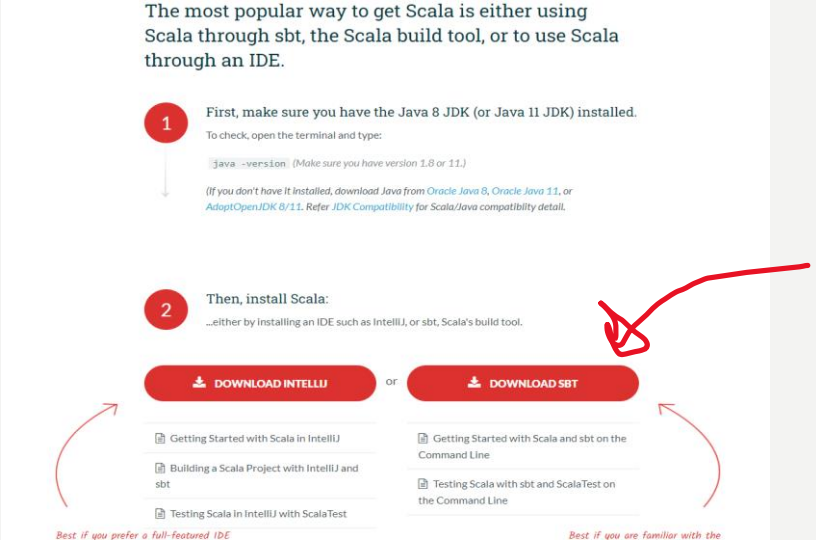
SCALA

THE LANGUAGE

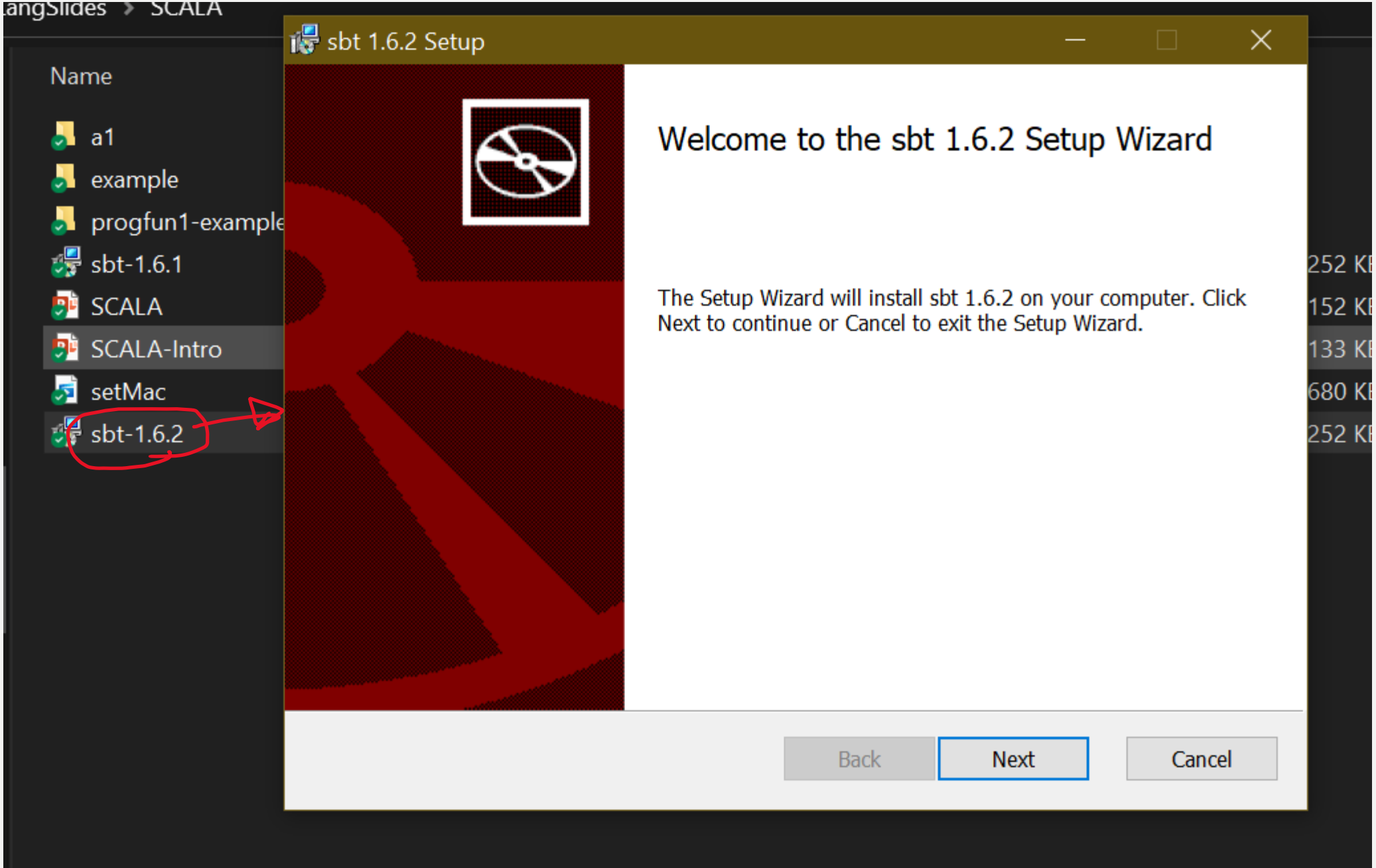
- Statically typed
- Runs on JVM (mix Java and SCALA)
- OO & Functional

SBT (SCALA BUILD TOOL)

- Compile, run, test!
- It comes with REPL (Read-Eval_Print loop).
  - Takes single input, executes it and returns the result of execution.
- To install sbt
  - Install Java.
  - Set environment variable (System variables -> Path) to know the path to bin folder of jdk and jre.
  - Set User Variable to have JAVA_HOME -> use path for the jdk folder.
  - Then search scala download on google, or go to https://www.scala-lang.org/download/scala2.html
  - I'm using version 2 for this course! (there is a version 3!)



- For windows, download and install MSI file.



- Copy path for bin folder of sbt.
- Set it as environment variable (System variables -> Path).

YOUR FIRST SCALA PROJECT

- Let's create folder a1.
- Then use command line to go into folder a1, then create another folder. Let's say "sbt_proj"
- In that folder, run the command sbt. You may need to wait.

Select C:\Windows\System32\cmd.exe - sbt

```
Microsoft Windows [Version 10.0.19043.1526]
(c) Microsoft Corporation. All rights reserved.

E:\Dropbox\teaching\ProgLangSlides\SCALA\a2\project1>sbt
[warn] Neither build.sbt nor a 'project' directory in the current directory: "E:\Dropbox\teaching\ProgLa
2\project1"
c) continue
q) quit
?c
[info] [launcher] getting org.scala-sbt sbt 1.6.2  (this may take some time)...
[warn] No sbt.version set in project/build.properties, base directory: E:\Dropbox\teaching\ProgLangSlide
ct1
[info] welcome to sbt 1.6.2 (AdoptOpenJDK Java 11.0.11)
[info] set current project to project1 (in build file:/E:/Dropbox/teaching/ProgLangSlides/SCALA/a2/proje
[info] sbt server started at local:sbt-server-823b2c7787da8509f36e
[info] started sbt server
sbt:project1>
```

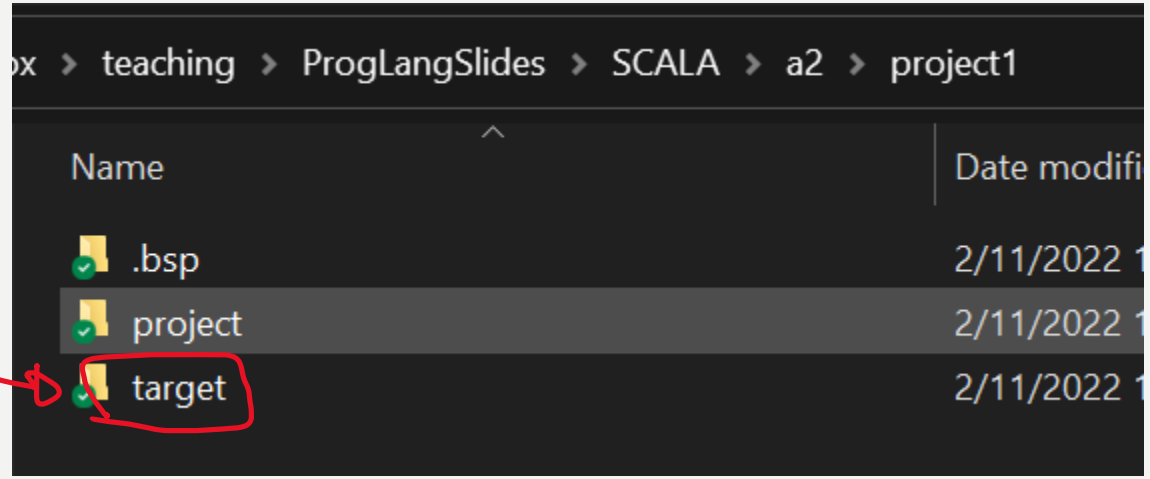
GETTING READY TO RUN REPL

- In sbt, type in **console**.
- You will get a Scala command prompt (we call it REPL).

```
sbt:project1> console
[info] Starting scala interpreter...
Welcome to Scala 2.12.15 (OpenJDK 64-Bit Server VM, Java 11.0.11).
Type in expressions for evaluation. Or try :help.

scala>
```

Temp files are stored. Scala is a compiled language so it needs to create a code file in order to run.



