

Scala is statically-typed

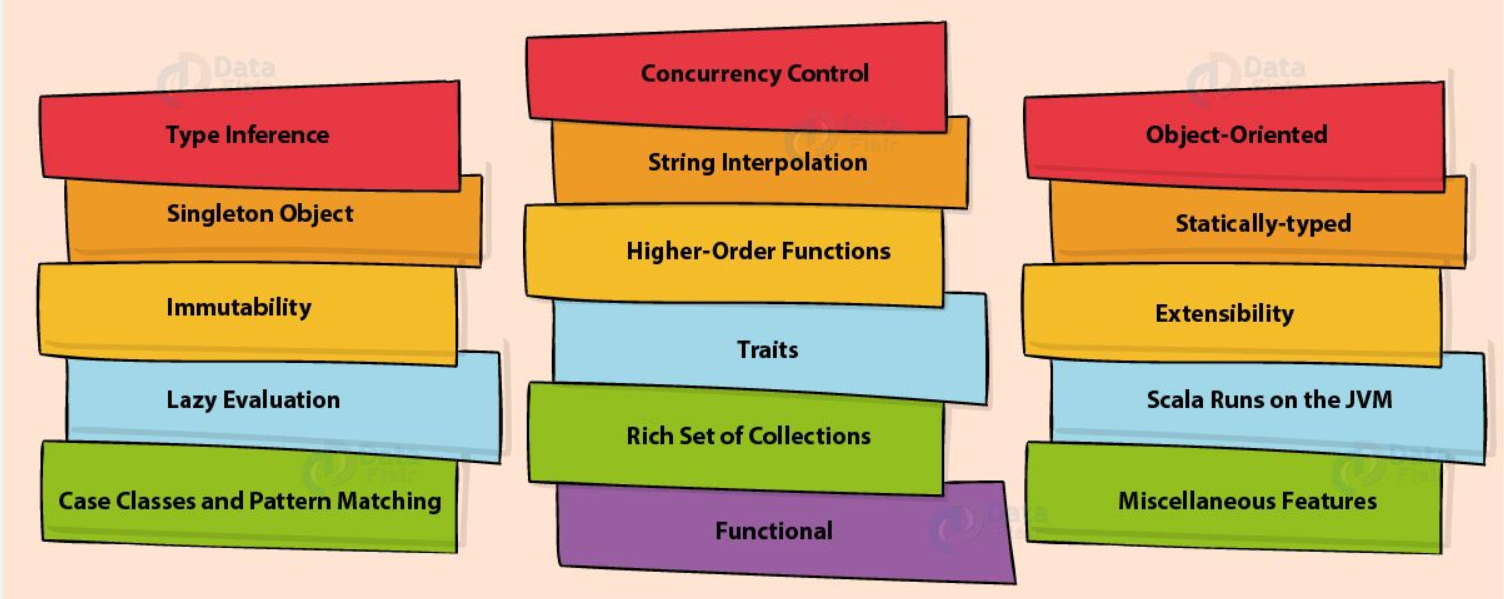
No more getters and setters as you have in java. Enjoy hazzle free

