DATA WAREHOUSE & MINING

TA COURSE #04

SCALA

TA COURSE #04

SCALA

WHAT IS SCALA

WHAT IS SCALA

- Functional language
- Typed and Obj-Oriented
- Born from Java, be beyond Java

TA COURSE #04

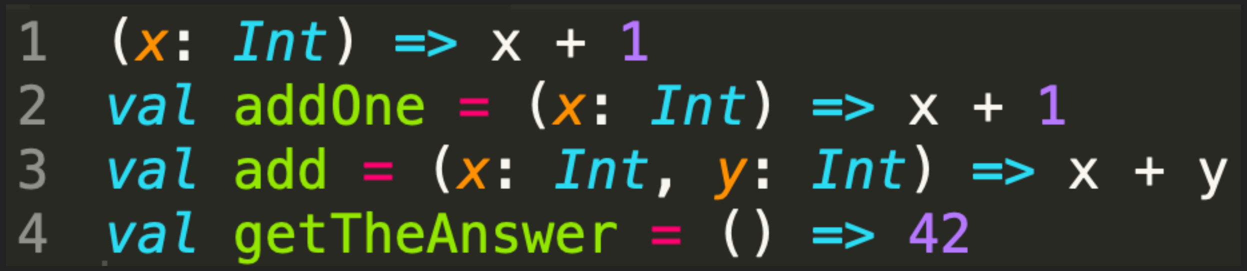
SCALA

KUSOMISO TECHNIQUE

Mainly copied from here

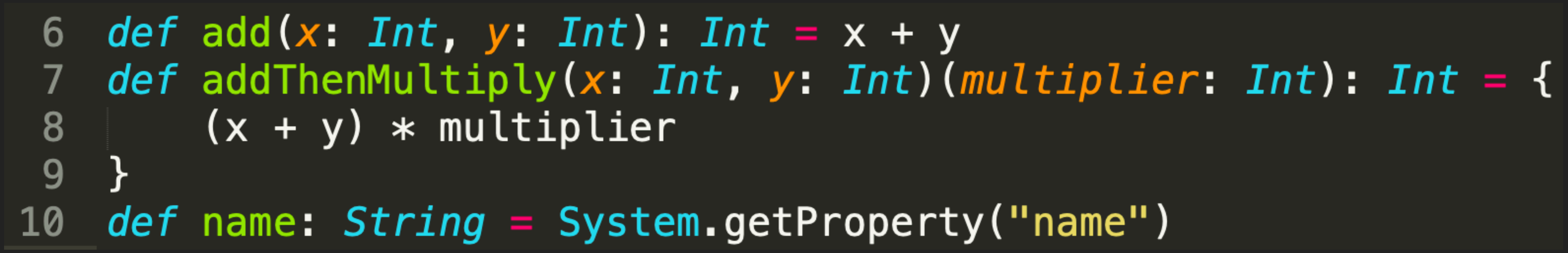
BASICS

- val & var
- (lambda) functions



```scala
(x: Int) => x + 1
val addOne = (x: Int) => x + 1
val add = (x: Int, y: Int) => x + y
val getTheAnswer = () => 42
```

