Scala

DS4300 – Large-Scale Storage and Retrieval

Scala: A Scalable Language

SCALA WAS USED TO BUILD

LinkedIn

twitter

FASCINATING FACTS ABOUT SCALA

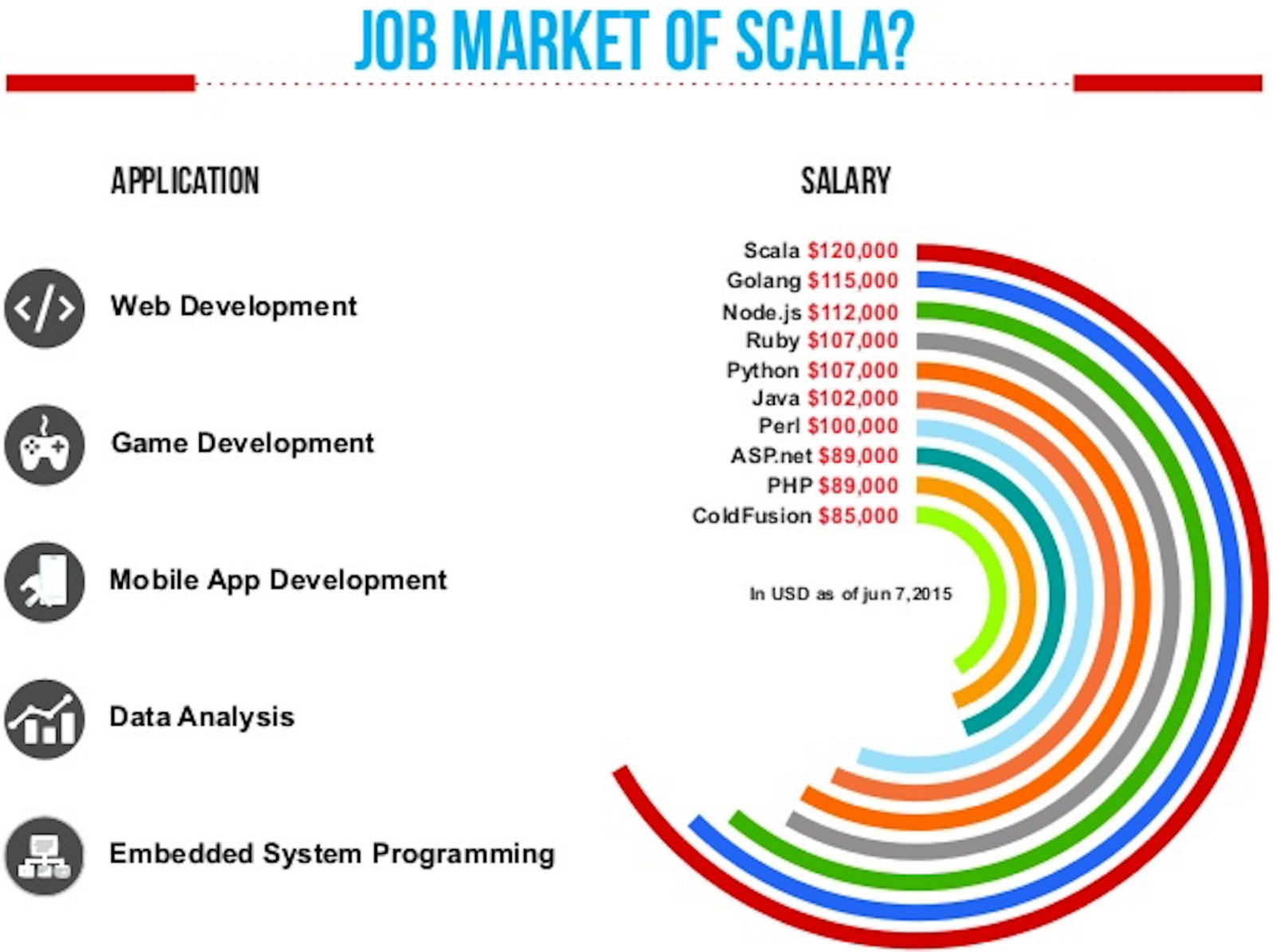
Scala is a general purpose programming language introduced in January 20, 2004 by:

Martin Odersky

Martin Odersky is a German computer scientist and professor of programming methods at EPFL in Switzerland.

What is Scala used for?

Why learn Scala?



Scala: A *scalable* language

- Scala can be used for both scripting and larger projection systems.

- Scala is designed to be extended via library modules with a relatively small core syntax.
  - ✓ Add new types: BigInteger, Complex, Etc.
  - ✓ Add new control constructs, e.g. the Akka Actor system for distributed computing.

- Scala is both more purely object-oriented as well as functional. In Scala functions ARE objects!

The Cathedral
(proprietary software)

- Development occurs behind walls
- Source code is usually not provided - kept locked up
- Corporate hierarchy

The Bazaar
(open source software)

- Code developed over the Internet with several others in public view
- Source code open to all users
- "Given enough eyeballs, all bugs are shallow"

Spark: A framework for distributed computing

- Spark was written in Scala using the Akka library

- Spark has APIs for Scala, Python, Java, and R but the Scala API's tend to be the cleanest and most performant.

Scala: A better Java

- Scala is a JVM language: All of Java's libraries are available and can be used in Scala!
- Although Scala is both object-oriented and functional, you aren't required to adopt a purely functional style from the start.

```
val ints = list.map(s => s.toInt)
```

```
List<Integer> ints = new ArrayList<Integer>();
for (String s : list) {
    ints.add(Integer.parseInt(s));
}
```

Scala: A better Java

```java
// Java
public class Person {
  private String name;
  private int age;
  public Person(String name, Int age) {  // constructor
    this.name = name;
   this.age = age;
  }
  public String getName() {              // name getter
    return name;
  }
  public int getAge() {                  // age getter
    return age;
  }
  public void setName(String name) {     // name setter
    this.name = name;
  }
  public void setAge(int age) {          // age setter
    this.age = age;
  }
}
```

```scala
// Scala
class Person(var name: String, var age: Int)
```

Scala: A better Java

Java:

```
import java.math.BigInteger

def factorial(x: BigInteger): BigInteger =
  if (x == BigInteger.ZERO)
    BigInteger.ONE
  else
    x.multiply(factorial(x.subtract(BigInteger.ONE)))
```

Scala:

```
def factorial(x: BigInt): BigInt =
  if (x == 0) 1 else x * factorial(x - 1)
```

Scala is “pure” Object-Oriented

```
// Every value is an object
1.toString

// Every operation is a method call
1 + 2 + 3  →  (1).+(2).+(3)

// Can omit . and ( )
"abc" charAt 1  →  "abc".charAt(1)
```

Scala Overview

SEAMLESS JAVA INTEROP

Scala runs on the JVM, so Java and Scala stacks can be freely mixed for totally seamless integration.

