1495 UNIVERSITY OF ABERDEEN

JC2002 Java Programming

Lecture 2: Introduction to Java language

Why Java?

- Java is one of the world’s most widely used computer programming languages.
- For many organizations, the preferred language for meeting their enterprise programming needs is Java.
- According to Oracle’s 2016 JavaOne conference keynote presentation (http://bit.ly/JavaOne2016Keynote), there are now 10 million Java developers worldwide and Java runs on 15 billion devices, including two billion vehicles and 350 million medical devices.
- Android is a flavor of Java.

Java history

- Java is an old language
  - Sun Microsystems in 1991 funded an internal corporate research project led by James Gosling, which resulted in a C++-based object-oriented programming language that Sun called Java.
- The Internet helped Java grow
  - Java drew the attention of the business community because of the phenomenal interest in the Internet.
  - Java programs run on a great variety of computer systems and computer-controlled devices (***"write once, run anywhere"*** – details will be introduced later).
  - Now used to develop large-scale enterprise applications, to enhance the functionality of web servers, to provide applications for consumer devices, to develop robotics software and for many other purposes

Java versions

- There are different JAVA versions in use; the latest release of **Standard Edition** (SE) ***Java Development Kit*** (JDK) is 23.
    - The latest version with long term support (LTS) is JDK 21.
    - Not all versions are equal; this course is mostly based on Java 11 with long term support (LTS)
    - For the basic concepts in this course, any code from Java 8 (or later) will look almost identical.
- Older Java RE runtimes may not run code written for newer versions and newer runtimes may be missing libraries required by programs written for older programs!

Java criticism

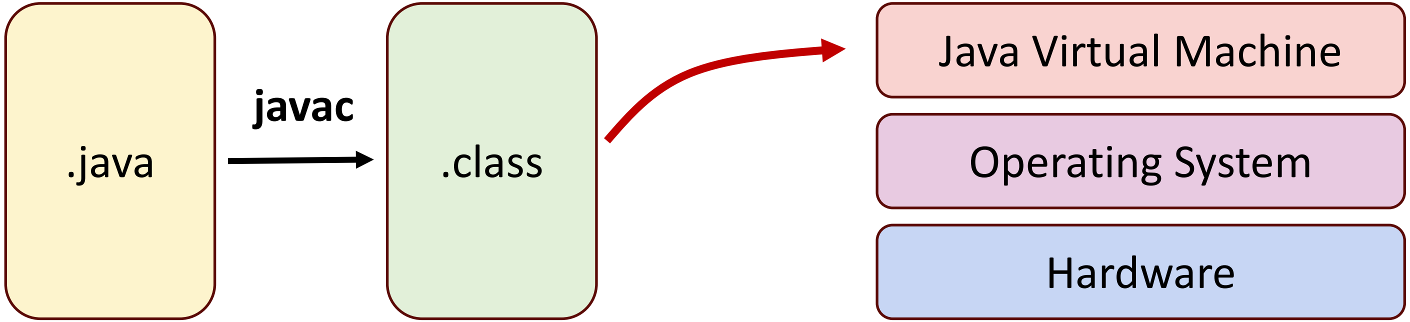
- Too verbose
  - However, easier to read!
  - The extra verbosity can be a benefit when you are responding to an outage call, or when you need to maintain and patch code that was written by developers who have long since moved on
- Slow to change
  - The new language features that have arrived in recent versions are a significant step toward addressing the most common complaints about missing features
- Low performance
  - True for the very early releases, but no longer a constraint

Java criticism (continues)

- Too verbose
- Slow to change
- Low performance
- Security concerns
  - During 2013, there were a lot of security vulnerabilities in the Java platform, which caused the release date of Java 8 to be pushed back
  - Many of these vulnerabilities involved the desktop and GUI components of the Java system, and do not affect websites or other server-side code written in Java
- Too corporate
  - Today, this is the opposite: Java is a widely used language for open-source software projects

Java Virtual Machine (JVM)

- The JVM is a program that provides the “**runtime environment” (or execution environment)** necessary for Java programs to execute



- To compile .java file to .class file, use `javac Program.java`
- To run .class file, use `java <arguments> Program`

Benefits of JVM

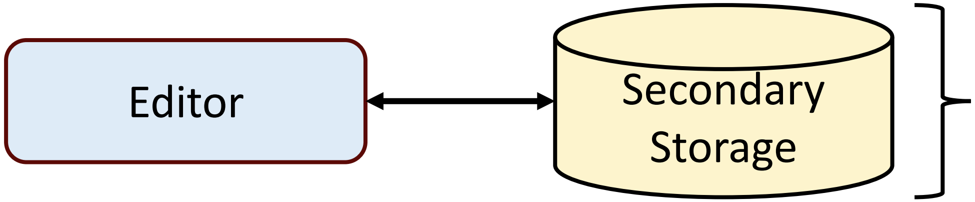
- Comprise a container for application code to run inside.
- Provide a secure and reliable execution environment (as compared to C/C++).
- Take memory management out of the hands of developers.
- Provide a cross-platform execution environment, i.e., “write once, run anywhere” (WORA).
- Make use of runtime information to self-manage, i.e., just-in-time (JIT) compilation.