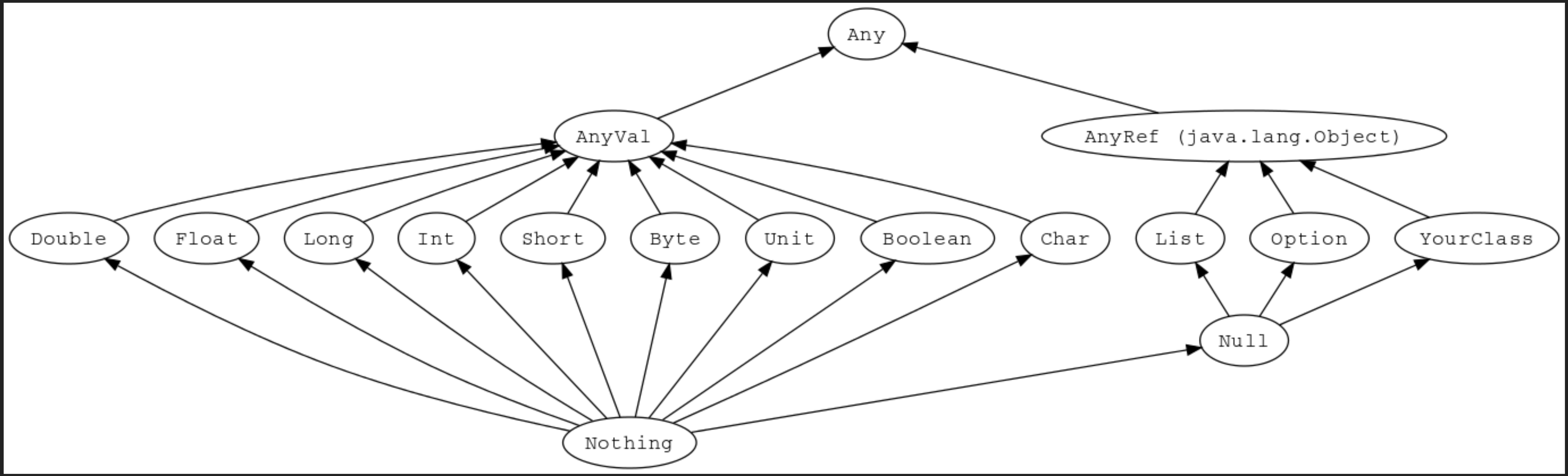
- (customized parameter list) methods

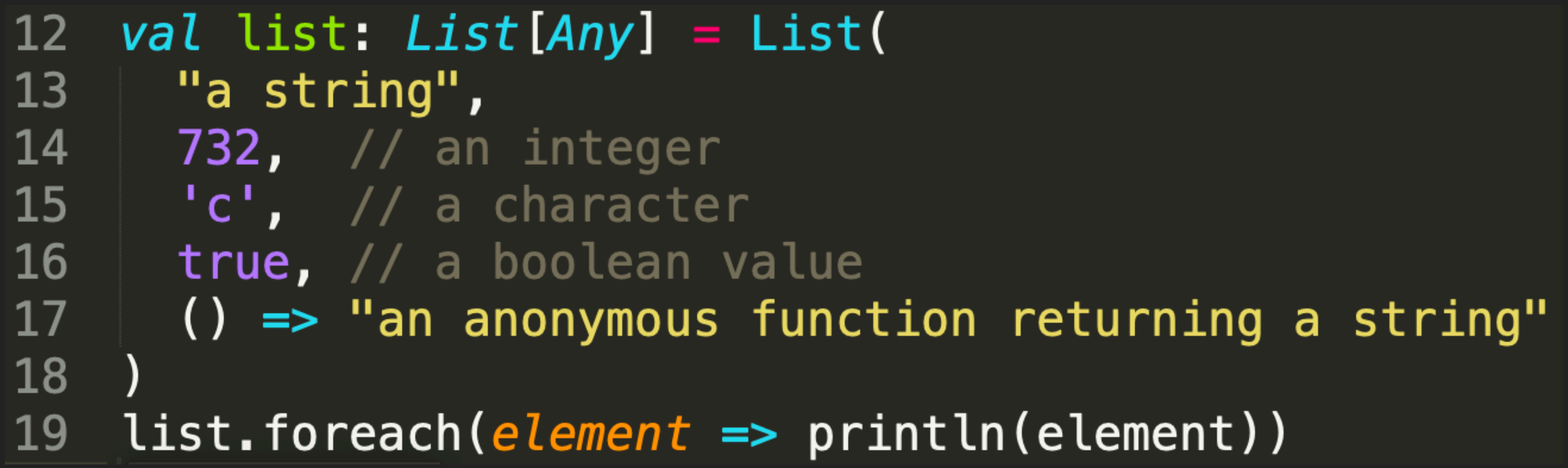


```scala
def add(x: Int, y: Int): Int = x + y
def addThenMultiply(x: Int, y: Int)(multiplier: Int): Int = {
    (x + y) * multiplier
}
def name: String = System.getProperty("name")
```

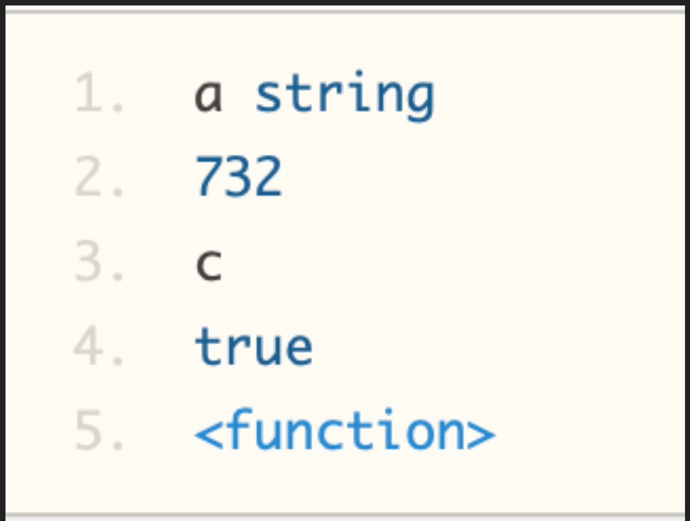
- Entry called “main”

