RCS 122
OBJECT ORIENTED PROGRAMMING I

Lecture 03: Introduction to Java

A Brief Background of Java

- It was built upon the rich legacy inherited from C and C++.
- Invented by James Gosling, Patrick Naughton, Cris Warth, Ed Frank and Mike Sheridon at Sun Microsystems in 1991.
- Was originally called Oak, renamed Java in 1995.
- Older computer programming languages like C++ were designed to be compiled for a specific CPU.
- Due to its unique design, Java programs are more portable than programs written using other higher level programming languages like C++.

Java’s main Philosophy

- Simple
- Familiar to Developers
- Object Oriented
- Portable
- Interpreted
- Multi-threaded
- Robust and Secure

Java: A compiled and interpreted language

- Source code is a program file made up of programming statements written in high level language using a text editor.
- Many high level programming languages' source code files are compiled into object code files compatible with a specific CPU's machine language. This is not the case with Java.
- Java compiler (javac) produces an output called bytecode designed to be executed by Java interpreter which is within the Java Virtual Machine (JVM).
- Bytecode is a binary representation into which a program source code is converted by Java compiler.

Java: A compiled and interpreted language

- Translating Java programs into bytecode makes it easier to run programs in a wide range of environments because only appropriate JVM needs to be implemented for each platform.
- Once appropriate JVM for a given system (CPU design and therefore machine language) has been implemented, any Java program can run on it.
- Although details of JVM differ from one CPU platform to another, all JVMs understand the same bytecode.
- If Java programs were compiled to computer's native code, then different versions of the same program would have to exit for each CPU and then fail to be cross-platform.

Java: A compiled and interpreted language

- Because Java programs are isolated from the OS, they are also insulated from the particular hardware on which they run.
- This isolation helps to protect Java programs against malicious programs that access the computer hardware through the OS.
- Java is modeled after C++ language, and therefore there is close similarity in the syntax and vocabulary between the two languages.
- Despite this similarity, Java eliminates most of the difficult-to-understand features inherent to C++.