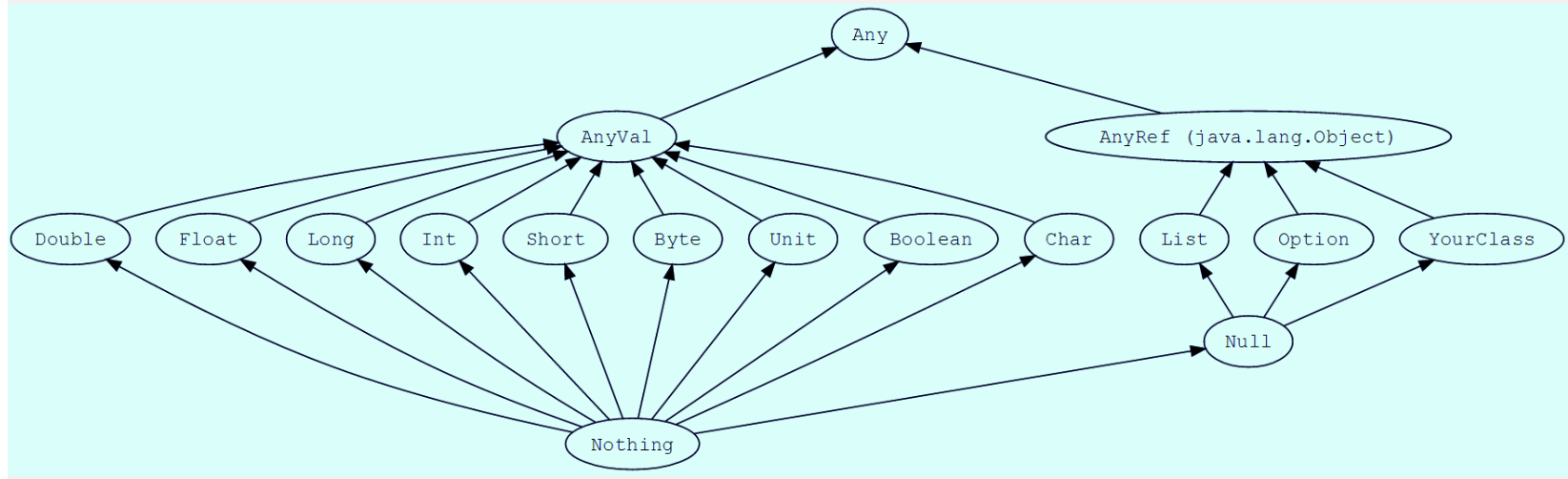
I almost get vomit if I see getters and setters everywhere.

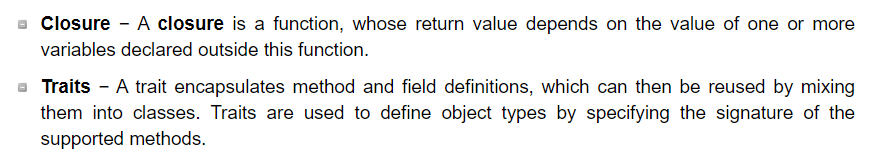
I don’t know which idiot has invented that. I have to say that.

Because entire world is forcefully writing and suffering from getters and setters though that is not prime important.

At least Lombok has helped us to some extent.

