

JAVA

Lecture 001 – Introduction to Java

MODULE INFORMATION

- Lecturer: Yuan Yao
- Office: PMB438
- Office Hours: Friday at 16:00 (two hours)
- Email: Yuan.Yao@nottingham.edu.cn

LECTURES

- Thursday 17:00 – 18:00
- Friday 9:00 – 11:00
- ***Java Programming: A Comprehensive Introduction***, Herbert Schildt and Dale Skrien, 2013.
-
- Exam is mostly knowledge based
- Labs and Coursework is mostly comprehension and application
- Possibly learn more in the labs than the lectures (I would)

ROUGHLY 6 TOPICS

- 1. Fundamentals
- 2. Classes, Objects, Methods
- 3. Inheritance
- 4. Interfaces
- 5. Exception Handling and IO
- 6. Packages
- Chapters I-II of “Java Programming: A Comprehensive Introduction, Herbert Schildt and Dale Skrien, 2013.”

EXPERIENCE WITH JAVA AND OOP

JAVA ON INTERNET

- What's in this module:
 - A very basic introduction to Java and OOP
 - We assume you do not have any Java experience.
 - All the materials covered in this module are available on the internet
 - You may find online tutorials useful
 - But please try yourself

WHAT IS JAVA



- Java is a **high-level** programming language conceived by James Gosling in 1991. (Originally called Oak and then Green)
- It was named “Java” in 1995 (apparently after the coffee they drunk)
- Its syntax is similar to the older languages C and especially C++
- It is an important language for the Internet (Servlets, Apps...more recently Machine learning, simulation)
- Best known for:
 - **Cross-platform**
 - **Safety**

