

# Object-Oriented Programming in

Guillaume Muller

*based on work from:*

Ch. Gravier, F. Laforest, J. Subercaze

Télécom Saint-Étienne

`{pénom.nom}@univ-st-etienne.fr`

14 September 2020

# While I'm talking :)

If you don't have another JDK/IDE:

- Download OpenJDK 8+ (LTS) / HotSpot  
<https://adoptopenjdk.net/installation.html>
- Download Eclipse IDE 2020-06  
<https://www.eclipse.org/downloads/>

This course is an **introduction**

- Does **NOT** cover **all** Java
- Goal: give you enough to understand future TD's intros
- Goes up to Java 8
- A more complete course (in French)

<http://jmdoudoux.developpez.com/cours/developpons/java/>

- Thanks Christophe Gravier for the material!



# Outline

- 1 Introduction
- 2 Functioning
  - General Compilation Chain<sup>1</sup>
  - Keywords
  - Naming Conventions
  - Pointers/References
  - Packages
  - JAR files
  - Classpath
- 3 Your turn
  - Java – Manually
  - Java – with Eclipse
- 4 References
- 5 Summary

---

<sup>1</sup>Images from "Java Head First" book.

# Java: What & Why

- Created in May **1995**
- It's an **Object-Oriented** language
- Syntax is close to C/C++ [1], similar to C# [2]
- **2nd “most used”** language in August 2020



# Java: What & Why

Aug 2020	Aug 2019	Change	Programming Language	Ratings	Change
1	2	⬆️	C	16.98%	+1.83%
2	1	⬆️	Java	14.43%	-1.60%
3	3		Python	9.69%	-0.33%
4	4		C++	6.84%	+0.78%
5	5		C#	4.68%	+0.83%
6	6		Visual Basic	4.66%	+0.97%
7	7		JavaScript	2.87%	+0.62%
8	20	⬆️	R	2.79%	+1.97%
9	8	⬆️	PHP	2.24%	+0.17%
10	10		SQL	1.46%	-0.17%
11	17	⬆️	Go	1.43%	+0.45%
12	18	⬆️	Swift	1.42%	+0.53%
13	19	⬆️	Perl	1.11%	+0.25%
14	15	⬆️	Assembly language	1.04%	-0.07%
15	11	⬆️	Ruby	1.03%	-0.28%
16	12	⬆️	MATLAB	0.86%	-0.41%
17	16	⬆️	Classic Visual Basic	0.82%	-0.20%
18	13	⬆️	Groovy	0.77%	-0.46%
19	9	⬆️	Objective-C	0.76%	-0.93%
20	28	⬆️	Rust	0.74%	+0.29%

# Java:What & Why

- In 2011:
  - **97% of machines** have a JVM in enterprises
  - **9 millions+ Java developers** in the world
  - **3 billions+ mobile devices** run Java (Android...)
- In 2020:
  - 3rd in **Job Postings**
  - 3rd in **Average Salary**



# Characteristics

- Both **Compiled** (to bytecode) & **Interpreted** (on the JVM)
- **Hardware independent** ( $\Rightarrow$  truly portable)



- **Strongly Typed**

float  
double  
boolean  
void

- **Object Oriented**

- Java manages the memory

(**NO** explicit **pointers** \o/, Garbage Collector)





There are several **distributions** of Java

- Sun/Oracle vs. OpenJDK vs. Eclipse vs. GNU...
- **JRE**: Java Runtime Environment
  - **Execution** environment only (JVM)
  - ⇒ for **end users**!
- **JDK**: Java Development Kit
  - Tools & APIs to **develop** in Java
  - ⇒ for **devs**!
- **J2EE**: Java Enterprise Edition
  - JDK + tools & APIs for **web development**

**Note: Installing a JDK also installs a JRE!**

- **Since 2017:** 1 new version every 6 months
  - New Features vs. LTS system
  - Today (2020): latest = Java 14
  - Big changes in Java 8 (Collections, Stream, **Lambdas**)
  - Java is **backward compatible**
- Check installed version (Command Prompt)  
`$ java -version`  
openjdk version "11.0.8" 2020-07-14