A compiled language: e.g. C++

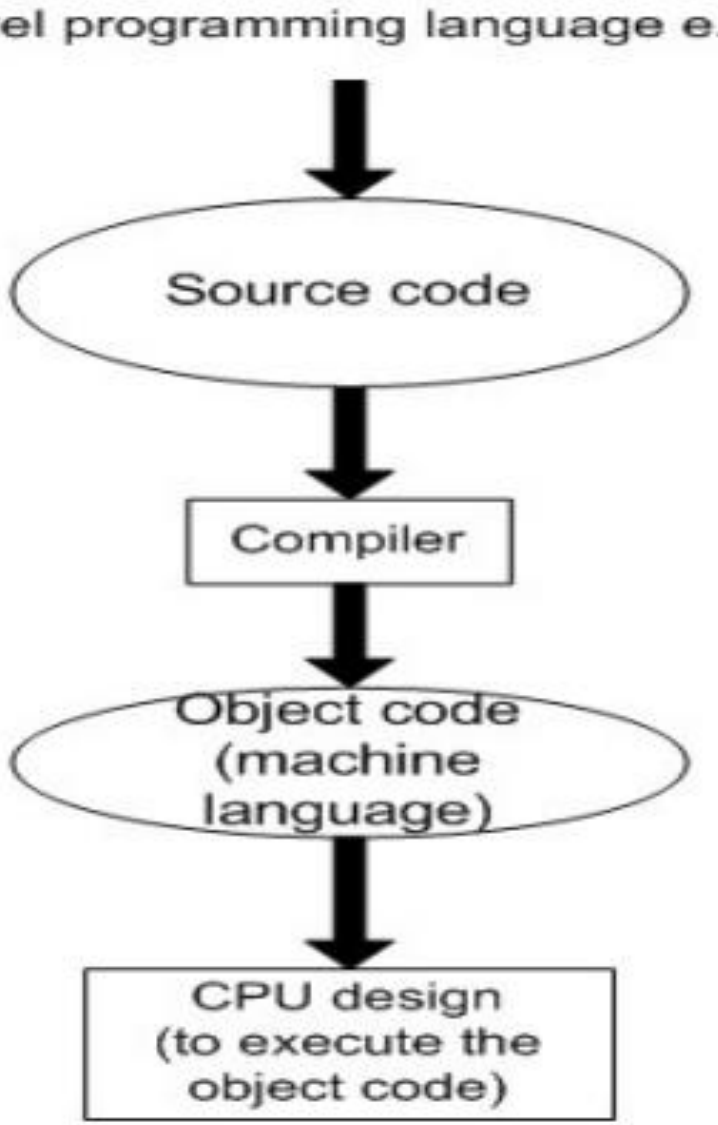


Figure: A compiled High level language like C++

An interpreted language: e.g. Java

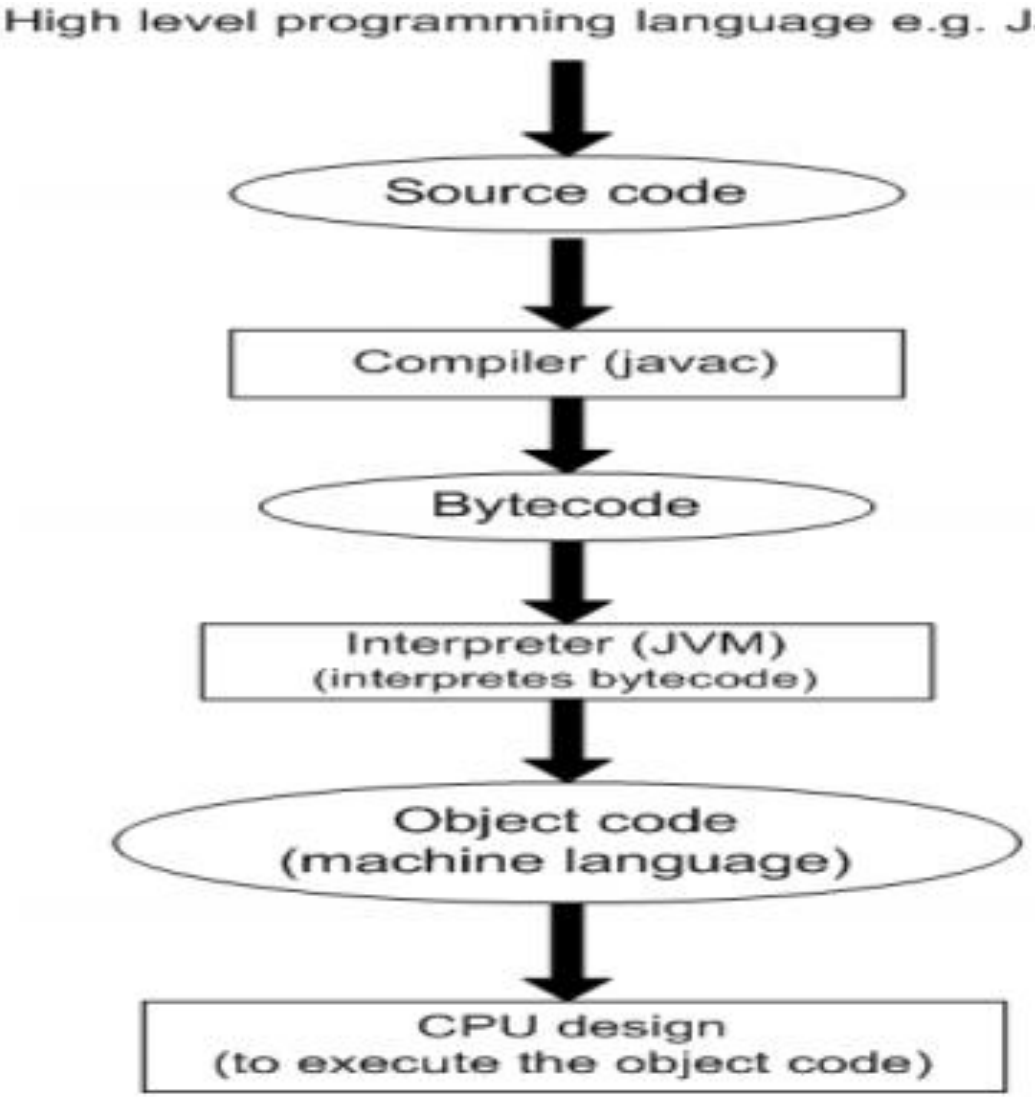


Figure: An interpreted language, Java

Requirements for writing Java applications

- A working computer installed with OS like Windows, Linux or Mac OS.
- Well installed and configured Java Software Development Kit (SDK) such as Java Development Kit (JDK).
- Text editor such as TextPad, Notepad etc.
- Integrated Development Environment (IDE) such as Eclipse, NetBeans, JBuilder etc.
- Knowledge and skill on Java programming.

Program errors

- Errors contained in the program source code are bugs.
- A process of tracking (identifying) bugs and correcting them is called *Debugging*.
- Different errors in a program have different outcomes.
- Distinguishing them properly helps programmers to avoid or track them more quickly.
- Most common types of errors are:
  - Compile-time errors
  - Run-time errors
  - Logical errors

Compile-time errors

- Java compiler can only translate a program that is syntactically correct.
- When compilation fails, the program can not run.
- Syntax: Refers to the structure of the program and the rules about the structure.
- Compilers of many programming language, including Java, are not forgiving because once they discover any syntax error anywhere in the program, they print them and quit.
- Syntax errors are common to new programmers.

Run-time errors

- They are so called because they do not appear until you run the program.
- In Java, they occur when interpreter is running the bytecode.
- In Java, run-time errors are called *exceptions*.
- Java is a safe language; meaning that it has few such run-time errors.

Run-time errors

Examples of run-time errors are:

- Dividing a number by zero.
- Performing an illegal operation that violates security.
- Attempt to use an index that is outside the bounds of the object it is applied to.

Logical errors

- Also called semantics errors.
- The program runs and compiles successfully.
- Program does not generate any errors, but will not produce the desired output.
- The program you wrote is not what you wanted to write.
