Функции и все о них

Продолжение

BINARY DISTRICT

Стратегии вычисления функций

- Call by value
- Call by name

Call by value

```
def randomJoke(): Int = {
  println("chosen by fair dice roll. guaranteed to be random")
  1
}

def callByValue(x: Int) = {
  println("callByValue")
  println(s"x1 = $x")
  println(s"x2 = $x")
}

callByValue(randomJoke())
```

```
chosen by fair dice roll. guaranteed to be random
callByValue
x1 = 1
x2 = 1
```

Call by name

```scala
def randomJoke(): Int = {
  println("chosen by fair dice roll. guaranteed to be random")
  1
}

def callByName(x: => Int) = {
  println("callByName")
  println(s"x1 = $x")
  println(s"x2 = $x")
}

callByName(randomJoke())
```

```
callByName
chosen by fair dice roll. guaranteed to be random
x1 = 1
chosen by fair dice roll. guaranteed to be random
x2 = 1
```

Lazy val

```
lazy val soooLazy = {
 println("laaaaazy")
 1
}
println("start")
println(soooLazy)
println(soooLazy)
```

```
start
laaaaazy
1
1
```

Lazy val

```
lazy val soooLazy = {
 println("laaaaazy")
 1
}
println("start")
// println(soooLazy)
// println(soooLazy)
```

start

Классы типов (Обобщения/generics/параметрический полиморфизм/полиморфные типы)

```
class IntContainer(t: Int) {
  override def toString: String = t.toString
}
val s = new IntContainer(1)
println(s)
```

```
1
```

Классы типов (Обобщения/generics/параметрический полиморфизм/полиморфные типы)

```
class Container[T](t: T) {
  override def toString: String = t.toString
}
val s = new Container[Int](1)
println(s)
val s2 = new Container[String]("1!")
println(s2)
```

```
1
1!
```

Отношения полиморфных типов

Если T’ является подклассом T, является ли C[T’] подклассом C[T]? Разница в замечаниях позволяет выразить следующие отношения между иерархией классов и полиморфными типами:

	Означает	**Нотация языка Scala**
ковариант	C[T’] это подкласс класса C[T]	[+T]
контрвариант	C[T] это подкласс класса C[T’]	[-T]
инвариант	C[T] и C[T’] не взаимосвязаны	[T]

Классы типов (Обобщения/generics/параметрический полиморфизм/полиморфные типы)

```scala
trait Animal { def sound: String }
class Dog extends Animal { override val sound = "wuf" }
class Cat extends Animal { override val sound = "meow" }
class Container[T](t: T) {
  override def toString: String = t.toString
}
val s: Container[Animal] = new Container[Dog](new Dog)
println(s)
val s2: Container[Animal] = new Container[Animal](new Dog)
println(s2)
```

```
Error:(9, 30) type mismatch;
 found   :
wtf.scala.lectures.e03.Types2.Container[wtf.scala.lectures.e03.Dog]
 required:
wtf.scala.lectures.e03.Types2.Container[wtf.scala.lectures.e03.Animal]
Note: wtf.scala.lectures.e03.Dog <: wtf.scala.lectures.e03.Animal, but
class Container is invariant in type T.
You may wish to define T as +T instead. (SLS 4.5)
  val s: Container[Animal] = new Container[Dog](new Dog)
```

Классы типов (Обобщения/generics/параметрический полиморфизм/полиморфные типы)

```
// covariant
class Container[+T](t: T) {
  override def toString: String = t.toString
}
val s: Container[Animal] = new Container[Dog](new Dog)
println(s)
val s2: Container[Animal] = new Container[Animal](new Dog)
println(s2)
```

Классы типов (Обобщения/generics/параметрический полиморфизм/полиморфные типы)

```
// contravariant
class Container[-T](t: T) {
  override def toString: String = t.toString
}
val s: Container[Dog] = new Container[Animal](new Dog)
println(s)
val s2: Container[Animal] = new Container[Dog](new Dog)
println(s2)
```

```
Error:(10, 31) type mismatch;
 found   :
wtf.scala.lectures.e03.Types4.Container[wtf.scala.lectures.e03.Dog]
 required:
wtf.scala.lectures.e03.Types4.Container[wtf.scala.lectures.e03.Animal]
  val s2: Container[Animal] = new Container[Dog](new Dog)
```

Классы типов (Обобщения/generics/параметрический полиморфизм/полиморфные типы)

```
trait Function1 [-T1, +R] extends AnyRef
```

```
val getCatSound: (Cat => String) = (a: Animal) => a.sound
val getAnimalSound: (Animal => String) = ((a: Cat) => a.sound)
val catBirth: (() => Animal) = () => new Cat
val birth: (() => Animal) = () => new AnyRef
```

```
Error:(5, 54) type mismatch;
 found   : wtf.scala.lectures.e03.Cat => String
 required: wtf.scala.lectures.e03.Animal => String
  val getAnimalSound: (Animal => String) = ((a: Cat) => a.sound)
```

```
Error:(7, 32) type mismatch;
 found   : wtf.scala.lectures.e03.Animal
 required: wtf.scala.lectures.e03.Cat
val birth: (() => Cat) = () => new Animal
```