TYPE INFERENCE

So the type system doesn't feel so static. Don't work for the type system. Let the type system work for you!

CONCURRENCY & DISTRIBUTION

Use data-parallel operations on collections, use actors for concurrency and distribution, or futures for asynchronous programming.

TRAITS

Combine the flexibility of Java-style interfaces with the power of classes. Think principled multiple-inheritance.

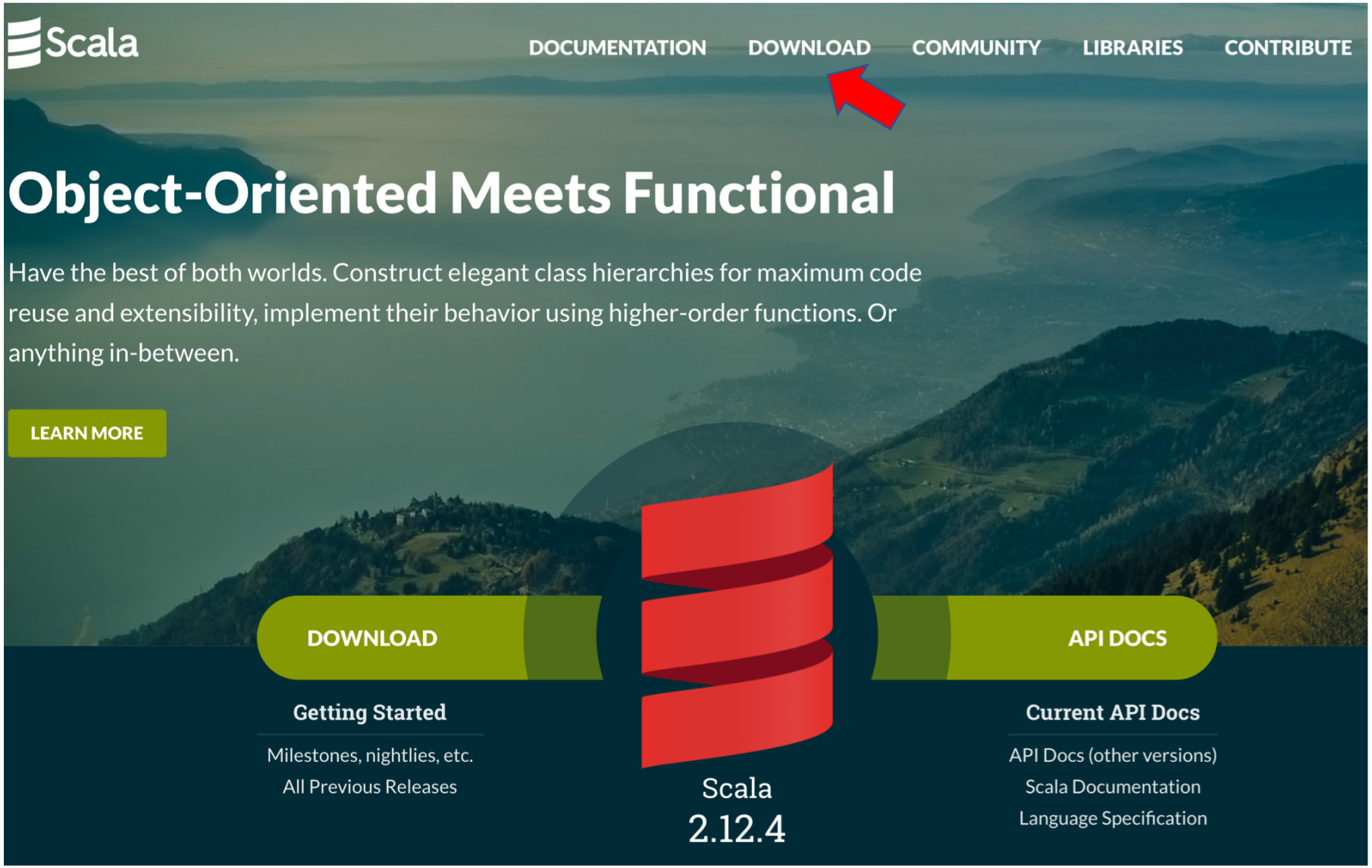
PATTERN MATCHING

Think "switch" on steroids. Match against class hierarchies, sequences, and more.

HIGHER-ORDER FUNCTIONS

Functions are first-class objects. Compose them with guaranteed type safety. Use them anywhere, pass them to anything.

Getting Started: scala-lang.org



Installing Scala

1

First, make sure you have the Java 8 JDK installed.

To check, open the terminal and type:

`java -version` *(Make sure you have version 1.8.)*

(If you don't have it installed, download Java here.)

2

Then, install Scala:

...either by installing an IDE such as IntelliJ, or sbt, Scala's build tool.

DOWNLOAD INTELLIJ or **DOWNLOAD SBT**

- Getting Started with Scala in IntelliJ
- Building a Scala Project with IntelliJ and sbt
- Testing Scala in IntelliJ with ScalaTest

- Getting Started with Scala and sbt on the Command Line
- Testing Scala with sbt and ScalaTest on the Command Line

...

Other ways to install Scala

- Download the Scala binaries for osx

  Need help running the binaries?

- Use Scastie to run single-file Scala programs in your browser using multple Scala compilers; the production Scala 2.x compilers, Scala.js, Dotty, and Typelevel Scala. Save and share executable Scala code snippets.
- Try Scala in the browser via ScalaFiddle. This lets you run single-file Scala programs in your browser using Scala.js, including graphical/interactive examples such as Oscilloscope or Ray Tracer
- Get Ammonite, a popular Scala REPL

Or are you looking for previous releases of Scala?

