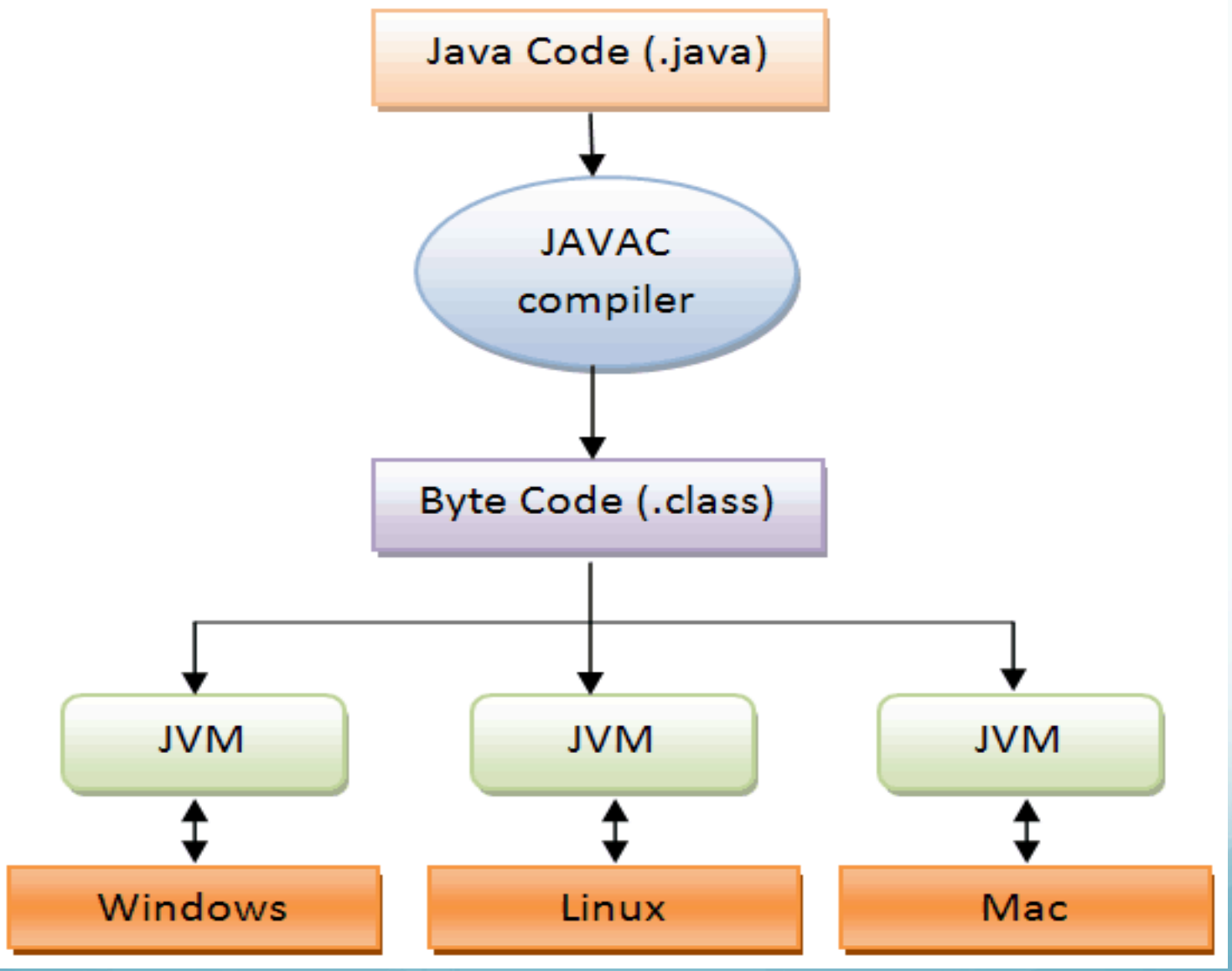
Scala Programming Introduction

March 22, 2016

Agenda

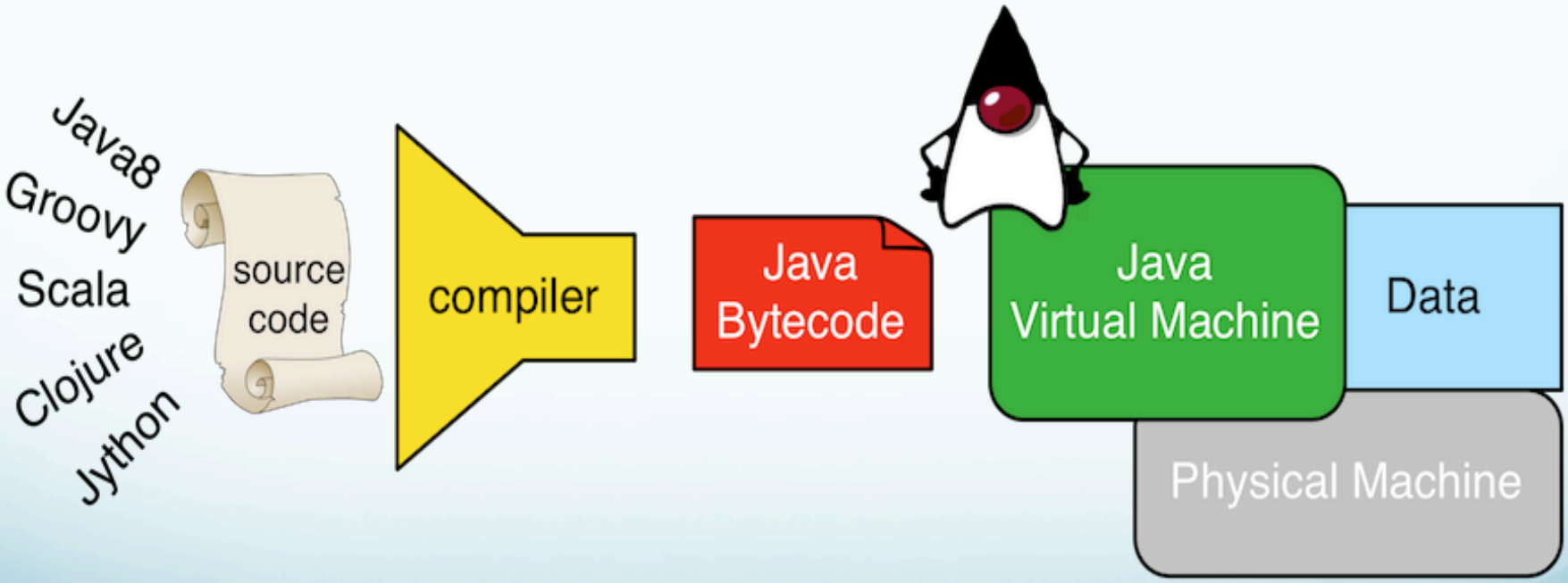
- Java/Scala Development workflow
- What is wrong with Java ?
- Why Should I learn Scala ?
  - Scala more “scalable” than Java
  - Scala is more “pure” than Java
  - Scala is typesafe but require less typing
  - Scala is much more “concise” and expressive
  - Scala encourage good programming practices
- Scala Type Hierarchy
- Scala Pattern matching

Java/Scala Development Workflow



A “Versatile” Virtual Machine

The Java Bytecode is a “contract” between the programming language and Java Virtual Machine which is responsible for providing a uniform abstraction for the underlying hardware.



Java Wrong with Java ?

- What's wrong with Java?
  - Not designed for highly concurrent programs
    - The original Thread model was just *wrong* (it's been fixed)
    - Java 5+ helps by including `java.util.concurrent`
  - Verbose
    - Too much of `Thing thing = new Thing();`
    - Too much "boilerplate," for example, getters and setters

- What's right with Java?
  - Very popular
  - Object oriented (mostly), which is important for large projects
  - Strong typing (more on this later)
  - The fine large library of classes
  - **The JVM!** Platform independent, highly optimized