Ограничения типов. Верхняя граница типов

```
def cacophony[T](things: Seq[T]) = things map (_.sound)
```

```
def biophony[T <: Animal](things: Seq[T]) = things map (_.sound)
```

```
def biophony(things: Seq[_ <: Animal]) = things map (_.sound)
```

Ограничения типов. Нижняя граница типов

```
class Node[T](x: T) { def sub(v: T): Node[T] = new Node(v) }
=>
class Node[+T](x: T) { def sub(v: T): Node[T] = new Node(v) }
```

```
covariant type T occurs in
contravariant position in type T of
value v
```

```
class Node[+T](x: T) { def sub[U >: T](v: U): Node[U] = new Node(v) }
```

```
(new Node(new Cat)).sub(new Animal)
```

Implicits

```
case class MyString(s: String) {
  def whose = s"I'm yours :] $s"
}
implicit def strToMyString(x: String): MyString = MyString(x)
println("heh".whose)
```

I'm yours :] heh

Implicits. Преобразования

```scala
case class MyString(s: String) {
  def whose = s"I'm yours :] $s"
}
implicit def strToMyString(x: String): MyString = MyString(x)
val mine: MyString = "!11"
println("heh".whose)
```

I'm yours :] !11

Implicits. Параметры

```scala
case class MyString(s: String) {
  def whose = s"I'm yours :] $s"
}
implicit def smtn(implicit x: MyString): Unit =
println(x.whose)
implicit val mine: MyString = MyString("!11")
smtn
```

I'm yours :] !11

Где ищем Implicit?

1. First look in current scope
   - Implicits defined in current scope
   - Explicit imports
   - wildcard imports
2. Now look at associated types in
   - Companion objects of a type
   - Implicit scope of an argument’s type (2.9.1)
   - Implicit scope of type arguments (2.8.0)
   - Outer objects for nested types

Где ищем Implicit?

```
List(1, 2, 3).sorted
```

```
def sorted[B >: A](implicit ord: Ordering[B]): ......
```

```
Ordering.Int
```

Где ищем Implicit?

```scala
class A(val n: Int)
object A {
    implicit def str(a: A) = s"A: ${a.n}"
}
class B(val x: Int, y: Int) extends A(y)
val b = new B(5, 2)
val s: String = b  // s == "A: 2"
```

View Bound

```
def f[A <% B](a: A) = a.bMethod
```

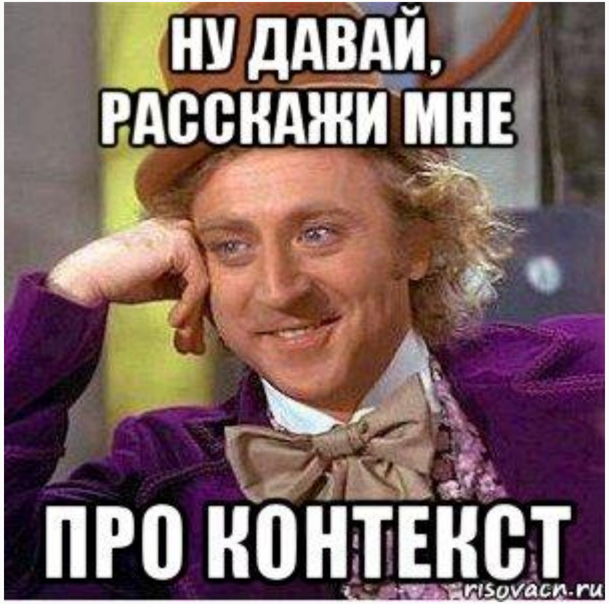
```
def f[A <% Ordered[A]](a: A, b: A) = if (a < b) a else b
```

```
def f[A](a: A)(implicit ev: A => B) = a.bMethod
```

PS DEPRECATED

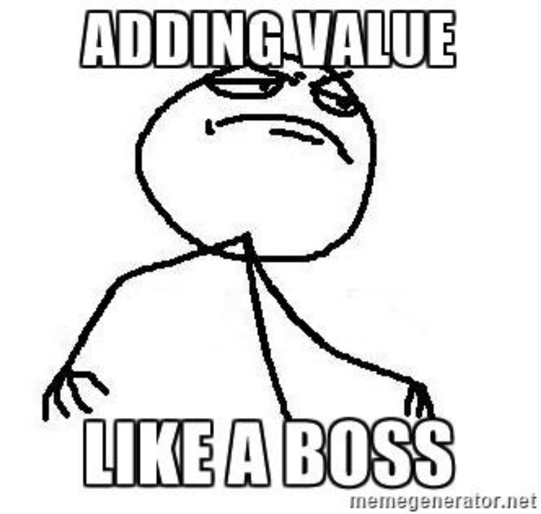
Context Bound

```
def g[A : B](a: A) = h(a)

def g[A](a: A)(implicit ev: B[A]) = h(a)

def f[A : Ordering](a: A, b: A) = if
(implicitly[Ordering[A]].lt(a, b)) a else b
```



