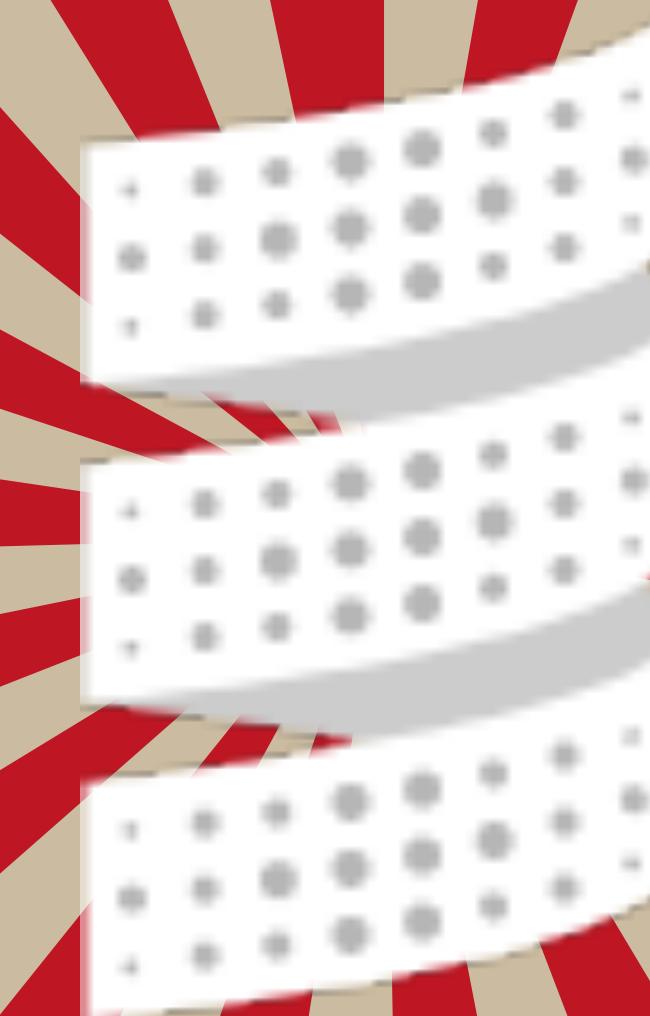


# ESCALATE WITH SCALA 3



Daniel Hinojosa

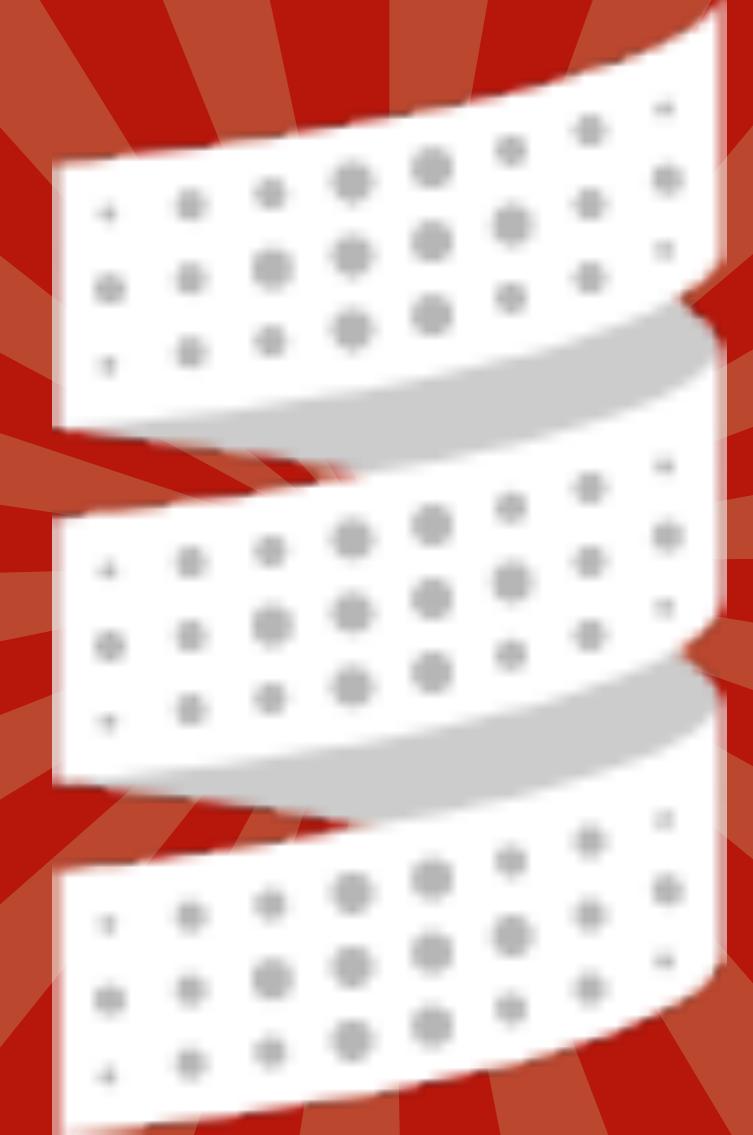
@dhinojosa

[dhinojosa@evolutionnext.com](mailto:dhinojosa@evolutionnext.com)

# Content and Material

[https://github.com/dhinojosa/oreilly\\_escalate\\_scala\\_3](https://github.com/dhinojosa/oreilly_escalate_scala_3)

# Scala Today



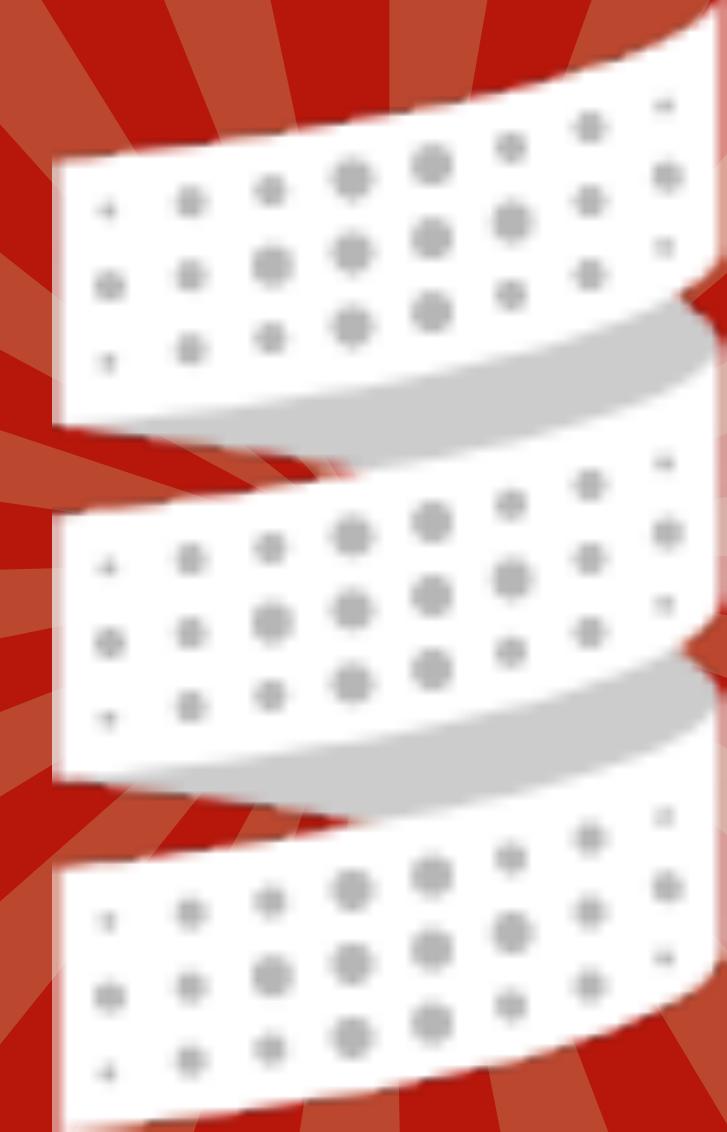


- ▶ Enjoyed by many JVM Developers
- ▶ Particularly those that desire type discipline and functional programming
- ▶ First appeared in 2004
- ▶ Designed at [École Polytechnique Fédérale de Lausanne](#)
- ▶ [scala-lang.org](#)



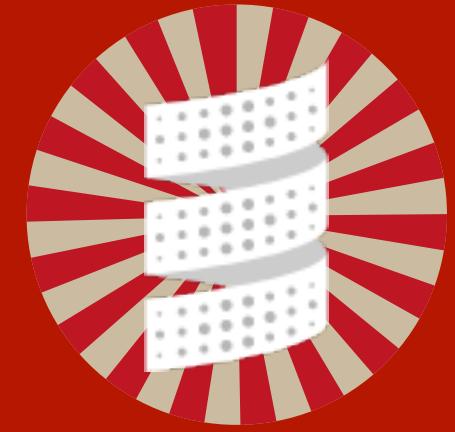
- ▶ New Compiler
- ▶ Scheduled for Release 2021
- ▶ <http://dotty.epfl.ch/>
- ▶ Currently @ 3.0.0-M3
- ▶ Most Scala 2 code compiles to Scala 3
- ▶ Things are still-a-changin'!

# How to Install Scala 3





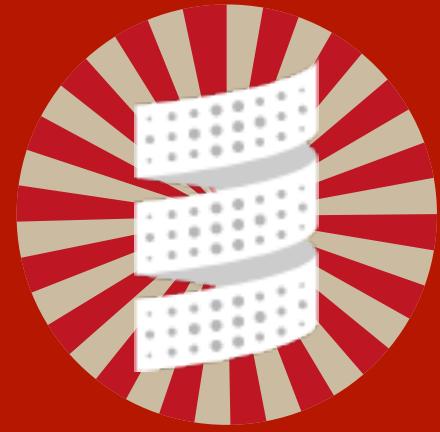
- Scala 3 currently *has not been released*
- Best way to "Install Scala 3" is through SBT (Simple Build Tool), version 1.4 and higher.
- Best *IDE* right now is VisualStudioCode with Metals plugin
- Alternate means: You can also download and build Scala 3 by downloading, unzip, and map it in your shell as your preferred Scala.



# Using SBT and Giter8

```
> sbt new scala/scala3.g8
```

```
name [Scala 3 Project Template]: Scala 3 Demo
```



# Manually Create a Project

- Create a *build.sbt* file
- Include name, version, description settings
- Set `scalaVersion` setting to the latest 3 version available (e.g. 3.0.0-M3)
- Set any `scalacOptions` required and `libraryDependencies`
- Open *project/plugins.sbt* file, and add `addSbtPlugin("ch.epfl.lamp" % "sbt-dotty" % "0.5.1")`

Demo:

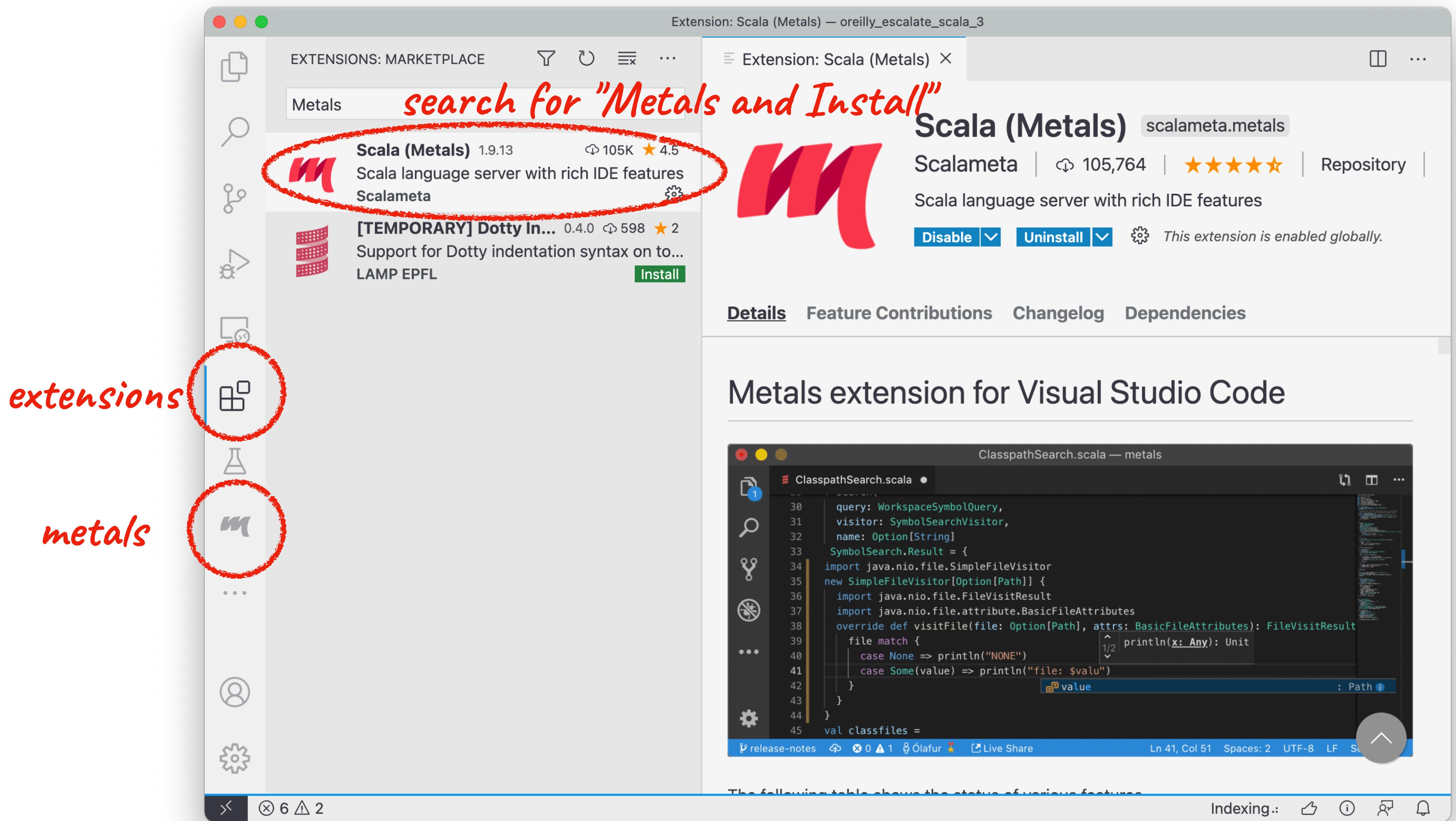
`build.sbt`

`project/plugins.sbt`



scalajs

Visual Studio Code



The screenshot shows the IntelliJ IDEA interface with the following details:

- Title Bar:** 1\_UseTuples.scala — oreilly\_escalate\_scala\_3
- Toolbars:** EXPLORER, OPEN EDITORS, OREILLY\_ESCALATE\_SCALA\_3, OUTLINE, NPM SCRIPTS.
- Editor Area:** Displays the Scala code for '1\_UseTuples.scala'. The code uses parameter untupling instead of pattern matching for multiplication.

```
1 package com.xyzcorp.demo.tuples
2
3 /**
4 * The following is different in Scala 3, since we
5 * do not need {case (x,y) => x * y}, we can just
6 * treat the tuple as separate arguments, this is called
7 * parameter untupling
8 */
run | debug
9 @main def assertUsingTuplesInFunctions:Unit =
10   val xs = List(1.0, 4.0)
11   val ys = List(10.0, 50.0)
12   println(xs.zip(ys).map(_ * _).sum)
```

- Status Bar:** 0 ▲ 0, Launch UsingJavaIntersectionTypes (oreilly\_escalate\_scala\_3), Column Selection, Ln 7, Col 23, Spaces: 2, UTF-8, LF, Scala, ⌂, ⌂, ⌂.

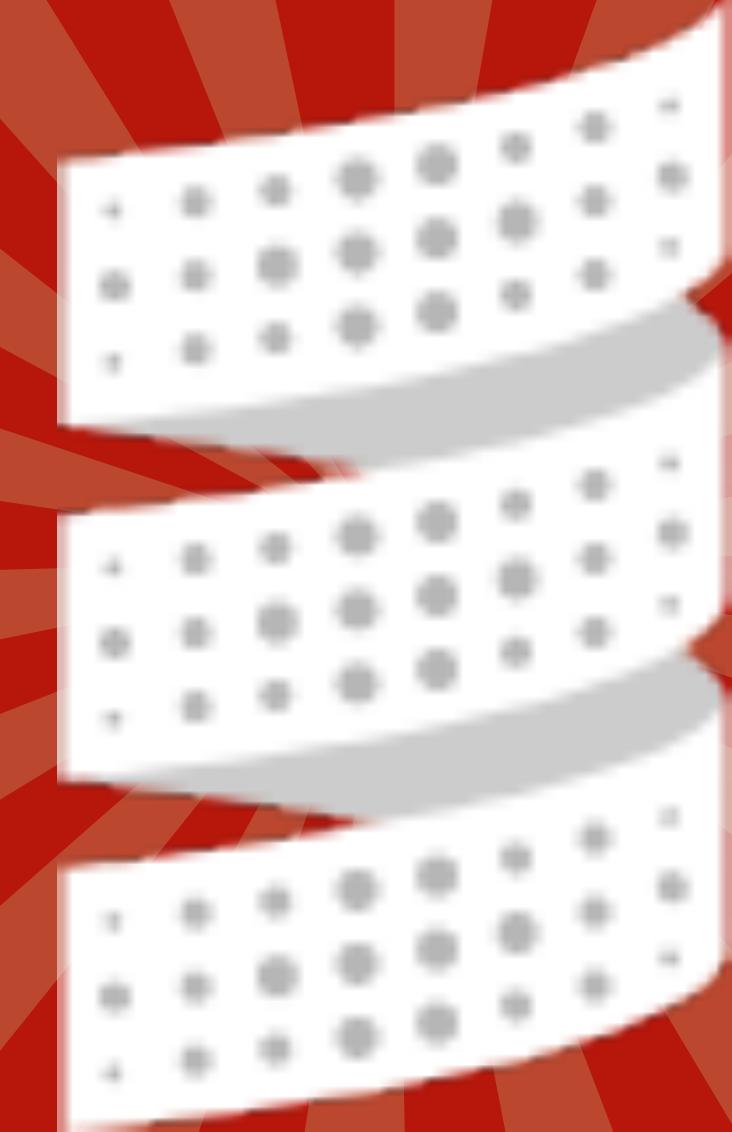
▶ File | Open | Select Scala Project Root Folder