UNIFIED TYPES



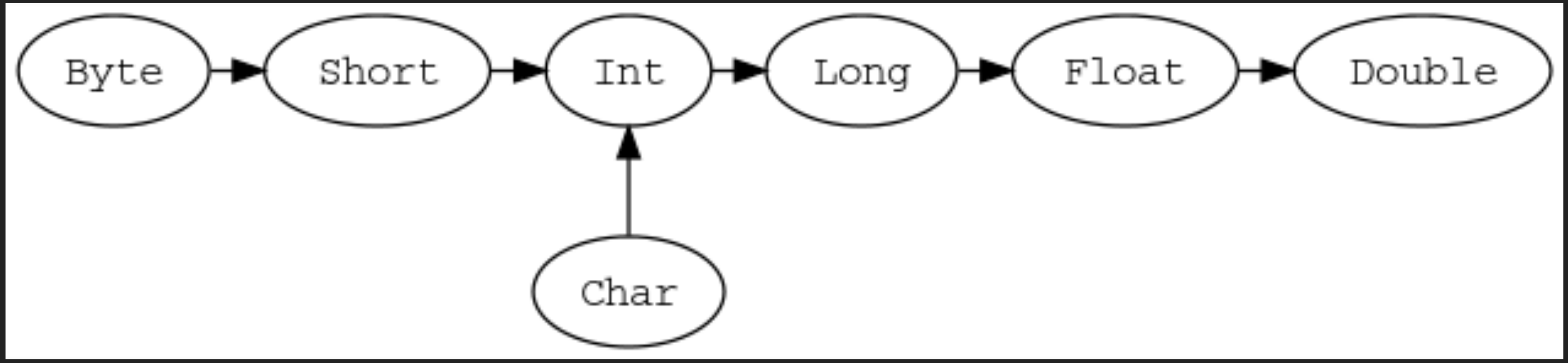


```scala
val list: List[Any] = List(
  "a string",
  732,  // an integer
  'c',  // a character
  true, // a boolean value
  () => "an anonymous function returning a string"
)
list.foreach(element => println(element))
```



