- Scala
- Scala

- Scala & Scalaz
  FP in Scala

Scala Scala Odersky
Scala Scala
Scala 2.8

Scala

Scala & Scalaz[*1] Paul Chiusano
Rúnar Bjarnason
Scala Scalaz Scala

List Stream Future Scala

Scala

-

[*1]

- Java C 1
-
- Twitter API

1

- must
- should
- may

Image File doesn't exist

1

Scala

Scala

Scala 2003 EPFL Martin Odersky
Scala
JVM Java
Scala better Java
Scala Scala

Scala

Scala Scala

Scala Java

*1

Scala

*1 first-class

Scala

Scala case class Value Object

implicit parameter
Haskell

Scala

1

Scala

- case class
-
- implicit parameter
- for
-

Scala

Scala immutability
var val mutable immutable
*2 case class immutable

Scala

*2 http://docs.scala-lang.org/ja/overviews/collections/overview.html

Scala

- trait mixin
-
- variance
- self type annotation
- implicit class conversion
- `private[this]`
- Java

Scala Scala
trait mixin

Java

Scala Java Scala Java
Java Scala Java

Scala Java Java
class javap

JVM
JVM

Scala Future

Scala

Scala Akka Akka

Akka
Akka

sbt

Scala Scala [3]
sbt Simple Build Tool
sbt

Mac OS

Mac OS Homebrew

```
$ brew install sbt
```

Windows

chocolatey chocolatey Windows
chocolatey sbt

```
> choco install sbt
```

Windows sbt

Windows/Linux sbt
sbt sbt PATH
Windows Rapid Environment Editor

[3] “ ” `scalac`

REPL sbt

REPL Read Eval Print Loop Scala

```
sbt console
```

sbt console Windows Mac

```
$ sbt console
```

OK

```
[info] Loading global plugins from /Users/.../.sbt/0.13/plugins
[info] Set current project to sandbox (in build file:/Users/.../sandbox/)
[info] Updating {file:/Users/.../sandbox/}sandbox...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Starting scala interpreter...
[info]
Welcome to Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_45).
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

sbt console

```
scala> :quit
```

sbt console

sbt _target_

Scala REPL sbt

Scala REPL

build.sbt

```
scalaVersion := "2.11.8"
```

*.sbt sbt REPL

.sbt

sbt

“sbt ”

sbt `sbt --version` version [*4] sbt 0.13[*5]
0.13 0.13.6 0.13.7
sbt

0.12 0.12.4 0.12
0.13

Scala

Scala REPL
Mac OS Mac OS
Windows

Scala

sbt sbt
Scala

REPL Scala

Scala REPL Read Eval Print
Loop Scala REPL
Scala

Windows

```
$ sbt console
```

[*4] 1 2
[*5] 2016 02 0.13.11

```
Welcome to Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_45).
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Scala $

[*6]

Hello, World!

Hello, World!

```
scala> println("Hello, World!")
Hello, World!
```

Hello, World! println()

```
scala> "Hello, World!"
res1: String = Hello, World!
```

res1: String = Hello, World "Hello, World"

String "Hello, World"

Scala

C Java

REPL

- 0xff
- 1e308
- 9223372036854775807L
- 9223372036854775808L
- 922337203685477580 7

[*6] Windows $ >

- 922337203685477580.7
- 1.00000000000000000001 == 1
- "\u3042"
- "\ud842\udf9f"

```
scala> 1 + 2
res1: Int = 3
```

3 Int REPL resN
REPL REPL

```
scala> res1
res2: Int = 3
```

REPL Int
+,-,*,/

```
scala> 1 + 2
res2: Int = 3

scala> 2 * 2
res3: Int = 4

scala> 4 / 2
res4: Int = 2

scala> 4 % 3
res5: Int = 1
```

Double Int
dbl.asInstanceOf[Int]

```
scala> 1.0 + 2.0
res6: Double = 3.0
```

```
scala> 2.2 * 2
res7: Double = 4.4

scala> 4.5 / 2
res8: Double = 2.25

scala> 4 % 3
res9: Int = 1

scala> 4.0 % 2.0
res10: Double = 0.0

scala> 4.0 % 3.0
res11: Double = 1.0
```

+,-,*,/,%

- 2147483647 + 1
- 9223372036854775807L + 1
- 1e308 + 1
- 1 + 0.0000000000000000001
- 1 - 1
- 1 - 0.1
- 0.1 - 1
- 0.1 - 0.1
- 0.0000000000000000001 - 1
- 0.1 * 0.1
- 20 * 0.1
- 1 / 3
- 1.0 / 3
- 1 / 3.0
- 3.0 / 3.0
- 1.0 / 10 * 1 / 10
- 1 / 10 * 1 / 10.0

REPL

Scala C Java

Scala `val` `var` 2

C Java `val`

Java final

Scala `var` `val`

Scala

```
scala> val x = 3 * 2
x: Int = 6
```

x `3 * 2` C Java

x `3 * 2` `Int` x `Int`

Scala

`Int` `String`

`val` `var`

`var` `val`

```
scala> var x = 3 * 3
x: Int = 9

scala> x = "Hello, World!"
<console>:8: error: type mismatch;
 found   : String("Hello, World!")
 required: Int
       x = "Hello, World!"
           ^

scala> x = 3 * 4
x: Int = 12
```

- `var` `3 * 3` `Int`
- `var` `9` `12`

`val` `var` `:` `=`

```
scala> val x: Int = 3 * 3
x: Int = 9
```

- Q. ¥3,950,000 2.3 8

A. ¥60,566

- Q. ¥1,980,000 1.6 ¥26,400

A. 18

2 Scala

sbt

REPL Scala Scala
sbt REPL Hello,
World! REPL REPL
REPL [*7]

[*7] `:quit` `:q`

```
>:quit
```

Scala 2.10 `exit` Scala 2.11 `sys.exit` REPL
REPL

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

`object` `def`
`scalac`
`{}` REPL `println("Hello, World!")`
HelloWorld.scala

sbt _sandbox_

`sandbox`
`HelloWorld.scala`
`build.sbt`

build.sbt Scala `scalac`

```scala
// build.sbt
scalaVersion := "2.11.8"

scalacOptions ++= Seq("-deprecation", "-feature", "-unchecked", "-Xlint")
```

`scalac`

- API `-deprecation`
- `-feature`
- `-unchecked`
- `-Xlint`

sandbox sbt

sbt sbt HelloWorld

run

```
> run
[info] Compiling 1 Scala source to ...
[info] Running HelloWorld
Hello, World!
[success] Total time: 1 s, completed 2015/02/09 15:44:44
```

HelloWorld Hello, World!

run main

sbt Scala console REPL

HelloWorld.scala _User.scala_

```
// User.scala
class User(val name: String, val age: Int)

object User {
  def printUser(user: User) = println(user.name + " " + user.age)
}
```

User.scala User User User User

printUser

sandbox

HelloWorld.scala

User.scala

build.sbt

sbt console REPL REPL User User

```
scala> val u = new User("dwango", 13)
u: User = User@20daebd4

scala> User.printUser(u)
dwango 13
```

sbt

IDE (Integrated Development Environment)

Scala Emacs, Vim, Sublime Text
IDE Scala IDE IntelliJ IDEA
+ Scala Plugin Scala IDE for Eclipse 2 IntelliJ IDEA IDE
IntelliJ IDEA + Scala Plugin

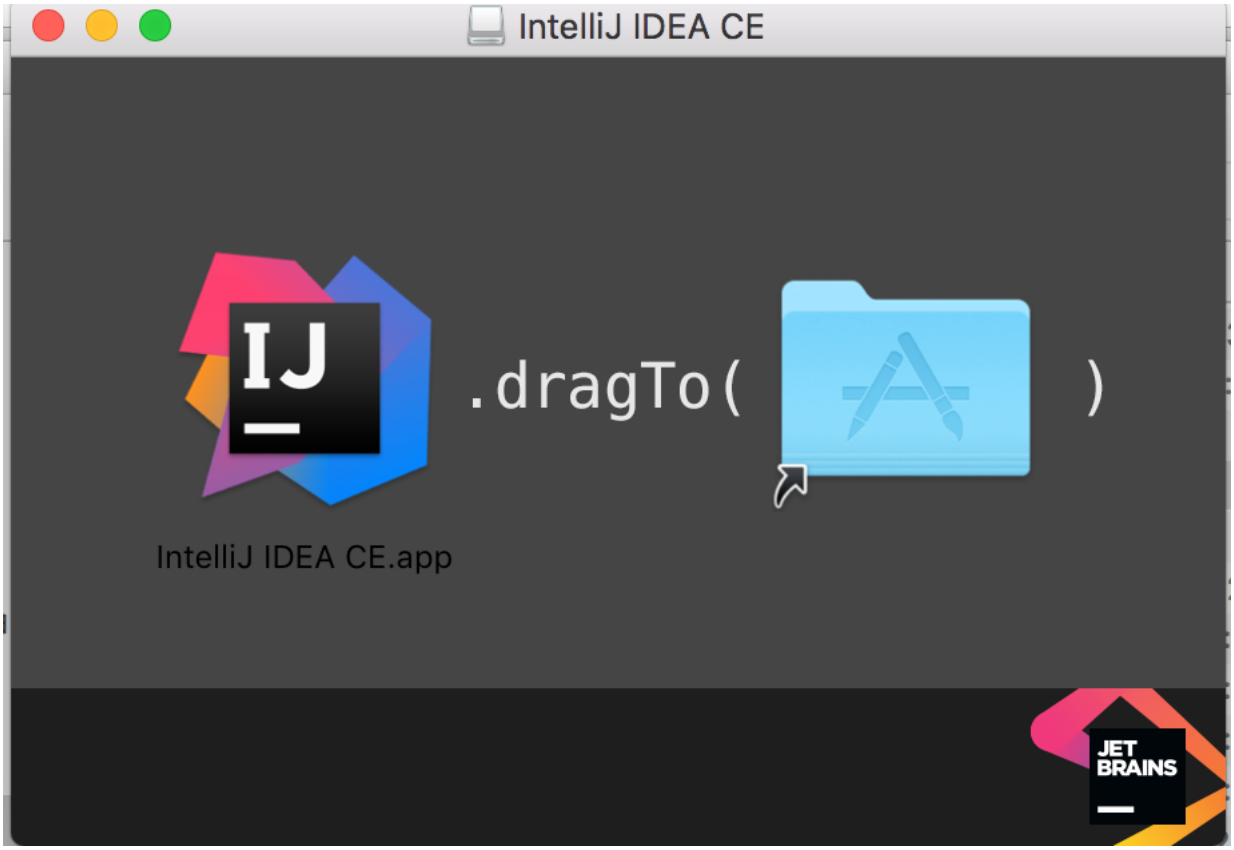
IDE **REPL** IDE
IDE

IntelliJ IDEA JetBrains IDE Java IDE
Ultimate Edition Community Edition
Community Edition Scala Scala
Community Edition

IntelliJ IDEA Download Windows, Mac OS X, Linux 3
OS Download Community
IDEA Mac OS X

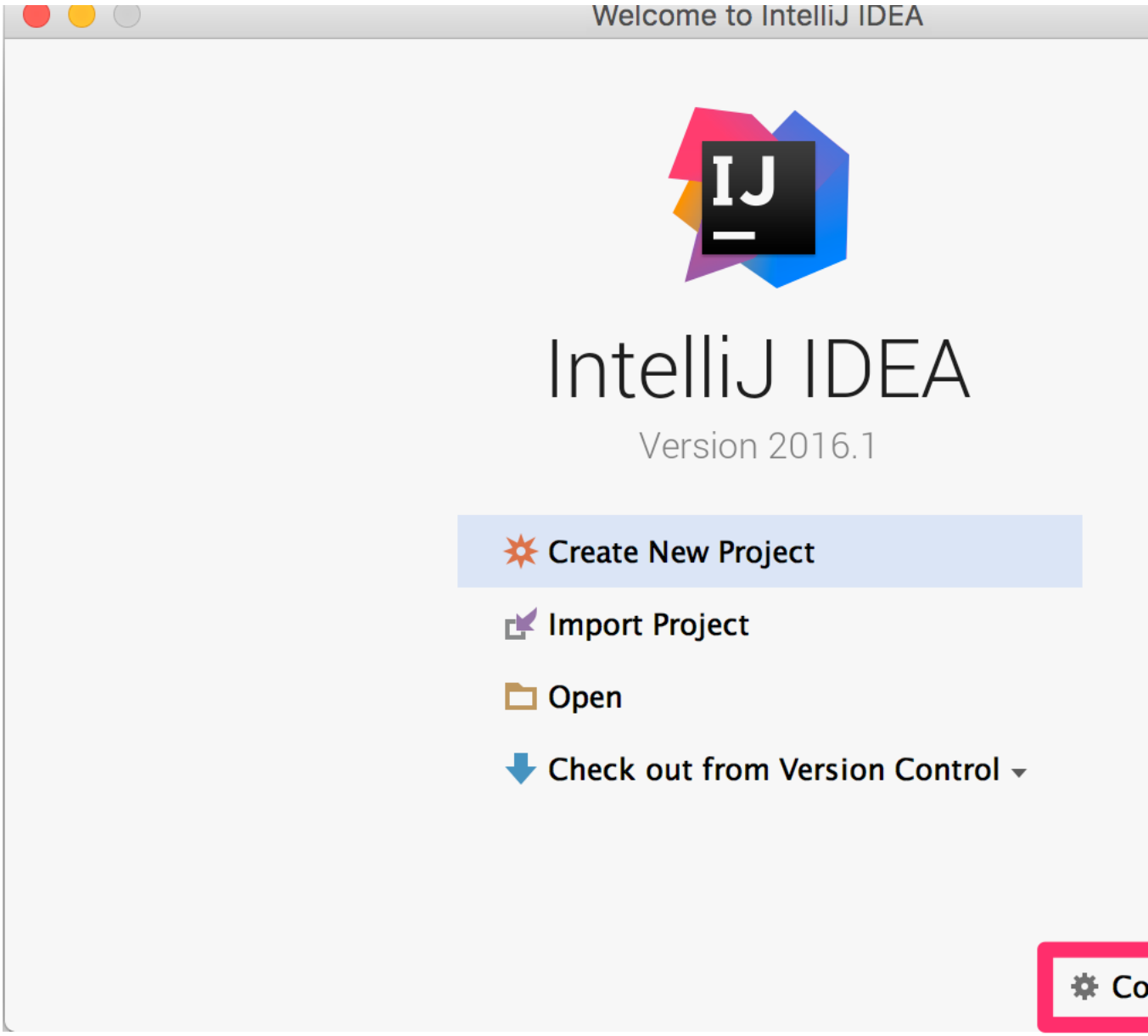

IntelliJ IDEA
Down
OS X
WI
Commu
For JVM and
developmer
DOWNLOAD
292 MB
Version: 15.0.4
Build: 143.1821
Released: February
25, 2016
System requirements
Installation
Instructions
Previous versions