# Other Editor Alternatives

- IntelliJ IDEA Scala Plugin
- Metals Works with more than just VSCode
  - VIM, Sublime, Emacs, Eclipse
  - <https://scalameta.org/metals/docs/editors/overview.html>

# Introduction to Scala 3





**scalaj**

Aesthetic Changes



- Optionally, now we can write our code without brackets/braces. Think Python, or don't, if it will cause you acid reflux
- Violations are flagged as *warnings*
- Internally, compiler will insert <indent> and <outdent> tags
- Can be turned off with -noindent flag
- Indentations and Braces can be intermingled

Demo: `src/main/scala/com.xyzcorp.demo.optionalbraces.*`



- @main annotations marks a method as a main method
- Situated either at the top-level or in a statically accessible object
- App will be deprecated, maybe.
- Arguments to the main method will be converted to parameters if there is a `scala.util.FromString` type
- If enough arguments aren't provided or wrong quality, then your app will terminate with an error message

Demo:

```
src/main/scala/com.xyzcorp.demo.mainmethod.*
```

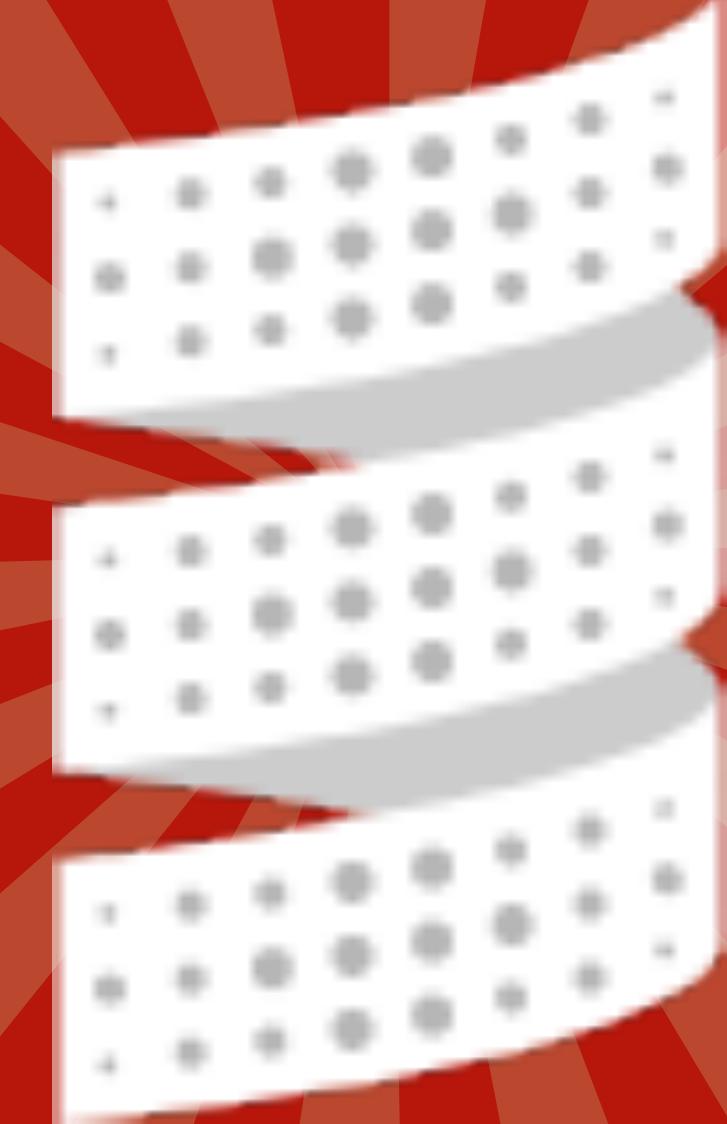


- `if`, `else`, `else if` doesn't need a condition wrapped in a parenthesis but can be followed by `then`.
- for "loops" (not comprehensions) will make use of `do` keyword if the condition is not wrapped in parenthesis
- `for` comprehension where the condition has no parenthesis can use the `yield` keyword
- `while` loops do not need parenthesis, and would also use `do` keyword

Demo:

`src/main/scala/com.xyzcorp.demo.newcontrolsyntax.*`

# Creating Data Types





# scala<sup>3</sup>

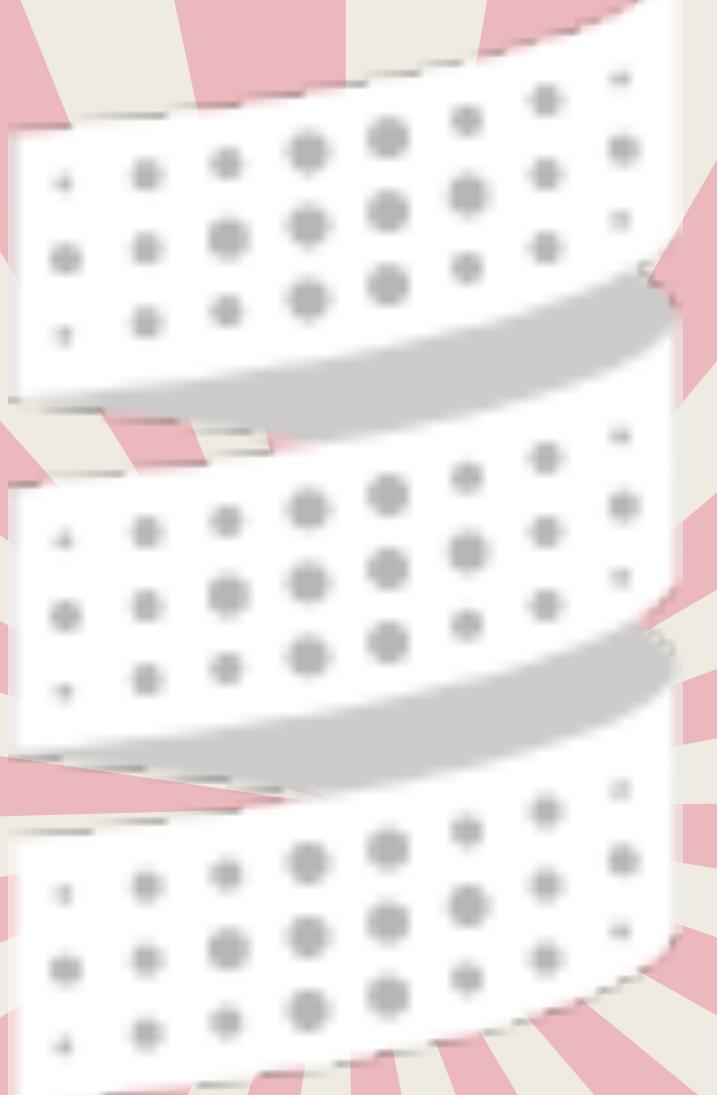
## Intersection Types



- ▶ Combination of two or more types
- ▶ "And" relationship
- ▶ Very akin to how they are implemented in Java

Demo:

`src/main/java/com.xyzcorp.demo.intersectiontypes.*`  
`src/main/scala/com.xyzcorp.demo.intersectiontypes.*`



# scala<sup>3</sup>

## Union Types

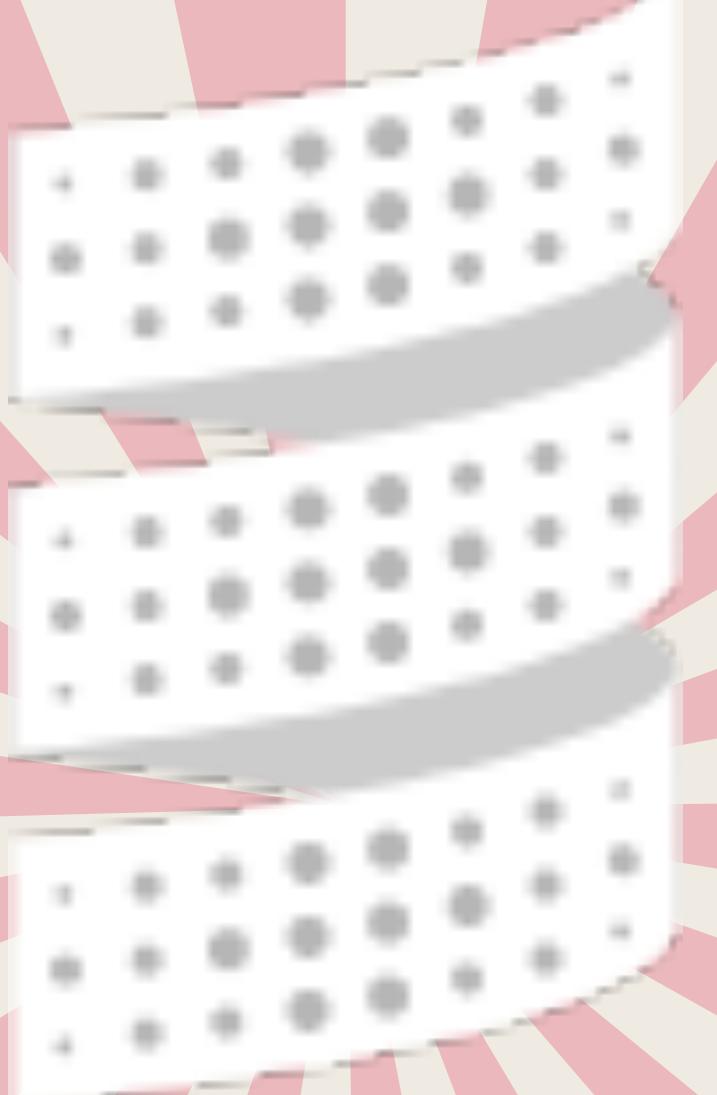


- ▶ Combination of two or more types
- ▶ "Or" relationship
- ▶ Very akin to how they are implemented in Haskell or Elm

```
data Bool = False | True
```

Demo:

[src/main/scala/com.xyzcorp.demo.uniontypes.\\*](#)



# scala<sup>3</sup>

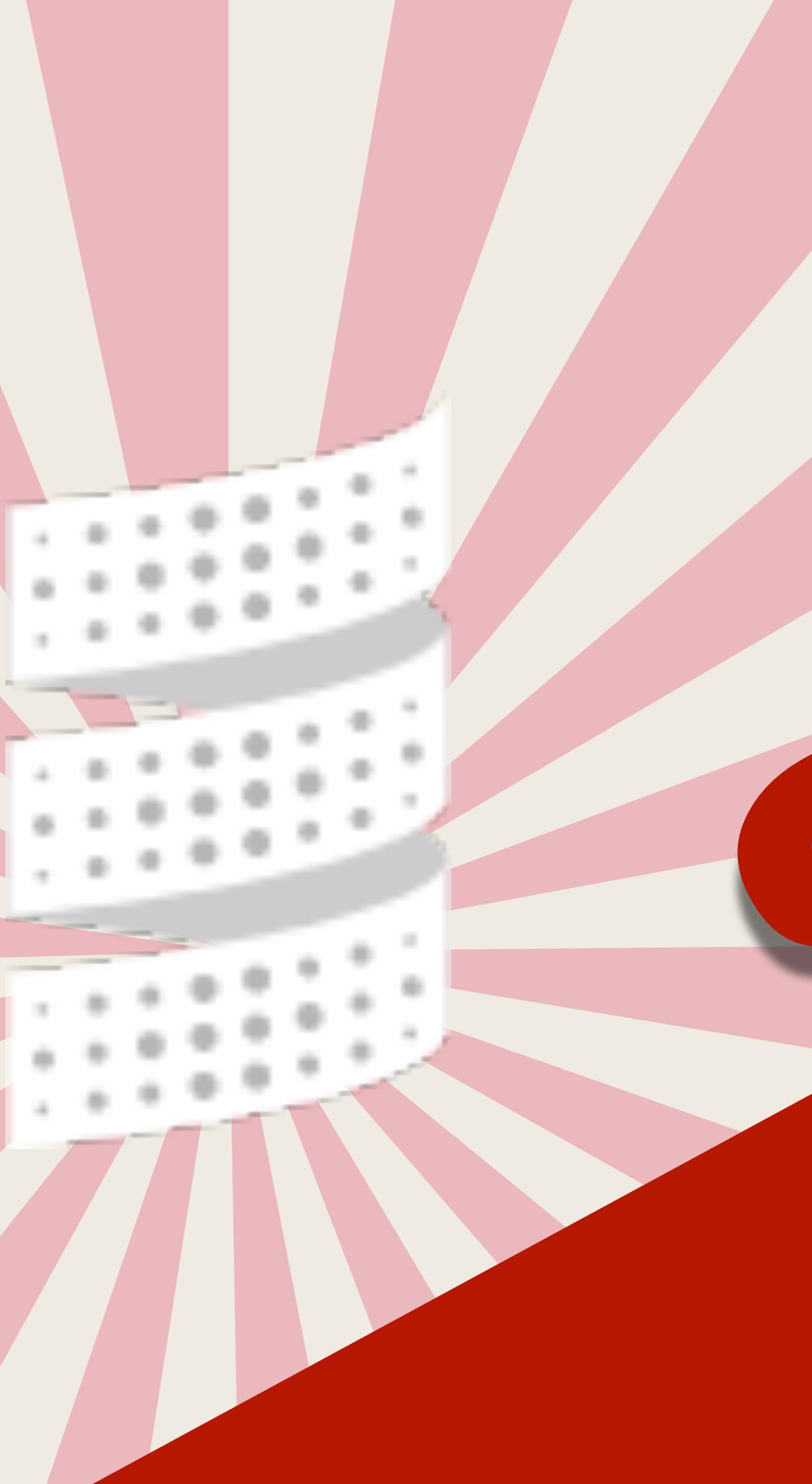
## Enumerations



- Enumerations in Scala 2 originally *were* kind of terrible and ignored
- Redesigned with simplicity in mind
- *union types* can model an enumeration
- Compatible with Java Enumerations

Demo:

`src/main/scala/com.xyzcorp.demo.enums.*`  
`src/main/java/com.xyzcorp.demo.enums.*`



scalaj<sup>3</sup>

Algebraic Data Types



- ▶ Enumerations and their ease of use makes it the best choice now for Algebraic Data Types
- ▶ Algebraic Data Types is a type associated operations but whose representation is hidden
- ▶ Also great for Generalized Algebraic Data Types to represent expressions

Demo:

`src/main/scala/com.xyzcorp.algebraicdatatype.*`

# Lab: Creating Domains

**Step 0:** If you are using scastie, be sure to use Scala 3

**Step 1:** In the project and in the `src/main/scala/exercises/objects/1_Shapes.scala` file in the project or using scastie, create the domain object Shape as an enum, with values for Triangle, Rectangle, and Circle. Triangle should accept a base and height as Double. Rectangle should accept a width and height, both as a Double. Circle should accept a radius as a Double

**Step 2:** Create a method named `area`. `area` should use a match to calculate the area of Rectangle, Triangle, and Circle. For a math refresher:

$$\text{Area (triangle)} = \frac{1}{2} \times b \times h$$

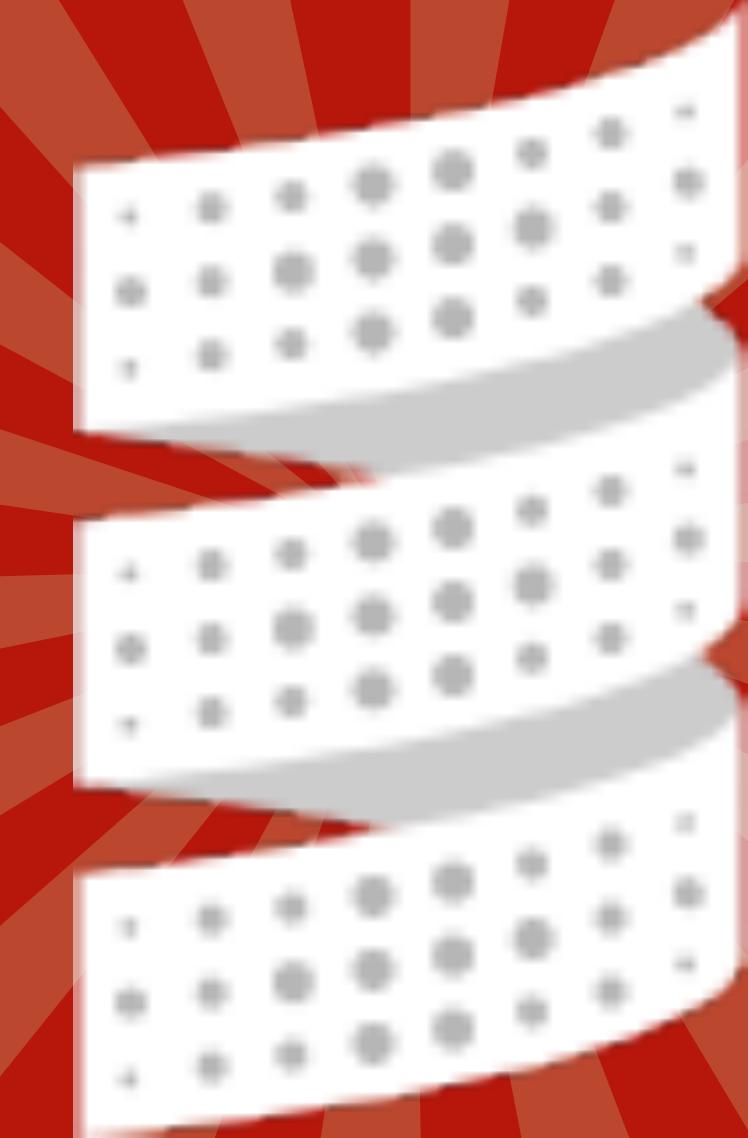
$$\text{Area (rectangle)} = w \times h$$