DATA TYPES

Boolean	true or false
Byte	8 bit signed value
Short	16 bit signed value
Char	16 bit unsigned Unicode character
Int	32 bit signed value
Long	64 bit signed value
Float	32 bit IEEE 754 single-precision float
Double	64 bit IEEE 754 double-precision float
String	A sequence of characters
Unit	Corresponds to no value
Null	null or empty reference
Nothing	subtype of every other type; includes no
Any The	supertype of any type; any object is of
AnyRef	The supertype of any reference type

DECLARING VARIABLES

- 1. using **var**
  - **This creates a normal (modifiable) variable.**

Separate type and name of a variable.

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

Don't need a semicolon at the end !

  - **You can then use variable a in other statements**

```
scala> a
res0: Int = 5

scala> a + 30
res1: Int = 35

scala>
```

```
scala> a = 25
a: Int = 25
```

- Variables need to be initialized when they are created.

```
scala> var c: Int
<console>:11: error: only traits and abstract classes can have declared but undefined membe
(Note that variables need to be initialized to be defined)
       var c: Int
           ^
```

- But you do not need to give the data type. It can detect the type by the initial value!

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

scala> var d = false
d: Boolean = false

scala> var e = 1.25
e: Double = 1.25
```

```
scala> var g = 4.44f
g: Float = 4.44
```

เป็น val = ค่าคงที่

- 2. using val
  - This is defining a constant.

```
scala> val b : Int =40
b: Int = 40

scala> b + 10
res2: Int = 50

scala> b = 10
<console>:12: error: reassignment to val
       b = 10
         ^
```

lazy สามารถจำเป็น expression ได้ เช่น แบบจำ 1+5-19 ทั้งก้อน

- Initialization of val can be delayed until the first read!

```
scala> lazy val k = 8
k: Int = <lazy>

scala> var v = a + k
v: Int = 18
```

แอบจำ

Not have value yet.

The initial value is assigned now.

EXECUTE A BLOCK OF CODE

เอาตัวแปรมารับ

block

```
scala> var x = {var h = 22.3; var i =1; e+h+i}
x: Double = 24.55
```

```
scala> var y = { var m = 5;
     | var n =1;
     | e+m+n}
y: Double = 7.25
```

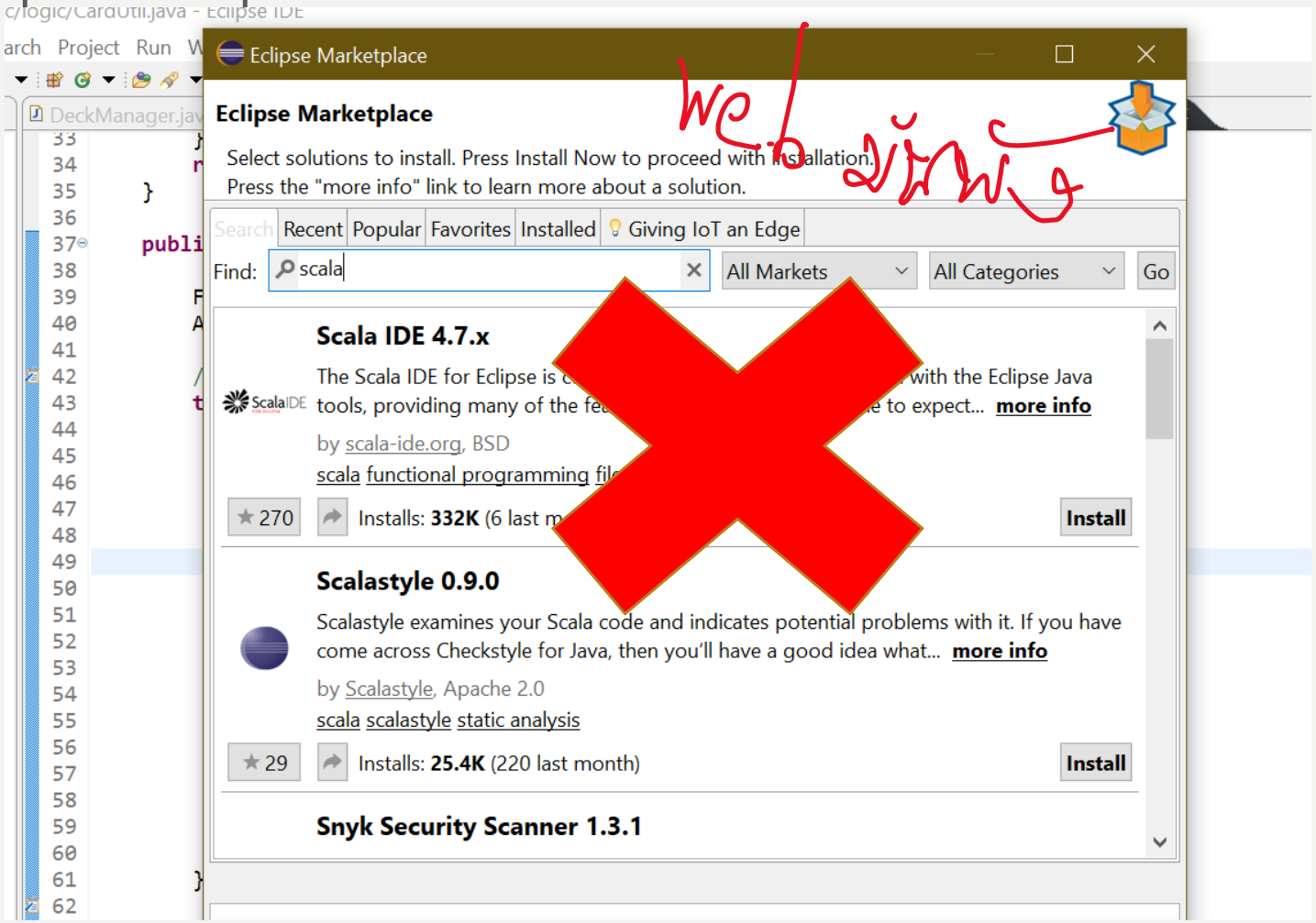
The code can be put on several lines.

