CSE 15L: Software Tools and Techniques Laboratory

Winter 2021 - http://ieng6.ucsd.edu/~cs15x

Gary Gillespie Keith Muller

Relevance

(Why do we need to know about XML and Ant?)

- XML is language of Internet applications to talk to each other (together with JSON)
- Ant is a build tool for Java projects, and it uses XML for describing the build properties and rules
- Many Java projects today are managed using Ant

Markup: Document Metatdata

- Markup
  - text added to the data content of a document in order to convey information about data
- Marked-up document contains
  - data and
  - information about that data (markup)
- Markup language
  - formalized system for providing markup
- Definition of markup language specifies
  - what markup is allowed
  - how markup is distinguished from data
  - what markup means …

What is XML

- XML stands for eXtensible Markup Language.
- A markup language is used to provide information about a document.
- Tags are added to the document to provide the extra information.
- HTML tags tell a browser how to display the document.
- XML tags give a reader some idea what some of the data means.

XML vs HTML

XML

- Extensible set of tags
- Content orientated
- Standard Data infrastructure
- Allows multiple output forms

HTML

- Fixed set of tags
- Presentation oriented
- No data validation capabilities
- Single presentation

XML Element Tags

- Elements (Tags) are the primary component of XML documents
- An XML element is made up of a start tag, an end tag, and data in between
- **Example**:

  <director> Matthew Dunn </director>

- Example of another element with the same value:

  <actor> Matthew Dunn </actor>

- XML tags are case-sensitive:

  <CITY> <City> <city>

- XML can abbreviate empty elements, for example:

  <married> </married> can be abbreviated to

  <married/>

- Comments start with <!- and end with ->

XML Attributes

- An attribute is a name-value pair separated by an equal sign (=)
- Example:

  <City ZIP="94608"> Emeryville </City>
- Attributes are used to attach additional, secondary information to an element
  - Named values – not hierarchical
  - Only one attribute with a given name per element
  - Order does NOT matter

XML Rules

- Tags are enclosed in angle brackets
- Tags come in pairs with start-tags and end-tags
- Tags must be properly nested
  - **<name><email>…</name></email> is not allowed**
  - **<name><email>…</email><name> is**
- Tags that do not have end-tags must be terminated by a '/'
  -
 is an html example

More XML Rules

- Tags are case sensitive.
  - **<address> is not the same as <Address>**
- XML in any combination of cases is not allowed as part of a tag
- Tags may not contain ‘<‘ or ‘&’.
- Tags follow Java naming conventions, except that a single colon and other characters are allowed
  - They must begin with a letter and may not contain white space
- Documents must have a single *root* tag (node) that begins the document

Basic XML Document

- A basic XML document is an XML element that can, but might not, include nested XML elements
- Example:

```
<books>
    <book isbn="123">
        <title> Second Chance </title>
        <author> Matthew Dunn </author>
    </book>
</books>
```

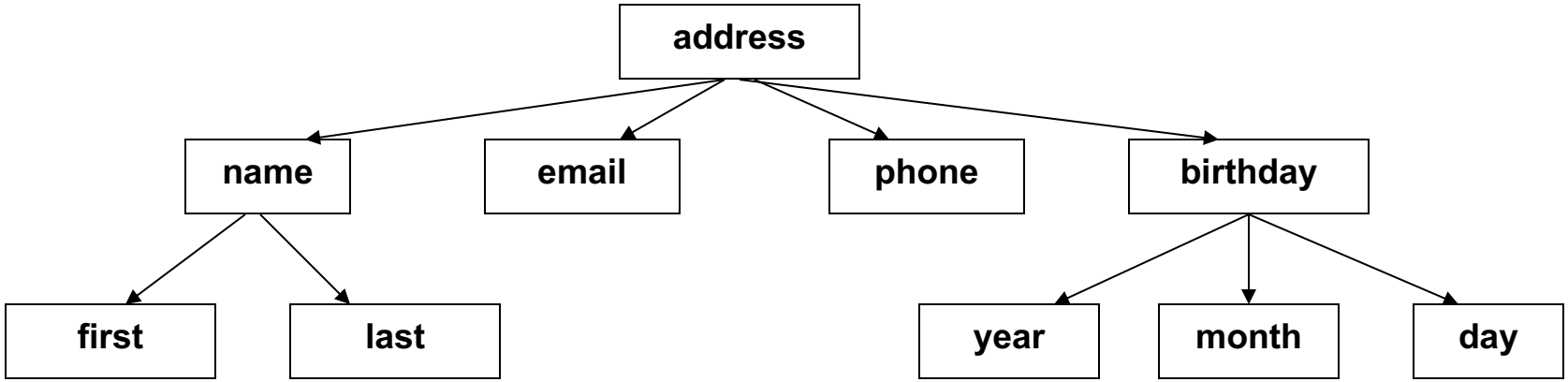
Example of an HTML Document

```
<html>
  <head><title>Example</title></head.
<body>
  <h1>This is an example of a page.</h1>
  <h2>Some information goes here.</h2>
</body>
</html>
```

