> Attribute:** Beide NE:=nicht erforderlich.

- **path:** Doppelpunkt oder Semikolon separierter Path
- **location:** Single file oder Verzeichnis.
- **id:** eindeutige id, die über refid angesprochen wird.

- **<classpath> geschachteltes Element <pathelement>**

- **<pathelement>:** für längere classpath. Seine Attribute: path, location
- `<pathelement path="{java.class.path}"/>` //Java-Path
- `<pathelement location="{junit_home}"/>` //JUnit-Path

<classpath> Beispiele

- **<classpath> mit id versehen**

```
<target name="Compile">  
  <javac srcdir= "${src}" destdir="${bin}">  
    <classpath id = compl>  
      <pathelement path = "${java.class.path}"/>  
      <pathelement location = "${JUNIT_HOME}\junit.jar"/>  
    </classpath></javac></target>
```

- **Auf ein <classpath> mit refid referenzieren**

```
<target name="Javadoc">  
  <javadoc destdir= "${doc}" author="true" version="true"  
    use="true" package="true">  
    <classpath refid= compl/>  
  </javadoc></target>
```

Zusammenhang: Properties, Fileset und Classpath

- **<property>**: definiert eine symbolische Konstante name-value
 - `<property name="junit_homelocation=C:\junit >`
 - Die symbolische Konstante werden von fileset und classpath benutzt.
- **<fileset>** : Selektiert die erforderlichen Dateien/Verzeichnisse
 - Selektiere ein bestimmtes Package: `<fileset dir=Brc\math">`
 - Selektiere alle .java Files aus math: `<include name= "**/*.java"/>`
 - Deselektiere davon die Testklassen: `<exclude name= "**/*Test*"/>`
 - Benutzt property symbolische Konstante und wird von Tasks benutzt.
- **<classpath>**: spezifiziert die erforderlichen Verzeichnisse/JAR
 - Spezifiziere den Java Path: `<pathelement path="$java.class.path"/>`
 - Spezifiziere die JUnit location: `<pathelement path="$junit_home"/>`

Ant Tasks

● Ein Ant Task

- kommt innerhalb eines Targets vor.
- erledigt eine bestimmte eindeutige Aufgabe.
- ist vergleichbar zu einer Anweisung in einer Methode.

● Vorteile von Ant Tasks

- Sind plattformübergreifend: Path (/,\) werden automatisch angepasst.
- Ant bietet über 100 Tasks an. (<http://wiki.apache.org/ant/FrontPage>)
- **Core-Task**: <http://ant.apache.org/manual/CoreTasks/>
- **Optional-Task**: <http://ant.apache.org/manual/OptionalTasks/>
- Wichtige Tasks: echo,copy,delete,mkdir,tstamp,javac,javadoc,jar.

echo

● **<echo> Funktionsweise**

- Schreibt eine Nachricht zu System.out, also in der Console(default).
- Kann auch eine Nachricht in einer file, log oder listener auch schreiben.
- echo wird fast in allen Tasks verwendet, um Nachrichten auszugeben.
- echo ist sehr nützlich für das Debugging von Tasks.

● **<echo> Attribute**

- **message**: der zu schreibende Nachrichttext (erforderlich)
- **file**: Output file (nicht erforderlich)
- **append**: anhängen zu einem file.

● **<echo> Beispiele**

- `<echo message= "Build successful">`
- `<echo"> Build successful </echo>`

copy

• **<copy> Funktionsweise**

- kopiert eine oder mehrere Dateien (file Attribut/filesset element).
- Beschränkung: Zielfile muss nicht existieren/älter als Quelldatei sein.
- Beschränkung aufheben: setze das Attribut `overwrite=true` statt `false`.

