Tools in JDK
Tools

- javac
- javadoc
- jdb
- javah
- jconsole
- jshell
- ...

...
javac
javac

- arguments
  -cp
  -encoding

- g debugging info
- g:none

-target version of bytecode (6, 7, 8, 9,...)
--release

-source version of language

-d directory for generated bytecode

...
jshell
jshell

• interactive shell
• since Java 9
javadoc
Overview

- a tool for automated generation of documentation from source codes
- class declarations etc. plus documentation comments
  - documentation directly in the code
  - easily kept up-to-date
- output – (implicitly) HTML pages
- documentation comments
  /** comment */
  - written next to a documented element
  - contains – text + special tags + html code
- the javadoc program
  - included in JDK
  - generates documentation
Comments

• written next a documented element (without any empty new lines)

```java
/** Commenting class */
public class MyClass {
    /** Commenting field */
    public int a;
    /** Commenting method */
    public void foo() {
        ...
    }
}
```
Comments

• ignored otherwise (considered as normal comments)

```java
/** ignored */
import java.util.*;

public class MyClass {
    void foo() {
        /** ignored */
    }
}
```
Multi-line comments

• comments typically over several lines
• initial spaces and stars on second and subsequent lines are ignored
• without stars, the space are not ignored (since 1.3)

/** This is a multi-line comment.
 * Initial spaces and stars
 * are ignored and removed.
 */

/** Initial spaces are not ignored as there is no star.
 */
Parts of comments

- two parts in documentation comments
  - main description
  - part with tags
- first the main description, then the part with tags
  - cannot be swapped
  - the part with tags starts with a first tag (@something)

/** This is the main description. This is
 * still the main description.
 * @see java.lang.Object
 */

- comment can have only a single section
Types of tags

- "block tags"
  - `@tag`
  - standalone tags
  - can be placed only at the beginning of a line (initial spaces and stars ignored)
    - character `@` is considered as normal character elsewhere
- "in-line tags"
  - `{@tag}`
  - can be anywhere in the text
  - also in the main description

@deprecated As of JDK 1.1, replaced by `{@link #setBounds(int,int,int,int,int)}`
Comments

• first sentence = overview
  – a sentence ends with first dot followed by a white space (or by first tag)
  – shown
    • in a overview of class elements (methods, fields)
    • in the short description of a class

• one comment for several fields
  /** A comment for both fields */
  public int x, y;
• text of comments ~ HTML
• HTML tags can be used
  /** This is a <b>documentation</b>
   * comment.
   */

• characters < > & should be written in a HTML form
  - < ... &lt;
  - > ... &gt;
  - & ... &amp;

• usage of some tags is not recommended
  - e.g. headers <h1> <h2>
  - can break the structure of generated documentation
Inheriting comments

- if the comment is not present it is inherited from parents
  - overridden methods
  - implemented methods
- inherited only the part that is not defined
  - since 1.4
  - till 1.3 – presence of documentation comment prevents inheriting of anything
- explicit inheriting `{@inheritdoc}`
Package documentation

- documentation comments for a package
- the package.html file
- in the same directory as the package
- contains a HTML page
- to the documentation, everything between the tags <body> a </body> is included
- it is written without /** ... */
- first sentence – short description of the package

- description of a group of classes
- the overview.html file
- the same structure as package.html
## Tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>since</th>
<th>Tag</th>
<th>since</th>
</tr>
</thead>
<tbody>
<tr>
<td>@author</td>
<td>1.0</td>
<td>@return</td>
<td>1.0</td>
</tr>
<tr>
<td>@{code}</td>
<td>1.5</td>
<td>@see</td>
<td>1.0</td>
</tr>
<tr>
<td>@{docRoot}</td>
<td>1.3</td>
<td>@serial</td>
<td>1.2</td>
</tr>
<tr>
<td>@deprecated</td>
<td>1.0</td>
<td>@serialData</td>
<td>1.2</td>
</tr>
<tr>
<td>@exception</td>
<td>1.0</td>
<td>@since</td>
<td>1.1</td>
</tr>
<tr>
<td>{@inheritDoc}</td>
<td>1.4</td>
<td>@throws</td>
<td>1.2</td>
</tr>
<tr>
<td>{@link}</td>
<td>1.2</td>
<td>{@value}</td>
<td>1.4</td>
</tr>
<tr>
<td>{@linkplain}</td>
<td>1.4</td>
<td>@version</td>
<td>1.0</td>
</tr>
<tr>
<td>{@literal}</td>
<td>1.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@param</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
/** Main description.
 * @param p1 description of p1
 * @param p2 description of p2
 * @throws IOException when the exception is thrown
 * @throws MyException when the exception is thrown
 * @returns what is returned
 */

int foo(int p1, long p2) throws IOException, MyException;
Other tags

- **@since text**
  - can be used everywhere
  - meaning: since which version of a sw the particular element exists
  - @since 1.4