## Java IS NOT JavaScript



The Java platform has some **specificities**:

- A **single** (public) class per java source file
- **Compiles** source code into bytecode (hardware independent)
- This **bytecode** is executed (**interpreted**) on the **JVM** <sup>2</sup>
- **No** header files, **no** linking
- **Packages** allow to organize code (**Classes**)
- **jar** (**J**ava **A**Rchive): format to distribute an “executable”
- **CLASSPATH** indicates where to find classes

---

<sup>2</sup>Java Virtual Machine

## 1 Introduction

## 2 Functioning

- General Compilation Chain<sup>3</sup>
- Keywords
- Naming Conventions
- Pointers/References
- Packages
- JAR files
- Classpath

## 3 Your turn

- Java – Manually
- Java – with Eclipse

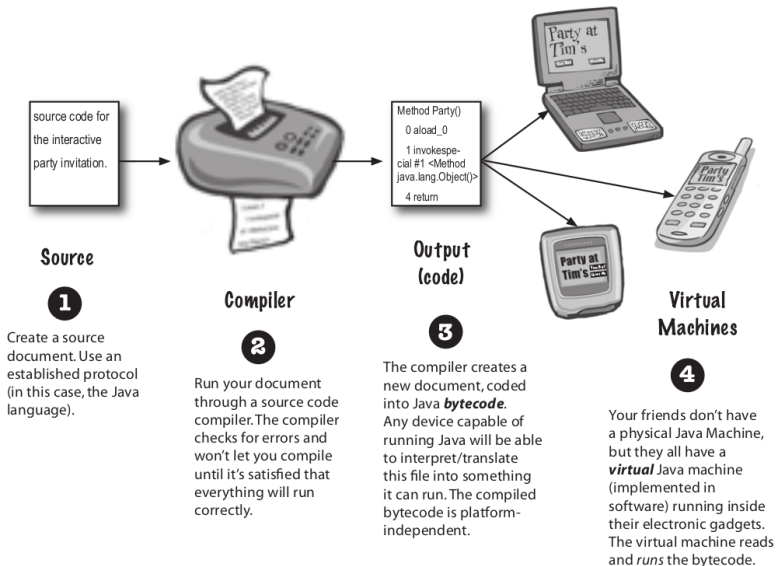
## 4 References

## 5 Summary

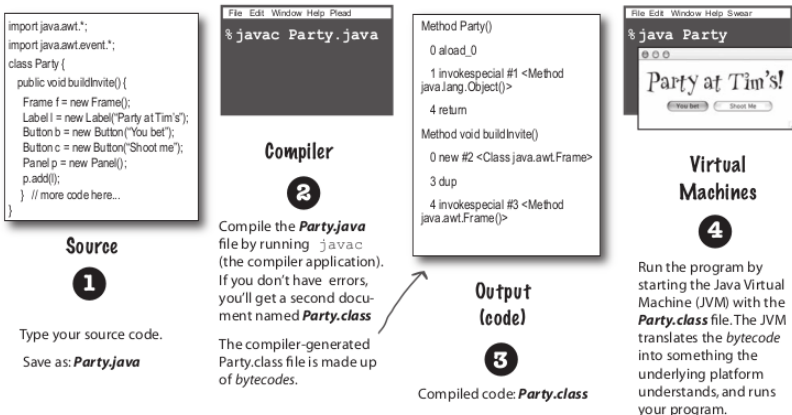
---

<sup>3</sup>Images from "Java Head First" book.

# Compilation Chain – User's View



# Compilation Chain – Developer's View



# Source code & bytecode

- Source code in `MyClass.java` file<sup>4</sup>
- File name = (public) class name
- Compile with: `javac <file>.java` (**java** compiler)
- Results in `.class` file (bytecode)
- Execute with: `javac <file>`<sup>5</sup> (JVM)

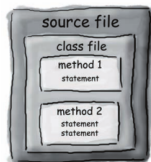
---

<sup>4</sup>One single (public) class per file

<sup>5</sup>NOTE: there is no extension here!



