SCALA 3.0 DOTTY

Daniel Hinojosa

@dhinojosa dhinojosa@evolutionnext.com

Slides and Demos

https://github.com/dhinojosa/dotty-study/

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

Scala 3

Dotty Compiler

- New Compiler
- Scheduled for Release in 2020
- http://dotty.epfl.ch/
- Currently @ 0.16-RC3
- Things are still-a-changin'!

Scala 3

Intersection Types

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

```
Demo:
com.evolutionnext.intersectiontypes.IntersectionTypes
```

Scala 3

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:
com.evolutionnext.uniontypes.UnionTypes
com.evolutionnext.uniontypes.UnionTypesWithObjects
```

Scala 3

Enumerations

- Enumerations in Scala originally were kind of terrible and ignored
- Redesigned with simplicity in mind
- May not be quite as useful, since *union types* can model an enumeration
- Compatible with Java Enumerations

```
Demo:
com.evolutionnext.enums.JavaEnums
com.evolutionnext.enums.ScalaEnums
com.evolutionnext.enums.UnionOfDisparateChildren
```

Scala 3

Abstract Data Types

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

```
Demo:
com.evolutionnext.abstractdatatypes.AbstractDataTypes
```

Scala 3

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:
`com.evolutionnext.traitparameters.TraitParameters`

Scala 3

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.