Example of an XML Document

```
<?xml version="1.0"/>
<address>
   <name>Alice Lee</name>
   <email>alee@aol.com</email>
   <phone>212-346-1234</phone>
   <birthday>1985-03-22</birthday>
</address>
```

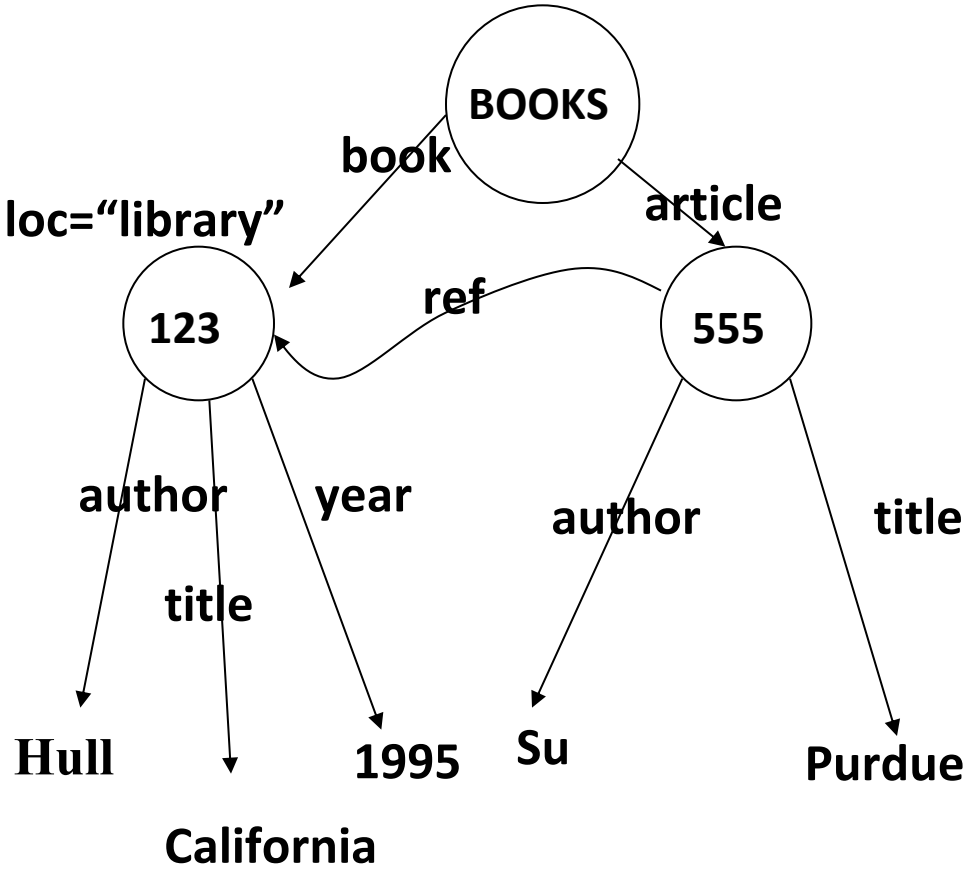
XML Files are Trees



```
<table>
    <description> People on the fourth floor </description>
        <people>
            <person>
                <name> Alan </name>
                <age> 42 </age>
                <email> agb@abc.com </ email >
            </person>
            <person>
                <name> Patsy </name>
                <age> 36 </age>
                <email> ptn@abc.com </ email >
            </person>
            <person>
                <name> Ryan </name>
                <age> 58 </age>
                <email> rgz@abc.com </ email >
            </person>
        </people>
</table>
```