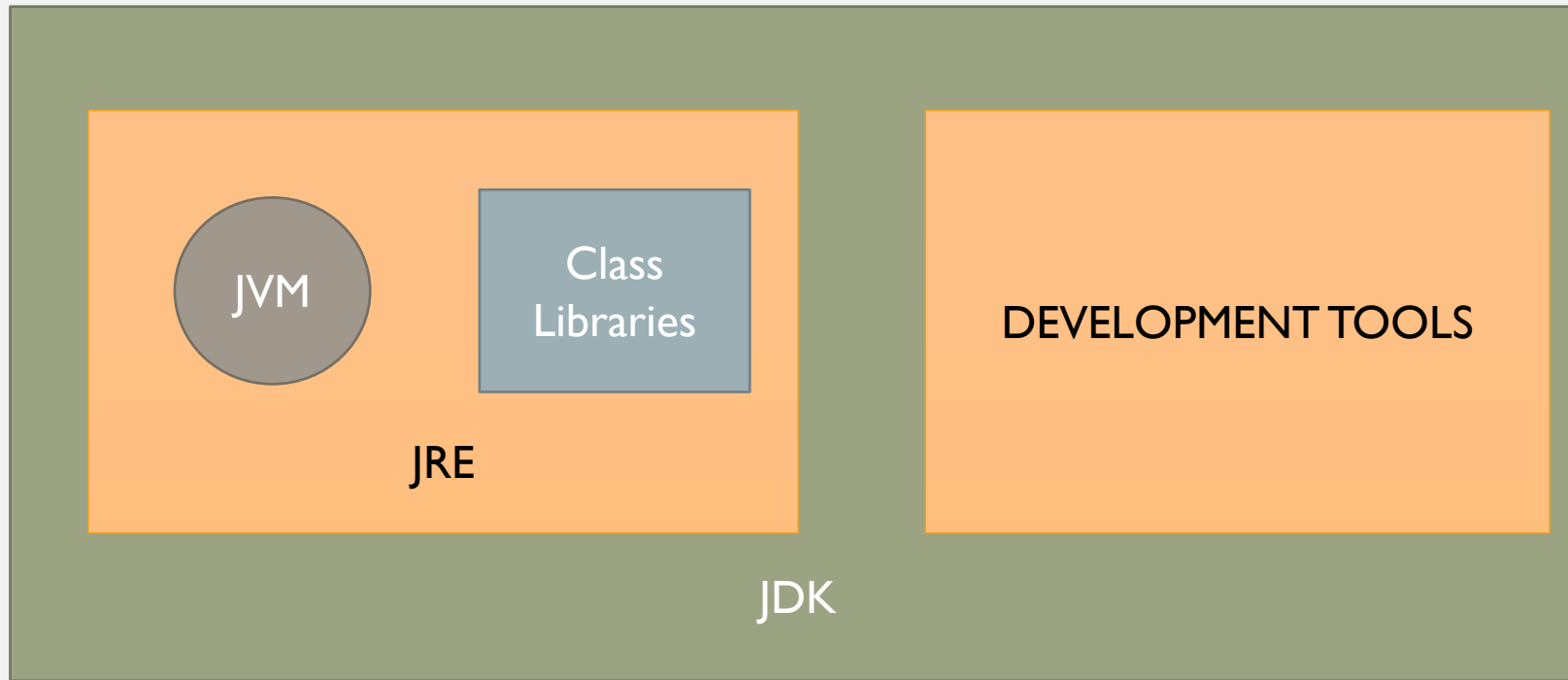
WHY JAVA

- Java is popular and in demand-ish
- Good for jobs, and almost 'expected' of CS grads
- Also used in many later modules
- And can help you to understand more complex languages like C++
- Java 'forces' some kind of object orientedness (Unlike C++ and Python)
- Everything is defined in a class in Java
- Java has a lot of classes available (**java.util.HashMap**, **java.lang.Exception....**)
- Cross-platform – no recompilation for platforms

BEFORE START PROGRAMMING

- Java Development Kit (JDK): a software development environment used for developing Java applications and applets.
 - JRE
 - Development Tools like compilers, debuggers
- Java Runtime Environment (JRE): minimum requirements for executing a Java application
 - Java Virtual Machine (JVM).
 - Java Class Library.
 - ...

JDK, JRE AND JVM



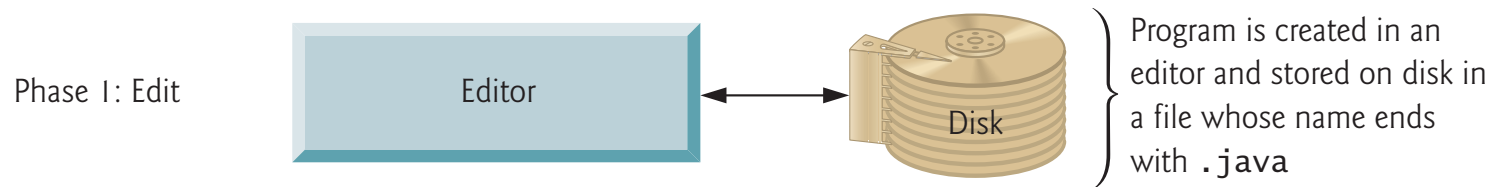
JVM

- Compiler: a special program that translates high-level programming languages' source code into machine code (for specific operating systems and computer architecture)
- Java: source code is translated into bytecode (intermediate language).
- Java Virtual Machine: a program to execute Java bytecode.

JAVA

- Creating and executing a Java application using a Java development environment normally include 5 steps:
 - Step 1: Creating a program
 - Step 2: Compiling a Java program into Bytecodes
 - Step 3: Loading the bytecode into Memory
 - Step 4: Bytecode Verification
 - Step 5: Execution

STEP I: CREATING A PROGRAM



- Write Java program:
 - Plain text editor: Notepad++, VS Code, Sublime, Vim, Emacs...
 - Integrated Development Environments (IDEs): Eclipse, IntelliJ, Netbeans... (for complex development)
- The program is created and stored on a storage device in a file whose name ends with `.java`
 - Each file contains one or more class definitions.
 - Each file contains at most one public class. (For now, each file contains only one class)
 - Filename must match the class name.

FIRST PROGRAM

```
/*
```

This is a simple Java program.