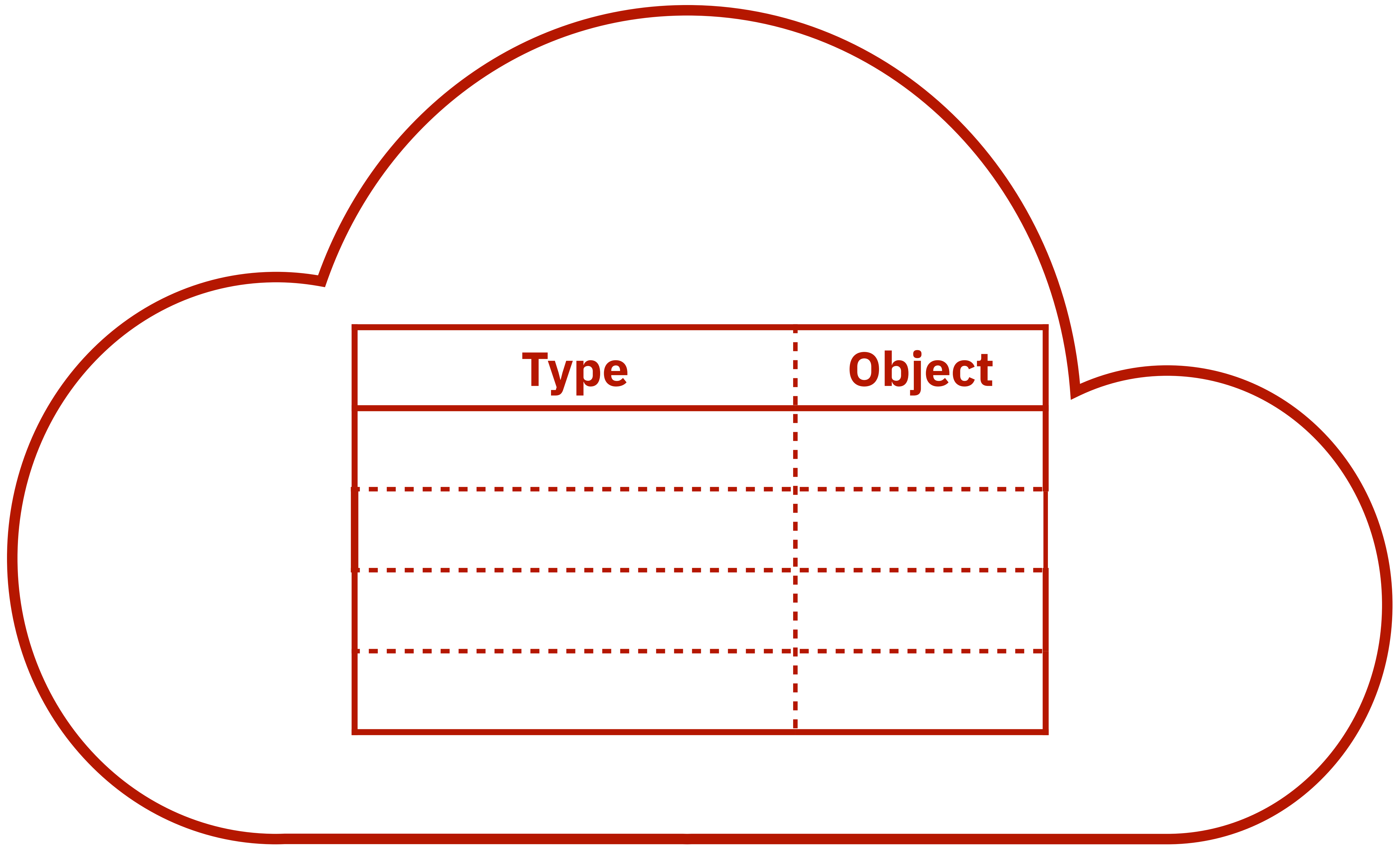
```
Demo:
com.evolutionnext.matchtypes.MatchTypes
```

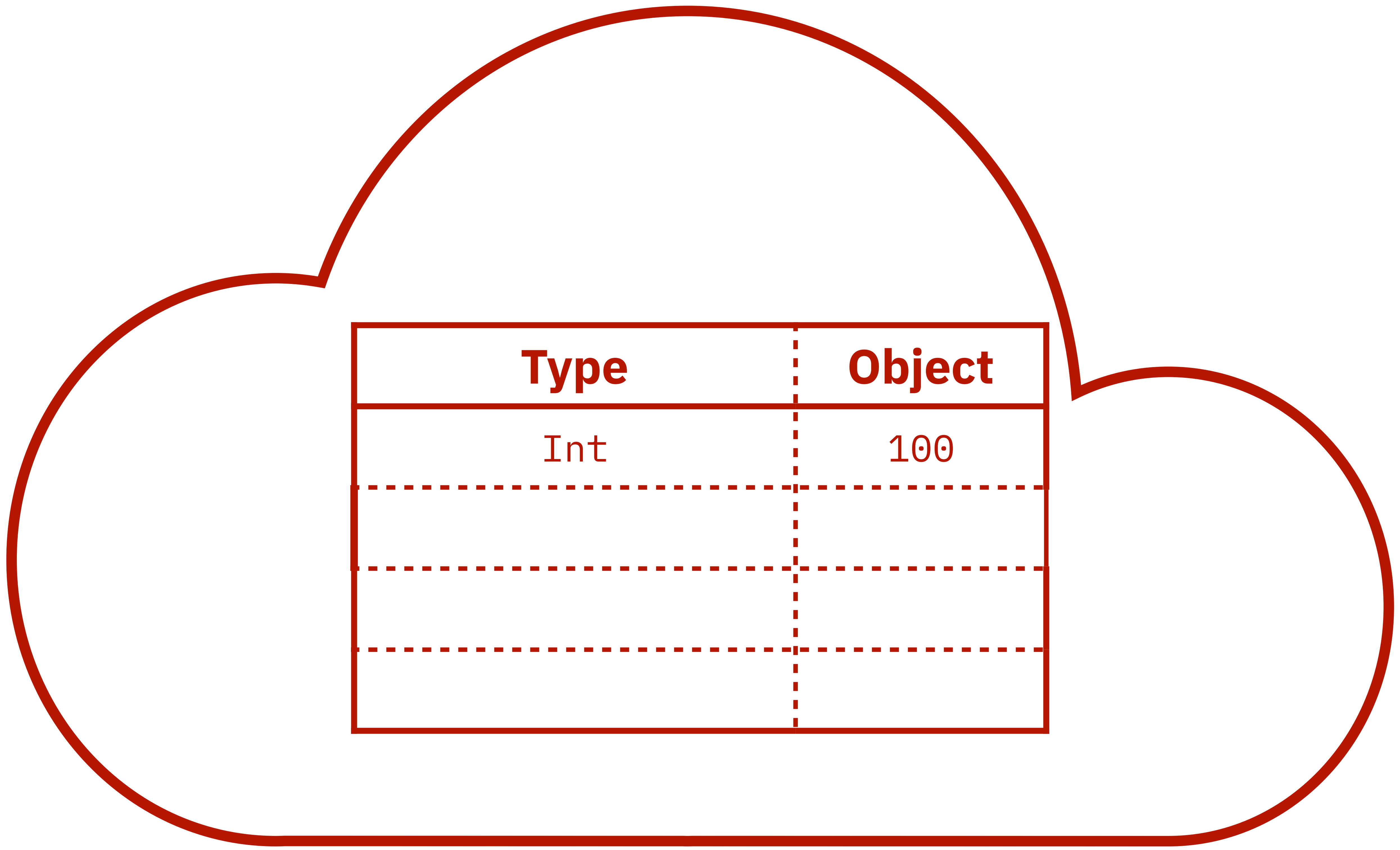
Scala 3

Implicit Replacement

- `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

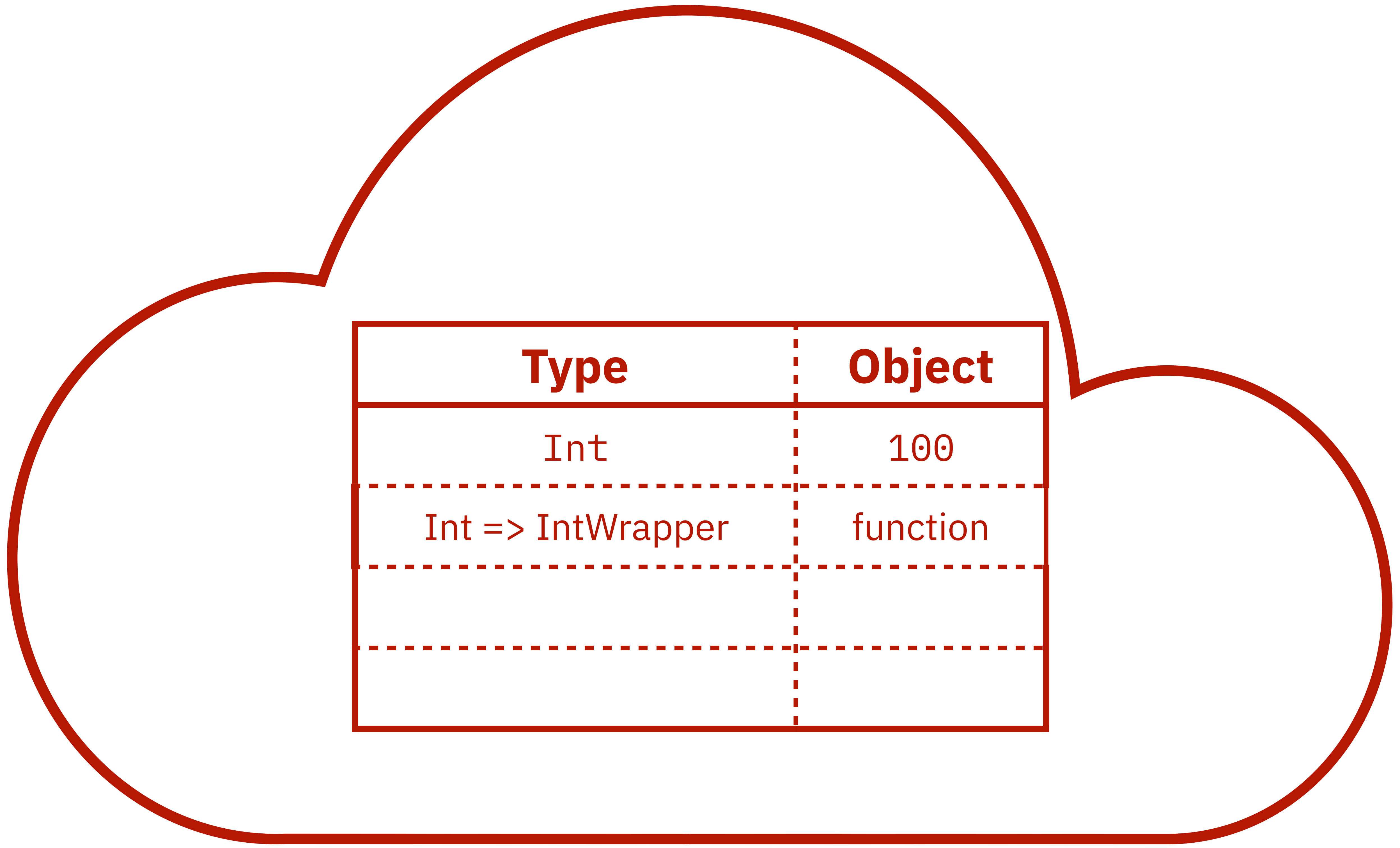
- It is the mean by which Scala programmers:
  - Create typeclasses
  - Create context values
