

# The Java Environment

---

## Object Oriented Programming

Cataldo Basile

<cataldo.basile@polito.it>




**SoftEng**  
<http://softeng.polito.it>

Version 3:1:0  
© Maurizio Morisio, Marco Torchiano, 2018



# Java Timeline

---

- 1991:  Sun develops a programming language for cable TV set-top boxes
  - ◆ Simple, OO, platform independent
- 1994: Java-based web browser (HotJava),
  - ◆ The idea of “applet” appears
- 1996: first version of Java (1.0)

See also: <http://oracle.com.edgesuite.net/timeline/java/>

# Java timeline (cont' d)

---

- 1996: Netscape supports Java
  - ◆ Java 1.02 released,
- 1997: Java 1.1 released, major leap over for the language
- 1998: Java 2 platform (v. 1.2) released (libraries)
- 2000: J2SE 1.3 (platform enhancements, HotSpot)

# Java timeline (cont' d)

---

- 2002: J2SE 1.4 (several new APIs), e.g.
  - ◆ XML
  - ◆ Logging
- 2005: J2SE 5.0 (Language enhancements)
  - ◆ Generics
- 2006: Java SE 6 (Faster Graphics)
  - ◆ goes open source
- 2010: Acquisition by **ORACLE®**
- 2011: Java SE 7 (I/O improvements)

# Java timeline (cont' d)

---

- 2014: Java SE 8 (Language evolution)
  - ◆ Lambda expressions
  - ◆ Functional paradigm
- 2017: Java 9 releases (21 / 9)
  - ◆ Modularization,
  - ◆ jshell
- 2018: Java 10 (expected 20 / 3)
  - ◆ Local `var` type inference

# Java Ecosystem

---

- Java language
- Java platform
  - ◆ JVM
  - ◆ Class libraries (API)
  - ◆ SDK

# Java development environment

---

- Java SE 8  
(<http://www.oracle.com/technetwork/java/javase>)
  - ♦ **javac** compiler
  - ♦ **jdb** debugger
  - ♦ **JRE** (Java Run Time Environment)
    - JVM
    - Native packages (awt, swing, system, etc)
- Docs
  - ♦ <http://docs.oracle.com/javase/8/>
- Eclipse: <http://www.eclipse.org/>
  - ♦ Integrated development environment (IDE)
  - ♦ Eclipse IDE for Java Developers  
<https://eclipse.org/downloads/packages/eclipse-ide-java-developers/oxygen2>

# Java – Classes

---

- There is only one first level concept: the **class**

```
public class First {  
}
```

- The source code of a class sits in a *.java* file having the *same name*
  - ◆ Rule: one file per class
  - ◆ Enforced automatically by IDEs
  - ◆ Case-wise name correspondence



# Java – Methods

---

- In Java there are no functions, but only methods within classes
- The execution of a Java program starts from a special method:

```
public static void main(String[] args)
```

- Note 

```
In C: int main(int argc, char* argv[])
```

  - ◆ return type is `void`
  - ◆ `args[0]` is the first argument on the command line (after the program name)

# Example: source code

---

File: First.java:

```
public class First {  
    public static void main(String[] args) {  
        int a;  
        a = 3;  
        System.out.println(a) ;  
    }  
}
```

