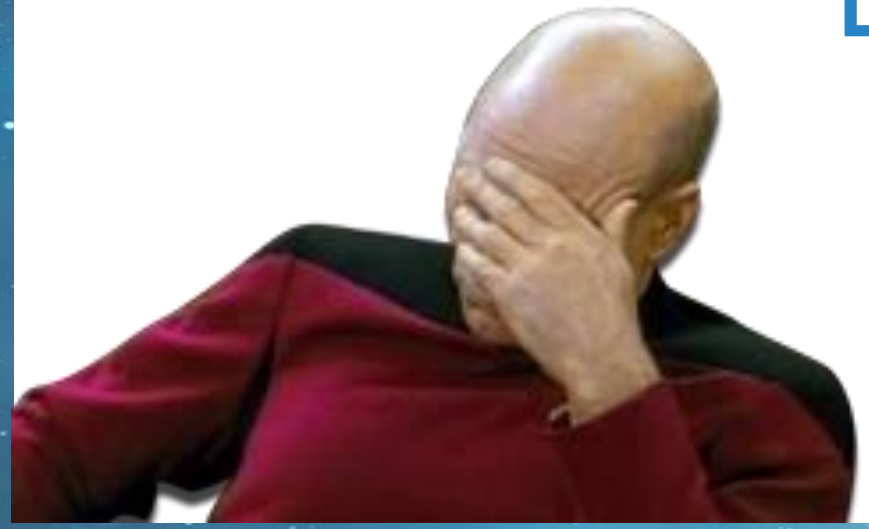

AWESOME SCALA
Boldy go where no Java has gone before!

so.....

We can port Java programs...

But why Scala?



No Boilerplate


```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```


```scala
object Main {
    def main(args: Array[String]) {
        println("Hello World");
          }
}

object Main extends App {
    println("Hello World");
}
```

```scala
object Main extends App {
    println("Hello")
    println("World")
}
```

NO BOILERPLATE!!

For-Comprehensions

```
val books = List(
    Book("How to use the Holodeck", 20),
    Book("How to win Kobayashi Maru", 800)
)
```

```
var ls : List[Int] = List()
for(b <- books) {
    ls = ls :+ b.price
}
```

```
var ls = for(b <- books) yield b.price
```

```
var ls : List[Book] = List()
for(b <- books) {
    if(b.price < 100)
        ls = ls :+ b
}
```

```
var ls = for(b <- books
    if (b.price < 100)) yield b
```

No Boilerplate…

For-Comprehensions

```
val books = List(
    Book("How to use the Holodeck", 20),
    Book("How to win Kobayashi Maru", 800)
)
val stores = List(
    Store("Khan – Books and Revenge", "30000m"),
    Store("Warp Library", "10m")
)
```


```
var ls = for(
    b <- books
    s <- stores
    if (b.price < 100))
yield {
    (b.name, s.name)
}
```

Take Home Points

```
var ls = for(b <- books) yield b.price
```

No Boilerplate

yield: Allow us to map values

No Boilerplate!!

For Comprehensions allow simple Syntax

No return needed – Simply last value returned

```
var ls = for(
    b <- books
    s <- stores
    if (b.price < 100))
yield {
    (b.name, s.name)
}
```

But:
Even if it's ugly…
Java can do the same!

Higher-Order Functions

Lambda Functions

```
val ls1 : List[Int] = List(4,3,2,1)
val ls2 = ls1.sortWith((e1, e2) => e1 < e2)
```

Typesafe

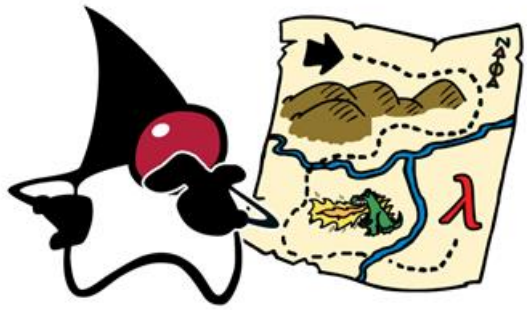
```
(e1, e2) => e1 < e2