ideaIC-${version}.dmg(${version} IDEA
) Windows exe

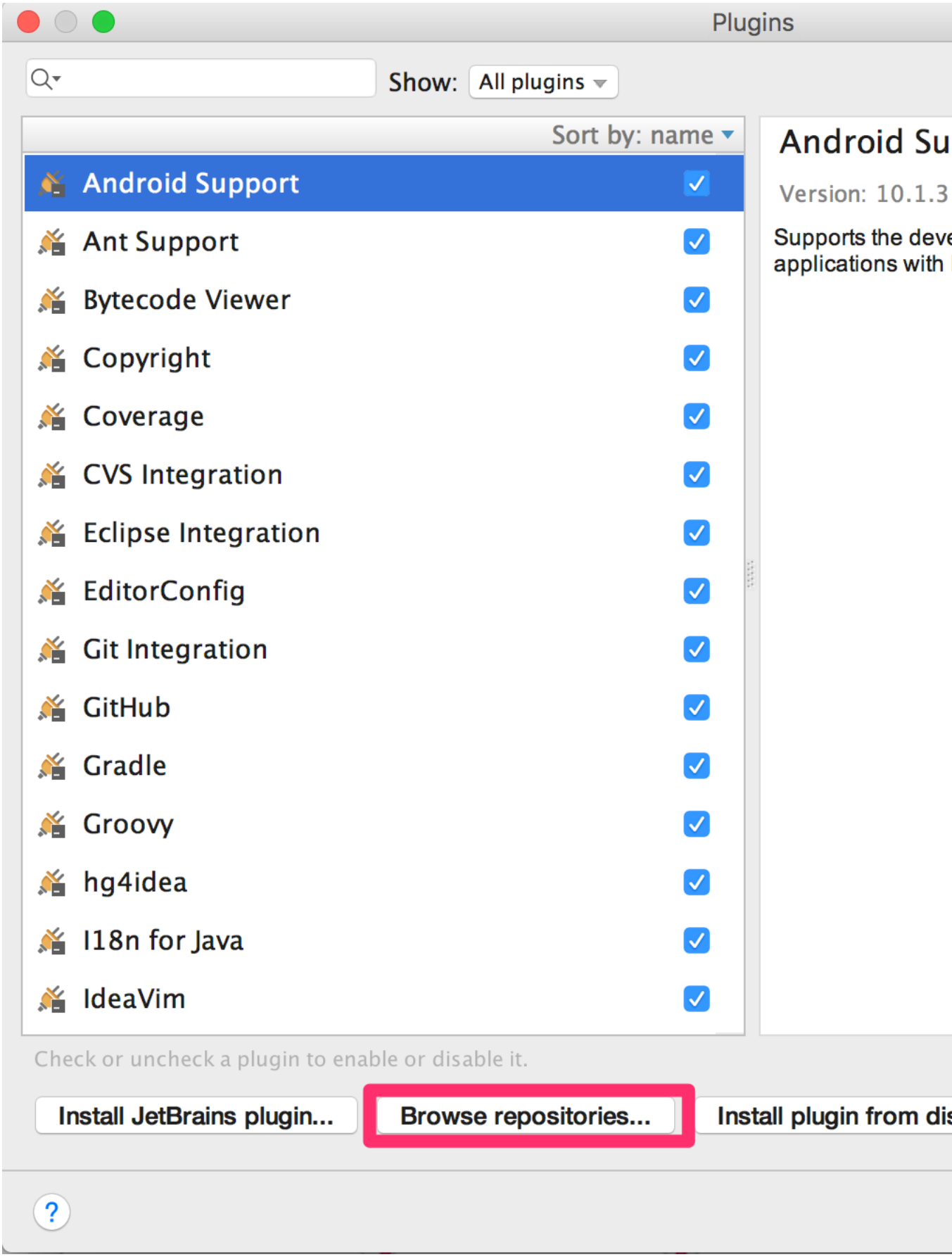


IDEA IDEA

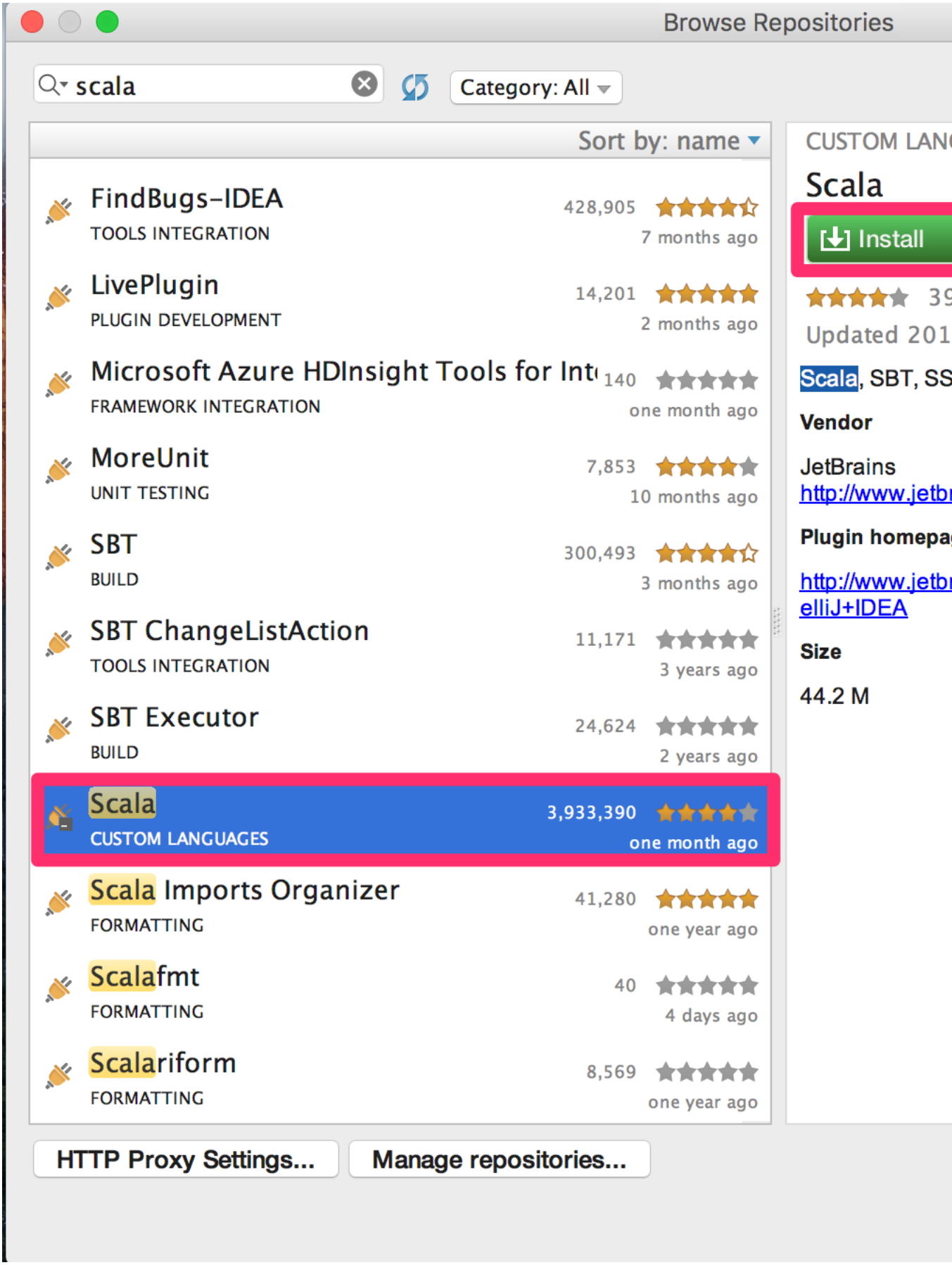
Scala `Configure->Plugins`



Browse repositories

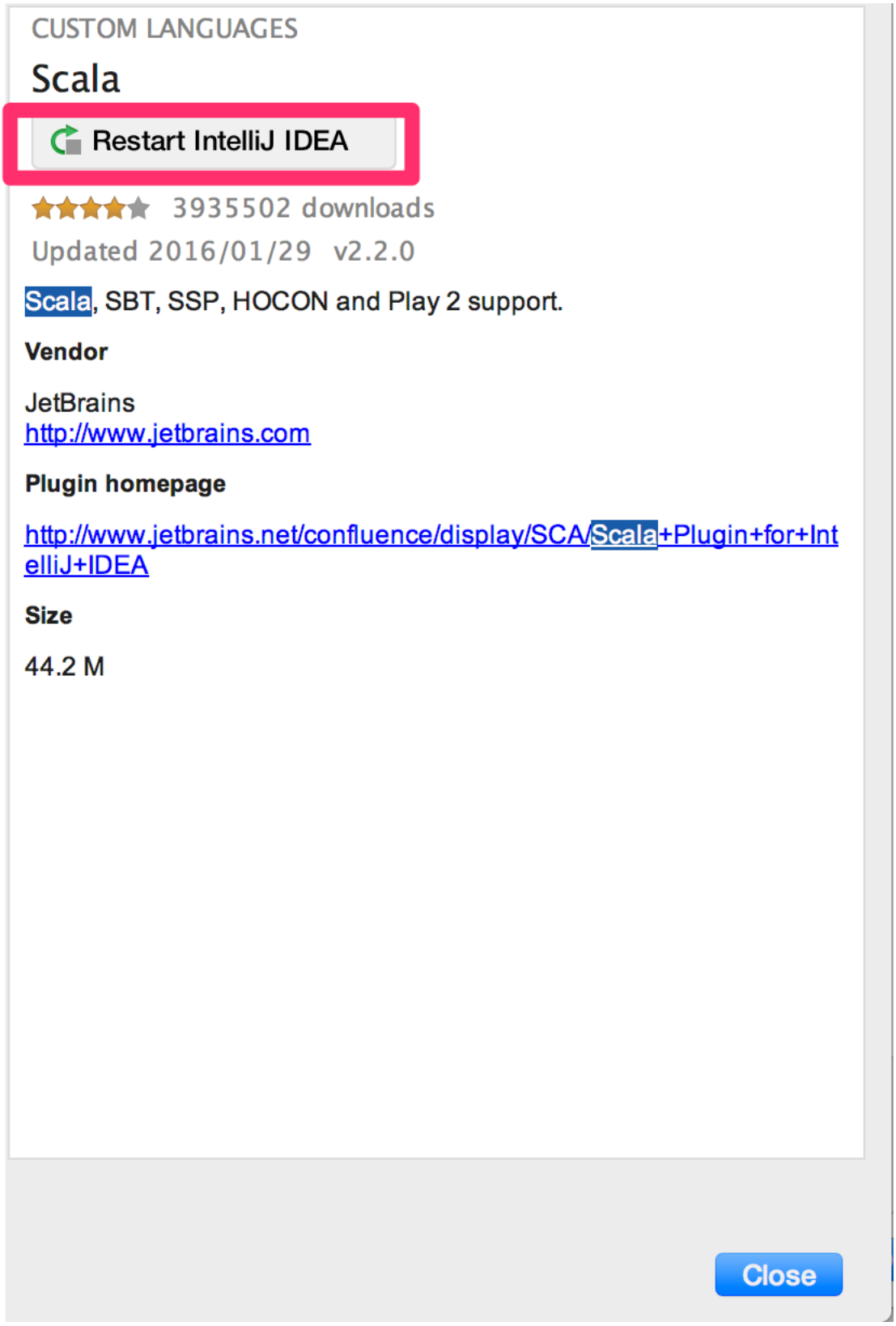

Plugins
Show:
All plugins
Sort by: name
Android Support
Ant Support
Bytecode Viewer
Copyright
Coverage
CVS Integration
Eclipse Integration
EditorConfig
Git Integration
GitHub
Gradle
Groovy
hg4idea
I18n for Java
IdeaVim
Android Su
Version: 10.1.3
Supports the deve
applications with
Check or uncheck a plugin to enable or disable it.
Install JetBrains plugin...
Browse repositories...
Install plugin from dis

`scala` Install

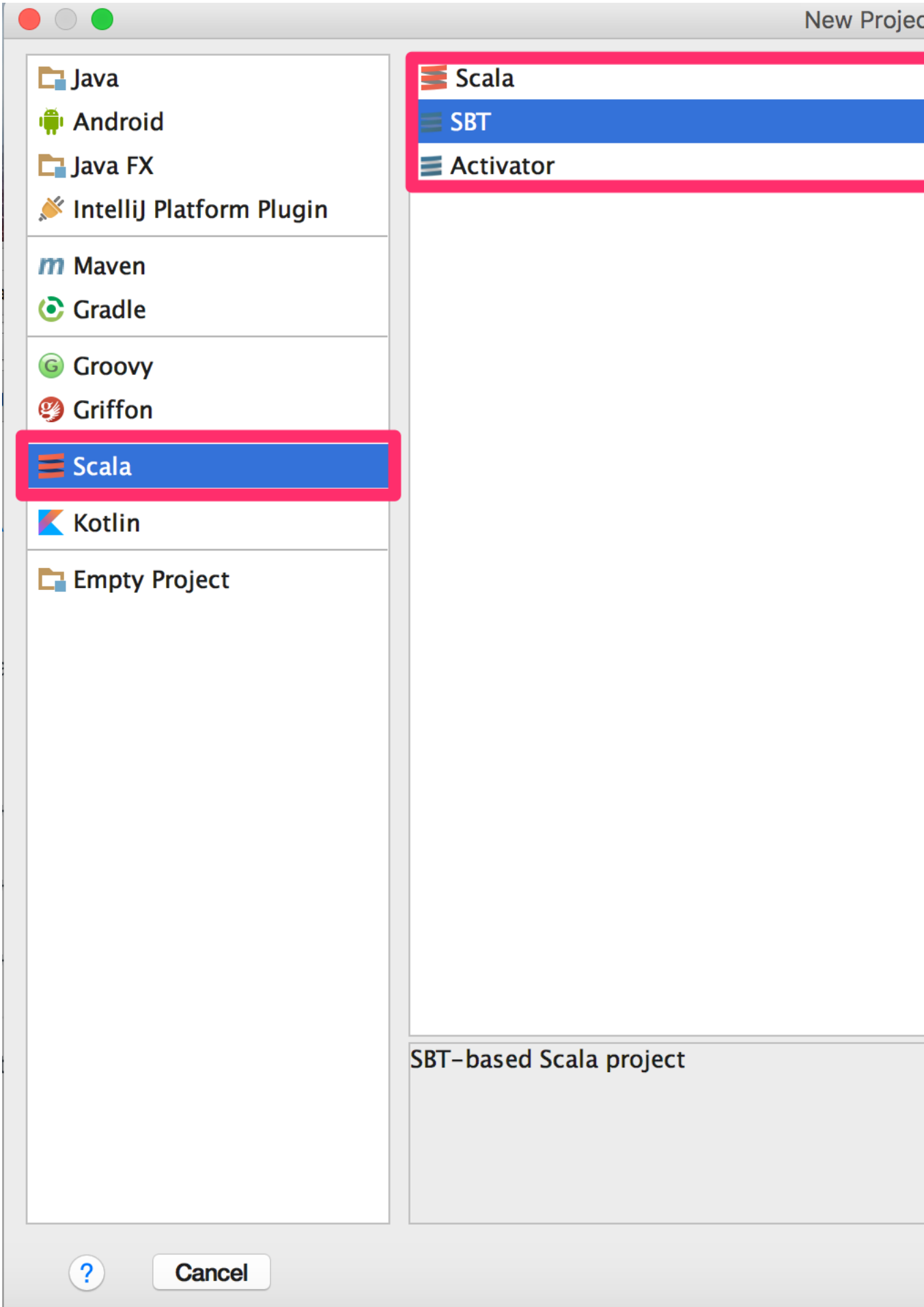

Browse Repositories
scala
Category: All
Sort by: name
FindBugs-IDEA
TOOLS INTEGRATION
428,905
7 months ago
LivePlugin
PLUGIN DEVELOPMENT
14,201
2 months ago
Microsoft Azure HDInsight Tools for Int
FRAMEWORK INTEGRATION
140
one month ago
MoreUnit
UNIT TESTING
7,853
10 months ago
SBT
BUILD
300,493
3 months ago
SBT ChangeListAction
TOOLS INTEGRATION
11,171
3 years ago
SBT Executor
BUILD
24,624
2 years ago
Scala
CUSTOM LANGUAGES
3,933,390
one month ago
Scala Imports Organizer
FORMATTING
41,280
one year ago
Scalafmt
FORMATTING
40
4 days ago
Scalariform
FORMATTING
8,569
one year ago
HTTP Proxy Settings...
Manage repositories...
CUSTOM LAN
Scala
Install
Updated 201
Scala, SBT, SS
Vendor
JetBrains
http://www.jetbr
Plugin homepa
http://www.jetbr
elliJ+IDEA
Size
44.2 M

Scala

Restart IntelliJ IDEA


CUSTOM LANGUAGES
Scala
Restart IntelliJ IDEA
3935502 downloads
Updated 2016/01/29 v2.2.0
Scala, SBT, SSP, HOCON and Play 2 support.
Vendor
JetBrains
http://www.jetbrains.com
Plugin homepage
http://www.jetbrains.net/confluence/display/SCA/Scala+Plugin+for+IntelliJ+IDEA
Size
44.2 M
Close

Create New Project


New Projec
Java
Android
Java FX
IntelliJ Platform Plugin
Maven
Gradle
Groovy
Griffon
Scala
Kotlin
Empty Project
Scala
SBT
Activator
SBT-based Scala project
?
Cancel

sbt

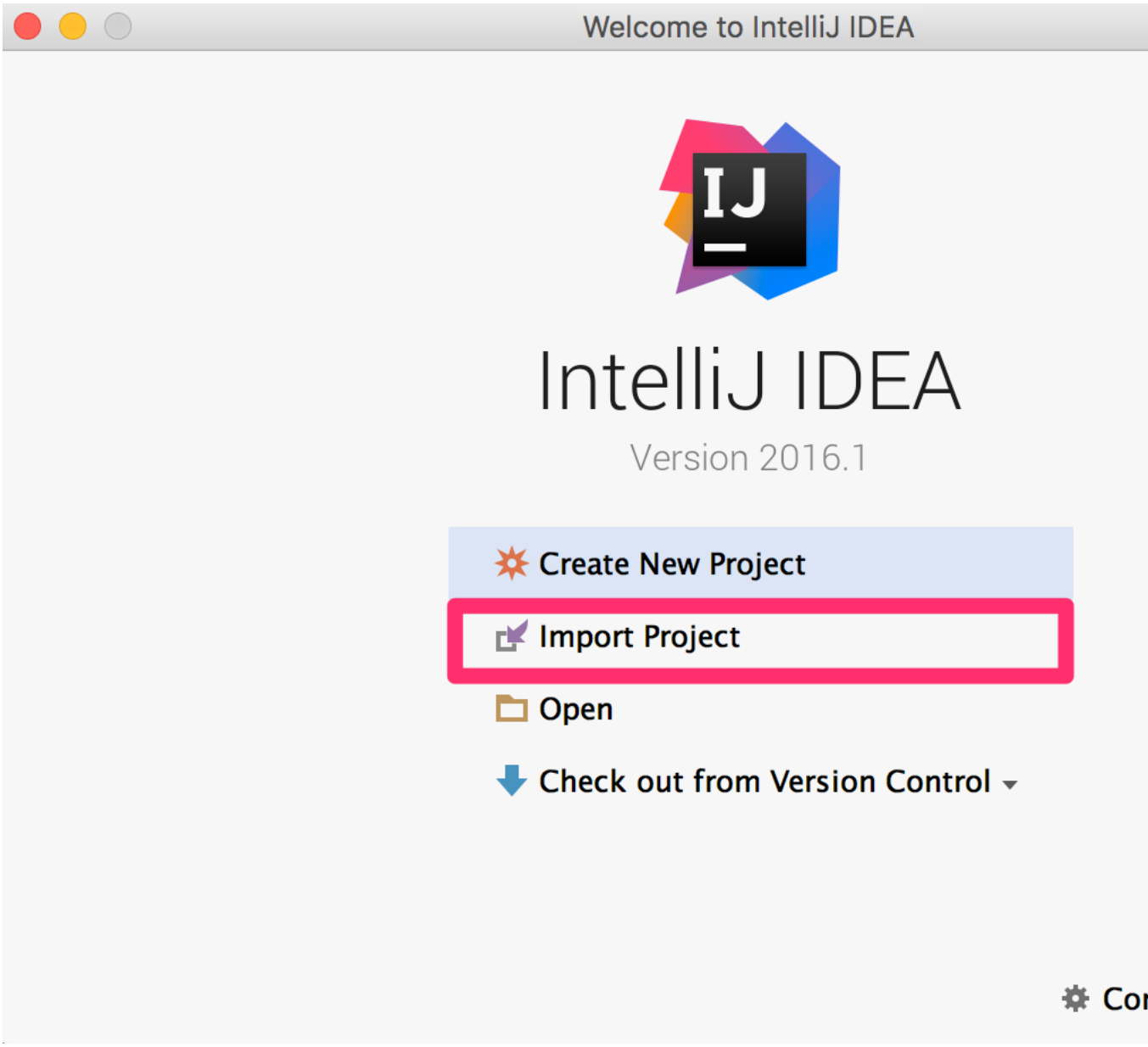
sbt IntelliJ IDEA

scala-sandbox scala-sandbox git clone clone scala-sandbox

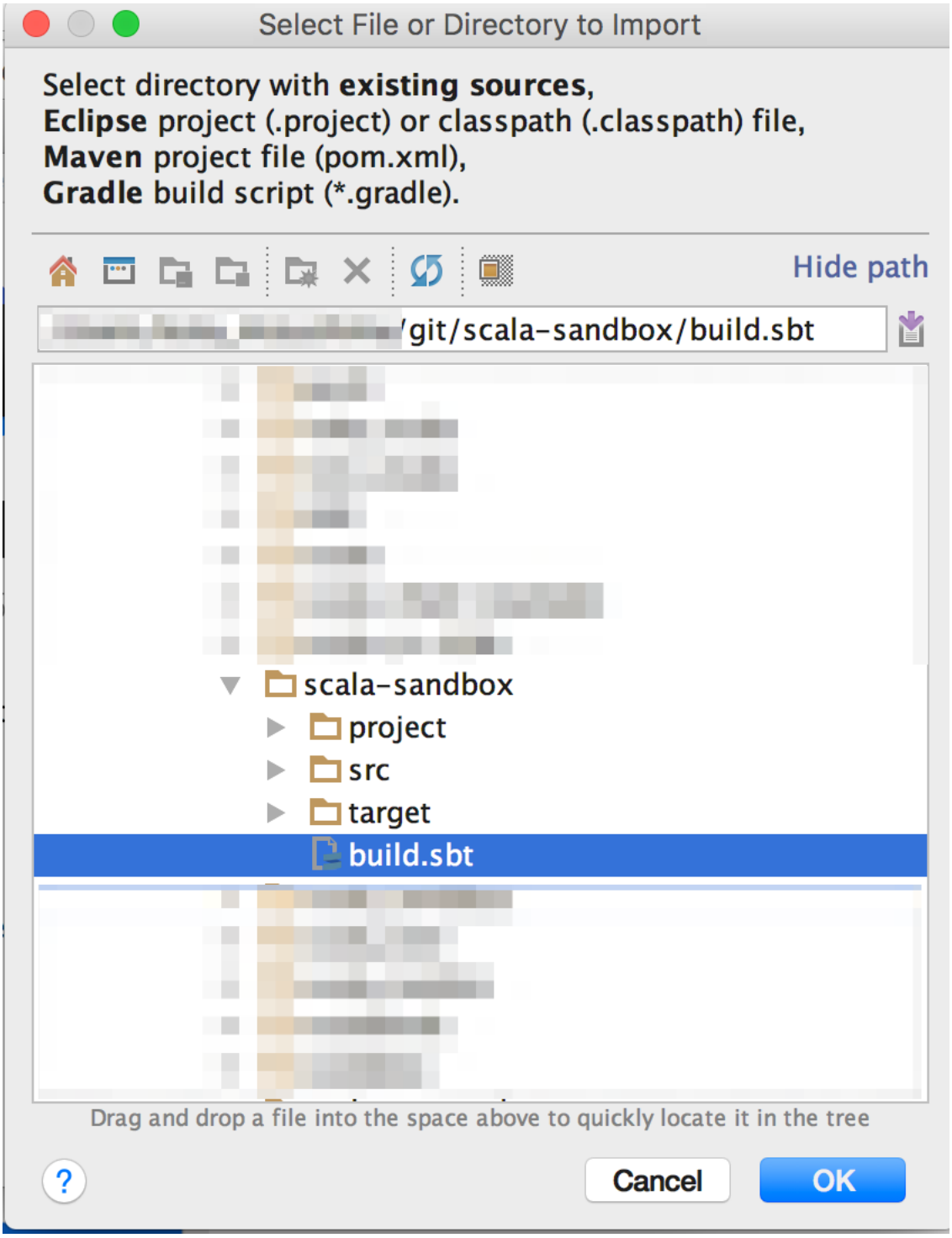
```
scala-sandbox
    build.sbt
    src/main/scala/HelloWorld.scala
    project/build.properties
    build.sbt
```

scala-sandbox IntelliJ IDEA

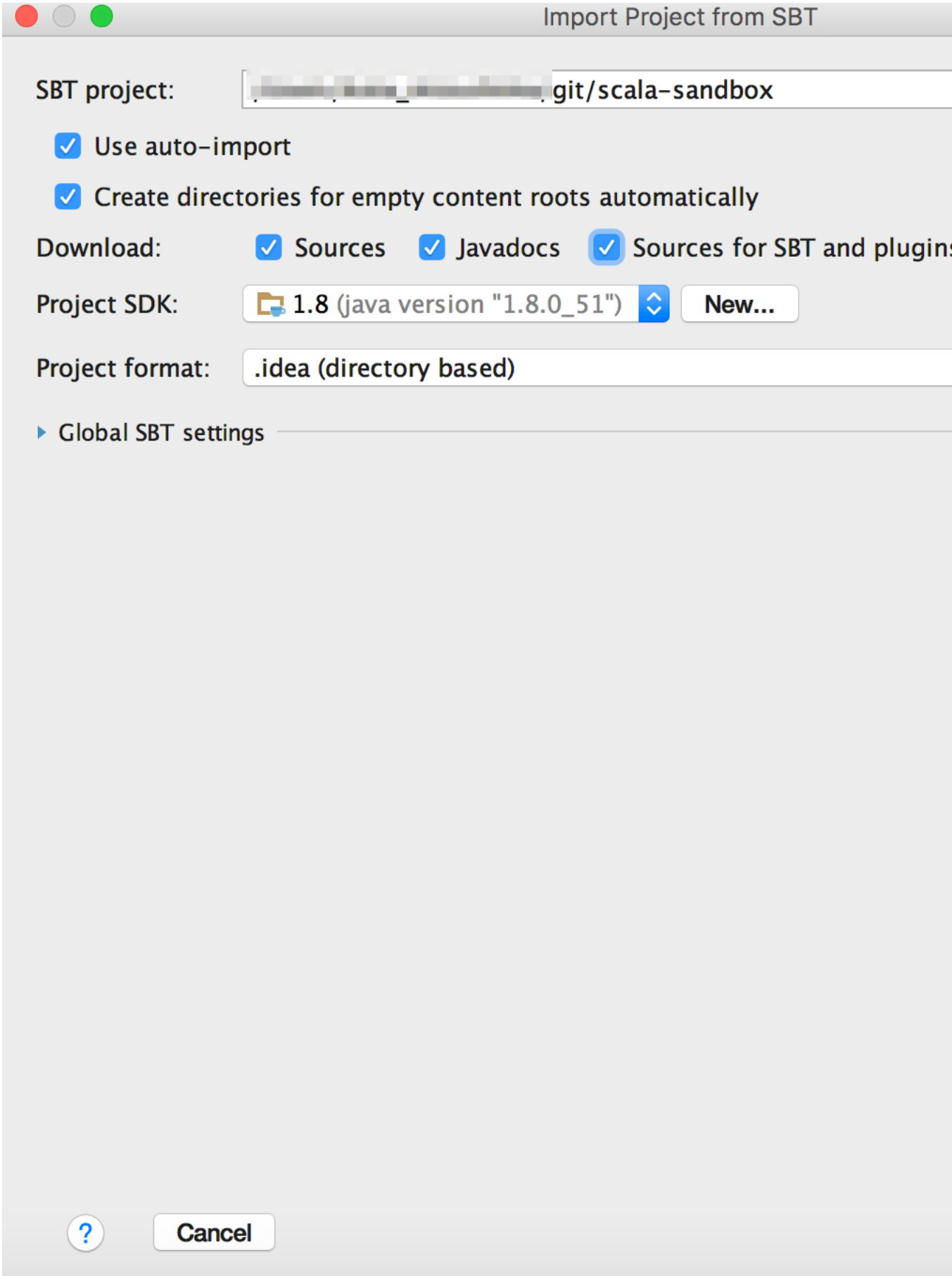
IntelliJ IDEA Import Project


Welcome to IntelliJ IDEA
IntelliJ IDEA
Version 2016.1
Create New Project
Import Project
Open
Check out from Version Control

build.sbt OK

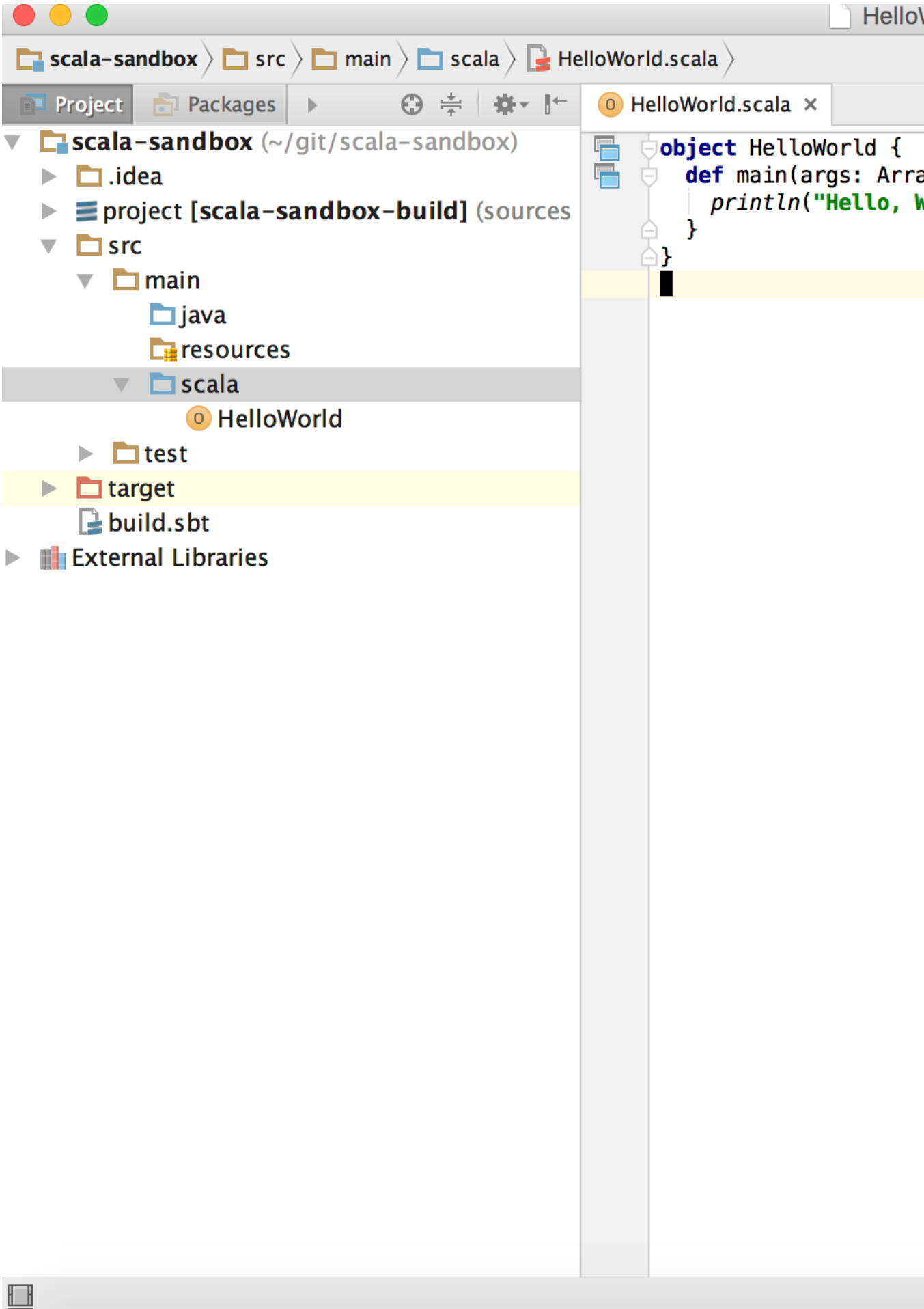

Select File or Directory to Import
Select directory with existing sources,
Eclipse project (.project) or classpath (.classpath) file,
Maven project file (pom.xml),
Gradle build script (*.gradle).
Hide path
/git/scala-sandbox/build.sbt
scala-sandbox
project
src
target
build.sbt
Drag and drop a file into the space above to quickly locate it in the tree
Cancel
OK