# Source File



**Put a class in a source file.**

**Put methods in a class.**

**Put statements in a method.**

## What goes in a source file?

A source code file (with the *.java* extension) holds one *class* definition. The class represents a *piece* of your program, although a very tiny application might need just a single class. The class must go within a pair of curly braces.

```
public class Dog {  
  
}
```

**class**

## What goes in a class?

A class has one or more *methods*. In the Dog class, the *bark* method will hold instructions for how the Dog should bark. Your methods must be declared *inside* a class (in other words, within the curly braces of the class).

```
public class Dog {  
    void bark() {  
  
    }  
}
```

**method**

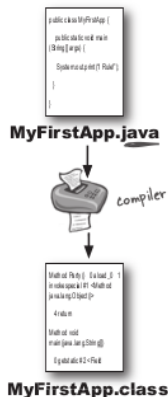
## What goes in a method?

Within the curly braces of a method, write your instructions for how that method should be performed. Method *code* is basically a set of statements, and for now you can think of a method kind of like a function or procedure.

```
public class Dog {  
    void bark() {  
        statement1;  
        statement2;  
    }  
}
```

**statements**

# Compilation Chain – First Run



```
public class MyFirstApp {  
  
    public static void main (String[] args) {  
        System.out.println("I Rule!");  
        System.out.println("The World");  
    }  
}
```

## 1 Save

MyFirstApp.java

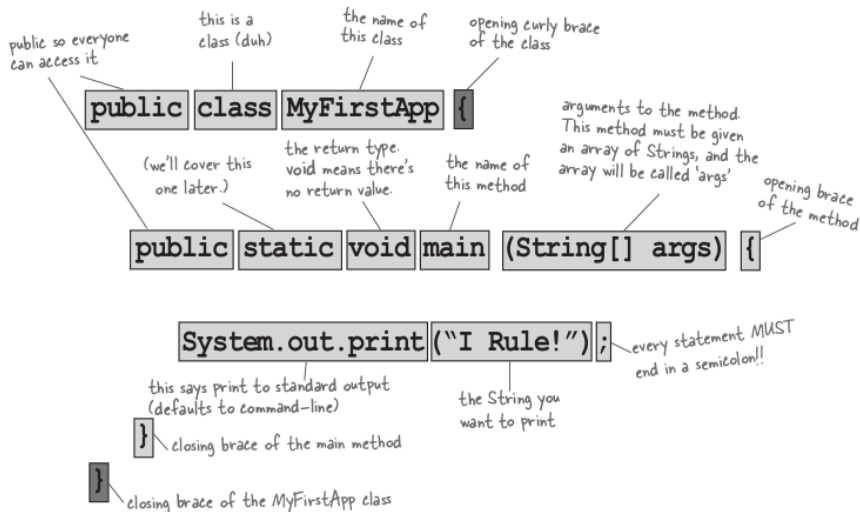
## 2 Compile

```
javac MyFirstApp.java
```

### 3 Run



# My First Class



# Outline

- 1 Introduction
- 2 Functioning
  - General Compilation Chain<sup>6</sup>
  - Keywords
  - Naming Conventions
  - Pointers/References
  - Packages
  - JAR files
  - Classpath
- 3 Your turn
  - Java – Manually
  - Java – with Eclipse
- 4 References
- 5 Summary

---

<sup>6</sup>Images from "Java Head First" book.

# Java Keywords

- **Comments:** `/* comment */`, `/** Java Doc */`, `// single line`
- **(Basic) Types:**  
`void`, `boolean`, `char`, `byte`, `short`, `int`, `float`, `long`, `double`, `[]`  
`String`, `List`, `Set`, `Map`...
- **Control Structures:**  
`for`, `while/do`, `break/continue`  
`if/else`, `switch/case`
- **Methods:** `return`, `super`
- **Packages:** `package`, `import`
- **Objects:**  
`interface/class`, `abstract`, `implements`, `extends`  
`new`, `this`  
`static`, `final`
- **Visibility:** `private`, `public`, `protected`
- **Exceptions:**  
`throw/throws`, `try/catch/finally`

# Outline

- 1 Introduction
- 2 Functioning
  - General Compilation Chain<sup>7</sup>
  - Keywords
  - **Naming Conventions**
  - Pointers/References
  - Packages
  - JAR files
  - Classpath
- 3 Your turn
  - Java – Manually
  - Java – with Eclipse
- 4 References
- 5 Summary

---

<sup>7</sup>Images from "Java Head First" book.

