Java

Introduction

What is Java

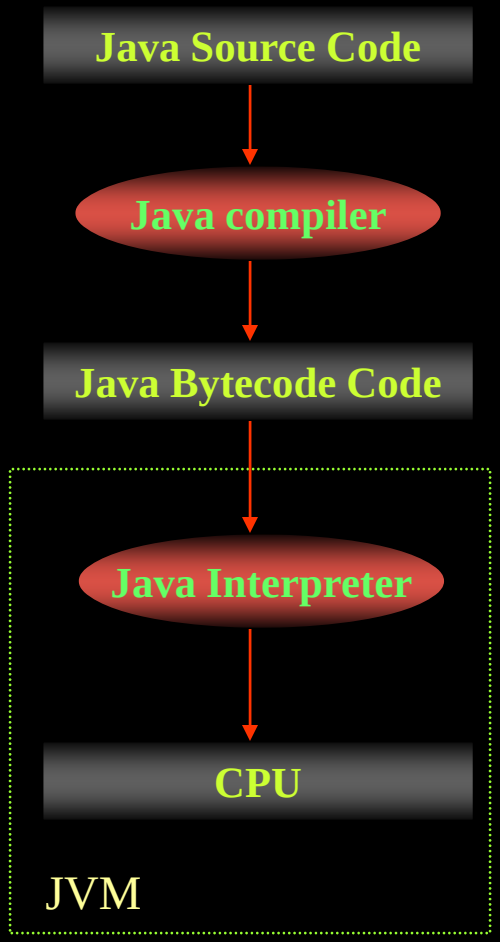
- A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language

  -- Sun

- ***Object-Oriented*** :
  - No free functions.
  - All code belong to some class.
  - Classes are in turn arranged in a hierarchy or package structure.

What is Java

- ***Distributed***
  - Fully supports IPv4, with structures to support IPv6.
  - Includes support for Applets: small programs embedded in HTML documents.
- ***Interpreted***
  - The program are compiled into Java Virtual Machine (JVM) code called bytecode
  - Each bytecode instruction is translated into machine code at the time of execution. (Penalty : Speed)



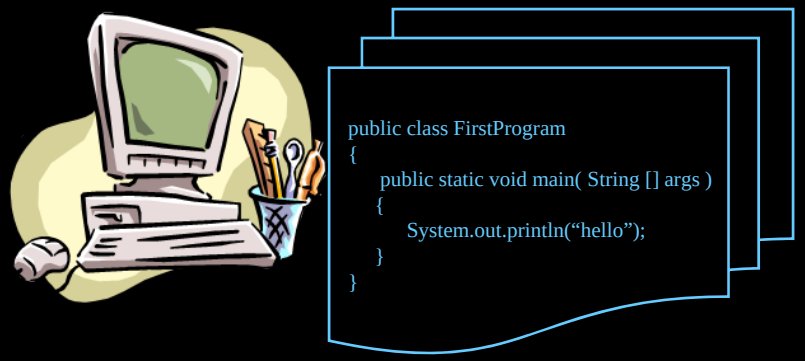
What is Java

- ***Robust***
  - Java is simple-no ***pointers/stack*** concerns.
  - In-bound checking at runtime of array pointers-no memory corruption and cryptic error messages.
  - Exception handling: try/catch/finally series allows for simplified error recovery
  - Strongly typed language: many errors caught during compilation.

Java Editions

- Java has 3 editions.
- ***Java 2 Platform, Standard Edition (J2SE)***
  - Used for developing Desktop-based application and networking applications.
- ***Java 2 Platform, Enterprise Edition (J2EE)***
  - Used for developing large-scale, distributed networking applications and Web-based applications.
- ***Java 2 Platform, Micro Edition (J2ME)***
  - Used for developing applications for small memory-constrained devices, such as cell phones, pagers and PDAs.