• **<copy> Attribute**

- 1 **file**: Quelldatei-name (erforderlich, falls kein fileset benutzt wird)
- 2 **tofile**: Zielfile-name (erforderlich, falls kein todir benutzt wird)
- 3 **todir**: Zielverzeichnis (erforderlich, falls mehr als eine Datei kopiert)
- 4 **overwrite**: überschreibt neuere Zielfile(nicht erforderlich;false)
- 5 **includeEmptyDirs**: kopiert leere Verzeichnisse(nicht erforderlich;false)
- 6 **failonerror**: Stoppt build falls file nicht gefunden(nicht erforderlich;true)
- 7 **verbose**: listet kopierte Files (nicht erforderlich; default=false)

• **<copy> Beispiele**

- `<copy file=log4j.propertiestodir="bin"/>`
- `<copy file=neu.logtofile=öld.log/>`

mkdir

• **<mkdir> Arbeitsweise**

- Erzeugt ein Verzeichnis auf der Festplatte.
- Wird z.B. im Initialize Task benutzt.
- Mit mkdir werden die nötigen Build-Verzeichnisse erzeugt.

• **<mkdir> Attribut**

- **dir**: ist das zu erzeugende Verzeichnis (Erforderlich)

• **<mkdir> Beispiel**

- `<mkdir dir="bin">`
- `<mkdir dir="dist">`
- `<mkdir dir="dist\doc /">`
- `<mkdir dir="dist\lib /">`

timestamp

- **<timestamp> Funktionsweise**

- setzt die (Zeit-) properties DSTAMP, TSTAMP, und TODAY.
- kann ein nested Element <format> haben.
- Wird bei dem Initialize Task benutzt(Datum/Uhrzeit)

- **<timestamp> geschachteltes Element**

- **<format>** ändert mit JAVA SimpleDateFormat das Zeit-Format:
DSTAMP yyyyMMdd **TSTAMP** hhmm **TODAY** MMM dd yy

- **<timestamp> Beispiel**

- <timestamp/> //Setzt timestamp danach gibt die Zeit aus:
- <echo message="Build time: \${TODAY} \${TSTAMP}"/> .

buildnumber

• <buildnumber> Funktionsweise

- Erstellt beim ersten Mal die Datei build.number
- In build.number ist der Eintrag build.number=1
- Nach jeder Ausführung erhöht build.number um 1.
- build.number ist ein Zähler (Wie oft wurde build ausgeführt).
- file Attribut erstellt statt build.number z.B build.txt
- buildnumber wird oft im Initialize Task benutzt.

• <buildnumber> Attribut

- **file** = „dateiname“: erstellt dateiname statt build.number(default)
- file ist also die zu lesende File (default: build.number)
- file Attribut ist nicht erforderlich.

• <buildnumber> Beispiel

- Erstelle stat build.number die Datei buildnumber.txt:
- <buildnumber file="buildnumber.txt" />
- Jetzt wird nach jedme build der Zähler in buildnumber.txt erhöht.

delete

• **<delete> Funktionsweise**

- löscht eine/mehrere Dateien oder ein Verzeichnis(file/fileset/dir)
- Wird in dem Clean Task benutzt, um vorhandene Build-Ordner zu löschen.

• **<delete> Attribute**

- 1 **file**: die zu löschende Datei (erforderlich, falls kein fileset/dir benutzt)
- 2 **dir**: das zu löschende Verzeichnis(erforderlich, falls kein file/fileset)
- 3 **verbos**: listet gelöschte Files (nicht erforderlich; default=false)
- 4 **failonerror**: Stoppt build bei Fehlern.(nicht erforderlich, default=true)
- 5 **includeEmptyDirs**: löscht Verzeichnisse(fileset vorhanden. NE,false)

• **<delete> Beispiele**

- Datei löschen: `<delete file=fehler.log/>`
- Verzeichnis löschen: `<delete dir="dist"/>`