Scala is more “Scalable” than Java

- Java has *operators* (+, <, ...) and *methods*, with different syntax
- In Scala, operators are just methods, and in many cases you can use either syntax

```
class Complex(val real: Int, val imaginary: Int) {
  def +(operand: Complex) : Complex = {
    new Complex(real + operand.real, imaginary + operand.imaginary)
  }
  override def toString() : String = {
  real + (if (imaginary < 0) "" else "+") + imaginary + "i"
  }
}
```

```
val c1 = new Complex(1, 2)
val c2 = new Complex(2, -3)
val sum = c1 + c2
println("(" + c1 + ") + (" + c2 + ") = " + sum)
```

- Scala as a programming language core library is very small and most functionality is provided through libraries so that programmers have more control over the structure of their programs.

Scala is more Consistent / Pure than Java

- Java is not a pure object oriented language as it has primitives which are there for performance reasons e.g. memory footprint of primitive is approximately 8 times less that of corresponding wrapper classes.
- But this introduces inconsistency in treatment of primitives and object references. E.g. "==" operator for primitive types is equality by value in case of primitives but "==" is equality by reference in case of object types. You need to use equals method for value comparison in case of object types.

```
int pnum1 = 10;
int pnum2 = 10;
pnum1 == pnum2 // comparison by value in case of primitives

Integer onum1 = new Integer(10);
Integer onum2 = new Integer(10);
onum1 == onum2 // comparison by reference type

onum1.equals(onum2) // comparison by value in case of objects
```

In Scala, all values are objects. Period. The compiler turns them into primitives, so no efficiency is lost.

```
val snum1: Int = 10
val snum2: Int = 10

snum1 == snum2 //comparison by value

snum1.eq(snum1) // comparison by reference
```

Type Safety without too much "Typing"

- Java is statically typed--a variable has a type, and can hold *only* values of that type
  - You must specify the type of every variable
  - Type errors are caught by the compiler, not at runtime--this is a big win
  - However, it leads to a lot of typing (pun intended)

- Languages like Ruby and Python don't make you declare types
  - Easier (and more fun) to write programs
  - Less fun to debug, especially if you have even slightly complicated types

- Scala is *also* statically typed, but it uses type inferencing--that is, it figures out the types, so you don't have to
  - The good news: Less typing, more fun, type errors caught by the compiler
  - The bad news: More kinds of error messages to get familiar with

Scala is much more “Concise”

- Java:
  - ```
    class Person {
        private String firstName;
        private String lastName;
        private int    age;

        public Person(String firstName, String lastName, int age) {
            this.firstName = firstName;
            this.lastName  = lastName;
            this.age  = age;
        }

        public void setFirstName(String firstName) { this.firstName = firstName; }
        public void String getFirstName() { return this.firstName; }
        public void setLastName(String lastName) { this.lastName = lastName; }
        public void String getLastName() { return this.lastName; }
        public void setAge(int age) { this.age = age; }
        public void int getAge() { return this.age; }
    }
    ```
- Scala:
  - ```
    class Person(var firstName: String, var lastName: String, var age: Int)
    ```

A Billion Dollar Mistake

"I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years."

--Sir Tony Hoare

Scala Encourages Good Programming Practices

- In Java, any method that is *supposed* to return an object *could* return null
  - Here are your options:
    - Always check for null
    - Always put your method calls inside a try...catch
    - Make sure the method can't possibly return null