Typical Java development environment

- Normally, there are five phases in Java program development:
    - Phase 1: **Edit** the code
    - Phase 2: **Compile** to *bytecode*
    - Phase 3: **Load** the bytecode
    - Phase 4: **Verify** the bytecode
    - Phase 5: **Execute** the bytecode

Phase 1: creating a program

- Phase 1 consists of editing a file with an *editor program* (editor)
- Using the editor, you:
  - Type a Java program (source code)
  - Make any necessary corrections
  - Save it on a secondary storage device



Program is created in an editor and stored in a file with name ending `.java`

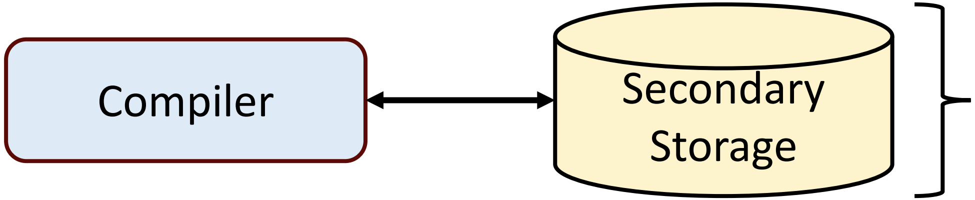
Phase 1: Editing the program file

- You can edit source code file with any text editor (Vim, Notepad, TextEdit etc.)
- You can also use an ***Integrated Development Environment*** (IDE)
  - Provides tools to support software development process, such as editor, debuggers for locating logic errors, etc.
  - The most popular Java IDEs include:
    - Eclipse (http://www.eclipse.org)
    - IntelliJ IDEA (http://www.jetbrains.com)
    - NetBeans (http://www.netbeans.org)

Phase 2: Compile a Java program

- Use the command javac (the Java compiler) to compile source code to a program
- To compile source file Welcome.java, you would type:

```
javac Welcome.java
```



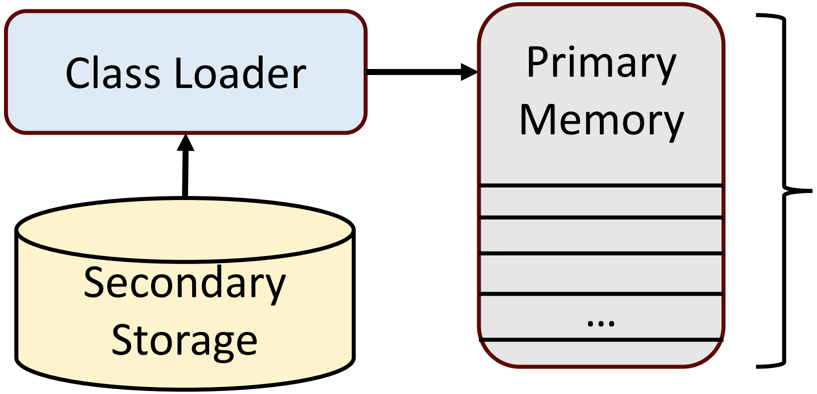
Compiler creates bytecodes and stores them in a file ending in `.class`

Phase 2: Compiled bytecodes

- Java compiler translates Java source code into bytecodes that represent the tasks to execute in the execution phase:
  - VMs can hide the underlying operating system and hardware from the programs that interact with it: If the same VM is implemented on many computer platforms, applications written for that type of VM can be used on all those platforms.
  - The JVM —a part of the JDK and the foundation of the Java platform—executes bytecode.
- So, Java’s bytecodes are **portable**, the same bytecode instructions can execute on any platform containing a JVM that understands the version of Java in which the bytecodes were compiled.