# Coding conventions

---

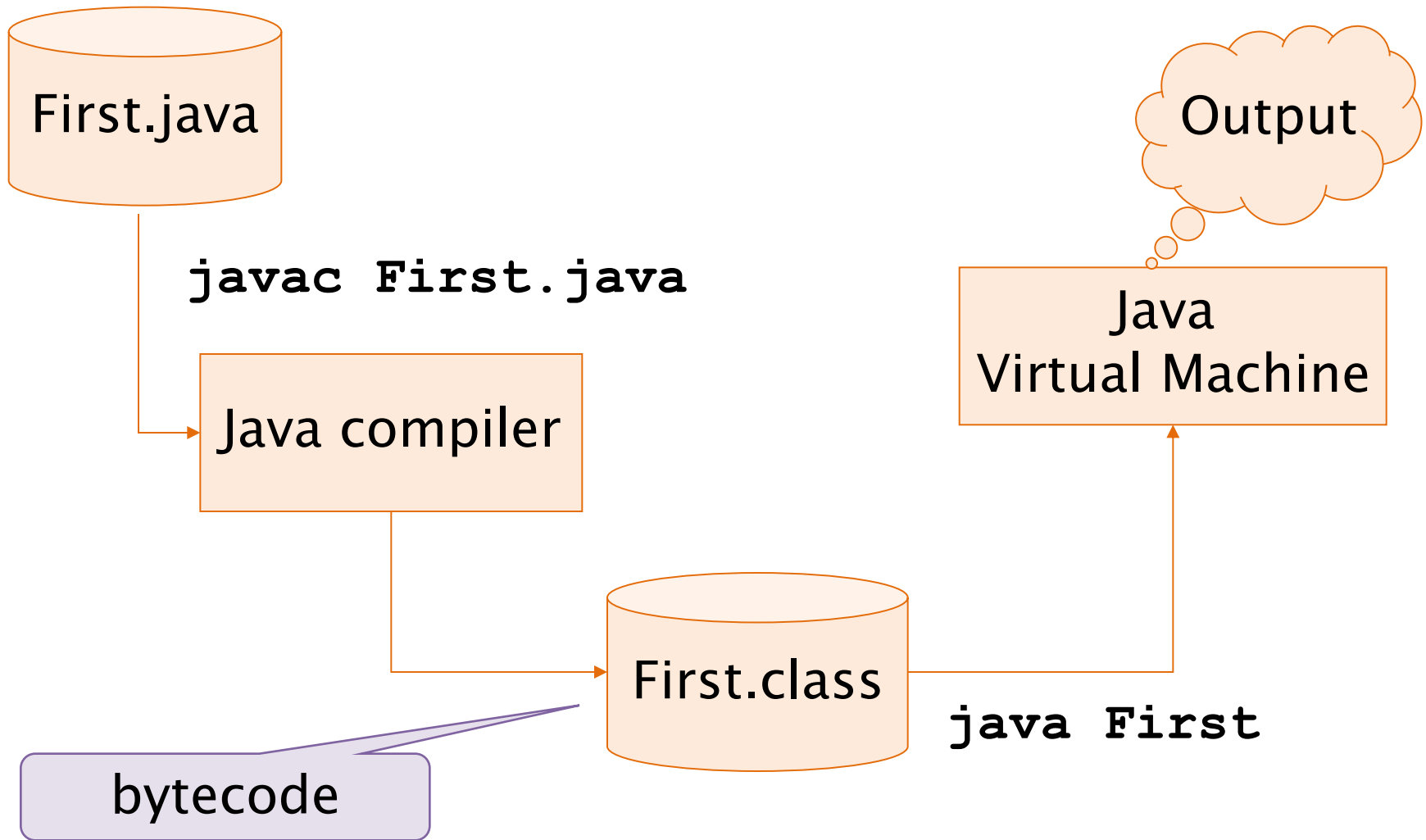
- Use **camelBackCapitalization** for compound names, not underscore
- Class name must be **C**apitalized
- Method names, object instance names, attributes, method variables must all start in lowercase
- Constants must be all uppercase (w/ underscore)
- Indent properly

# Coding conventions (example)

---

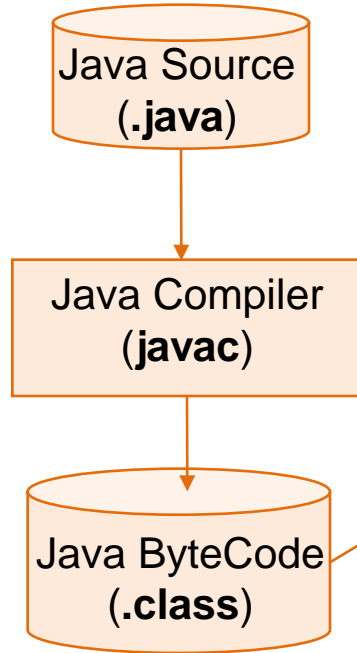
```
class ClassName {  
  
    final static double PI = 3.14;  
  
    private int attributeName;  
  
        public void methodName {  
            int var;  
            if ( var==0 ) {  
                }  
        }  
    }  
}
```