- Identifying logical errors is tricky and requires a lot of efforts.

Logical errors

Examples:

- Writing a wrong arithmetic operator such as + instead of -.
- Placing a condition wrongly, a loop that iterates more or less than required number.
- Using data values such as int which after certain arithmetic operations the output exceeds the valid range of int.

First Java Program

```
// The first Java program
// The famous Hello World!
public class HelloWorld {
        public static void main(String[] args){
          System.out.println("Hello world");
        }
}
```

First Program Explained

- The first two lines are comments (to be explained later)
- The 3rd line start with: **`public class … {`**
  - All java programs must have at least one statement of this kind. In java, it is about creating and using classes
  - The keyword **`public`** means this class is accessible everywhere
  - The keyword **`class`** means what follows enclosed by **`{…}`** is a class definition.

First Program Explained

The 4th line: **public static void main(String[] args){**

- This is a method header. One main method must exist in every java program, if it doesn't exist the program will compile but won't run. The main method is the entry/starting point for running
- **public**: the method is accessible everywhere
- **static**: the method is static, i.e it is not instantiable (more on this in later chapters)
- **void**: the method doesn't return anything (more on this in later)
- **main**: name of method, it is a special method. Should be declared as above
- Anything enclosed with (…) after method name is a parameter list. So for main method, args is a parameter of type array of string. (more on this in later)
- The statements enclosed by {…} after method header are the method implementation. i.e the statement: **System.out.println("Hello world ");** is the only statement of method main

Comments

- Use Comments to make your programs readable
- They can appear anywhere in a program
- They are ignored by compiler
- Type 1: // up to the end of the line is a comment
- Type 2: /* all character enclosed between these are comments -hence ignored*/
- But don’t overdo it: x = y; // assigning y to x (This is obvious!)