(e1 : Int, e2 : Int) => e1 < e2
```

```
(Int, Int) => Boolean
```

But can't we do the same in Java 8?

Well… Java 8 can do Lambda Functions

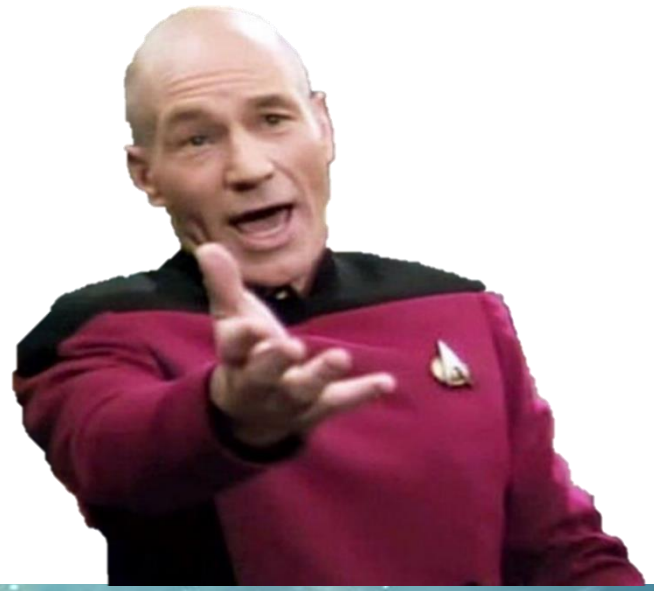
```
(Int e1, Int e2) -> e1 < e2;
```



But Scala can even more!!

Functions = First Class Citizen

```
var f1 = (x : String) => println(x)
f1("Hello World")

def foo(f: String => Unit) {
    f("bar")
}
foo(f1)
```



Take Home Points

Functions are First Class Citizens

```
var f1 = (x : String) => println(x)
```

```
def foo(f: String => Unit) {
    f("bar")
}
```

Treat functions like objects in Java

Scala is typesafe – It can infer types!

Reduce Boilerplate.. again

Pattern Matching

```
case class Book(name : String, price : Int)
```

Case Class are like a normal Class, but…

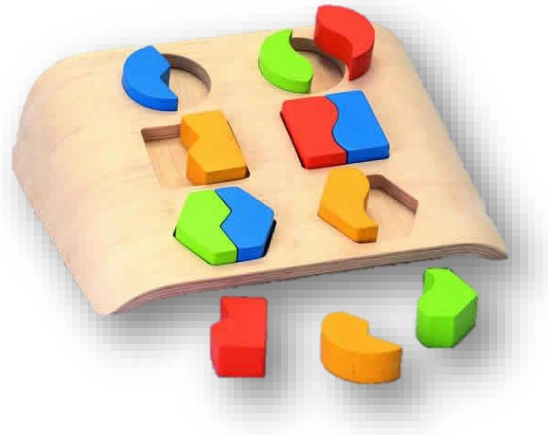
- Provide Apply/Unapply Functions
- Provide equals Function

```
case class Store(name : String, distance : Int)
var anything : Any = Book("How to use the Holodeck", 20)
anything match {
    case b : Book => println(b.name)
    case _ => println("Error")
}
```

wildcard

Pattern Matching

```
var anything : Any = Book("How to use the Holodeck", 20)
anything match {
    case Book(name, 10) => println("Cheap: " + b.name)
    case Book(name, price) if (price > 300) => println(b.name)
    case Book(name, price) => println("Some Book: " + name)
    case _ => println("No Book")
}
```



No more if – else or switch for checking values…

We can just use Scala Pattern Matching!

Pattern Matching

```
try { ... } catch {
    case ioe: IOException => { ... }
    case se: SQLException => { ... }
}
```

```
val (x,y,z) = (1,2,3)
```

Examples

```
val ls : List[String] = List("Foo", "Bar", "Baz")
ls match {
    case Nil => println("This list is empty")
    case e::Nil => println("One Element: " + e)
    case h::t => println("Head Element: " + h + " and some tail")
    case _ => println("no match")
}
```

Take Home Points

You can match nearly anything using **match**

Case Classes are specifically designed for pattern matching

(As they provide **Apply/Unapply** and **equals**)

No More Type-Casting – Use Pattern Matching!

BUT WAIT! .... There is more!

Language Features

Macros

For Comprehensions

Implicits

Domain Specific Languages

Many maaaany more...

Eco System

Use Scala!

To boldly go, where no Java has gone before!