# Build and run

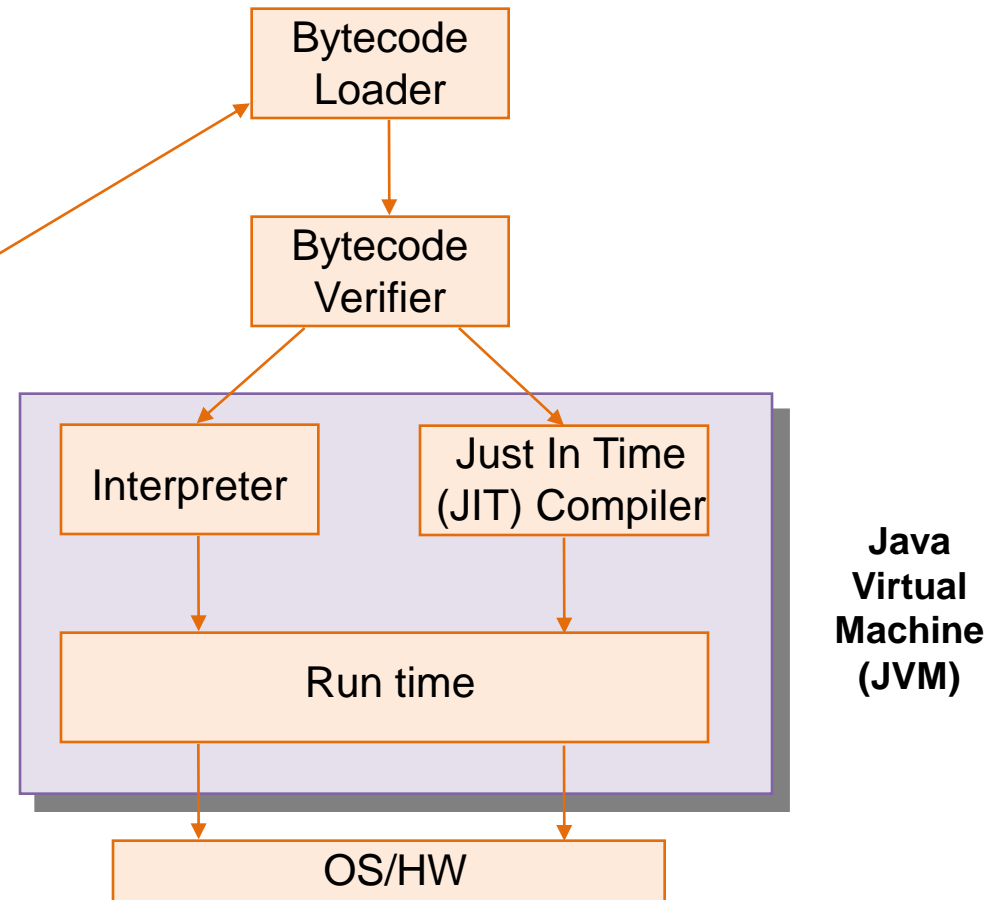


# Build and run (zoom in)

## Build environment



## Run-Time environment



# Dynamic class loading

---

- JVM loading is based on the **classpath**:
  - ◆ locations whence classes can be loaded
- When class X is required:
  - ◆ For each location in the classpath:
    - Look for file X.class
    - If present load the class
    - Otherwise move to next location

# Example: execution

---

Name of the class

- Command: `java First`
  - ◆ Take the name of the class (`First`)
  - ◆ Look for the bytecode for that class
    - In the classpath (and possibly `./`)
  - ◆ Load the class's bytecode
    - And perform all due initializations
  - ◆ Look for the `main()` method
  - ◆ Start execution from the `main()` method



---

# JAVA FEATURES

# OO language features

---

- OO language provides constructs to:
  - ◆ Define classes (types) in a hierarchic way (inheritance)
  - ◆ Create/destroy objects dynamically
  - ◆ Send messages (w/ dynamic binding)
- No procedural constructs (pure OO language)
  - ◆ no functions, class methods only
  - ◆ no global vars, class attributes only