# Naming Conventions

- **Packages:** lowercase + dots

```
com.sun.eng
```

- **Classes / Interfaces:** (upper) CamelCase

```
class ImageSprite;  
interface RasterDelegate;
```

- **Methods:** dromedaryCase == (lower) camelCase

```
run();  
runFast();
```

- **Variables / Attributes:** dromedaryCase

```
int            i;  
char           c;  
float          myWidth;
```

- **Constants:** ALL CAPS + snake\_case

```
static final int MIN_WIDTH = 4;
```

# Outline

- 1 Introduction
- 2 Functioning
  - General Compilation Chain<sup>8</sup>
  - Keywords
  - Naming Conventions
  - Pointers/References
  - Packages
  - JAR files
  - Classpath
- 3 Your turn
  - Java – Manually
  - Java – with Eclipse
- 4 References
- 5 Summary

---

<sup>8</sup>Images from "Java Head First" book.



# Pointers and parameters

Listing 1: MyCode.cpp

```
1  Pair origin;
2  Pair *p, *q, *r;
3  origin.x = 0;
4  p = new Pair;
5  p -> y = 5;
6  q = p;
7  r = \& origin;
```

Listing 2: MyCode.java

```
1  Pair origin = new Pair();
2  Pair p, q, r;
3  origin.x = 0;
4  p = new Pair();
5  p.y = 5;
6  q = p;
7  // not possible
```

⚠ Objects in Collections **can be modified!!!**  
(behind the hood, the Collection contains references)!!!

# No delete?

- The `new` keyword creates instances

# No delete?

- The `new` keyword creates instances
- There is **no** delete in Java!!!

# No delete?

- The `new` keyword creates instances
- There is **no** `delete` in Java!!!
- **Java manages the memory itself**
- The « **Garbage Collector** » does the job *It's a module of the JVM that removes Objects from memory when they are not use (referenced) anymore*

# No delete?

- The `new` keyword creates instances
- There is **no** `delete` in Java!!!
- **Java manages the memory itself**
- The « **Garbage Collector** » does the job *It's a module of the JVM that removes Objects from memory when they are not use (referenced) anymore*



# No delete?

- The `new` keyword creates instances
- There is **no** `delete` in Java!!!
- **Java manages the memory itself**
- The « **Garbage Collector** » does the job *It's a module of the JVM that removes Objects from memory when they are not use (referenced) anymore*
- **Pros? Cons?**



# No delete?

- The new keyword creates instances
- There is **no** delete in Java!!!
- **Java manages the memory itself**
- The « **Garbage Collector** » does the job *It's a module of the JVM that removes Objects from memory when they are not use (referenced) anymore*
- **Pros? Cons?**



**Attila Szegedi**  
@asz

Follow

My kids are like Java programs; they assume there's an external garbage collector taking care of stuff they leave around.

RETWEETS  
**577**

FAVORITES  
**128**



# Outline

- 1 Introduction
- 2 Functioning
  - General Compilation Chain<sup>9</sup>
  - Keywords
  - Naming Conventions
  - Pointers/References
  - Packages
  - JAR files
  - Classpath
- 3 Your turn
  - Java – Manually
  - Java – with Eclipse
- 4 References
- 5 Summary

---

<sup>9</sup>Images from "Java Head First" book.