```
scala> {val s = 10;
     | a =10
     | s+a
     | }
res3: Int = 20
```

Does not need to give a variable on LHS.
It will create a temporary variable to store the result.

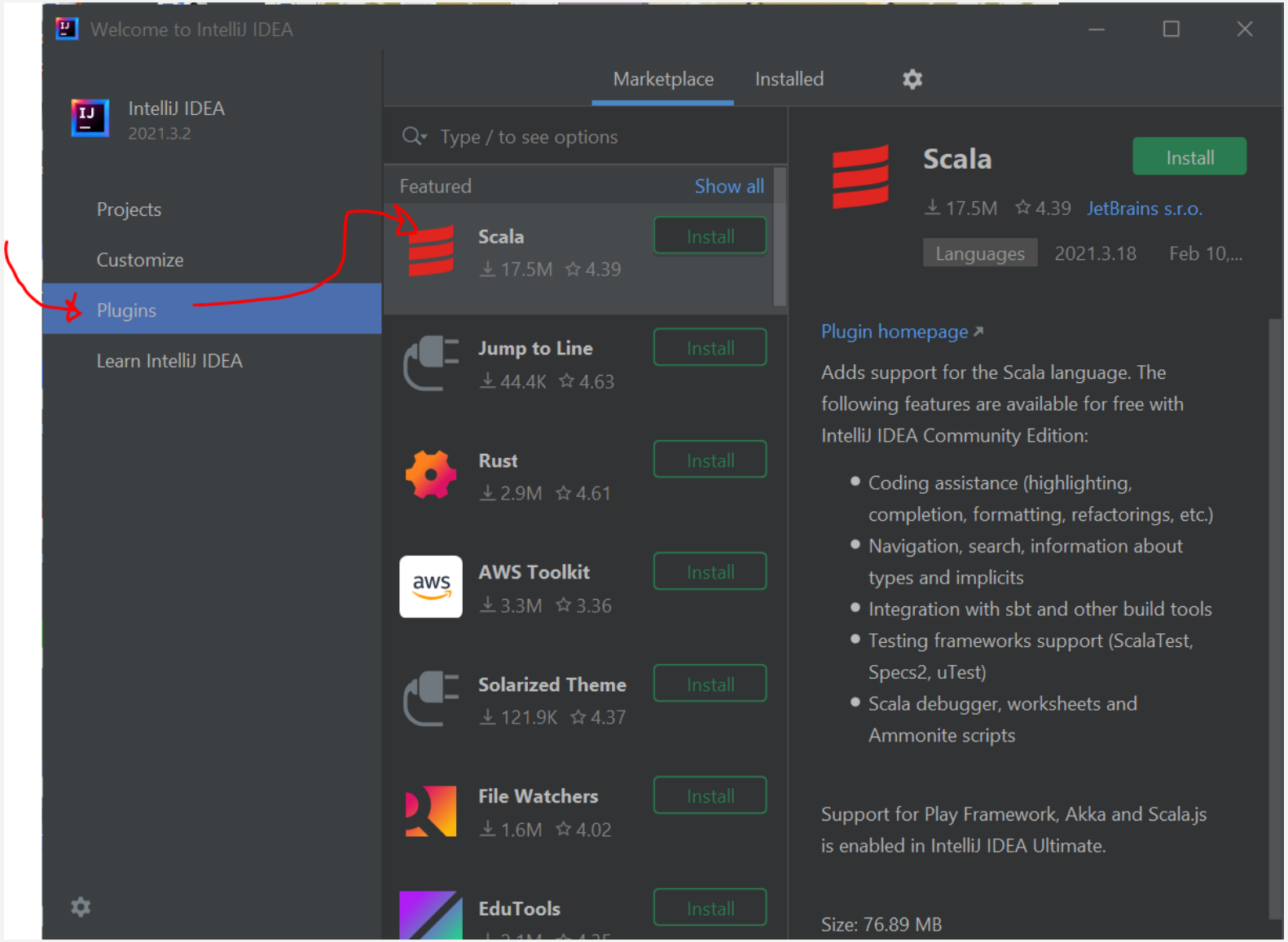
WHAT ABOUT ANY IDE, ECLIPSE?

- Let's install Scala IDE for Eclipse.
- Go to Eclipse Marketplace



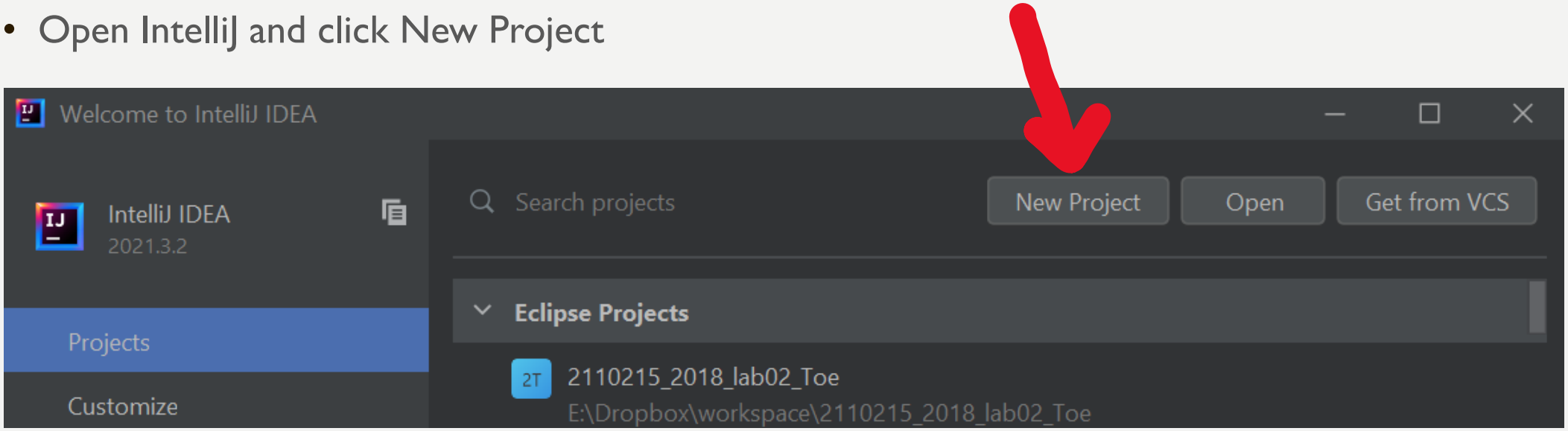
WHAT ABOUT INTELLIJ?

- Install IntelliJ
- Then install Scala plugin

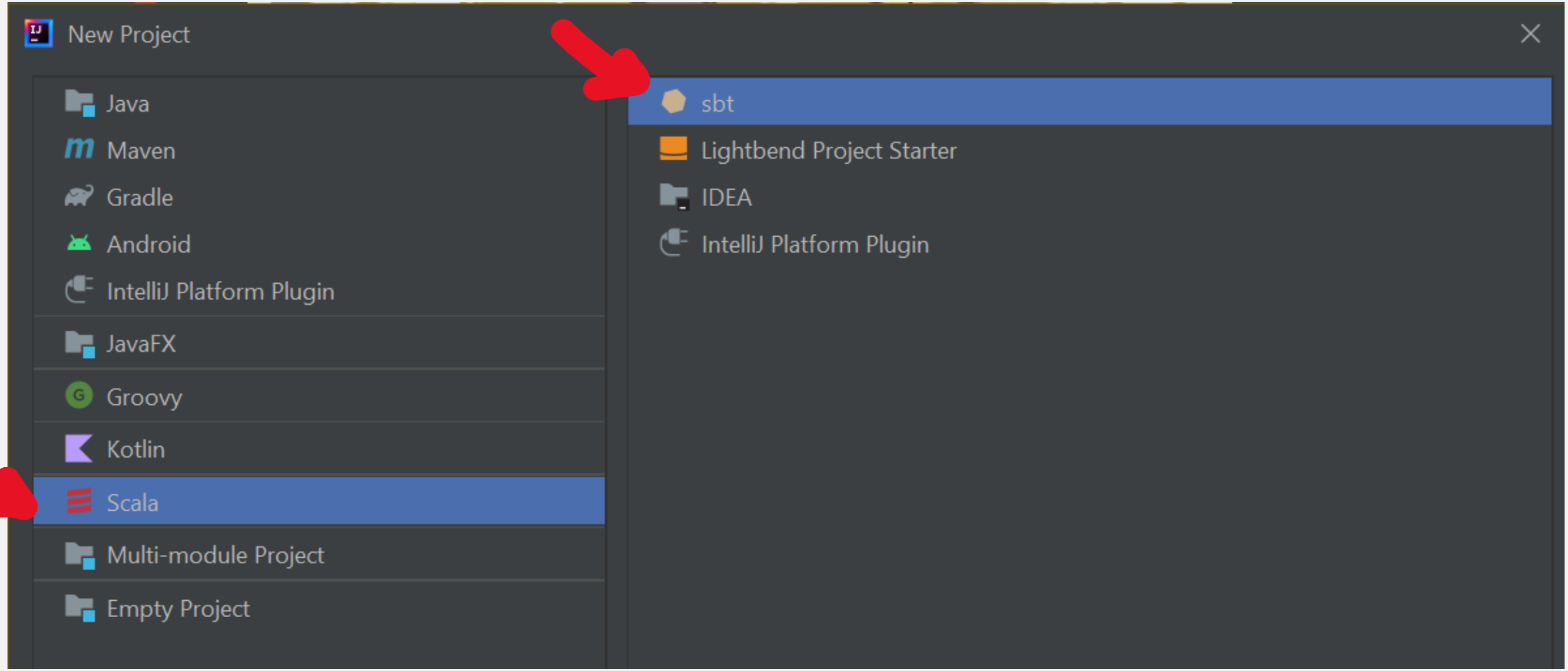


YOUR "HELLO WORLD" PROJECT