Setting up the SDK

Un-tar the binaries download into some convenient location, say “apps”

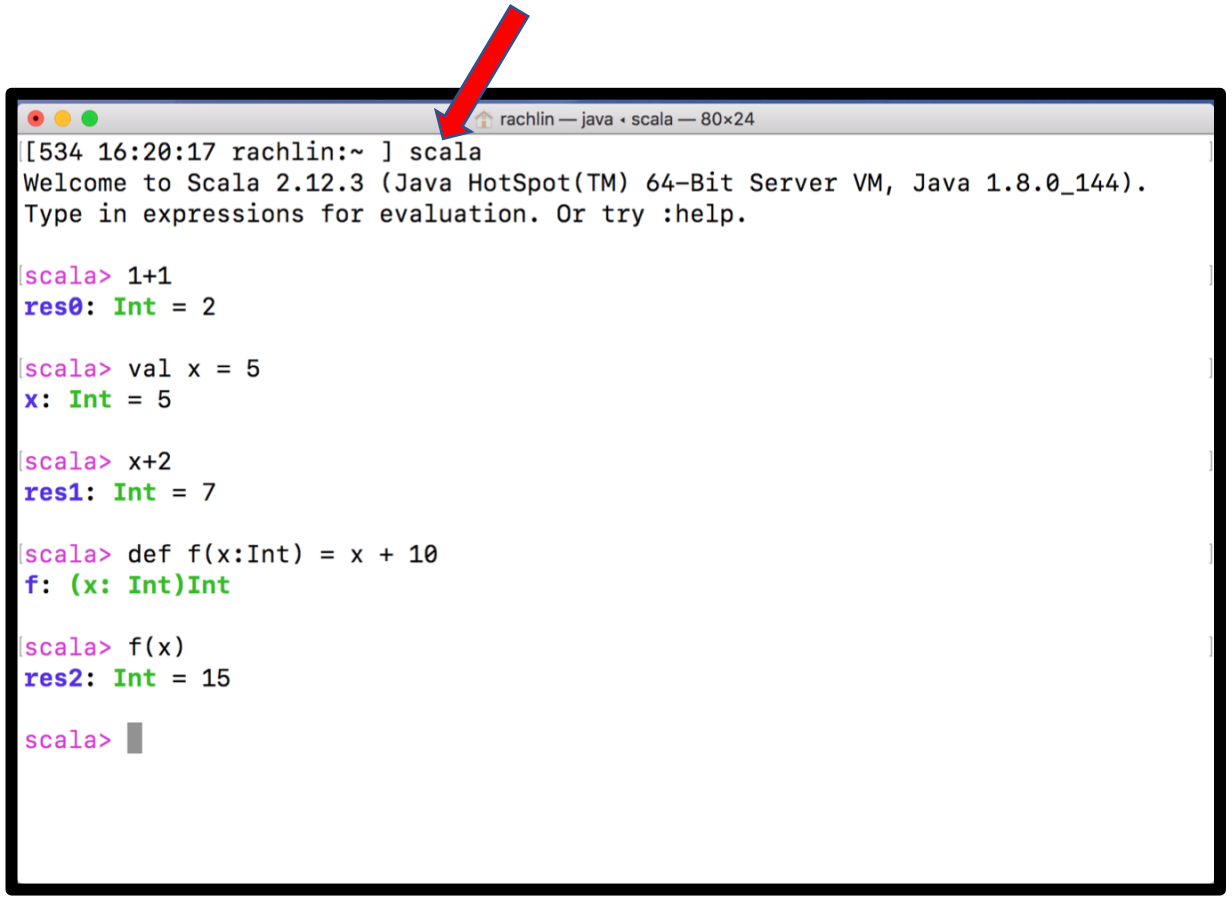
```
[531 15:57:39 rachlin:~/apps ] ll
total 0
drwxr-xr-x  9 rachlin  staff  288 Mar 29  2017 riak-2.2.3
drwxr-xr-x  6 rachlin  staff  192 Jul 25 06:51 scala-2.12.3
drwxr-xr-x  4 rachlin  staff  128 Sep  8 14:40 neo4j
drwxr-xr-x  7 rachlin  staff  224 Sep  8 20:32 mongodb
drwxr-xr-x  5 rachlin  staff  160 Dec  3 09:20 redis
[532 15:57:40 rachlin:~/apps ]
```

Update your .bashrc:

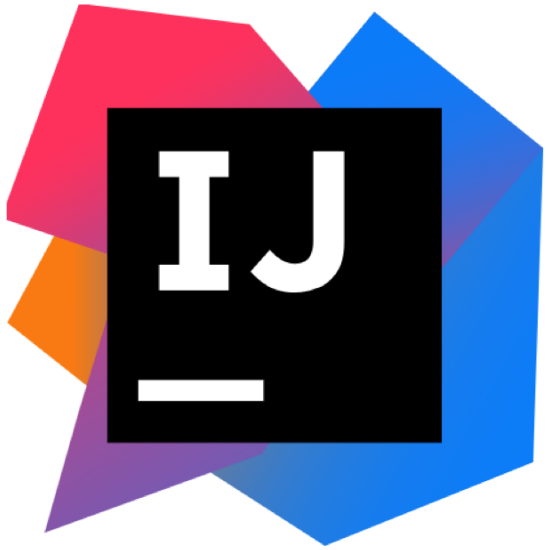
```
#EXPORTS
export APPS=/Users/rachlin/apps
export SCALA_HOME=$APPS/scala-2.12.3

#PATH

Export PATH=.:SCALA_HOME/bin:$PATH
```

Running the Scala REPL (Read-Eval-Print-Loop)