$$\text{Area (circle)} = \pi \times r^2$$

**Step 3:** Create a main method that calls `area` with samples and print the output

Bonus: Try to use Spaces instead of Braces

# The New Implicits

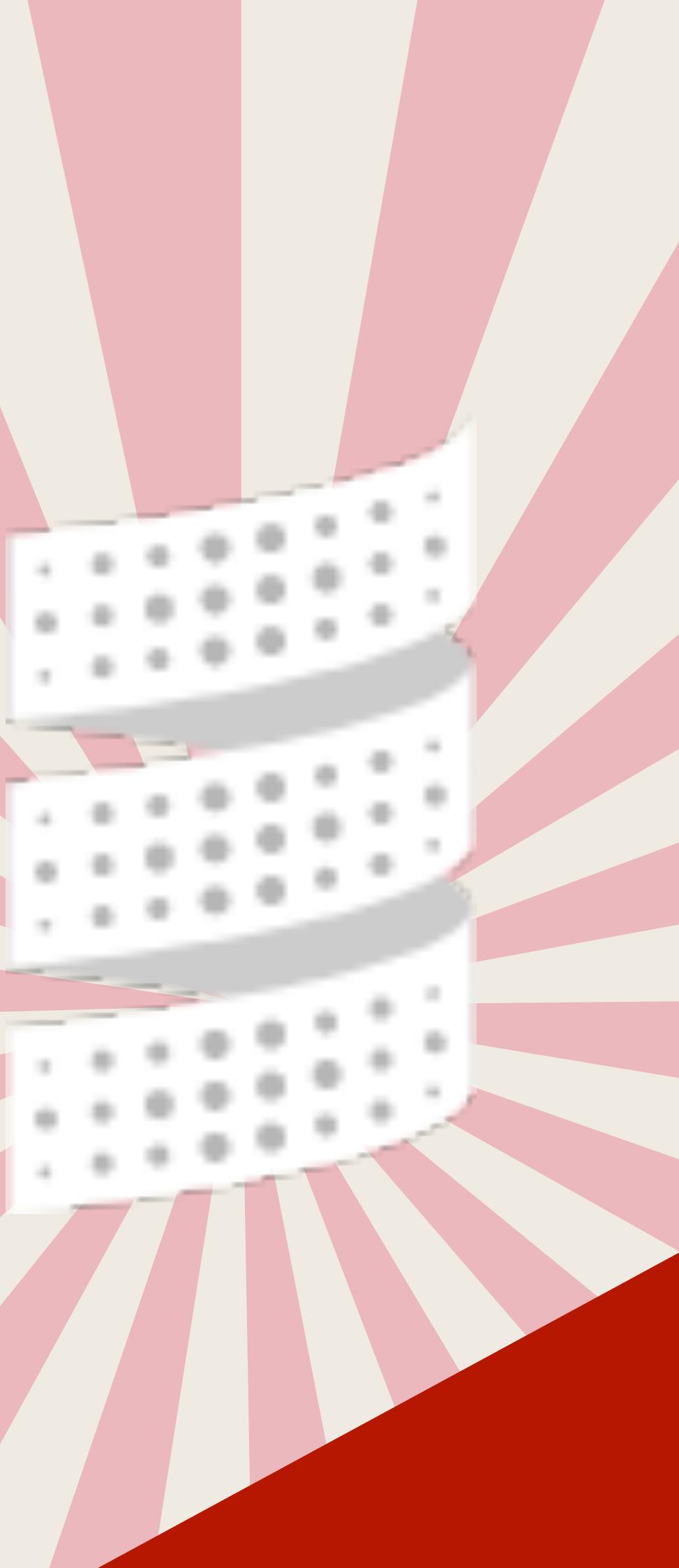




- **implicit** is a defining feature in Scala
- Binds a type to an object that can be used within a scope
- When a type is required it will retrieve that object, this has the following advantages and disadvantages
  - **Advantage** - Lessens boilerplate and cruft making your code lean
  - **Disadvantage** - Higher Learning Curve; Can be tough to chase down definition

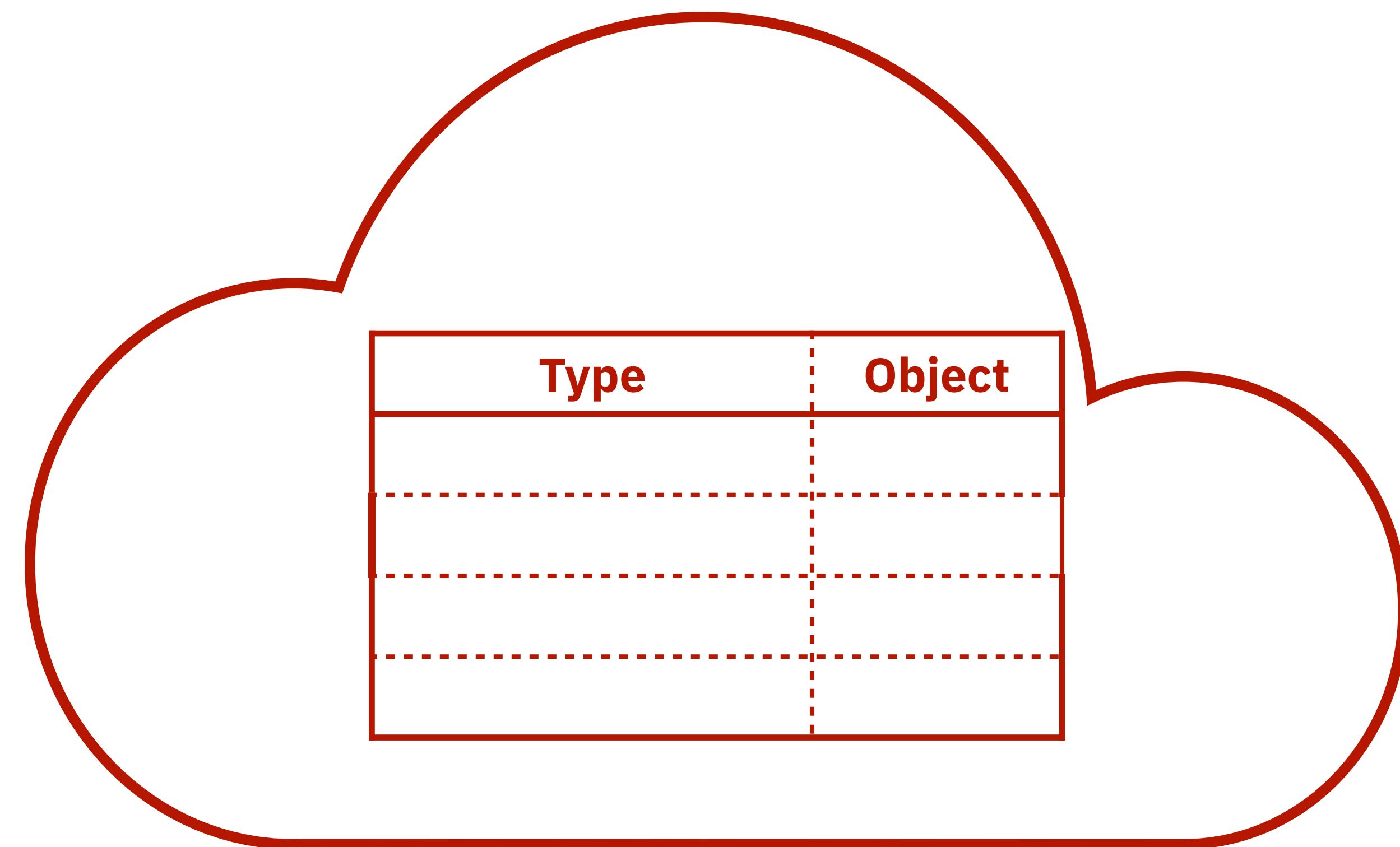


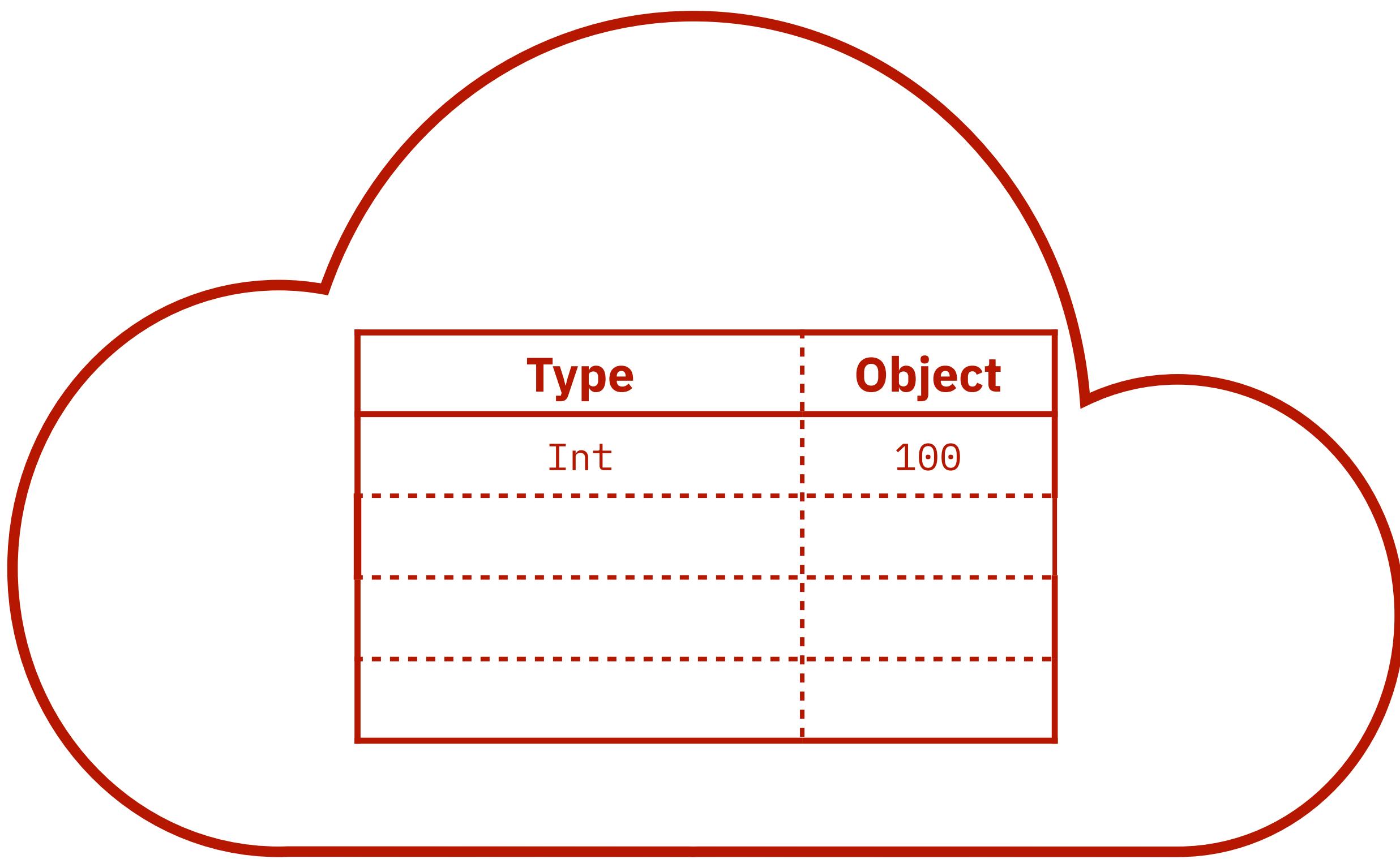
- ▶ It is the means by which Scala programmers:
- ▶ Create typeclasses
- ▶ Create context values
- ▶ Extends functionality
- ▶ Perform Dependency Injection