Terminating statements

- Each statement must ends with a semicolon '**;**'
- The statements:

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

- Have exactly the same effect as the statements

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

- But the later is considered to be poor style and is not recommended

Concatenation

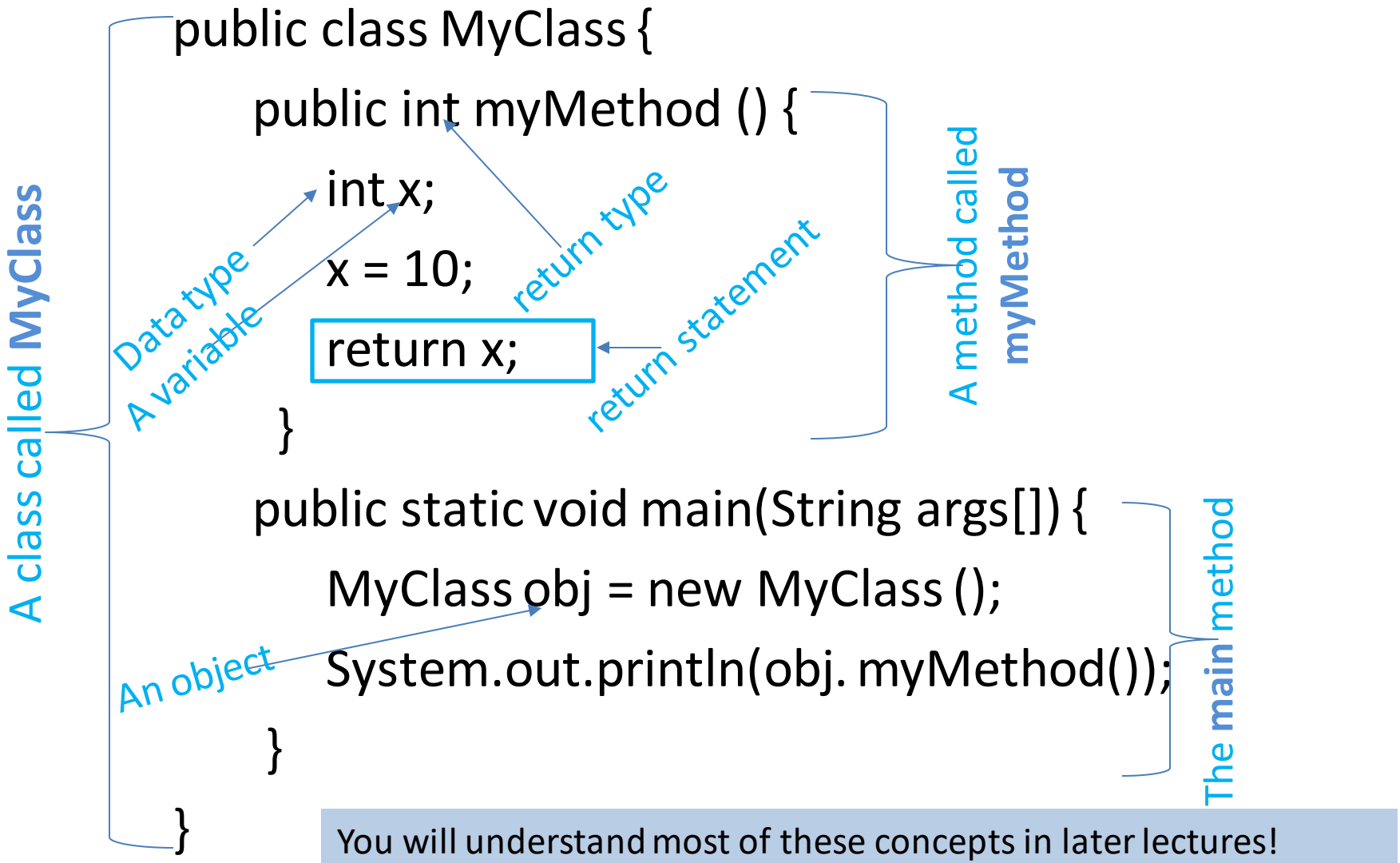
- In Java, the + operator is used to concatenate strings.
- For example,

"Hello," + " world" + "!"

is equivalent to:

"Hello, world!"

Another Java Program



You will understand most of these concepts in later lectures!

End