  - Extends functionality
  - Perform Dependency Injection


Type
Object



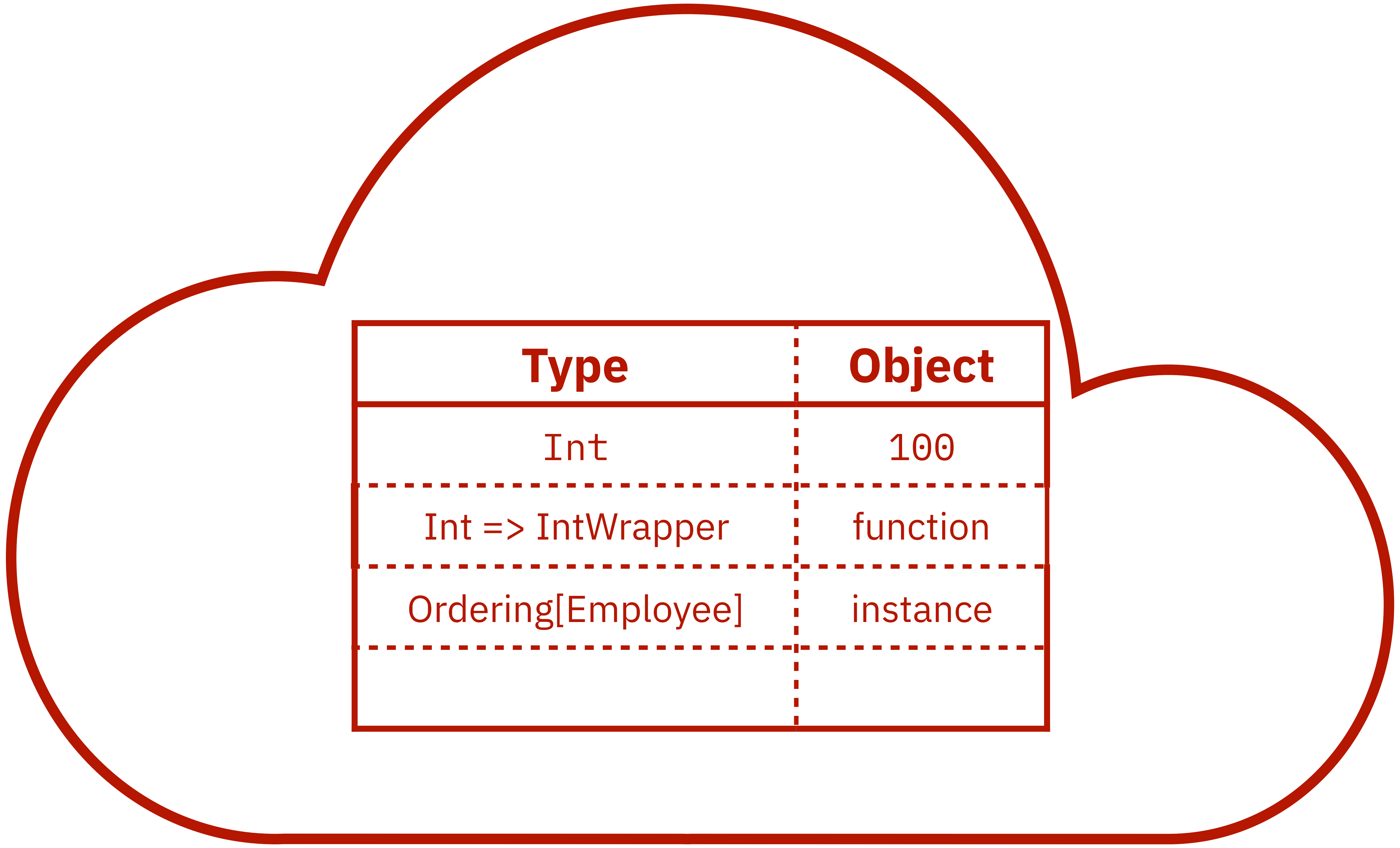
Type	Object
Int	100

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



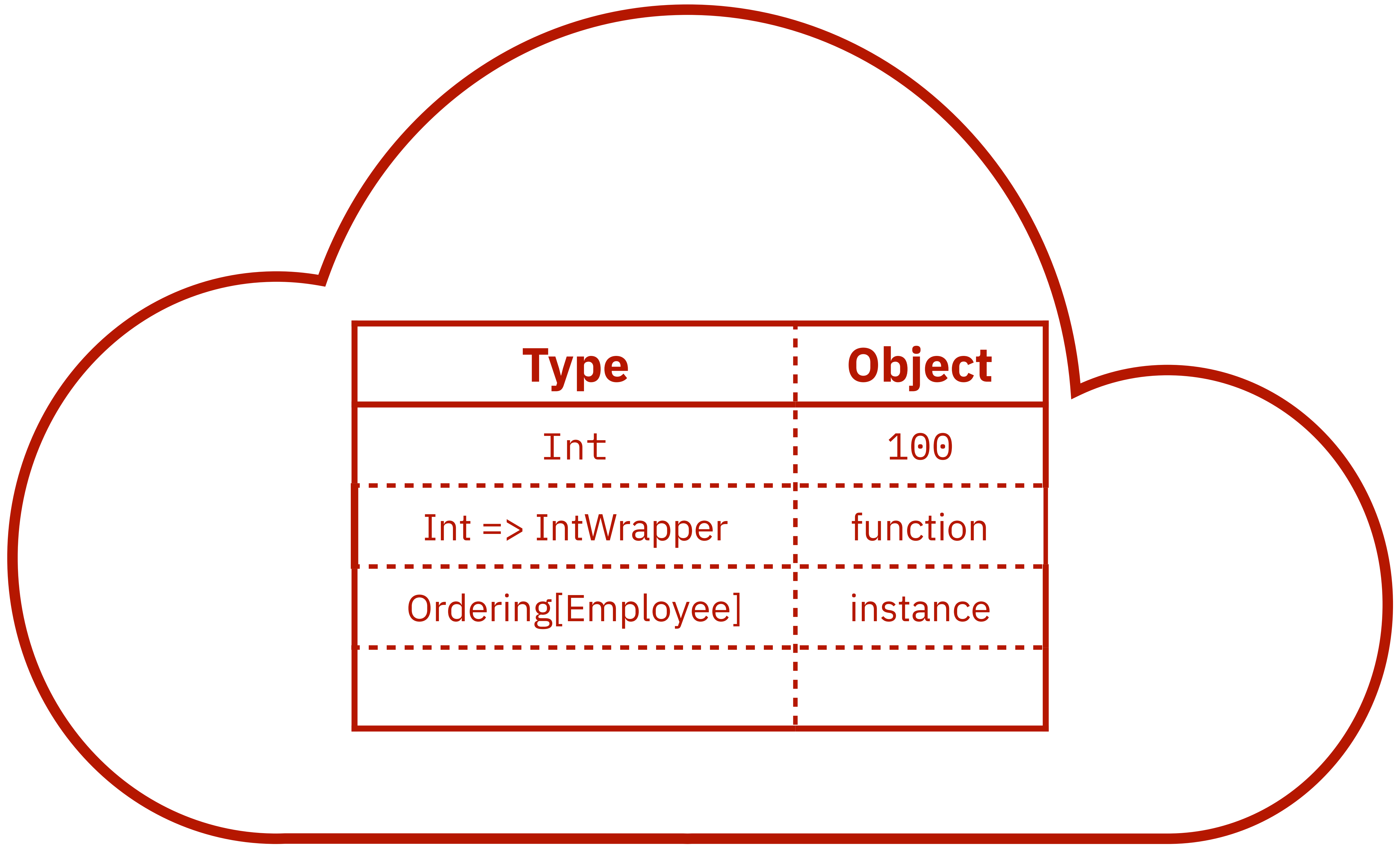
Type	Object
Int	100
Int => IntWrapper	function

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



Type	Object