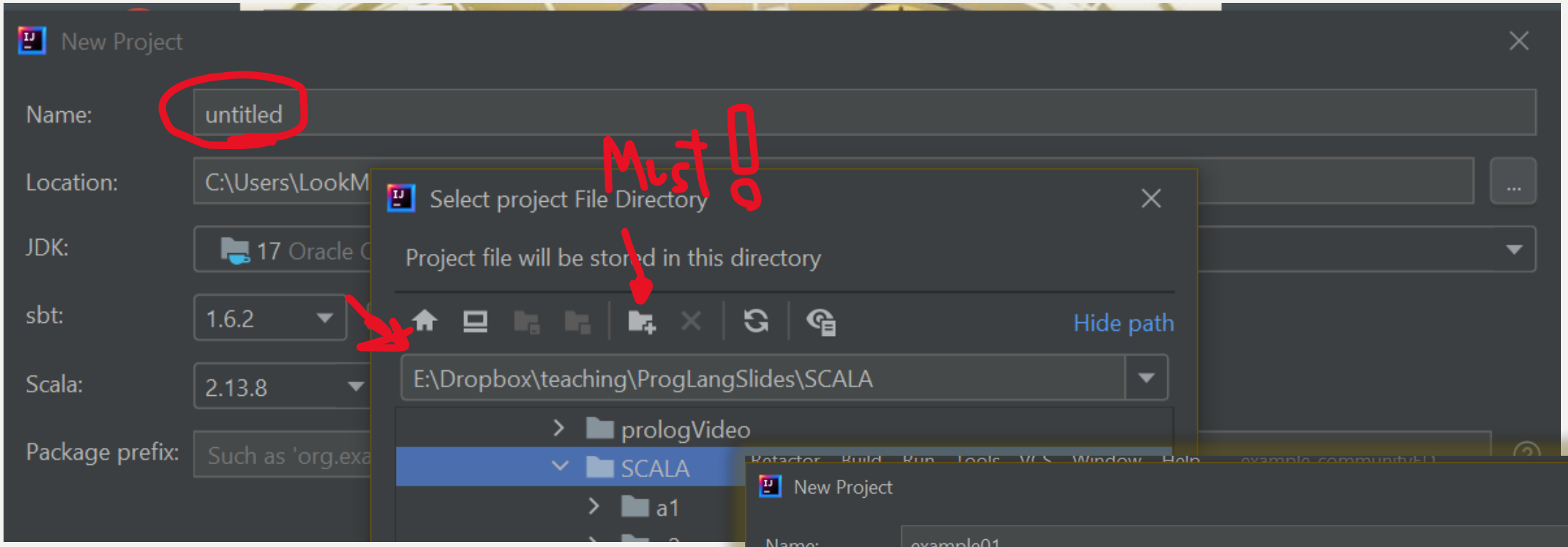
- Open IntelliJ and click New Project

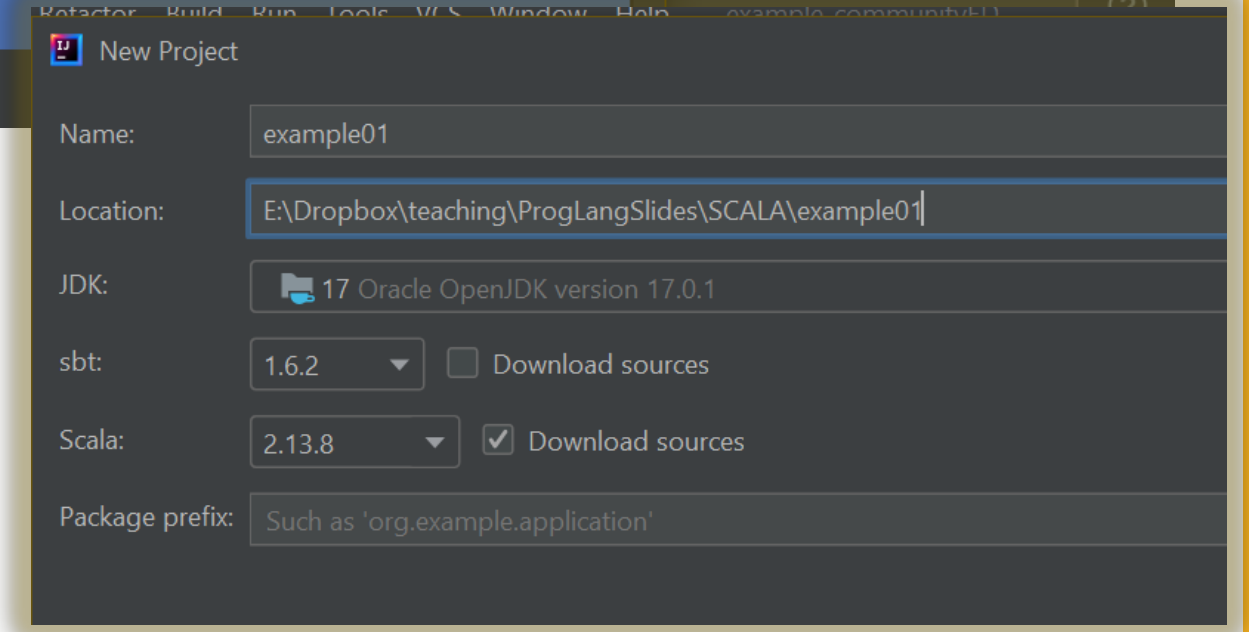


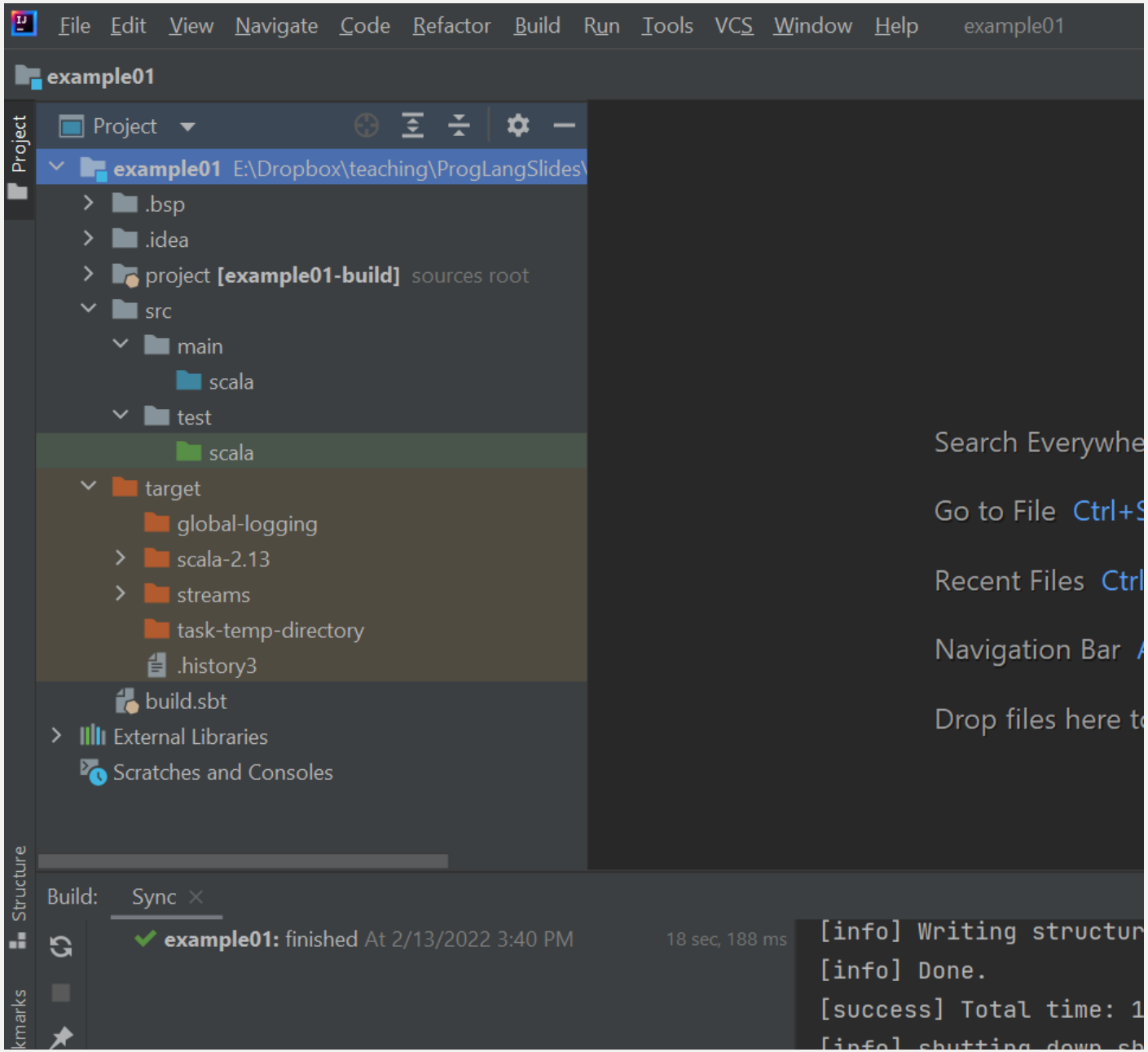
- Select sbt project.



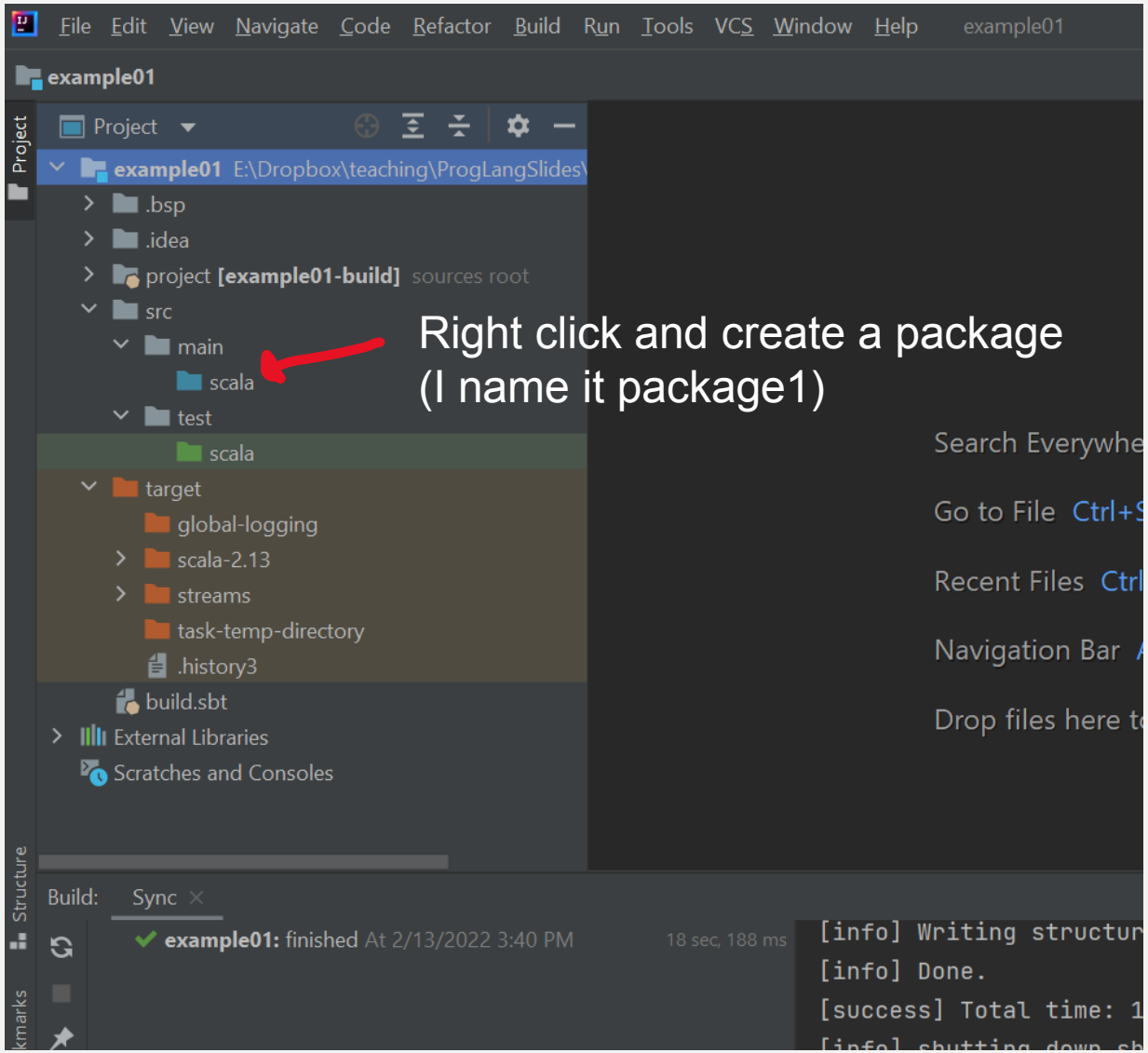
- Name your project and choose its folder.

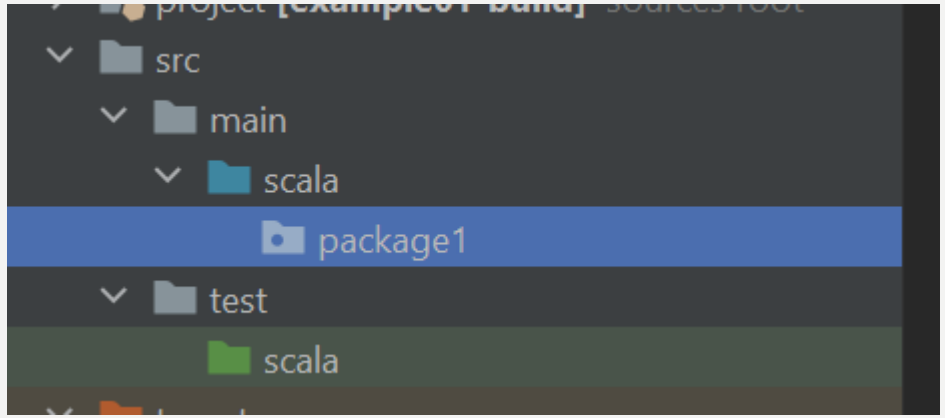




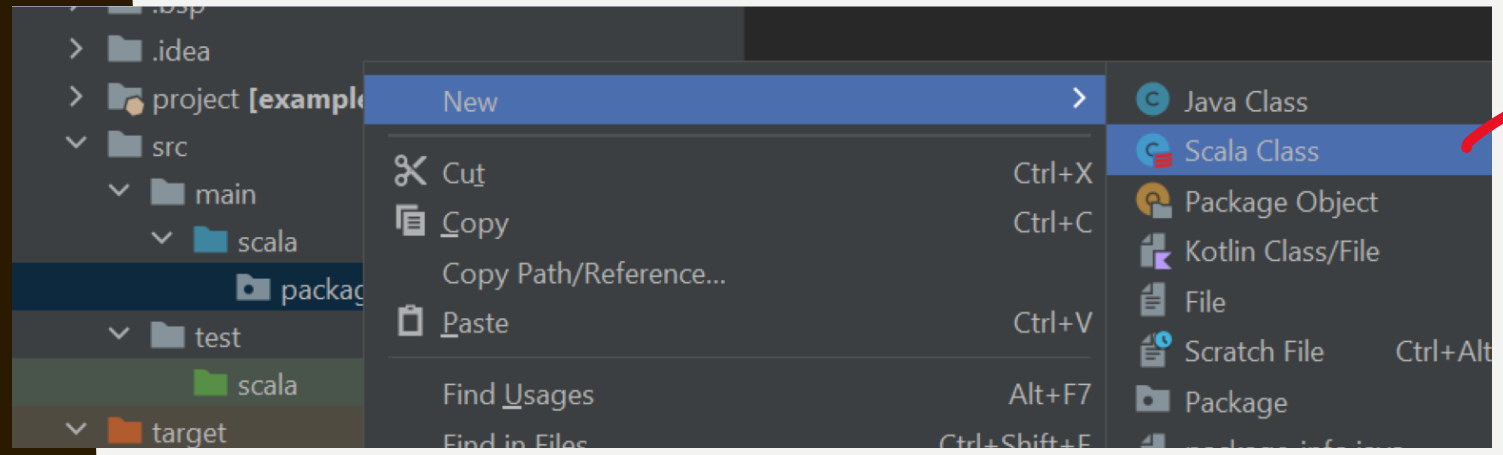

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help example01
example01
Project
example01 E:\Dropbox\teaching\ProgLangSlides\
.bsp
.idea
project [example01-build] sources root