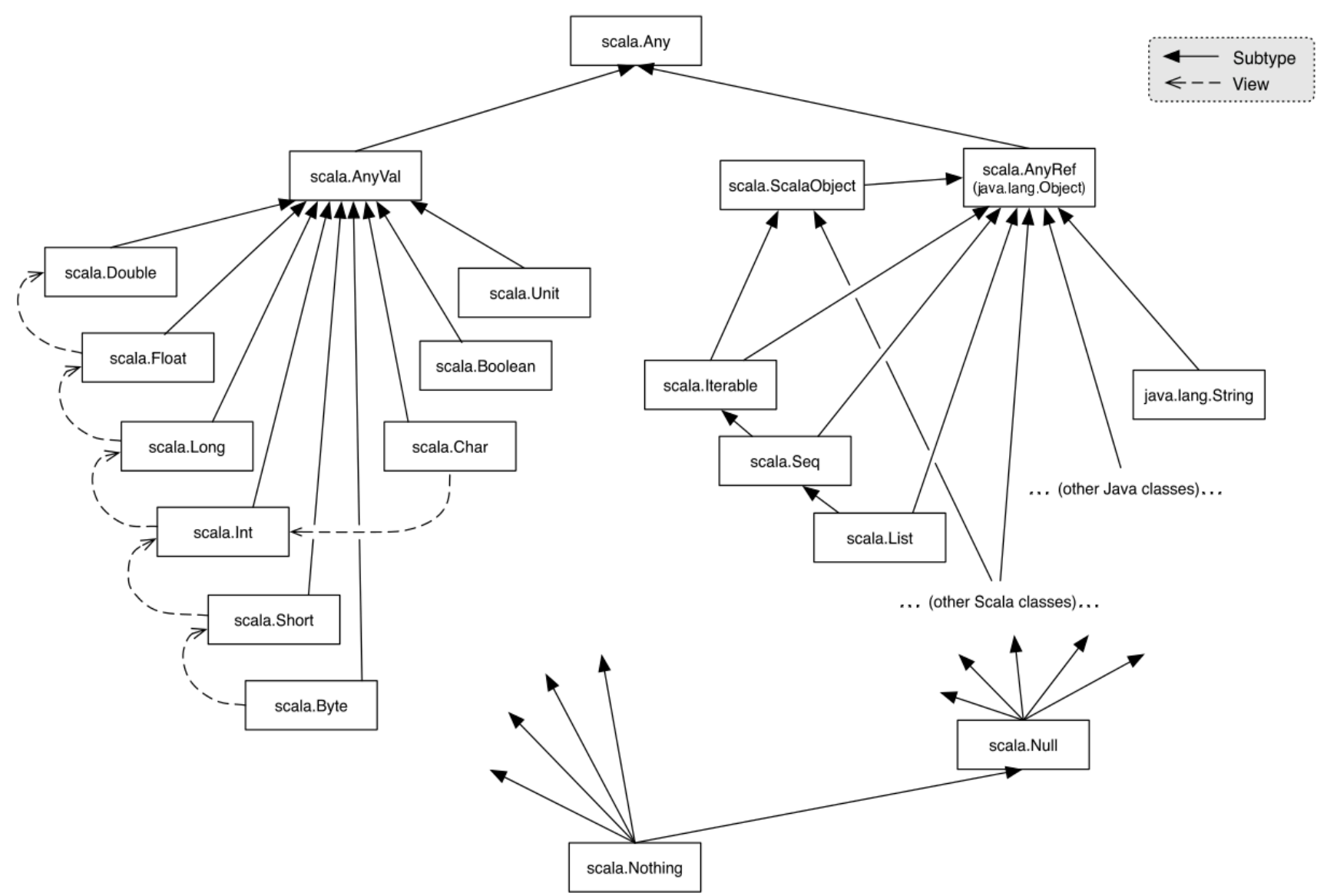
- Yes, Scala has null--but only so that it can talk to Java
- In Scala, if a method *could* return "nothing," write it to return an Option object, which is either Some(***theObject***) or None

```
//written by library which is exposed as external library
val getGreeting: Option[String] = Some("Hello World")

//client has not direct reference to String which could be null
// Has to think what happens when greeting is not present
getGreeting getOrElse "Default Greeting"

// If getGreeting changes client is immune
// will not get null pointer exception
val getGreeting: Option[String] = None
```

Scala Type Hierarchy



Deep Pattern Matching

```
object PatternMatchingDemo extends App{

  def matchTest(x: Any): Any = x match {
    case 1 => "one"
    case "two" => 2
    case y: Int => s"scala.Int - $y"
    case Some(str) => str  // pattern matching and binding
    case None => "None"
    case _ => "Not matched"  //catch all
  }


  println(matchTest(1))
  println(matchTest("two"))
  println(matchTest(2))
  println(matchTest(Some("someValue")))
  println(matchTest(None))
  println(matchTest("Some Random Value"))

}
```