Project SDK New JDK
JDK


Import Project from SBT
SBT project:
git/scala-sandbox
Use auto-import
Create directories for empty content roots automatically
Download:
Sources
Javadocs
Sources for SBT and plugins
Project SDK:
1.8 (java version "1.8.0_51")
New...
Project format:
.idea (directory based)
Global SBT settings
Cancel

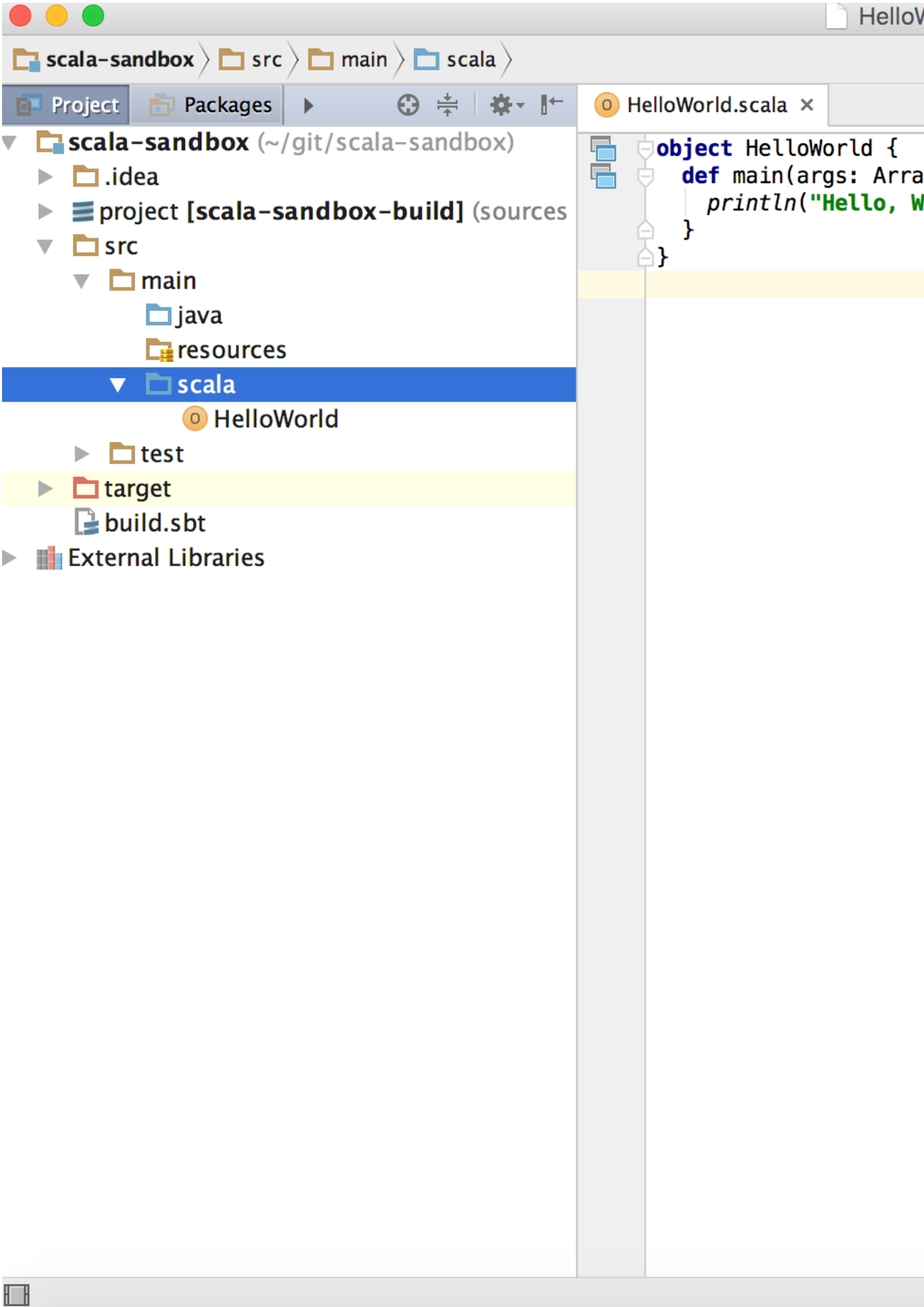
OK sbt

HelloWorld.scala

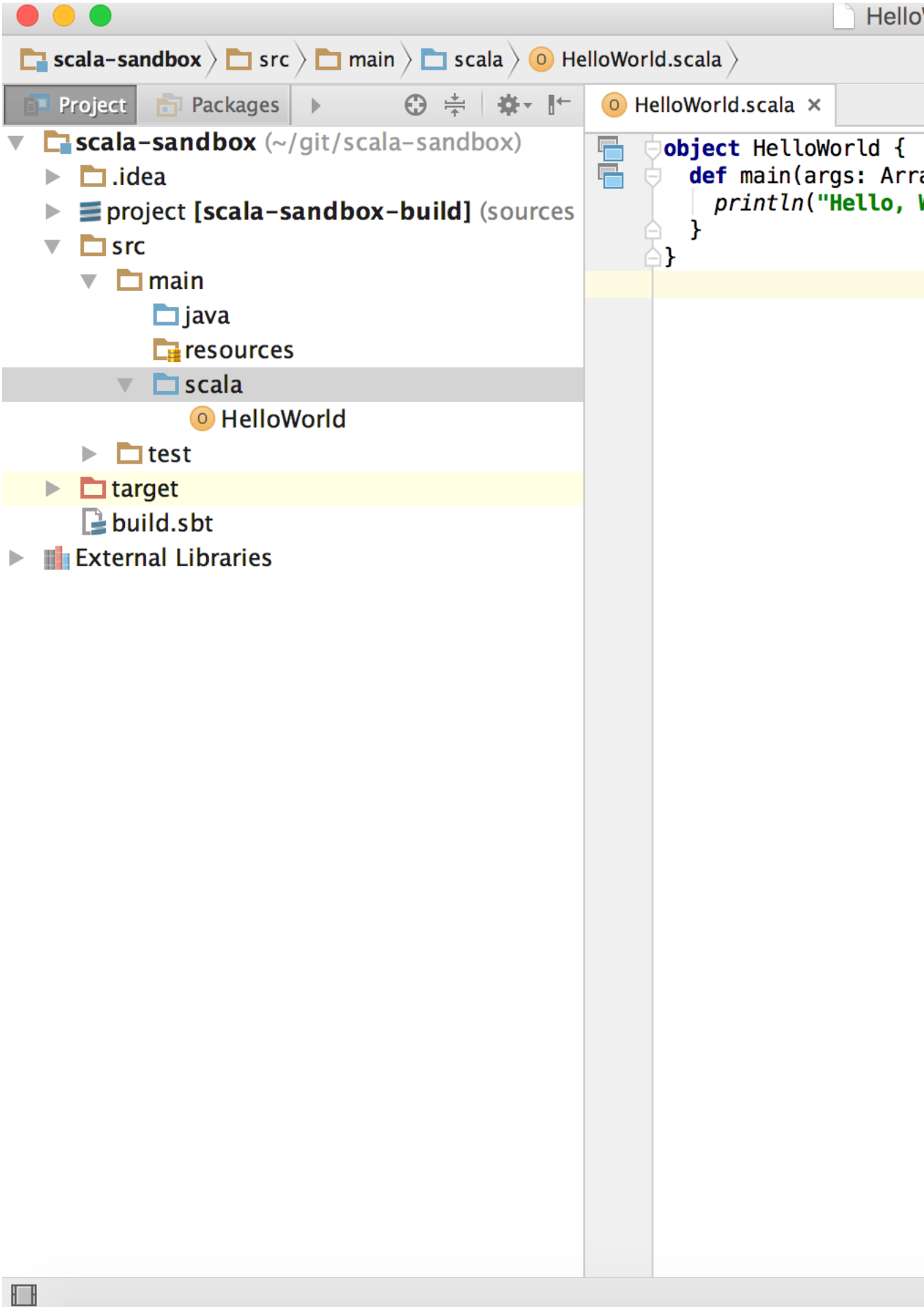

scala-sandbox
src
main
scala
HelloWorld.scala
Project
Packages
HelloWorld.scala
scala-sandbox (~/git/scala-sandbox)
.idea
project [scala-sandbox-build] (sources
src
main
java
resources
scala
HelloWorld
test
target
build.sbt
External Libraries
object HelloWorld {
def main(args: Arra
println("Hello, W
}
}

IntelliJ IDEA sbt IDE
IntelliJ IDEA sbt-idea sbt IDEA
IntelliJ IDEA 14
sbt

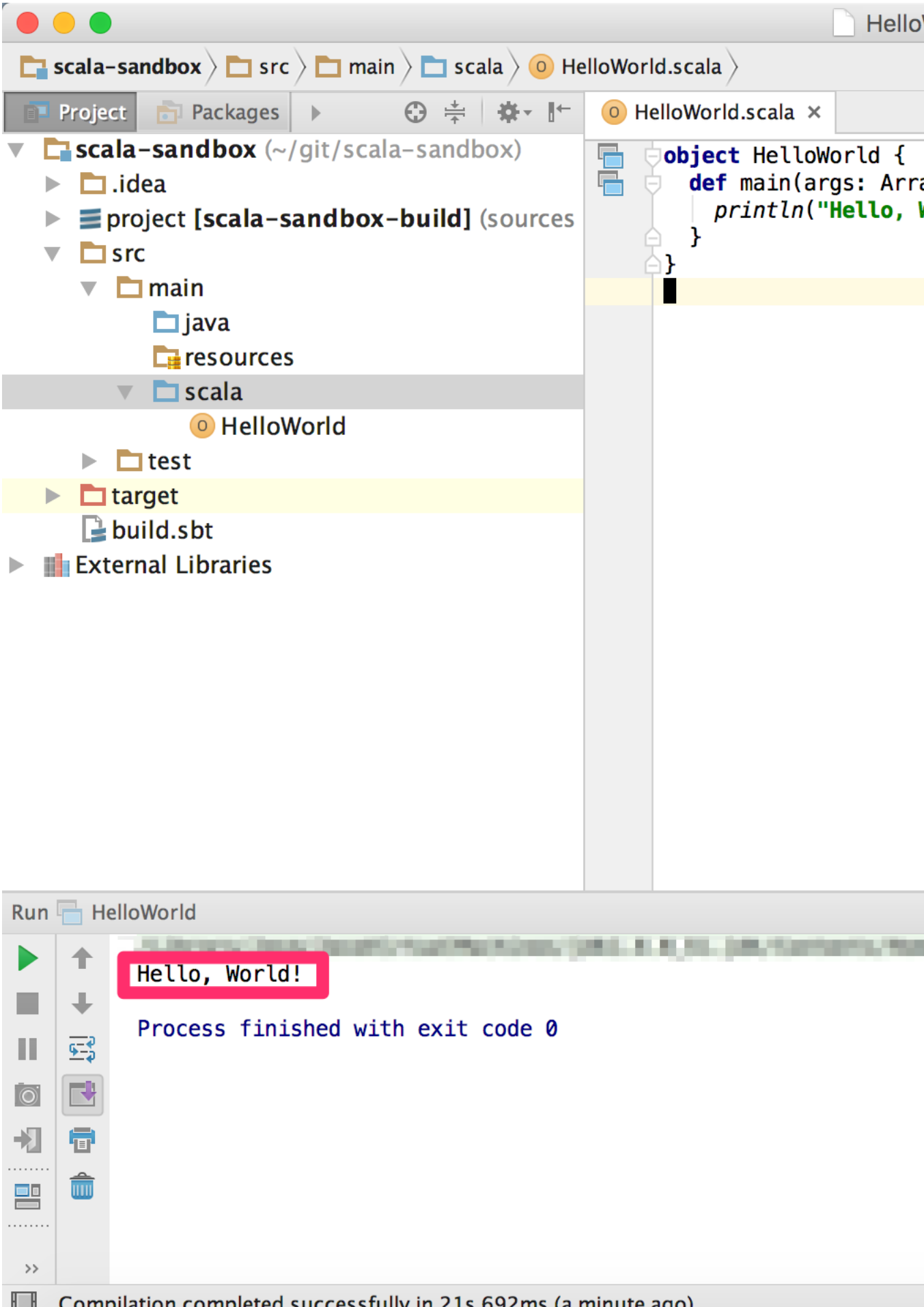
HelloWorld _HelloWorld.scala_ IDE
`Run -> Run`


scala-sandbox
src
main
scala
Project
Packages
HelloWorld.scala
scala-sandbox (~/git/scala-sandbox)
.idea
project [scala-sandbox-build] (sources
src
main
java
resources
scala
HelloWorld
test
target
build.sbt
External Libraries
object HelloWorld {
def main(args: Arra
println("Hello, W
}
}

Hello World


scala-sandbox
src
main
scala
HelloWorld.scala
Project
Packages
HelloWorld.scala
scala-sandbox (~/git/scala-sandbox)
.idea
project [scala-sandbox-build] (sources
src
main
java
resources
scala
HelloWorld
test
target
build.sbt
External Libraries
object HelloWorld {
def main(args: Arra
println("Hello,
}
}

Hello World


scala-sandbox
src
main
scala
HelloWorld.scala
Project
Packages
HelloWorld.scala
scala-sandbox (~/git/scala-sandbox)
.idea
project [scala-sandbox-build] (sources
src
main
java
resources
scala
HelloWorld
test
target
build.sbt
External Libraries
object HelloWorld {
def main(args: Arra
println("Hello,
}
}
Run HelloWorld
Hello, World!
Process finished with exit code 0
Compilation completed successfully in 21s 692ms (a minute ago)

IntelliJ IDEA

Scala IDE for Eclipse

Scala

3

Syntax

`class` `val` `if`

`class`

`{ }` Scala

Expression

`1` `1 + 2` `"hoge"`

Statement

`val i = 1` `i` `i`

`1`

Scala C Java Scala

`{}`

`{}`

```
{ exp1; exp2; ... expN; }
```

exp1 expN {}
exp1 expN expN

```
scala> { println("A"); println("B"); 1 + 2 }
A
B
res0: Int = 3
```

A B 1 + 2 3 {}

Scala

```
def foo(): String = {
  "foo" + "foo"
}
```

{} {}
{}
{}

if

if Java if if

```
if          A [else B]
```

Boolean else B A true
B false

if

```
scala> var age = 17
age: Int = 17

scala> if(age < 18) {
     |   "18         "
     | } else {
     |   "18         "
```

```
     | }
res1: String = 18

scala> age = 18
age: Int = 18

scala> if(age < 18) {
     |   "18         "
     | } else {
     |   "18         "
     | }
res2: String = 18
```

age 18

if Scala Java

?: if

else

```
if         A else ()
```

Unit

{#control_syntax_ex1}

var age: Int = 5 var isSchoolStarted: Boolean = false

1 6

“ ” “ ”

```
scala> var age: Int = 5
age: Int = 5

scala> var isSchoolStarted: Boolean = false
isSchoolStarted: Boolean = false

scala> if(1 <= age && age <= 6 && !isSchoolStarted) {
     |   println("       ")
     | } else {
     |   println("               ")
     | }
```

while

`while` Java

```
while          A
```

`Boolean` `while` `true` A
`while` `while` `Unit`
`()`

`Unit` Java `void` `()`

while 1 10

```
scala> var i = 1
i: Int = 1

scala> while(i <= 10) {
     |   println("i = " + i)
     |   i = i + 1
     | }
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
i = 10
```

Java `while` `do while` Java

{#control_syntax_ex2}

`do while` 0 9 10
`print0To9` `loopFrom0To9` `???`

```
def loop0To9(): Unit = {
  do {
    ???
  } while(???)
}
```

```
def loop0To9(): Unit = {
  var i = 0
  do {
    println(i)
    i += 1
  } while(i < 10)
}
```

```
scala> loop0To9()
0
1
2
3
4
5
6
7
8
9
```

for

Scala for Java

for flatMap, map, withFilter, foreach

for

for

for(1; 2; ... n) A

1 = a1 <- exp1; 2 = a2 <- exp2; ...

n = an <- expn

a1 an exp1 expn

1 to 10 1 10 10 1 until 10 1 10

10

for

```
scala> for(x <- 1 to 5; y <- 1 until 5){
     |   println("x = " + x + " y = " + y)
     | }
x = 1 y = 1
x = 1 y = 2
x = 1 y = 3
x = 1 y = 4
x = 2 y = 1
x = 2 y = 2
x = 2 y = 3
x = 2 y = 4
x = 3 y = 1
x = 3 y = 2
x = 3 y = 3
x = 3 y = 4
x = 4 y = 1
x = 4 y = 2
x = 4 y = 3
x = 4 y = 4
x = 5 y = 1
x = 5 y = 2
x = 5 y = 3
x = 5 y = 4
```

x 1 5 y 1 4 x, y

2

for

until if x != y x y

```
scala> for(x <- 1 to 5; y <- 1 until 5 if x != y){
     |   println("x = " + x + " y = " + y)
     | }
x = 1 y = 2
x = 1 y = 3
x = 1 y = 4
x = 2 y = 1
x = 2 y = 3
x = 2 y = 4
x = 3 y = 1
x = 3 y = 2
x = 3 y = 4
x = 4 y = 1
x = 4 y = 2
x = 4 y = 3
x = 5 y = 1
x = 5 y = 2
```

```
x = 5 y = 3
x = 5 y = 4
```

for 1 1
"A", "B", "C", "D", "E" 5

```
scala> for(e <- List("A", "B", "C", "D", "E")) println(e)
A
B
C
D
E
```

for
Pre

```
scala> for(e <- List("A", "B", "C", "D", "E")) yield {
     |   "Pre" + e
     | }
res9: List[String] = List(PreA, PreB, PreC, PreD, PreE)
```

yield for yield

yield for for-comprehension

{#control_syntax_ex3}

1 1000 3 a, b, c a, b, c

a ^ 2 == b ^ 2 + c ^ 2 a, b, c

```
for(a <- 1 to 1000; b <- 1 to 1000; c <- 1 to 1000) {
  if(a * a == b * b + c * c) println((a, b, c))
}
```

match

match match

match {

```
  case         1 [if       1] =>   1
  case         2 [if       2] =>   2
  case         3 [if       3] =>   3
  case ...
  case         N =>    N
}
```

Java

switch-case

```
scala> val taro = "Taro"
taro: String = Taro

scala> taro match {
     |   case "Taro" => "Male"
     |   case "Jiro" => "Male"
     |   case "Hanako" => "Female"
     | }
res11: String = Male
```

taro "Taro" case

"Taro" "Male"

match match =>

```
scala> val one = 1
one: Int = 1

scala> one match {
     |   case 1 => "one"
     |   case 2 => "two"
     |   case _ => "other"
     | }
res12: String = one
```

_ switch-case default

match

Java C switch-case Scala
fall through

```
"abc" match {
  case "abc" => println("first")   //
  case "def" => println("second") //
}
```

C switch-case
Java C
Scala
|

```
"abc" match {
  case "abc" | "def" =>
    println("first")
    println("second")
}
```

switch-case

```
scala> val lst = List("A", "B", "C", "D", "E")
lst: List[String] = List(A, B, C, D, E)

scala> lst match {
     |   case List("A", b, c, d, e) =>
     |     println("b = " + b)
     |     println("c = " + c)
     |     println("d = " + d)
     |     println("e = " + e)
     |   case _ =>
     |     println("nothing")
     | }
b = B
c = C
d = D
e = E
```

List "A" 5 b, c, d, e List 2

=> match

Boolean

```
scala> val lst = List("A", "B", "C", "D", "E")
lst: List[String] = List(A, B, C, D, E)

scala> lst match {
     |   case List("A", b, c, d, e) if b != "B" =>
     |     println("b = " + b)
     |     println("c = " + c)
     |     println("d = " + d)
     |     println("e = " + e)
     |   case _ =>
     |     println("nothing")
     | }
nothing
```

List 2 "B"