src
main
scala
test
scala
target
global-logging
scala-2.13
streams
task-temp-directory
.history3
build.sbt
External Libraries
Scratches and Consoles
Search Everywhe
Go to File Ctrl+S
Recent Files Ctrl
Navigation Bar
Drop files here t
Build: Sync
example01: finished At 2/13/2022 3:40 PM
18 sec, 188 ms
[info] Writing structur
[info] Done.
[success] Total time: 1

LET'S CREATE A PACKAGE

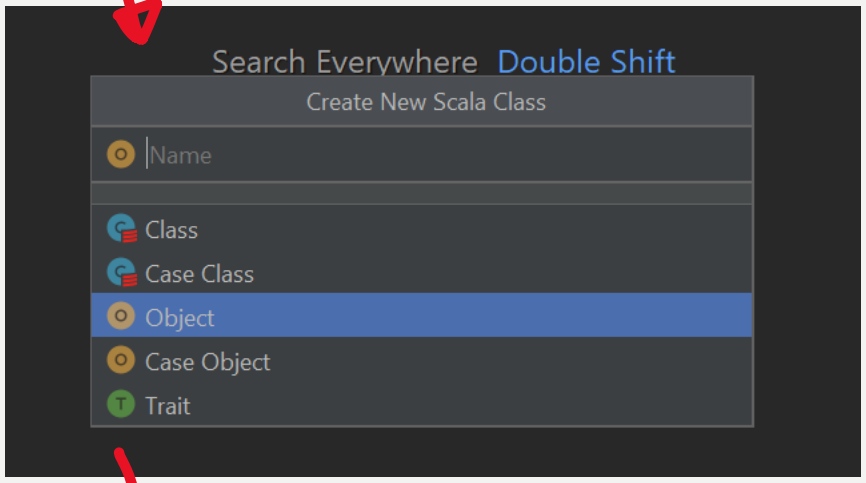


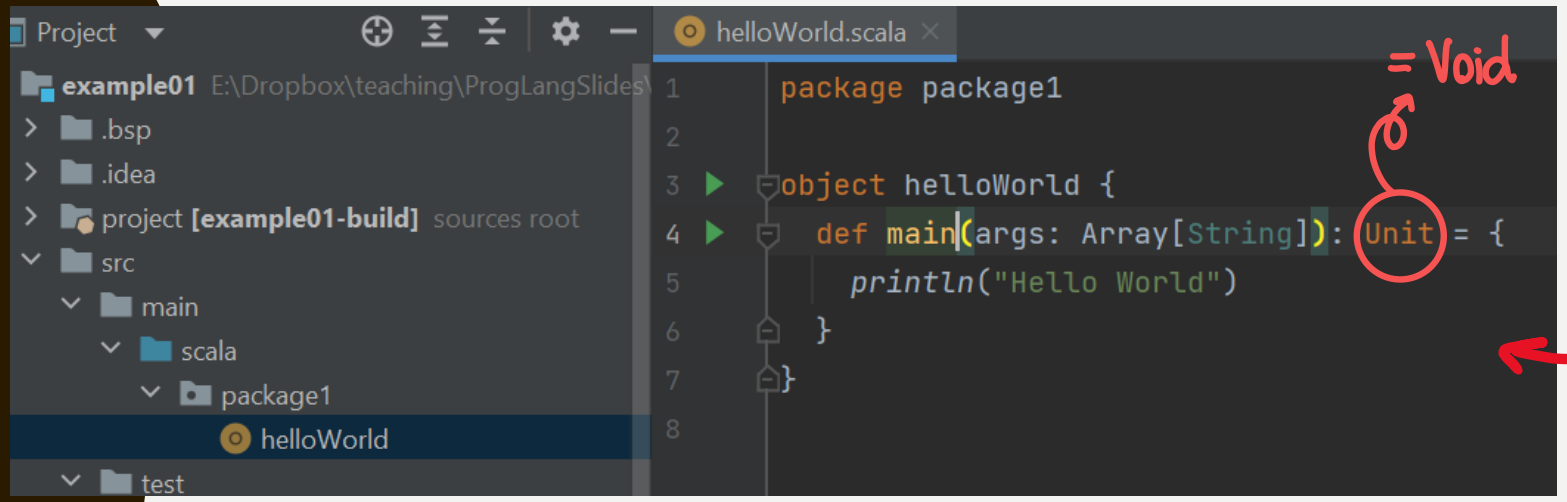


THEN CREATE A SCALA OBJECT THAT HAS “HELLO WORLD”



