Scalable language

A language which grows with the demands of its users!!

- Scala is a modern multi-paradigm programming language designed to express common programming patterns in a concise, elegant, and type-safe way
- Scala has been created by Martin Odersky and he released the first version in 2003.
- Scala smoothly integrates the features of object-oriented and functional languages.

Scala

- Runs on JVM
- Functional : Every function is a value
- Object oriented: Every value is an Object
- Concise and elegant
- Descent IDE support(Netbeans, Eclipse IntelliJ)

Scala is compiled into Java Byte Code which is executed by the Java Virtual Machine (JVM). This means that Scala and Java have a common runtime platform. You can easily move from Java to Scala.

Scala allows you to express general programming patterns in an effective way. It reduces the number of lines and helps the programmer to code in a type-safe way. It allows you to write codes in an immutable manner, which makes it easy to apply concurrency and parallelism (Synchronize).

Installing scala

The most popular way to get Scala is either using Scala through sbt, the Scala build tool, or to use Scala through an IDE.

https://www.scala-lang.org/download/

Download SBT : http://www.scala-sbt.org/download.html

sudo apt-get install scala

REPL : Read Eval Print Loop

Fiddle

https://scalafiddle.io/sf/gKgxQY0/1

https://scastie.scala-lang.org/

Variable declaration

1. **Mutable** variable

It is a variable that can change value

2. **Immutable** variable (Constant)

It is a variable that can not be changed

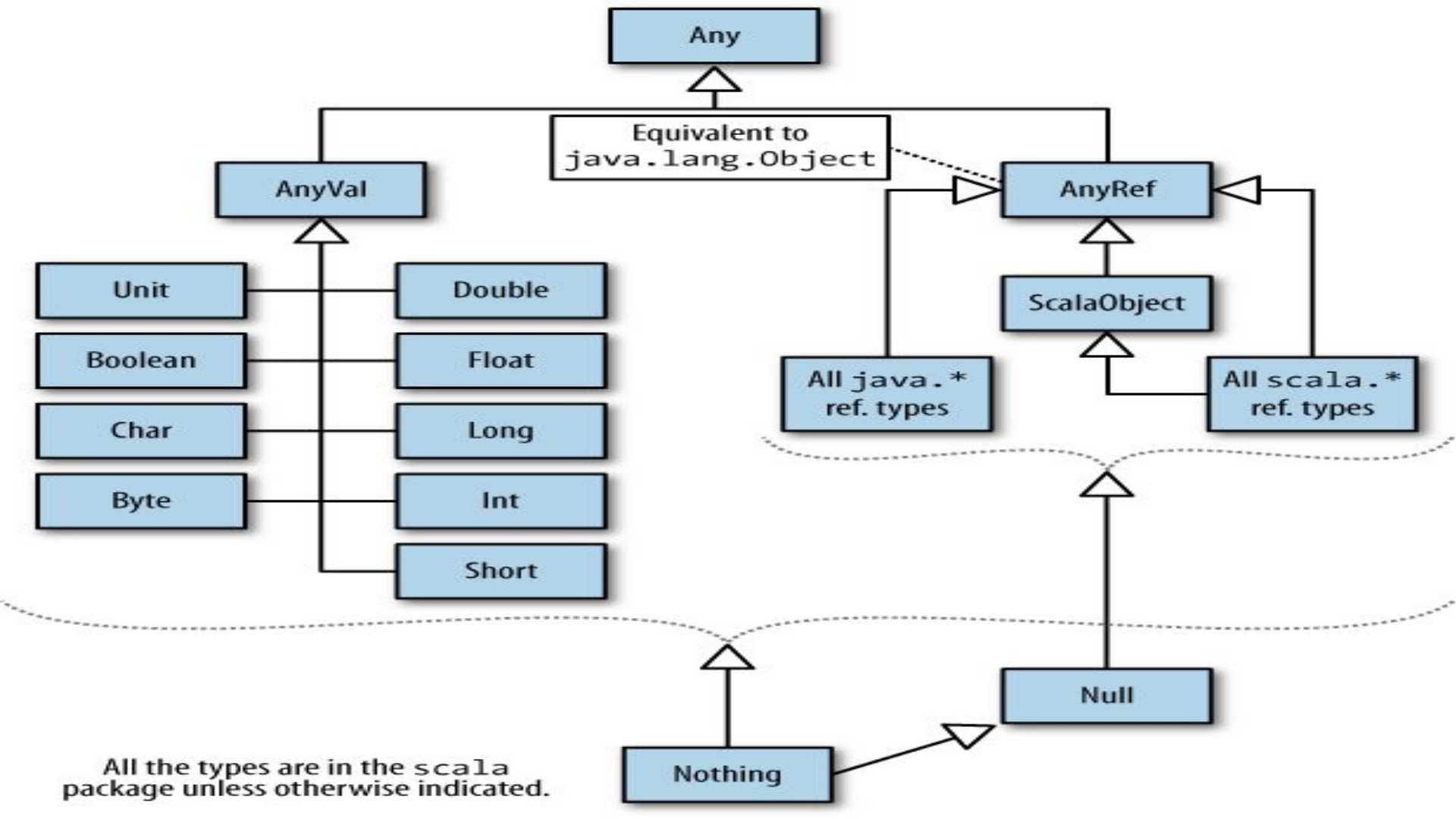
Variable Type Inference

When you assign an initial value to a variable, the Scala compiler can figure out the type of the variable based on the value assigned to it. This is called variable type inference

DataType

- Byte
- Short
- Int
- Long
- Float
- Double
- Char
- String
- Boolean
- Unit
- Null
- Nothing
- Any
- AnyRef

All the data types listed here are objects. There are no primitive types like in Java. This means that you can call methods on an Int, Long, etc.