Note: $SCALA_HOME/bin is now in my $PATH



IntelliJ IDE (recommended!): www.jetbrains.com/idea



Download IntelliJ IDEA

Windows | macOS | Linux

Version: 2017.3.2
Build: 173.4127.27
Released: December 25, 2017
Release notes

System requirements
Installation Instructions
Previous versions

Ultimate

For web and enterprise development

DOWNLOAD

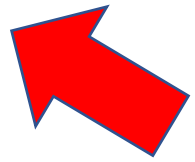
Free trial

Community

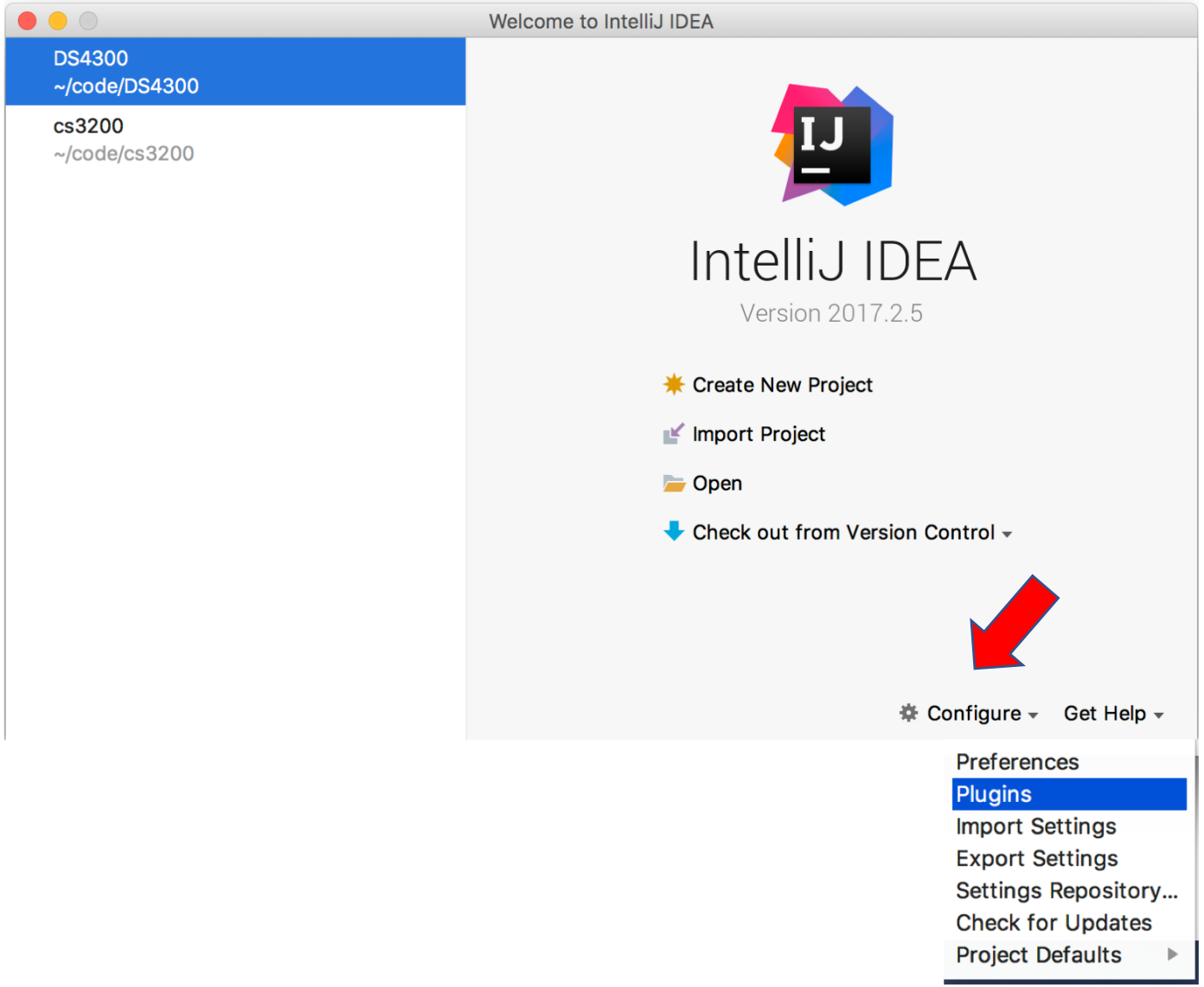
