

The Java Environment

Object–Oriented Programming



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

Version 3.3.2
© Maurizio Morisio, Marco Torchiano, 2025








This work is licensed under the Creative Commons Attribution–NonCommercial–NoDerivatives 4.0 International License.

To view a copy of this license, visit

<http://creativecommons.org/licenses/by-nc-nd/4.0/>.

You are free: to copy, distribute, display, and perform the work

Under the following conditions:


-  **Attribution.** You must attribute the work in the manner specified by the author or licensor.
-  **Non–commercial.** You may not use this work for commercial purposes.
-  **No Derivative Works.** You may not alter, transform, or build upon this work.
 - For any reuse or distribution, you must make clear to others the license terms of this work.
 - Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

Learning objectives

- Understand the basic features of Java
 - ◆ What are portability and robustness?
- Understand the concepts of bytecode and interpreter
 - ◆ What is the JVM?
- Learn few coding conventions
 - ◆ How shall I name identifiers?

Java Timeline

- 1991:  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/>
<https://docs.oracle.com/en/java/javase/20/language/java-language-changes.html>

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 (Lang evolution, LTS)
 - ◆ Lambda expressions
 - ◆ Functional paradigm
- 2017: Java 9 released
 - ◆ Modularization (Jigsaw),
 - ◆ `jshc11` (REPL)
- 2018: Java 10, Java 11 (LTS)
 - ◆ Local `var` type inference
- 2019: Java 12, Java 13
 - ◆ Switch expressions (preview)
 - ◆ Text blocks



Start 6-month
release plan

Java timeline (cont' d)

- 2020: Java 14, Java 15
 - ♦ Text blocks
 - 2021: Java 16, **Java 17 (LTS)**
 - ♦ Records
 - ♦ Jpackage
 - 2022/23: Java 18, Java 19, Java 20
 - ♦ Simple web server
 - ♦ Virtual threads
 - Latest releases
 - ♦ **Java 21 (LTS)**: Sept. 23
 - ♦ Java 23: Sept. 24
-

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)