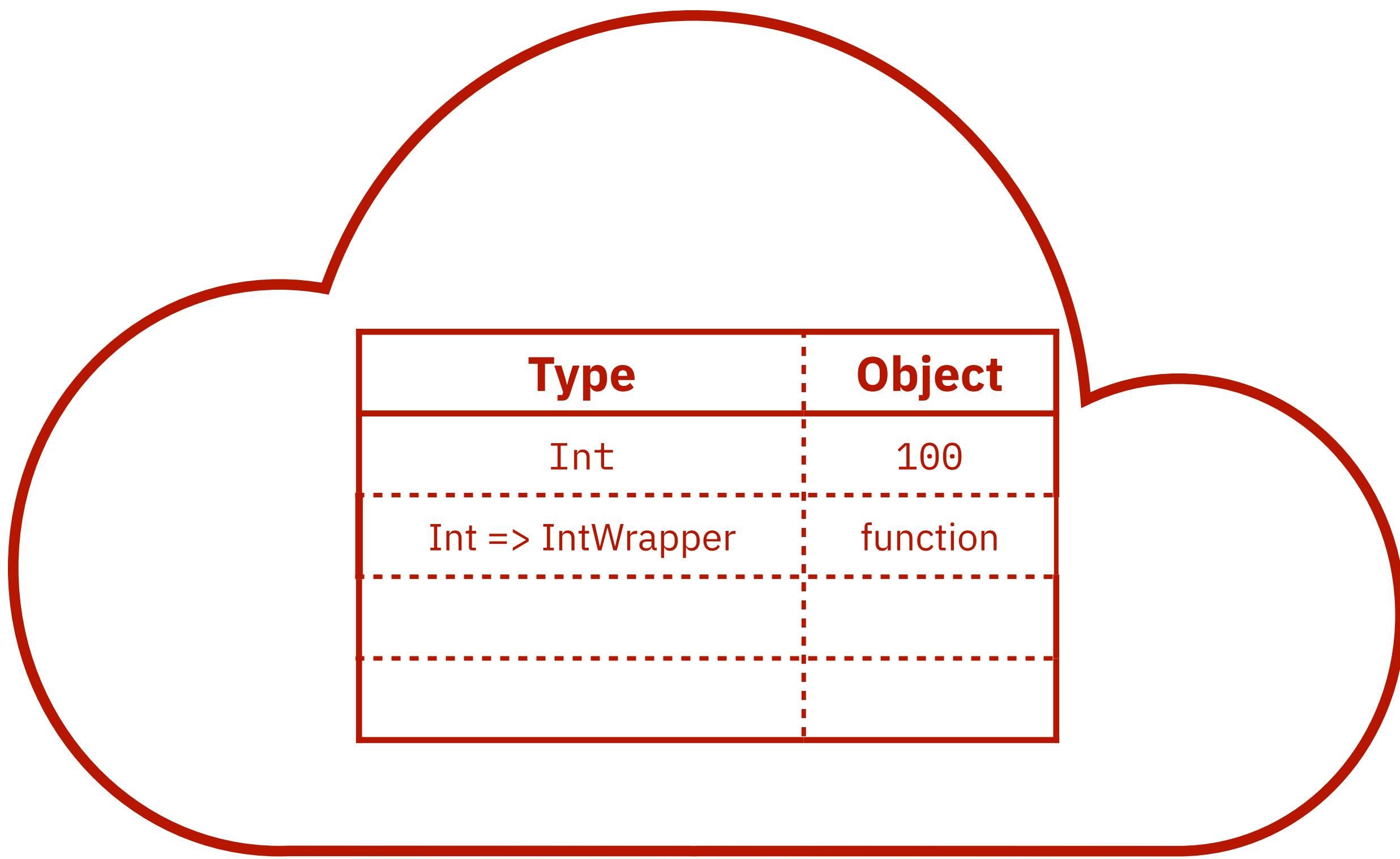
# scala<sup>3</sup>

## Implicits in Scala 2

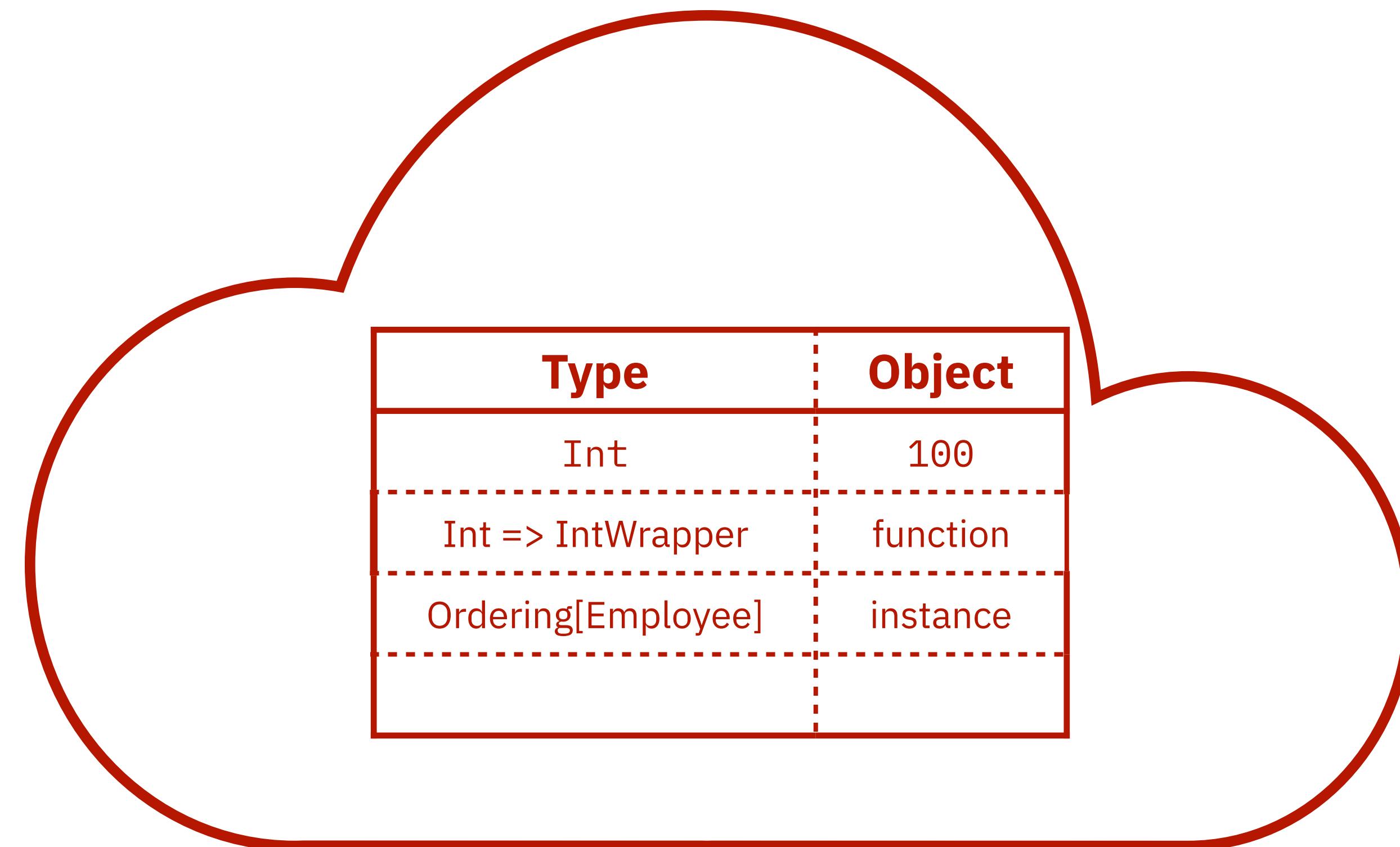




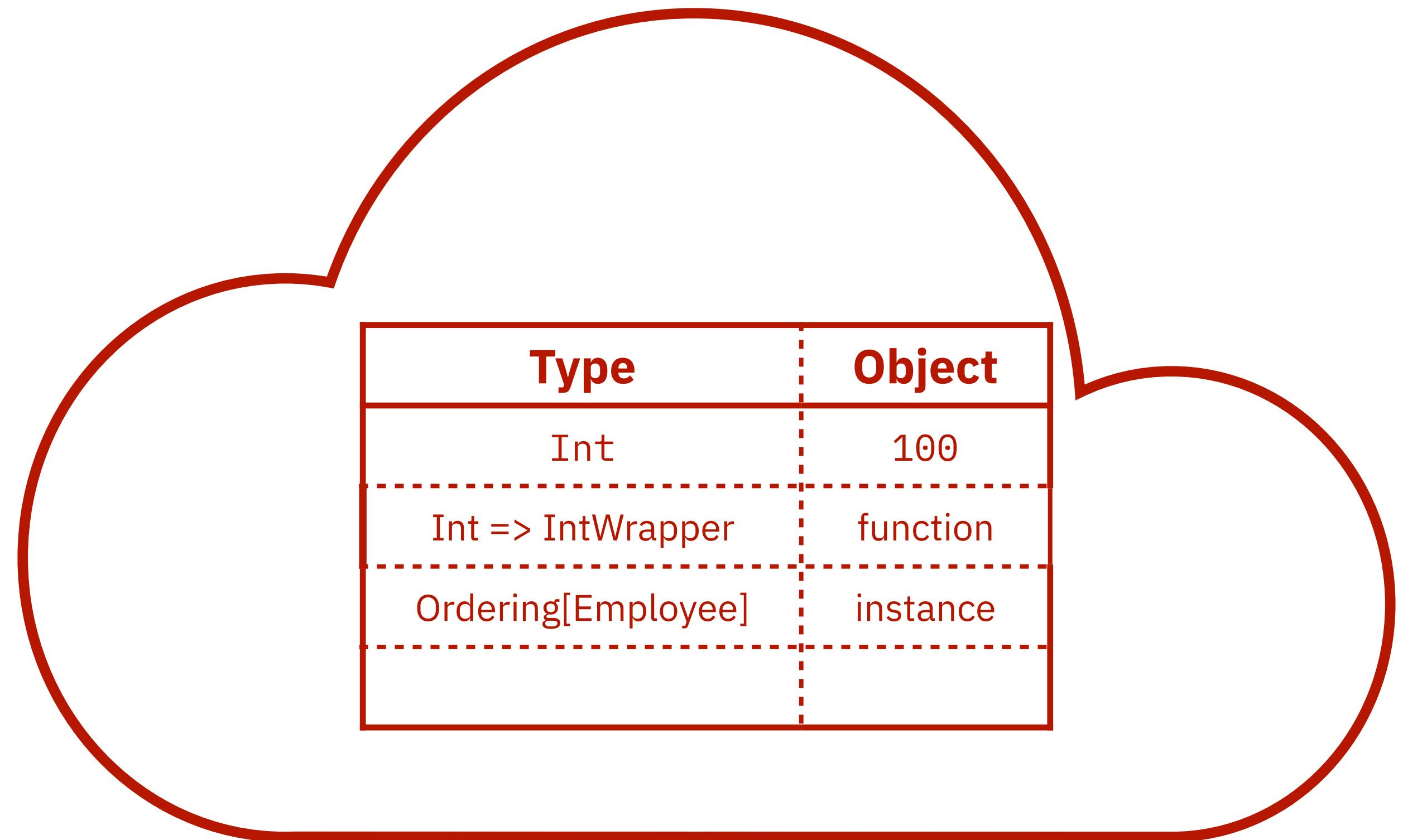
```
implicit val x:Int = 100
```



```
implicit val x = i => new IntWrapper(i)
```

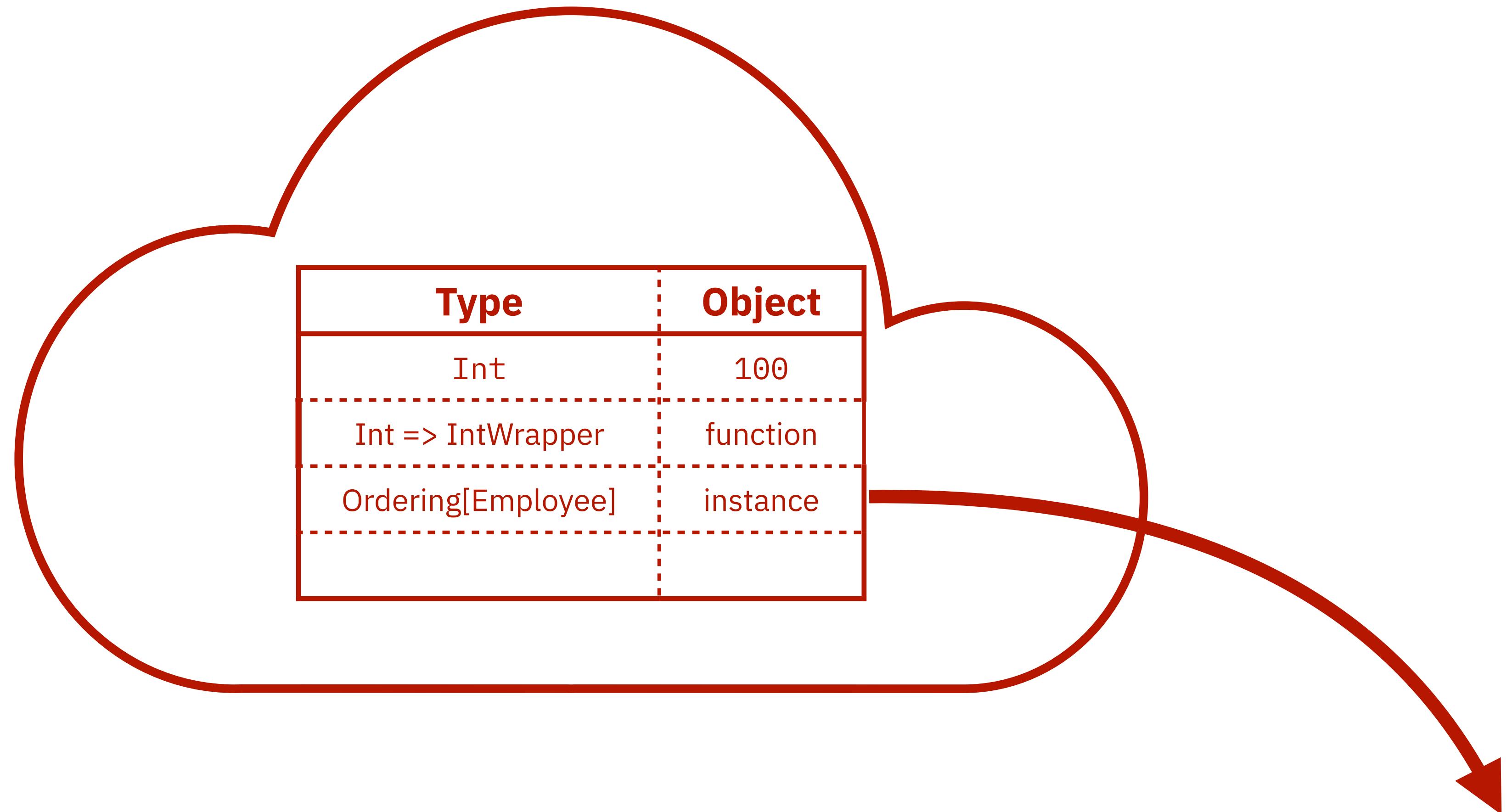


```
implicit ord: Ordering[Employee] = new Ordering[Employee]{...}
```

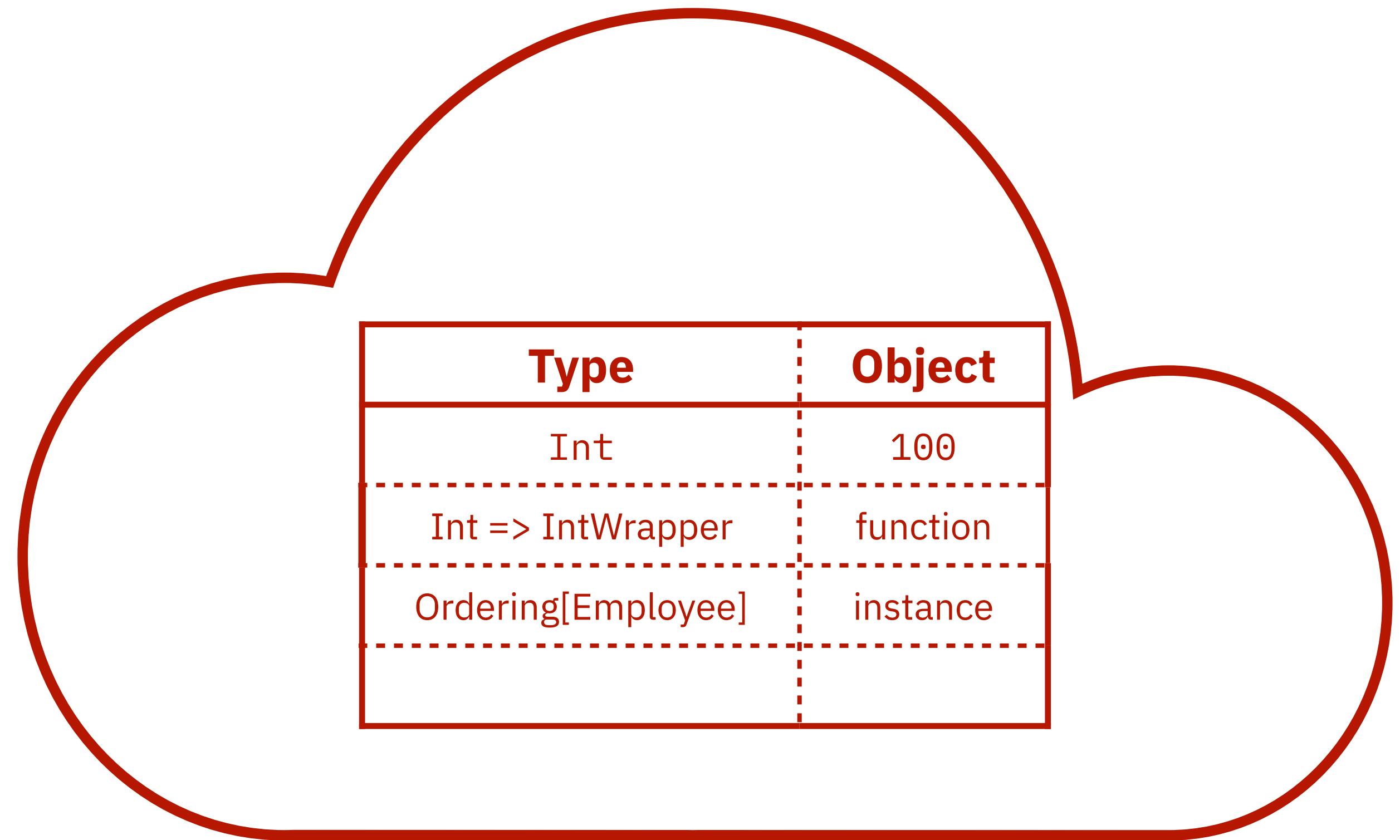


?

```
def mySortingMethod(list>List[Employee])(implicit o:Ordering[Employee])
```

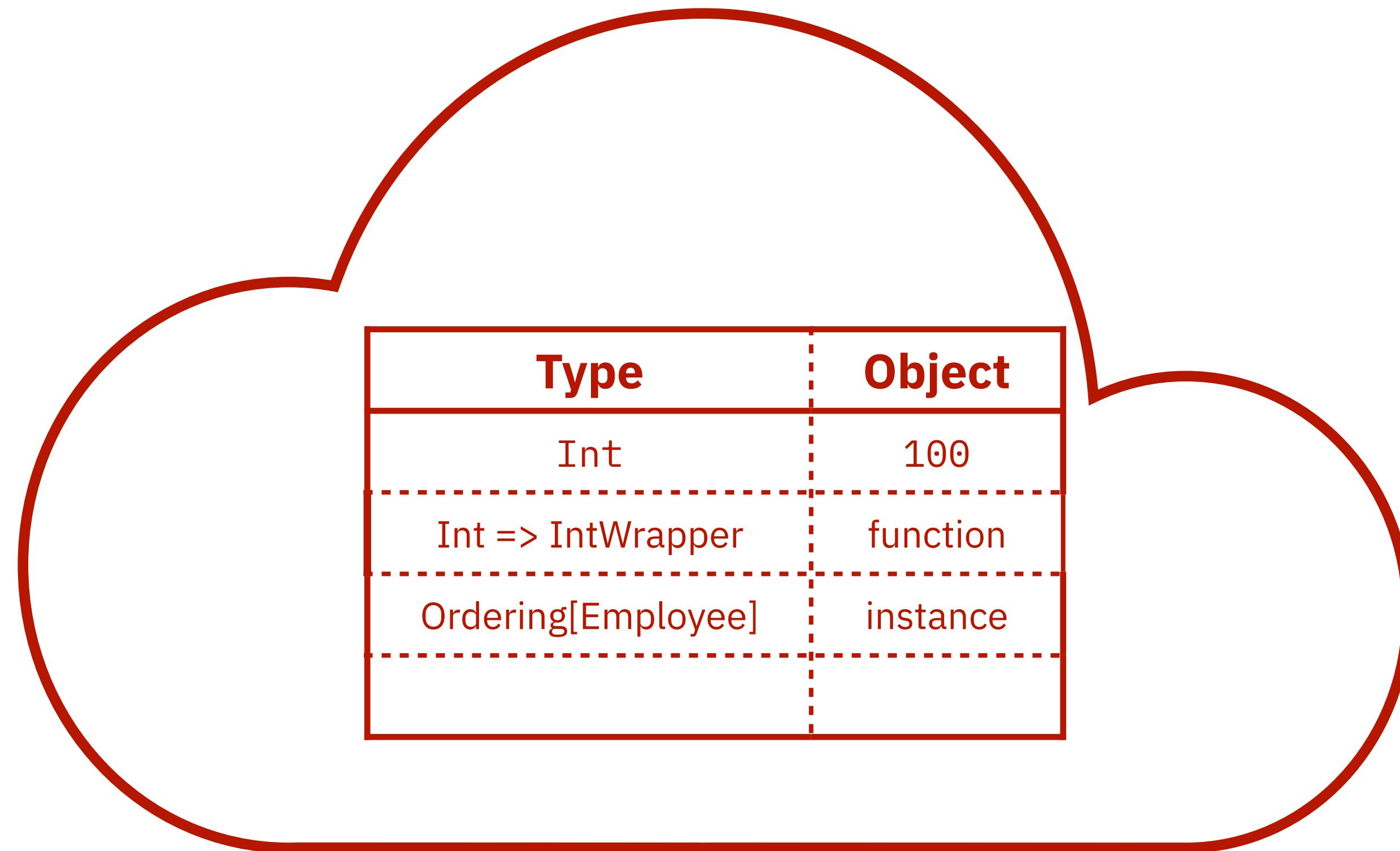


```
def mySortingMethod(list>List[Employee])(implicit o:Ordering[Employee])
```



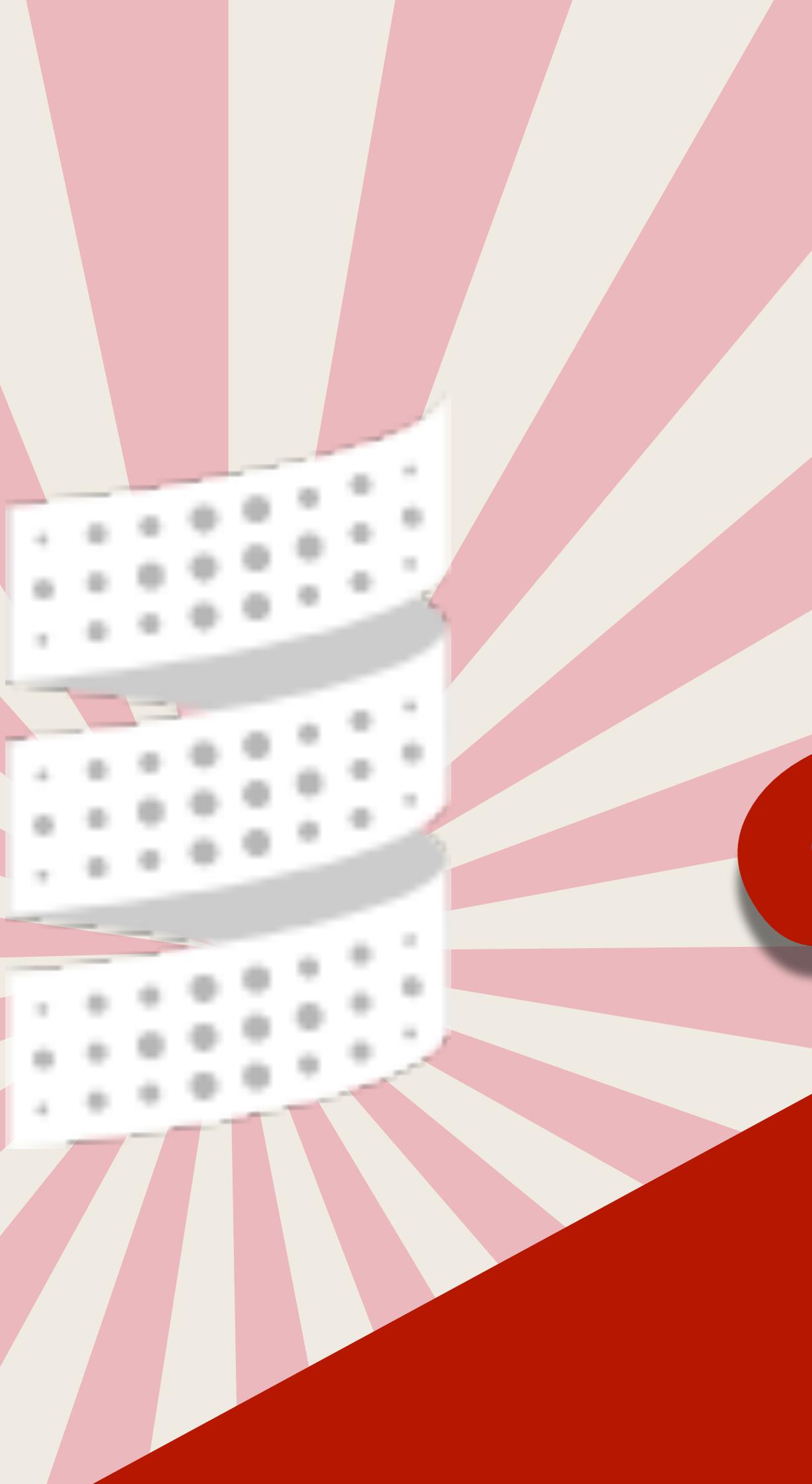
?

```
def mySortingMethod(list>List[Employee])(implicit o:Ordering[Department])
```