Lösung Build-Teilaufgabe 3: Initialize und Clean Tasks

- **Bestimme RunTarget-Art: Sammel- oder Kettenabhängigkeit?**
 - Wir entscheiden uns z.B für `ßschrittweiseSSammelabhängigkeit`.
 - `<target name=RunTarget"depends=Ïnit"/>`
- **Initialize Target: `<target name=Ïnit">`**
 - Initialize erzeugt die Build-Prozess Struktur: build, build/bin, build/dist
 - Initialize setzt auch die Buildzeit mit `<tstamp/>`
 - Initialize hängt von keinem anderen Target ab.
- **Clean Target: `<target name=Clean">`**
 - Clean löscht die Build-Prozess Struktur: build, build/bin, build/dist
 - Clean ist nicht in der RunTarget depends-List.
 - Clean hängt auch von keinem Target ab.
 - Consloe Befehl für Clean Target: `ant Clean`

Lösung Build-Teilaufgabe 3: Initialize und Clean Tasks

```

1 <project name="testant" default="RunTarget" basedir=".>
2 <!-- Projektordner Properties -->
3 <property name="src" location="src" />
4 <property name="lib" location="lib" />
5 <property name="assets" location="assets" />
6 <!-- Build-Prozess Struktur -->
7 <property name="build" location="build" />
8 <property name="bin" location="${build}\bin" />
9 <property name="dist" location="${build}\dist" />
10 <!-- Haupt-Target als Ausführungsstartpunkt -->
11 <target name="RunTarget" depends="Init">
12     <echo message="Teste Mich..." />
13 </target>
14 <!-- Setzt Building-Zeit und erzeugt build/bin,dist Ordner -->
15 <target name="Init">
16     <tstamp>
17     </tstamp>
18     <mkdir dir="${build}" />
19     <mkdir dir="${bin}" />
20     <mkdir dir="${dist}" />
21     <echo message="----- Initialize Successful -----" />
22 </target>
23 <!-- Löscht Build-Prozess Struktur also bin und dist Ordner -->
24 <target name="Clean">
25     <delete dir="${bin}" />
26     <delete dir="${dist}" />
27     <echo message="----- Clean Successful -----" />
28 </target>
29 </project>

```

javac

• **<javac> Funktionsweise**

- Kompiliert eine oder mehrere .java aus dem spezifizierten Brcdir".
- Kompilierte Klassen .class landen in dem spezifizierten "destdirÖrdner
- Es wird standardmäßig ohne Debug Informationen kompiliert
- Für debug build muss das Attribut debug="yes"gesetzt werden.
- Oft werden 2 builds gebildet: release Build und debug Build .
- Ant orientiert sich an Verzeichnisse statt single Datei (leichtes Build)

• **<javac> Geschachtelte Elemente**

- 1 **<classpath>** kann statt classpath Attribut benutzt werden.
- 2 **<jar>**: akzeptiert die gleichen Nested Elements wie <fileset>

<javac> Attribute und Regeln

• <javac> Attribute

- 1 **srcdir**: Quellcode-Verzeichnis(Erforderlich, falls kein nested <src>)
- 2 **destdir**: Output-Verzeichnis normalerweise "bin"(Nicht erforderlich)
- 3 **includes**: die zu kompilierenden Java Datei.(NE, alle.java)
- 4 **excludes**: die nicht zu kompilierenden Java Datei.(NE, keine)
- 5 **classpath**: classpath, der benutzt wird (NE).
- 6 **debug**: schließt debug Information ein (NE, false)
- 7 **optimize**: nutzt Optimierung (NE, false)
- 8 **verbose**: bietet verbose output (NE)
- 9 **failonerror**: stoppt bei Fehlern das build (NE, true)
- 10 **encoding**: Zeichensatz z.B UTF-8 (NE)

• Regeln für <javac> Attribute

- includes und excludes Attribute können auch benutzt werden.
- srcdir referenziert oft auf den src Ordner des Projekts.
- bis auf srcdir sind alle Attribute nicht erforderlich.