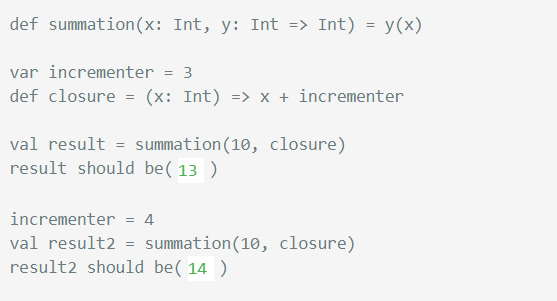


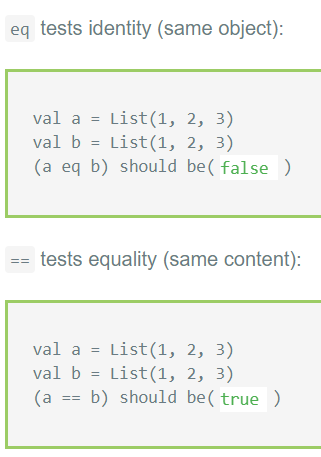


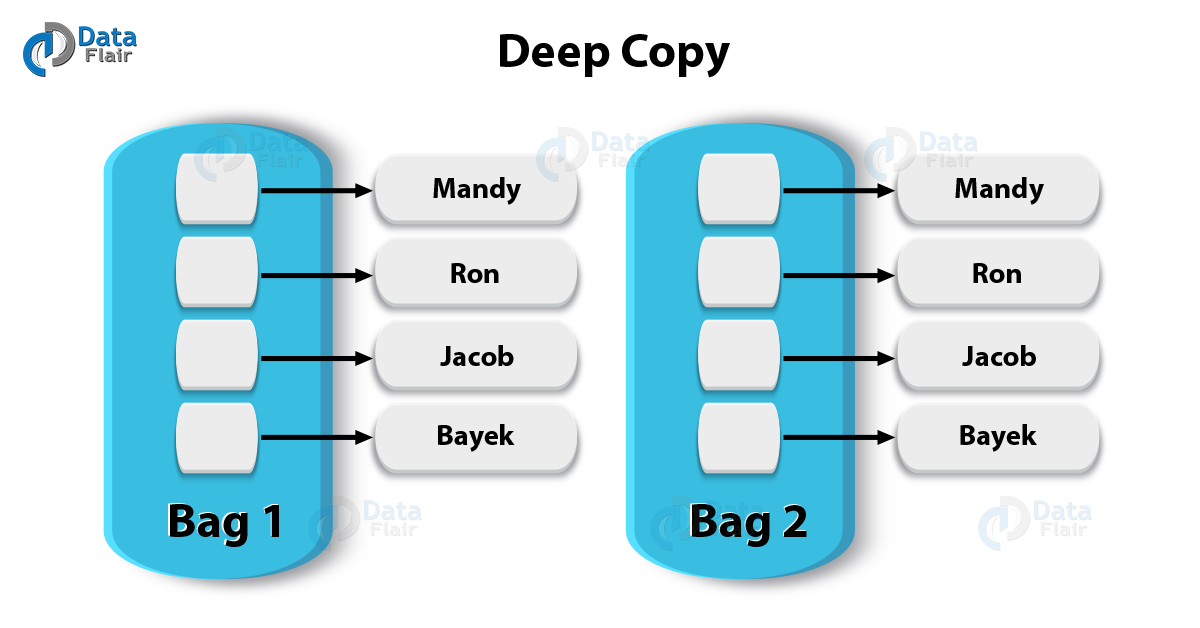


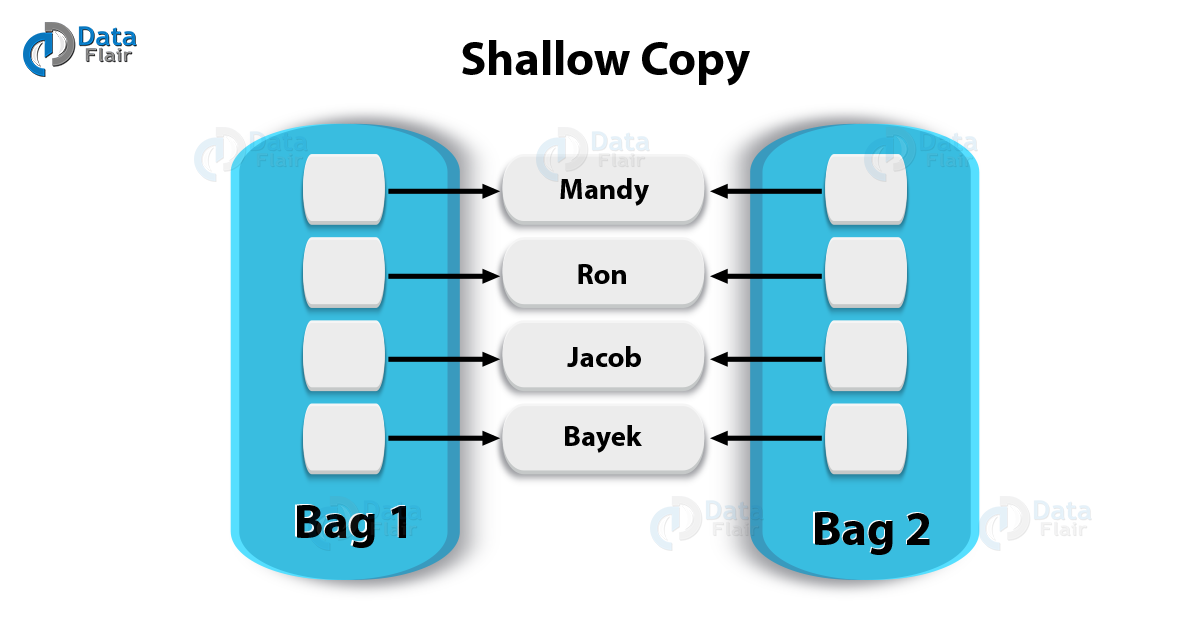
Higher Order Functions: Higher Order Functions are functions that take functions as arguments and/or return functions.