Java platform



HelloWorld.java

Compile
javac

HelloWorld.class

2387D47803
A96C16A484
54B646F541
06515EE464

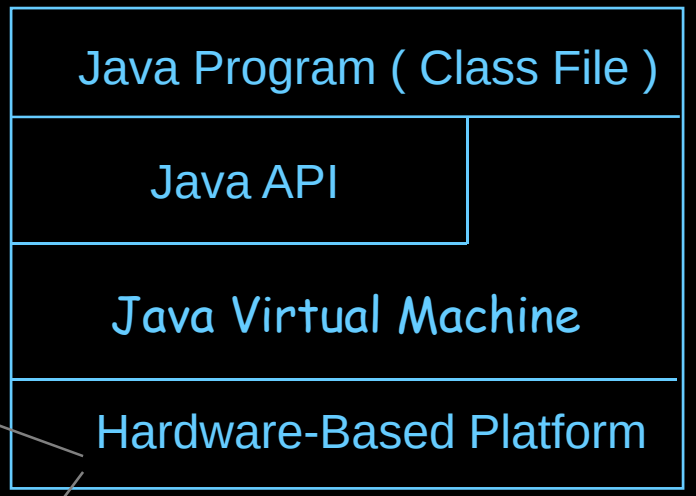
Java
Bytecode

Java
Interpreter

Windows NT

Java
Interpreter

Power Macintosh



Java Development Environment

- Java programs normally go through *5* phases.
  - ***Edit***
    - Create/edit the source code
  - ***Compile***
    - Compile the source code
  - ***Load***
    - Load the compiled code
  - ***Verify***
    - Check against security restrictions
  - ***Execute***
    - Execute the compiled

Phase 1: Creating a Program

- Any text editor or Java IDE (Integrated Development Environment) can be used to develop Java programs.
- Java source-code file names must end with the *.java* extension.
- Some popular Java IDEs are
  - NetBeans
  - Eclipse
  - Jbuilder
  - Kawa
  - jEdit
  - JCreator etc.

Phase 2: Compiling a Java Program

- ***javac Welcome.java***
- Compiles the source file ***Welcome.java*** (and other files if necessary), transforms the Java source code into bytecodes and places the bytecodes in a file named ***Welcome.class***.
- It searches the file ***Welcome.java*** in the current directory.

Bytecodes

- They are not machine language binary code.
- They are independent of any particular microprocessor or hardware platform.
- They are platform-independent instructions.
- Another entity or interpreter is required to convert the bytecodes into machine codes that the underlying microprocessor understands.
- This is the job of the ***JVM (Java Virtual Machine).***

JVM (Java Virtual Machine)

- It is a part of the JDK and the foundation of the Java platform.
- It can be installed separately or with JDK.
- A virtual machine (VM) is a software application that simulates a computer, but hides the underlying operating system and hardware from the programs that interact with the VM.
- One of the main contributors for the slowness of Java programs compared to compiled machine language programs (i.e. C, C++, Pascal etc.).
- ***It is the JVM that makes Java a portable language.***

JVM (Java Virtual Machine)

- The same bytecodes can be executed on any platform containing a compatible JVM.
- JVM is available for Windows, Unix, Linux and Solaris.
- The JVM is invoked by the ***java*** command.

  java Welcome
- It searches the class ***Welcome*** in the current directory and in the directories listed in the ***CLASSPATH*** environment variable and executes the ***main*** method of class ***Welcome***.
- It issues an error if it cannot find the class ***Welcome*** or if class ***Welcome*** does not contain a method called ***main*** with proper signature.

Phase 3: Loading a Program

- One of the components of the JVM is the class loader.
- The class loader takes the .class files containing the programs bytecodes and transfers them to primary memory (RAM).
- The class loader also loads any of the .class files provided by Java that our program uses.
- The .class files can be loaded from a disk on our system or over a network (another PC in our LAN, or the Internet).

Phase 4: Bytecode Verification

- Another component of the JVM is the bytecode verifier.
- Its job is to ensure that bytecodes are valid and do not violate Java's security restrictions.
- This feature helps to prevent Java programs arriving over the network from damaging our system.
- Another contributor for making Java programs slow.

Phase 5: Execution

- Now the actual execution of the program begins.
- Bytecodes are converted to machine language suitable for the underlying OS and hardware.
- So, we can say that Java programs actually go through two compilation phases –
  - Source code -> bytecodes
  - Bytecodes -> machine language

Editing a Java Program

```
/*
    File Name: JavaTest.java
*/
public class JavaTest
{
    public static void main(String args[ ])
    {
        System.out.println("Hello Java");
        System.out.printf("I like %s\n", "Java");
        String strDepartment = "CSE";
        System.out.print("We study in " + strDepartment + "\n");
    } // end method main
} // end class JavaTest - NOTE: no semicolon is required here
```

Examining JavaTest.java

- Every program in Java consists of ***at least one class*** declaration defined by the programmer.
- A Java source file can contain multiple classes, but only one class can be a ***public class***.
- Typically Java classes are grouped into ***packages*** (similar to ***namespaces*** in C++).
- A public class is accessible across packages.
- The source file name must match the name of the **public class** defined in the file with the ***.java*** extension.