<javac> Beispiele

● Kompilieren von alle .Java inklusive JUnit Klassen

- ```
<target name="Compile">
 <javac srcdir="src" destdir="bin"
 <classpath id=compl>
 <pathelement path="{java.class.path}"/>
 <pathelement location="{junit_home}"/> </classpath>
 <pathelement location="C:\log4j\apache-log4j-1.2.15.jar"/>
 <echo message="Compiled."/>
 </javac>
</target>
```

## ● Kompilieren von math package aus "\${basedir}öhne JUnit

- ```
<javac srcdir="{math}" destdir="bin"  
  includes="**\*.java"  
  excludes="**\*Test*">  
  <classpath refid = compl />  
</javac>
```

Lösung: Build-Teilaufgabe 4: Javac Task

1 Erweitere RunTarget depends-Liste um den Compile Target

```
<target name="RunTarget" depends="Init, Compile">  
  ..</target>
```

2 Definiere den Compile Target: <target name=Compile">

- Compile hängt von dem Target Initialize ab.
- Kompiliert .java aus src zur .class in build/bin.
- Definiert ein build.classpath aus lib Ordner. für Javac

Lösung: Build-Teilaufgabe 4: Javac Task

```

1 <project name="testant" default="RunTarget" basedir=".">
2   <!-- Projektordner Properties -->
3   <property name="src" location="src" />
4   <property name="lib" location="lib" />
5   <property name="assets" location="assets" />
6   <!-- Build-Prozess Struktur -->
7   <property name="build" location="build" />
8   <property name="bin" location="${build}\bin" />
9   <property name="dist" location="${build}\dist" />
10  <!-- Haupt-Target als Ausführungsstartpunkt -->
11  <target name="RunTarget" depends="Init, Compile" />
12  <!-- Setzt Building-Zeit und erzeugt build/bin,dist Ordner -->
13  <target name="Init">
14    <tstamp>
15    </tstamp>
16    <mkdir dir="${build}" />
17    <mkdir dir="${bin}" />
18    <mkdir dir="${dist}" />
19    <echo message="----- Initialize Successful -----" />
20  </target>
21  <!-- Löscht Build-Prozess Struktur also bin und dist Ordner -->
22  <target name="Clean">
23    <delete dir="${bin}" />
24    <delete dir="${dist}" />
25    <echo message="----- Clean Successful -----" />
26  </target>
27  <!-- Definiert build.classpath und kompiliert .java aus src zu build -->
28  <target name="Compile" depends="Init">
29    <javac srcdir="${src}" destdir="${bin}" />
30    <echo message="----- Compilant Successful -----" />
31  </target>

```

javadoc

• **<javadoc> Funktionsweise**

- erzeugt HTML-Dokumentation für javadoc-dokumentierter Code.
- Javadoc funktioniert für alle *.java, für ein Package oder einzige Klasse.
- Includes und Excludes Optionen sind wie bei Java Task.
- Javadoc für alle *.java: `<javadoc sourcepathref=βrc">`
- Javadoc für ein einziges Package: `<fileset dir=packagepath">`
- Javadoc für eine einzelne Klasse: `<javadoc source=βrc">`
- Das Ergebnis wird in einem Ordner abgelegt: `destdir="doc"`

• **<javadoc> Geschachtelte Elemente**

- `<fileset>` selektiert eine fileset. Ant addiert **/*.java
- `<packageset>` selektiert Package-Ordner.
- `<classpath>` setzt classpath.