Phase 3: Load program into memory

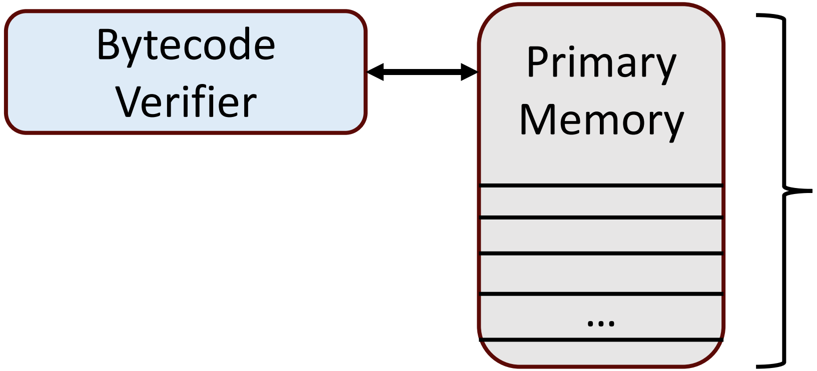
- The JVM places the program in memory to execute it — this is known as loading.
- The class loader takes the `.class` files containing the program's bytecodes and transfers them to primary memory. It also loads any of the `.class` files provided by Java that your program uses.



Class loader reads bytecodes from `.class` files and puts bytecodes in memory

Phase 4: Verify the bytecode

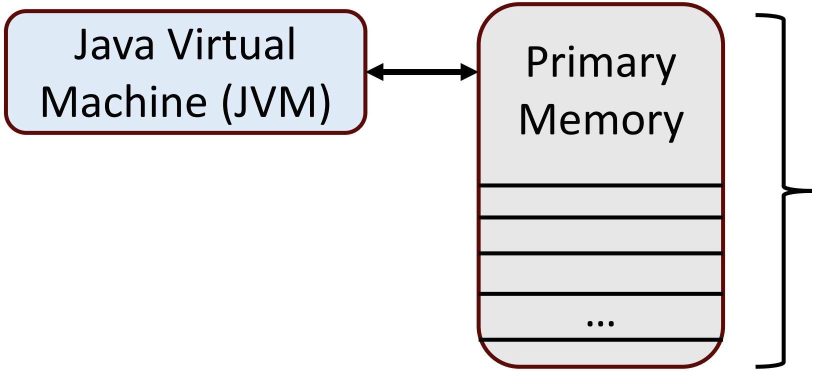
- As the classes are loaded, the bytecode verifier examines their bytecodes to ensure that they are valid and **do not violate Java's security restrictions**:
  - Java enforces strong security to make sure that Java programs do not damage your files or your system (as, for example, computer viruses).



Bytecode verifier confirms that all bytecodes are valid and do not violate security restrictions

Phase 5: Execution

- The JVM executes the program’s bytecodes
  - Today’s JVMs typically execute bytecodes using a combination of interpretation and so-called *just-in-time* (JIT) compilation.
  - With JIT, the JVM can analyze the bytecodes as they are interpreted, searching for hot spots – bytecodes that execute frequently.



To execute the program, the JVM reads bytecodes, and JIT compiles them into a language the computer understands

Phase 5: Execution with just-in-time (JIT)

- A JIT compiler—such as Oracle's Java HotSpot™ compiler—translates the bytecodes into the computer's machine language
    - When the JVM encounters these compiled parts again, the faster machine-language code executes.
- With JIT, Java programs go through two compilation phases
    - The first, in which source code is translated into bytecodes (for portability across JVMs on different computer platforms).
    - The second, in which, during execution, the bytecodes are translated into machine language for the actual computer on which the program executes.

Common errors

- When using `javac`, an error message such as *“Bad command or filename”* or *“javac: command not found”* means that your Java software installation was not properly completed
  - Often, PATH environment variable was not set properly; carefully review the installation instructions if this happens
  - On some systems you need to reboot your computer after correcting PATH to make the change to take effect
- When using `java` to run a .class file, an error message such as *“java.lang.NoClassDefFoundError”* often means that Java CLASSPATH environment variable is not properly set

Java Classpath

- Java interpreter needs to know where to look for classes (`.jar` or `.class` files) that are not part of core Java
  - Classpath defines where to look for external bytecode files
- Two options to set Classpath:
  - Define CLASSPATH environment variable:

```
export CLASSPATH=.:/path/to/external/library.jar
```

(LINUX)

```
set CLASSPATH=.;/path/to/external/library.jar
```

(Windows)

  - Use command line switch `–cp` or `–classpath` with `java` command:

```
java -classpath .:/path/to/external/library.jar ProgramName arg
java -cp .:/path/to/external/library.jar ProgramName arg
```

Summary

- Java is an old language widely used on many platforms:
  - Java is a common language in internet programming and embedded systems.
  - Java is the primary language used for writing Android apps.
- Java development cycle involves five phases:
  - Writing and editing the code, compiling to bytecode, loading the bytecode, verifying the bytecode, and executing the program.
  - The last three phases are done automatically by JVM when a Java program is started with command `java`.
- Java bytecode is portable without re-compilation.

Questions, comments?

1495 UNIVERSITY OF ABERDEEN