-

List("A") List

```
scala> val lst = List(List("A"), List("B", "C", "D", "E"))
lst: List[List[String]] = List(List(A), List(B, C, D, E))

scala> lst match {
     |   case List(a@List("A"), x) =>
     |   println(a)
     |   println(x)
     |   case _ => println("nothing")
     | }
List(A)
List(B, C, D, E)
```

lst List("A") List("B", "C", "D", "E") 2 List match

List("A")

@ as @

@ a as

:

AnyRef Java Object

AnyRef

```
scala> import java.util.Locale
import java.util.Locale

scala> val obj: AnyRef = "String Literal"
obj: AnyRef = String Literal

scala> obj match {
     |   case v:java.lang.Integer =>
     |     println("Integer!")
     |   case v:String =>
     |     println(v.toUpperCase(Locale.ENGLISH))
     | }
STRING LITERAL
```

java.lang.Integer String

equals

String v String

toUpperCase Scala

JVM

1 Scala JVM

REPL

```
scala> val obj: Any = List("a")
obj: Any = List(a)

scala> obj match {
     |   case v: List[Int]    => println("List[Int]")
     |   case v: List[String] => println("List[String]")
     | }
<console>:16: warning: non-variable type argument Int in type pattern List[Int] (the underlying of List[Int]) is uncho
         case v: List[Int]    => println("List[Int]")
                 ^
```

```
<console>:17: warning: non-variable type argument String in type pattern List[String] (the underlying of List[String]
         case v: List[String] => println("List[String]")
                 ^
<console>:17: warning: unreachable code
         case v: List[String] => println("List[String]")
                                         ^
List[Int]
```

List[Int] List[String]

2 Scala List[Int] Int

2

2 3

```
obj match {
  case v: List[_] => println("List[_]")
}
```

{#control_syntax_ex4}

```
new scala.util.Random(new java.security.SecureRandom()).alphanumeric.take(5).toList
```

5

1000 new scala.util.Random(new java.security.SecureRandom()).alphanumeric.take(5)

5 Char

List(a, b, d, e, f)

List(a, b, d, e, a)

```
for(i <- 1 to 1000) {
  val s = new scala.util.Random(new java.security.SecureRandom()).alphanumeric.take(5).toList match {
    case List(a,b,c,d,_) => List(a,b,c,d,a).mkString
  }
  println(s)
}
```

match switch-case

match

Java

Scala Java Scala

```
class ( 1 : 1 , 2 : 2 , ...) {
  0
}
```

Point Point x x Int
y Int Point Scala

```
class Point(_x: Int, _y: Int) {
  val x = _x
  val y = _y
}
```

```
class Point(val x: Int, val y: Int)
```

-
- val/var

Scala 1 1

Scala

1

val/var

```
class Point(val x: Int, val y: Int) {
  def +(p: Point): Point = {
    new Point(x + p.x, y + p.y)
  }
  override def toString(): String = "(" + x + ", " + y + ")"
}
```

+

```
(private[this]/protected[package ]) def ( 1: 1 , 2: 2 , ...): = {

}
```

1

```
(private[this]/protected[package ]) def ( 1: 1 , 2: 2 , ...): =
```

= {} {}

private

protected

private[this] protected[ ]

private

protected public

Point REPL

```
scala> class Point(val x: Int, val y: Int) {
     |   def +(p: Point): Point = {
     |     new Point(x + p.x, y + p.y)
     |   }
     |   override def toString(): String = "(" + x + ", " + y + ")"
     | }
defined class Point
```

```
scala> val p1 = new Point(1, 1)
p1: Point = (1, 1)

scala> val p2 = new Point(2, 2)
p2: Point = (2, 2)

scala> p1 + p2
res0: Point = (3, 3)
```

```
def ( 1: 1, 2: 2, ...)( N: N, ..., M: N): =
```

Scala

implicit parameter

```
scala> class Adder {
     |   def add(x: Int)(y: Int): Int = x + y
     | }
defined class Adder

scala> val adder = new Adder()
adder: Adder = Adder@12316367

scala> adder.add(2)(3)
res1: Int = 5

scala> adder.add(2) _
res2: Int => Int = <function1>
```

`obj.m(x, y)` `obj.m(x)(y)`

```
(private/protected) (val/var)          :            =
```

val var
private protected
private protected
public

1 2 1
1
*8

Java 1 Java 8

Scala

Scala Scala

```
class (....) extends           {
  ....
}
```

Java

override

```
scala> class APrinter() {
     |   def print(): Unit = {
     |     println("A")
     |   }
     | }
defined class APrinter

scala> class BPrinter() extends APrinter {
     |   override def print(): Unit = {
```

*8
Scala

```
     |     println("B")
     |   }
     | }
defined class BPrinter

scala> new APrinter().print
A

scala> new BPrinter().print
B
```

override

```
scala> class BPrinter() extends APrinter {
     |   def print(): Unit = {
     |     println("B")
     |   }
     | }
<console>:14: error: overriding method print in class APrinter of type ()Unit;
 method print needs `override' modifier
         def print(): Unit = {
             ^
```

Java

Scala override

object

Scala

Java static static

object

1

object

object

- Java static
-
- Singleton

3 Singleton

2

object

`object extends with 1 with 2 ... {`

`}`

Scala Predef object

println()

Predef object

2 Point object

apply Scala

`Point(x)` Point object apply

`Point.apply(x)` Point object apply

`Point(3, 5)`

```
scala> class Point(val x:Int, val y:Int)
defined class Point

scala> object Point {
     |   def apply(x: Int, y: Int): Point = new Point(x, y)
     | }
defined object Point
warning: previously defined class Point is not a companion to object Point.
Companions must be defined together; you may wish to use :paste mode for this.
```

new Point() Point

- Point
- Point

```
scala> case class Point(x: Int, y: Int)
defined class Point
```

`equqls()` `hashCode()` `toString()`

`case class Point(x: Int, y: Int)`

Point equals()

```
Point(1, 2).equals(Point(1, 2))
```

true

weight private

```
class Person(name: String, age: Int, private val weight: Int)

object Hoge {
  val taro = new Person("Taro", 20, 70)
  println(taro.weight)
}
```

NG

```
class Person(name: String, age: Int, private val weight: Int)

object Person {
  val taro = new Person("Taro", 20, 70)
  println(taro.weight)
}
```

OK private[this]

private

REPL REPL :paste

REPL

REPL

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

class Person(name: String, age: Int, private val weight: Int)

object Person {
```

```
  val taro = new Person("Taro", 20, 70)
  println(taro.weight)
}

// Exiting paste mode, now interpreting.

defined class Person
defined object Person
```

private

private[this]

Scala

Scala

Scala

- 1
-
-

1

Scala 1

```
trait TraitA

trait TraitB

class ClassA

class ClassB

//
class ClassC extends ClassA with TraitA with TraitB

//
// class ClassB needs to be a trait to be mixed in
class ClassD extends ClassA with ClassB
```

ClassA TraitA TraitB ClassC ClassA ClassB ClassD class ClassB needs to be a trait to be mixed in ClassB

Scala

```
trait TraitA

object ObjectA {
  //
  // trait TraitA is abstract; cannot be instantiated
  val a = new TraitA
}
```

1

1

```
trait TraitA

class ClassA extends TraitA
```

```
object ObjectA {
  //
  val a = new ClassA

  //
  val a2 = new TraitA {}
}
```

Scala

*9

```
//
class ClassA(name: String) {
  def printName() = println(name)
}

//
// traits or objects may not have parameters
trait TraitA(name: String) {
  def printName: Unit = println(name)
}
```

```
trait TraitA {
  val name: String
  def printName(): Unit = println(name)
}

//            name
class ClassA(val name: String) extends TraitA

object ObjectA {
  val a = new ClassA("dwango")

  // name
  val a2 = new TraitA { val name = "kadokawa" }
```

*9 http://docs.scala-lang.org/sips/pending/trait-parameters.html

```
}
```

Schärli 2003 ECOOP Traits: Composable Units of Behaviour Scala

Scala Scala

1

greet TraitA greet TraitB TraitC TraitB TraitC ClassA

```
trait TraitA {
  def greet(): Unit
}

trait TraitB extends TraitA {
  def greet(): Unit = println("Good morning!")
}

trait TraitC extends TraitA {
  def greet(): Unit = println("Good evening!")
```

```
}

class ClassA extends TraitB with TraitC
```

TraitB TraitC greet ClassA greet
TraitB greet TraitC
greet

Scala

```
ClassA.scala:13: error: class ClassA inherits conflicting members:
  method greet in trait TraitB of type ()Unit  and
  method greet in trait TraitC of type ()Unit
(Note: this can be resolved by declaring an override in class ClassA.)
class ClassA extends TraitB with TraitC
      ^
one error found
```

Scala override

1 Note: this can be resolved by declaring an override in class ClassA. ClassA greet override

```
class ClassA extends TraitB with TraitC {
  override def greet(): Unit = println("How are you?")
}
```

ClassA super TraitB TraitC

```
class ClassB extends TraitB with TraitC {
  override def greet(): Unit = super[TraitB].greet()
}
```

```
scala> (new ClassA).greet()
How are you?

scala> (new ClassB).greet()
Good morning!
```

TraitB TraitC 1

TraitB TraitC

```
class ClassA extends TraitB with TraitC {
  override def greet(): Unit = {
    super[TraitB].greet()
    super[TraitC].greet()
  }
}
```

Scala linearization

linearization

Scala

TraitB TraitC greet

override

```
trait TraitA {
  def greet(): Unit
}

trait TraitB extends TraitA {
  override def greet(): Unit = println("Good morning!")
}

trait TraitC extends TraitA {
  override def greet(): Unit = println("Good evening!")
}

class ClassA extends TraitB with TraitC
```

ClassA greet

```
scala> (new ClassA).greet()
Good evening!
```

ClassA greet TraitC greet

TraitC

TraitB

```
class ClassB extends TraitC with TraitB
```

ClassB greet TraitB greet

```
scala> (new ClassB).greet()
Good morning!
```

super

```
trait TraitA {
  def greet(): Unit = println("Hello!")
}

trait TraitB extends TraitA {
  override def greet(): Unit = {
    super.greet()
    println("My name is Terebi-chan.")
  }
}

trait TraitC extends TraitA {
  override def greet(): Unit = {
    super.greet()
    println("I like niconico.")
  }
}

class ClassA extends TraitB with TraitC
class ClassB extends TraitC with TraitB
```

greet

```
scala> (new ClassA).greet()
Hello!
My name is Terebi-chan.
I like niconico.

scala> (new ClassB).greet()
Hello!
I like niconico.
My name is Terebi-chan.
```

Scala

Stackable Trait

Scala

abstract override

super

Scala

abstract override

abstract override override abstract override

```
trait TraitA {
  def greet(): Unit
}

//
// method greet in trait TraitA is accessed from super. It may not be abstract unless it is overridden by a member dec
trait TraitB extends TraitA {
  override def greet(): Unit = {
    super.greet()
    println("Good morning!")
  }
}

//
trait TraitC extends TraitA {
  abstract override def greet(): Unit = {
    super.greet()
    println("Good evening!")
  }
}
```

abstract TraitB

TraitA greet abstract override

abstract override

abstract override 1

```
trait TraitA {
  def greet(): Unit
}
```

```
trait TraitB extends TraitA {
  def greet(): Unit =
    println("Hello!")
}

trait TraitC extends TraitA {
  abstract override def greet(): Unit = {
    super.greet()
    println("I like niconico.")
  }
}

//
// class ClassA needs to be a mixin, since method greet in trait TraitC of type ()Unit is marked `abstract' and `overr
class ClassA extends TraitC

//
class ClassB extends TraitB with TraitC
```

Scala

self type annotations self types

```
trait Greeter {
  def greet(): Unit
}

trait Robot {
  self: Greeter =>

  def start(): Unit = greet()
}
```

Robot start greet Robot
Greeter greet

greet

REPL

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

trait HelloGreeter extends Greeter {
  def greet(): Unit = println("Hello!")
```

```
}

// Exiting paste mode, now interpreting.

defined trait HelloGreeter

scala> val r = new Robot with HelloGreeter
r: Robot with HelloGreeter = $anon$1@1e5756c0

scala> r.start()
Hello!
```

Dependency Injection

```
trait Greeter {
  def greet(): Unit
}

trait Robot2 extends Greeter {
  def start(): Unit = greet()
}
```

Robot2 Greeter

```
scala> val r: Robot = new Robot with HelloGreeter
r: Robot = $anon$1@10470bfa

scala> r.greet()
<console>:9: error: value greet is not a member of Robot
              r.greet()

scala> val r: Robot2 = new Robot2 with HelloGreeter
r: Robot = $anon$1@10470bfa

scala> r.greet()
Hello!
```

Robot2 Greeter greet Robot greet

Robot Greeter

1

```
//
trait Greeter {
  self: Robot =>

  def greet(): Unit = println(s"My name is $name")
}

trait Robot {
  self: Greeter =>

  def name: String

  def start(): Unit = greet()
}
```

```
//
// illegal cyclic reference involving trait Greeter
trait Greeter extends Robot {
  def greet(): Unit = println(s"My name is $name")
}

trait Robot extends Greeter {
  def name: String

  def start(): Unit = greet()
}
```

Scala val

A foo B foo bar

C foo bar

```
trait A {
  val foo: String
}

trait B extends A {
  val bar = foo + "World"
}

class C extends B {
  val foo = "Hello"

  def printBar(): Unit = println(bar)
}
```

REPL C printBar

```
scala> (new C).printBar()
nullWorld
```

nullWorld C foo

Scala

C B B

A A foo

null B bar null

foo “World” "nullWorld" bar

Scala val

val

foo

bar lazy val def

```
trait A {
  val foo: String
}

trait B extends A {
  lazy val bar = foo + "World" //         def bar
}

class C extends B {
  val foo = "Hello"

  def printBar(): Unit = println(bar)
}
```

nullWorld bar lazy val
bar
C foo foo

C printBar HelloWorld

```
scala> (new C).printBar()
HelloWorld
```

lazy val val
val def
val val
lazy val def

val 1 Early Definitions

```
trait A {
  val foo: String
}

trait B extends A {
  val bar = foo + "World" // val
}

class C extends {
  val foo = "Hello" //
} with B {
```

```
  def printBar(): Unit = println(bar)
}
```

C printBar HelloWorld

B

B

type parameter

0

```
class [ 1, 2, ..., N]( 1 : 1 ,
{
  0
}
```

1 N

1 put get

Cell Cell

```
class Cell[T](var value: T) {
  def put(newValue: T): Unit = {
    value = newValue
  }

  def get(): T = value
}
```

REPL

```
scala> class Cell[T](var value: T) {
     |   def put(newValue: T): Unit = {
     |     value = newValue
     |   }
     |
     |   def get(): T = value
```

```
     | }
defined class Cell

scala> val cell = new Cell[Int](1)
cell: Cell[Int] = Cell@192aaffb

scala> cell.put(2)

scala> cell.get()
res1: Int = 2

scala> cell.put("something")
<console>:10: error: type mismatch;
 found   : String("something")
 required: Int
              cell.put("something")
                       ^
```

```
scala> val cell = new Cell[Int](1)
cell: Cell[Int] = Cell@66428d6c
```

Int 1 Int
Cell REPL String put
Cell

-
-

2
2 Pair Pair
toString

```
class Pair[T1, T2](val t1: T1, val t2: T2) {
  override def toString(): String = "(" + t1 + "," + t2 + ")"
}
```

Pair divide
divide

```
def divide(m: Int, n: Int): Pair[Int, Int] = new Pair[Int, Int](m / n, m % n)
```

REPL

```
scala> class Pair[T1, T2](val t1: T1, val t2: T2) {
     |   override def toString(): String = "(" + t1 + "," + t2 + ")"
     | }
defined class Pair

scala> def divide(m: Int, n: Int): Pair[Int, Int] = new Pair[Int, Int](m / n, m % n)
divide: (m: Int, n: Int)Pair[Int,Int]

scala> divide(7, 3)
res0: Pair[Int,Int] = (2,1)
```

7 3 res0 new Pair[Int, Int](m / n, m % n)

Pair Int Int new Pair(m / n, m % n)

Pair 2

Pair Scala Tuple1 Tuple22(Tuple

```
scala> val m = 7
m: Int = 7

scala> val n = 3
n: Int = 3

scala> new Tuple2(m / n, m % n)
res1: (Int, Int) = (2,1)
```

```
scala> val m = 7
m: Int = 7

scala> val n = 3
n: Int = 3

scala> (m / n, m % n)
res2: (Int, Int) = (2,1)
```

variance

covariant

Scala invariant

G T1 T2 T1 = T2

```
val : G[T1] = G[T2]
```

Java

G T1 T2 T1 T2

```
val : G[T2] = G[T1]
```

Scala

```
class G[+T]
```

+

Java Scala

Java G = T1 = String, T2 = Object

```
Object[] objects = new String[1];
objects[0] = 100;
```

Java Object

String

java.lang.ArrayStoreException objects

String String 2 int
Integer 100

Scala
Any AnyRef AnyVal

scala scala> val arr: Array[Any] = new Array[String](1) <console>:7: error: type mismatch; found : Array[String] required: Array[Any]

Scala
Scala Java

Scala

Pair[T1, T2] Pair[T1, T2]
ArrayStoreException
Pair[T1, T2] class Pair[+T1, +T2]

```
scala> class Pair[+T1, +T2](val t1: T1, val t2: T2) {
     |   override def toString(): String = "(" + t1 + "," + t2 + ")"
     | }
defined class Pair

scala> val pair: Pair[AnyRef, AnyRef] = new Pair[String, String]("foo", "bar")
pair: Pair[AnyRef,AnyRef] = (foo,bar)
```

Pair ArrayStoreException
immutable

immutable *Stack* ??? *Stack*
E >: T E T Nothing
Stack[T] Stack[Nothing]
Stack

```
trait Stack[+T] {
  def pop: (T, Stack[T])
```

```
  def push[E >: T](e: E): Stack[E]
  def isEmpty: Boolean
}

class NonEmptyStack[+T](private val top: T, private val rest: Stack[T]) extends Stack[T] {
  def push[E >: T](e: E): Stack[E] = ???
  def pop: (T, Stack[T]) = ???
  def isEmpty: Boolean = ???
}

case object EmptyStack extends Stack[Nothing] {
  def pop: Nothing = throw new IllegalArgumentException("empty stack")
  def push[E >: Nothing](e: E): Stack[E] = new NonEmptyStack[E](e, this)
  def isEmpty: Boolean = true
}

object Stack {
  def apply(): Stack[Nothing] = EmptyStack
}
```

```
class NonEmptyStack[+T](private val top: T, private val rest: Stack[T]) extends Stack[T] {
  def push[E >: T](e: E): Stack[E] = new NonEmptyStack[E](e, this)
  def pop: (T, Stack[T]) = (top, rest)
  def isEmpty: Boolean = false
}
```

contravariant

G T1 T2 T1 T2

val : G[T1] = G[T2]

Scala

```
class G[-T]
```

-

1 T1 T2

```
val x1: T1 => AnyRef = T2 => AnyRef
x1(T1      )
```

T1 T2

T1 = String, T2 = AnyRef

```
val x1: String => AnyRef = AnyRef => AnyRef
x1(String     )
```

x1 AnyRef => AnyRef String

AnyRef String

T1 T2 T1 = AnyRef, T2 = String String AnyRef

x1

REPL

```
scala> val x1: AnyRef => AnyRef = (x: String) => (x:AnyRef)
<console>:7: error: type mismatch;
 found   : String => AnyRef
 required: AnyRef => AnyRef
       val x1: AnyRef => AnyRef = (x: String) => (x:AnyRef)
                                                 ^

scala> val x1: String => AnyRef = (x: AnyRef) => x
x1: String => AnyRef = <function1>
```

bounds

T T

T Any

T

bounds 2

upper bounds

1 upper bounds

<:

show Show Show

ShowablePair

```
abstract class Show {
  def show: String
}
class ShowablePair[T1 <: Show, T2 <: Show](val t1: T1, val t2: T2) extends Show {
  override def show: String = "(" + t1.show + "," + t2.show + ")"
}
```

T1 T2 Show t1 t2
show Any

lower bounds

2 lower
bounds

Stack Stack

```
abstract class Stack[+E]{
  def push(element: E): Stack[E]
  def top: E
  def pop: Stack[E]
  def isEmpty: Boolean
}
```

```
error: covariant type E occurs in contravariant position in type E of value element
         def push(element: E): Stack[E]
                               ^
```

T

T
Stack

F push
Stack E

```
abstract class Stack[+E]{
  def push[F >: E](element: F): Stack[F]
  def top: E
  def pop: Stack[E]
```

```
  def isEmpty: Boolean
}
```

Stack E

F

Stack

Scala

Scala Scala Function0 Function22

2 add

```
scala> val add = new Function2[Int, Int, Int]{
     |   def apply(x: Int, y: Int): Int = x + y
     | }
add: (Int, Int) => Int = <function2>

scala> add.apply(100, 200)
res0: Int = 300

scala> add(100, 200)
res1: Int = 300
```

Function0 Function22 apply

apply Scala x.apply(y) x(y)

Function0 Function22

Scala

Scala Function0 Function22

add

```
scala> val add = (x: Int, y: Int) => x + y
add: (Int, Int) => Int = <function2>
```

add

add Scala

FunctionN

Scala First Class Object

```
(n1: N1, n2: N2, n3: N3, ...nn: NN) => B
```

n1 nn N1 NN B

B Scala

Function0 Function22

22

FunctionN[...]

```
(n1: N1, n2: N2, n3: N3, ...nn: NN) => B
```

FunctionN[N1, N2, N3, ...NN, B]

```
(N1, N2, N3, ...NN) => B
```

FunctionN

Scala

(Int, Int) => Int Int => Int => Int

Scala add

```
scala> val add = (x: Int, y: Int) => x + y
add: (Int, Int) => Int = <function2>

scala> val addCurried = (x: Int) => ((y: Int) => x + y)
addCurried: Int => (Int => Int) = <function1>

scala> add(100, 200)
res2: Int = 300

scala> addCurried(100)(200)
res3: Int = 300
```

Scala

Scala

def

REPL

Web 2

Scala Scala

def

Scala

```
scala> def double(n: Int, f: Int => Int): Int = {
     |   f(f(n))
     | }
double: (n: Int, f: Int => Int)Int
```

f 2 n double

f g 1

```
scala> double(1, m => m * 2)
res4: Int = 4

scala> double(2, m => m * 3)
res5: Int = 18

scala> double(3, m => m * 4)
res6: Int = 48
```

1 2 1 * 2 * 2 = 4

2 2 3 2

* 3 * 3 = 18 3 4

3 * 4 * 4 = 48

1.
2.
3.