javadoc Attribute

• <javadoc> Attribute

- 1 **sourcepath**: Sourceverzeichnis(E, falls kein sourcefiles/sourcepathref)
- 2 **sourcepathref**: Referenz auf Code-Ordner(E, falls kein 1,3).
- 3 **sourcefile**: Liste von Komma-separierten Klassen .java (E, falls kein 1,2)
- 4 **destdir**: Ergebnis-Ordner (E, falls doclet nicht spezifiziert ist)
- 5 **classpath**: Der Classpath (NE)
- 6 **public**: Zeigt public Klassen/Methoden/Variablen (NE)
- 7 **protected**: Zeigt protected Klassen/Methoden/Variablen (NE, true)
- 8 **package**: Zeigt package, protected und public class member(NE)
- 9 **private**: Zeigt alle Klassen und member (NE)
- 10 **version**: Zeigt @version Information (NE)
- 11 **use**: Zeigt @use Information (NE)
- 12 **author**: Zeigt @author Information (NE)
- 13 **failonerror**: Stoppt build Prozess bei Fehlern (NE)

• <javadoc> Attribute Regeln

- Standardmäßig ist sourcepath=src"(alle *.java im Projekt).

<javadoc> Beispiele

1 Javadoc für eine einzige Javaklasse

```
<javadoc destdir="doc" sourcefiles="src\math\Fib.java" />
```

2 Javadoc für ein einziges Package

```
<javadoc destdir="doc" author="true" use="true">  
  <fileset dir="${src}\math"/> (Rest wie unten)
```

3 Javadoc für alle .java ohne Testklassen

```
<javadoc destdir="doc" author="true"  
use="true" package="true">  
  <fileset dir="${src}">  
    <include name="**/*.java" />  
    <exclude name="**/*Test*" />  
  </fileset>  
  <classpath refid="compil" />  
</javadoc>
```

Lösung Build-Teilaufgabe 4: Javadoc Target

1 Erweitere RunTarget depends-Liste um Javadoc Target

```
<target name="RunTarget" depends="Init, Compile, Javadoc
```

2 Javadoc Target

- Javadoc hängt von Compile ab.
- Javadoc erzeugt HTML-Javadoc unter dist/doc
- Javadoc erzeugt also den Ordner dist/doc
- Junit-Klassen werden von Javadoc ausgeschlossen.

Lösung Build-Teilaufgabe 4: Javadoc Target

```
32<!-- Erzeugt Javadoc für alle Klassen außer Testklassen
33und speichert sie unter dist/doc -->
34<target name="Javadoc" depends="Compile">
35    <mkdir dir="${dist}\doc" />
36    <javadoc destdir="${dist}\doc" author="true" use="true" package="true">
37        <fileset dir="${src}">
38            <include name="**\*.java" />
39            <exclude name="**\*Test*" />
40        </fileset>
41    </javadoc>
42    <echo message="----- Javadoc Successful -----" />
43</target>
```

java

- **<java> Funktionsweise**

- Nutzt dieselbe ANT JVM und ruft eine -ausführbare-Klasse auf.
- Ungetestete -bad- Code stoppen Build, <java> startet neue JVM.
- Startet mit fork Attribut eine neue JVM: <java fork="yes"/>
- Erforderliche Attribute: classname oder jar (nicht zusammen)

- **<java > Attribute: classname, jar, classpath, fork,..**

- 1 classname: Name der auszuführenden Klasse(erforderlich, falls kein jar)
- 2 jar: Name der ausführbaren JAR (erforderlich, falls kein classname)
- 3 classpath: die zu benutzende Path (nicht erforderlich)
- 4 fork: führt die klasse oder JAR mit einer neuen JVM(NE,false)
- 5 failonerror: build wird bei Fehler gestoppt(NE, false)
- 6 output: Output file (NE: nicht erforderlich)
- 7 append: Anhängen oder Überschreiben von default file (NE)

- **<java > nested Elemente: <classpath> und <arg>**

- <classpath>: kann statt das classpath Attribut benutzt werden.
- <arg>: zum Spezifizieren von Command-Line Argumente

<java> Beispiele:

- Beispiel: <java> in Test-Tasks zum starten neuer JVM

- ```
<java fork="yes" classname="junit.textui.TestRunner"
taskname = "junit" failonerror="true">
 <arg value="${alltest}" />
 <classpath>
 <pathelement path="${java.class.path}" />
 <pathelement location="${bin}" />
 <pathelement location="${java_home}" />
 </classpath>
</java>
```

