Implicits

Mikhail Mutcianko, Alexey Otts
СПБгУ, СП

Implicits

```
class Context

def foo(i: Int, context: Context): String = ???
def bar(s: String, context: Context): String = ???

def baz(i: Int, context: Context): String = bar(foo(i, context), context)

val context = new Context
baz(1, context)
```

Implicits

implicit arguments

```
class Context

def foo(i: Int)(implicit context: Context): String = ???
def bar(s: String)(implicit context: Context): String = ???

def baz(i: Int)(implicit context: Context): String = bar(foo(i))

val context = new Context
baz(1)(context)
```

Implicits

implicit arguments

```
class Context

def foo(i: Int)(implicit context: Context): String = ???
def bar(s: String)(implicit context: Context): String = ???

def baz(i: Int)(implicit context: Context): String = bar(foo(i))

implicit val context = new Context
baz(1)
```

Implicits

implicit arguments

```
class Context

def foo(i: Int)(implicit context: Context): String = ???
def bar(s: String)(implicit context: Context): String = ???

def baz(i: Int)(implicit context: Context): String = bar(foo(i))

implicit def context = new Context
baz(1)
```

Implicits

implicit conversions

```
class A(i: Int) {
 def foo() = println(i)
}

implicit def intToA(i: Int): A = new A(i)

1.foo()
```

Implicits

implicit conversions

```
class A(val n: Int)
class B(val m: Int, val n: Int)
class C(val m: Int, val n: Int, val o: Int) {
  def total = m + n + o
}

implicit def toA(n: Int): A = new A(n)
implicit def aToB(a: A): B = new B(a.n, a.n)
implicit def bToC(b: B): C = new C(b.m, b.n, b.m + b.n)
// не скомпилируется
println(5.total)
println(new A(5).total)
println(new B(5, 5).total)
println(new C(5, 5, 10).total)
```

Implicits

implicit conversions: view bounds

```
class A(val n: Int)
class B(val m: Int, val n: Int)
class C(val m: Int, val n: Int, val o: Int) {
  def total = m + n + o
}

implicit def toA(n: Int): A = new A(n)
implicit def aToB[A1](a: A1)(implicit f: A1 => A): B = new B(a.n, a.n)
implicit def bToC[B1](b: B1)(implicit f: B1 => B): C = new C(b.m, b.n, b.m + b.n)
// скомпилируется
println(5.total)
println(new A(5).total)
println(new B(5, 5).total)
println(new C(5, 5, 10).total)
```

Implicits

implicit conversions: bad practices

```
def someBusinessLogic(name: String, email: String, phone: String): User = {
 User(name, email, phone)
}
```

Implicits

implicit conversions: bad practices

```
implicit def anyToOption[A](a: A): Option[A] = Some(a)
implicit def anyToSeq[A](a: A): Seq[A] = Seq(a)


def someBusinessLogic(name: String, email: String, phone: String): User = {
 User(name, email, phone)
}


case class User(name: String, email: Option[String], phones: Seq[String])
```

Implicits

implicit conversions: good practices

```
val route = {
 parameters("color", 'count.as[Int]) { (color, count) =>
   complete(s"The color is '$color' and you have $count of it.")
 }
}
```

Implicits

implicit class and extension functions

```
implicit class IntOps(i: Int) {
 def factorial: Int = ???
}

5.factorial
```

Implicits

implicit class and extension functions

```
implicit class IntOps(i: Int) {
 def factorial: Int = ???
}

new IntOps(5).factorial
```

Implicits

implicit class and extension functions

```
implicit class IntOps(val i: Int) extends AnyVal {
 def factorial: Int = ???
}

5.factorial
IntOps.factorial(5)
```

Implicits

implicit class and extension functions

```
object SomeObject {
 implicit class IntOps(val i: Int) extends AnyVal {
   def factorial: Int = ???
 }
}

imprort SomeObject._
5.factorial
```

Implicits

Type classes

```
trait Show[A] {
 def show(a: A): String
}

implicit val showInt: Show[Int] = new Show[Int] {
 def show(a: Int): String = a.toString
}

def stringify[A](a: A)(implicit s: Show[A]): String = s.show(a)

stringify(1)
```

Implicits

Порядок поиска implicit’ов: 1) текущая область

```
trait Show[A] {
 def show(a: A): String
}

implicit val showInt: Show[Int] = new Show[Int] {
 def show(a: Int): String = a.toString
}

def stringify[A](a: A)(implicit s: Show[A]): String = s.show(a)

stringify(1)
```

Implicits

Порядок поиска implicit’ов: 2) конкретные импорты

```
trait Show[A] {
 def show(a: A): String
}

def stringify[A](a: A)(implicit s: Show[A]): String = s.show(a)

import Show.instances.intShow
stringify(1)
```

Implicits

Порядок поиска implicit’ов: 3) wildcard импорты

```
trait Show[A] {
 def show(a: A): String
}

def stringify[A](a: A)(implicit s: Show[A]): String = s.show(a)

import Show.instances._
stringify(1)
```

Implicits

Порядок поиска implicit’ов: 4) компаньон класса

```
trait Show[A] {
 def show(a: A): String
}
case class Foo(a: Int)
object Foo {
 implicit val fooShow: Show[Foo] = new Show[Foo] {
   def show(a: Foo): String = s"Foo(a = ${a.a})"
 }
}

def stringify[A](a: A)(implicit s: Show[A]): String = s.show(a)

stringify(Foo(1))
```

Implicits

Порядок поиска implicit’ов: 5) компаньон тайп класса

```
trait Show[A] {
 def show(a: A): String
}
object Show {
 implicit def showOpt[A](implicit s: Show[A]): Show[Option[A]] = new Show[Option[A]] {
   def show(a: Option[A]): String = a.fold("None")(v => s"Some(${s.show(v)}")
 }
}

def stringify[A](a: A)(implicit s: Show[A]): String = s.show(a)

stringify(Option(Foo(1)))
```

Implicits

Context bound

```
def stringify[A: Show](a: A): String = implicitly[Show[A]].show(a)

stringify(1)
```

Implicits

Context bound

```
def stringify[A: Show](a: A): String = Show[A].show(a)

object Show {
 def apply[A](implicit s: Show[A]): Show[A] = s
}

stringify(1)
```

Implicits

Context bound + implicit class

```
object Show {
 object syntax {
   implicit class ShowSyntax[A](val a: A) extends AnyVal {
     def stringify(implicit s: Show[A]): String = s.show(a)
   }
 }
}

import Show.syntax._
def stringify[A: Show](a: A): String = a.stringify

stringify(1)
1.stringify
```