around

```
scala> def around(init: () => Unit, body: () => Any, fin: () => Unit): Any = {
     |   init()
     |   try {
     |     body()
     |   } finally {
     |     fin()
     |   }
     | }
around: (init: () => Unit, body: () => Any, fin: () => Unit)Any
```

try-finally Java

around

```
scala> around(
     |   () => println("            "),
     |   () => println("                 "),
     |   () => println("              ")
     | )



res7: Any = ()
```

around body

throw Java

```
scala> around(
     |   () => println("            "),
     |   () => throw new Exception("          "),
     |   () => println("              ")
     | )


java.lang.Exception:
  at $anonfun$3.apply(<console>:16)
  at $anonfun$3.apply(<console>:16)
  at .around(<console>:15)
  ... 806 elided
```

body fin

around

1.
2.
3.

around 1 3 1 3

1

Java 7 try-with-resources

Scala immutable mutable

Scala Array List Map Set
Scala
Scala immutable
mutable mutable
Scala immutable

immutable

-
-
-
-

mutable
mutable

Scala

- Array(mutable)
- List(immutable)
- Map(immutable) Map(mutable)
- Set(immutable) Set(mutable)

Array

```
scala> val arr = Array(1, 2, 3, 4, 5)
arr: Array[Int] = Array(1, 2, 3, 4, 5)
```

1 5 arr Scala
0
Array(1, 2, 3, 4, 5) Int

```
scala> val arr = Array[Int](1, 2, 3, 4, 5)
arr: Array[Int] = Array(1, 2, 3, 4, 5)
```

[Int] Array

[Int] Array

Array

Int

```
scala> arr(0) = 7

scala> arr
res1: Array[Int] = Array(7, 2, 3, 4, 5)

scala> arr(0)
res2: Int = 7
```

arr[0]

0 7

arr.length

```
scala> arr.length
res3: Int = 5
```

Array[Int] Java int[] Scala

Collection[ElementType]

Java

JVM

equals **true** , **ClassTag**

Array

i j swapArray

swapArray

```
def swapArray[T](arr: Array[T])(i: Int, j: Int): Unit = ???
```

i j

```
def swapArray[T](arr: Array[T])(i: Int, j: Int): Unit = {
  val tmp = arr(i)
  arr(i) = arr(j)
  arr(j) = tmp
}
```

```
scala> val arr = Array(1, 2, 3, 4, 5)
arr: Array[Int] = Array(1, 2, 3, 4, 5)

scala> swapArray(arr)(0, 4)

scala> arr
res5: Array[Int] = Array(5, 2, 3, 4, 1)

scala> swapArray(arr)(1, 3)

scala> arr
res7: Array[Int] = Array(5, 4, 3, 2, 1)
```

Range

Range Range to until toList List REPL Range

```
scala> 1 to 5
res8: scala.collection.immutable.Range.Inclusive = Range(1, 2, 3, 4, 5)

scala> (1 to 5).toList
res9: List[Int] = List(1, 2, 3, 4, 5)

scala> 1 until 5
res10: scala.collection.immutable.Range = Range(1, 2, 3, 4)

scala> (1 until 5).toList
res11: List[Int] = List(1, 2, 3, 4)
```

to until Range toList List

List

Array Scala Array List Vector Vector

List immutable
Scala
List

```
scala> val lst = List(1, 2, 3, 4, 5)
lst: List[Int] = List(1, 2, 3, 4, 5)
```

```
scala> lst(0) = 7
<console>:14: error: value update is not a member of List[Int]
       lst(0) = 7
       ^
```

List List
List List
List
List

Nil List

Nil Scala List Nil Ruby
nil Scala
object Nil
::

:: - List

:: List
REPL

```
scala> val a1 = 1 :: Nil
a1: List[Int] = List(1)

scala> val a2 = 2 :: a1
a2: List[Int] = List(2, 1)

scala> val a3 = 3 :: a2
a3: List[Int] = List(3, 2, 1)

scala> val a4 = 4 :: a3
a4: List[Int] = List(4, 3, 2, 1)

scala> val a5 = 5 :: a3
a5: List[Int] = List(5, 3, 2, 1)
```

:: List List

::

Scala 1 1 :: Nil

:

```
scala> 1 :: 2 :: 3 :: 4 :: Nil
res13: List[Int] = List(1, 2, 3, 4)
```

```
scala> Nil.::(4).::(3).::(2).::(1)
res14: List[Int] = List(1, 2, 3, 4)
```

List

List

++ List

++ List REPL

```
scala> List(1, 2) ++ List(3, 4)
res15: List[Int] = List(1, 2, 3, 4)

scala> List(1) ++ List(3, 4, 5)
res16: List[Int] = List(1, 3, 4, 5)

scala> List(3, 4, 5) ++ List(1)
res17: List[Int] = List(3, 4, 5, 1)
```

++ 1 :

```
scala> List(1, 2) ++ List(3, 4)
res18: List[Int] = List(1, 2, 3, 4)
```

```
scala> List(1, 2).++(List(3, 4))
res19: List[Int] = List(1, 2, 3, 4)
```

List

mkString

Scala Scala

3

mkString List

```
scala> List(1, 2, 3, 4, 5).mkString
res20: String = 12345
```

mkString ()

```
scala> List(1, 2, 3, 4, 5).mkString()
<console>:13: error: overloaded method value mkString with alternatives:
  => String <and>
  (sep: String)String <and>
  (start: String,sep: String,end: String)String
 cannot be applied to ()
       List(1, 2, 3, 4, 5).mkString()
                           ^
```

Scala 0 () ()

() () ()

Scala

mkString(sep: String) sep List sep

```
scala> List(1, 2, 3, 4, 5).mkString(",")
res22: String = 1,2,3,4,5
```

mkString(start: String, sep: String, end: String) mkString(sep)

start end

```
scala> List(1, 2, 3, 4, 5).mkString("[", ",", "]")
res23: String = [1,2,3,4,5]
```

mkString start end

start,...,end

joinByComma Range

mkString

```
def joinByComma(start: Int, end: Int): String = {
  ???
}
```

```
def joinByComma(start: Int, end: Int): String = {
  (start to end).mkString(",")
}
```

```
scala> joinByComma(1, 10)
res24: String = 1,2,3,4,5,6,7,8,9,10
```

foldLeft

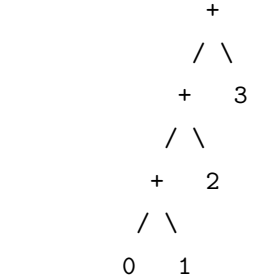
foldLeft List foldLeft

foldLeft Scala API

```
def foldLeft[B](z: B)(f: (B, A)    B): B
```

z foldLeft f

foldLeft List(1, 2, 3).foldLeft(0)((x, y) => x + y)



```

- / \ 0 1 “‘

+ 0 1 f

fold foldLeft

List

```
scala> List(1, 2, 3).foldLeft(0)((x, y) => x + y)
res25: Int = 6
```

List

```
scala> List(1, 2, 3).foldLeft(1)((x, y) => x * y)
res26: Int = 6
```

*10 foldLeft

foldLeft List reverse

```
def reverse[T](list: List[T]): List[T] = list.foldLeft(Nil: List[T])((a, b) => b :: a)
```

```
scala> reverse(List(1, 2, 3, 4, 5))
res27: List[Int] = List(5, 4, 3, 2, 1)
```

foldRight

foldLeft List foldRight

foldRight Scala API

```
def foldRight[B](z: B)(op: (A, B)  B): B
```

foldRight op foldLeft

foldRight List(1, 2, 3).foldRight(0)((y, x) => y + x)

*10 foldLeft
```

```
  +
 / \
1   +
   / \
  2   +
     / \
    3   0
```

foldLeft foldRight

foldRight

List sum foldRight

sum List 0

```
def sum(list: List[Int]): Int = ???
```

```
def sum(list: List[Int]): Int = list.foldRight(0){(x, y) => x + y}
```

```
scala> sum(List(1, 2, 3, 4, 5))
res28: Int = 15
```

List mul foldRight

mul List 1

```
scala> def mul(list: List[Int]): Int = ???
mul: (list: List[Int])Int
```

```
def mul(list: List[Int]): Int = list.foldRight(1){(x, y) => x * y}
```

```
scala> mul(List(1, 2, 3, 4, 5))
res29: Int = 120
```

mkString mkString

foldLeft foldRight List

List API mkString

```
def mkString[T](list: List[T])(sep: String): String = ???
```

2 mkString

```
def mkString[T](list: List[T])(sep: String): String = list match {
  case Nil => ""
  case x::xs => xs.foldLeft(x.toString){ case (x, y) => x + sep + y }
}
```

map List

map 1

List List(1, 2, 3, 4, 5) 2

```
scala> List(1, 2, 3, 4, 5).map(x => x * 2)
res30: List[Int] = List(2, 4, 6, 8, 10)
```

x => x * 2

Scala List

map Scala

map foldLeft reverse

```
def map[T, U](list: List[T])(f: T => U): List[U] = {
  list.foldLeft(Nil:List[U]){(x, y) => f(y) :: x}.reverse
}
```

filter List

filter Boolean 1 true

List List(1, 2, 3, 4, 5)

```
scala> List(1, 2, 3, 4, 5).filter(x => x % 2 == 1)
res31: List[Int] = List(1, 3, 5)
```

filter foldLeft reverse

scala def filter[T](list: List[T])(f: T => Boolean): List[T] = { list.foldLeft(Nil:List[T]){(x, y) => if(f(y)) y::x else x}.reverse }

find

find Boolean 1
true Some Option 1
None Option List(1, 2, 3, 4, 5)

```
scala> List(1, 2, 3, 4, 5).find(x => x % 2 == 1)
res32: Option[Int] = Some(1)
```

Option Scala

takeWhile

takeWhile Boolean 1
true List List(1, 2, 3, 4, 5) 5 4

```
scala> List(1, 2, 3, 4, 5).takeWhile(x => x != 5)
res33: List[Int] = List(1, 2, 3, 4)
```

count List

count Boolean 1
true List(1, 2, 3, 4, 5) 2

```
scala> List(1, 2, 3, 4, 5).count(x => x % 2 == 0)
res34: Int = 2
```

count foldLeft

```
def count(list: List[Int])(f: Int => Boolean): Int  = {
  list.foldLeft(0){(x, y) => if(f(y)) x + 1 else x}
}
```

flatMap List

flatMap

flatMap Scala API

```
final def flatMap[B](f: (A)  GenTraversableOnce[B]): List[B]
```

GenTraversableOnce[B]

flatMap f

(A) => GenTraversableOnce[B] flatMap f

List

```
scala> List(List(1, 2, 3), List(4, 5)).flatMap{e => e.map{g => g + 1}}
res35: List[Int] = List(2, 3, 4, 5, 6)
```

List flatMap map List 1

```
scala> List(1, 2, 3).flatMap{e => List(4, 5).map(g => e * g)}
res36: List[Int] = List(4, 5, 8, 10, 12, 15)
```

List(1, 2, 3) List(4, 5) 2 List

List for-comprehension

```
for(x <- col1; y <- col2;) yield z
```

```
col1.flatMap{x => col2.map{y => z}}
```

flatMap map

for

List List List

List

List

List

List

```
scala> List(1, 2, 3, 4)
res37: List[Int] = List(1, 2, 3, 4)

scala> 5 :: List(1, 2, 3, 4) // List
res38: List[Int] = List(5, 1, 2, 3, 4)

scala> List(1, 2, 3, 4) :+ 5 //                    List
res39: List[Int] = List(1, 2, 3, 4, 5)
```

mkString List List

Range Array

List

Set Set

Scala API

Vector

Vector Vector

immutable

immutable

Vector

```
scala> Vector(1, 2, 3, 4, 5) //
res40: scala.collection.immutable.Vector[Int] = Vector(1, 2, 3, 4, 5)

scala> 6 +: Vector(1, 2, 3, 4, 5)
res41: scala.collection.immutable.Vector[Int] = Vector(6, 1, 2, 3, 4, 5)

scala> Vector(1, 2, 3, 4, 5) :+ 6
res42: scala.collection.immutable.Vector[Int] = Vector(1, 2, 3, 4, 5, 6)

scala> Vector(1, 2, 3, 4, 5).updated(2, 5)
res43: scala.collection.immutable.Vector[Int] = Vector(1, 2, 5, 4, 5)
```

Map

Map

Scala Map immutable Map
mutable Map 2

scala.collection.immutable.Map

Scala Map scala.collection.immutable.Map

scala.collection.immutable.HashMap scala.collection.immutable.TreeMap 2
HashMap

```
scala> val m = Map("A" -> 1, "B" -> 2, "C" -> 3)
m: scala.collection.immutable.Map[String,Int] = Map(A -> 1, B -> 2, C -> 3)

scala> m.updated("B", 4) //       Map
res44: scala.collection.immutable.Map[String,Int] = Map(A -> 1, B -> 4, C -> 3)

scala> m //     Map
res45: scala.collection.immutable.Map[String,Int] = Map(A -> 1, B -> 2, C -> 3)
```

scala.collection.mutable.Map

Scala Map scala.collection.mutable.Map
scala.collection.mutable.HashMap scala.collection.mutable.LinkedHashMap
scala.collection.mutable.ListMap HashMap

```
scala> import scala.collection.mutable
import scala.collection.mutable

scala> val m = mutable.Map("A" -> 1, "B" -> 2, "C" -> 3)
m: scala.collection.mutable.Map[String,Int] = Map(A -> 1, C -> 3, B -> 2)

scala> m("B") = 5 // B -> 5

scala> m //
res47: scala.collection.mutable.Map[String,Int] = Map(A -> 1, C -> 3, B -> 5)
```

Set

Set Set 2
Int Set 1 2 REPL Set

```
scala> Set(1, 1, 2, 3, 4)
res48: scala.collection.immutable.Set[Int] = Set(1, 2, 3, 4)
```

1 1 1

scala.collection.immutable.Set

Scala Set scala.collection.immutable.Set
immutable Map
scala.collection.immutable.HashSet scala.collection.immutable.TreeSet
2 HashSet

```
scala> val s = Set(1, 2, 3, 4, 5)
s: scala.collection.immutable.Set[Int] = Set(5, 1, 2, 3, 4)

scala> s - 5 // 5
res49: scala.collection.immutable.Set[Int] = Set(1, 2, 3, 4)

scala> s //     Set
res50: scala.collection.immutable.Set[Int] = Set(5, 1, 2, 3, 4)
```

scala.collection.mutable.Set

Scala Set scala.collection.mutable.Set
scala.collection.mutable.HashSet scala.collection.mutable.TreeSet
HashSet

```
scala> import scala.collection.mutable
import scala.collection.mutable

scala> val s = mutable.Set(1, 2, 3, 4, 5)
s: scala.collection.mutable.Set[Int] = Set(1, 5, 2, 3, 4)

scala> s -= 5 // 5
res51: s.type = Set(1, 2, 3, 4)
```

```
scala> s //
res52: scala.collection.mutable.Set[Int] = Set(1, 2, 3, 4)
```

http://docs.scala-lang.org/ja/overviews/collections/introduction.html

Scala C Java switch

case class

```
sealed abstract class DayOfWeek
case object Sunday extends DayOfWeek
case object Monday extends DayOfWeek
case object Tuesday extends DayOfWeek
case object Wednesday extends DayOfWeek
case object Thursday extends DayOfWeek
case object Friday extends DayOfWeek
case object Saturday extends DayOfWeek
```

C Java enum

DayOfWeek Sunday

```
val x: DayOfWeek = Sunday
```

object

DayOfWeek object

```
  match {
  case pat1 =>
  case pat2 =>
  ...
}
```

DayOfWeek

```
scala> x match {
     |   case Sunday => 1
     |   case Monday => 2
     |   case Tuesday => 3
     |   case Wednesday => 4
     |   case Thursday => 5
     |   case Friday => 6
     | }
<console>:21: warning: match may not be exhaustive.
It would fail on the following input: Saturday
       x match {
       ^
res0: Int = 1
```

x Sunday 1 Monday 2 ...

sealed

/ /

sealed

/ sealed

C Java

Exp

lhs rhs

Lit Int

Scala

```
sealed abstract class Exp
case class Add(lhs: Exp, rhs: Exp) extends Exp
case class Sub(lhs: Exp, rhs: Exp) extends Exp
case class Mul(lhs: Exp, rhs: Exp) extends Exp
case class Div(lhs: Exp, rhs: Exp) extends Exp
case class Lit(value: Int) extends Exp
```

case

1 + ((2 * 3) / 2)

```
scala> val example = Add(Lit(1), Div(Mul(Lit(2), Lit(3)), Lit(2)))
example: Add = Add(Lit(1),Div(Mul(Lit(2),Lit(3)),Lit(2)))
```

example

```
scala> def eval(exp: Exp): Int = exp match {
     |   case Add(l, r) => eval(l) + eval(r)
     |   case Sub(l, r) => eval(l) - eval(r)
     |   case Mul(l, r) => eval(l) * eval(r)
     |   case Div(l, r) => eval(l) / eval(r)
     |   case Lit(v) => v
     | }
eval: (exp: Exp)Int
```

REPL eval(example)

```
scala> eval(example)
res1: Int = 4
```

1 + ((2 * 3) / 2)

1.
2.
3.

3

match case Lit(v) => v

DayOfWeek

```
<console>:16: warning: match may not be exhaustive.
It would fail on the following input: Lit(_)
       def eval(exp: Exp): Int = exp match {
```

match

Point

```
scala> case class Point(x: Int, y: Int)
defined class Point
```

Point

```
scala> val Point(x, y) = Point(10, 20)
x: Int = 10
y: Int = 20
```

x 10 y 20

scala.MatchError

DayOfWeek nextDayOfWeek

```
def nextDayOfWeek(d: DayOfWeek): DayOfWeek = ???
```

```
def nextDayOfWeek(d: DayOfWeek): DayOfWeek = d match {
  case Sunday => Monday
  case Monday => Tuesday
  case Tuesday => Wednesday
  case Wednesday => Thursday
  case Thursday => Friday
  case Friday => Saturday
  case Saturday => Sunday
}
```

```
scala> nextDayOfWeek(Sunday)
res2: DayOfWeek = Monday

scala> nextDayOfWeek(Monday)
res3: DayOfWeek = Tuesday

scala> nextDayOfWeek(Saturday)
res4: DayOfWeek = Sunday
```

2 Tree Branch, Empty

```
sealed abstract class Tree
case class Branch(value: Int, left: Tree, right: Tree) extends Tree
case object Empty extends Tree
```

2 2 3 1

```
scala> val tree: Tree = Branch(1, Branch(2, Empty, Empty), Branch(3, Empty, Empty))
tree: Tree = Branch(1,Branch(2,Empty,Empty),Branch(3,Empty,Empty))
```

1. max
2. min
3. depth

```
def max(tree: Tree): Int = ???
def min(tree: Tree): Int = ???
def depth(tree: Tree): Int = ???
```

```
depth(Empty) == 0
depth(Branch(10, Empty, Empty)) = 1
```

<= <

sort

```
def sort(tree: Tree): Tree = ???
```

sort

```
object BinaryTree {
  sealed abstract class Tree
  case class Branch(value: Int, left: Tree, right: Tree) extends Tree
  case object Empty extends Tree

  def max(t: Tree): Int = t match {
    case Branch(v1, Branch(v2, Empty, Empty), Branch(v3, Empty, Empty)) =>
      val m = if(v1 <= v2) v2 else v1
      if(m <= v3) v3 else m
    case Branch(v1, Branch(v2, Empty, Empty), Empty) => if(v1 <= v2) v2 else v1
    case Branch(v1, Empty, Branch(v2, Empty, Empty)) => if(v1 <= v2) v2 else v1
    case Branch(v, l, r) =>
      val m1 = max(l)
      val m2 = max(r)
      val m3 = if(m1 <= m2) m2 else m1
      if(v <= m3) m3 else v
    case Empty => throw new RuntimeException
  }


  def min(t: Tree): Int = t match {
    case Branch(v1, Branch(v2, Empty, Empty), Branch(v3, Empty, Empty)) =>
      val m = if(v1 >= v2) v2 else v1
      if(m >= v3) v3 else m
    case Branch(v1, Branch(v2, Empty, Empty), Empty) => if(v1 >= v2) v2 else v1
    case Branch(v1, Empty, Branch(v2, Empty, Empty)) => if(v1 >= v2) v2 else v1
    case Branch(v, l, r) =>
      val m1 = min(l)
      val m2 = min(r)
      val m3 = if(m1 > m2) m2 else m1
      if(v >= m3) m3 else v
    case Empty => throw new RuntimeException
  }

  def depth(t: Tree): Int = t match {
    case Empty => 0
    case Branch(_, l, r) =>
      val ldepth = depth(l)
      val rdepth = depth(r)
      (if(ldepth < rdepth) rdepth else ldepth) + 1
  }

  def sort(t: Tree): Tree = {
    def fromList(list: List[Int]): Tree = {
      def insert(value: Int, t: Tree): Tree = t match {
        case Empty => Branch(value, Empty, Empty)
        case Branch(v, l, r) =>
          if(value <= v) Branch(v, insert(value, l), r)
          else Branch(v, l, insert(value, r))
      }
      list.foldLeft(Empty:Tree){ case (t, v) => insert(v, t) }
    }
```

```
    def toList(tree: Tree): List[Int] = tree match {
      case Empty => Nil
      case Branch(v, l, r) => toList(l) ++ List(v) ++ toList(r)
    }
    fromList(toList(t))
  }

  def find(t: Tree, target: Int): Boolean = t match {
    case Branch(v, l, r) => if(v == target) true else (find(l, target) || find(r, target))
    case Empty => false
  }

  def findBinaryTree(t: Tree, target: Int): Boolean = t match {
    case Branch(v, l, r) => if(v == target) true else (if(target <= v) findBinaryTree(l, target) else findBinaryTree(r
    case Empty => false
  }
}
```

Scala Scala
Option Either Try 2

1

-
-
-

-
- Cookie
-

- Twitter Facebook
- iPhone Android
-

-
- MySQL Redis
-
-

1

Java

Java Scala Java

null

Java
null Java
null
null
NullPointerException NPE
2ch

`null` `null`
Java `null`
`null`
Scala Option

Java

Java `throws`

catch

catch

Java HTTP
MySQL HTTPException
SQLException catch
HTTPException SQLException

catch
catch

API Scala

API

Java Scala

catch

catch catch

catch

Scala

catch

Scala Java

-
-
-

Scala 1 Scala

Scala

1

Scala

Option

Option Scala 1 Java null

Option 1 Option

Option Option Option

- Some
- None

2 Some Option None

Option

Option

```
scala> val o: Option[String] = Option("hoge")
o: Option[String] = Some(hoge)

scala> o.get
res0: String = hoge

scala> o.isEmpty
res1: Boolean = false

scala> o.isDefined
res2: Boolean = true
```

null Option

```
scala> val o: Option[String] = Option(null)
o: Option[String] = None

scala> o.isEmpty
res3: Boolean = true

scala> o.isDefined
res4: Boolean = false
```

```
scala> o.get
java.util.NoSuchElementException: None.get
  at scala.None$.get(Option.scala:347)
  at scala.None$.get(Option.scala:345)
  ... 946 elided
```

Option apply null
null None get
java.util.NoSuchElementException NPE
Option

```
scala> o.getOrElse("")
res6: String = ""
```

Option[String] None

```
scala> o.getOrElse(throw new RuntimeException("null "))
java.lang.RuntimeException: null
  at $anonfun$1.apply(<console>:14)
  at $anonfun$1.apply(<console>:14)
  at scala.Option.getOrElse(Option.scala:121)
  ... 1014 elided
```

Option Option

```
scala> val s: Option[String] = Some("hoge")
s: Option[String] = Some(hoge)

scala> s match {
     |   case Some(str) => println(str)
```

```
     |   case None => throw new RuntimeException
     | }
hoge
```

Some None Some

str

List

Option

Option Option

```
scala> Some(3).map(_ * 3)
res9: Option[Int] = Some(9)
```

map None

```
scala> val n: Option[Int] = None
n: Option[Int] = None

scala> n.map(_ * 3)
res10: Option[Int] = None
```

None None 3

None Option Some

None

Java

```
scala> if (n.isDefined) {
     |   n.get * 3
     | } else {
     |   throw new RuntimeException
     | }
java.lang.RuntimeException
  ... 1024 elided
```

Java

map map Some

None Int

None fold fold Scala API

```
fold[B](ifEmpty:  B)(f: (A)  B): B
```

```
scala> n.fold(throw new RuntimeException)(_ * 3)
java.lang.RuntimeException
  at $anonfun$2.apply(<console>:14)
  at $anonfun$2.apply(<console>:14)
  at scala.Option.fold(Option.scala:158)
  ... 1021 elided
```

None

```
scala> Some(3).fold(throw new RuntimeException)(_ * 3)
res13: Int = 9
```

Some(3) Int 9

Option Option

Scala Option

1 2

Option

```
scala> val v1: Option[Int] = Some(3)
v1: Option[Int] = Some(3)

scala> val v2: Option[Int] = Some(5)
v2: Option[Int] = Some(5)

scala> v1.map(i1 => v2.map(i2 => i1 * i2))
res14: Option[Option[Int]] = Some(Some(15))
```

map ...