- Beispiel für nested Elements

- ```
<java classname="HalloWorld">
  <java classname="Add" classpath="${basedir}/bin">
    <args value=150 />
    <args value=400 />
  </java>
```

jar

• **<jar> Funktionsweise**

- komprimiert eine „set of files“ zu einer JAR file.
- JAR files können API-Bibliothek oder ausführbare JARs sein.
- Ausführbare JARs benutzen das geschachtelte Element **<manifest>**.

• **<jar> Geschachtelte Elemente**

- **<fileset dir="\$lib"/>** : Gibt das Ziel-Verzeichnis an.
- **<manifest>** Gibt Main-Class und ihr Path an:
- **<attribute name=MainClass value=Path"/>**
- **</manifest>**

• **<jar> Attribute**

- 1 **destfile**: JAR Filename (Erforderlich)
- 2 **basedir**: .class Verzeichnis (NE)
- 3 **includes**: Die zu komprimierende Java Files
- 4 **excludes**: Die für JAR zu ignorierende JAVA Files (NE)

<jar> Beispiel

- JAR Erstellung für ein Package ohne JUnit Test

```
<jar destfile="dist\mymath.jar" basedir="bin"  
include="anttest\math\**" excludes="*Test*.class" />
```

- Ausführbares JAR Erstellen

```
<target name="Jar">  
<jar destfile="Ziel\My.jar" basedir=".class Ordner">  
<manifest><attribute name="Main-Class" value="MainPath"/>  
</manifest>  
</jar>  
</target>
```

- Erstellte JAR ausführen: In der Console: ant run MyMath.jar eintippen

```
<target name="run">  
<java jar="${dist}\lib\MyMath.jar" fork="true"/>  
</target>
```

Lösung Build-Teilaufgabe 4: Ausführbares JAR

- Wir erweitern RunTarget depends-Liste um den Target Jar.
- Jar erstellt ausführbares MyMath.jar unter dist/lib Ordner.
- Um MyMath.jar zu starten schreiben wir auch run Target.
- run Target startet man in der Console mit: ant run

```

48
49     <!-- Jar erstellt uner dist\lib ein ausführbares MyMath.jar -->
50     <target name="Jar">
51         <mkdir dir="${dist}\lib" />
52         <jar destfile="${dist}\lib\MyMath.jar" basedir="${bin}">
53             <manifest>
54                 <attribute name="Main-Class" value="probieren.ProbiereMath" />
55             </manifest>
56         </jar>
57         <echo message="----- Jar Successful -----" />
58     </target>
59
60     <!-- RunJar Target zum Starten von MyMath.jar >
61     In der Console folgendes eintippen: ant RunJar MyMath.jar -->
62     <target name="RunJar">
63         <java jar="${dist}\lib\MyMath.jar" fork="true" />
64         <echo message="----- run Successful -----" />
65     </target>
66 </project>

```

JUnit Task

- **<junit> Funktionsweise**

- Führt die Test von dem JUnit testing Framework.
- hängt von junit.jar, die nicht mit Ant geliefert ist.
- Man muss also junit.jar bekannt machen z.B in lib Ordner.

- **<junit > Attribute und Geschachtelte Elemente**

- 1 Junit hat 20 Attribute und über 10 geschachtelte Elemente.
- 2 Das wichtigste Attribut ist fork, die eine neue JVM starten könnte.
- 3 **fork = "yes"**: startet eine neue JVM.
- 4 SIEHE: <http://ant.apache.org/manual/OptionalTasks/junit.html>

- **<junit > Erklären**

- 1 Oft wird das Ergebnis von junit in Report dokumentiert.
- 2 junit und Report-Erstellung werden Anhand eines Beispiels erklärt.

Beispiel: testen mit Junit

- testen mit Junit