For JVM and Android development

DOWNLOAD

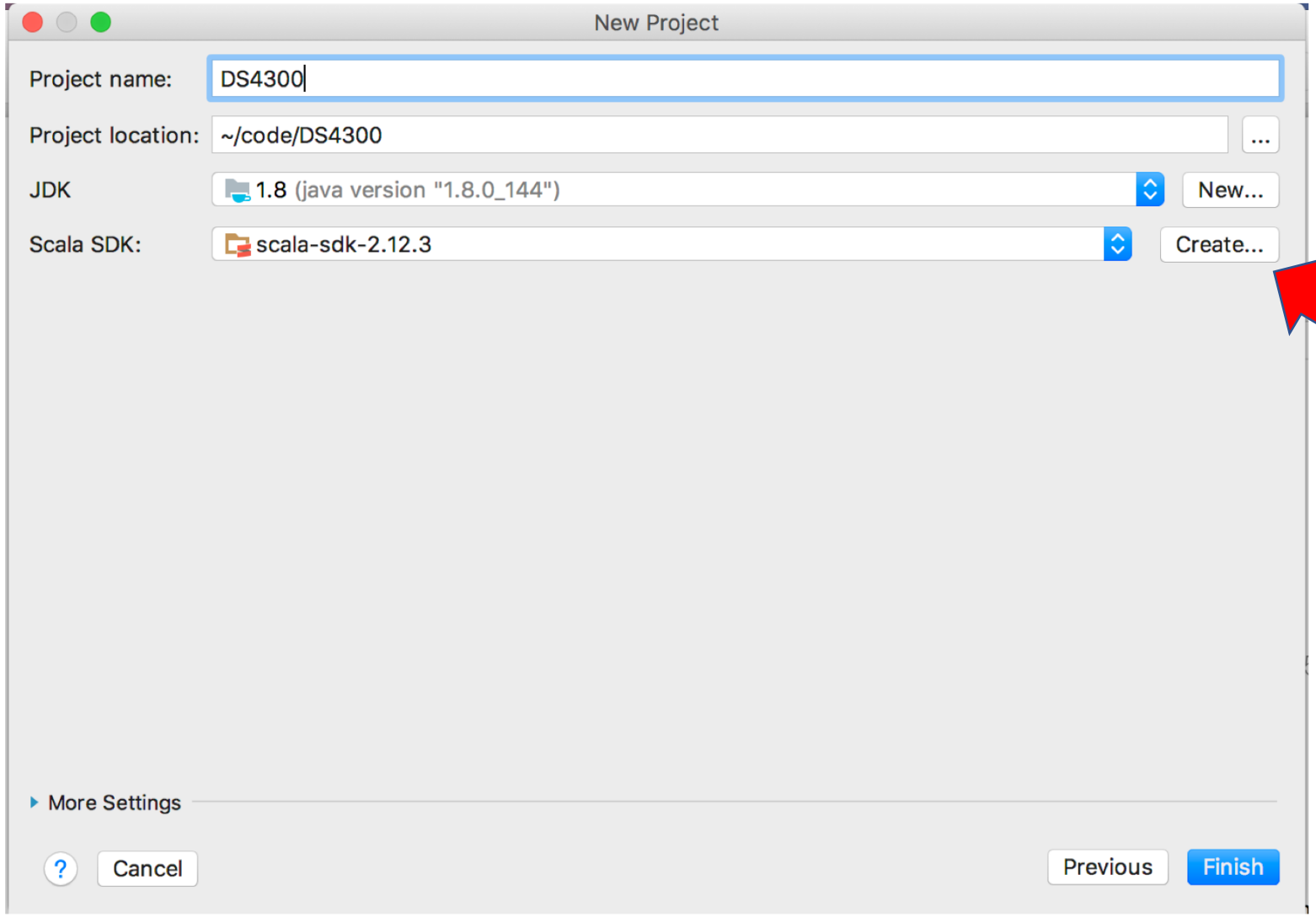
Free, open-source



Installing the IntelliJ IDE Plugin for Scala

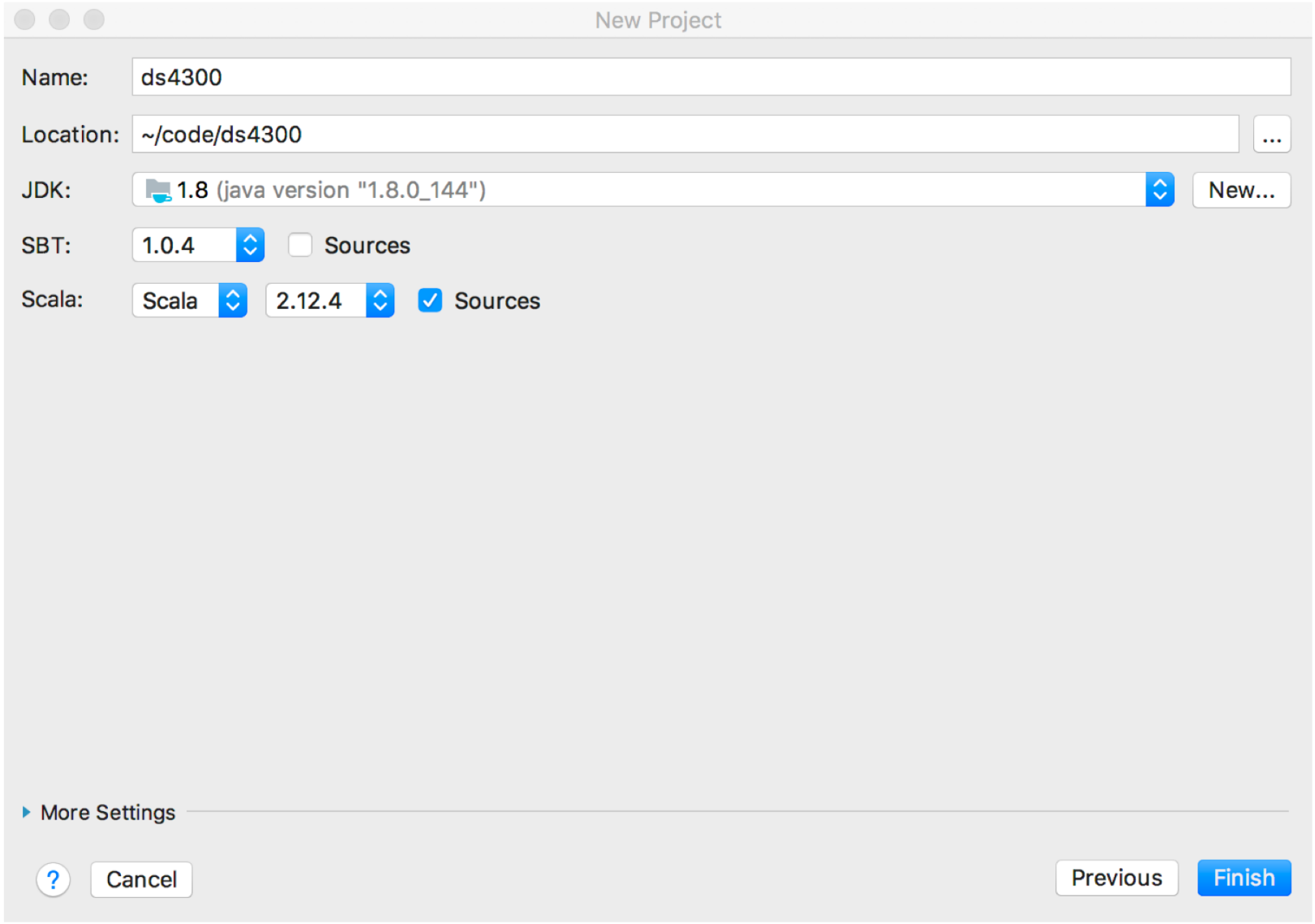


Creating a Project. (Standard IDEA project)

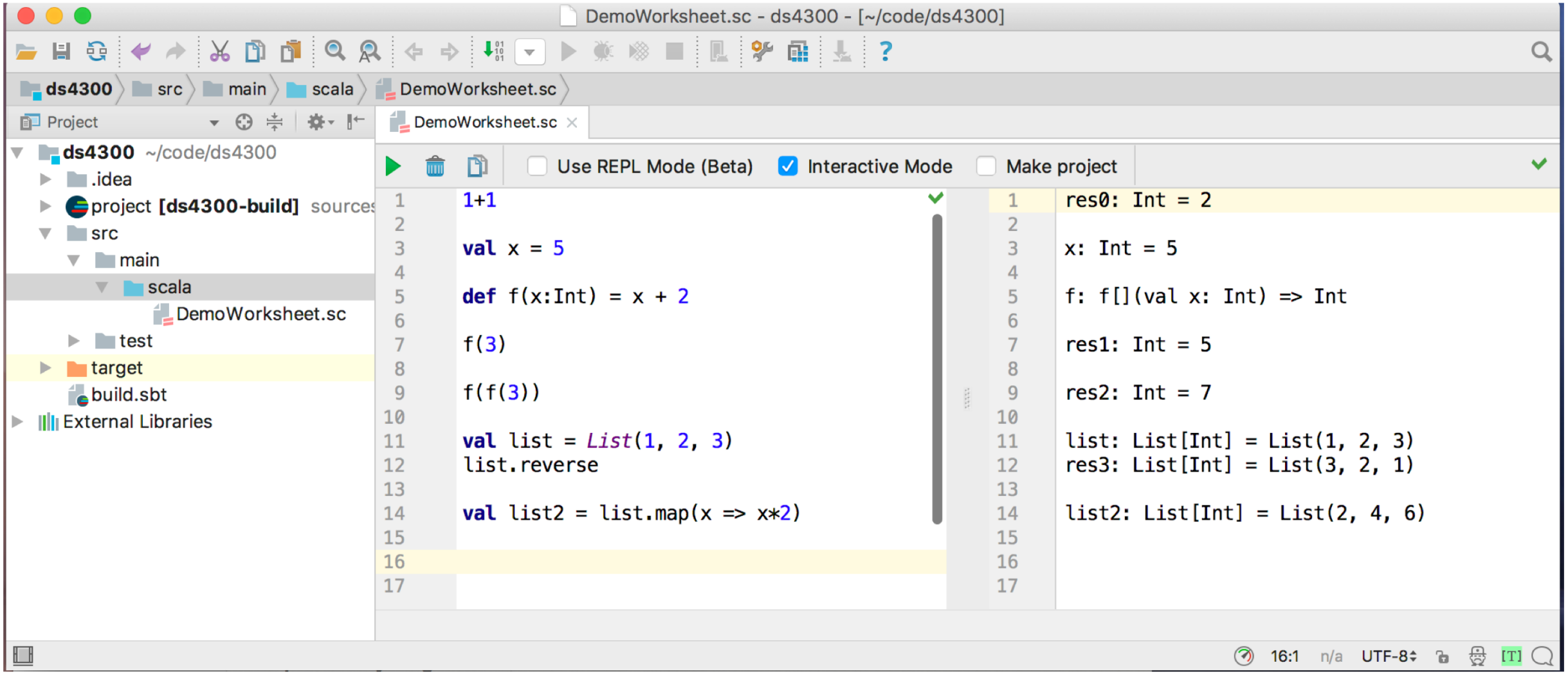


For your first project you may have to press this "Create" button and browse to your Scala installation home directory. Thereafter it should be automatically populated.