Any
Equivalent to
java.lang.Object
AnyVal
AnyRef
Unit
Double
Boolean
Float
Char
Long
Byte
Int
Short
ScalaObject
All java.*
ref. types
All scala.*
ref. types
Null
Nothing
All the types are in the scala
package unless otherwise indicated.

Null– It is a Trait.

null– It is an instance of Null- Similar to Java null.

Nil – Represents an empty List of anything of zero length. It is not that it refers to nothing but it refers to List which has no contents.

None– Used to represent a sensible return value. Just to avoid null pointer exception. **Option** has exactly 2 subclasses - **Some** and **None**. None signifies no result from the method.

Unit– Type of method that doesn't return a value of any sort. Similar to void

If any of these is a little difficult to get, it's Nothing. **Nothing** is another trait.

Nothing is a subtype of every other type. **Any** is the root type of the entire Scala type system.

Tuples

Scala tuple combines a fixed number of items together so that they can be passed around as a whole

Unlike an array or list, a tuple can hold objects with different types but they are also immutable

Multiple assignments

If a code block or method returns a Tuple (**Tuple** − Holds collection of Objects of different types), the Tuple can be assigned to a val variable.

```
val (a, b) = (1, 2)
```

```
a: Int = 1
```

```
b: Int = 2
```

```
val head :: tail = 1 :: 2 :: 3 :: Nil
```

```
head: Int = 1
```

```
tail: List[Int] = List(2, 3)
```

Function : Functions are expressions that take parameters.

- Anonymous functions

- Name the function

- No parameter

Methods

- Methods look and behave very similar to functions

- Methods are defined with the def keyword. def is followed by a name, parameter lists, a return type, and a body.

- The last expression in the body is the method’s return value. (Scala does have a return keyword, but it’s rarely used.)

Named arguments

Default parameters

Classes

You can define classes with the **class** keyword followed by its name and constructor parameters.

```
class Greeter(prefix: String, suffix: String) {
  def greet(name: String): Unit =
    println(prefix + name + suffix)
}
```

You can make an instance of a class with the new keyword.

```
val greeter = new Greeter("Hello, ", "!")
greeter.greet("Scala developer") // Hello, Scala developer!
```

Singleton / Companion object

- Instead of static variables and methods scala class can have what is called a Singleton object.
- No object is instantiated. It is like calling a static method in java.
- When a singleton object is named same as a class, it is called a companion object
- Companion object must be defined inside same source file.

Case classes

- Case classes are like regular classes

- By default, case classes are **immutable** and **compared by value**

- Instantiate without “new” keyword

```
case class Point(x: Int, y: Int)
```

Curried functions

- Currying is technique of transforming a function that takes multiple arguments into a function that takes a single argument (the other arguments having been specified by the curry).
- Currying is the process of transforming a function that takes multiple arguments into a sequence of functions that each have only a single parameter.

Currying is the technique of transforming a function with multiple arguments into a function with just one argument. The single argument is the value of the first argument from the original function and the function returns another single argument function. This in turn would take the second original argument and itself return another single argument function. This chaining continues over the number of arguments of the original. The last in the chain will have access to all of the arguments and so can do whatever it needs to do.

```scala
def add(a: Int)(b: Int) = a + b

val onePlusFive = add(1)(5) // 6

val addFour = add(4)_ // (Int => Int)

val twoPlusFour = addFour(2) // 6
```

traits

A trait encapsulates method and field definitions, which can then be reused by mixing them into classes. Unlike class inheritance, in which each class must inherit from just one superclass, a class can mix in any number of traits.

```
trait Equal {

 def isEqual(x: Any): Boolean

 def isNotEqual(x: Any): Boolean = !isEqual(x)

}
```

Pattern matching / match expressions

```
var test = "theVal"

val result = test match {

 case "someVal" => println(test + "1")

 case "theVal" => println(test + "4")

}
```

Match expression can also return value

Loop

For loop

```
for(i <- 1 to 10){
 println(i)
}
//1,2,3...10
for(i <- 1 until 10){
 println(i)
}
```

1,2,3 ...9

Higher order functions

```
val plusOne = (x: Int) => x + 1

val nums = List(1,2,3)

nums.map(plusOne);

nums.map(_ + 4);

nums.exists(_ == 2)

nums.find(_ == 1)

nums.indexWhere(_ == 3)
```

Option[A]

If you have worked with Java at all in the past, it is very likely that you have come across a NullPointerException at some time

Scala tries to solve the problem by getting rid of null values altogether and providing its own type for representing optional values, i.e. values that may be present or not: the Option[A] trait.

```
def maybeItWillReturnSomething(flag: Boolean): Option[String] = {
  if (flag) Some("Found value") else None
}
```

Tail recursion

In order for a recursive call to be tail recursive, the call back to the function must be the last action performed in the function. In the example above, because the total returned from the recursive call is being multiplied by number, the recursive call is NOT the last action performed in the function, so the recursive call is NOT tail recursive.

Tail calls are simple to do. If your method is recursive, try to use an accumulator to pass the results back into the method itself so that all the branches of your recursive method either end in a call to itself or a value, without requiring any type of calculations after the recursive call.

Resources

https://www.scala-exercises.org

https://twitter.github.io/scala_school/

Good article on handling data in a functional way:https://github.com/ghik/opinionated-scala/wiki/Handling-data-in-functional-way

https://www.coursera.org/specializations/scala

http://danielwestheide.com/scala/neophytes.html