Java features – Classes

- 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 features – Methods

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

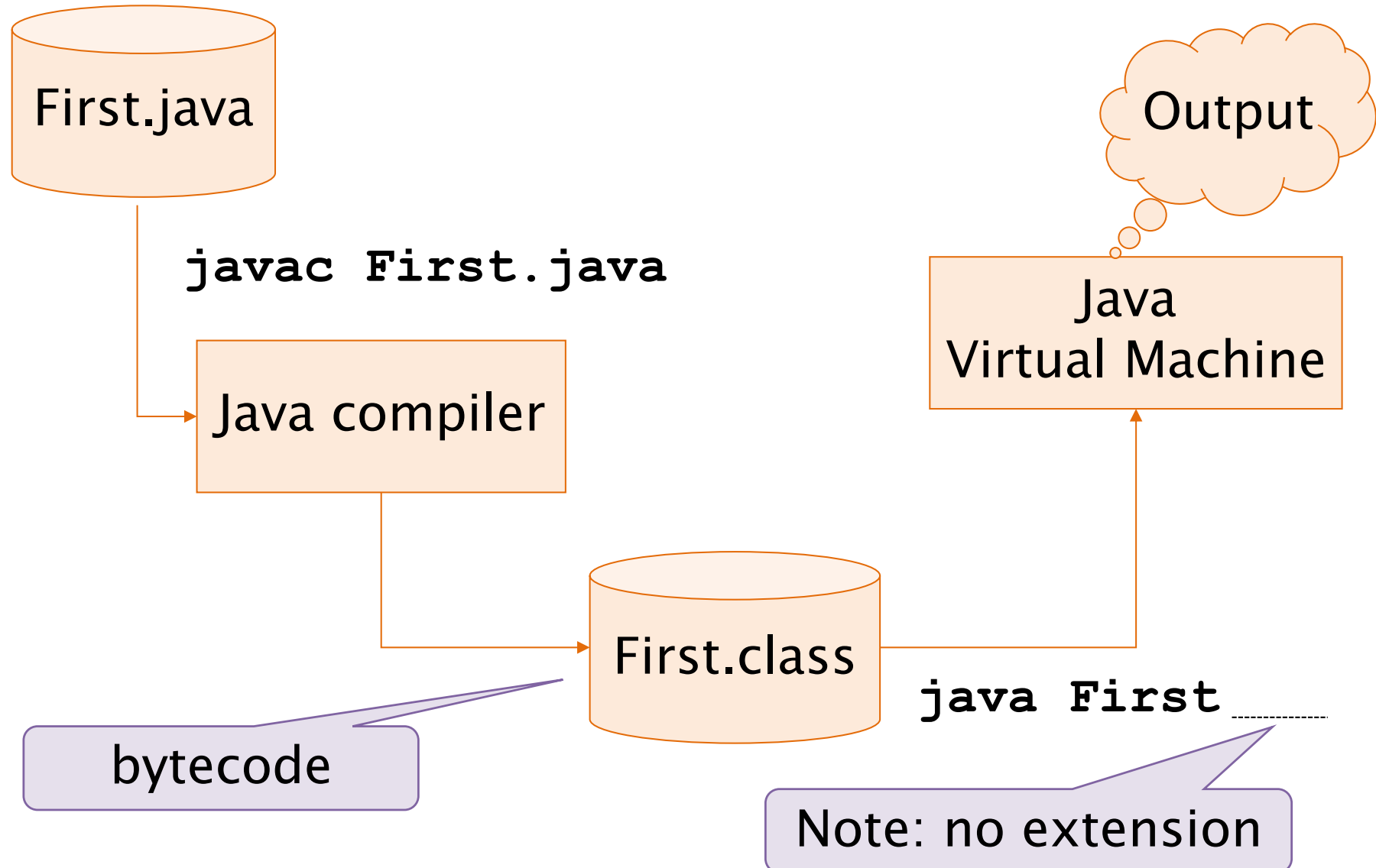
```
public static void main(String[] args)
```

- 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)
-

Build and run



Java Ecosystem

- Java language
 - Java platform Java SE JDK
 - ♦ javac compiler
 - ♦ jdb debugger
 - ♦ JRE (Java Run Time Environment)
 - JVM
 - Native packages (awt, swing, system, etc)
 - Docs
 - ♦ <https://docs.oracle.com/en/java/javase/17>
 - Integrated development environment (IDE)
 - ♦ Visual Studio Code + Java Extension Pack
 - ♦ <https://code.visualstudio.com>
-

Example: source code

File: First.java:

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

Java Compiler

- Command: `javac First.java`
 - ◆ takes a list of source files (e.g. `*.java`)
 - ◆ generates the corresponding `.class`
 - ◆ For each reference to a class
 - looks in the predefined libraries
 - then looks for the source file locally and adds it to the list
 - ◆ After all files have been compiled, the checks for cross-reference integrity.
-

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 JVM:
 - Looks for file X.class
 - If present loads the class
 - Otherwise moves to next location
-

Cafe Babe

- Magic number
 - ◆ Specific initial sequence of a file that let identify the type of the file
 - E.g. PDF files start with chars “%PDF”
- Java class files start with the following 4 bytes:

0xCAFEBAFE

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 ‘.’ eventually)
 - File named *First.class*
 - ◆ Load the class’s bytecode
 - ◆ Perform all due initializations
 - ◆ Look for the `main()` method
 - ◆ Start execution from the `main()` method
-

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

Deprecated since Java 9

- **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

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 uppercases (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 ) {  
                }  
        }  
    }  
}
```

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 run 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
```

FAQ

- Which is more “*powerful*”: Java or C?
 - ◆ Performance: C is better though 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: <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: IDEs often use a different compiler than javac minor differences can be expected
-

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

- 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”