In the pop up,
change it to object and name it!

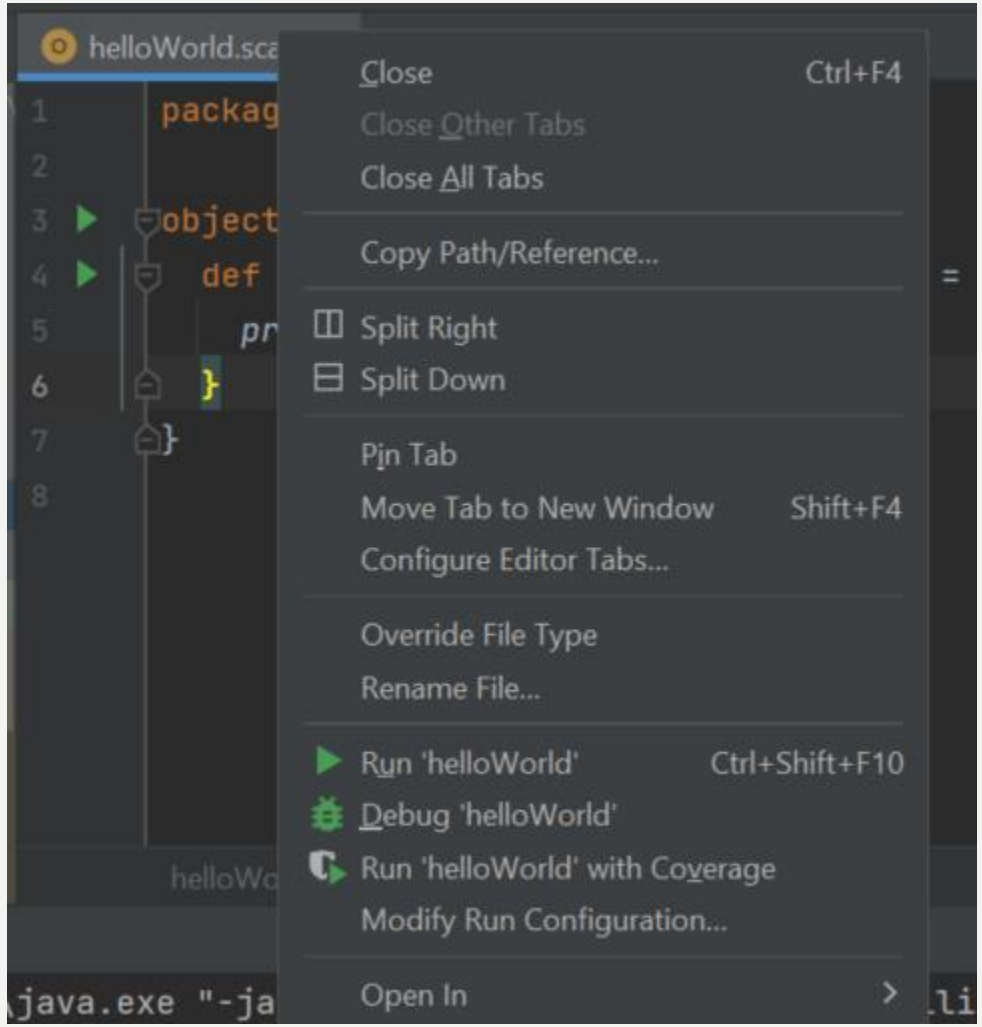




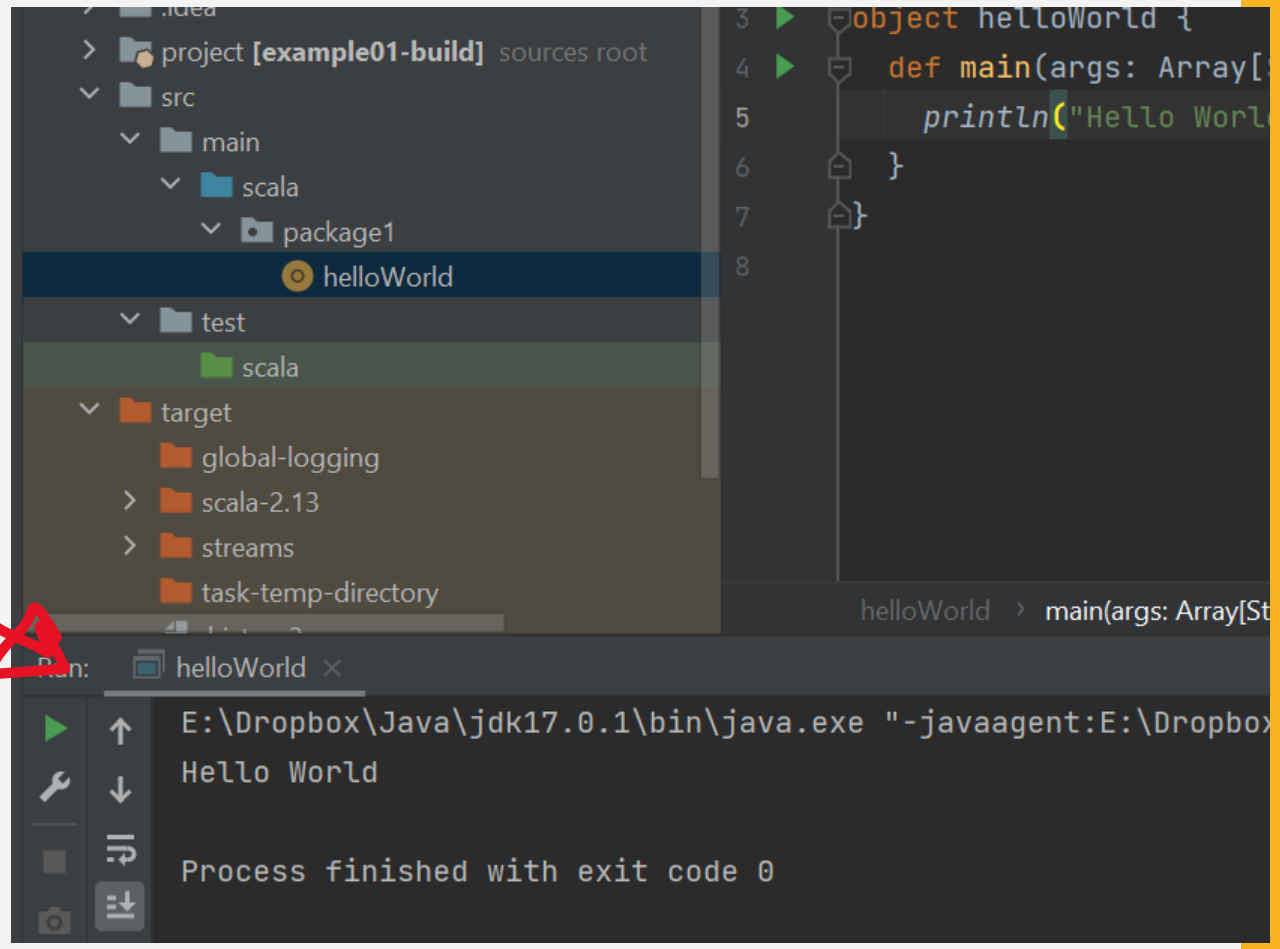
Main method is similar to Java’s.

NOW WE RUN THE PROGRAM

Right click on the tab or inside the file.
Then choose Run 'helloWorld'



If you run your program for the first time, it may take a while.



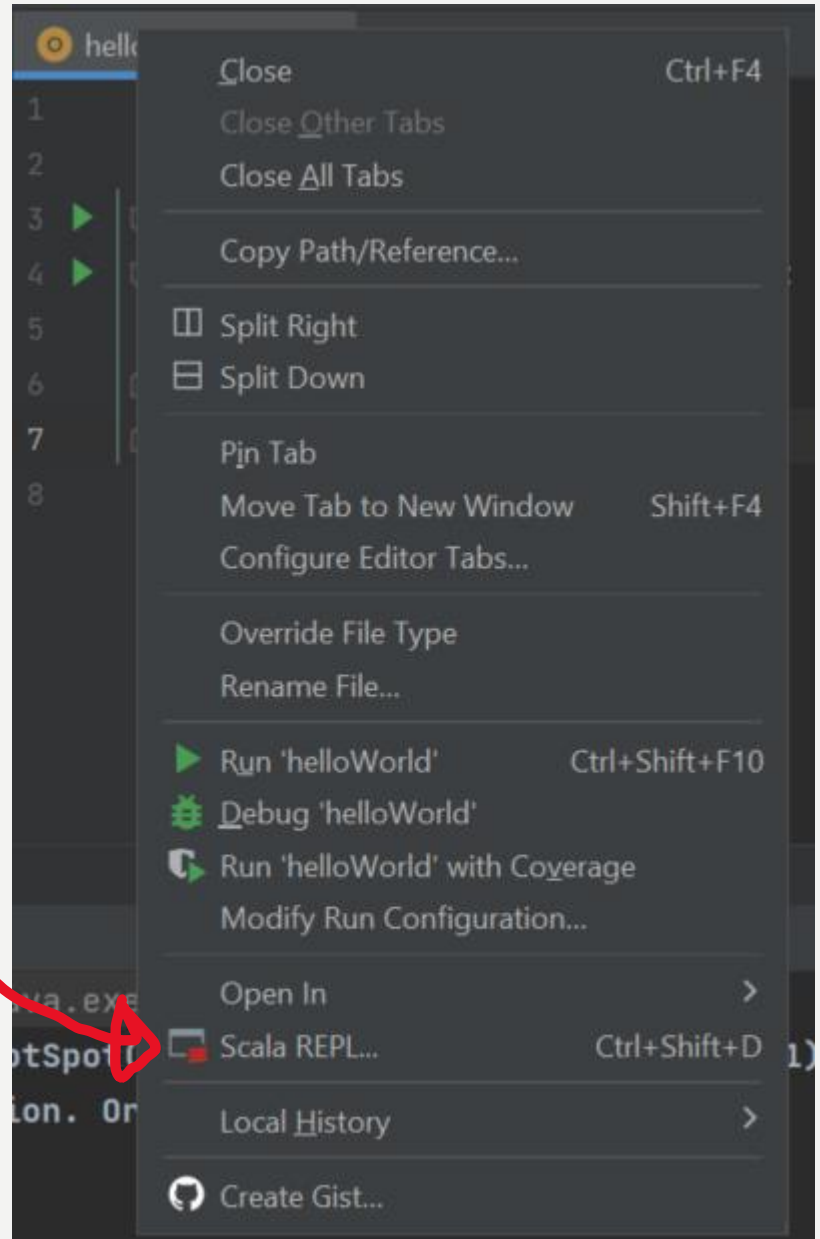
LET'S LOOK AT THE CODE

An instance of class helloWorld. (A class like this cannot have another instance. It is a Singleton!)