Call this file HelloWorld.java.

```
*/
```

```
public class HelloWorld {
```

```
    // A Java program begins with a call to main().
```

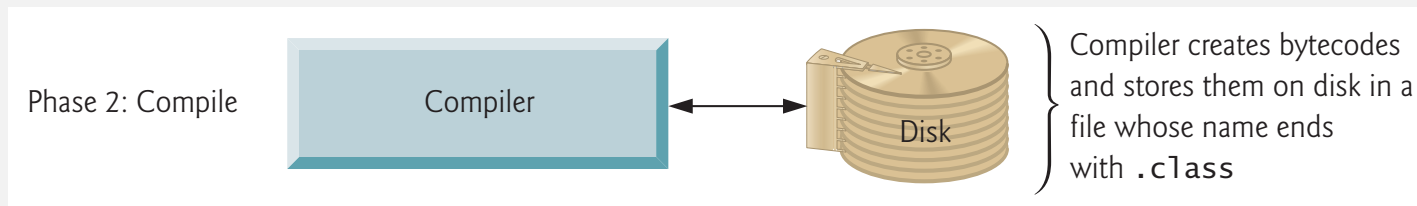
```
    public static void main(String[] args) {
```

```
        System.out.println("Hello World");
```

```
    }
```

```
}
```

STEP2: COMPILING

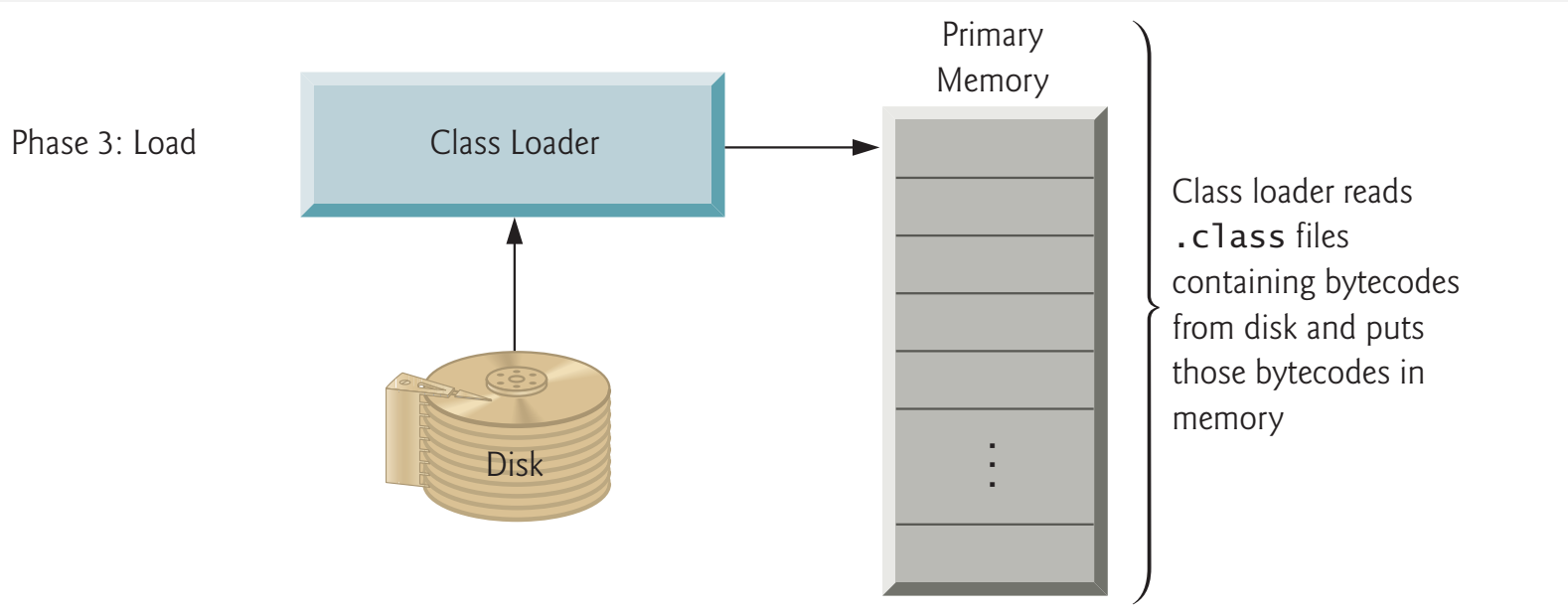


- To compile the program, on the command line, type in `javac` (Java compiler):
 - `javac HelloWorld.java`
- The java compiler translate Java source code into bytecodes and produce a .class file (e.g., HelloWorld.class)
- Java compilers do not generate machine code for a CPU but for a Java Virtual Machine (JVM)
- JVM: a part of the JDK and the foundation of the Java platform.
- Bytecodes are executed by a JVM on each computer.