Another XML Tree

```
<BOOKS>
<book id="123" loc="library">
   <author>Hull</author>
   <title>California</title>
   <year> 1995 </year>
</book>
<article id="555" ref="123">
   <author>Su</author>
   <title> Purdue</title>
</article>
</BOOKS>
```



```
<?xml version="1.0" encoding="UTF-8"?>
<note date="10-OCT-2016">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
</note>
```

Which of the tags below is NOT an attribute name?

a. to
b. from
c. date
d. Both a and b
e. None of the above

What is Ant?

- Ant is a Java based tool for automating the build process
- Ant can determine which products depend on which sources, and only build the parts that are out-of-date.
- Like make but implemented using Java
  - Platform independent commands (works on Windows, Mac & Unix)
- XML based format
  - Avoids the dreaded tab issue in make files
- Ant is an open source (free) Apache project
- According to Ant's original author, James Duncan Davidson. The name is an acronym for "Another Neat Tool".

What can we do with Ant?

- Can be used to:
  - compile java programs
  - create javadoc documentation
  - create jar, zip, tar, war files
  - delete and copy files
  - send mail
  - validate XML files
  - etc. (anything you want)

Structure of Ant

- Project
  - a top-level collection of targets
- Property
  - an Ant variable
- Target
  - a collection of tasks executed to achieve a particular purpose (a goal)
- Task
  - a unit of Ant execution (a step)

How Does Ant Work?

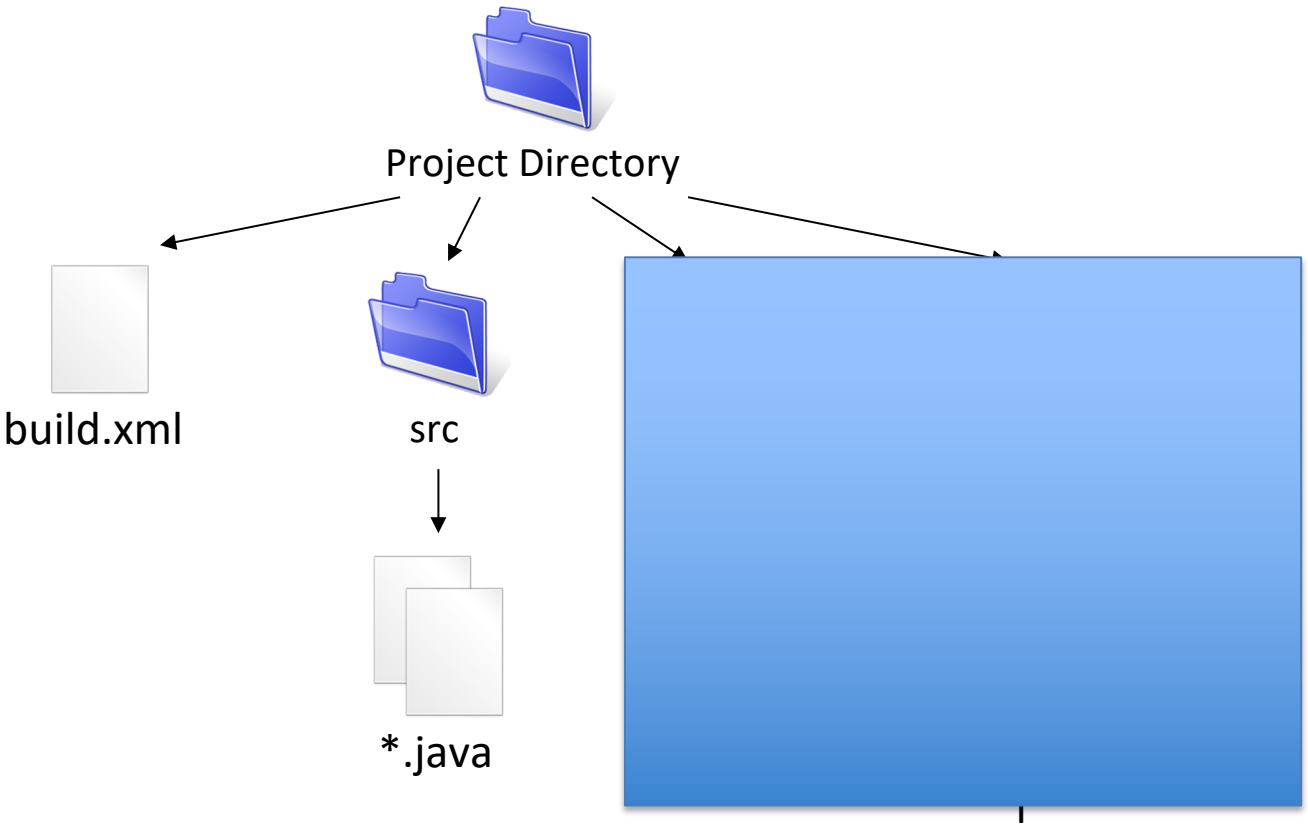
- Each Project will have a build file (build.xml)
- Each build file will contain one or more Targets
- The Target to be executed:
  - Is either explicitly selected on the command line
  - Or a project default Target is executed
- Each Target is executed only once
- Each Target will contain one or more Tasks
- Some Tasks are executed conditionally
- Tasks are implemented as Java classes