Option[Option[Int]] Option

option flatten

```
scala> v1.map(i1 => v2.map(i2 => i1 * i2)).flatten
res15: Option[Int] = Some(15)
```

flatten Option
v2 None flatten

```
scala> val v1: Option[Int] = Some(3)
v1: Option[Int] = Some(3)

scala> val v2: Option[Int] = None
v2: Option[Int] = None

scala> v1.map(i1 => v2.map(i2 => i1 * i2)).flatten
res16: Option[Int] = None
```

Some

{#error_handling_ex1} map flatten Some(2) Some(3) Some(5)
Some(7) Some(11) Some(2310)

```
val v1: Option[Int] = Some(2)
val v2: Option[Int] = Some(3)
val v3: Option[Int] = Some(5)
val v4: Option[Int] = Some(7)
val v5: Option[Int] = Some(11)
v1.map { i1 =>
    v2.map { i2 =>
        v3.map { i3 =>
            v4.map { i4 =>
                v5.map { i5 => i1 * i2 * i3 * i4 * i5 }
            }.flatten
        }.flatten
    }.flatten
}.flatten
```

flatMap

map flatten
2 flatMap
Option flatMap Option map
flatten

Some(3) Some(5)

```
scala> val v1: Option[Int] = Some(3)
v1: Option[Int] = Some(3)

scala> val v2: Option[Int] = Some(5)
v2: Option[Int] = Some(5)

scala> v1.flatMap(i1 => v2.map(i2 => i1 * i2))
res18: Option[Int] = Some(15)
```

`Some(3)` `Some(5)` `Some(7)`

```
scala> val v1: Option[Int] = Some(3)
v1: Option[Int] = Some(3)

scala> val v2: Option[Int] = Some(5)
v2: Option[Int] = Some(5)

scala> val v3: Option[Int] = Some(7)
v3: Option[Int] = Some(7)

scala> v1.flatMap(i1 => v2.flatMap(i2 => v3.map(i3 => i1 * i2 * i3)))
res19: Option[Int] = Some(105)
```

v1, v2, v3 None `flatten`

`None`

```
scala> val v3: Option[Int] = None
v3: Option[Int] = None

scala> v1.flatMap(i1 => v2.flatMap(i2 => v3.map(i3 => i1 * i2 * i3)))
res20: Option[Int] = None
```

{#error_handling_ex2} `flatMap` `Some(2)` `Some(3)` `Some(5)` `Some(7)` `Some(11)` `Some(2310)`

```
val v1: Option[Int] = Some(2)
val v2: Option[Int] = Some(3)
val v3: Option[Int] = Some(5)
val v4: Option[Int] = Some(7)
val v5: Option[Int] = Some(11)
v1.flatMap { i1 =>
    v2.flatMap { i2 =>
```

```
        v3.flatMap { i3 =>
            v4.flatMap { i4 =>
                v5.map { i5 => i1 * i2 * i3 * i4 * i5 }
            }
        }
    }
}
```

for flatMap

Option for Option

for flatMap map

Some(3) Some(5) Some(7)

flatMap for

```
scala> val v1: Option[Int] = Some(3)
v1: Option[Int] = Some(3)

scala> val v2: Option[Int] = Some(5)
v2: Option[Int] = Some(5)

scala> val v3: Option[Int] = Some(7)
v3: Option[Int] = Some(7)

scala> for { i1 <- v1
     |       i2 <- v2
     |       i3 <- v3 } yield i1 * i2 * i3
res22: Option[Int] = Some(105)
```

for flatMap map flatMap

map for

{#error_handling_ex3} for Some(2) Some(3) Some(5) Some(7)

Some(11) Some(2310)

```
val v1: Option[Int] = Some(2)
val v2: Option[Int] = Some(3)
val v3: Option[Int] = Some(5)
val v4: Option[Int] = Some(7)
val v5: Option[Int] = Some(11)
for { i1 <- v1
      i2 <- v2
      i3 <- v3
      i4 <- v4
```

```
      i5 <- v5 } yield i1 * i2 * i3 * i4 * i5
```

Either

Option null Option

None

Option Either Option

Either 2

Option Some None 2 Either

Right Left 2

```
scala> val v1: Either[String, Int] = Right(123)
v1: Either[String,Int] = Right(123)

scala> val v2: Either[String, Int] = Left("abc")
v2: Either[String,Int] = Left(abc)
```

Option

```
scala> v1 match {
     |   case Right(i) => println(i)
     |   case Left(s)  => println(s)
     | }
123
```

Either Either Left Right

"right"

Left sealed trait case class

Throwable Try

Either Left LoginError

sealed

```
sealed trait LoginError
//
case object InvalidPassword extends LoginError
```

```
// name
case object UserNotFound extends LoginError
//
case object PasswordLocked extends LoginError
```

API

```
case class User(id: Long, name: String, password: String)

object LoginService {
  def login(name: String, password: String): Either[LoginError, User] = ???
}
```

`login` `User`
Either Right `LoginError` Either Left

`login`

```
LoginService.login(name = "dwango", password = "password") match {
  case Right(user) => println(s"id: ${user.id}")
  case Left(InvalidPassword) => println(s"Invalid Password!")
}
```

`println`
Left `InvalidPassword` `UserNotFound`
`PasswordLocked`

```
<console>:11: warning: match may not be exhaustive.
It would fail on the following inputs: Left(PasswordLocked), Left(UserNotFound)
             LoginService.login(name = "dwango", password = "password") match {
                                ^
```

`Left(PasswordLocked)` `Left(UserNotFound)`
warning Either

Either map flatMap Either
Option Scala Either map flatMap
Scala Either Option map flatMap Either

```
scala> val v: Either[String, Int] = Right(123)
v: Either[String,Int] = Right(123)

scala> v.
asInstanceOf   fold   isInstanceOf   isLeft   isRight   joinLeft   joinRight   left   right   swap   toString
```

`fold` `isLeft` `isRight` Option map flatMap for Either

Scala Either map map Either map 2

- Right Right
-

Haskell Scala Either `left` `right`

`right`

```
scala> val v: Either[String, Int] = Right(123)
v: Either[String,Int] = Right(123)

scala> val e = v.right
e: scala.util.Either.RightProjection[String,Int] = RightProjection(Right(123))

scala> e.
asInstanceOf   canEqual   copy   e   exists   filter   flatMap   forall   foreach   get   getOrElse   isInstanceOf
```

Either `right` `RightProjection` `RightProjection` List Option Either Right

`RightProjection` `map` [*11]

[*11] Scala2.11 `Product with Serializable with scala.util.Either` Scala2.12 https://github.com/scala/scala/pull/4355 https://issues.scala-lang.org/browse/SI-9173

```
scala> val v: Either[String, Int] = Right(123)
v: Either[String,Int] = Right(123)

scala> v.right.map(_ * 2)
res25: Product with Serializable with scala.util.Either[String,Int] = Right(246)
```

map Right Either Left
Option None map

RightProjection map Either
map flatMap Either RightProjection

Option Either

```
scala> val v1: Either[String, Int] = Right(3)
v1: Either[String,Int] = Right(3)

scala> val v2: Either[String, Int] = Right(5)
v2: Either[String,Int] = Right(5)

scala> val v3: Either[String, Int] = Right(7)
v3: Either[String,Int] = Right(7)

scala> for {
     |   i1 <- v1.right
     |   i2 <- v2.right
     |   i3 <- v3.right
     | } yield i1 * i2 * i3
res26: scala.util.Either[String,Int] = Right(105)
```

Right Either RightProjection
Scala Either Haskell Either
Right Either map
flatMap right RightProjection

Try

Scala Try Either Either
2 Throwable 1
Try 2

- Success

- Failure

Success Failure Throwable
Try apply
catch Failure

```
scala> import scala.util.Try
import scala.util.Try

scala> val v: Try[Int] = Try(throw new RuntimeException("to be caught"))
v: scala.util.Try[Int] = Failure(java.lang.RuntimeException: to be caught)
```

Try Failure
Try

Try Either map flatMap

```
scala> val v1 = Try(3)
v1: scala.util.Try[Int] = Success(3)

scala> val v2 = Try(5)
v2: scala.util.Try[Int] = Success(5)

scala> val v3 = Try(7)
v3: scala.util.Try[Int] = Success(7)

scala> for {
     |   i1 <- v1
     |   i2 <- v2
     |   i3 <- v3
     | } yield i1 * i2 * i3
res27: scala.util.Try[Int] = Success(105)
```

NonFatal Try.apply catch NonFatal
NonFatal
NonFatal catch

Try

```
import scala.util.control.NonFatal

try {
  ???
```

```
} catch {
  case NonFatal(e) => //
}
```

Option Either Try

Option Either Try

Java null Option

Option

Either Option

Java

Either

Try Java

Option Either

Scala

None flatMap for

None

match case

ID

ID

-

-
-
-

4

```
object MainBefore {

  case class Address(id: Int, name: String, postalCode: Option[String])
  case class User(id: Int, name: String, addressId: Option[Int])

  val userDatabase: Map[Int, User] = Map (
    1 -> User(1, "    ", Some(1)),
    2 -> User(2, "    ", Some(2)),
    3 -> User(3, "       ", None)
  )

  val addressDatabase: Map[Int, Address] = Map (
    1 -> Address(1, "    ", Some("150-0002")),
    2 -> Address(2, "                  ", None)
  )

  sealed abstract class PostalCodeResult
  case class Success(postalCode: String) extends PostalCodeResult
  abstract class Failure extends PostalCodeResult
  case class UserNotFound() extends Failure
  case class UserNotHasAddress() extends Failure
  case class AddressNotFound() extends Failure
  case class AddressNotHasPostalCode() extends Failure

  //       None                          for
  def getPostalCodeResult(userId: Int): PostalCodeResult = {
    findUser(userId) match {
      case Some(user) =>
        user.addressId match {
          case Some(addressId) =>
            findAddress(addressId) match {
              case Some(address) =>
                address.postalCode match {
                  case Some(postalCode) => Success(postalCode)
                  case None => AddressNotHasPostalCode()
                }
              case None => AddressNotFound()
            }
          case None => UserNotHasAddress()
        }
      case None => UserNotFound()
    }
  }
```

```
  def findUser(userId: Int): Option[User] = {
    userDatabase.get(userId)
  }

  def findAddress(addressId: Int): Option[Address] = {
    addressDatabase.get(addressId)
  }

  def main(args: Array[String]): Unit = {
    println(getPostalCodeResult(1)) // Success(150-0002)
    println(getPostalCodeResult(2)) // AddressNotHasPostalCode()
    println(getPostalCodeResult(3)) // UserNotHasAddress()
    println(getPostalCodeResult(4)) // UserNotFound()
  }
}
```

getPostalCodeResult match case

Either

find Either Failure Left Right

find Failure

```
object MainRefactored {

  case class Address(id: Int, name: String, postalCode: Option[String])
  case class User(id: Int, name: String, addressId: Option[Int])

  val userDatabase: Map[Int, User] = Map (
    1 -> User(1, "    ", Some(1)),
    2 -> User(2, "    ", Some(2)),
    3 -> User(3, "       ", None)
  )

  val addressDatabase: Map[Int, Address] = Map (
    1 -> Address(1, "    ", Some("150-0002")),
    2 -> Address(2, "                  ", None)
  )

  sealed abstract class PostalCodeResult
  case class Success(postalCode: String) extends PostalCodeResult
  abstract class Failure extends PostalCodeResult
  case class UserNotFound() extends Failure
  case class UserNotHasAddress() extends Failure
  case class AddressNotFound() extends Failure
  case class AddressNotHasPostalCode() extends Failure
```

```
  //
  def getPostalCodeResult(userId: Int): PostalCodeResult = {
    (for {
      user <- findUser(userId).right
      address <- findAddress(user).right
      postalCode <- findPostalCode(address).right
    } yield Success(postalCode)).merge
  }

  def findUser(userId: Int): Either[Failure, User] = {
    userDatabase.get(userId).toRight(UserNotFound())
  }

  def findAddress(user: User): Either[Failure, Address] = {
    for {
      addressId <- user.addressId.toRight(UserNotHasAddress()).right
      address <- addressDatabase.get(addressId).toRight(AddressNotFound()).right
    } yield address
  }

  def findPostalCode(address: Address): Either[Failure, String] = {
    address.postalCode.toRight(AddressNotHasPostalCode())
  }

  def main(args: Array[String]): Unit = {
    println(getPostalCodeResult(1)) // Success(150-0002)
    println(getPostalCodeResult(2)) // AddressNotHasPostalCode()
    println(getPostalCodeResult(3)) // UserNotHasAddress()
    println(getPostalCodeResult(4)) // UserNotFound()
  }
}
```

```
  def getPostalCodeResult(userId: Int): PostalCodeResult = {
    (for {
      user <- findUser(userId).right
      address <- findAddress(user).right
      postalCode <- findPostalCode(address).right
    } yield Success(postalCode)).merge
  }
```

getPostalCodeResult

Either for .right

RightProjection for merge

implicit conversion implicit parameter

Scala implicit conversion implicit parameter
2 Scala
implicit conversion implicit parameter
2 2

Implicit Conversion

implicit conversion

implicit conversion

```
implicit def ( : ): =
```

implicit 1 *12
implicit conversion

implicit conversion 1

```
scala> implicit def intToBoolean(arg: Int): Boolean = arg != 0
warning: there was one feature warning; re-run with -feature for details
intToBoolean: (arg: Int)Boolean

scala> if(1) {
     |   println("1 ")
     | }
1
```

Boolean if Int

*12 2 implicit def implicit def implicit
implicit def 2
implicit conversion

implicit conversion

if Boolean

Scala implicit conversion

pimp my library

1 pimp my library

Scala

(1 to 5)

Int to to pimp my library

implicit conversion

implicit conversion implicit conversion

implicit conversion

String ":-)" implicit conversion

```
scala> class RichString(val src: String) {
     |   def smile: String = src + ":-)"
     | }
defined class RichString

scala> implicit def enrichString(arg: String): RichString = new RichString(arg)
warning: there was one feature warning; re-run with -feature for details
enrichString: (arg: String)RichString

scala> "Hi, ".smile
res1: String = Hi, :-)
```

":-)" smile

implicit conversion

Scala 2.10 class implicit

Scala 2.11

Implicit Class

Scala 2.10

```
scala> implicit class RichString(val src: String) {
     |   def smile: String = src + ":-)"
     | }
defined class RichString

scala> "Hi, ".smile
res0: String = Hi, :-)
```

implicit class pimp my library

implicit def Scala 2.10 pimp my library

{#implicit_ex1}

`Int` `Boolean` implicit conversion

implicit conversion

{#implicit_ex2}

pimp my library implicit conversion 1

```
object Taps {
  implicit class Tap[T](self: T) {
    def tap[U](block: T => U): T = {
      block(self) //
      self
    }
  }

  def main(args: Array[String]): Unit = {
    "Hello, World".tap{s => println(s)}.reverse.tap{s => println(s)}
  }
}
```

```
scala> import Taps._
import Taps._

scala> Taps.main(Array())
Hello, World
dlroW ,olleH
```

{#implicit_ex3} Scala pimp my library
1

Implicit Parameter

implicit parameter 2 1

Connection

Connection

```
def useDatabase1(...., conn: Connection)
def useDatabase2(...., conn: Connection)
def useDatabase3(...., conn: Connection)
```

3 Connection Connection
implicit parameter

```
def useDatabase1(....)(implicit conn: Connection)
def useDatabase2(....)(implicit conn: Connection)
def useDatabase3(....)(implicit conn: Connection)
```

implicit
implicit parameter

```
(implicit conn: Connection)
```

Scala
implicit
implicit

```
implicit val connection: Connection = connectDatabase(....)
```

Connection

implicit parameter Play 2 Framework Scala O/R

implicit parameter 1 List
sum
List +
2

2 0
Additive Additive

```
trait Additive[A] {
  def plus(a: A, b: A): A
  def zero: A
}
```

Additive A List

- zero: A **0**
- plus: A 2

Additive List

```
def sum[A](lst: List[A])(m: Additive[A]) = lst.foldLeft(m.zero)((x, y) => m.plus(x, y))
```

0 object String Int

```
object StringAdditive extends Additive[String] {
  def plus(a: String, b: String): String = a + b
  def zero: String = ""
}

object IntAdditive extends Additive[Int] {
  def plus(a: Int, b: Int): Int = a + b
  def zero: Int = 0
}
```

```
trait Additive[A] {
  def plus(a: A, b: A): A
  def zero: A
```

```
}

object StringAdditive extends Additive[String] {
  def plus(a: String, b: String): String = a + b
  def zero: String = ""
}

object IntAdditive extends Additive[Int] {
  def plus(a: Int, b: Int): Int = a + b
  def zero: Int = 0
}

def sum[A](lst: List[A])(m: Additive[A]) = lst.foldLeft(m.zero)((x, y) => m.plus(x, y))
```

Int List String List sum

```
scala> sum(List(1, 2, 3))(IntAdditive)
res6: Int = 6

scala> sum(List("A", "B", "C"))(StringAdditive)
res7: String = ABC
```

List
IntAdditive, StringAdditive
implicit parameter
StringAdditive IntAdditive implicit
sum m implicit implicit parameter

```
scala> trait Additive[A] {
     |   def plus(a: A, b: A): A
     |   def zero: A
     | }
defined trait Additive

scala> implicit object StringAdditive extends Additive[String] {
     |   def plus(a: String, b: String): String = a + b
     |   def zero: String = ""
     | }
defined object StringAdditive

scala> implicit object IntAdditive extends Additive[Int] {
     |   def plus(a: Int, b: Int): Int = a + b
     |   def zero: Int = 0
     | }
defined object IntAdditive
```

```
scala> def sum[A](lst: List[A])(implicit m: Additive[A]) = lst.foldLeft(m.zero)((x, y) => m.plus(x, y))
sum: [A](lst: List[A])(implicit m: Additive[A])A

scala> sum(List(1, 2, 3))
res8: Int = 6

scala> sum(List("A", "B", "C"))
res9: String = ABC
```

List sum sum(List(1, 2, 3))
implicit parameter Haskell
Haskell Additive
StringAdditive IntAdditive Additive

implicit parameter

```
scala> List[Int]().sum
res10: Int = 0

scala> List(1, 2, 3, 4).sum
res11: Int = 10

scala> List(1.1, 1.2, 1.3, 1.4).sum
res12: Double = 5.0
```

Scala

{#implicit_ex4}

m: Additive[T] t1: T, t2: T, t3: T

```
m.plus(m.zero, t1) == t1  //
m.plus(t1, m.zero) == t1  //
m.plus(t1, m.plus(t2, t3)) == m.plus(m.plus(t1, t2), t3) //
```

T zero plus Additive[T]

Additive[T] implicit T sum

x y

Point

```
object Additives {
  trait Additive[A] {
    def plus(a: A, b: A): A
    def zero: A
  }

  implicit object StringAdditive extends Additive[String] {
    def plus(a: String, b: String): String = a + b
    def zero: String = ""
  }

  implicit object IntAdditive extends Additive[Int] {
    def plus(a: Int, b: Int): Int = a + b
    def zero: Int = 0
  }

  case class Point(x: Int, y: Int)

  implicit object PointAdditive extends Additive[Point] {
    def plus(a: Point, b: Point): Point = Point(a.x + b.x, a.y + b.y)
    def zero: Point = Point(0, 0)
  }

  def sum[A](lst: List[A])(implicit m: Additive[A]) = lst.foldLeft(m.zero)((x, y) => m.plus(x, y))
}
```

```
scala> import Additives._
import Additives._

scala> println(sum(List(Point(1, 1), Point(2, 2), Point(3, 3)))) // Point(6, 6)
Point(6,6)

scala> println(sum(List(Point(1, 2), Point(3, 4), Point(5, 6)))) // Point(9, 12)
Point(9,12)
```

{#implicit_ex5}

`List[Int]` `List[Double]` sum 1

2 Scala

- Numeric[T]

- IntIsIntegral

- DoubleAsIfIntegral

implicit

implicit def implicit parameter

-
- import
-
-

implicit

Rational

Additive

```
case class Rational(num: Int, den: Int)

object Rational {
  implicit object RationalAdditive extends Additive[Rational] {
    def plus(a: Rational, b: Rational): Rational = {
      if (a == zero) {
        b
      } else if (b == zero) {
        a
      } else {
        Rational(a.num * b.den + b.num * a.den, a.den * b.den)
      }
    }
    def zero: Rational = Rational(0, 0)
  }
}
```

import Additive

```
scala> sum(List(Rational(1, 1), Rational(2, 2)))
res0: Rational = Rational(4,2)
```

