Software Development

COMP220/COMP285

Seb Coope

Introducing Ant

These slides are mainly based on "Java Development with Ant" - E. Hatcher & S.Loughran. Manning Publications, 2003

Introducing **Ant**

Ant is Java based ***build tool*** which is

- easy to use,

- cross-platform,

- extensible, and

- scalable.

It can be used either in

- ***small*** personal or

- ***large***, multi-team ***software projects***.

What is a **build process** and **why** do we need one?

In order to build a software product, we manipulate our source code in various ways:

- compile
- generate documentation
- unit test
- package
- deploy

What is a **build process** and **why** do we need one?

Initially this can be done ***manually***.

But when we are tired of doing ***repetitive actions***, we look for ***tools***, that can ease the burden of repetitions.

Why **Ant** is a **good build tool**?

- **Ant**
  - has a very *simple syntax* which is
  - *easy to learn*
  - *easy to use*
  - *cross-platform*
  - is very *fast* — uses its own JVM, reducing start-up delays
  - does tasks' *dependency checking* to avoid doing any more work than necessary

Why **Ant** is a **good build tool**?

- *integrates* tightly with **JUnit** test framework
- easily *extensible* using **Java**
- can be used for *automated deployment*
- *de facto standard* for most open source **Java** projects

Why **Ant** is a **good build tool**?

- Because **Ant** *understands testing and deployment*, it can be used for a
  - ***unified build-test-deploy process***.
- In a software project experienced constant change, an ***automated build*** can provide a ***foundation of stability***.
- **Ant** is ***the means of controlling the building and deployment*** that would *otherwise overwhelm a team*.

The **Core Concepts** of **Ant**

To understand **Ant**, you need to understand the *core concepts of **Ant** build files*:

- **XML** format
- declarative syntax

The **Core Concepts** of **Ant**

- A build file contains one ***project*** (to build, test, deploy, etc.)
- ***Large projects*** may be composed of
  - smaller ***subprojects***, each with its own build file
  - a higher-level or ***master build file*** can ***coordinate*** the builds of ***subprojects***

The **Core Concepts** of **Ant**

- Each **Ant** ***project*** contains multiple ***targets*** *to* represent ***stages*** in the build process:
  - *compiling* source,
  - *testing*,
  - *deploying* redistributable file to a remote server,
  - etc.
- Targets can have ***dependencies*** on other targets:
  - e.g. redistributables are built, only <u>*after*</u> sources get compiled

The **Core Concepts** of **Ant**

- *Targets* contain ***tasks*** doing actual work
- **Ant** has various predefined tasks such as **`<javac>`**, **`<copy>`** and many others
- *New tasks* can easily be added to **Ant** as new Java classes

  - because **Ant** itself is implemented in **Java**
- It may be that somebody have already written a specific task you need;

  - so *you can use it* (or vice versa)

An example project

- The **next slide** shows the *conceptual view* of an ***Ant build file*** **`build.xml`**