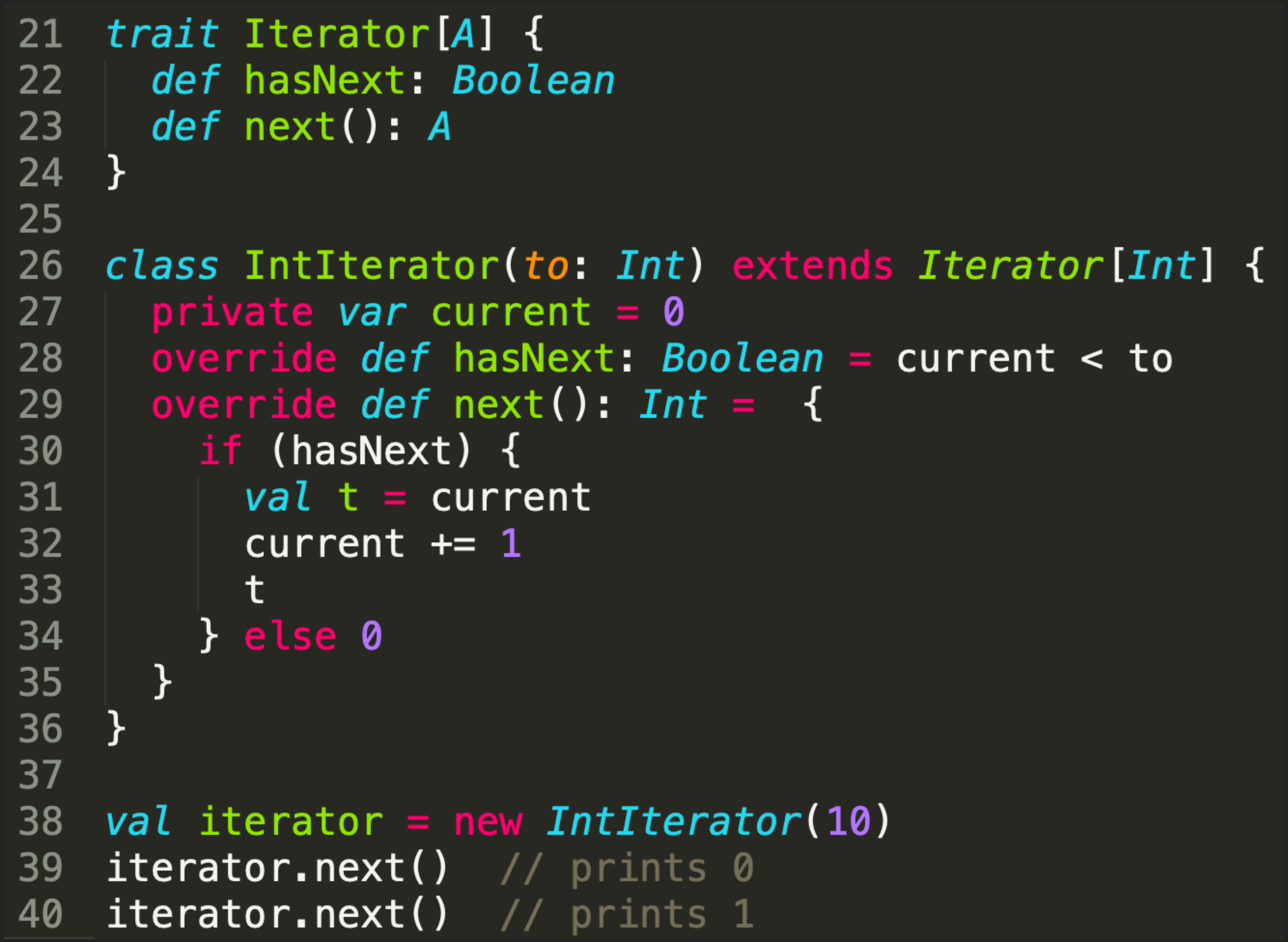
```
a string
732
c
true
<function>
```

TYPE CASTING



TRAIT

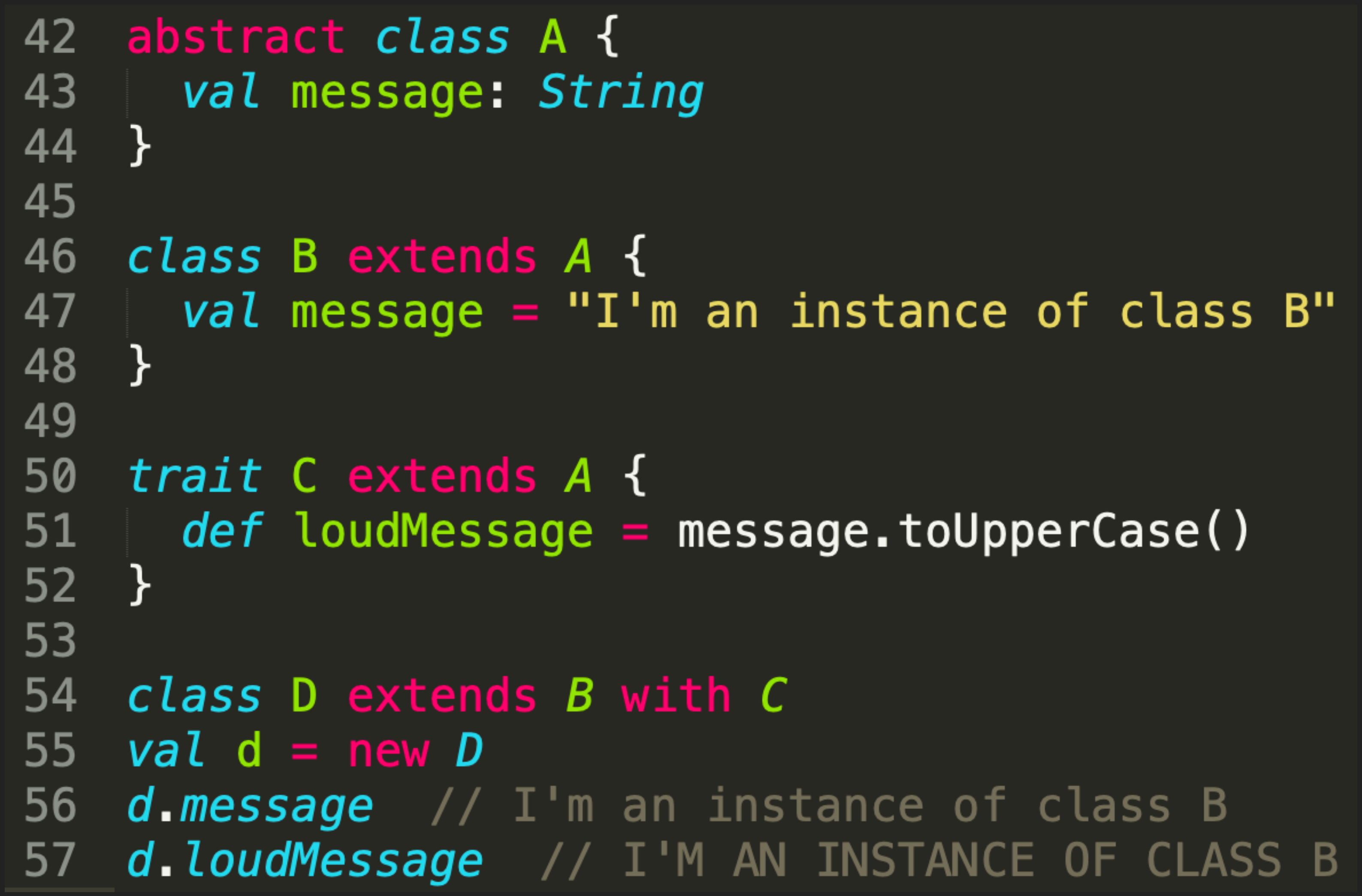
- Class method template
- To implement to class before to use
- Different from abstract class since it can't be instantiated



```scala
trait Iterator[A] {
  def hasNext: Boolean
  def next(): A
}

class IntIterator(to: Int) extends Iterator[Int] {
  private var current = 0
  override def hasNext: Boolean = current < to
  override def next(): Int =  {
    if (hasNext) {
      val t = current
      current += 1
      t
    } else 0
  }
}

val iterator = new IntIterator(10)
iterator.next()  // prints 0
iterator.next()  // prints 1
```