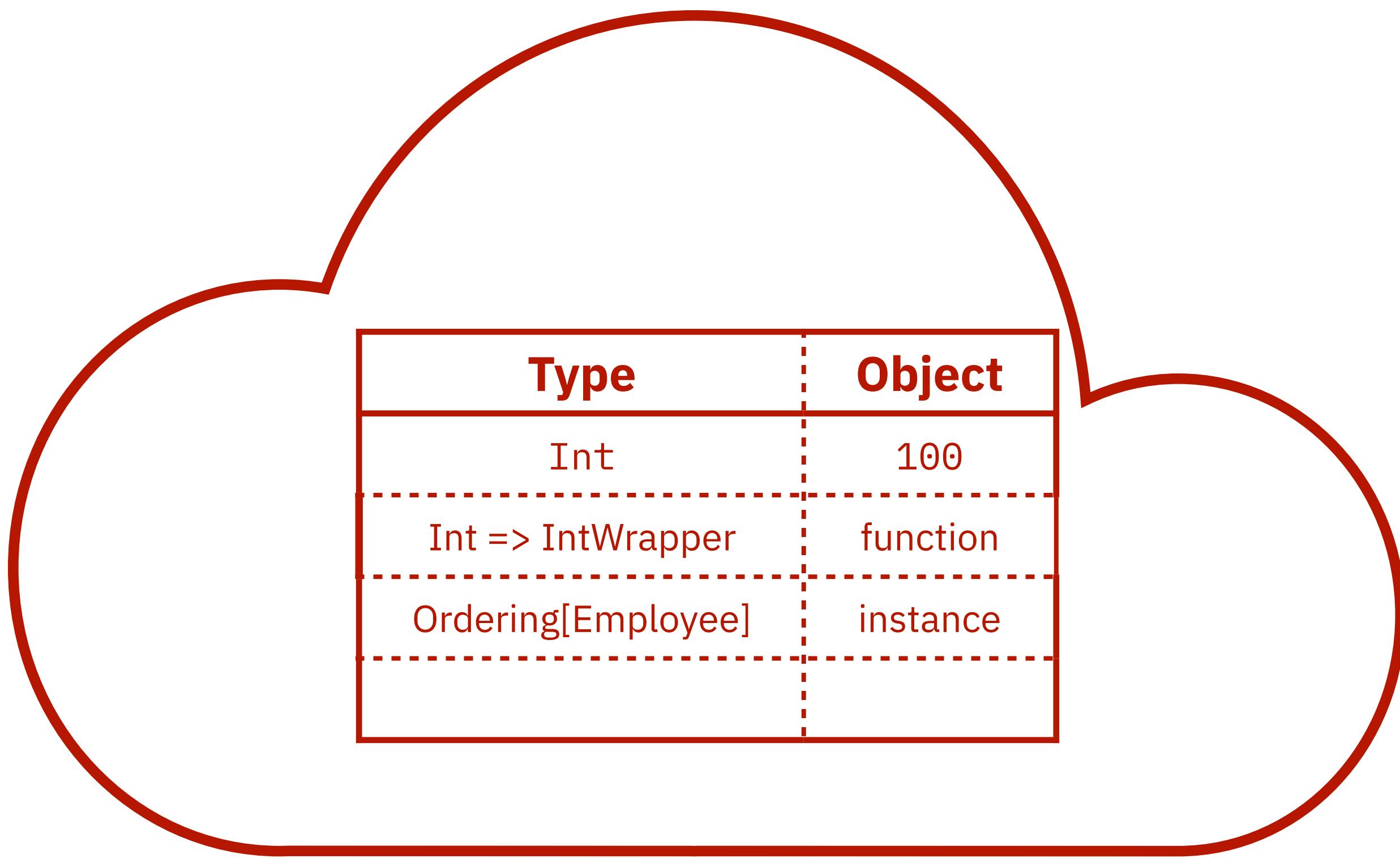
```
def mySortingMethod(list>List[Employee])(implicit o:Ordering[Department])
```

*No implicit ordering found for implicit*

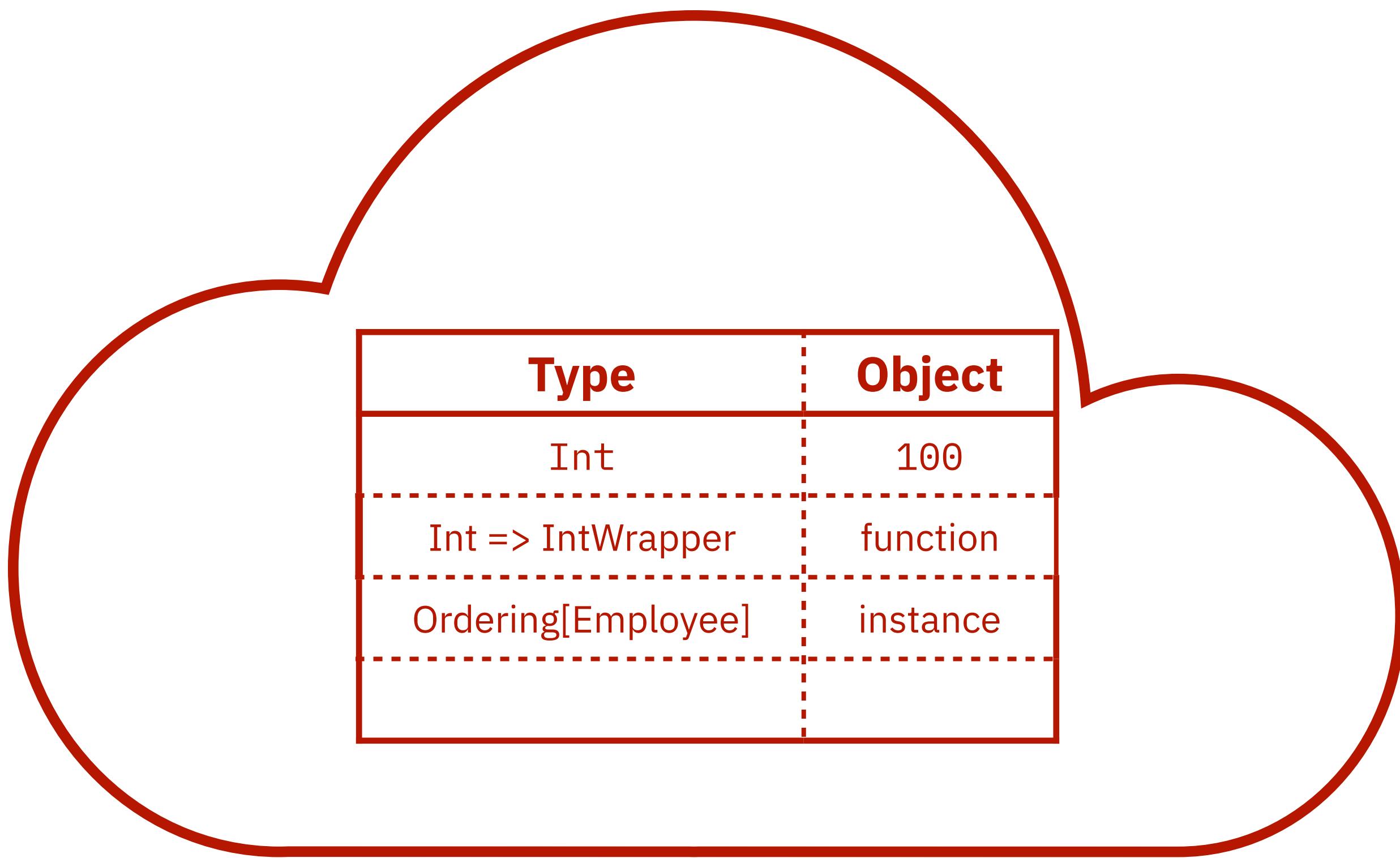


# Scala 3

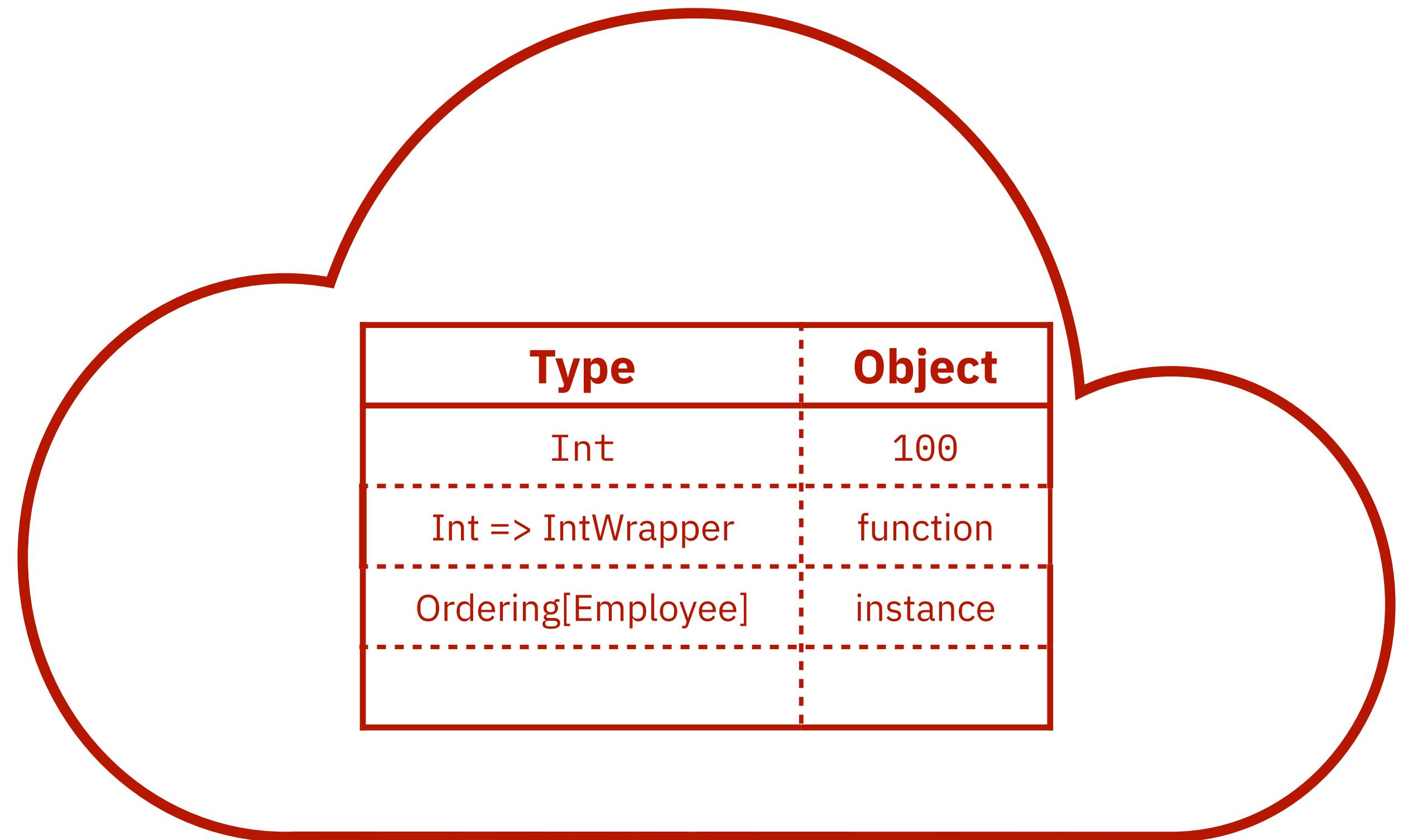
## Given and Using The New "Implicits"



```
given Ordering[Employee] = new Ordering[Employee] {  
    def compare (x:Employee,y:Employee)  
}
```

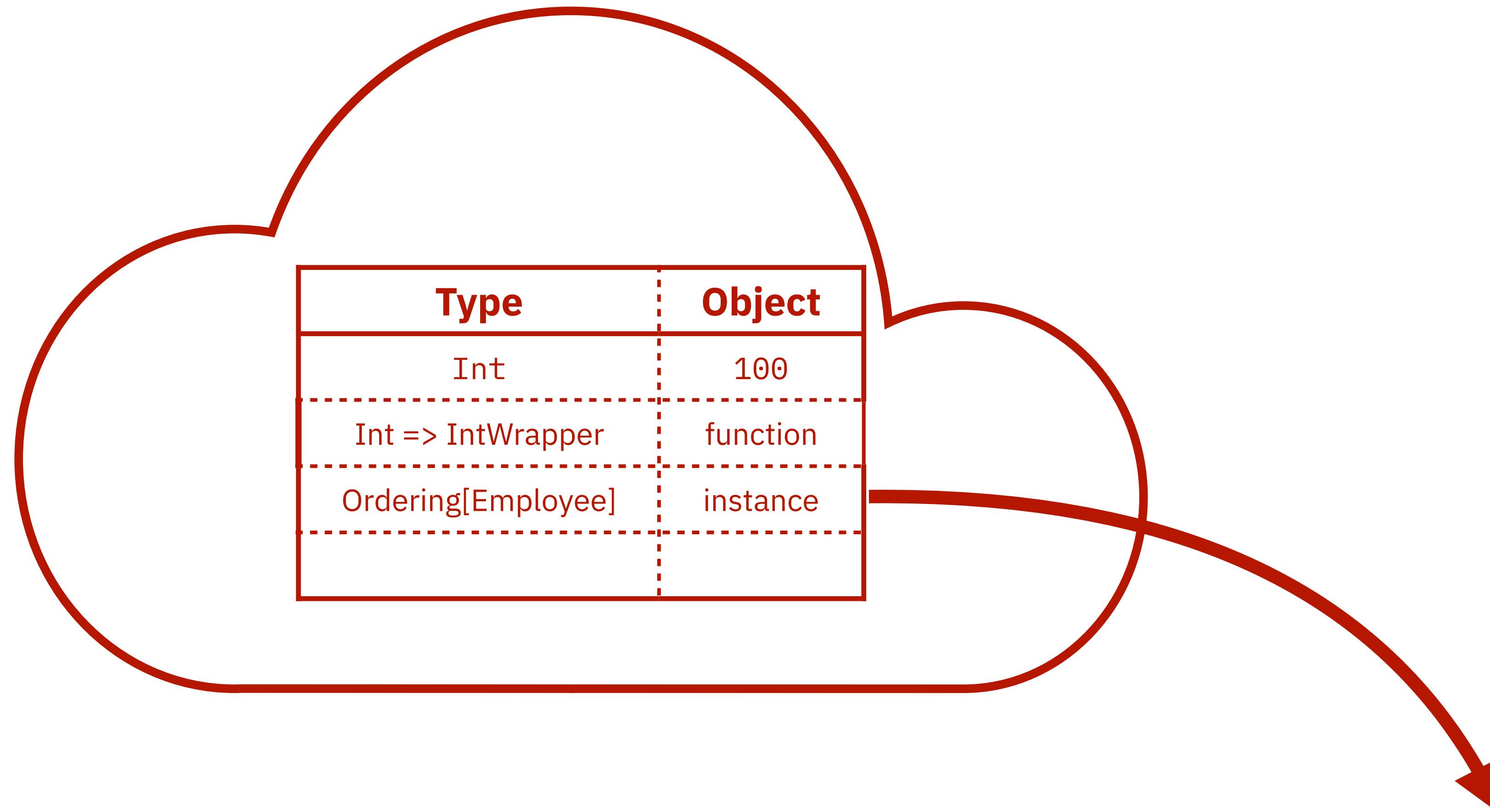


```
given Ordering[Employee] with {
    def compare(x:Employee,y:Employee)
}
```



?

```
def mySortingMethod(list>List[Employee])(using o:Ordering[Employee])
```



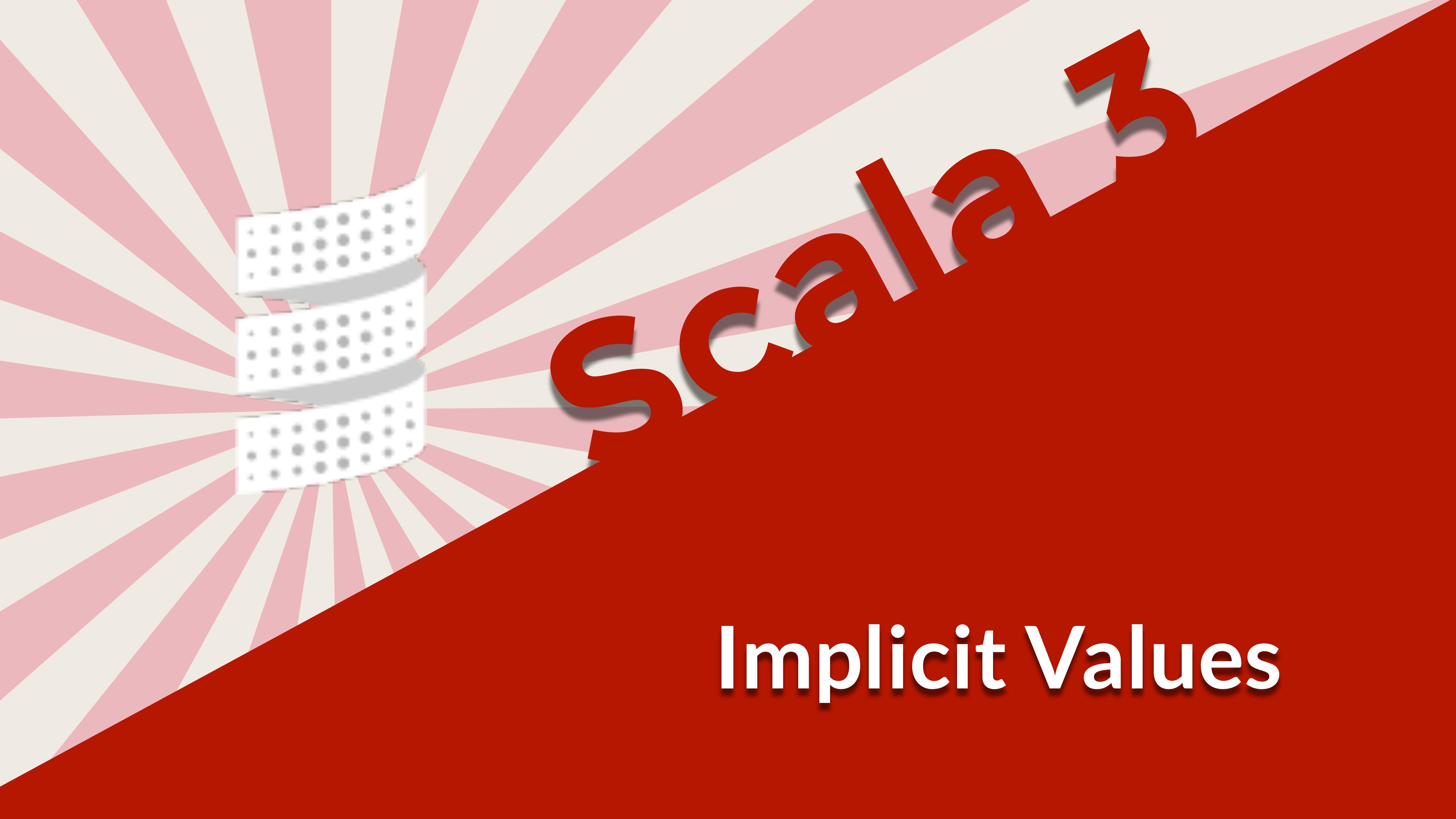
```
def mySortingMethod(list>List[Employee])(using o:Ordering[Employee])
```



- What's the difference?