We can take that closure and throw it into a Higher Order Function and it will still hold the environment:













8.7 Closures

scala> var more = 1

more: Int = 1

scala> val addMore = (x: Int) => x + more

addMore: (Int) => Int = <function>

scala> addMore(10)

res19: Int = 11

scala> more = 9999

more: Int = 9999

scala> addMore(10)

res21: Int = 10009

scala> def isStringArray(x: Any) = x match {

         case a: Array[String] => "yes"

         case \_ => "no"

       }

isStringArray: (Any)java.lang.String

scala> val as = Array("abc")

as: Array[java.lang.String] = Array(abc)

scala> isStringArray(as)

res19: java.lang.String = yes

scala> val ai = Array(1, 2, 3)

ai: Array[Int] = Array(1, 2, 3)

scala> isStringArray(ai)

res20: java.lang.String = no

You can use a typed pattern as a convenient replacement for type tests and type casts. [Listing 15.11](https://www.artima.com/pins1ed/case-classes-and-pattern-matching.html#lst:typed-patterns) shows an example:

  def generalSize(x: Any) = x match {

    case s: String => s.length

    case m: Map[\_, \_] => m.size

    case \_ => -1

  }

##### Listing 15.11 - A pattern match with typed patterns.

Here are a few examples of using the generalSize method in the interpreter:

scala> generalSize("abc")

res14: Int = 3

scala> generalSize(Map(1 -> 'a', 2 -> 'b'))

res15: Int = 2

scala> val capitals =

         Map("France" -> "Paris", "Japan" -> "Tokyo")

capitals:

  scala.collection.immutable.Map[java.lang.String,

  java.lang.String] = Map(France -> Paris, Japan -> Tokyo)

scala> capitals get "France"

res21: Option[java.lang.String] = Some(Paris)

scala> capitals get "North Pole"

res22: Option[java.lang.String] = None

 scala> for ((country, city) <- capitals)

           println("The capital of "+ country +" is "+ city)

  The capital of France is Paris

  The capital of Japan is Tokyo

scala> def show(x: Option[String]) = x match {

         case Some(s) => s

         case None => "?"

       }

show: (Option[String])String

scala> show(capitals get "Japan")

res23: String = Tokyo

scala> show(capitals get "France")

res24: String = Paris

scala> show(capitals get "North Pole")

res25: String = ?

  scala> val myTuple = (123, "abc")

  myTuple: (Int, java.lang.String) = (123,abc)

  scala> val (number, string) = myTuple

  number: Int = 123

  string: java.lang.String = abc

 scala> val results = List(Some("apple"), None,

             Some("orange"))

  results: List[Option[java.lang.String]] = List(Some(apple),

      None, Some(orange))

  scala> for (Some(fruit) <- results) println(fruit)

  apple

  orange

scala> import scala.collection.mutable.\_

scala> val q = Queue(1, 2, 3)

q: Queue[Int] = Queue(1, 2, 3)

scala> val q1 = q append 4

q1: Queue[Int] = Queue(1, 2, 3, 4)

scala> q

res0: Queue[Int] = Queue(1, 2, 3, 4)