- **@exception**
  - the same as @throws

- **@author name**
  - name of the author
  - can be used with classes, packages and overview
Other tags

- **@see** reference
  - "See also" header in the generated docs.
  - three possible formats
    - **@see** "string"
      - **@see** "The Java language specification"
    - **@see** <a href="URL#value">label</a>
    - **@see** package.class#member label
      - **@see** String#equals(Object) equals
      - **@see** java.io.File#exists() exists
  - **{@link** package.class#member label**}**
    - a reference in a text (e.g. in the main description)
    - similar to **@see**
Other tags

- `{@linkplain package.class#member label}`
  - the same as `{@link ...}`
  - printed using the same font as for plain text
    - for `{@link ...}` another font is used (typically monospaced)
- `{@deprecated text}`
  - denotes API, which should not be used (intended for removal in future)
  - text – explanation why deprecated
  - the compiler checks for this tag
    - prints out a warning if deprecated API is used
  - since 5.0 – annotation `@deprecated`
- `{@docRoot}`
  - relative path to the root directory of the generated documentation
Other tags

- `{@literal text}`
  - a text that will not be interpreted
  - `{@literal a<b>c}`
    - the generated documentation will contain \texttt{a\textless b\textgreater c}
    - \texttt{<b>} will not be interpreted as a tag
- `{@code text}`
  - the same as <code>{@literal text}</code>
javadoc

• generating documentation – javadoc
  – a part of the JDK
  – execution:

    javadoc [arguments] [packages]
    [source_files]
    [-subpackages pkg1:pkg2:...]
Arguments for javadoc

- `overview path/file`
  - a path to the file overview.html
- `public`
  - include only public elements to the documentation
- `protected`
  - include only public and protected elements
  - default behavior
- `package`
  - include public, protected and package-private elements
- `private`
  - include all elements
Arguments for javadoc

- `doclet class`
  - doclet generates the documentation
  - default doclet generates HTML
- `source 1.4`
  - version of source codes accepted
- `sourcepath list_of_paths`
  - path for source files
- `verbose  quiet`
  - level of verbosity
- `locale language_country_variant`
  - if present it must be as first argument
- `encoding encoding`
  - encoding of source files
Arguments for javadoc

- **-d path**
  - directory for generated documentation
- **-version**
  - include tag `@version`
- **-author**
  - include tag `@author`
- **-windowtitle text**
- **-doctitle text**
- **-header text**
  - placed to the beginning of each page
- **-footer text**
  - paced to the end of each page
- **-nodeprecated**
- **-nosince**
Overview

- a tool for (not only) building of Java programs
- close to `make`
- written in Java
- extensible
  - by adding classes
- input file (buildfile)
  - (as makefile in `make`)
  - XML
- NetBeans internally uses Ant for compilation, execution, ... of projects
**Buildfile**

- **default name** `build.xml`
- **contains a single** `project`
- **and at least one** `target`

```xml
<?xml version="1.0" encoding="us-ascii" ?>
<project ...>
  <target ...>
    ...
  </target>
  <target ...>
    ...
  </target>
</project>
```
Project

- attributes
  - name
    - name of the project
  - default
    - default target that will be executed if no target is explicitly given
      - mandatory attribute
  - basedir
    - a base directory for all paths in the file
- optional element <description>
  - description of the project

```xml
<project name="Project" default="compile" basedir=".">
  <description>A long description of the project</description>
</project>
```
Target

- a sequence of tasks that have to be executed
- can depend on other targets
  - is executed after them
- attributes
  - name
    - mandatory
  - depends
    - a list of targets on which the targets depend
  - description
    - short description
  - if
    - the name of a property that must be set
  - unless
    - the name of a property that must not be set
<target name="compile" depends="init"
description="Compile the app">
....
</target>
Task

- executable code
- different number of arguments
  - depends on the particular task
- core
- optional
- own

```<name attr1="value" attr2="value" .../>>
<javac srcdir="..." destdir="..."/>
```
Property

• name and value
• name – case sensitive
• obtaining the value - ${property}
• built-in properties
  - basedir
  - ant.file
  - ant.version
  - ant.project.name
  - ant.java.version
  - system properties of Java
• own properties
  - <property name="name" .... />
Example

<?xml version='1.0' encoding='us-ascii'?>
<project basedir="." default="compile" name="Project">
   <description>Project description</description>

   <property name="src" location="src"/>
   <property name="classes" location="classes"/>

   <target name="init">
      <mkdir dir="${classes}"/>
   </target>

   <target name="compile" depends="init" description="Compile">
      <javac debug="true" destdir="${classes}" srcdir="${src}" includes="**/*.java"
             classpath="${src}"/>
   </target>