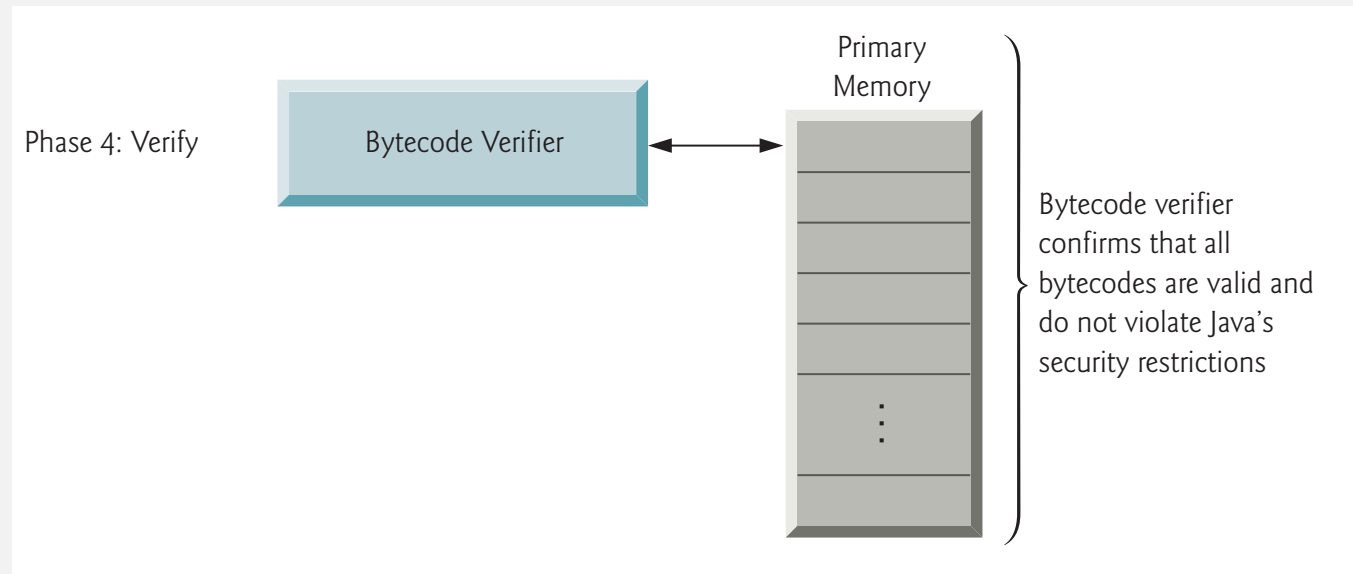
JVM

- JVM: A software application that simulates a computer but hides the underlying operating system and hardware from the program that interact with it.
 - JVM is a much simpler machine
 - Machine languages: dependent on specific computer hardware.
 - Bytecodes: platform independent.
- Portable: the same bytecodes can execute on any platform containing JVM without recompiling.
- more convenient to program than real CPUs
- JVM is invoked by the *java* command, i.e., to execute a java application
 - *java HelloWorld*