## Listing 3: MyClass.java

```
1 public class MyClass {  
2     public static void main(String[] args) {  
3         System.out.println("Hello World!");  
4     }  
5 }
```

- To compile (creates bytecode into `hello.class`):  
`javac MyClass.java`
- To excute (no extension! it the class name!):  
`java MyClass`
- Result:  
`|| Hello World!`

## Listing 4: HelloWorld.java

```
1 package fr.tse.java;    // Declare Package (name == directory where stored!!)
2 import java.util.List;  // Refers to another class in another Package
3
4 public class HelloWorld {
5     public static void main(String[] args) {
6         List myList = ...;
7     }
8 }
```

- Source code is organised in « packages »
- Strings separated by points (.)
- Similar to file path: **it is indeed!**  
File: fr/tse/java/HelloWorld.java
- To compile:  
javac fr/tse/java/MyClass.java
- To excute (FQN):  
java fr.tse.java.MyClass

# Packages included in Java platform

- `java.lang`: Basic classes *imported automatically*  
`String`, `Character`, `Integer`...
- `java.io`: Input/Output  
`File`...
- `java.util`: useful structures & methods  
`Collections`, `List`, `Set`, `Random`...
- `java.math`: math operations `cos()`, `sin()`...
- ...
- Extensive list in the official doc:  
<http://docs.oracle.com/javase/8/docs/api/>

# Example of importing java.util.ArrayList

## Listing 5: MyArrayList.java

```
1 package fr.tse.java;
2
3 import java.util.ArrayList;
4
5 public class MyArrayList {
6     public static void main(String[] args) {
7         System.out.println("Bonjour");
8         ArrayList<String> v = new ArrayList<String>();
9     }
10 }
```

- Description of Java's ArrayList class:  
<http://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>

# Outline

- 1 Introduction
- 2 Functioning
  - General Compilation Chain<sup>10</sup>
  - Keywords
  - Naming Conventions
  - Pointers/References
  - Packages
  - JAR files
  - Classpath
- 3 Your turn
  - Java – Manually
  - Java – with Eclipse
- 4 References
- 5 Summary

---

<sup>10</sup>Images from "Java Head First" book.

# Java ARchive (JAR files)

- Assemble all the .class files into an executable<sup>11</sup>  
`jar -cf MyClass.jar ./fr`

---

<sup>11</sup>A Java ARchive (JAR) file, with extension .jar

# Java ARchive (JAR files)

- Assemble all the .class files into an executable<sup>11</sup>  
`jar -cf MyClass.jar ./fr`
- Execute this jar file  
`java -jar MyClass.jar`

---

<sup>11</sup>A Java ARchive (JAR) file, with extension .jar

# Java ARchive (JAR files)

- Assemble all the .class files into an executable<sup>11</sup>  
`jar -cf MyClass.jar ./fr`
- Execute this jar file  
`java -jar MyClass.jar`
- Requires the name of the main Class (entry point: `main()`):
  - Config file: `./META-INF/MANIFEST.MF`:  
`Main-Class: fr.tse.java.MyClass`
  - Add this file to the jar:  
`jar -cmvf MyClass.jar ./fr`

---

<sup>11</sup>A Java ARchive (JAR) file, with extension `.jar`



# Outline

- 1 Introduction
- 2 Functioning
  - General Compilation Chain<sup>12</sup>
  - Keywords
  - Naming Conventions
  - Pointers/References
  - Packages
  - JAR files
  - Classpath
- 3 Your turn
  - Java – Manually
  - Java – with Eclipse
- 4 References
- 5 Summary

---

<sup>12</sup>Images from "Java Head First" book.

- Context:
  - My code is in a jar file
  - It imports library classes from other jar files
- Problem: How do I tell the JVM where to look for imported classes?
- Solution: The **CLASSPATH**, is a list of all the pathes where to find these librairies (both Java's and our own's)

- **CLASSPATH** = list of pathes to external code
- It is generally stored in an Environment Variable
- Each element can be:
  - A directory (with .class in its sub-directories)
  - A .jar
  - A .zip file (mostly used internally by JVM)
- Separator: “:” (Unices: Linux/MacOS) or “;” (Windows)

Example of a CLASSPATH:

```
./:/home/prog/lib/:/usr/lib/java/log4j-1.2.11.jar:../lib-ext/junit.jar
```