</project>

<!-- continuation -->
Example

<target name="run" depends="init,compile"
description="Execute">
  <java fork="true" classname="Main"
       classpath="${classes}" />
</target>

</project>
Execution

- `ant [arguments] [target [target2 ... ]]

- arguments
  - `-projecthelp, -p`
    - project help
    - description of the project + description of tasks
  - `-propertyfile <file>`
    - defines properties from the file
  - `-D<property>=<name>`
    - definition of properties
  - `-buildfile <file>`
  - `-file <file>`
  - `-f <file>`
    - buildfile
Task javac

- executes the Java compiler
- compiles only those file that have to be compiled
  - no .class file or .class file is older than .java
  - warning!
    - only by names of files
    - i.e. does not know about inner classes, etc.
- attributes
  - srcdir
    - directory with .java files
    - mandatory
  - destdir
    - directory for .class files
  - classpath
    - CLASSPATH
Task javac

- attributes
  - encoding
    - encoding
  - source
    - -source attribute for javac
  - compiler
    - which compiler should be used
  - fork
    - true or false (default is false)
    - whether to execute the compiler in the same JVM as ANT or in a new one

- srcdir, classpath (and others) can be substituted by nested elements <src>, <classpath> (and others)
Task java

- executes a Java program
- attributes
  - `classname`
    - a class to be run
  - `jar`
    - jar-file to be run
  - mandatory either `classname` or `jar`
  - `classpath`
  - `fork`
    - run in a new JVM
- nested elements
  - `<arg>`
    - command-line arguments
Task property

- sets property(-ies) to a given value(s)
- value cannot be changed
- attributes
  - name
    - name of the property
  - value
    - value of the property
  - location
    - absolute path of the given files
  - file
    - file from which the properties should be read
  - url
    - url from which the properties should be read
Task property

• example

```xml
<property name="src" location="src"/>
<property name="foo.dist" value="dist"/>
<property file="foo.properties"/>
<property url="http://...."/>
```
Task javadoc

• runs javadoc
• attributes
  - `sourcepath` – directories with sources
  - `sourcefiles` – source files to be processed
  - `packagenames` – for which packages docs should be generated
  - `destdir` – directory for generated docs
  - `public`, `protected`, `package`, `private` – for which elements docs should be generated
  - `author` – include `@author`
  - `version` – include `@version`
  - … many others
Others

• many other tasks
  – delete
    • deletes files/directories
  – move
    • move/rename
  – mkdir
    • creating a directory
  – copy
    • copying
  – echo
    • prints out a text to the std output
Java, winter semester 2019
5.11.2019

JAVA

Maven
Overview

- http://maven.apache.org/
- a tool for managing projects
  - roughly, Maven can be seen as an Ant extension
    - but it is not an Ant extension
- provides
  - dependency management
  - project building
  - usage of “best practices”
  - extensibility by new modules
  - …
Usage

- a project generation
  mvn archetype:generate
    -DarchetypeGroupId=org.apache.maven.archetypes
    -DgroupId=com.mycompany.app
    -DartifactId=my-app

- archetype ~ a project template
- generates the following structure
Project structure

my-app
|-- pom.xml
`-- src
   |-- main
      |-- java
         |-- com
             |-- mycompany
                 |-- app
                     |-- App.java
   `-- test
      |-- java
         |-- com
            |-- mycompany
                 |-- app
                     |-- AppTest.java
POM – Project Object Model

• a project definition

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```
Build lifecycle

• mvn “phase”
  - previous phases are also executed

1. process-resources
2. compile
3. process-test-resources
4. test-compile
5. test
6. package
7. install
8. deploy
Others

• generating different project types
  
  mvn archetype:generate \  
  -DarchetypeGroupId=org.apache.maven.archetypes  
  -DarchetypeArtifactId=maven-archetype-webapp  
  -DgroupId=com.mycompany.app  
  -DartifactId=my-webapp

• generating documentation
  
  mvn archetype:generate  
  -DarchetypeGroupId=org.apache.maven.archetypes  
  -DarchetypeArtifactId=maven-archetype-site  
  -DgroupId=com.mycompany.app  
  -DartifactId=my-app-site
Gradle
Gradle

- https://gradle.org/

- similar to Maven
  - the same repositories for dependencies
  - but own language for project specification
    - DSL in Groovy
    - DSL in Kotlin

- support for multiple languages/environments
  - Java, Android, Groovy, Scala, Kotlin, C++
Project structure

- gradle init --type java-application

- build.gradle
  - gradle
    - wrapper
      - gradle-wrapper.jar
      - gradle-wrapper.properties
  - gradlew
  - gradlew.bat
  - settings.gradle
- src
  - main
    - java
      - App.java
  - test
    - java
      - AppTest.java
Gradle

• gradle build
• gradle run
• ...

• gradle tasks
  − a list of possible tasks