CLASS MIXIN

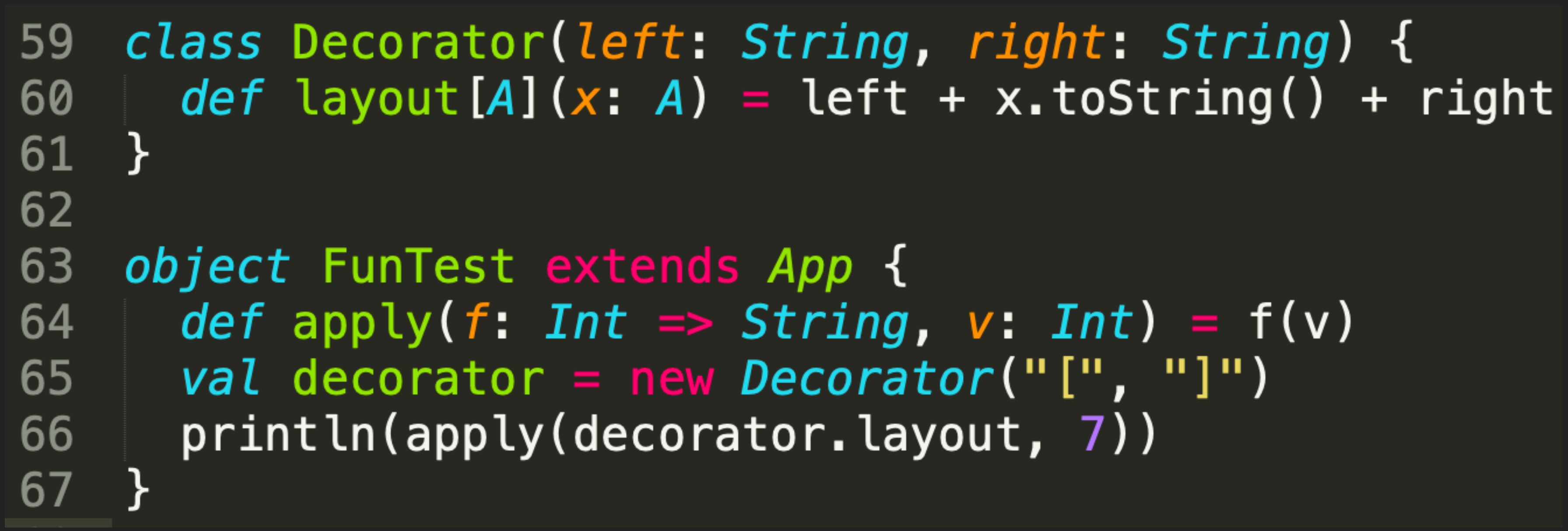
- Mix multiple classes and traits into one



```scala
abstract class A {
  val message: String
}

class B extends A {
  val message = "I'm an instance of class B"
}

trait C extends A {
  def loudMessage = message.toUpperCase()
}

class D extends B with C
val d = new D
d.message  // I'm an instance of class B
d.loudMessage  // I'M AN INSTANCE OF CLASS B
```

HIGHER-ORDER FUNCTION

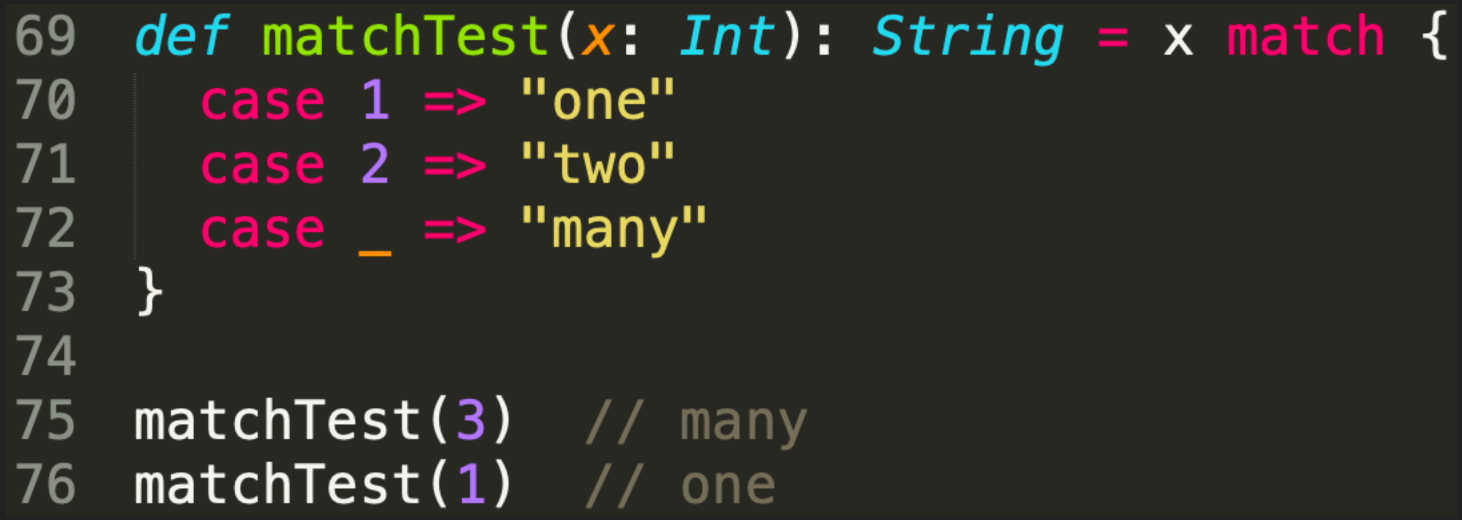
- Function as parameter



```scala
class Decorator(left: String, right: String) {
  def layout[A](x: A) = left + x.toString() + right
}

object FunTest extends App {
  def apply(f: Int => String, v: Int) = f(v)
  val decorator = new Decorator("[", "]")
  println(apply(decorator.layout, 7))
}
```