Core Ant Concept

- 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 *after* sources get compiled
- *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**

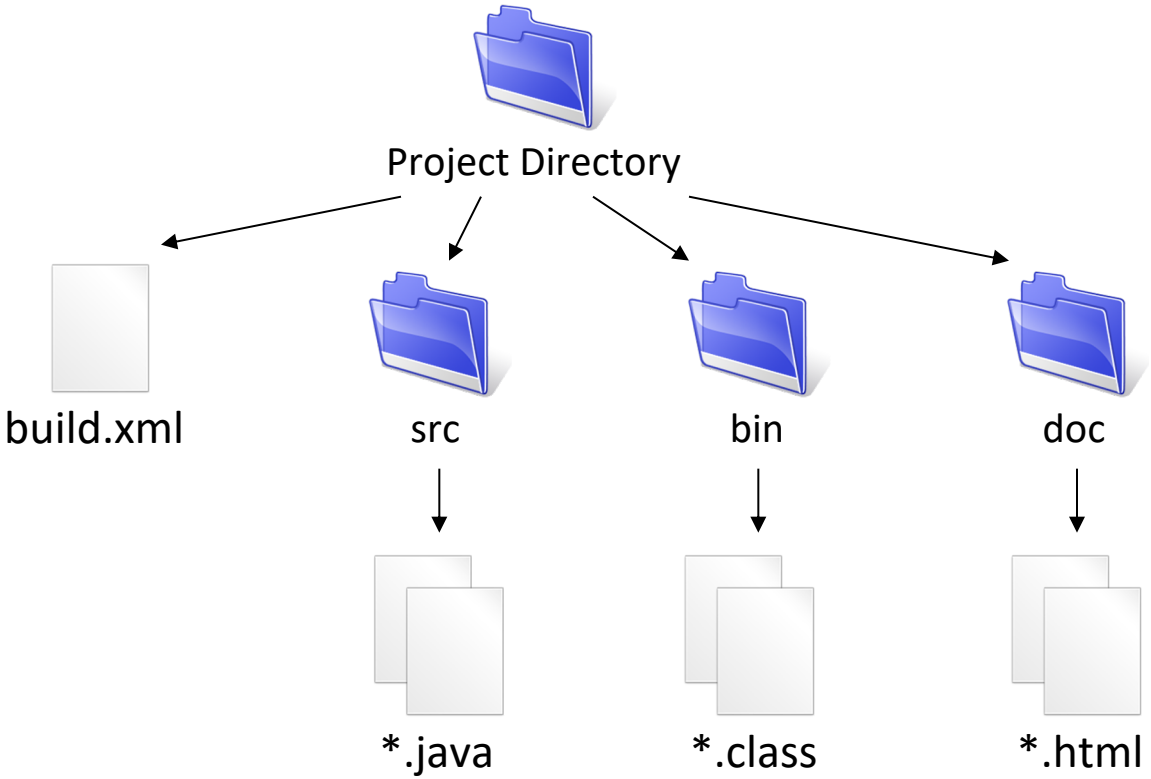
Project Organization

- A typical workspace will be organized like so…



Project Organization

- A typical workspace will be organized like so…



Anatomy of a Build File

- Ant's build files are written in XML
  - Convention is to call file build.xml
- Each build file contains
  - A project
  - At least 1 target
- Targets are composed of some number of tasks
- Build files may also contain properties
  - Like macros in a make file
- Comments are within <!-- --> blocks

Projects

- The project tag is used to define the project you wish to work with
- Projects tags typically contain 3 attributes
  - name – a logical name for the project
  - default – the default target to execute
  - basedir – the base directory for which all operations are done relative to
- Additionally, a description for the project can be specified from within the project tag

A BuildFile – Project Element

```
<project name="MyProject" default="compile" basedir=".">
    <!–- properties and targets will come here...-->
</project>
```

Properties

- Property tags consist of a name/value pair
  - Analogous to macros from make
- Properties (like global values) are defined as follows:

  <property name="propName" value="propVal" />
- Properties are XML elements without contents; therefore, we use />
- A property "propName" can be referred to later using the syntax ${propName}
- You can define any properties you want

A BuildFile – Adding Properties

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

  <property name="source.dir" location="src"/>
   <property name="build.dir" location="bin"/>