```
object helloWorld {
  def main(args: Array[String]){
    println("Hello World")
  }
}
```

Used to define method.

TO RUN REPL IN INTELLIJ




```
scala> object helloWorld {
   def main(args: Array[String]): Unit = {
     println("Hello World")
   }
 }
     |       |       |       | object helloWorld
```

It tells us what is defined.

Empty array

```
scala> helloWorld.main(Array(""))
Hello World
```

STRING INTERPOLATION

- Concatanation: this is just like Java.

```
object helloWorld {
  def main(args: Array[String]): Unit = {
    var name = "Tanjiro"
    var age  = 15
    println("Hello " + name + ", age =" + age)
  }
}
```

- S string interpolation

```
object helloWorld {
  def main(args: Array[String]): Unit = {
    var name = "Tanjiro"
    var age  = 15
    //println("Hello "  + name + ", age =" + age)
    println(s"$name is $age years old.")

  }
}
```

***

Comment uses //
Or /* */ just like in Java.

```
E:\Dropbox\Java\jdk17.0.1
Tanjiro is 15 years old.
```

- F string interpolation (type safe)

```scala
object helloWorld {
  def main(args: Array[String]): Unit = {
    var name = "Tanjiro"
    var age  = 15.5
    //println("Hello "  + name + ", age =" + age)
    //println(s"$name is $age years old.")
    println(f"$name%s is $age%d years old.")
  }
}
```

check ว่า age เป็น type int หรือเปล่า

Note that the type here is not int.

```
example01: build failed At 2/13/2022 7:12 PM with 1 1 sec, 493 ms
  Chart
  helloWorld.scala src\main\scala\package1 1 error
    type mismatch; :9
```

```
E:\Dropbox\teaching\ProgLangSlides\SCALA\exam
type mismatch;
 found   : Double
 required: Int
    println(f"$name%s is $age%d years old.")
```

- Raw string interpolation

```
println("Kamado \nNezuko")
println(raw"Kamado \nNezuko")
```

ไม่เอา Character พิง

```
Run:  Scala REPL ×  helloV
Kamado
Nezuko
Kamado \nNezuko
```

IF-ELSE

```
object IfElseExample {
  def main(args: Array[String]): Unit = {
    var age = 15
    var x = 3;
    var message = ""
    if(age == 15){
      message = "age is 15"
      x += 1
    } else{
      message = "age is NOT 15"
      x -= 1
    }
    println(message)
    println(x)
  }
}
```

Don’t forget to initialize!

```
E:\Dropbox\Ja
age is 15
4
```

A MORE COMPLEX IF EXAMPLE

```
def main(args: Array[String]): Unit = {
  var a = 15
  var b = 3;
  var c = 20
  if(a<16){
    if(b>3 && c <=20){
      println("case 1.1")
    }else if (b>3 && c ==20){
      println("case 1.2")
    }else if (b>3 && c>20){
      println("case 1.3")
    }else {
      println("case 1.4")
    }
  } else if (a == 16 || b!=4){
    println("case 2.1")
  } else {
    println("case 3.1")
  }
}
```

มีก็ได้ไม่มีก็ได้
(แต่ไม่มีเถอะ)

And, or, not, nested if are just like Java!

IF EXPRESSION

- Very similar to C++

```
object IfExpression {
  def main(args: Array[String]): Unit = {
    var age = 15
    var x = 3;
    var message = ""

    var result = if(age !=15) "age is not 15" else "age is 15"
    println(result)

  }
}
```

```
E:\Dropbox\J
age is 15
```

MATCH (SWITCH STATEMENT)

```
object MatchStatement {
  def main(args: Array[String]): Unit = {
    var x = 45
    x match {
      case 10 => println("x is 10")
      case 20 => println("x is 20")
      case 25 => {
        println("x is 25")
        println("and that's it")
      }
      case 30 => println("x is 30")
      case _ =>
    }
  }
}
```

default

ไม่ Fall through
(ไม่ต้องมี break)

- Can be used with other data types like string
- Does not need a “break” statement

Default is doing nothing

MATCH EXPRESSION

```scala
def main(args: Array[String]): Unit = {
  var x = 25
  var res = x match {
    case 10 => 10.0
    case 20 => 20.0
    case 25 => {
      25.0
      x = 33
    }
    case 30 => 30.0
    case _ =>
  }
  println(res)
  println(x)
}
```

เอาค่าที่ได้จากการ match มาใส่ res

ไม่มีของไปใส่

```
10.0
10
```

→ res

→ ค่า x

กรณี x = 10

```
()
33
```

ไม่มีค่า

→ res

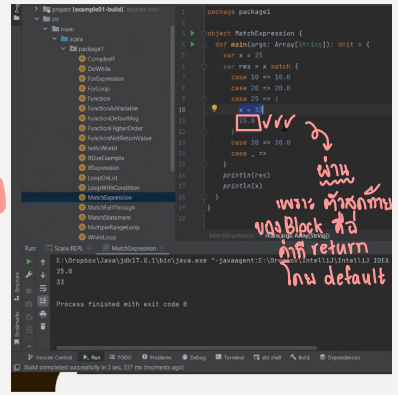
```
()
44
```

