Scala Training 2

Basic Syntax + Type System + Lambdas

Letgo - 10/02/2017

Dani Jimenez
Sergi Gonzalez

Contents

Basic Syntax

Basic Syntax - Functions

PARAMETER NAME

PARAMETER TYPE

```
def find(name: String, age: Int): User = {
  // go to database
  User(name, age)
}
```

RETURN TYPE

FUNCTION NAME

IMPLICIT RETURN STATEMENT

Basic Syntax - Functions

INLINE FUNCTION BODY

```
def find(name: String = "John Doe", age: Int = 18) = User(name, age)
```

DEFAULT PARAMETER VALUE

Basic Syntax - Functions

```
def find(name: String = "John Doe", age: Int = 18) = User(name, age)
```

```
find("Donald Trump", 99) // User("Donald Trump", 99)

find() // User("John Doe", 18)

find(name = "bbbb") // User("bbbb", 18)

find(age = 20) // User("John Doe", 20)

find(age = 80, name = "Mariano Rajoy") // User("Mariano Rajoy", 80)
```

Basic Syntax - Classes

CLASS CONSTRUCTOR

```
class User(name: String, age: Int) {
  def getName: String = name

  def getAge = age
}
```

CLASS TYPE

GETTERS

TYPE INFERENCE

Basic Syntax - Classes

```
case class User(name: String, age: Int)
```

Basic Syntax - Classes

REGULAR CLASS

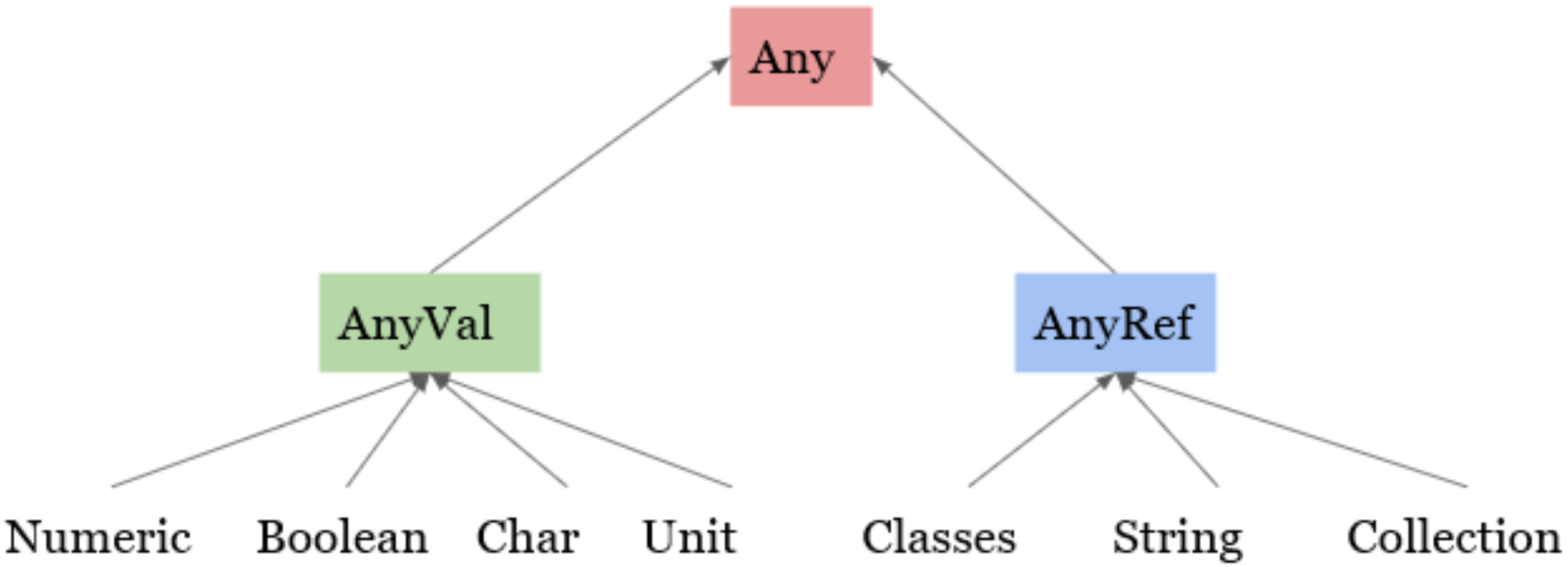
```
val user = new User("Donald Trump", 99)
user.getName // "Donald Trump"
user.getAge // 99
```