Examining JavaTest.java

- We can also place our classes into packages, but it is optional.
- When we do not specify any package for our class, the class is placed in a default unnamed package (more on packages will be discussed later).
- By convention, all class names in Java begin with a capital letter and capitalize the first letter of each word they include (e.g. SampleClassName)
- Java is case sensitive. In fact Java file names are also case sensitive.
- Both ***javac*** and ***java*** work with case sensitive file names.

Examining JavaTest.java

- In Java, there is no provision to declare a class, and then define the member functions outside the class.
- Body of every member function of a class (called **method** in Java) must be written when the method is declared.
- So, there is no concept of declaring a prototype of a method in Java.
- Java methods can be written in any order in the source file.
- A method defined earlier in the source file can call a method defined later in the source file.

Examining JavaTest.java

- ***public static void main(String args[ ])*** is the starting point of every Java application.
- Here,
  - ***public*** is used to make the method accessible by all.
  - ***static*** is used to make main a static method of class JavaTest. Static methods can be called without using any object; just using the class name is enough. Being static, main can be called by the JVM using the ***ClassName.methodName*** notation.
  - ***void*** means main does not return anything.
  - ***String args[ ]*** represents a function parameter that is an array of **String** objects. This array holds the command line arguments passed to the application. *Where is the length of args array ?*

Examining JavaTest.java

- Think of JVM as a Java entity who tries to access the **main** method of class **JavaTest**.
- To do that **main** must be accessible from outside of class **JavaTest**.
  - *So, **main** must be declared as a **public member** of class **JavaTest**.*
- Also, JVM wants to access **main** without creating an object of class **JavaTest**.
  - *So, **main** must be declared as **static**.*
- Also JVM wants to pass an array of **String** objects containing the command line arguments.
  - *So, **main** must take an array of **String** as parameter.*

Examining JavaTest.java

- ***System.out.println()*** is used to print a line of text followed by a new line.
  - ***System*** is a class inside the Java API.
  - ***out*** is a public static member of class ***System***.
  - ***out*** is an object of another class of the Java API (the actual class name is not important now).
  - ***out*** represents the standard output (similar to ***stdout*** or ***cout***).
  - ***println*** is a public method of the class of which ***out*** is an object.

Examining JavaTest.java

- ***System.out.print()*** is similar to ***System.out.println()***, but does not print a new line automatically.
- ***System.out.printf()*** is used to print formatted output like ***printf()*** in C.
- In Java, characters enclosed by double quotes (“”) represents a ***String*** object, where **String** is a class of the Java API.
- We can use the plus operator (+) to concatenate multiple ***String*** objects and create a new ***String*** object.

Compiling a Java Program

- Place the source file in the bin directory of your Java installation.
  - *C:\Program Files\Java\jdk1.6.0_07\bin*
- Open a Command Prompt window and go to the bin directory.
- Execute the following command –
  - ***javac JavaTest.java***
- If the source code is ok, then ***javac*** (the Java compiler) will produce a file called ***JavaTest.class*** in the current directory (bin).

Compiling a Java Program

- If the source file contains multiple classes then ***javac*** will produce separate ***.class*** files for each class.
- Every compiled class in Java will have their own ***.class*** file having the name of the class with a ***.class*** extension.
- ***.class*** files contain the bytecodes of each class.
- So, a ***.class*** file in Java contains the bytecodes of a single class only.

Executing a Java Program

- Open a Command Prompt window and go to the bin directory (or use the same window used in the compilation step).
- Execute the following command –
  - ***java JavaTest***
  - Note that we have omitted the ***.class*** extension here.
- Here ***java*** (the JVM) will look for the class file ***JavaTest.class*** and search for a ***public static void main(String args[ ])*** method inside the class.
- If the JVM finds the above two, it will execute the body of the main method.
- If the JVM fails to locate the class or fails to find the proper main method inside the class, it will generate an error and will exit immediately.