Value class

```
class Wrapper(val underlying: Int) extends AnyVal {
  def foo: Wrapper = new Wrapper(underlying * 19)
}
```



Extension methods

```
implicit class RichInt(val self: Int) extends AnyVal {
  def toFunny: String = s"$self :P"
}
println(1.toFunny)
```

Проверка корректности

```
class Meter(val value: Double) extends AnyVal {
  def +(m: Meter): Meter = new Meter(value + m.value)
}

val x = new Meter(3.4)
val y = new Meter(4.3)
val z = x + y
val zz = x + 3.1
```

Implicits. Когда что-то идет не так

- IntelliJ Idea
- scalac -Xprint:typer

https://github.com/ljwagerfield/debugging-scala-implicits-in-intellij

Материалы

https://twitter.github.io/scala_school/ru/type-basics.html

https://twitter.github.io/scala_school/ru/advanced-types.html

https://github.com/anton-k/ru-neophyte-guide-to-scala

https://habrahabr.ru/post/318960/

http://virtuslab.com/blog/debugging-implicits/

https://www.scala-exercises.org/scala_tutorial/

https://www.scala-exercises.org/std_lib/

Фазы компиляции, полезные флаги компилятора или как Scala это делает?!

```
scalac -help
```

```
Usage: scalac <options> <source files>
where possible standard options include:
  -Dproperty=value                    Pass -Dproperty=value directly to the
runtime system.
  -J<flag>                            Pass <flag> directly to the runtime system.
  -P:<plugin>:<opt>                   Pass an option to a plugin
  -X                                  Print a synopsis of advanced options.
  -bootclasspath <path>               Override location of bootstrap class
files.
  -classpath <path>                   Specify where to find user class files.
  -d <directory|jar>                  destination for generated classfiles.
  -dependencyfile <file>              Set dependency tracking file.
  -deprecation                        Emit warning and location for usages of
deprecated APIs.
  -encoding <encoding>                Specify character encoding used by
source files.
  -explaintypes                       Explain type errors in more detail.
  -extdirs <path>                     Override location of installed extensions.
  -feature                            Emit warning and location for usages of
features that should be imported explicitly.
  -g:<level>                          Set level of generated debugging info.
...
```

Фазы компиляции, полезные флаги компилятора или как Scala это делает?!

```
scalac -Xshow-phases
```

```
    phase name  id  description
    ----------  --  -----------
        parser   1  parse source into ASTs, perform simple desugaring
         namer   2  resolve names, attach symbols to named trees
packageobjects   3  load package objects
         typer   4  the meat and potatoes: type the trees
        patmat   5  translate match expressions
superaccessors   6  add super accessors in traits and nested classes
    extmethods   7  add extension methods for inline classes
       pickler   8  serialize symbol tables
     refchecks   9  reference/override checking, translate nested objects
       uncurry  10  uncurry, translate function values to anonymous classes
        fields  11  synthesize accessors and fields, add bitmaps for lazy vals
     tailcalls  12  replace tail calls by jumps
    specialize  13  @specialized-driven class and method specialization
  explicitouter  14  this refs to outer pointers
       erasure  15  erase types, add interfaces for traits
   posterasure  16  clean up erased inline classes
     lambdalift  17  move nested functions to top level
  constructors  18  move field definitions into constructors
       flatten  19  eliminate inner classes
         mixin  20  mixin composition
       cleanup  21  platform-specific cleanups, generate reflective calls
    delambdafy  22  remove lambdas
           jvm  23  generate JVM bytecode
      terminal  24  the last phase during a compilation run
```

Фазы компиляции, полезные флаги компилятора или как Scala это делает?!

```
scalac -Xprint:typer Lazy.scala
```

```
[[syntax trees at end of                     typer]] // Lazy.scala
package wtf.scala.lectures.e03 {
  object Lazy extends AnyRef with App {
    def <init>(): wtf.scala.lectures.e03.Lazy.type = {
      Lazy.super.<init>();
      ()
    };
    <stable> <accessor> lazy val soooLazy: Int = {
      scala.Predef.println("laaaaazy");
      1
    };
    scala.Predef.println("start")
  }
}
```

Фазы компиляции, полезные флаги компилятора или как Scala это делает?!

```
scalac -print Lazy.scala
```

```
[[syntax trees at end of                  cleanup]] // Lazy.scala
package wtf.scala.lectures.e03 {
  object Lazy extends Object with App {
    @deprecatedOverriding("args should not be overridden", "2.11.0")
protected def args(): Array[String] = Lazy.super.args();
    @deprecated("the delayedInit mechanism will disappear", "2.11.0")
override def delayedInit(body: Function0): Unit =
Lazy.super.delayedInit(body);
    @deprecatedOverriding("main should not be overridden", "2.11.0")
def main(args: Array[String]): Unit = Lazy.super.main(args);
...
```