Implicit

Scala Scala

Scala Scala

Functor

List Option map map

Functor [13]

```
trait Functor[F[_]] {
  def map[A, B](fa: F[A])(f: A => B): F[B]
}
```

2

```
def identityLaw[F[_], A](fa: F[A])(implicit F: Functor[F]): Boolean =
  F.map(fa)(identity) == fa

def compositeLaw[F[_], A, B, C](fa: F[A], f1: A => B, f2: B => C)(implicit F: Functor[F]): Boolean =
  F.map(fa)(f2 compose f1) == F.map(F.map(fa)(f1))(f2)
```

Option Functor

```
scala> import scala.language.higherKinds
import scala.language.higherKinds

scala> trait Functor[F[_]] {
     |   def map[A, B](fa: F[A])(f: A => B): F[B]
     | }
```

[13] F
List Option
VI

```
defined trait Functor

scala> def identityLaw[F[_], A](fa: F[A])(implicit F: Functor[F]): Boolean =
     |   F.map(fa)(identity) == fa
identityLaw: [F[_], A](fa: F[A])(implicit F: Functor[F])Boolean

scala> def compositeLaw[F[_], A, B, C](fa: F[A], f1: A => B, f2: B => C)(implicit F: Functor[F]): Boolean =
     |   F.map(fa)(f2 compose f1) == F.map(F.map(fa)(f1))(f2)
compositeLaw: [F[_], A, B, C](fa: F[A], f1: A => B, f2: B => C)(implicit F: Functor[F])Boolean

scala> implicit object OptionFunctor extends Functor[Option] {
     |   def map[A, B](fa: Option[A])(f: A => B): Option[B] = fa.map(f)
     | }
defined object OptionFunctor

scala> val n: Option[Int] = Some(2)
n: Option[Int] = Some(2)

scala> identityLaw(n)
res1: Boolean = true

scala> compositeLaw(n, (i: Int) => i * i, (i: Int) => i.toString)
res2: Boolean = true
```

Applicative Functor

Functor

1 Applicative Functor

```
trait Applicative[F[_]] {
  def point[A](a: A): F[A]
  def ap[A, B](fa: F[A])(f: F[A => B]): F[B]
}
```

Applicative Functor Functor Applicative Functor

`map`

```
def map[F[_], A, B](fa: F[A])(f: A => B)(implicit F: Applicative[F]): F[B] =
  F.ap(fa)(F.point(f))
```

Applicative Functor

```
def identityLaw[F[_], A](fa: F[A])(implicit F: Applicative[F]): Boolean =
  F.ap(fa)(F.point((a: A) => a)) == fa

def homomorphismLaw[F[_], A, B](f: A => B, a: A)(implicit F: Applicative[F]): Boolean =
  F.ap(F.point(a))(F.point(f)) == F.point(f(a))
```

```
def interchangeLaw[F[_], A, B](f: F[A => B], a: A)(implicit F: Applicative[F]): Boolean =
  F.ap(F.point(a))(f) == F.ap(f)(F.point((g: A => B) => g(a)))
```

ap point map Functor

Option Applicative Functor

```
scala> trait Applicative[F[_]] {
     |   def point[A](a: A): F[A]
     |   def ap[A, B](fa: F[A])(f: F[A => B]): F[B]
     |   def map[A, B](fa: F[A])(f: A => B): F[B] = ap(fa)(point(f))
     | }
defined trait Applicative

scala> def identityLaw[F[_], A](fa: F[A])(implicit F: Applicative[F]): Boolean =
     |   F.ap(fa)(F.point((a: A) => a)) == fa
identityLaw: [F[_], A](fa: F[A])(implicit F: Applicative[F])Boolean

scala> def homomorphismLaw[F[_], A, B](f: A => B, a: A)(implicit F: Applicative[F]): Boolean =
     |   F.ap(F.point(a))(F.point(f)) == F.point(f(a))
homomorphismLaw: [F[_], A, B](f: A => B, a: A)(implicit F: Applicative[F])Boolean

scala> def interchangeLaw[F[_], A, B](f: F[A => B], a: A)(implicit F: Applicative[F]): Boolean =
     |   F.ap(F.point(a))(f) == F.ap(f)(F.point((g: A => B) => g(a)))
interchangeLaw: [F[_], A, B](f: F[A => B], a: A)(implicit F: Applicative[F])Boolean

scala> implicit object OptionApplicative extends Applicative[Option] {
     |   def point[A](a: A): Option[A] = Some(a)
     |   def ap[A, B](fa: Option[A])(f: Option[A => B]): Option[B] = f match {
     |     case Some(g) => fa match {
     |       case Some(a) => Some(g(a))
     |       case None => None
     |     }
     |     case None => None
     |   }
     | }
defined object OptionApplicative

scala> val a: Option[Int] = Some(1)
a: Option[Int] = Some(1)

scala> val f: Int => String = { i => i.toString }
f: Int => String = <function1>

scala> val af: Option[Int => String] = Some(f)
af: Option[Int => String] = Some(<function1>)

scala> identityLaw(a)
res5: Boolean = true
```

```
scala> homomorphismLaw(f, 1)
res6: Boolean = true

scala> interchangeLaw(af, 1)
res7: Boolean = true

scala> OptionApplicative.map(a)(_ + 1) == OptionFunctor.map(a)(_ + 1)
res8: Boolean = true
```

Monad

Applicative Functor
Monad

```
trait Monad[F[_]] {
  def point[A](a: A): F[A]
  def bind[A, B](fa: F[A])(f: A => F[B]): F[B]
}
```

`bind` Option List `flatMap`

Monad

```
def rightIdentityLaw[F[_], A](a: F[A])(implicit F: Monad[F]): Boolean =
  F.bind(a)(F.point(_)) == a

def leftIdentityLaw[F[_], A, B](a: A, f: A => F[B])(implicit F: Monad[F]): Boolean =
  F.bind(F.point(a))(f) == f(a)

def associativeLaw[F[_], A, B, C](fa: F[A], f: A => F[B], g: B => F[C])(implicit F: Monad[F]): Boolean =
  F.bind(F.bind(fa)(f))(g) == F.bind(fa)((a: A) => F.bind(f(a))(g))
```

Monad Applicative Functor Monad `point` `ap`
`point`

```
def ap[F[_], A, B](fa: F[A])(f: F[A => B])(implicit F: Monad[F]): F[B] =
  F.bind(f)((g: A => B) => F.bind(fa)((a: A) => F.point(g(a))))
```

Option

```
scala> trait Monad[F[_]] {
     |   def point[A](a: A): F[A]
     |   def bind[A, B](fa: F[A])(f: A => F[B]): F[B]
```

```
     | }
defined trait Monad

scala> def rightIdentityLaw[F[_], A](a: F[A])(implicit F: Monad[F]): Boolean =
     |   F.bind(a)(F.point(_)) == a
rightIdentityLaw: [F[_], A](a: F[A])(implicit F: Monad[F])Boolean

scala> def leftIdentityLaw[F[_], A, B](a: A, f: A => F[B])(implicit F: Monad[F]): Boolean =
     |   F.bind(F.point(a))(f) == f(a)
leftIdentityLaw: [F[_], A, B](a: A, f: A => F[B])(implicit F: Monad[F])Boolean

scala> def associativeLaw[F[_], A, B, C](fa: F[A], f: A => F[B], g: B => F[C])(implicit F: Monad[F]): Boolean =
     |   F.bind(F.bind(fa)(f))(g) == F.bind(fa)((a: A) => F.bind(f(a))(g))
associativeLaw: [F[_], A, B, C](fa: F[A], f: A => F[B], g: B => F[C])(implicit F: Monad[F])Boolean

scala> implicit object OptionMonad extends Monad[Option] {
     |   def point[A](a: A): Option[A] = Some(a)
     |   def bind[A, B](fa: Option[A])(f: A => Option[B]): Option[B] = fa match {
     |     case Some(a) => f(a)
     |     case None => None
     |   }
     | }
defined object OptionMonad

scala> val fa: Option[Int] = Some(1)
fa: Option[Int] = Some(1)

scala> val f: Int => Option[Int] = { n => Some(n + 1) }
f: Int => Option[Int] = <function1>

scala> val g: Int => Option[Int] = { n => Some(n * n) }
g: Int => Option[Int] = <function1>

scala> rightIdentityLaw(fa)
res11: Boolean = true

scala> leftIdentityLaw(1, f)
res12: Boolean = true

scala> associativeLaw(fa, f, g)
res13: Boolean = true
```

Monoid

2 Monoid

```
trait Monoid[F] {
  def append(a: F, b: F): F
  def zero: F
}
```

Additive Monoid

```
def leftIdentity[F](a: F)(implicit F: Monoid[F]): Boolean = a == F.append(F.zero, a)
def rightIdentity[F](a: F)(implicit F: Monoid[F]): Boolean = a == F.append(a, F.zero)
def associativeLaw[F](a: F, b: F, c: F)(implicit F: Monoid[F]): Boolean = {
  F.append(F.append(a, b), c) == F.append(a, F.append(b, c))
}
```

Option[Int] Monoid

```
scala> trait Monoid[F] {
     |   def append(a: F, b: F): F
     |   def zero: F
     | }
defined trait Monoid

scala> def leftIdentity[F](a: F)(implicit F: Monoid[F]): Boolean = a == F.append(F.zero, a)
leftIdentity: [F](a: F)(implicit F: Monoid[F])Boolean

scala> def rightIdentity[F](a: F)(implicit F: Monoid[F]): Boolean = a == F.append(a, F.zero)
rightIdentity: [F](a: F)(implicit F: Monoid[F])Boolean

scala> def associativeLaw[F](a: F, b: F, c: F)(implicit F: Monoid[F]): Boolean = {
     |   F.append(F.append(a, b), c) == F.append(a, F.append(b, c))
     | }
associativeLaw: [F](a: F, b: F, c: F)(implicit F: Monoid[F])Boolean

scala> implicit object OptionIntMonoid extends Monoid[Option[Int]] {
     |   def append(a: Option[Int], b: Option[Int]): Option[Int] = (a, b) match {
     |     case (None, None) => None
     |     case (Some(v), None) => Some(v)
     |     case (None, Some(v)) => Some(v)
     |     case (Some(v1), Some(v2)) => Some(v1 + v2)
     |   }
     |   def zero: Option[Int] = None
     | }
defined object OptionIntMonoid

scala> val n: Option[Int] = Some(1)
n: Option[Int] = Some(1)

scala> val m: Option[Int] = Some(2)
m: Option[Int] = Some(2)

scala> val o: Option[Int] = Some(3)
o: Option[Int] = Some(3)

scala> leftIdentity(n)
res14: Boolean = true

scala> rightIdentity(n)
```

```
res15: Boolean = true

scala> associativeLaw(n, m, o)
res16: Boolean = true
```

Monoid

Monoid

Future/Promise

Future Promise

Future Promise

Future

JVM

JavaScript

CPU

UI

Concurrent

CPU

Parallel

Future Promise

Future

Future Option

isCompleted onSuccess
onFailure flatMap filter for
Option List

Future

Java Future [14] Option
ECMAScript 6 Promise
Scala Future ECMAScript 6 Promise Scala Promise

```
import scala.concurrent.Future
import scala.concurrent.ExecutionContext.Implicits.global

object FutureSample extends App {

  val s = "Hello"
  val f: Future[String] = Future {
    Thread.sleep(1000)
    s + " future!"
  }

  f.onSuccess { case s: String =>
    println(s)
  }

  println(f.isCompleted) // false

  Thread.sleep(5000) // Hello future!

  println(f.isCompleted) // true
}
```

```
false
Hello future!
true
```

[14] Java 8 java.util.concurrent.Future CompletableFuture

Future sbt
console Future
Future[+T] 1000
"Hello" " future!"
future future 5000

5000 Future
Thread.sleep(5000) Await.ready(f, 5000 millisecond)
Future 5000

```
import scala.concurrent.Await
import scala.concurrent.duration._
import scala.language.postfixOps
```

import

```
import scala.concurrent.{Await, Future}
import scala.concurrent.ExecutionContext.Implicits.global
import scala.concurrent.duration._
import scala.language.postfixOps

object FutureSample extends App {

  val s = "Hello"
  val f: Future[String] = Future {
    Thread.sleep(1000)
    println(s"[ThreadName] In Future: ${Thread.currentThread.getName}")
    s + " future!"

  }

  f.onSuccess { case s: String =>
    println(s"[ThreadName] In onSuccess: ${Thread.currentThread.getName}")
    println(s)
  }

  println(f.isCompleted) // false

  Await.ready(f, 5000 millisecond) // Hello future!

  println(s"[ThreadName] In App: ${Thread.currentThread.getName}")
  println(f.isCompleted) // true
}
```

```
false
[ThreadName] In Future: ForkJoinPool-1-worker-5
[ThreadName] In App: main
true
[ThreadName] In onSuccess: ForkJoinPool-1-worker-5
Hello future!
```

Future onSuccess ForkJoinPool-1-worker-5
main

Future

Await.ready(f, 5000 millisecond)
isCompleted "Hello future!"

ForkJoinPool Java ExecutorService

Future Future Option

```
import scala.concurrent.ExecutionContext.Implicits.global
import scala.concurrent.Future
import scala.util.{Failure, Random, Success}

object FutureOptionUsageSample extends App {
  val random = new Random()
  val waitMaxMilliSec = 3000

  val futureMilliSec: Future[Int] = Future {
    val waitMilliSec = random.nextInt(waitMaxMilliSec);
    if(waitMilliSec < 1000) throw new RuntimeException(s"waitMilliSec is ${waitMilliSec}" )
    Thread.sleep(waitMilliSec)
    waitMilliSec
  }

  val futureSec: Future[Double] = futureMilliSec.map(i => i.toDouble / 1000)

  futureSec onComplete {
    case Success(waitSec) => println(s"Success! ${waitSec} sec")
    case Failure(t) => println(s"Failure: ${t.getMessage}")
  }
```

```
  Thread.sleep(3000)
 }
```

`Success! 1.538 sec  Failure: waitMilliSec is 971`

3000 Future 1000

Future `futureMilliSec` map `Int` `Doubule` `onSuccess` `onComplete`

Future Option map Option flatMap flatMap Future map Future Future `val futureSec: Future[Double] = futureMilliSec.map(i => i.toDouble / 1000)` 100 Future

```
val futureSec: Future[Double] = futureMilliSec.flatMap(i => Future {
  Thread.sleep(100)
  i.toDouble / 1000
})
```

map Option flatten Future Future flatMap

Future

flatMap for Future Future

```
import scala.concurrent.ExecutionContext.Implicits.global
import scala.concurrent.Future
import scala.language.postfixOps
import scala.util.{Failure, Success, Random}

object CompositeFutureSample extends App {
  val random = new Random()
  val waitMaxMilliSec = 3000
```

```
  def waitRandom(futureName: String): Int = {
    val waitMilliSec = random.nextInt(waitMaxMilliSec);
    if(waitMilliSec < 500) throw new RuntimeException(s"${futureName} waitMilliSec is ${waitMilliSec}" )
    Thread.sleep(waitMilliSec)
    waitMilliSec
  }

  val futureFirst: Future[Int] = Future { waitRandom("first") }
  val futureSecond: Future[Int] = Future { waitRandom("second") }

  val compositeFuture: Future[(Int, Int)] = for {
    first: Int <- futureFirst
    second: Int <- futureSecond
  } yield (first, second)

  compositeFuture onComplete  {
    case Success((first, second)) => println(s"Success! first:${first} second:${second}")
    case Failure(t) => println(s"Failure: ${t.getMessage}")
  }

  Thread.sleep(5000)
 }
```

3 500

Future 2

for Future Future

Success! first:1782 second:1227 Failure: first waitMilliSec is 412 Failure: second waitMilliSec is 133

Future filter API

Future

Promise

Promise

Future

Promise

```
import scala.concurrent.ExecutionContext.Implicits.global
import scala.concurrent.{Promise, Future}
import scala.util.{Success, Failure, Random}

object PromiseSample extends App {
  val random = new Random()
  val promiseGetInt: Promise[Int] = Promise[Int]

  val futureGetInt: Future[Int] = promiseGetInt.success(1).future

  futureGetInt.onComplete {
    case Success(i) => println(s"Success! i: ${i}")
    case Failure(t) => println(s"Failure! t: ${t.getMessage}")
  }

  Thread.sleep(1000)
}
```

`Success! i: 1` `promiseGetInt.success(1).future` `promiseGetInt.future` onComplete

1 Future success success trySuccess Promise success `IllegalStateException`

```
import scala.concurrent.ExecutionContext.Implicits.global
import scala.concurrent.{Future, Promise}
import scala.util.{Failure, Random, Success}

object PromiseFutureCompositionSample extends App {
  val random = new Random()
  val promiseGetInt: Promise[Int] = Promise[Int]

  val firstFuture: Future[Int] = Future {
    Thread.sleep(100)
    1
  }
  firstFuture.onSuccess{ case i => promiseGetInt.trySuccess(i)}

  val secondFuture: Future[Int] = Future {
    Thread.sleep(200)
    2
  }
  secondFuture.onSuccess{ case i => promiseGetInt.trySuccess(i)}

  val futureGetInt: Future[Int] = promiseGetInt.future
```

```
  futureGetInt.onComplete {
    case Success(i) => println(s"Success! i: ${i}")
    case Failure(t) => println(s"Failure! t: ${t.getMessage}")
  }

  Thread.sleep(1000)
}
```

Success! i: 1 100 1 firstFuture 200
2 secondFuture firstFuture
promise future 1 Promise

Future Promise 0 1000
8 Future 3 3
Java

CountDownLatch

```
import java.util.concurrent.atomic.AtomicInteger
import scala.concurrent.ExecutionContext.Implicits.global
import scala.concurrent.{Promise, Future}
import scala.util.Random

object CountDownLatchSample extends App {
  val indexHolder = new AtomicInteger(0)
  val random = new Random()
  val promises: Seq[Promise[Int]] = for {i <- 1 to 3} yield Promise[Int]
  val futures: Seq[Future[Int]] = for {i <- 1 to 8} yield Future[Int] {
    val waitMilliSec = random.nextInt(1000)
    Thread.sleep(waitMilliSec)
    waitMilliSec
  }
  futures.foreach { f => f.onSuccess {case waitMilliSec =>
    val index = indexHolder.getAndIncrement
    if(index < promises.length) {
      promises(index).success(waitMilliSec)
    }
  }}
  promises.foreach { p => p.future.onSuccess{ case waitMilliSec => println(waitMilliSec)}}
  Thread.sleep(5000)
}
```

Future

AtomicInteger index
Promise Promise
Future Future
Promise AtomicInteger
Future

AtomicInteger Java [*15]

Java concurrent

RxScala
Rx C# Reactive Extensions

[*15] Java Concurrency in Practice Java
API Effective Java

-
-
  -
-
  -
-
  -
- JSTQB [*16]
- component testing
  -
  -
  -
-
  -
-
  -
-
  -
  -
-
- Unit Test
  -

[*16] http://www.jstqb.jp/dl/JSTQB-glossary.V2.3.J02.pdf

–

Java JUnit PHP PHPUnit xUnit

- (Integration Test)
  –

Selenium

- (System Test)
  –

Scala

3

1.

2.

3.

TDD: Test

Driven Development TDD

1.
2.

4

1.
2.
3.
4.

TDD

1.
2.
3.
4.

Scala 2

- Specs2
- ScalaTest

power assert ScalaTest

[17]

ScalaTest BDD :Behavior Driven Development BDD

sbt

scalatest_study

src/main/scala src/test/scala 2

build.sbt

```
name := "scalatest_study"

version := "1.0"

scalaVersion := "2.11.8"

libraryDependencies += "org.scalatest" %% "scalatest" % "2.2.6" % "test"
```

scalatest_study sbt compile

```
[info] Set current project to scalatest_study (in build file:/Users/dwango/workspace/scalatest_st
[info] Updating {file:/Users/dwango/workspace/scalatest_study/scalatest_study/}scalatest_study...
[info] Resolving jline#jline;2.12.1 ...
[info] downloading http://repo1.maven.org/maven2/org/scalatest/scalatest_2.11/2.2.6/scalatest_2.1
[info]  [SUCCESSFUL ] org.scalatest#scalatest_2.11;2.2.6!scalatest_2.11.jar(bundle) (10199ms)
[info] Done updating.
[success] Total time: 11 s, completed 2015/04/09 16:48:42
```

[17] assert “power assert”

Calc

Calc

- sum
- 2 div
- 1 isPrime

src/main/scala/Calc.scala

```
class Calc {

  /**
    *
    * Int
    */
  def sum(seq: Seq[Int]): Int = seq.foldLeft(0)(_ + _)

  /**      2
    *
    * 0
    */
  def div(numerator: Int, denominator: Int): Double = {
    if (denominator == 0) throw new ArithmeticException("/ by zero")
    numerator.toDouble / denominator.toDouble
  }

  /**                                                    */
  def isPrime(n: Int): Boolean = {
    if (n < 2) false else !((2 to Math.sqrt(n).toInt) exists (n % _ == 0))
  }
}
```

- sum
  - –
  - Int
- div
  - – 2
  - 0
- isPrime
  -

– 100

1.
2.
3.