- implicits are an overused term for multiple purpose
- Not intuitive except for seasoned developers.
- Conflicts with in many code instances; for example, below requires an apply method

```
def currentMap(implicit ctx: Context): Map[String, Int]
```



# scalaz

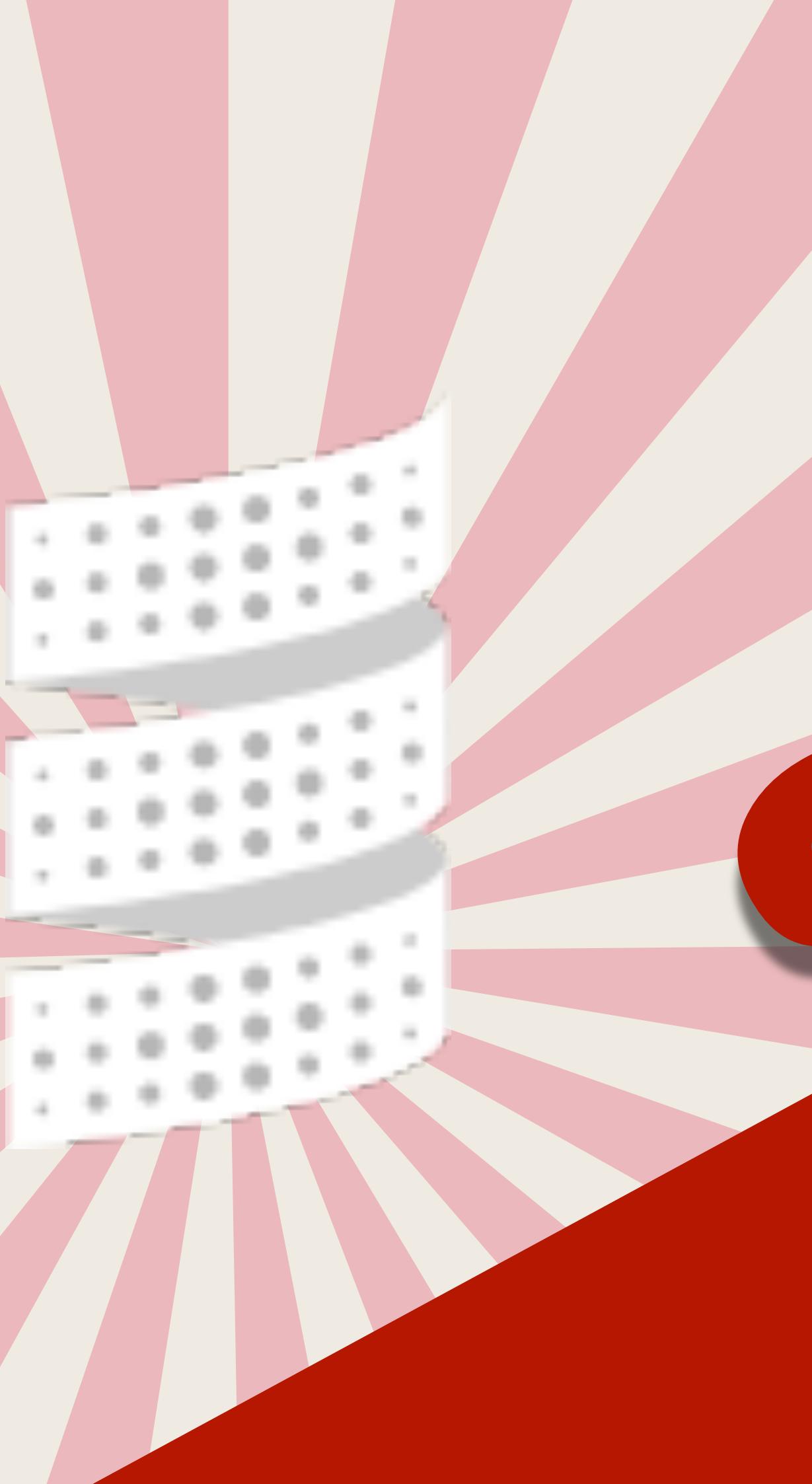
## Implicit Values



- ▶ An implicit value is bound per scope
- ▶ It is available when required
- ▶ Can always be overridden with your own implementation
- ▶ The binding is after the given keyword

Demo:

`src/main/scala/com.xyzcorp.givenusing.*`



# scala<sup>3</sup>

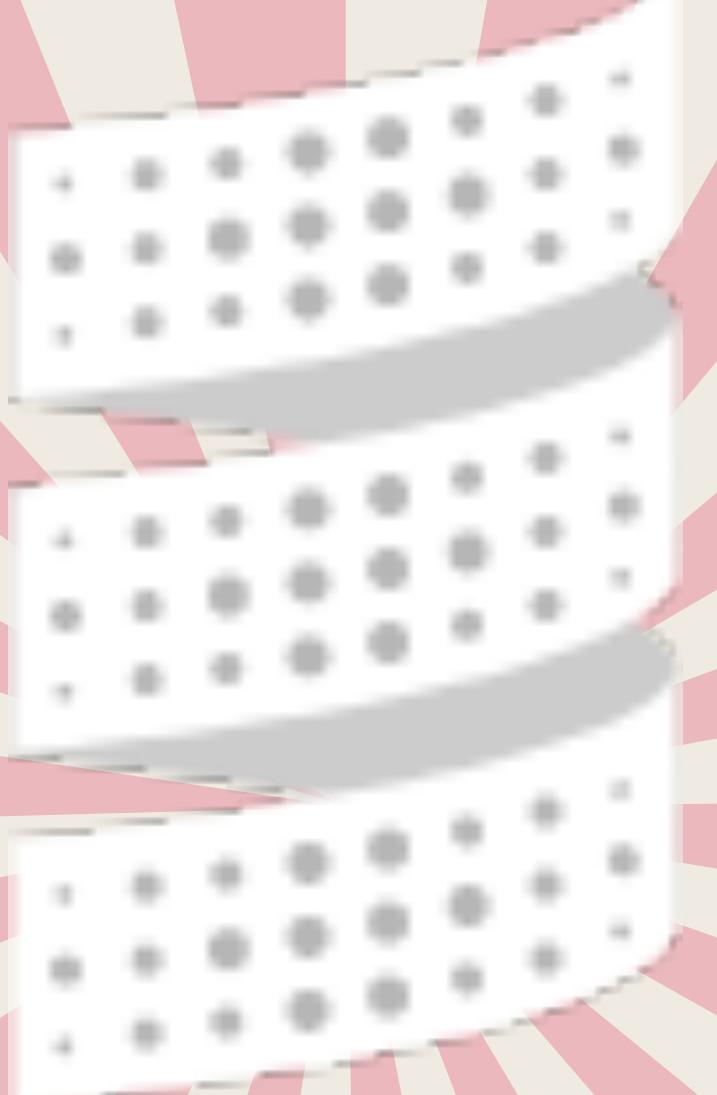
## Extension Methods



- Can we add methods to a type that already exist?
- Can we add `isOdd` and `isEven` to `Integer`
- Many languages have this mechanism, under different terms
- Scala has used it with `implicits`, but is now under a different identity

Demo:

`src/main/scala/com.xyzcorp.extensionmethods.*`



# scala<sup>3</sup>

## Conversions



- ▶ Previously in Scala 2 conversions can be performed by either implicitly defining:
  - ▶ A function of the conversion
  - ▶ A method of the conversion
- ▶ Scala 3 Dotty used a type `Conversion` that uses an `apply` much like `Function1`, and are automatically converted to the `Conversion` values.

Demo:

`src/main/java/com.xyzcorp.conversions.*`



# scala<sup>3</sup>

## Type Classes



- ▶ TypeClasses are implicit behaviors used in functional programming languages
- ▶ Think of a behavior that would like to work for varying types. (e.g. folding, mapping, flatMapping, producing Strings, determining equality, effects, side-effects, etc.)
- ▶ Ad-hoc polymorphism - *polymorphism in which polymorphic functions can be applied to arguments of different types*

Demo:

`src/main/scala/com.xyzcorp.typeclasses.*`

# Lab: Creating a Typeclass

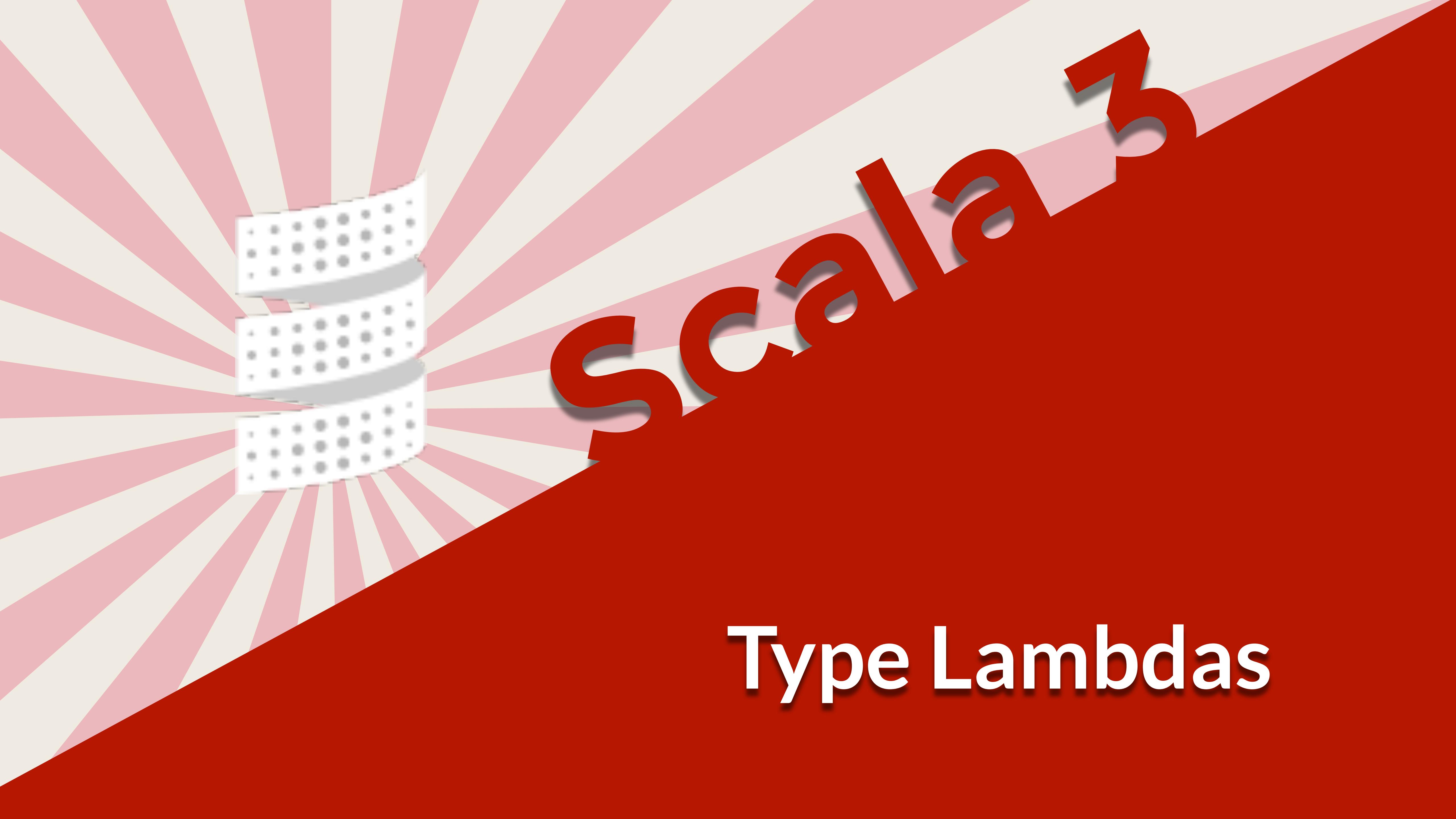
**Step 0:** If you are using scastie, be sure to use Scala 3

**Step 1:** Go to `src/main/scala/exercises/typeclasses/1_Samples` in the project or using scastie, create a trait `Sample[A]` with a method called `sample` that returns a generic `A`

**Step 2:** Create an object called `Samples` that uses a `scala.util.Random` to create a sample `Int`, `Double`, `String`, `Long`, and for a bonus `List[Int]`, `List[Long]`

**Step 3:** Create a method in `Samples` called `drawSample[A]` (`using:Sample[A]`) = `s.sample`

**Step 4:** Create a main method that can draw various samples. For example, `println(Samples.drawSample[List[String]])`



# scala<sup>3</sup>

## Type Lambdas



- ▶ Defines a function of types to types
- ▶ They may carry bounds and variances
- ▶ Allows us to compose to a higher kinded type
- ▶ Avoids  $\{ \text{type } \lambda[\alpha] = \text{List}[\text{Option}[\alpha]] \} \# \lambda$
- ▶ Translated with TypeLambdas:  $[A] \Rightarrow \text{List}[\text{Option}[A]]$



- Higher Kinded Types?
  - What if the List in List[A] can be generic? T[A] or M[A]?
  - That's a higher kinded type and they are useful, particularly in functional programming

Demo:

`src/main/scala/com.xyzcorp.higherkindedtypes.*`

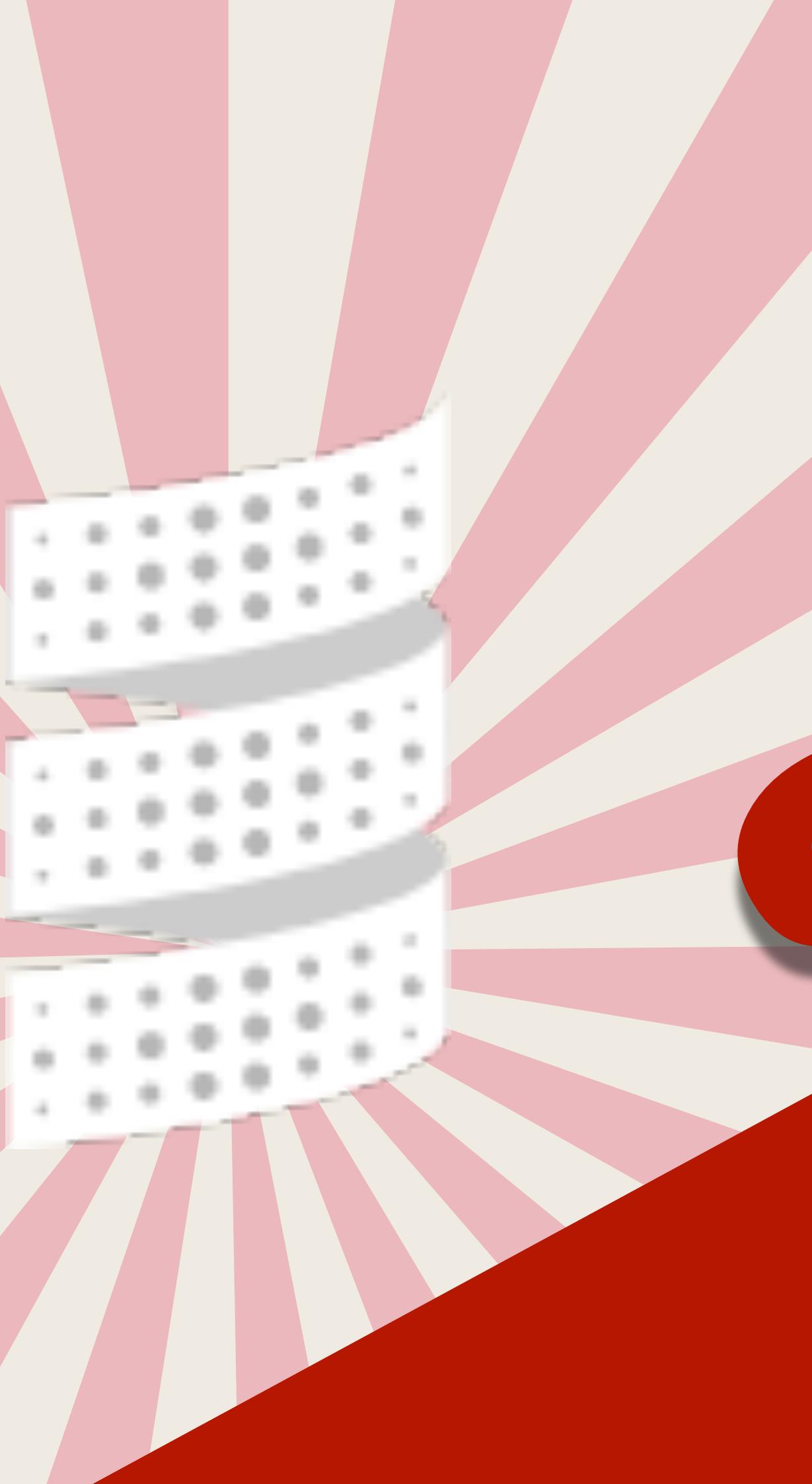
# Lab: Creating a Monad

**Step 0:** If you are using scastie, be sure to use Scala 3

**Step 1:** In `src/main/scala/exercises` in the project or using scastie, open `monads .1_Monads`, there you will see the definition for `Monad` , and `MyBox`.