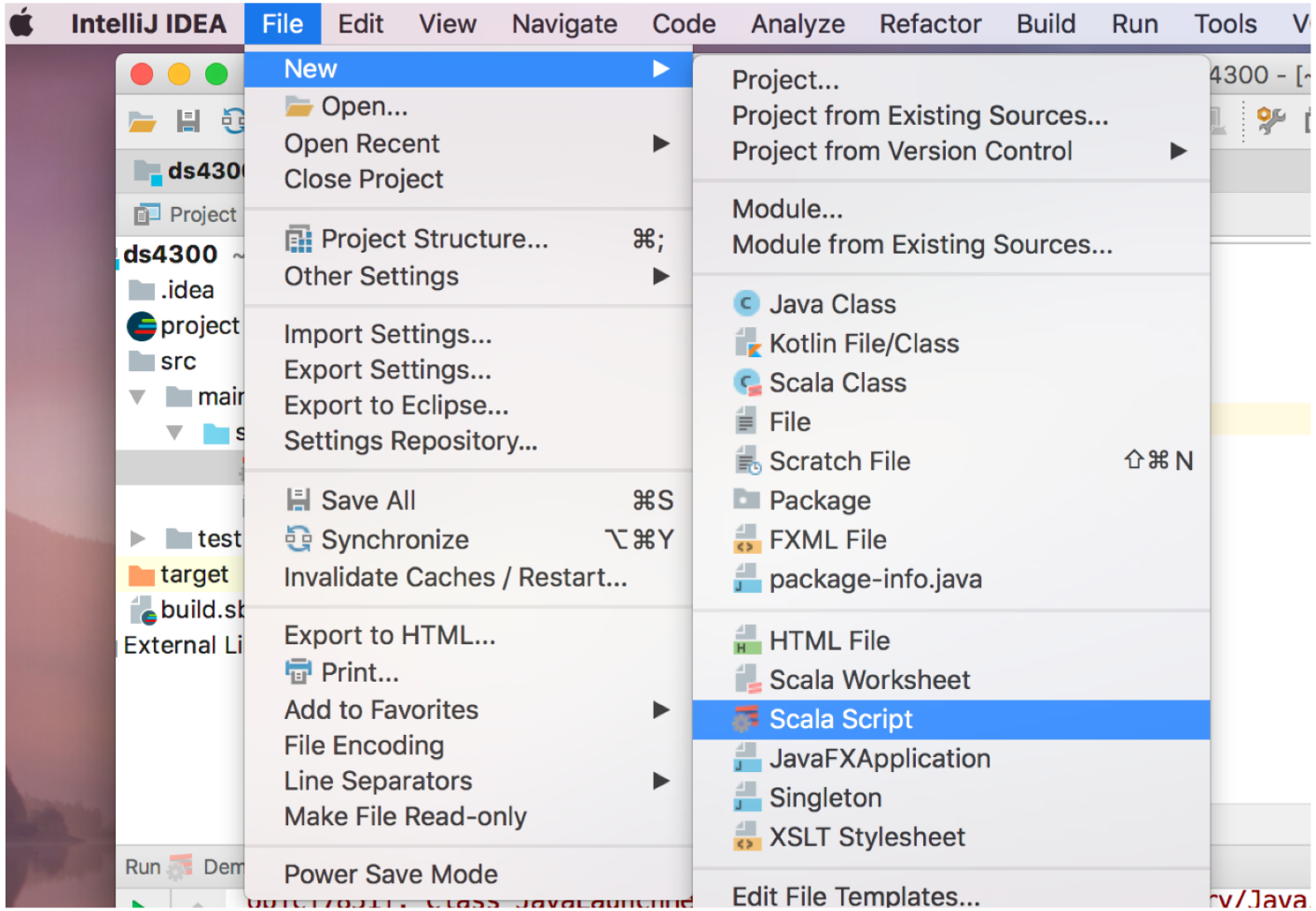
Creating a Project. (SBT project)



A simple worksheet (.sc)



Creating a simple Scala script



DemoScript.scala

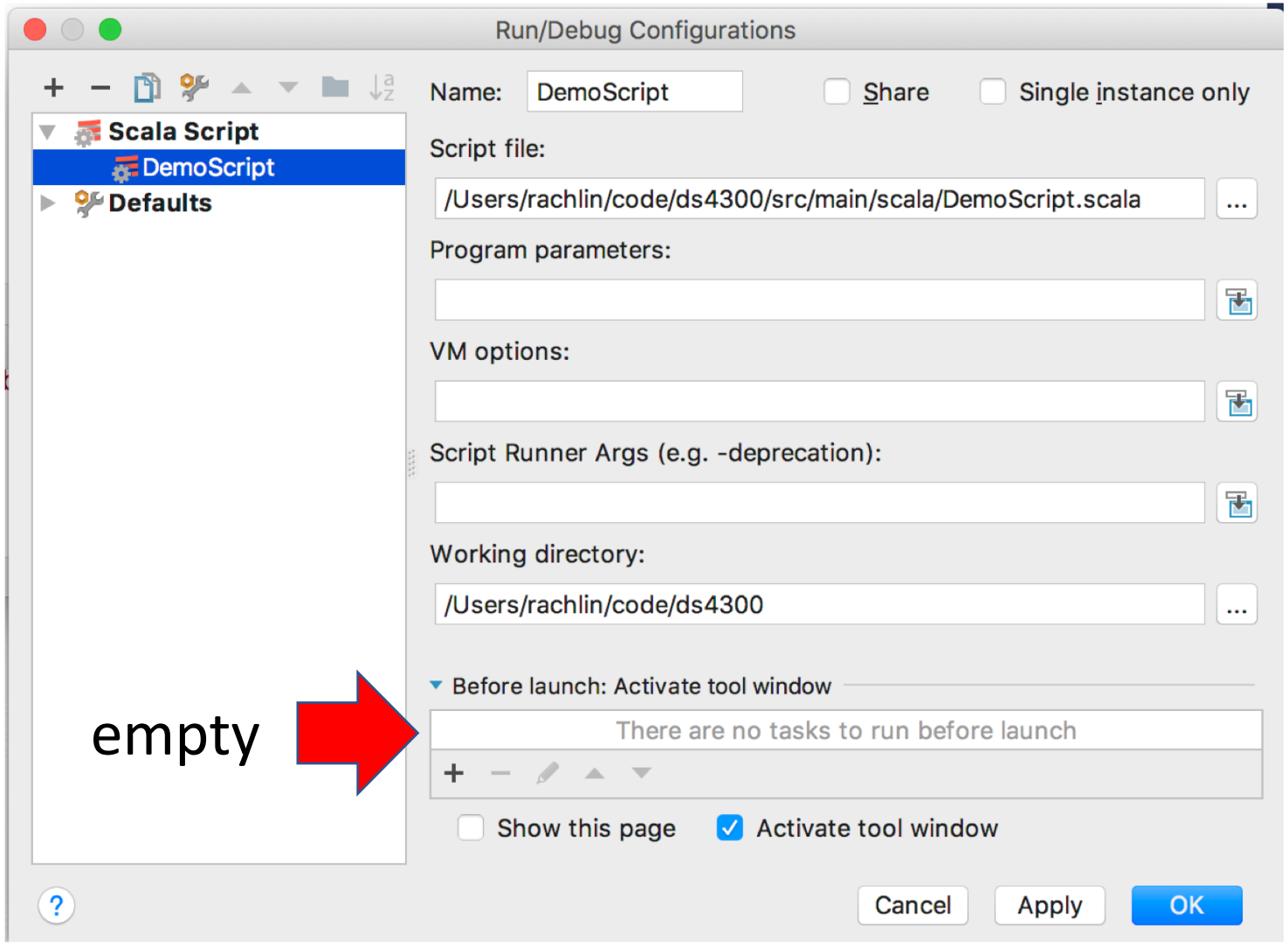
```
println("Hello World")

val x = 5
println(x*x)
```

