
# Java Programming I

Juan Carlos Moreno - UCLA Ex

# Intro

# Recommended reading


## Java: A Beginner's Guide, Sixth Edition

Herbert Schildt


## Clean Code: A Handbook of Agile Software Craftsmanship

Robert C. Martin


## Design Patterns: Elements of Reusable Object-Oriented Software

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Grady Booch

# Class Structure

Make it as close as possible to a real-life experience

- Hands on Exercises
- Working in teams
- Code reviews
- 2 Quizes
- Final Project

# Computer languages

- Interpreted vs Precompiled vs Native
- Static typed vs Dynamic typed
- Strongly typed vs weak typed
- Brackets vs Indentation
- Spaces vs Tabs

# So what's so great about Java?

| | |
|---|---|
| ***Portable*** | Write once run everywhere |
| ***Secure*** | Memory management and isolation |
| ***Object-oriented*** | The current icon of OOP |
| ***Robust*** | strongly typed, good exception management |
| ***Multithreaded*** | Good for multithread and distributed environments |
| ***Fast*** | One of the fastest languages around |
| ***Popular*** | 2nd most popular language on github |

# Who is using Java now?

- Google

- Spring Framework

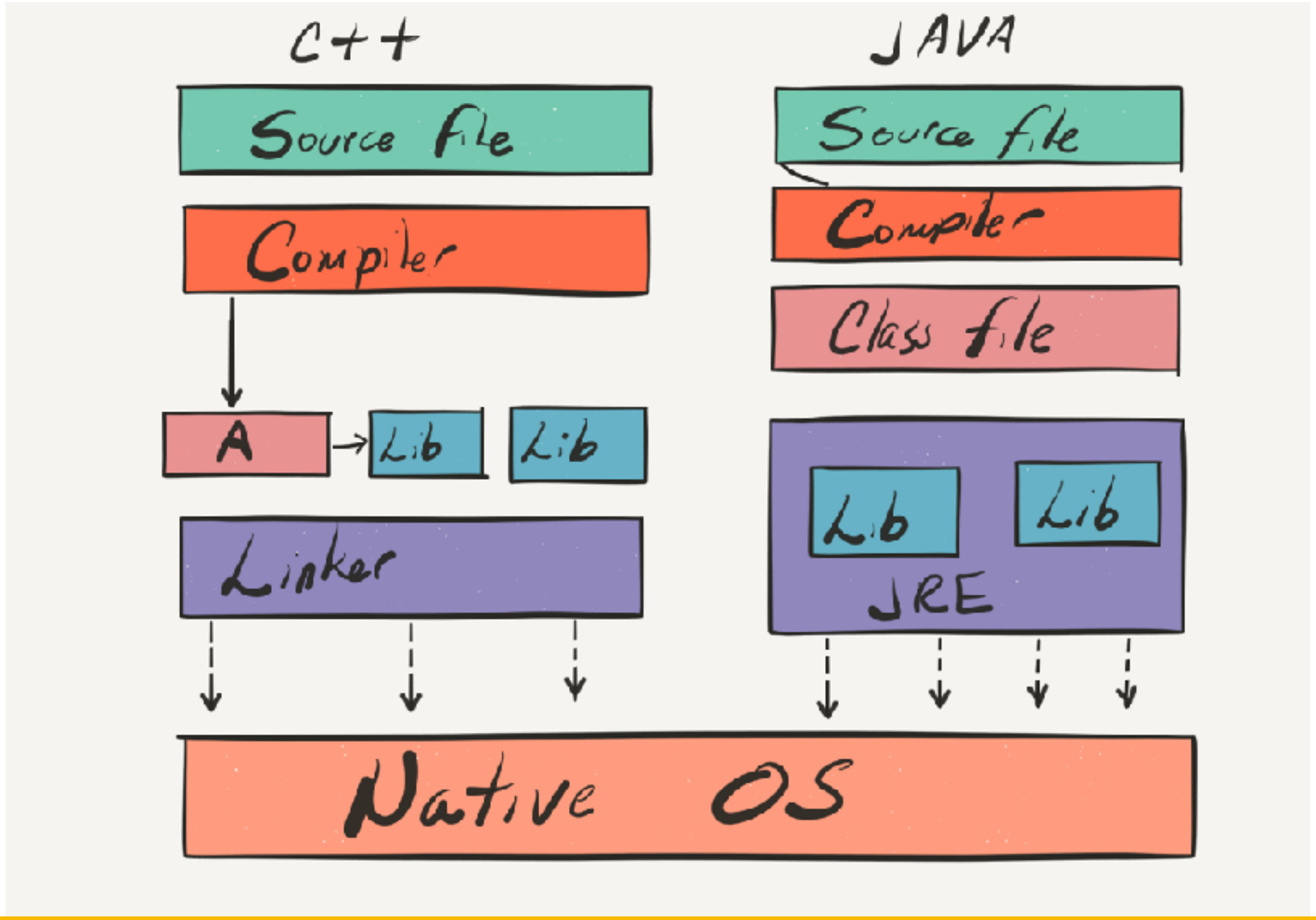
- Hadoop (Google FS clone)