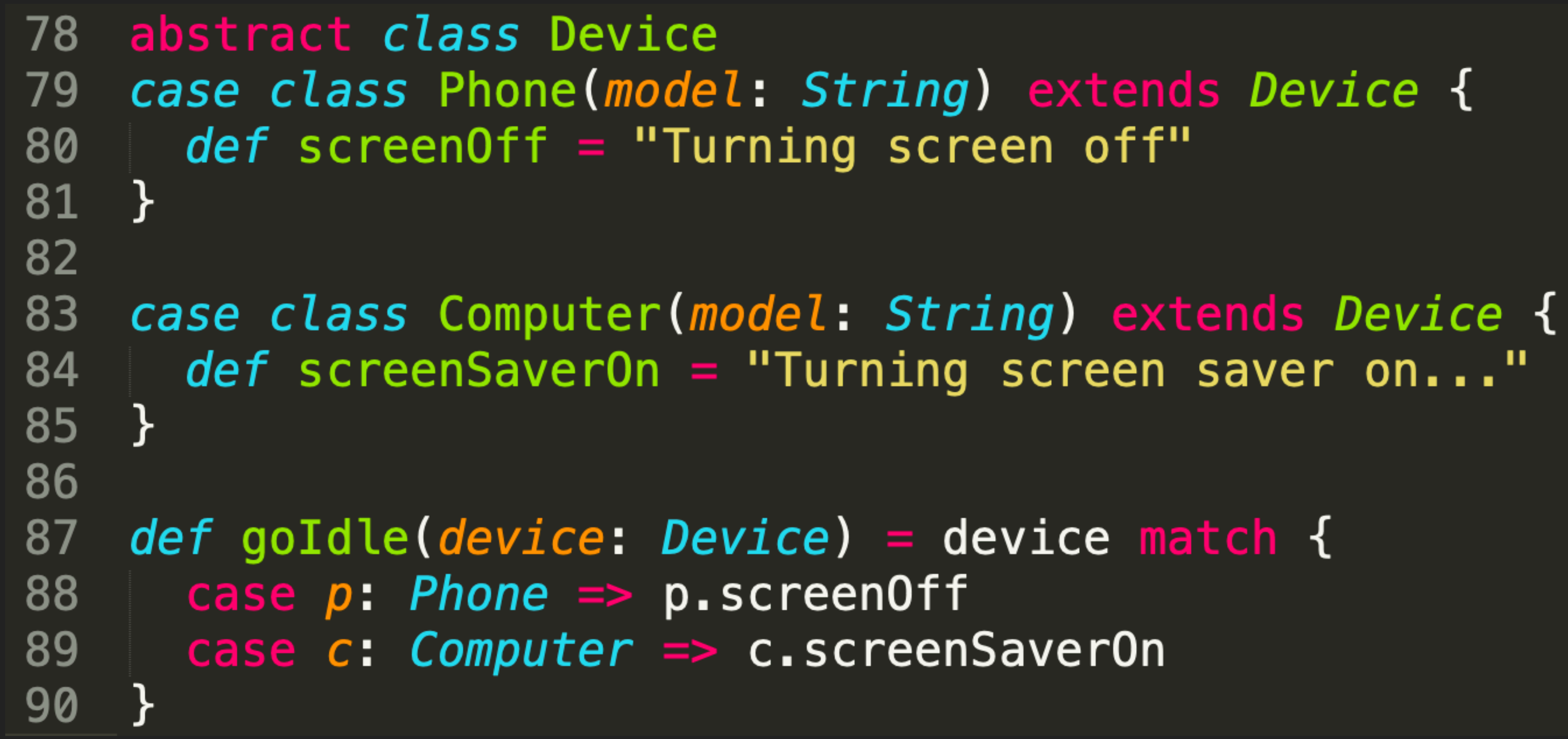
PATTERN MATCHING

- Value matching



```scala
def matchTest(x: Int): String = x match {
  case 1 => "one"
  case 2 => "two"
  case _ => "many"
}

matchTest(3)  // many
matchTest(1)  // one
```