   <property name="doc.dir" location="doc"/>

   <!–- targets will come here...-->

</project>
```

Targets

- The target tag has the following required attribute
  - name – the logical name for a target
- Targets have the attributes:
  - **name:** name of the target (required)
  - **depends:** comma separated list of targets on which the target depends (optional)
  - **if, unless, description:** details omitted (read about it in the Ant documentation)
- Targets contain tasks as sub-elements, these tasks define the actions performed when the target is executed

Tasks

- A task represents an action that needs execution
- Tasks have a variable number of attributes which are task dependent
- There are several build-in tasks, most of which are things which you would typically do as part of a build process
  - Create a directory
  - Compile java source code
  - Run the javadoc tool over some files
  - Create a jar file from a set of files
  - Remove files/directories
  - And many, many others…
    - For a full list see: https://ant.apache.org/manual/tasksoverview.html

A BuildFile – Adding a Target

```
<project name="MyProject" default="compile">
   <property name="buildDir" value="build"/>
   <property name="srcDir" value="."/>
   <target name="compile">
      <javac srcdir="${src}" destdir="${build}"/>
   </target>
</project>
```

A Task

We call also have written:

<javac srcdir="." destdir="build"/>

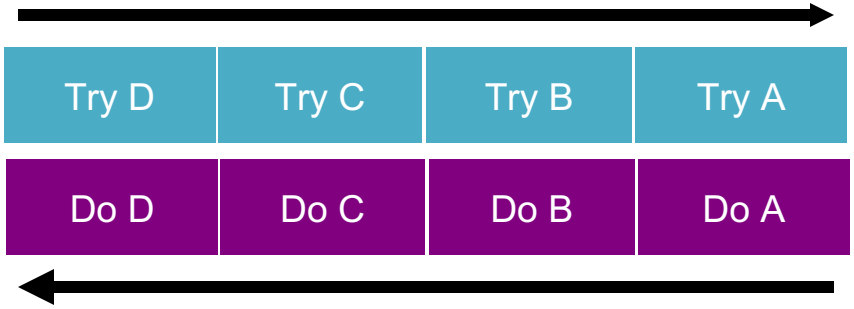
More about Depends

- Ant tries to execute the targets in “depends” from left to right.
- However, a target may be executed early when another one depends on it.

Example 1

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

- Execute: ant D
- In what order will the tasks be performed?



- Note: B is executed before C!
- Note: B is executed once!

Example 2