Another Java Program

```
public class A
{
    private int a;
    public A()
    {
        this.a=0;
    }
    public void setA(int a)
    {
        this.a=a;
    }
    public int getA()
    {
        return this.a;
    }
```

```
public static void main(String args[])
{
    A ob;
    ob=new A();
    ob.setA(10);
     System.out.println(ob.getA());
}

} // end of class A
```

Examining A.java

- The variable of a class type is called a **reference**.
  - Here ***ob*** is a reference to ***A*** object.
- Merely declaring a class reference is not enough. We have to use ***new*** to create an object.
- We access a public member of a class using the dot operator (.).
- Dot (.) is the only member access operator in Java.
- Java does not have scope resolution operator (::), arrow operator (->), address operator (&) or indirection operator (*).
- ***Every Java object has to be instantiated using keyword new.***

Primitive (built-in) Datatypes

- *Integers*
  - byte 8-bit integer (new).
  - short 16-bit integer.
  - int 32-bit signed integer.
  - long 64-bit singed integer.
- *Real Numbers*
  - float 32-bit floating-point number.
  - double 64-bit floating-point number.
- *Other types*
  - char 16-bit,Unicode 2.1 character.
  - boolean true or false, false is not 0 in Java

Non-primitive Data types

- The non-primitive data types in java are
  - Objects
  - Array
- Non-primitive types are also called reference types
- Object example

```
class Box
{
   int L, W, H;
   Box(int l, int w, int h) { L = l; W = w; H = h; }
}
Box p; // p is a reference pointing to null
p = new Box(1, 2, 3);  // now the actual object is created.
```

Primitive Vs. Non-primitive type

- Primitive types are handled ***by value*** – the actual primitive values are stored in variable and passed to methods.
  - int x = 10, y = x;
  - private MyFunction(int x, float m) { }
- All objects and arrays (non-primitive data types) are handled ***by reference*** – the reference is stored in variable and passed to methods.

Java References

- Java references are used to point to Java objects created by **new**.
- Java objects are always passed by reference to other functions.
- In Java you can never pass an object to another function by-value.
- Java references act as pointers but does not allow pointer arithmetic.
- We cannot read the value of a reference and hence cannot find the address of a Java object.
- We cannot take the address of a Java reference.

Java References (contd.)

- But, we can make a Java reference point to a new object.
  - By copying one reference to another
    - ***ClassName ref2 = ref1***; // Here ref1 is declared in some earlier code
  - By creating a new object using **new** and placing the returned address to the reference in question.
    - ***ClassName ref1 = new ClassName();***
- We cannot place arbitrary values to a reference except the special value **null** which means that the reference is pointing to nothing.
  - ***ClassName ref1 = 100;*** // compiler error
  - ***ClassName ref2 = null;*** // no problem

Java References (contd.)

Code:	***Memory:***
`Box box1;` `Box box2;`	box1 —H box2 —H
`box1 = new Box(8, 5, 7);`	box1 → L=8, W=5, H=7 box2 —H

`box2 = box1;`
`// note: two references`
`// but only one object`
	box1 → L=8, W=5, H=7
	box2 → L=8, W=5, H=7

`box1 =` `new Box(3, 9, 2);`	box1 → L=3, W=9, H=2 box2 → L=8, W=5, H=7
`box1 = box2;` `// Old reference lost!`	box1 → L=8, W=5, H=7 box2 → L=8, W=5, H=7

Boolean Type

- Java boolean type cannot be cast to other types
- In Java, The values 0 and null are not the same as false, and non-zero and non-null values are not the same as true.

C-version

```
int i = 10;
while (i--) {
    int  o = get_int();
    if (o) {
        do { … } while (j);
    }
}
```

Java-version

```
int i = 10;
while (i-- > 0) {
    int  o = get_int();
    if (o != 0) {
        do { … } while (j != 0);
    }
}
```

Textbook

- Java : The Complete Reference (7th Edition)
  - Herbert Schildt
- Java How To Program (7th Edition)
  - Deitel and Deitel
- Course materials and others can be found at
  - http://teacher.buet.ac.bd/rifat/CSE201.html

Thanks