Int	100
Int => IntWrapper	function
Ordering[Employee]	instance

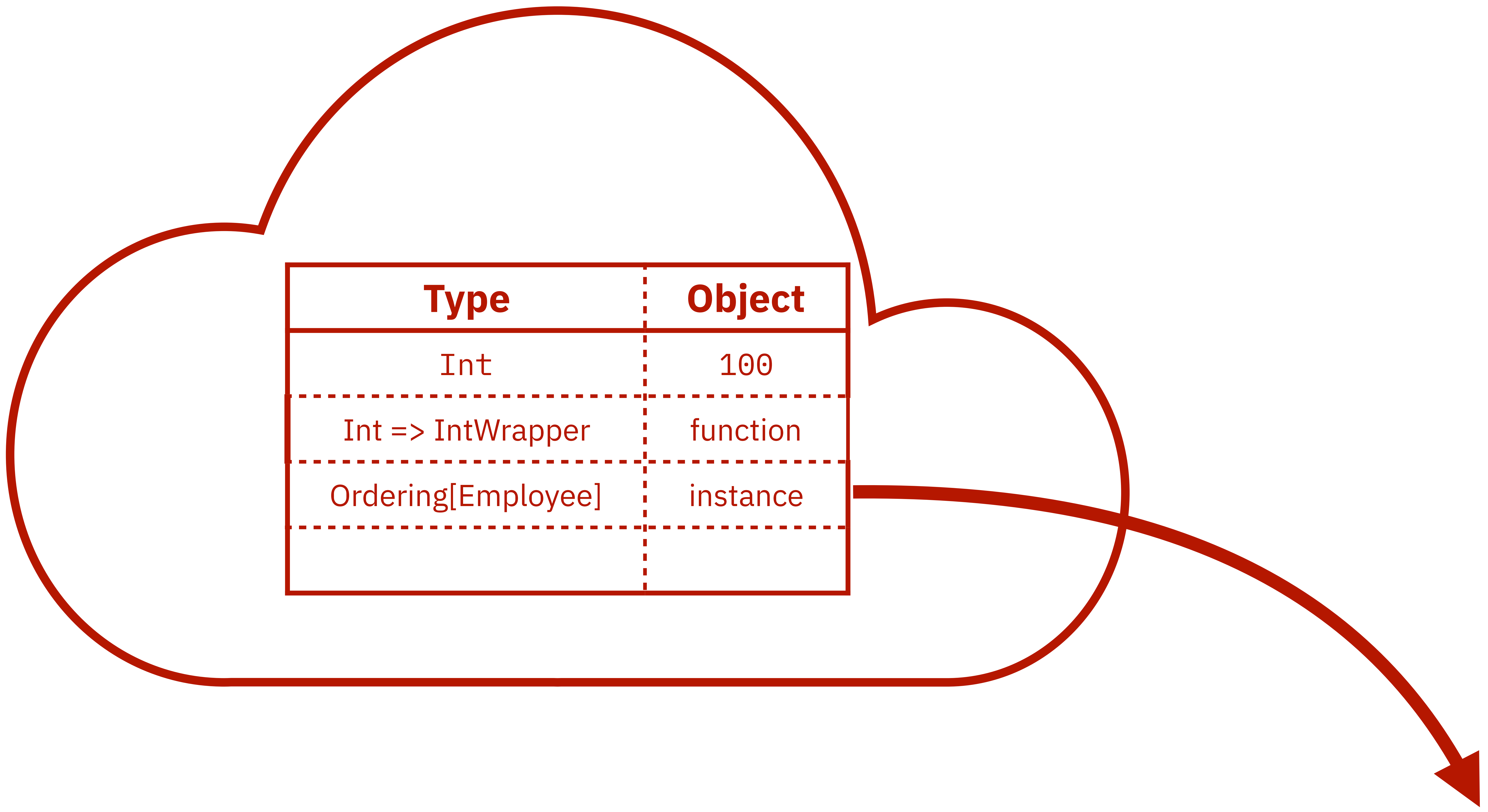
```
implicit ord:Ordering[Employee] = new Ordering[Employee]{...}
```



Type	Object
Int	100
Int => IntWrapper	function
Ordering[Employee]	instance

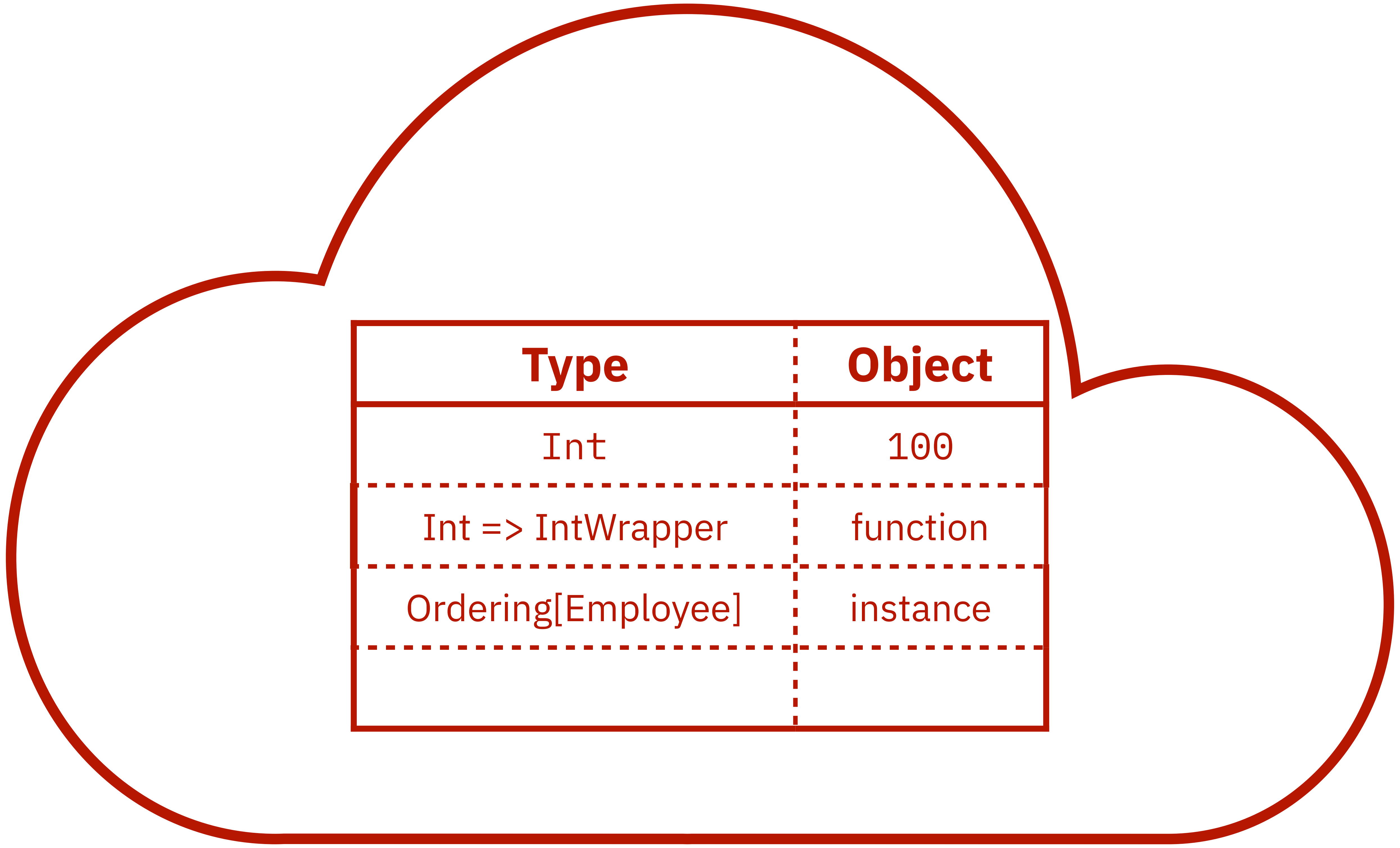
?

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