# Java features

---

- Platform independence (portability)
  - ◆ Write once, run everywhere
  - ◆ Translated to intermediate language (bytecode)
  - ◆ Interpreted (with optimizations, i.e. JIT)
- High dynamicity
  - ◆ Run time loading and linking
  - ◆ Dynamic array sizes

# Java features (cont' d)

---

- Robust language, less error prone
  - ◆ Strong type model and no explicit pointers
    - Compile-time checks
  - ◆ Run-time checks
    - No array overflow
  - ◆ Garbage collection
    - No memory leaks
  - ◆ Exceptions as a pervasive mechanism to check errors

# Java features (cont' d)

---

- Shares many syntax elements w/ C++
  - ◆ Learning curve is less steep for C/C++ programmers
- Quasi-pure OO language
  - ◆ Only classes and objects (no functions, pointers, and so on)
  - ◆ Basic types deviates from pure OO...
- Easy to use

# Java features (cont' d)

---

- Supports “*programming in the large*”
  - ◆ JavaDoc
  - ◆ Class libraries (Packages)
- Lots of standard utilities included
  - ◆ Concurrency (thread)
  - ◆ Graphics (GUI) (library)
  - ◆ Network programming (library)
    - socket, RMI
    - applet (client side programming)

---

# PROGRAM TYPES AND JARS

# Types of Java programs

---

- **Application**

- ◆ It's a common program, similarly to C executable programs
- ◆ Runs through the Java interpreter (**java**) of the installed Java Virtual Machine

```
public class HelloWorld {  
    public static void main(String args[]) {  
        System.out.println("Hello world!");  
    }  
}
```



# Types of Java programs

---

- **Applet** (client browser)
  - ◆ Java code dynamically downloaded
  - ◆ Execution is limited by “sandbox”
- **Servlet** (web server)
  - ◆ In J2EE (Java 2 Enterprise Edition)
- **Midlet** (mobile devices)
  - ◆ In J2ME (Java 2 Micro Edition)
- **Android App** (Android device)
  - ◆ Java

# Deployment – Jar

---

- Java programs are packaged and deployed in **jar** files.
- Jar files are compressed archives
  - ◆ Like zip files
  - ◆ Contain additional meta-information
- It is possible to directly execute the contents of a jar file from a JVM
  - ◆ JVM can load classes from within a JAR

# Jar command

---

- A jar file can be created using:

```
jar cvf my.jar *.class
```

- The contents can be seen with:

```
jar tf my.jar
```

- To run a class included in a jar:

```
java -cp my.jar First
```

- ♦ The “-cp my.jar” option adds the jar to the JVM classpath

# Jar Main class

---

- When a main class for a jar is defined, it can be executed simply by:

```
java -jar my.jar
```

- To define a main class, a manifest file must be added to the jar with:

```
jar cvfm my.jar manifest.txt
```



```
Main-Class: First
```

---

Issues and a few curiosities about Java

# FAQ

# FAQ

---

- Which is more “powerful”: Java or C?
  - ◆ Performance: C is better
    - though someone says non that much better (JIT)
  - ◆ Ease of use: Java
  - ◆ Error containment: Java
- How can I generate an “.exe” file?
  - ◆ You cannot. Use an installed JVM to execute the program
  - ◆ GCJ is dead: <http://gcc.gnu.org/java/>

# FAQ

---

- I downloaded Java on my PC but I cannot compile Java programs:
  - ◆ Check you downloaded Java SDK (including the compiler) not Java RTE or JRE (just the JVM)
  - ◆ Check the path includes *pathToJava/bin*
- Note: Eclipse uses a different compiler than javac

# FAQ

---

- Java cannot find a class (ClassNotFoundException)
  - ◆ The name of the class must not include the extension .class:
    - Es. java First
  - ◆ Check you are in the right place in your file system
    - java looks for classes starting from the current working directory



# Wrap-up session

---

- Java is a quasi-pure OO language
- Java is interpreted
- Java is robust (no explicit pointers, static/dynamic checks, garbage collection)
- Java provides many utilities (data types, threads, networking, graphics)
- Java can be used for different types of programs
- Coding conventions are not “just aesthetic”