CSE 250
Lecture 1

How to Scala

Textbook Ch. 1.1-1.6, 1.8-1.9

Announcements

- AI Quiz on Autolab available now.
  - Due Weds Sept 7 @ 11:59 PM
  - Submit as many times as you want
  - To pass the class, your final submission must indicate that you have satisfied the requirement (1.0 out of 1.0 score)
  - If you don’t have access to CSE-250 on Autolab, let course staff know.

- PA 0 will be assigned in the next 24 hr

Why Scala?

- Strongly Typed Language
  - The compiler helps you make sure you mean what you say.
- JVM-based, Compiled Language
  - Run anywhere, but also see the impacts of data layout.
- Interactive REPL Interpreter
  - It’s easy to test things out quickly (more on this later).
- Well Thought-Out Container Library
  - Clearly separates data structure role and implementation.

Environment

- IntelliJ
    - Ubuntu Linux
    - MacOS
    - **Windows**
- Emacs + SBT
    - Ubuntu Linux
    - MacOS
    - Windows / WSL

Projects come with an IntelliJ workspace and a SBT build.sbt file

Hello World

Everything is always enclosed in a class

Type

Function definition

```
object HelloWorld {

  def main(args: Array[String]): Unit =
  {
    println("Hello, World!")
  }

}
```

'=' is how you define the function

Brackets in types read as "of" (e.g., "Array of String")

Coding Style is Important

Indentation?

Names?

Useful comments?

Scala features?

Braces?

Return value?

```
def doThings() = {
       val IlikeLlamas = 10
val PeachesAreGreat = for (i <- 1 to 5) yield i

 val QQ = PeachesAreGreat.map(_+ILikeLlamas)

  // This is a for loop.
          for (q <- QQ) println(q)
          // This is a loop with a 4.
  for (i <- 0 until 4) println(i)
  5
}
```

Coding Style is Important

- Indent bracketed code uniformly.
- Give variables <u>semantically</u> meaningful names.
- Use comments to convey the “why” of your code, not the what.
- Scala has MANY ways to express identical concepts. Pick one and be consistent.
- Braces aren’t required, but can help to avoid bugs.
- Clearly indicate return values
- Imagine you’re writing a letter to future-you…
    - ...help future-you (and the TAs/me) understand.

Ways to succeed

- Never start with code.
- What do you have? How is it structured?
  - Draw diagrams
  - Use examples
- What do you want? How should it be structured?
  - Same as above
- How do the components map from one to the other
  - Connect the diagrams
  - Pseudocode: Break the big problem down into smaller ones

Ways to Obtain Assistance

- Explain what you've tried
    - Test cases that fail
    - Approaches that don't work
- Explain what you are trying to accomplish and why
    - Make sure your interlocutor has all the context
- Follow code style guidelines

If you still don’t feel comfortable with Scala

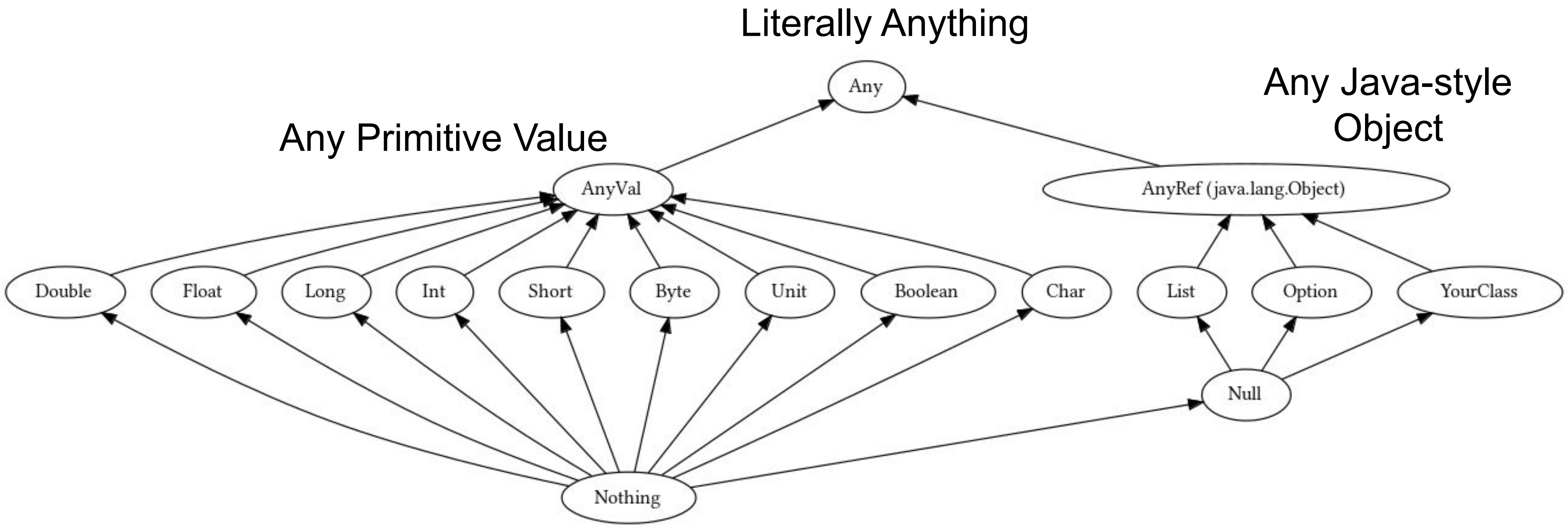
- **Guarantee**: If you bring us (mostly working) pseudocode, the TAs and I will help you translate it to Scala.
- Translation Challenges:
    - Syntax (e.g., “I don’t know how to break out of a for loop”)
        - Ask on Piazza, Office Hours, Recitation; We will help you!
    - Semantics (e.g., “I don’t know how to insert into a linked list”)
        - Ask, but we’ll ask you to be more precise
- Most questions I get about syntax are usually asking about semantics.