# Classpath – Example

`./:/home/prog/lib:/usr/lib/java/log4j-1.2.11.jar:../lib-ext/junit.jar`  
is composed of:

- All `.class` in current directory (`.` & subdirs)
- All `.class` in `/home/prog/lib/` (& subdirs)
- Classes from archive `log4j-1.2.11.jar` found in *absolute* path `/usr/lib/java/`
- Classes from archive `junit.jar` found in *relative* path `../lib-ext/`

# How to provide the **classpath** compile/run-time?

- Option `-classpath` du compilateur
  - `javac -classpath ./:/home/prog/lib/ MyClass.java`
- Option `-cp` of the JVM
  - `java -cp ./:/home/prog/lib/ MyClass`
  - `java -cp ./:/home/prog/lib/ fr.tse.java.MyClass`

# Outline

- 1 Introduction
- 2 Functioning
  - General Compilation Chain<sup>13</sup>
  - Keywords
  - Naming Conventions
  - Pointers/References
  - Packages
  - JAR files
  - Classpath
- 3 Your turn
  - Java – Manually
  - Java – with Eclipse
- 4 References
- 5 Summary

---

<sup>13</sup>Images from "Java Head First" book.

# Sump Up – Running Java Manually

Let's do a live coding demo with Text Editor+Command Line!!!

- Open a text editor
- Create a HelloWorld.java file + Type a HelloWorld class
- Compile with javac
- Run with java
- Create a fr/tse/java dir hierarchy
- Move the class in this directory
- Try to run the class  $\Rightarrow$  does not work
- Add the package directive  $\Rightarrow$  it works!!
- Create a jar file
- Try to execute it with java -jar  $\Rightarrow$  does not work
- Correct it to add ./META-INF/MANIFEST.MF file

Let's do a 2<sup>nd</sup> live coding demo, with Eclipse IDE!!!

- Start Eclipse
- Select workspace (& accept forever?)
- Create a **Java/Maven Project**
- Observe the Project herarchy is automatically created
- Create a Package ⇒ the dirs are created automatically
- Create a Class ⇒ file/class names correspond!
- Use autocompletion to create main
- Use autocompletion to enter sysout



# Outline

- 1 Introduction
- 2 Functioning
  - General Compilation Chain<sup>14</sup>
  - Keywords
  - Naming Conventions
  - Pointers/References
  - Packages
  - JAR files
  - Classpath
- 3 Your turn
  - Java – Manually
  - Java – with Eclipse
- 4 References
- 5 Summary

---

<sup>14</sup>Images from "Java Head First" book.

# References

- **Eclipse Key Bindings CheatSheet**  
[https://mootse.telecom-st-etienne.fr/pluginfile.php/9061/mod\\_resource/content/1/EclipseShortcutCheatSheet.pdf](https://mootse.telecom-st-etienne.fr/pluginfile.php/9061/mod_resource/content/1/EclipseShortcutCheatSheet.pdf)
- **A few useful CheatSheets**  
<https://dzone.com/refcardz/getting-started-eclipse>  
<https://dzone.com/refcardz/core-java>  
<https://dzone.com/refcardz/getting-started-java-gui>  
<https://dzone.com/refcardz/apache-maven-2>  
<https://dzone.com/refcardz/getting-started-git>  
<https://www.jrebel.com/system/files/git-cheat-sheet.pdf>  
<https://www.jrebel.com/search/results?keys=cheat%20sheet>
- **Tutorials sites**  
<https://www.baeldung.com/>  
<https://www.tutorialspoint.com/java/index.htm>
- **News sites**  
<https://dzone.com/>  
<https://jrebel.com/resources/>  
<https://www.infoq.com/>
- **Complete course**  
<http://jmdoudoux.developpez.com/cours/developpons/java/>
- **Glossary**  
<http://mindprod.com/jgloss/jcheat.html>
- **Books**
  - UK NoStarch, O'Reilly, Manning, Packt
  - FR Eyrolles, Dunod
- **Programming cleanly**  
[https://www.worldcat.org/title/java-by-comparison-become-a-java-craftsman-in-70-examples/oclc1035454599&referer=brief\\_results](https://www.worldcat.org/title/java-by-comparison-become-a-java-craftsman-in-70-examples/oclc1035454599&referer=brief_results)  
<https://github.com/GMTSE/ProjetsJavaMaterial/blob/master/JavaByComparisonSumUp.md>