STEP3: LOADING A PROGRAM INTO MEMORY

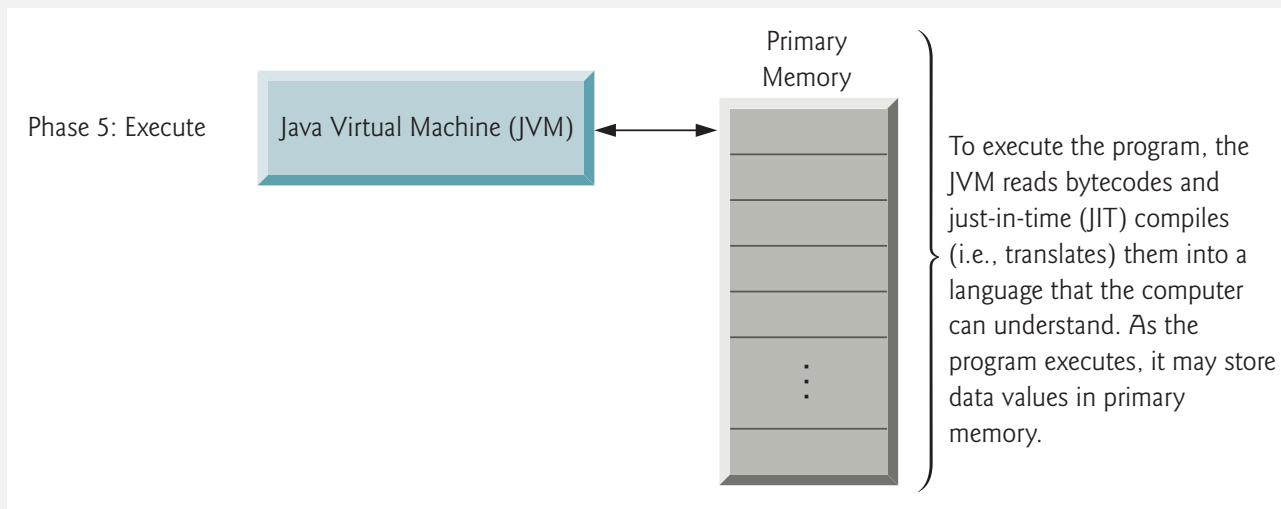


STEP4: BYTECODE VERIFICATION



- Java enforces strong security to make sure that Java programs arriving over the network do not damage your files or your systems (e.g., viruses and worms)

STEP5: EXECUTION



- JVM **executes** the program's bytecodes, thus performing the actions specified by the program.
 - Early versions: JVM was simply an interpreter for bytecodes. slow, interpret and execute one bytecode at a time.
 - Today: using a combination of JVM and so-called just-in-time (JIT) compilation.
 - JVM searches for hot spots, JIT translate such bytecode into machine languages.

FIRST PROGRAM

```
/*  
This is a simple Java program.  
Call this file HelloWorld.java.  
*/  
public class HelloWorld {  
    // A Java program begins with a call to main().  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

COMMENTS

```
/*
```

```
This is a simple Java program.
```

```
Call this file HelloWorld.java.
```

```
*/
```

Multiline Comments



```
public class HelloWorld {
```

```
    // A Java program begins with a call to main()
```

Single-line Comments



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

```
        System.out.println("Hello World");
```

```
    }
```

```
}
```

JAVA CLASS

- `public class HelloWorld {`
 - `.....`
- `}`
- **Declare a Class:**
 - class: a **keyword (reserved word)** for class declaration.
 - Class name: an identifier.
 - By convention, begin with a capital letter and capitalise the first letter of each word.
- **Brace {}:**
 - Braces limit the scope of variables and allow statements to be grouped
 - The elements between the braces are members of the class

MAIN IN JAVA

- `public static void main (String[] args){`
 - `.....`
- `}`
- Parentheses: indicating it is a programming building block called method.
- Methods are defined in a class.
- When the program starts up we need a starting method.
- Main is the starting point of every Java application.
- Each Java application must have a main method.

MAIN IN JAVA

- `public static void main (String[] args)`
- **public** – Access modifier. It means that you can call this method from outside of the current class. The main method must be declared as public.
- **static** – Keyword. When the JVM makes a call to the main method there is no object existing for the class being called. Static allows `main()` to be executed independently of any object.
- **void** - No particular type of data has to be returned through the function.
- **main** – The name of the method.
- **(String[] args)** – parameters. main method accepts an array of String as its input when you run it.

PRINTLN

- `System.out.println("Hello World");`
- **println()**: built-in method that display the given string.
- **System**: a pre-defined class which provides access to the system
- **out**: is the output stream that is connected to the console.
- **Semicolon**: a separator to terminate a statement. All statements end with a semicolon.

SYNTAX ERRORS

- **Errors in Java:**
 - Syntax errors: errors in the syntax of a programming language
 - Java compiler will report any syntax errors at compile time.
 - Runtime errors: errors that occur while the program is running.
 - We will learn it later.
- Spelling mistakes
- Using keywords as identifiers
- Incorrect use of characters in identifiers
- Case differences
- Missing {
-