Scala

Primitive Types

Type	Description	Examples
`Boolean`	Binary value	`true, false`
`Char`	16-bit unsigned integer	`‘x’, ‘y’`
`Byte`	8-bit signed integer	`42.toByte`
`Short`	16-bit signed integer	`42.toShort`
`Int`	32-bit signed integer	`42`
`Long`	64-bit signed integer	`42l`
`Float`	Single-precision floating-point number	`42.0f`
`Double`	Double-precision floating-point number	`42.0`
`Unit`	No value	`()`

Primitive Types are “sort of” Objects



(image: Scala-Lang Tour, Scala Type Hierarchy https://docs.scala-lang.org/tour/unified-types.html)

Every Expression Has A Type

- Optionally annotate anything with “: type”
  - Variables (declares the variable’s type)
  - Functions (declares the return type)
  - Parenthesized arithmetic (sanity checks the return type)
  - If you don’t annotate, Scala will try to infer it.

```
val x: Float = (5 / 2.0).toFloat

val income = 15 + 10.2 * 9.3f

def lotsOfFun(x: Int) = “fun” * x
```

Why?

Inconsistent Types

```
val res = if (x > 0) { “positive” * x }
          else { -1 }
```

What type does res have?

A: String

B: Int

C: Any

D: AnyRef

Inconsistent Types

```
val res = if (x > 0) { “positive” * x }
          else { -1.toString }
```

Every Block has a Return Value/Type

Don’t forget to include the ‘=’ in a function definition

```
def doThings() = {
  val IlikeLlamas = 10
  val PeachesAreGreat = for (i <- 1 to 5) yield i

  val QQ = PeachesAreGreat.map(_+ILikeLlamas)

  // This is a for loop.
  for (q <- QQ) println(q)
  // This is a loop with a 4.
  for (i <- 0 until 4) println(i)
  5
}
```

What value is returned?

A: 10

B: IlikeLlamas

C: 5

D: 4

The <u>last line</u> of every block is its value

Fall 2022

Blocks for Assignments

Separate multiple instructions on one line with semicolons

```
val blockAssign = { val x = 10; val y = 20; (x, y) }

val butterBlock = {
  val pastry = “croissant”
  val flavor = “PB&J”
  flavor + “ “ + pastry
}
```

Mutable vs Immutable

- Mutable
  - Something that can be changed
- Immutable
  - Something that cannot be changed

`val`	**val**ue that cannot be reassigned (immutable)
`var`	**var**iable that can be reassigned (mutable)

Mutable state can be updated, but is harder to reason about.

Val vs Var

```
scala> val s = mutable.Set(1, 2, 3)

scala> s += 4
res0: s.type = HashSet(1, 2, 3, 4)
```

Why are we allowed to modify s?

Scala Class Types

- `class`
  - Normal OOP type (instantiate with ‘new’)
- `object`
  - A ‘singleton’ class; Only one instance
- `trait`
  - A ‘mixin’ class; Can not be instantiated directly
- `case class`
  - Like class, but provides bonus features

A class can inherit from <u>one</u> superclass and <u>multiple</u> traits

Companion Objects

- An object with the same name as a class (same file)
  - Global (‘static’) methods pertaining to the class
  - e.g., to avoid `‘new’`:

```
class Register(val x : Int) {
  def addValue(y: Int) = x + y
}
object Register {
  def apply(x: Int) = new Register(x)
}
scala> val reg5 = new Register(5)
reg5: Register = Register@146f3d22
scala> val reg10 = Register(10)
reg10: Register = Register@43b172e3
```

Scala shorthand: `foo(x)` **is the same as** `foo.apply(x)`