# Outline

- 1 Introduction
- 2 Functioning
  - General Compilation Chain<sup>15</sup>
  - Keywords
  - Naming Conventions
  - Pointers/References
  - Packages
  - JAR files
  - Classpath
- 3 Your turn
  - Java – Manually
  - Java – with Eclipse
- 4 References
- 5 Summary

---

<sup>15</sup>Images from "Java Head First" book.

- Java follows **OOP** paradigm
- Java is **widely** used
- Code in `.java` files (filename = Class name)
- Code organized in **packages** (many provided in Java platform)
- Compile  
`javac MyClass.java`
- Execute  
`java MyClass`
- We will learn **Java 8** features
- We will use **Eclipse** to write code
- We will use **Maven** to compile/manage deps

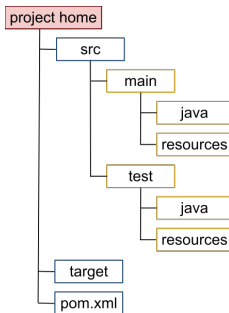
- Goal: Manage compilation chain & dependencies

- Goal: Manage compilation chain & dependencies
- Uses conventions

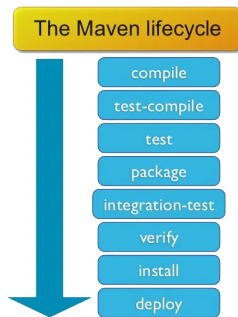
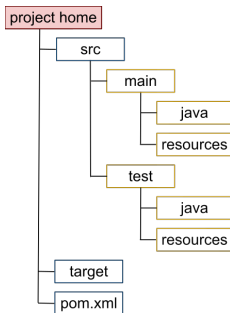
---

<sup>16</sup><https://dzone.com/refcardz/apache-maven-2>

- Goal: Manage compilation chain & dependencies
- Uses conventions
  - Project architecture



- Goal: Manage compilation chain & dependencies
- Uses conventions
  - Project architecture
  - Pre-configured rules: clean, compile...





- Goal: Manage compilation chain & dependencies
- Uses conventions
- Configuration via `pom.xml` file

- Goal: Manage compilation chain & dependencies
- Uses conventions
- Configuration via pom.xml file
- Example usage:  
`mvn clean compile`

- Goal: Manage compilation chain & dependencies
- Uses conventions
- Configuration via `pom.xml` file
- Example usage:  
`mvn clean compile`
- We will let Eclipse do (most of) that for us!!

---

<sup>16</sup><https://dzone.com/refcardz/apache-maven-2>

# Maven – Example pom.xml file

```
1  <project>
2  <!-- model version is always 4.0.0 for Maven 2.x POMs -->
3  <modelVersion>4.0.0</modelVersion>
4  <!-- project coordinates, i.e. a group of values which uniquely identify this project -->
5  <groupId>com.mycompany.app</groupId>
6  <artifactId>my-app</artifactId>
7  <version>1.0</version>
8  <!-- library dependencies -->
9  <dependencies>
10 <dependency>
11 <!-- coordinates of the required library -->
12 <groupId>junit</groupId>
13 <artifactId>junit</artifactId>
14 <version>3.8.1</version>
15 <!-- this dependency is only used for running and compiling tests -->
16 <scope>test</scope>
17 </dependency>
18 </dependencies>
19 </project>
```

- (Good way to create a portfolio)
- Distributed Management of source code files, diffs, versions. . .
- Repository: local copy vs. centrally share repo
- Git manages only files, **not** (directly) directories
- To create a repository:  
`git init mylocalcopy/`
- To get an existing repository:  
`git clone https://the-repo-url`     *From here, we are inside mylocalcopy/*
- To tell git to manage a given file:  
`git add myfile.java`
- To check what git thinks of the files:  
`git status [myfile.java]`
- To tell git to “store/accept” the new version locally:  
`git commit -m "My message" myfile.java`
- To tell git to “push” the new version to everybody (remote):  
`git push`
- To get the latest version from the others (remote):  
`git pull`