XP

2

1 1

src/test/scala/CalcSpec.scala

```
import org.scalatest._

class CalcSpec extends FlatSpec with DiagrammedAssertions {

  val calc = new Calc

  "sum    " should "                                                    " in {
    assert(calc.sum(Seq(1, 2, 3)) === 6)
    assert(calc.sum(Seq(0)) === 0)
    assert(calc.sum(Seq(-1, 1)) === 0)
    assert(calc.sum(Seq()) === 0)
  }

  it should "Int                                  " in {
    assert(calc.sum(Seq(Integer.MAX_VALUE, 1)) === Integer.MIN_VALUE)
  }
}
```

DiagrammedAssertions assert

*18 DiagrammedAssertions API

*18 Scala Predef assert

```
               sbt test

[info] Loading project definition from /Users/dwango/workspace/scalatest_study/project
[info] Set current project to scalatest_study (in build file:/Users/dwango/workspace/scalatest_st
[info] Compiling 1 Scala source to /Users/dwango/workspace/scalatest_study/target/scala-2.11/clas
[info] Compiling 1 Scala source to /Users/dwango/workspace/scalatest_study/target/scala-2.11/test
[info] CalcSpec:
[info] sum
[info] - should
[info] - should Int
[info] Run completed in 570 milliseconds.
[info] Total number of tests run: 2
[info] Suites: completed 1, aborted 0
[info] Tests: succeeded 2, failed 0, canceled 0, ignored 0, pending 0
[info] All tests passed.
[success] Total time: 12 s, completed 2015/12/25 1:25:56



[info] Loading project definition from /Users/dwango/workspace/scalatest_study/project
[info] Set current project to scalatest_study (in build file:/Users/dwango/workspace/scalatest_st
[info] Compiling 1 Scala source to /Users/dwango/workspace/scalatest_study/target/scala-2.11/test
[info] CalcSpec:
[info] sum
[info] - should
    *** FAILED ***
[info]   assert(calc.sum(Seq(1, 2, 3)) === 7)
[info]          |    |  ||   |  |  |   |   |
[info]          |    6  ||   1  2  3   |   7
[info]          |       |List(1, 2, 3) false
[info]          |       6
[info]          Calc@e72a964 (CalcSpec.scala:8)
[info] - should Int
[info] Run completed in 288 milliseconds.
[info] Total number of tests run: 2
[info] Suites: completed 1, aborted 0
[info] Tests: succeeded 1, failed 1, canceled 0, ignored 0, pending 0
```

```
[info] *** 1 TEST FAILED ***
[error] Failed tests:
[error]     CalcSpec
[error] (test:test) sbt.TestsFailedException: Tests unsuccessful
[error] Total time: 7 s, completed 2015/12/25 1:39:59
```

div

```
import org.scalatest._

class CalcSpec extends FlatSpec with DiagrammedAssertions {

  val calc = new Calc

  // ...

  "div    " should "      2                                          " in {
    assert(calc.div(6, 3) === 2.0)
    assert(calc.div(1, 3) === 0.3333333333333333)
  }

  it should "0                                      " in {
    intercept[ArithmeticException] {
      calc.div(1, 0)
    }
  }
}
```

intercept[Exception]

```
import org.scalatest._
import org.scalatest.concurrent.Timeouts
import org.scalatest.time.SpanSugar._

class CalcSpec extends FlatSpec with DiagrammedAssertions with Timeouts {

  val calc = new Calc

  // ...
```

```
  "isPrime    " should "                                    " in {
    assert(calc.isPrime(0) === false)
    assert(calc.isPrime(-1) === false)
    assert(calc.isPrime(2))
    assert(calc.isPrime(17))
  }

  it should "100                                      " in {
    failAfter(1000 millis) {
      assert(calc.isPrime(9999991))
    }
  }
}
```

Timeouts failAfter

sbt test

```
[info] Loading project definition from /Users/dwango/workspace/scalatest_study/project
[info] Set current project to scalatest_study (in build file:/Users/dwango/workspace/scalatest_st
[info] Compiling 1 Scala source to /Users/dwango/workspace/scalatest_study/target/scala-2.11/test
[info] CalcSpec:
[info] sum
[info] - should
[info] - should Int
[info] div
[info] - should       2
[info] - should 0
[info] isPrime
[info] - should
[info] - should 100
[info] Run completed in 280 milliseconds.
[info] Total number of tests run: 6
[info] Suites: completed 1, aborted 0
[info] Tests: succeeded 6, failed 0, canceled 0, ignored 0, pending 0
[info] All tests passed.
[success] Total time: 8 s, completed 2015/12/25 1:43:22
```

BDD

- ScalaMock
- EasyMock
- JMock
- Mockito

Mockito build.sbt

```
libraryDependencies += "org.mockito" %% "mockito-core" % "1.10.19" % "test"
```

Calc sum

```
import org.scalatest.{FlatSpec, DiagrammedAssertions}
import org.scalatest.concurrent.Timeouts
import org.scalatest.mock.MockitoSugar
import org.mockito.Mockito._

class CalcSpec extends FlatSpec with DiagrammedAssertions with Timeouts with MockitoSugar {

  // ...

  "Calc                  " should "                          " in {
    val mockCalc = mock[Calc]
    when(mockCalc.sum(Seq(3, 4, 5))).thenReturn(12)
    assert(mockCalc.sum(Seq(3, 4, 5)) === 12)
  }
}
```

MockitoSuger ScalaTest

Mockito `val mockCalc = mock[Calc]`

`when(mockCalc.sum(Seq(3, 4, 5)).thenReturn(12)`

`assert(mockCalc.sum(Seq(3, 4, 5)) === 12)`

*19

scoverage

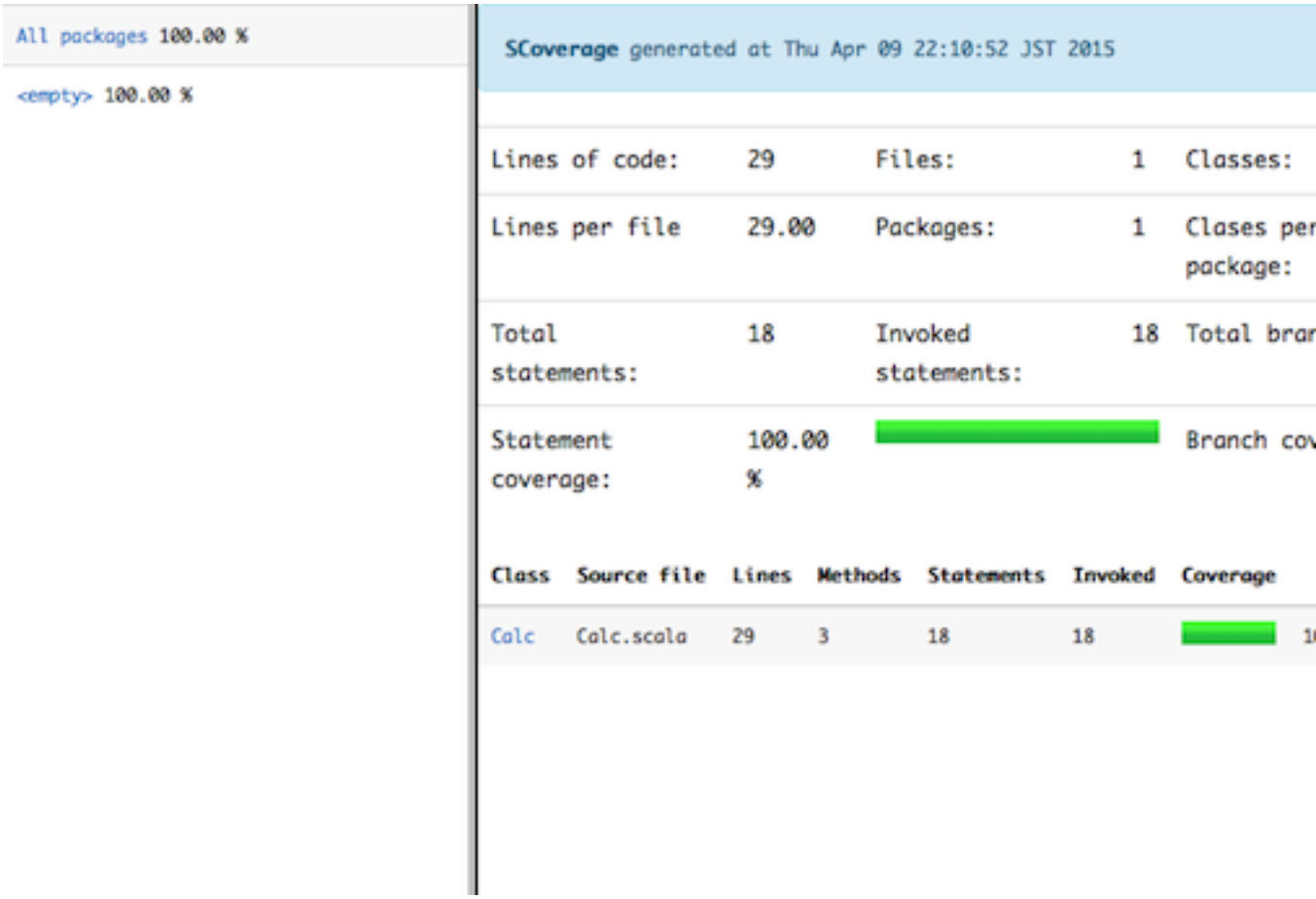
SCCT

project/plugins.sbt

```
resolvers += Classpaths.sbtPluginReleases

addSbtPlugin("org.scoverage" % "sbt-scoverage" % "1.3.3")
```

sbt clean coverage test target/scala-2.11/scoverage-report/index.html

*19 xUnit Test Patterns



100

” ”

N

JavaScript istanbul total 0

100%

Jenkins

CI

CI

ScalaStyle

project/plugins.sbt

```
addSbtPlugin("org.scalastyle" %% "scalastyle-sbt-plugin" % "0.6.0")
```

sbt scalastyleGenerateConfig sbt scalastyle

```
[info] Loading project definition from /Users/dwango/workspace/scalatest_study/project
[info] Set current project to scalatest_study (in build file:/Users/dwango/workspace/scalatest_st
[info] scalastyle using config /Users/dwango/workspace/scalatest_study/scalastyle-config.xml
[warn] /Users/dwango/workspace/scalatest_study/src/main/scala/Calc.scala:1: Header does not match
[info] Processed 1 file(s)
[info] Found 0 errors
[info] Found 1 warnings
[info] Found 0 infos
[info] Finished in 12 ms
[success] created output: /Users/dwango/workspace/scalatest_study/target
[success] Total time: 1 s, completed 2015/04/09 22:17:40
```

scalastyle-config.xml

Apache

-

-
-

Java

Scala Java

Scala JVM(Java Virtual Machine) Java
Scala Scala
Java Java
Scala

import

Java import Scala OK

```
import java.util.*;
import java.util.ArrayList;
```

```
import java.util._
import java.util.ArrayList
```

Java * _

Java Java

```
ArrayList<String> list = new ArrayList<>();
```

Scala

```
scala> val list = new ArrayList[String]()
list: java.util.ArrayList[String] = []
```

java.util.HashSet new

```
scala> import java.util.HashSet
import java.util.HashSet

scala> val set = new HashSet[String]
set: java.util.HashSet[String] = []
```

```
list.add("Hello");
list.add("World");
```

```
scala> list.add("Hello")
res0: Boolean = true

scala> list.add("World")
res1: Boolean = true
```

java.lang.System out println

"Hello, World!

```
scala> System.out.println("Hello, World!")
```

static static Java
1 Scala static static
A static foo
A B B.foo() A.foo()
Java

System.currentTimeMillis() Scala

```
scala> System.currentTimeMillis()
res0: Long = 1416357548906
```

java.lang.System static exit() 0

```
System.exit(0)
```

Scala

static static Java static
static
Java JFrame.EXIT_ON_CLOSE

```
import javax.swing.JFrame;

public class MyFrame extends JFrame {
  public MyFrame() {
    setDefaultCloseOperation(EXIT_ON_CLOSE); //JFrame EXIT_ON_CLOSE OK
  }
}
```

Scala

```
scala> import javax.swing.JFrame
```

```
class MyFrame extends JFrame {
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE) //JFrame.
}
```

Scala

Scala Java

Scala Java static 1

`java.lang.System` static `err`

Scala Java Java Scala
`System.currentTimeMillis()` long Scala `scala.Long`
Scala Java

Java Scala
`java.lang` Scala import

Java Java
`int[]` `Array[Int]` `AnyRef` Scala
`AnyRef` `Array[Int]` `AnyRef`
`AnyRef` value class

null Option Scala null Option
Java null Scala
null Scala
`Option(value)` `value` null `None` null `Some(value)`

java.util.Map

```
scala> val map = new java.util.HashMap[String, Int]()
map: java.util.HashMap[String,Int] = {}

scala> map.put("A", 1)
```

```
res3: Int = 0

scala> map.put("B", 2)
res4: Int = 0

scala> map.put("C", 3)
res5: Int = 0

scala> Option(map.get("A"))
res6: Option[Int] = Some(1)

scala> Option(map.get("B"))
res7: Option[Int] = Some(2)

scala> Option(map.get("C"))
res8: Option[Int] = Some(3)

scala> Option(map.get("D"))
res9: Option[Int] = None
```

null Option Scala Java
Option()

JavaConverters Java Scala
Scala Java Java
Scala
JavaConverters

```
import scala.collection.JavaConverters._
```

Java Scala asJava() asScala()

```
scala> import scala.collection.JavaConverters._
import scala.collection.JavaConverters._

scala> import java.util.ArrayList
import java.util.ArrayList

scala> val list = new ArrayList[String]()
list: java.util.ArrayList[String] = []

scala> list.add("A")
res10: Boolean = true

scala> list.add("B")
res11: Boolean = true
```

```
scala> val scalaList = list.asScala
scalaList: scala.collection.mutable.Buffer[String] = Buffer(A, B)
```

Buffer Scala asScala
Java Scala

API

scala.collection.mutable.ArrayBuffer JavaConverters
java.util.List ArrayBuffer 1

```
scala> import scala.collection.mutable.ArrayBuffer
import scala.collection.mutable.ArrayBuffer

scala> import scala.collection.JavaConverters._
import scala.collection.JavaConverters._

scala> val buffer = new ArrayBuffer[String]
buffer: scala.collection.mutable.ArrayBuffer[String] = ArrayBuffer()

scala> buffer += "A"
res12: buffer.type = ArrayBuffer(A)

scala> buffer += "B"
res13: buffer.type = ArrayBuffer(A, B)

scala> buffer += "C"
res14: buffer.type = ArrayBuffer(A, B, C)

scala> val list = buffer.asJava
list: java.util.List[String] = [A, B, C]
```

S99

Scala
S99

S-99: Ninety-Nine Scala Problems

URL: http://aperiodic.net/phil/scala/s-99/

Prolog Ninety-Nine Prolog Problems Scala

- P01 P28
- P31 P41
- P46 P50
- P55 P69
- P70 P73
- P80 P89
- P90 P99

Scala

P01 P28

S99

https://github.com/dwango/S99

sbt

UserService

include

include

include

UserService

- register
- login
-

ScalikeJDBC

jBCrypt

???

UserService

1 UserService UserService

register login

insert find

register

insert

insert

find login find

private

```
class UserService {
  //                        ???
  val maxNameLength = 32

  //
  private[this] def insert(user: User): User = ???

  private[this] def createUser(rs: WrappedResultSet): User = ???

  private[this] def find(name: String): Option[User] = ???

  private[this] def find(id: Long): Option[User] = ???

  //
  private[this] def hashPassword(rawPassword: String): String = ???

  private[this] def checkPassword(rawPassword: String, hashedPassword: String): Boolean = ???

  //
  def register(name: String, rawPassword: String): User = ???

  //
  def login(name: String, rawPassword: String): User = ???
}
```

insert find

UserService

```
trait UserService {
  val maxNameLength = 32

  def register(name: String, rawPassword: String): User

  def login(name: String, rawPassword: String): User
}
```

UserServiceImpl

```
class UserServiceImpl extends UserService {
  //                        ???

  //
  def insert(user: User): User = ???

  def createUser(rs: WrappedResultSet): User = ???

  def find(name: String): Option[User] = ???

  def find(id: Long): Option[User] = ???

  //
  def hashPassword(rawPassword: String): String = ???

  def checkPassword(rawPassword: String, hashedPassword: String): Boolean = ???

  //
  def register(name: String, rawPassword: String): User = ???

  //
  def login(name: String, rawPassword: String): User = ???
}
```

UserService

1 Java

UserService

UserService

1

PasswordService PasswordServiceImpl

include

UserServiceImpl PasswordServiceImpl

```
class UserServiceImpl extends UserService with PasswordServiceImpl {
  //                        ???

  //
  def insert(user: User): User = ???

  def createUser(rs: WrappedResultSet): User = ???

  def find(name: String): Option[User] = ???

  def find(id: Long): Option[User] = ???

  //
  def register(name: String, rawPassword: String): User = ???

  //
  def login(name: String, rawPassword: String): User = ???
}
```

UserRepository

include

UserServiceImpl

```
class UserServiceImpl extends UserService with PasswordServiceImpl with UserRepositoryImpl {
  //                        ???

  //
  def register(name: String, rawPassword: String): User = ???

  //
  def login(name: String, rawPassword: String): User = ???
}
```

UserService

UserService 1

UserServiceImpl

UserServiceImpl UserRepositoryImpl UserRepositoryImpl
ScalikeJDBC
UserServiceImpl

UserServiceImpl

UserRepositoryImpl

Dependency Injection DI

Wikipedia Dependency injection

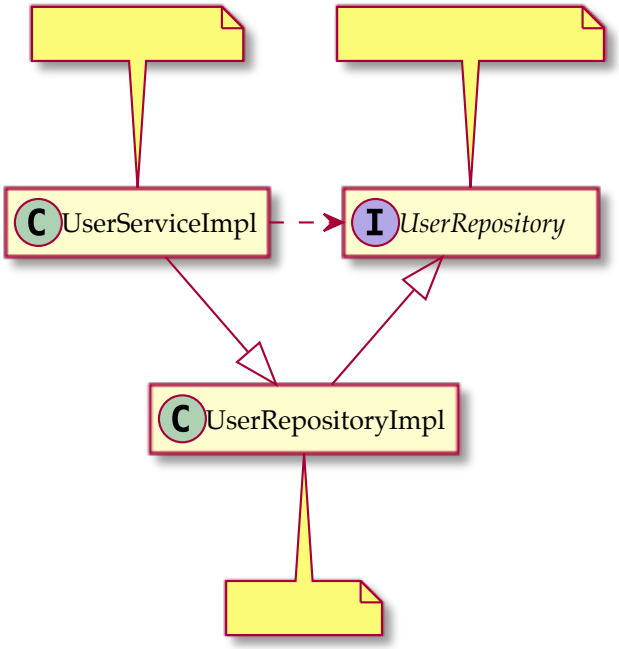
- Dependency
- Injection Dependency

DI 4

-
-
-
-

UserRepository UserRepositoryImpl UserServiceImpl
Wikipedia

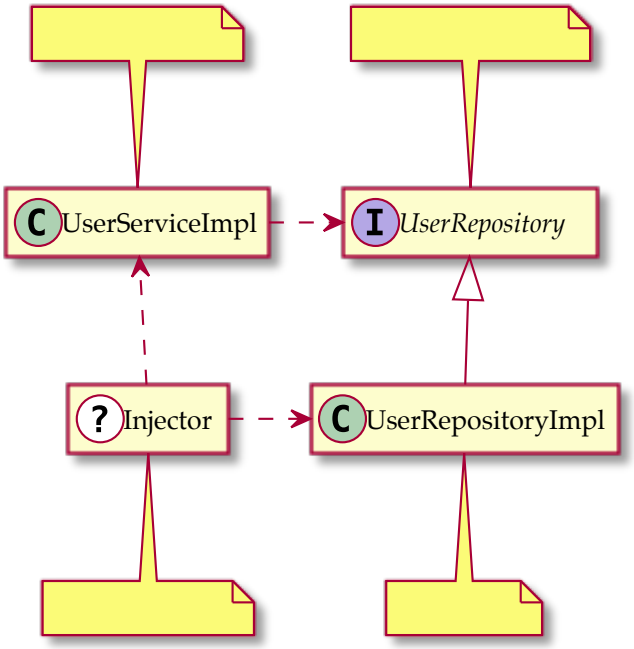
DI



DI	
	UserRepository
	UserRepositoryImpl
	UserServiceImpl UserRepository

DI `UserRepository`
`UserServiceImpl` `UserRepositoryImpl`
`UserServiceImpl` `UserRepository`
`UserRepositoryImpl`
DI

DI



`UserServiceImpl` `UserRepositoryImpl`
`UserRepositoryImpl`
Dependency `UserServiceImpl` Injection
Dependency Injection
DI

1
`UserRepositoryImpl`
`UserServiceImpl`

`UserRepositoryImpl` ScalikeJDBC
MongoDB `MongoUserRepositoryImpl` `UserServiceImpl`
MongoDB

DI DI

`UserService` `UserRepositoryImpl`
`UserService` `UserRepositoryImpl`

`UserRepository` `UserService` `UserRepository`
`UserRepositoryImpl`
`UserService`

DI
Java DI Spring Framework DI
DI Enterprise JavaBeans(EJB)
EJB Java Plain Old Java Object(POJO)
DI

Web Web
Web MySQL Redis Twitter
Facebook

DI

DI Dependency
Injection
DI

Java Spring Guice
DI Scala 1

UserService
UserService PasswordService UserRepository

```
trait UserService {
  self: PasswordService with UserRepository =>

  val maxNameLength = 32

  def register(name: String, rawPassword: String): User

  def login(name: String, rawPassword: String): User
}
```

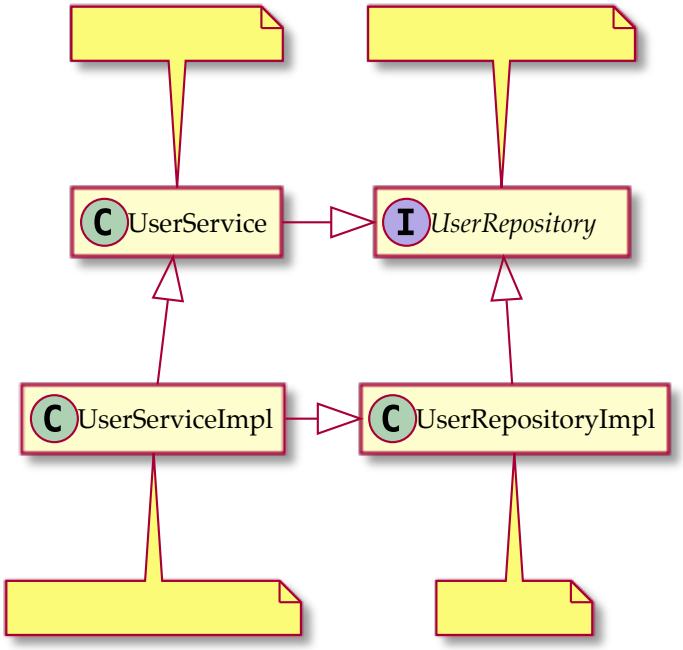
UserService PasswordService UserRepository
UserServiceImpl

```
trait UserService {
  self: PasswordService with UserRepository =>

  val maxNameLength = 32

  def register(name: String, rawPassword: String): User = {
    if (name.length > maxNameLength) {
      throw new Exception("Too long name!")
    }
    if (find(name).isDefined) {
      throw new Exception("Already registered!")
    }
    insert(User(name, hashPassword(rawPassword)))
  }

  def login(name: String, rawPassword: String): User = {
    find(name) match {
      case None       => throw new Exception("User not found!")
      case Some(user) =>
        if (!checkPassword(rawPassword, user.hashedPassword)) {
          throw new Exception("Invalid password!")
```

```
        }
        user
    }
  }
}
```

UserService PasswordServiceImpl UserRepositoryImpl

```
class UserServiceImpl extends UserService with PasswordServiceImpl with UserRepositoryImpl
```



UserServiceImpl UserService UserService
UserServiceImpl

`register` 2

ScalaTest

include

`sut` `UserService` `PasswordServiceImpl` `UserRepositoryImpl`

`user`

2