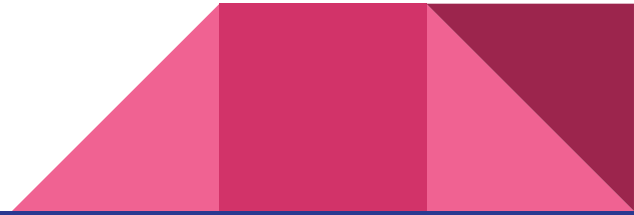
Scala Programming Language

By Group 9: Arash Yazdidoost, Jessica Davis, Tanner Smith



Introduction to Scala

- A high-level statically typed language based on Java. Scala code can even run on the Java Virtual Machine.
- Supports both the object-oriented programming (OOP) and functional programming (FP) paradigms
- Has a variety of applications, such as big data analysis and web applications

Names, Binding, and Scopes

- Names:
  - Scala identifies types, values, methods, and classes using names. These are collectively referred to as entities.
- Binding:
  - Names are introduced via local definitions and declarations, inheritances, imports, or packages.
  - High to low precedence
- Scopes:
  - Scopes in Scala are nested.
  - Bound by local definitions and declarations first, then bound by imports.

Data Types

- Scala does not have any primitive data types (e.g. Char, Int, etc.) instead all data types are objects
- Scala has a type hierarchy, where “Any” is the supertype of all data types and “Nothing” is the subtype of all data types
- Scala supports tuples, arrays, and lists

```
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))
```

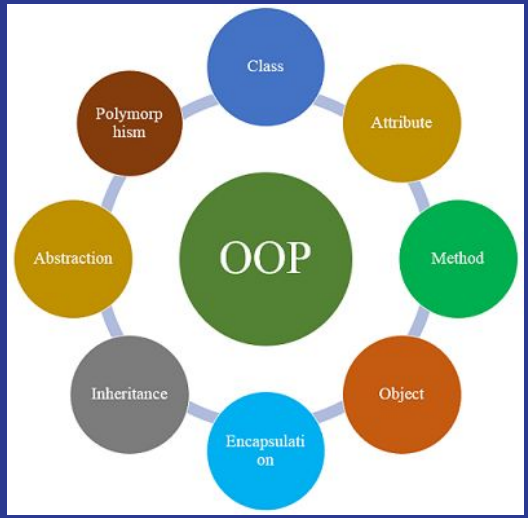
Expressions and Assignment Statements

- All operators are methods in Scala, and any method with a single parameter can be used as an infix operator
- Any legal identifier can be used as an operator, expressions can be created with words such as “and” and “or” as infix operators
- Assignment statements do not require explicit type declarations
- Scala only has two types of variables “val” and “var”

```
def not(x: MyBool) = x.negate
def xor(x: MyBool, y: MyBool) = (x or y) and not(x and y)
```

Support for Object Oriented Programming

- “Pure” Object-Oriented language
- Considered an improved version of Java
- Includes all of the standards of OOP
  - Inheritance
  - Polymorphism
  - Abstraction
  - Encapsulation



Concurrency

- Scala offers
  - Threads (multithreading)
  - Future values
  - Parallel Collections
  - Actor Model

- Functionality and Immutability are key

- Thread Locking a thing of the past

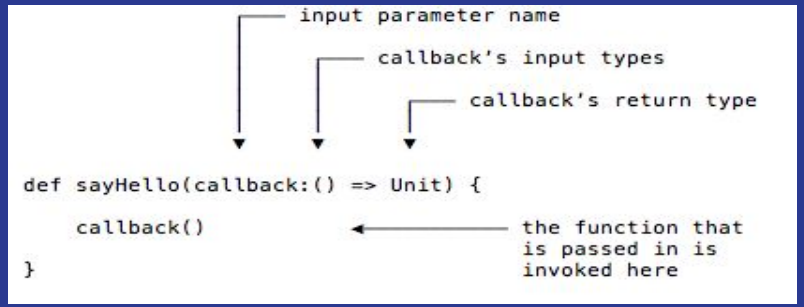
Exception Handling

- Similar to many other popular programming languages
- Try/Catch/Finally model
- Can return in *Options*, *Eithers*, or *Trys*

```
import scala.util.Try

case class FailureReason(msg: String)

object ExceptionsUsingEither extends App {

  //By convention failure values are left in Either
  def divide(a: Int, b: Int): Either[FailureReason, Int] =
    //Try(a / b).toEither
    try {
      Right(a / b)
    } catch {
      case _ => Left(FailureReason("Cannot divide by zero"))
    }

  override def main(args: Array[String]): Unit =
    divide(5, 0).fold(
      error => println("failed"),
      success => println(s"$success")
    )
}
```

Functional Programming

- Scala allows writing code in object-oriented, functional, and a hybrid style.
- Scala collections' classes have a very functional API
- Referential transparency:
  - Being able to use functions as values
- Scala syntax generally makes function signatures easier to read



Functional Error Handling

- No Null values or Exceptions
- *Option*, *Some*, and *None* object types
- Processed with *match* and *for* expressions

```
def toInt(s: String): Option[Int] = {
    try {
        Some(Integer.parseInt(s.trim))
    } catch {
        case e: Exception => None
    }
}
```

```
toInt(x) match {
    case Some(i) => println(i)
    case None => println("That didn't work.")
}

val y = for {
    a <- toInt(stringA)
    b <- toInt(stringB)
    c <- toInt(stringC)
} yield a + b + c
```

Questions?

References

https://builtin.com/software-engineering-perspectives/scala-uses
https://docs.scala-lang.org/overviews/scala-book/prelude-taste-of-scala.html
https://www.scala-lang.org/files/archive/spec/2.11/02-identifiers-names-and-scopes.html
https://docs.scala-lang.org/overviews/scala-book/built-in-types.html
https://docs.scala-lang.org/overviews/scala-book/two-types-variables.html
https://docs.scala-lang.org/tour/unified-types.html