CASE CLASS

```
val user = User("Donald Trump", 99)
user.name // "Donald Trump"
user.age
```

Types

Types - Type tree simplified



Types - Basic Types

```
val myInt: Int = 7
val myDouble: Double = 2.0
val myString: String = "John Doe"
val myClass: User = User(myString, myInt)
```

Types - Type inference

```
val myInt = 7
val myDouble = 2.0
val myString = "John Doe"
val myClass = User(myString, myInt)
```

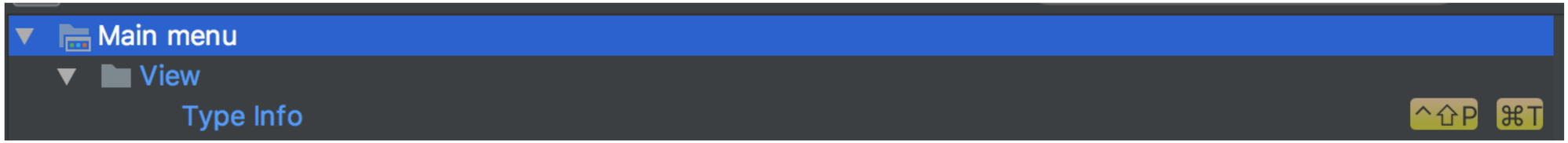
Types - Type safe

```
scala> var myVar = 8
myVar: Int = 8

scala> myVar = "John Doe"
<console>:11: error: type mismatch;
 found   : String("John Doe")
 required: Int
       myVar = "John Doe"
```

Types - Tip

- TIP: configure IntelliJ type info hotkey




```
Future[Seq[Message]]
def fetchWithCache(channelName: ChannelId): Future[Seq[Message]] = cache.getOrElse {
    val message = fetch(channelName)
    cache = Some(message)

    message
}
```

Types - Blocks

```
Int
val myVal = {
    val another = 1
    another
}
```

```
Int
val myVal = {
    val another = 1
    another
}
```

Types - Blocks

Exercise: What are the types of the following blocks?

```
{
    val myString = "john doe"
    myString
}

{
    val myString = "john doe"
}
```

- a) String & String
- b) String & Unit
- c) String & Any
- d) String & Val

Lambdas

Lambdas

```
val myFun: Int => Int = a => a + 1
```

VALUE TYPE

FUNCTION DECLARATION

Lambdas

```
val myFun: Int => Int = a => a + 1
```

```
myFun(3) // 4
```

```
val myList = List(1, 2, 3)
myList.map(a => myFun(a)) // List(2, 3, 4)
myList.map(myFun) // List(2, 3, 4)
```

Lambdas

```
myList.map(a => myFun(a))

myList.map(myFun)
```

Lambdas

```
val myList = List(1, 2, 3)
myList.map(_ => "hello") // List("hello", "hello", "hello")
```

Workshop

Workshop

https://github.com/letgoapp/scala_course/blob/3-syntax-types-lambdas/doc/lessons/3-basic-syntax-types.md

https://github.com/letgoapp/scala_course/blob/3-syntax-types-lambdas/doc/lessons/4-anonymous-functions.md

Workshop

```
"MessageCensorTest" should {
  "censor bad language" in {
    Given("a MessageCensorTest")

    val forbiddenKeywords = Set("fuck", "jorge")
    val censor = new MessageCensor(forbiddenKeywords)

    val myMessages = Seq(
      Message("go fuck yourself"),
      Message("i love jorge")
    )

    val expectedCensoredMessages = Seq(
      Message("go yourself"),
      Message("i love")
    )

    val censoredMessages = censor.filterMessages(myMessages)

    censoredMessages should be(expectedCensoredMessages)
  }
}
```

Workshop

```
class MessageCensor(forbiddenKeywords: Set[String]) {

  private val filterRule: Message => Message = ???

  def filterMessages(messages: Seq[Message]): Seq[Message] = messages.map(filterRule)
}
```