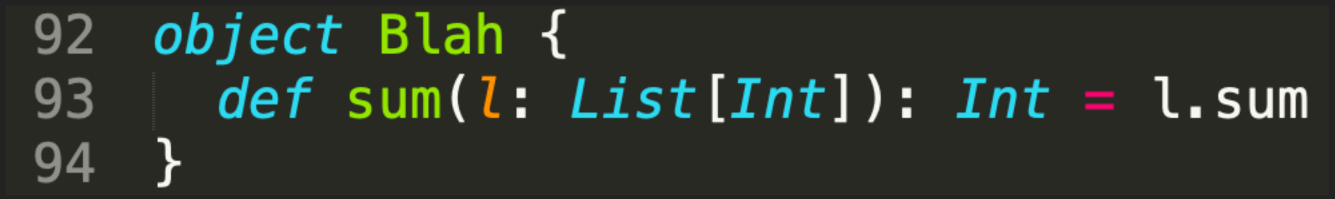
- Type matching



```scala
abstract class Device
case class Phone(model: String) extends Device {
  def screenOff = "Turning screen off"
}

case class Computer(model: String) extends Device {
  def screenSaverOn = "Turning screen saver on..."
}

def goIdle(device: Device) = device match {
  case p: Phone => p.screenOff
  case c: Computer => c.screenSaverOn
}
```

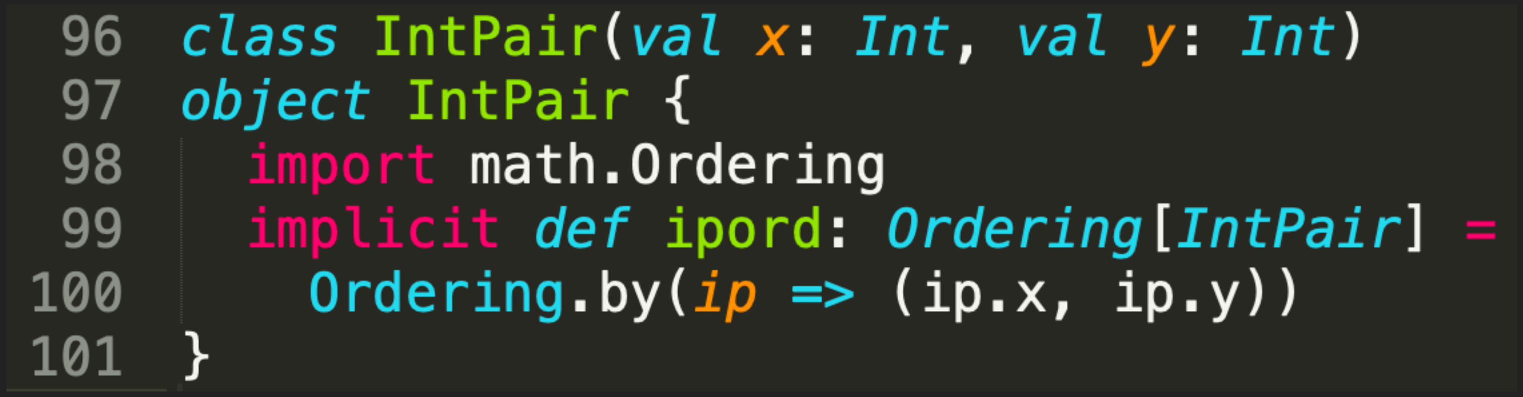
SINGLETON OBJECTS

- Single implementation classes



```scala
object Blah {
  def sum(l: List[Int]): Int = l.sum
}
```