```
<target name="A“ depends=“B”/>
<target name="B" depends="A"/>
```

- Execute: ant A
- In what order will the tasks be performed?
- The build fails, ant reacts with:
  - “Circular dependancy: a <- b <- a”

Running Ant

- Type: ant
- Ant looks for the file: build.xml, and performs the default task specified there.
- You can use the –buildfile option to specify a different buildfile
- You can specify a different task to be performed
- You can define parameters using the –D option

Examples

- Run Ant using build.xml on the default target

```
ant
```

- Run Ant using the test.xml file on the default target

```
ant -buildfile test.xml
```

- Run Ant using the test.xml file on a target called dist:

```
ant -buildfile test.xml dist
```

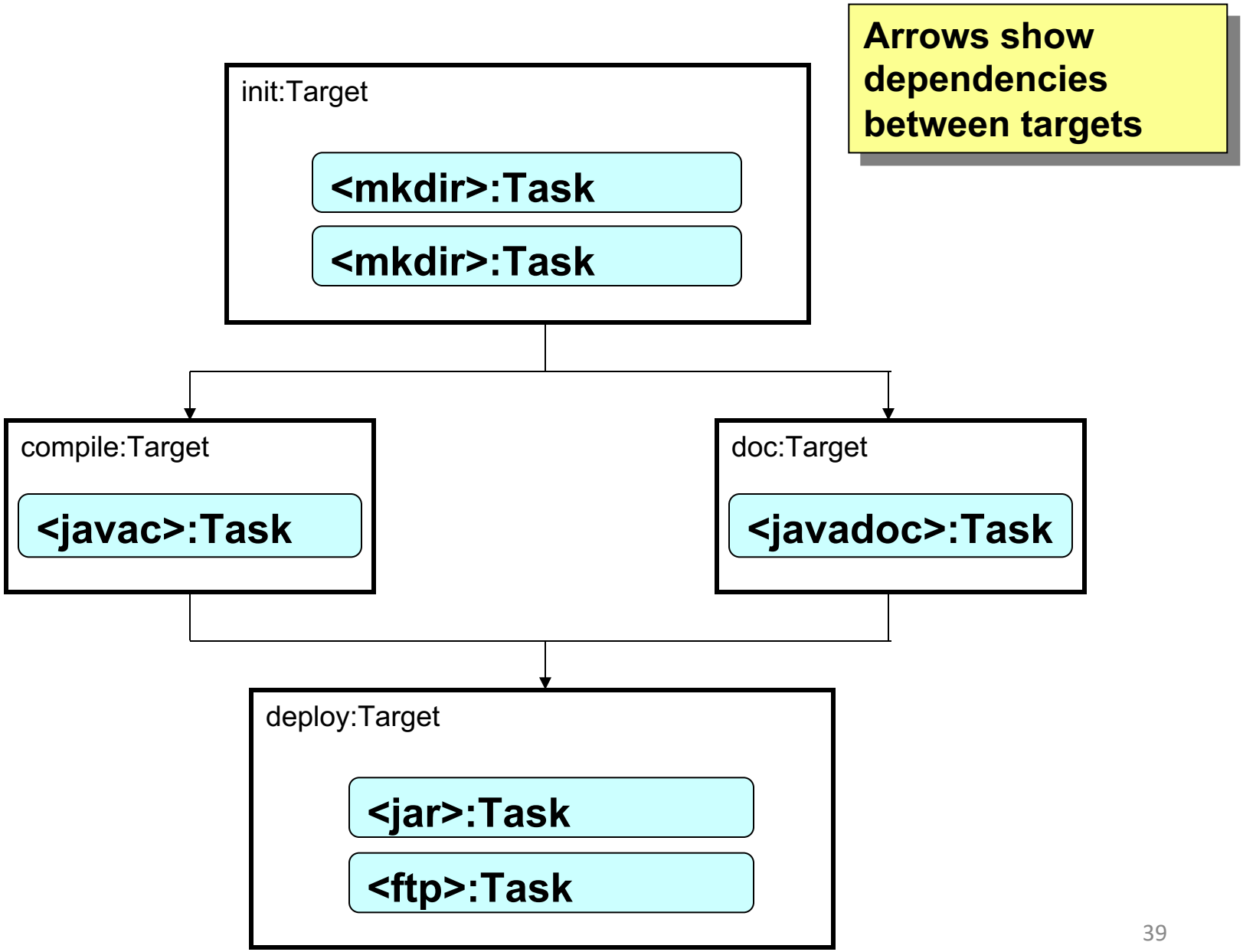
Examples (cont.)

- Run Ant using the test.xml file on a target called dist, setting the build property to the value build/classes:

```
ant -buildfile test.xml -Dbuild=build/classes dist
```

Example Project

- The **next slide** shows a *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*


Arrows show dependencies between targets
init:Target
<mkdir>:Task
<mkdir>:Task
compile:Target
<javac>:Task
doc:Target
<javadoc>:Task
deploy:Target
<jar>:Task
<ftp>:Task

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>
```

Initialization Target & Tasks

- Our initialization target creates the build and documentation directories
  - The mkdir task creates a directory

```
<project name="Sample Project" default="compile" basedir=".">
  ...

  <!-- set up some directories used by this project -->
  <target name="init" description="setup project directories">
    <mkdir dir="${build.dir}"/>
    <mkdir dir="${doc.dir}"/>
  </target>

  ...
</project>
```

Compilation Target & Tasks

- Our compilation target will compile all java files in the source directory
  - The javac task compiles sources into classes
  - Note the dependence on the init task

```
<project name="Sample Project" default="compile" basedir=".">
  ...

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

  ...
</project>
```

Javadoc Target & Tasks

- Our documentation target will create the HTML documentation
  - The javadoc task generates HTML documentation for all sources

```
<project name="Sample Project" default="compile" basedir=".">
  ...

  <!-- Generate javadocs for current project into ${doc.dir} -->
  <target name="doc" depends="init" description="generate documentation">
    <javadoc sourcepath="${source.dir}" destdir="${doc.dir}"/>
  </target>

  ...
</project>
```

Cleanup Target & Tasks

- We can also use ant to tidy up our workspace
  - The delete task removes files/directories from the file system

```
<project name="Sample Project" default="compile" basedir=".">
  ...
 <!-- Delete the build & doc directories and Emacs backup (*~) files -->
  <target name="clean" description="tidy up the workspace">
    <delete dir="${build.dir}"/>
    <delete dir="${doc.dir}"/>
    <delete>
      <fileset defaultexcludes="no" dir="${source.dir}" includes="**/*~"/>
    </delete>
  </target>
  ...
</project>
```

Completed Build File (1 of 2)

```
<project name="Sample Project" default="compile" basedir=".">

  <description>
    A sample build file for this project
  </description>

  <!-- global properties for this build file -->
  <property name="source.dir" location="src"/>
  <property name="build.dir" location="bin"/>
  <property name="doc.dir" location="doc"/>

  <!-- set up some directories used by this project -->
  <target name="init" description="setup project directories">
    <mkdir dir="${build.dir}"/>
    <mkdir dir="${doc.dir}"/>
  </target>

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

Completed Build File (2 of 2)

```
 <!-- Generate javadocs for current project into ${doc.dir} -->
  <target name="doc" depends="init" description="generate documentation">
    <javadoc sourcepath="${source.dir}" destdir="${doc.dir}"/>
  </target>

  <!-- Delete the build & doc directories and Emacs backup (*~) files -->
  <target name="clean" description="tidy up the workspace">
    <delete dir="${build.dir}"/>
    <delete dir="${doc.dir}"/>
    <delete>
      <fileset defaultexcludes="no" dir="${source.dir}" includes="**/*~"/>
    </delete>
  </target>

</project>
```

Running Ant – Command Line

- Simply cd into the directory with the build.xml file and type ant to run the project default target
  - Or type ant followed by the name of a target

- In Eclipse:
  - Eclipse comes with out of the box support for Ant
    - No need to separately download and configure Ant
  - Eclipse provides an Ant view
    - Window -> Show View -> Ant
  - Simply drag and drop a build file into the Ant view, then double click the target to run

Next Lecture

1. Unix Processes