```scala
package package1

object MatchExpression {
  def main(args: Array[String]): Unit = {
    var x = 44
    var res: Double = x match {
      case 10 => 10.0
      case 20 => 20.0
      case 25 => {
        25.0
        x = 33
      } : Unit
      case 30 => 30.0
      case _ =>
    }
    println(res)
    println(x)
  }
}
```

พอ Fix type ไว้จะ Error

```
MatchExpression > main(args: Array[String])
missing parameter type for expanded function ((<x$1: error>, x$2) => x$1.$plus(x$2))
    var zz = (_+_)
```



ผ่าน เพราะ ตัวสุดท้าย ของ Block ที่ว่า คือที่ return โดน default ♡♡♡

MATCH WITH MULTIPLE CASES (FALL THROUGH)

```
object MatchFallThrough {
  def main(args: Array[String]): Unit = {
    var x = 35
    x match {
      case 10 | 20 | 30 | 40 | 50 => println(s"x is $x")
      case 25 | 35 | 45 | 55 => {
        println(s"x is $x")
        println("and that's it")
      }
      case _ =>
    }
  }
}
```

WHILE LOOP

```
object WhileLoop {
  def main(args: Array[String]): Unit = {
    var x =0
    while(x<10){
      x += 1      //x++, ++x are NOT allowed in Scala
      println(x)
    }
  }
}
```

**

DO WHILE

```
def main(args: Array[String]): Unit = {
  var x =0
  do{
    x += 1     //x++, ++x are NOT allowed
    println(x)
  } while(x<0)
}
```

- This loop executes only once!

FOR LOOP

พิมพ์แค่ 0 ถึง 9

```
object ForLoop {
  def main(args: Array[String]): Unit = {
    for(x <- 0 ≤ to ≤ 9) { //step by 1 in each iteration
      println(x)    //print 0 to 9
    }
  }
}
```

```
for(x <- 0 ≤ .to( ≤ 9))
```

can also be used.

```
for(x <- 0 ≤ .until( < 10))
```

until จะไม่รวมค่าสุดท้าย

MULTIPLE RANGE FOR LOOP

```
object MultipleRangeLoop {
  def main(args: Array[String]): Unit = {
    for(x <- 0 ≤ .until( < 5); i <- 0 ≤  to  ≤ 4) {
      println(s"$x , $i")
    }
  }
}
```

loop ชั้นที่ 1

loop ชั้นที่ 2

```
0 , 0
0 , 1
0 , 2
0 , 3
0 , 4
1 , 0
1 , 1
1 , 2
1 , 3
1 , 4
```

This is like a nested loop.

LOOP ON A LIST

```
object LoopOnList {
  def main(args: Array[String]): Unit = {
    var mylist = List(1,3,5,7)
    for(m <- mylist) {
      println(m)
    }
  }
}
```

แสดงทุกอย่างใน list นี้

```
E:\Dropbox\Ja
1
3
5
7
```

FOR LOOP WITH BOOLEAN CONDITION

```
object LoopWithCondition {
  def main(args: Array[String]): Unit = {
    for(x <- 0 ≤ until < 5; if x%2==0) {
      println(x)
    }
    println("--------------------")
    var mylist = List(1,3,5,7)
    for(m <- mylist; if m >= 3 ) {
      println(m)
    }
  }
}
```

ถ้าตรงเงื่อนไขนี้
จะทำสิ่งที่อยู่ใน loop
ถ้าไม่ตรงจะข้ามไปตัวต่อไป

```
0
2
4
--------------------
3
5
7
```

- It goes through every value, but only execute code inside the loop if the condition is satisfied.

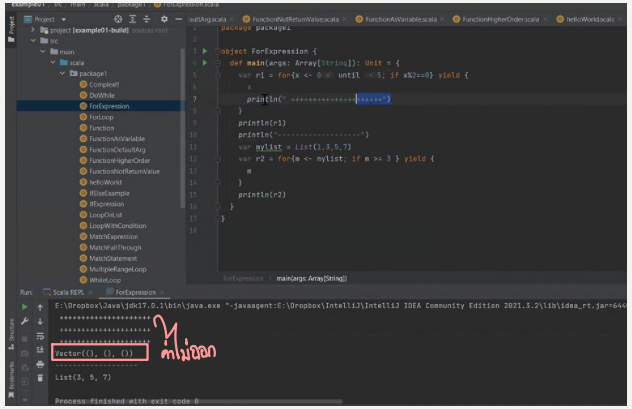
FOR LOOP EXPRESSION

เอาค่าจาก for มาใส่ r1

```
def main(args: Array[String]): Unit = {
  var r1 = for{x <- 0 ≤ until < 5; if x%2==0} yield {
    x
  }
  println(r1)
  println("-------------------")
  var mylist = List(1,3,5,7)
  var r2 = for{m <- mylist; if m >= 3 } yield {
    m
  }
  println(r2)
}
```

มีโค้ดตรงนี้ได้

ห้ามมีโค้ดต่อตรงนี้ ไม่งั้นมันจะไม่มีข้อมูลไปเก็บใน r1



ถ้าเป็นเลขช่วงแล้วมา for ตัวแปรที่มารับจะเป็น vector

```
E:\Dropbox\Java\jdk1
Vector(0, 2, 4)
-------------------
List(3, 5, 7)
```

เพราะเป็นของใน list

HOW TO WRITE YOUR OWN FUNCTION

```
object Function {
  def main(args: Array[String]): Unit = {
      println(area( width = 2, height = 3))
      println(areaScale(4,5))
  }

  def area(width: Int, height: Int): Int = {
    width * height
  }
  def areaScale(w: Int, h: Int): Int ={
    val w2 = w+1    // w += 1 is not allowed
    val h2 = h+1
    w2*h2  //last statement will be returned (you can use "return")
  }
}
```

แอปใส่ Function. ให้

ใช้ไม่เขียนก็ได้ ถ้าเราเชื่อมั่นใจว่าโปรแกรม (ทราบค่าที่ return)

Return type

บรรทัดสุดท้ายของฟังก์ชันคือ ค่า Return

!!!

แก้ : สร้างตัวแปรใหม่

Short function

```
def add(x:Int,y:Int):Int = x+y
```

***

Return type can even be removed if it is known for sure.

FUNCTION BELONGS TO AN OBJECT

```
object Function {
  object Math {
    def addM(x:Int,y:Int):Int = x+y
  }

  def main(args: Array[String]): Unit = {
    println(Function.area( width = 2, height = 3))
    println(areaScale(4,5))
    println(Math.addM(5,3))
  }

  def area(width: Int, height: Int): Int = {
    width * height
  }
```

ใช้ "+" ไม่ได้ก็ได้ (ไม่ใช่ operator overload)

อาจไม่มีก็ได้

ต้องมี!

objec ซ้อน object

You can use + here. It's not operator overload. It's just that it can be a function name. And it is used just like a function of object Math.

In fact +, -, * ,/ are not an operator in Scala. They are functions.

FUNCTION WITH 1 ARGUMENT

```
object Function {
  object Math {
    def addM(x:Int,y:Int):Int = x+y
    def squareM(x:Int):Int = x*x
  }

  def main(args: Array[String]): Unit = {
      println(Function.area( width = 2, height = 3))
      println(areaScale(4,5))
      println(Math.addM(5,3))  //function of object Math
      println(Math.squareM 3)  //one argument function call
  }
```

One Argument แบบนี้ได้

FUNCTION CAN HAVE DEFAULT ARGUMENT VALUE

```
object FunctionDefaultArg {
  object Math {
    def addM(x:Int =1,y:Int =1):Int = x+y
    def squareM(x:Int = 1):Int = x*x
  }

  def main(args: Array[String]): Unit = {
    println(Math.addM())
    println(Math.squareM())
  }
}
```

ค่า default

ตัวที่หายไป มันจะเอา
ค่า default มาให้

```
E:\Dropb
2
1
```

You can provide some first parameters too

```
println(Math.addM(5))
```

คือว่าเป็น parameter แรก

FUNCTION THAT DOES NOT RETURN VALUE

```scala
object FunctionNotReturnValue {
  def f1(x:Int):Unit ={
    println(s"x is given = $x")
  }
  def main(args: Array[String]): Unit = {
    f1(3)
  }
}
```

void

FUNCTION AS VARIABLE (ANONYMOUS FUNCTION)

```scala
object FunctionAsVariable {
  def main(args: Array[String]): Unit = {
    var x = (a:Int, b:Int) => a+b
    var z = (a:Int, b:Int) => {
      var c = a+b
      c*c
    }
    println(x(5,7))
    println(z(2,3))
  }
}
```

ฟังก์ชันไม่มีชื่อ

```
E:\Drop
12
25
```