Running a simple Scala script

From the command line:

```
$ scala DemoScript.scala
Hello World
25
```

Or from within IntelliJ using a run configuration:



Literals, Values, Variables, Types

- A *literal* (or literal data) is data that appears directly in the source code, like the number 5, the character *A*, and the text "Hello, World."
- A *value* is an immutable, typed storage unit. A value can be assigned data when it is defined, but can never be reassigned.
- A *variable* is a mutable, typed storage unit. A variable can be assigned data when it is defined and can also be reassigned data at any time.
- A *type* is the *kind* of data you are working with, a definition or classification of data. All data in Scala corresponds to a specific type, and all Scala types are defined as classes with methods that operate on the data.

Scala Numeric Types

Table 2-1. Core numeric types

Name	Description	Size	Min	Max
Byte	Signed integer	1 byte	–127	128
Short	Signed integer	2 bytes	–32768	32767
Int	Signed integer	4 bytes	-2^{31}	$2^{31}-1$
Long	Signed integer	8 bytes	-2^{63}	$2^{63}-1$
Float	Signed floating point	4 bytes	n/a	n/a
Double	Signed floating point	8 bytes	n/a	n/a

Table 2-2. Numeric literals

Literal	Type	Description
`5`	Int	Unadorned integer literals are `Int` by default
`0x0f`	Int	The "0x" prefix denotes hexadecimal notation
`5l`	Long	The "l" suffix denotes a `Long` type
`5.0`	Double	Unadorned decimal literals are `Double` by default
`5f`	Float	The "f" suffix denotes a `Float` type
`5d`	Double	The "d suffix denotes a `Double` type

Non-Numeric Types

Table 2-4. Core nonnumeric types

Name	Description	Instantiable
`Any`	The root of all types in Scala	No
`AnyVal`	The root of all value types	No
`AnyRef`	The root of all reference (nonvalue) types	No
`Nothing`	The subclass of all types	No
`Null`	The subclass of all AnyRef types signifying a null value	No
`Char`	Unicode character	Yes
`Boolean`	`true` or `false`	Yes
`String`	A string of characters (i.e., text)	Yes
`Unit`	Denotes the lack of a value	No