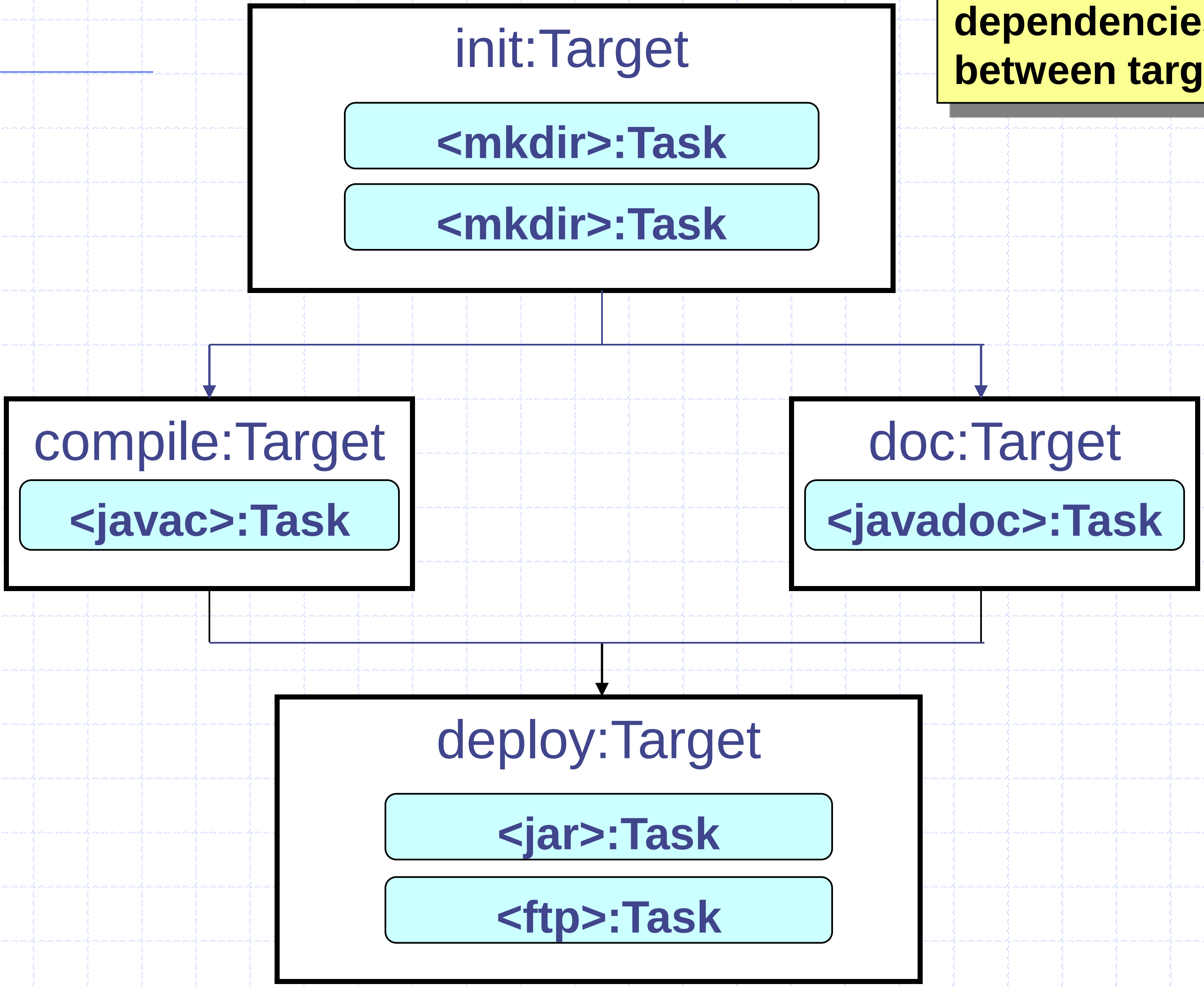
  - as a ***graph of targets***,

  - each target containing the ***tasks***.

- The **Ant** run time *determines which targets need to be executed*, and
- chooses an *order* of the execution that guarantees a target is executed after all those targets it *depends* on.
- If a task somehow *fails*, the whole build halts as *unsuccessful*.

OurProject : Project

Arrows show dependencies between targets



File **build.xml**:

```
<?xml version="1.0" ?>
<project name="OurProject" default="deploy">

  <target name="init">
    <mkdir dir="build/classes" />
    <mkdir dir="dist" />
  </target>

  <target name="compile" depends="init" >
    <javac srcdir="src"
           destdir="build/classes"
           includeAntRuntime="no"/>
  </target>

  <target name="doc" depends="init" >
     <javadoc destdir="build/classes"
                sourcepath="src"
                packagenames="org.*" />
  </target>
```

(continues)

```
  <target name="deploy" depends="compile,doc" >
    <jar destfile="dist/project.jar"
         basedir="build/classes"/>
    <ftp server="${server.name}"
         userid="${ftp.username}"
         password="${ftp.password}">
      <fileset dir="dist"/>
    </ftp>
  </target>
</project>
```

Compare yourself the values of ***depends*** attribute with the structure of the above graph.

Let us look at the output of our build to get some impression on the whole process.

```
C:\OurProject>ant -propertyfile ftp.properties
Buildfile: C:\OurProject\build.xml

init:
    [mkdir] Created dir: C:\OurProject\build\classes
    [mkdir] Created dir: C:\OurProject\dist

compile:
    [javac] Compiling 1 source file to C:\OurProject\build\classes

doc:
    [javadoc] Generating Javadoc
    ...
deploy:
    [jar] Building jar: C:\OurProject\dist\project.jar
    [ftp] sending files
    [ftp] 1 file sent

BUILD SUCCESSFUL
Total time: 5 seconds
```

An example project (cont.)

Note, that the command

```
>ant
```

invokes *by default* the file named as **`build.xml`**.

The command we used above

```
>ant -propertyfile ftp.properties
```

invokes additionally *property file*

```
ftp.properties
```

An example project (cont.)

`ftp.properties` file contains three properties (parameters)

```
server.name=ftp.texas.austin.building7.eblox.org
ftp.username=kingJon
ftp.password=password
```

The *property handling mechanism* allows *parameterisation* and *reusability* of our build file.

On the other hand, using as above the ***command-line option***

`-propertyfile`

is also ***atypical*** .

It is used in ***exceptional situations*** where ***override control*** is desired, such as ***forcing*** a build to *deploy to a server* ***other than*** *the default* **`server.name`** already described directly in **`build.xml`**.

The Beauty of Ant:

- Specify the build file correctly, and
  - **Ant** will work out dependencies and call the targets (with their tasks) in the right order.
- One or two lines of **XML** is often enough to describe what you want a task to do.

The Beauty of Ant:

- Imagine also how useful is **Ant** build file ***if a new developers join a team***.
- Imagine how many *build errors* could you make manually, without such a tool as **Ant**.
- Even very complex build repeated with **Ant** will give
  - ***always the same <u>standard</u> result***.