**Step 2:** In the object `MyBox` create a "given" `Monad` definition for `MyBox`

**Step 3:** Create a main method that will test that it works. For example `flatMap(MyBox(123))(a => MyBox(a + 20))` should be equal to a `MyBox(143)`



# scala<sup>3</sup>

"I still don't get this  
Type-Fu"



- ▶ It's intellectually satisfying stuff
- ▶ It's safe
- ▶ Referentially Transparent
- ▶ But it is not easy



- ▶ Draw some inspiration, in some projects
- ▶ Typelevel Cats
- ▶ Shapeless
- ▶ ScalaZ
- ▶ ZIO

# Other Cool Features





# scalia<sup>3</sup>

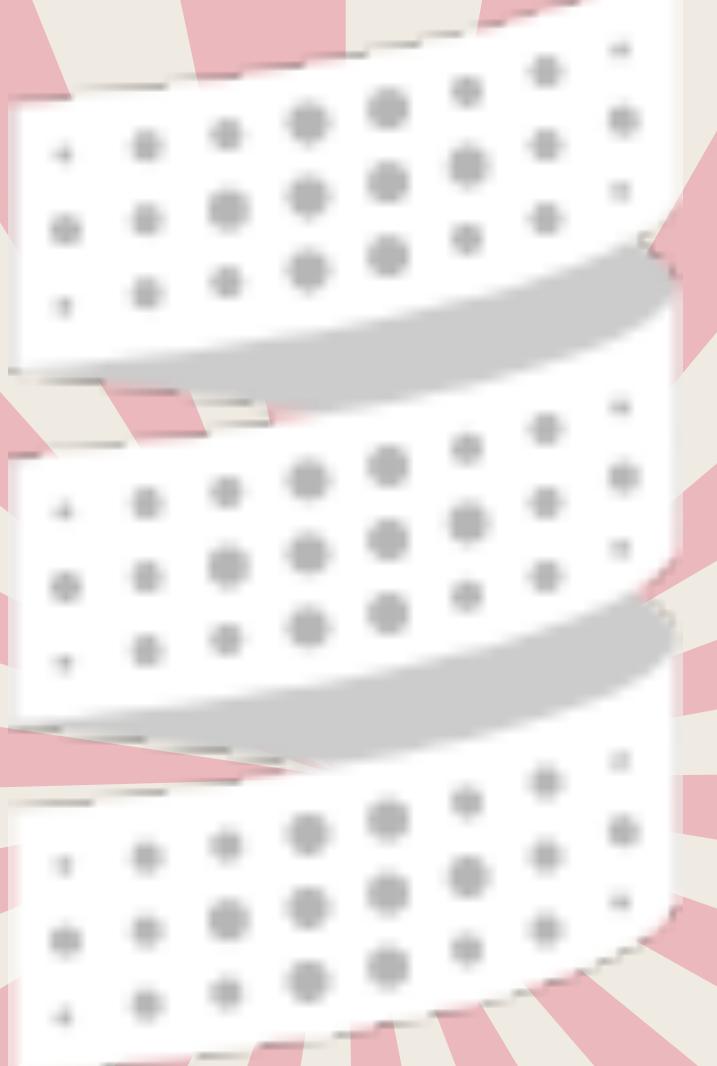
## Trait Parameters



- ▶ Traits are analogous to interfaces in Scala
- ▶ Like interfaces they do not have state or constructors
- ▶ With Dotty, they can, so we can declare a variable and that will be mixed in with a type.
- ▶ Arguments are evaluated immediately
- ▶ Strict rules apply as how inheritance of these traits will work

Demo:

`src/main/scala/com.xyzcorp.traitparameters.*`



# scala<sup>3</sup>

## Match Types



- ▶ Important to know that the type system in Scala has it's own language.
- ▶ A match type operates almost like Pattern Matching.
- ▶ The distinction is rather than extract values; we will be extracting types.

Demo:

`src/main/scala/com.xyzcorp.matchtypes.*`



# scalable<sup>↗</sup>

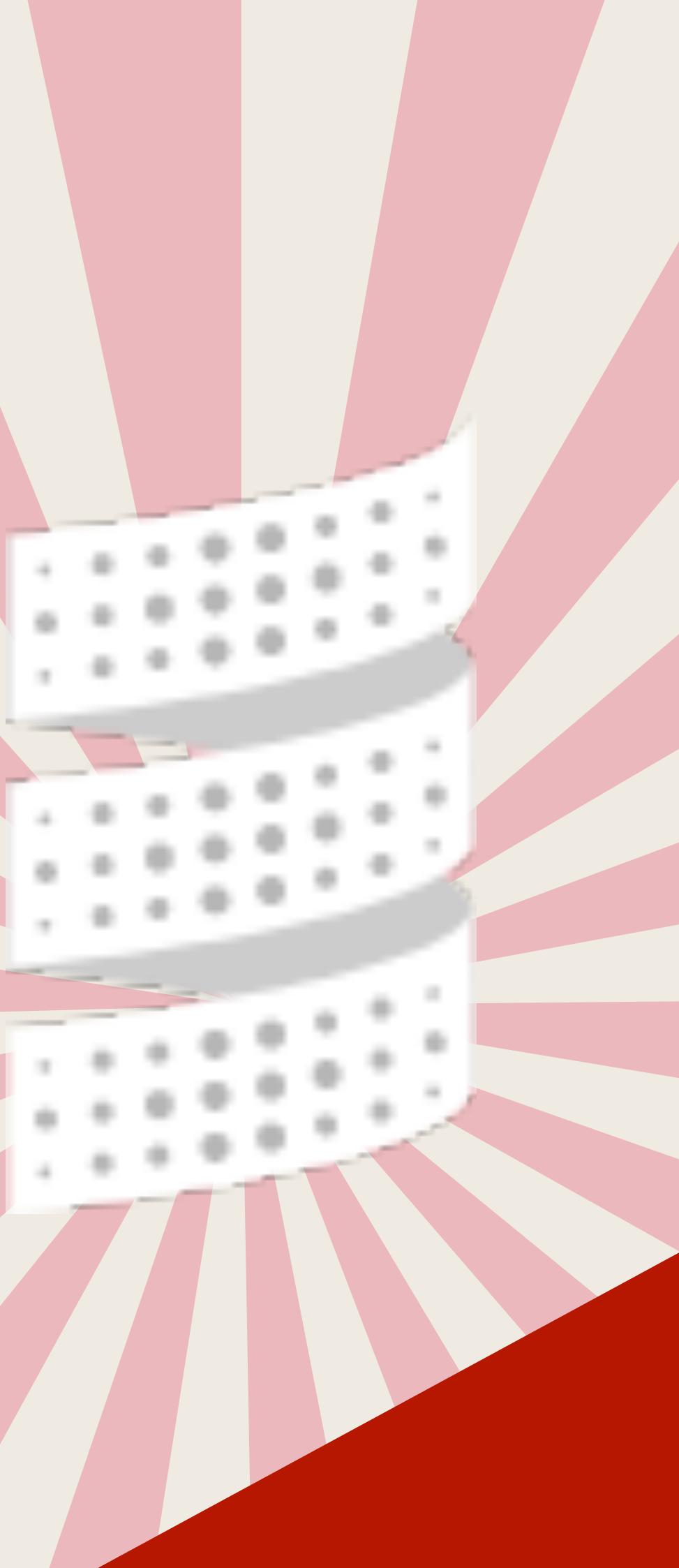
Multiversal Equality



- `==` and `!=` uses Java's `equals()` in objects
- `equals` is not type safe
- Multiversal Equality is an opt-in Haskell style way for determining equality
- Based on trait `Eql[-L, -R]`
- Uses type classes to determine the equality

Demo:

`src/main/scala/com.xyzcorp.multiversalequality.*`



# scala<sup>3</sup>

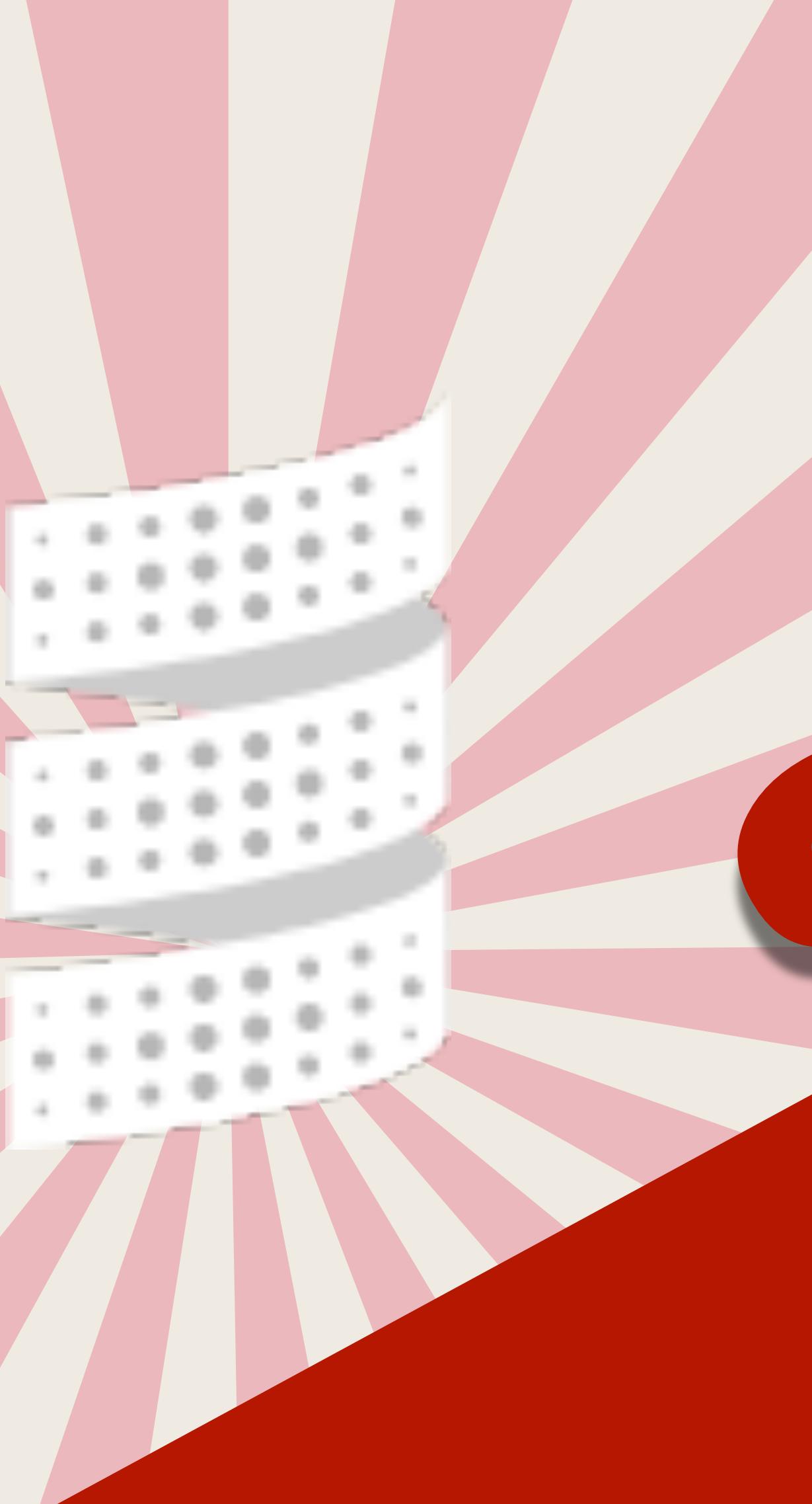
## Opaque Types



- ▶ Provides a completely new type based on the previous one, but...
- ▶ They are not synonymous or an alias
- ▶ Information hiding, we can treat

Demo:

`src/main/scala/com.xyzcorp.opaqueotypes.*`



# scalaz<sup>3</sup>

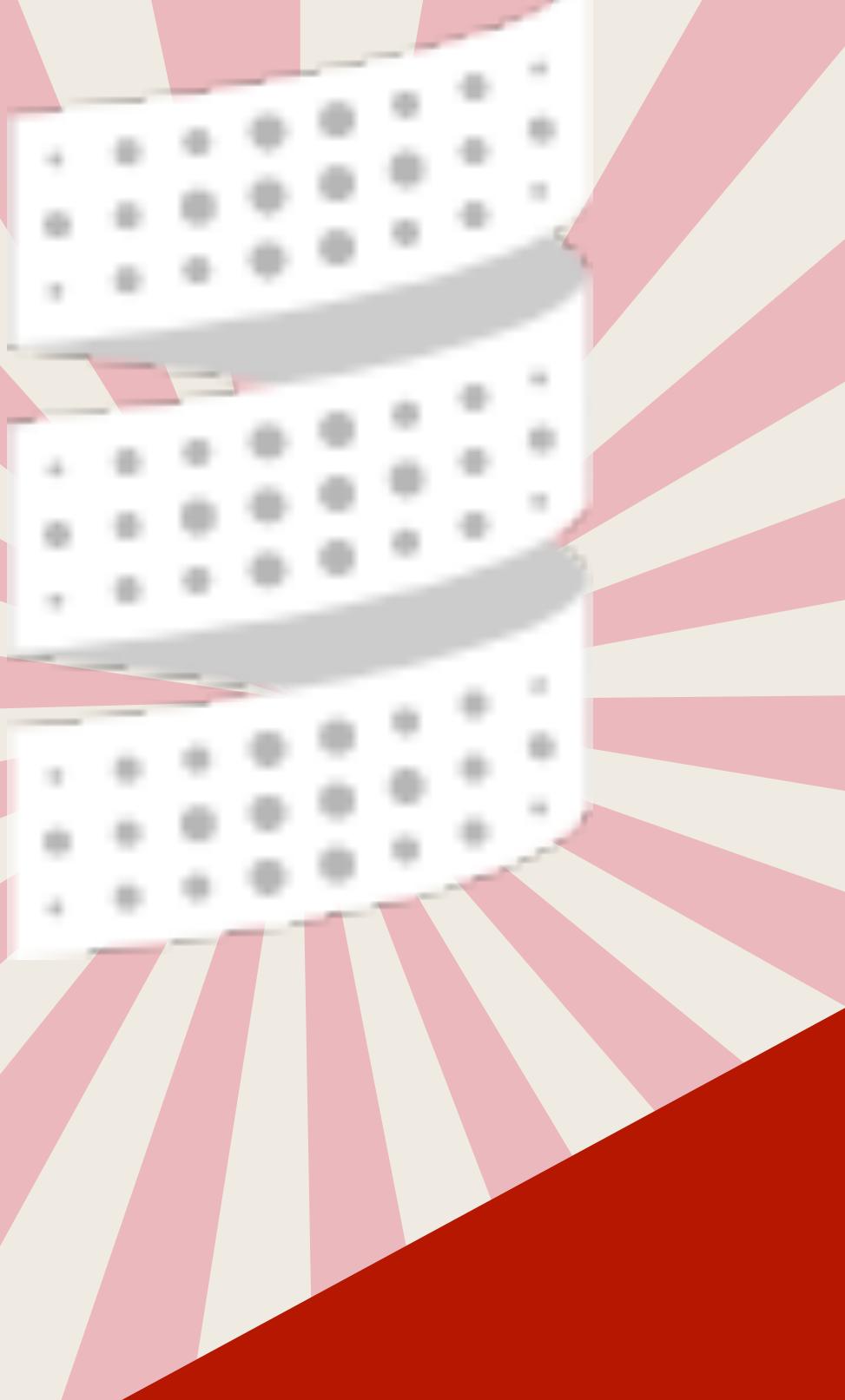
Parameter Untupling



- ▶ One pain point with Scala is the use of a partial function to destructure what is inside of a tuple.
- ▶ Scala 3 all that will disappear so you can express yourself without any extra ceremony

Demo:

`src/main/scala/com.xyzcorp.parameteruntupling.*`



# scala<sup>3</sup>

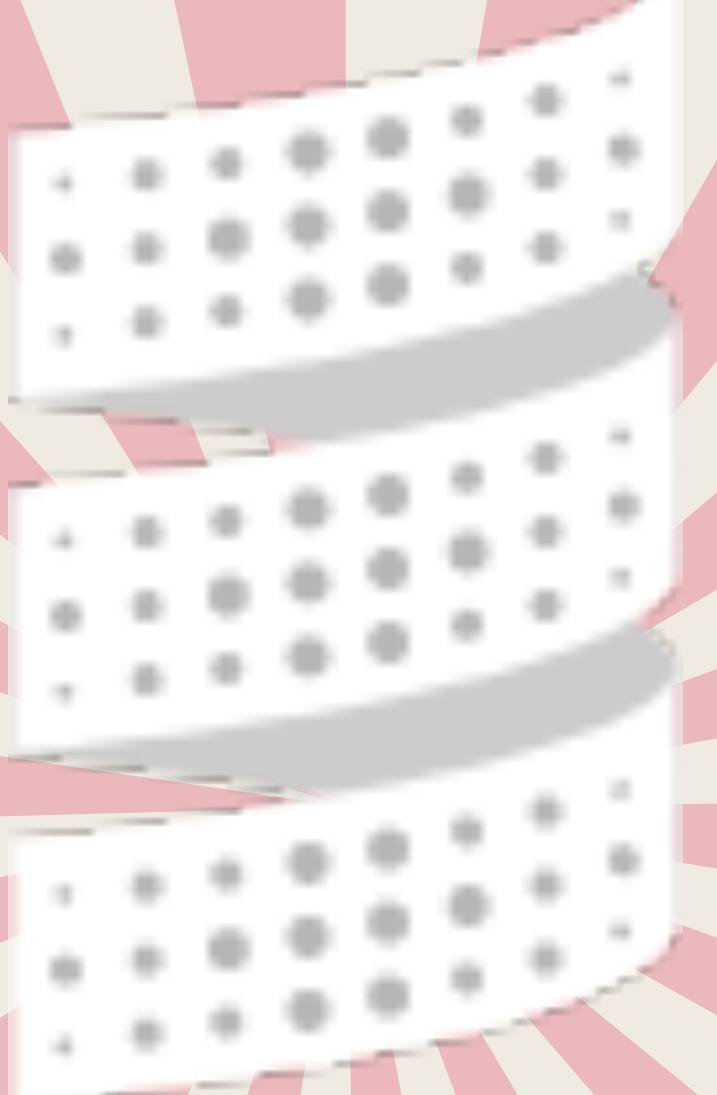
## Tupled Functions



- In Scala 2, Functions were declared using Function1, Function2, Function3, ..., Function22
- In Scala 3, there is no limit due to fancy programming with the new implicits and type handling.
- Important to remember, types are important and if you get the 33rd element of a homogenous tuple the type should be consistent

