Object Oriented Programming

Java: installation, configuration and tools

The `main` method – revision

- The JVM interpreter executes the **`main` method** of the class indicated in the command line:
  - Qualifiers: **public static**
  - Return: **void**
  - Parameters: **String[] args**
- All classes in the application may have a `main` method. The `main` method to execute is specified each time the program is run.

Executing Java programs (1)

- Steps to execute a Java program:
  1) **Create a directory** **`dir`**
  2) **In `dir` edit** **`File.java`**
     - The first line of this file must be **`package dir;`**
     - `File.java` contain class `File`.
     - All extra classes should be in the directories indicated in the CLASSPATH.
  3) **In `dir` compile** **`javac File.java`**

     Or in the parent directory of dir **`javac dir/File.java`**
     - The option **`-cp`** can be used to indicate needed directories that were not indicated in the CLASSPATH.
     - After compiling, a **`File.class`** is created.
  4) **Go back to the parent directory of `dir`**

This correspond to the Java package!!

Executing Java programs (2)

5) Execute `java dir.File` ← You need to provide the full path from the root package (and sub-packages) until the class that contains the main!!

- The class `File` should contain the `main` method.
- The option **`-cp`** can be used to indicate needed directories that were not indicated in the CLASSPATH.
  - In Windows, for instance:
    `java -cp %CLASSPATH%;C:\libs\lib.jar Fich`
  - In Linux, for instance:
    `java -cp \usr\libs\lib.jar:$CLASSPATH Fich`
- The option **`-verbose`** can be used, that list all steps and loaded classes.

The jar tool (1)

- The **jar tool** (*Java archive tool*) manages archive files `.class`, preserving directory hierarchy.
  - The directory hierarchy should preserve the package hierarchy. For instance, the class `String` of package `java.lang`, is defined inside `/java/lang/String.java`

The jar tool (2)

- The JAR archive may contain the directory
  `META-INF/`
  where the following file can be found
  `MANIFEST.MF`
  with the information about the class to run:
  - Directives (example: version, tool)
  - Main class (class to run)
  - White line

```
Manifest-Version: 1.0
Created-By: 1.5.0_01 (Sun Microsystems Inc.)
Main-Class: project.Simulator
```

Main file .class

Sub-directory (package)

The jar tool (3)

Command	Objective
`jar cf archive.jar file-list`	Creates a jar archive with a default manifest
`jar cfm archive.jar manifest-file file-list`	Creates a jar archive with a given manifest
`jar tf archive.jar`	List archive contents
`jar xf archive.jar [file-list]`	Extract files

The jar tool (4)

- The JVM interpreter can also run a jar archive:
  `java -jar project.jar`
  The class having the main method to run is indicates in the file `META-INF/MANIFEST.MF`
- The jar program, is developed in C, and it is available in JDK.
- In Windows, the JAR archive can be opened by WinRAR.

The jar tool (5)

- **Executable**:
  - To make a jar archive file executable the **`MANIFEST.MF`** file **should contain a line corresponding to the `Main-Class`**.
  - To execute a jar archive use **option –jar**.
- **Libraries**:
  - To distribute a library one just need to make available a **jar archive with the compiled classes** (in that case the `MANIFEST.MF` file should not contain a line corresponding to the `Main-Class`).
  - To use a library one just need to compile/execute the program having the **library jar archive in the CLASSPATH**.