- Companions



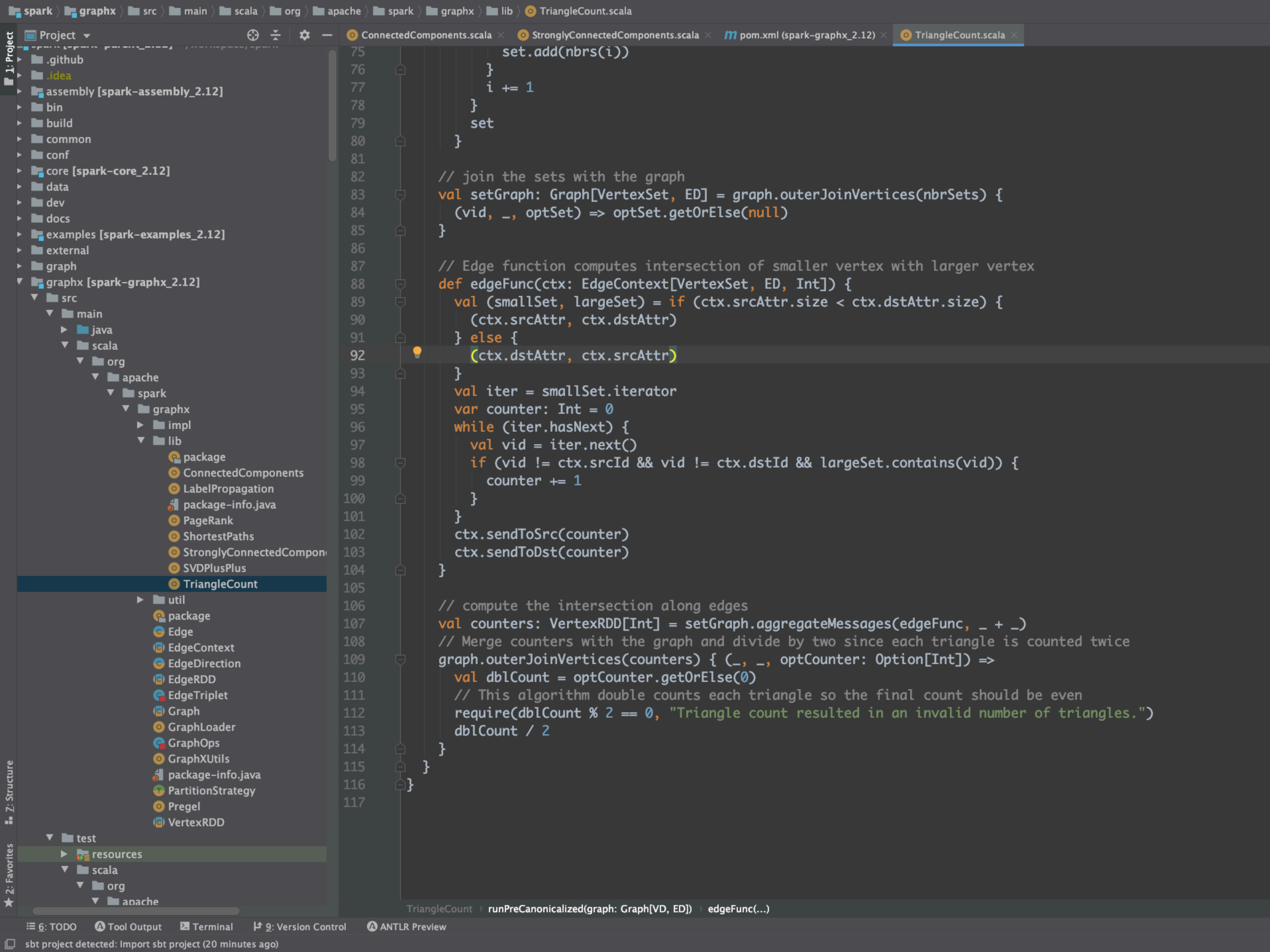
```scala
class IntPair(val x: Int, val y: Int)
object IntPair {
  import math.Ordering
  implicit def ipord: Ordering[IntPair] =
    Ordering.by(ip => (ip.x, ip.y))
}
```

TYPE BOUNDS

- Upper type bound
  - B <: A <==> B is a subtype of A
- Lower type bound
  - B >: A <==> B is a supertype of A

TA COURSE #04
SCALA

PUMP IT!



```scala
            set.add(nbrs(i))
          }
          i += 1
        }
        set
      }

    // join the sets with the graph
    val setGraph: Graph[VertexSet, ED] = graph.outerJoinVertices(nbrSets) {
      (vid, _, optSet) => optSet.getOrElse(null)
    }

    // Edge function computes intersection of smaller vertex with larger vertex
    def edgeFunc(ctx: EdgeContext[VertexSet, ED, Int]) {
      val (smallSet, largeSet) = if (ctx.srcAttr.size < ctx.dstAttr.size) {
        (ctx.srcAttr, ctx.dstAttr)
      } else {
        (ctx.dstAttr, ctx.srcAttr)
      }
      val iter = smallSet.iterator
      var counter: Int = 0
      while (iter.hasNext) {
        val vid = iter.next()
        if (vid != ctx.srcId && vid != ctx.dstId && largeSet.contains(vid)) {
          counter += 1
        }
      }
      ctx.sendToSrc(counter)
      ctx.sendToDst(counter)
    }

    // compute the intersection along edges
    val counters: VertexRDD[Int] = setGraph.aggregateMessages(edgeFunc, _ + _)
    // Merge counters with the graph and divide by two since each triangle is counted twice
    graph.outerJoinVertices(counters) { (_, _, optCounter: Option[Int]) =>
      val dblCount = optCounter.getOrElse(0)
      // This algorithm double counts each triangle so the final count should be even
      require(dblCount % 2 == 0, "Triangle count resulted in an invalid number of triangles.")
      dblCount / 2
    }
  }
}
```

sbt project detected: Import sbt project (20 minutes ago)

THX!