- Android Apps

# Disadvantages of Java

- Slow development process
- Deployments are complex
- Code can be “long and ugly”

# The flow of a Java Program



# Good programming practices

- Pair Programming
- Code Reviews
- IDE - IntelliJ
- Packaging, building, listing and distribution tools
- Clean Code

# Computer Program basics

- STDOUT, STDERR
- Exit codes (Zero is ok)
- Entry point (main function)
- What is Multithreading?
- Servers?

# Programming styles

- MVC - Model View Container
- MVVM - Model View ViewModel
- ReactiveX - Observer pattern
- Flux - Actions, stores, dispatcher

# Hello World

```
public class HelloWorld {
    public static void main(String[] args) {
        // Prints "Hello, World" to the terminal window.
        System.out.println("Hello, World");
    }

}
```

# Running it

- javac HelloWorld.java

- java HelloWorld.class

# Hello World ++

```
public class HelloWorldPlusPlus {

    public static void main(String[] args) {
        System.out.print(“Hello, ");
        System.out.print(args[0]);
        System.out.println(". How are you?");
    }

}
```

# Troubleshooting

- Handling compiler errors
- Handling runtime errors
- Debugging

# Java terminology

- JRE - The runtime (Virtual Machine) that interprets precompiled code and translates it to native code.
- Bytecode - A precompiled code that the virtual machine understands that is translated to native instructions
- Package - A group of similar classes organized by folders
- JAR - Java Archive: A file that contains one or more packages for distribution

# Object Oriented Programming

- Objects and Classes
- Encapsulation - (private, protected, public)
- Composition, Inheritance and Delegation
- Polymorphism

# Encapsulation

hiding the implementation details from users

```
public class EncapsulationDemo{
    private int age;

    public int getAge(){
        return age;
    }

    public void setAge(int newValue){
        age = newValue;
    }
}

public class EncapsTest{
    public static void main(String args[]){
         EncapsulationDemo obj = new EncapsulationDemo();
         obj.setAge(32);
         System.out.println("Employee Age: " + obj.getAge());
    }
}
```

# Inheritance

Allowing a class to inherit properties and methods from other classes

```
class Vehicle {
   String color;
   int speed;
   int size;

   void attributes() {
      System.out.println("Color : " + color);
      System.out.println("Speed : " + speed);
      System.out.println("Size : " + size);
   }
}

class Car extends Vehicle {
   int CC;
   int gears;

   void attributescar() {
      // The subclass refers to the members of the superclass
      this.attributes();
      System.out.println("CC of Car : " + CC);
      System.out.println("No of gears of Car : " + gears);
   }
}
```

# Polymorphism

capability of a method to do different things based on the object that it is acting upon

```
class Overload
{
    void demo (int a)
    {
        System.out.println ("a: " + a);
    }

    void demo (int a, int b)
    {
        System.out.println ("a and b: " + a + "," + b);
    }

    double demo(double a) {
        System.out.println("double a: " + a);
        return a*a;
    }
}
```

# Next Class

- git knowledge would be preferred

- When do you prefer to do it?