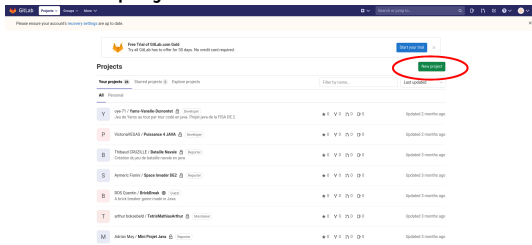
- Create an account

[https://github.com/join?ref\\_cta=Sign+up](https://github.com/join?ref_cta=Sign+up)

- Create an account

[https://github.com/join?ref\\_cta=Sign+up](https://github.com/join?ref_cta=Sign+up)

- Create a project



- Create an account

[https://github.com/join?ref\\_cta=Sign+up](https://github.com/join?ref_cta=Sign+up)

- Create a project

The screenshot shows the 'New project' form in the GitLab web interface. The form is titled 'New project' and includes a 'Blank project' tab. The form fields are as follows:

- Project name:** A text input field with a red circle around it.
- Project URL:** A text input field with a red circle around it.
- Project slug:** A text input field with a red circle around it.
- Project description (optional):** A text area field.
- Visibility level:** A dropdown menu with 'Private' selected, highlighted by a red circle.

The form also includes a 'Create project' button and a 'Cancel' button. The 'Project name' and 'Project URL' fields are also highlighted with red circles.



- Create an account

[https://github.com/join?ref\\_cta=Sign+up](https://github.com/join?ref_cta=Sign+up)

- Create a project

- Get it:

```
git clone https://gitlab.com/<user>/<project>
```

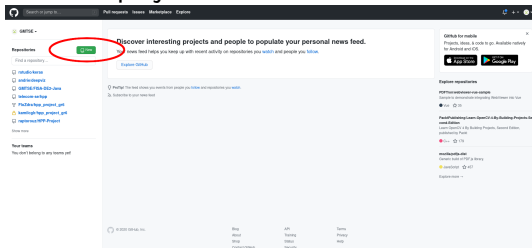
- Create an account

[https://gitlab.com/users/sign\\_in#register-pane](https://gitlab.com/users/sign_in#register-pane)

- Create an account

[https://gitlab.com/users/sign\\_in#register-pane](https://gitlab.com/users/sign_in#register-pane)

- Create a project



- Create an account

[https://gitlab.com/users/sign\\_in#register-pane](https://gitlab.com/users/sign_in#register-pane)

- Create a project

**Create a new repository**  
A repository contains all project files, including the revision history. Already have a project repository elsewhere?  
[Import a repository](#)

**Owner \*** Repository name \*

☐ **Public** Public  
Anyone in the internet can view this repository. You choose who can commit.

☐ **Private**  
You choose who can view and commit to this repository.

**Initialize this repository with:**  
Skip this step if you're importing an existing repository.

☐ **Add a README file**  
This is where you can write a brief description for your project. [Learn more](#)

☐ **Add .gitignore**  
Choose which files not to track from a list of templates. [Learn more](#)

☐ **Choose a license**  
A license lets others know how they can and can't use your code. [Learn more](#)

[Create repository](#)

- Create an account

[https://gitlab.com/users/sign\\_in#register-pane](https://gitlab.com/users/sign_in#register-pane)

- Create a project

- Get it:

```
git clone https://github.com/<user>/<project>.git
```

- Install the eGit plugin

<https://www.youtube.com/embed/WSdIbqw7Kz4>

- Import the repository

[https://www.youtube.com/embed/Fv\\_5KEN6Ix4](https://www.youtube.com/embed/Fv_5KEN6Ix4)

- Clone the content

<https://www.youtube.com/embed/r77J-r0Pv74>