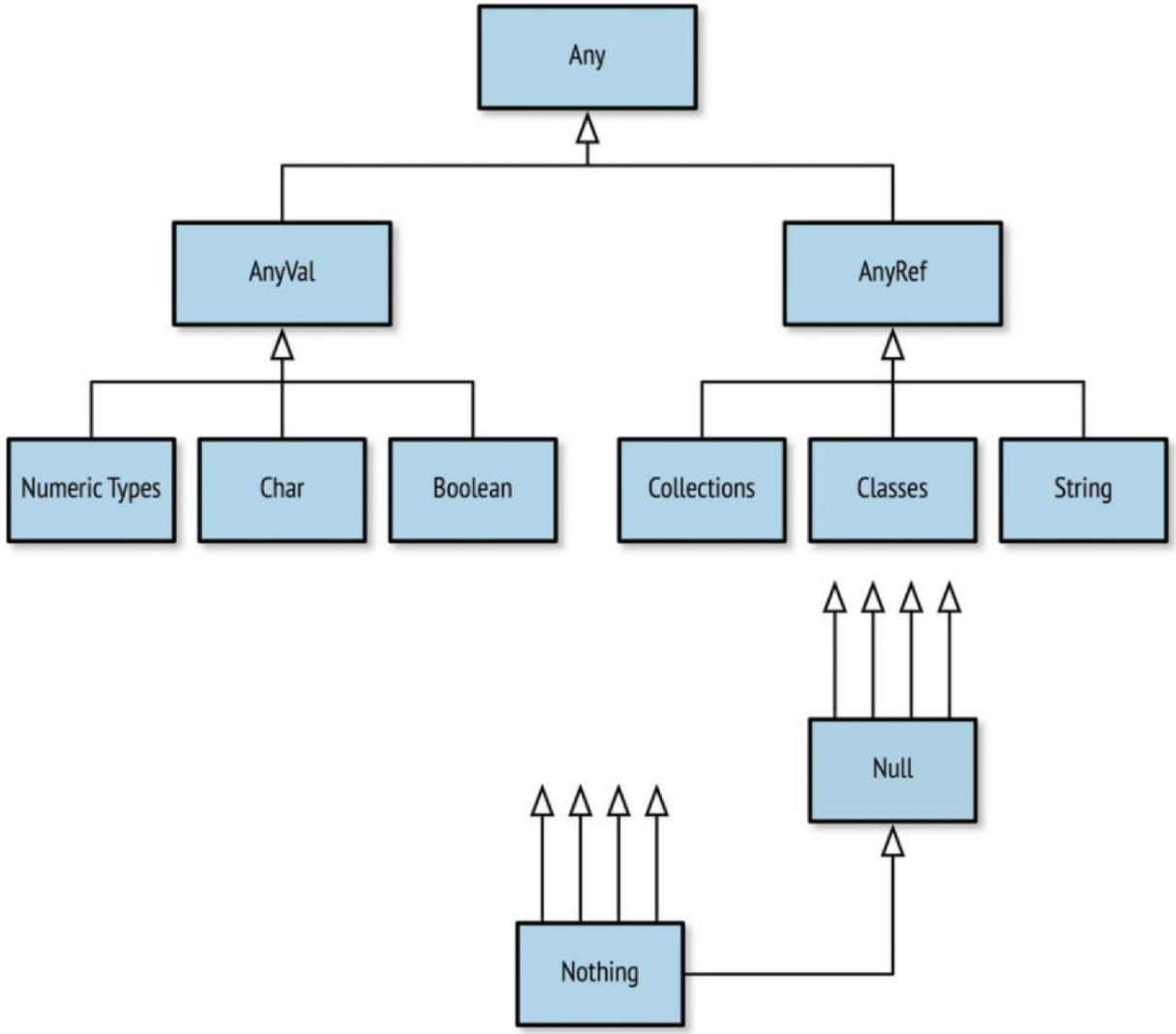


Figure 2-1. The Scala type hierarchy

Defining Functions

```
def add(a:Int, b:Int):Int = a + b

val m:Int = add(1, 2)

println(m)
```

Note the use of the type declaration "Int". Scala is a "statically typed" language. We define "add" to be a function which accepts two parameters of type Int and returns a value of type Int. Similarly, "m" is defined as a variable of type Int.

Type Inference

```
def add(a:Int, b:Int) = a + b

val m = add(1, 2)

println(m)
```

We have NOT specified the return type of the function or the type of the variable "m". Scala "infers" that!

Expressions

```
val i = 3
val p = if (i > 0) -1 else -2
val q = if (true) "hello" else "world"

println(p)
println(q)
```

Unlike languages like C/Java, almost everything in Scala is an "expression", ie, something which returns a value! Rather than programming with "statements", we program with "expressions"

Expressions

```
def errorMsg(errorCode: Int) = errorCode match {
  case 1 => "File not found"
  case 2 => "Permission denied"
  case 3 => "Invalid operation"
}

println(errorMsg(2))
```

Case automatically "returns" the value of the expression corresponding to the matching pattern.

Recursion

```
// sum n + (n-1) + (n-2) + ... + 0
def sum(n: Int): Int =
  if (n == 0) 0 else n + sum(n - 1)

val m = sum(10)
println(m)
```

Try calling the function "sum" with a large number (say 10000) as parameter! You get a stack overflow!

Recursion with stack overflow for large n

```
(sum 4)
(4 + sum 3)
(4 + (3 + sum 2))
(4 + (3 + (2 + sum 1)))
(4 + (3 + (2 + (1 + sum 0))))
(4 + (3 + (2 + (1 + 0))))
(4 + (3 + (2 + 1)))
(4 + (3 + 3))
(4 + 6)
(10)
```

Tail Recursion

```
def sum(n: Int, acc: Int):Int =
  if(n == 0) acc else sum(n - 1, acc + n)

val r = sum(10000, 0)

println(r)
```

```
(sum 4 0)
(sum 3 4)
(sum 2 7)
(sum 1 9)
(sum 0 10)
(10)
```

This is a "tail-recursive" version of the previous function - the Scala compiler converts the tail call to a loop, thereby avoiding stack overflow!

More summing functions

```
def sqr(x: Int) = x * x
def cube(x: Int) = x * x * x

def sumSimple(a: Int, b: Int): Int =
  if (a == b) a else a + sumSimple(a + 1, b)

def sumSquares(a: Int, b: Int): Int =
  if (a == b) sqr(a) else sqr(a) + sumSquares(a + 1, b)

def sumCubes(a: Int, b: Int): Int =
  if (a == b) cube(a) else cube(a) + sumCubes(a + 1, b)
```

Higher-order functions

```
def identity(x: Int) = x
def sqr(x: Int) = x * x
def cube(x: Int) = x * x * x
def sum(f: Int=>Int, a: Int, b: Int): Int =
  if (a == b) f(a) else f(a) + sum(f, a + 1, b)

println(sum(identity, 1, 10))
println(sum(sqr, 1, 10))
println(sum(cube, 1, 10))
```

"sum" is now a "higher order" function! It's first parameter is a function which maps an Int to an Int

The type "Int" represents a simple Integer value. The type `Int => Int` represents a function which accepts an Int and returns an Int.

Anonymous functions

```
def sum(f: Int=>Int, a: Int, b: Int): Int =
  if (a == b) f(a) else f(a) + sum(f, a + 1, b)

println(sum(x=>x, 1, 10))
println(sum(x=>x*x, 1, 10))
println(sum(x=>x*x*x, 1, 10))
```

We can create "anonymous" functions on-the-fly! `x => x*x` is a function which takes an "x" and returns x*x

Methods on lists involving higher-order functions

```
val a = List(1,2,3,4,5,6,7)

val b = a.map(x => x * x)
val c = a.filter(x => x < 5)
val d = a.reduce((x, y)=>x+y)

println(b)
println(c)
println(d)
```

Map applies a function on all elements of a sequence. Filter selects a set of values from a sequence based on the boolean value returned by a function passed as its parameter - both functions return a new sequence. Reduce combines the elements of a sequence into a single element.

More methods on Lists

```
def even(x: Int) = (x % 2) == 0

val a = List(1,2,3,4,5,6,7)
val b = List(2, 4, 6, 5, 10, 11, 13, 12)

// are all members even?
println(a.forall(even))

// is there an even element in the sequence?
println(a.exists(even))

//take while the element is even -
//stop at the first odd element
println(b.takeWhile(even))
```

Block Structure / Scoping

```
def fun(x: Int) = {
  val y = 1

  val r = {
    val y = 2
    x + y
  }
  println(r)
  println(x + y)
}

fun(10)
```

The "y" in the inner scope shadows the "y" in the outer scope

Nested functions

```
// fun returns a function of type Int => Int

def fun():Int => Int = {
  def sqr(x: Int):Int = x * x

  sqr
}

val f = fun()
println(f(10))
```

`def fun():Int=>Int` says "fun is a function which does not take any argument and returns a function which maps an Int to an Int. Note that it possible to have "nested" function definitions.

Immutability: val / var

```
val a = 1

a = 2 // does not work

a = a + 1 // does not work

var b = 1
b = 2
b = b + 1
```

"val" is immutable in the sense once you bind a value to a symbol defined as "val", the binding can't be changed

Immutable collections

```
val a = List(20, 30, 40, 50)
val b = 10 :: a
println(a) // still List(20, 30, 40, 50)
println(b) // List(10, 20, 30, 40, 50)

val c = a ++  List(60)
println(c) // List(20, 30, 40, 50, 60)
println(a) // still List(20, 30, 40, 50)
```