Demo:

`src/main/scala/com.xyzcorp.tupledfunction.*`



scalaj<sup>3</sup>

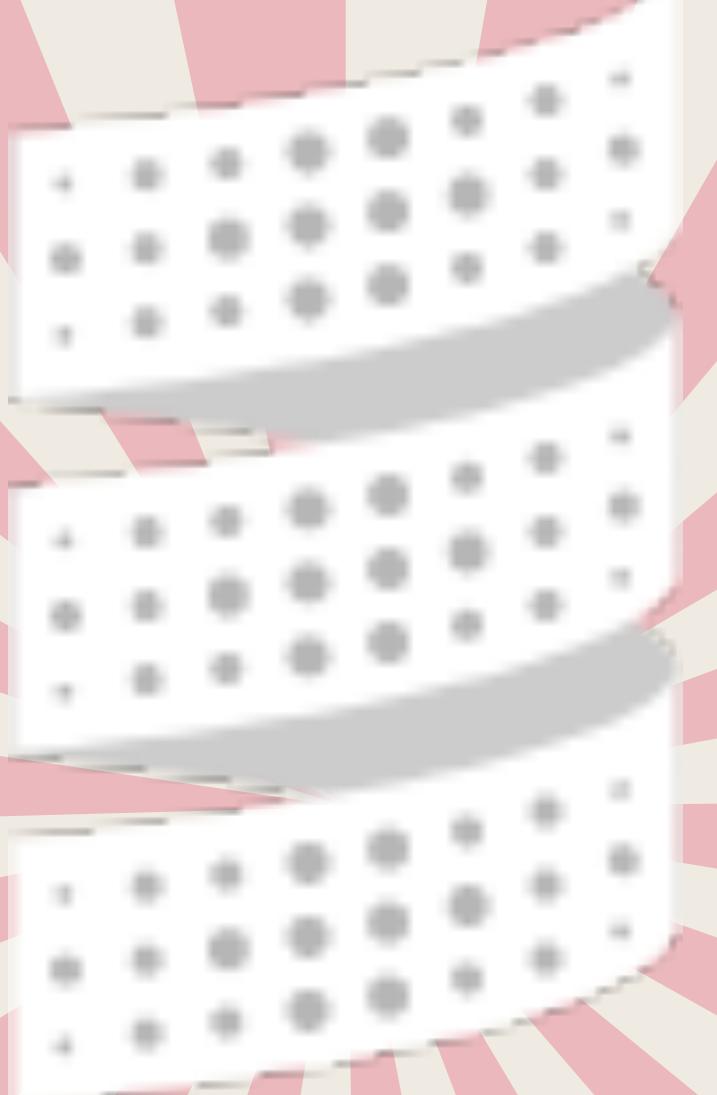
Universal Apply



- Instantiate a class without a new
- Makes the language more unified as to how to instantiate or create objects
- Compliments apply.
- Done internally via the compiler using a technique called *constructor proxies*

Demo:

`src/main/scala/com.xyzcorp.universalapply.*`



# scala<sup>3</sup>

## Structural Types



- ▶ In dynamic typed or optional typed languages, it is great to express oneself with more of a linguistic syntax.
- ▶ `db.findById(..).name`
- ▶ Static typed languages particularly those that have a rigorous typed system becomes harder.



- ▶ This is where the Selectable trait comes in
- ▶ Allows the language to convert `obj.name` into something searchable rather than expecting name to be a method in obj.

Demo:

`src/main/scala/com.xyzcorp.structuraltypes.*`



scala<sup>3</sup>

TargetName



- `@targetName` defines an alternate, usually linguistic, name for a function or method, and more.
- ```
import  
scala.annotation.targetName
```
- Backticked method names are no longer necessary
- Very useful to disambiguate two methods with the same signature usually due to erasure

Demo:

```
src/main/scala/com.xyzcorp.demo.targetname.*
```



# scalaz

Infix



- infix operators or methods can be situated between the object and the parameter
- Include the keyword `infix` to state intent that the method or function is meant to be used in an `infix` fashion.
- Will produce a warning if you use a method as `infix` without declaring it as such
- Not necessary for symbolic operators (e.g `+`)

Demo:

`src/main/scala/com.xyzcorp.infix.*`

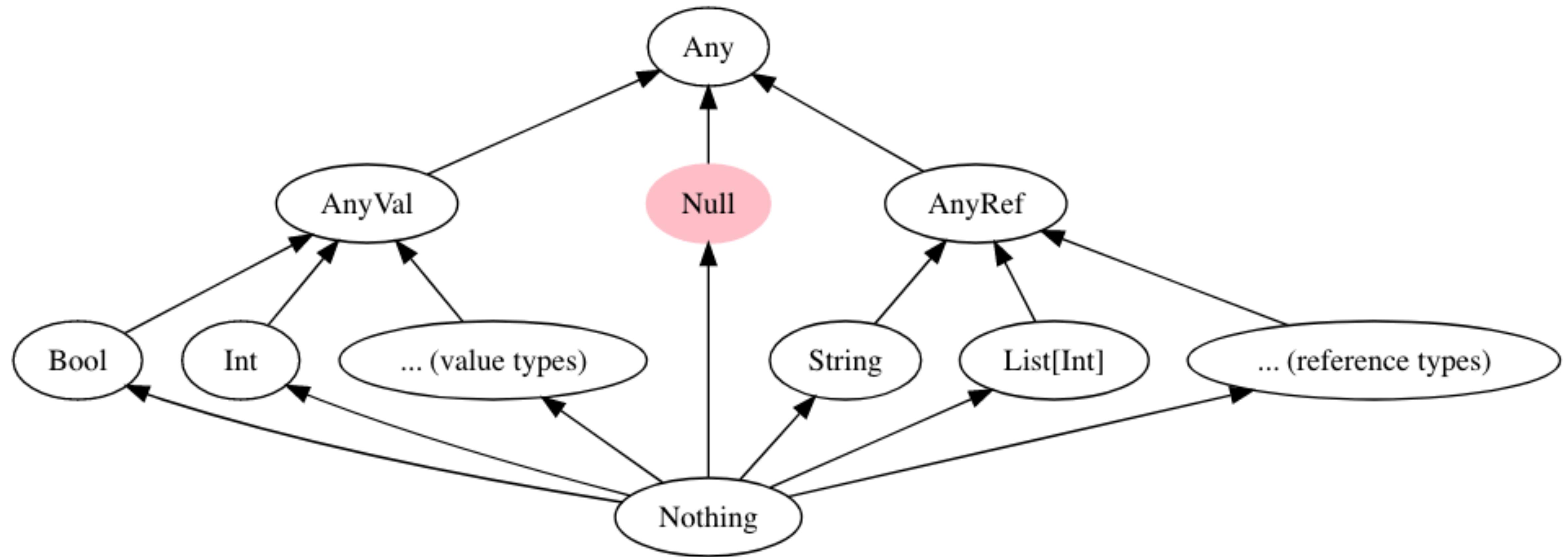


# scala<sup>3</sup>

## Explicit Nulls

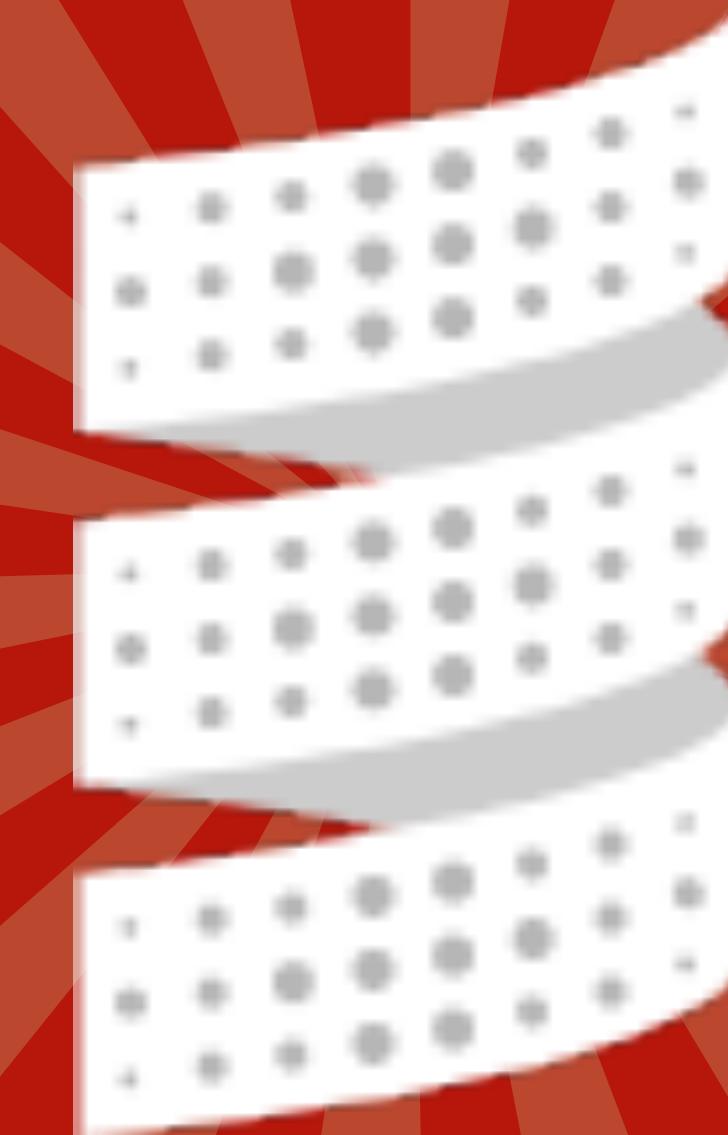


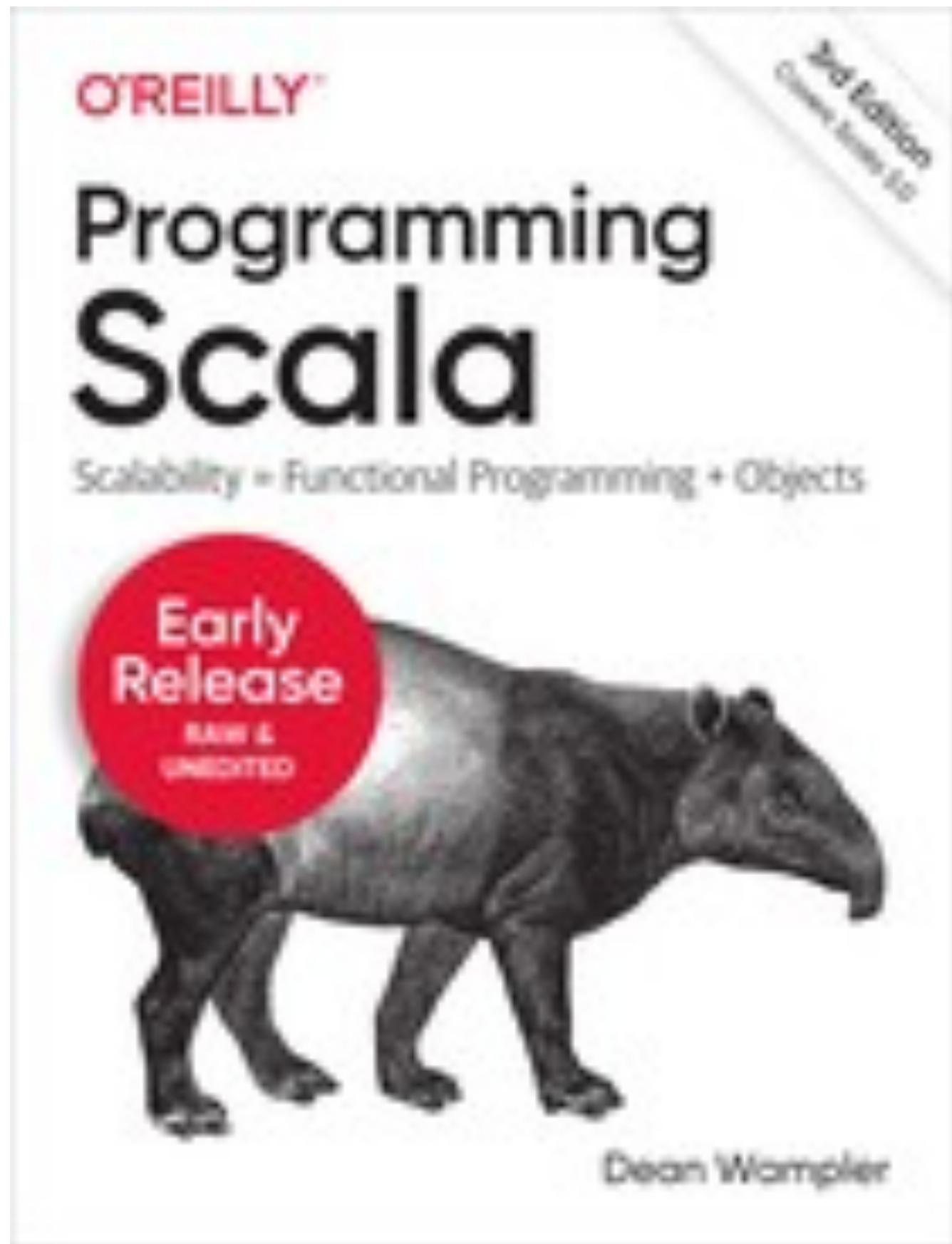
- Union Type of A | Null
  - Type meaning: Either A or Null
  - Can be the type used when calling Java that has the potential to be a null
  - Requires switch -Yexplicit-nulls
  - Aggressive null checking
- The Union Type can be carried forward from a Java API call.



Demo: com.xyzcorp.explicitnulls.\*

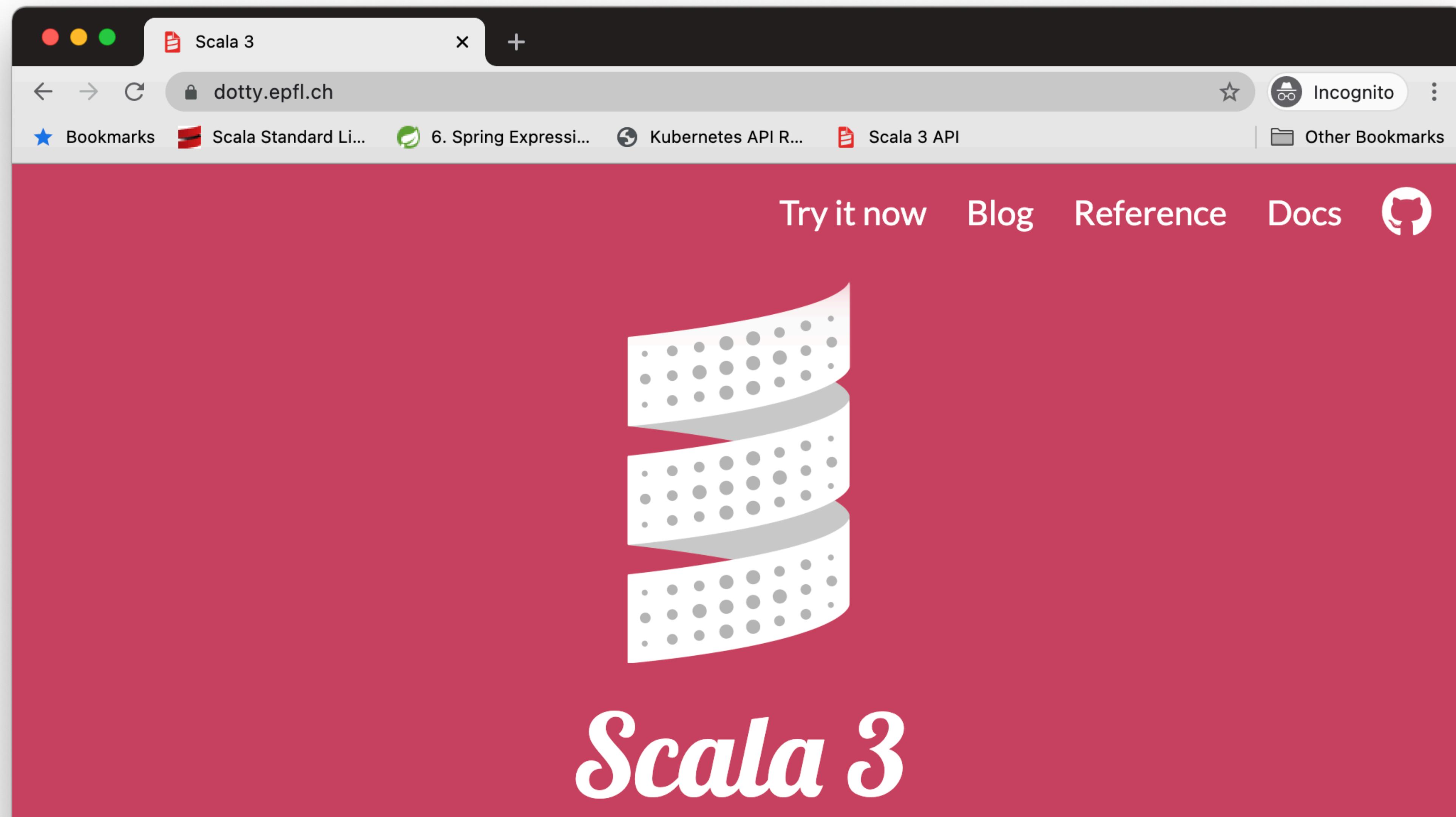
# References (For Now)





Programming Scala, 3rd Edition  
by Dean Wampler  
Publisher: O'Reilly Media, Inc.  
Release Date: August 2021  
ISBN: 9781492077893

<https://dotty.epfl.ch>





# Thank You

Daniel Hinojosa

Twitter: @dhinojosa

Email: [dhinojosa@evolutionnext.com](mailto:dhinojosa@evolutionnext.com)