Type	Object
Int	100
Int => IntWrapper	function
Ordering[Employee]	instance

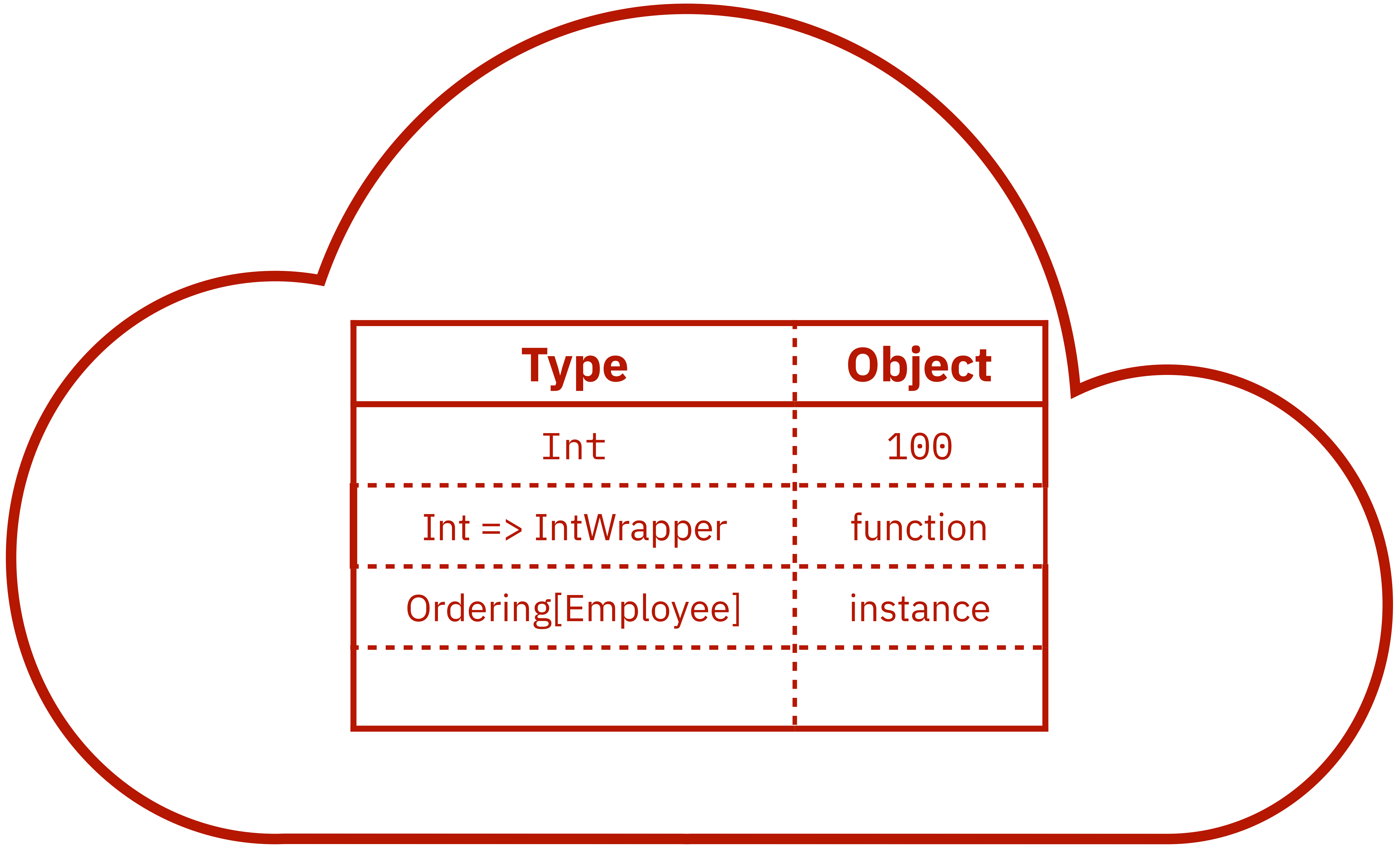
```
def mySortingMethod(list:List[Employee])(implicit o:Ordering[Employee])
```



Type	Object
Int	100
Int => IntWrapper	function
Ordering[Employee]	instance

?

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



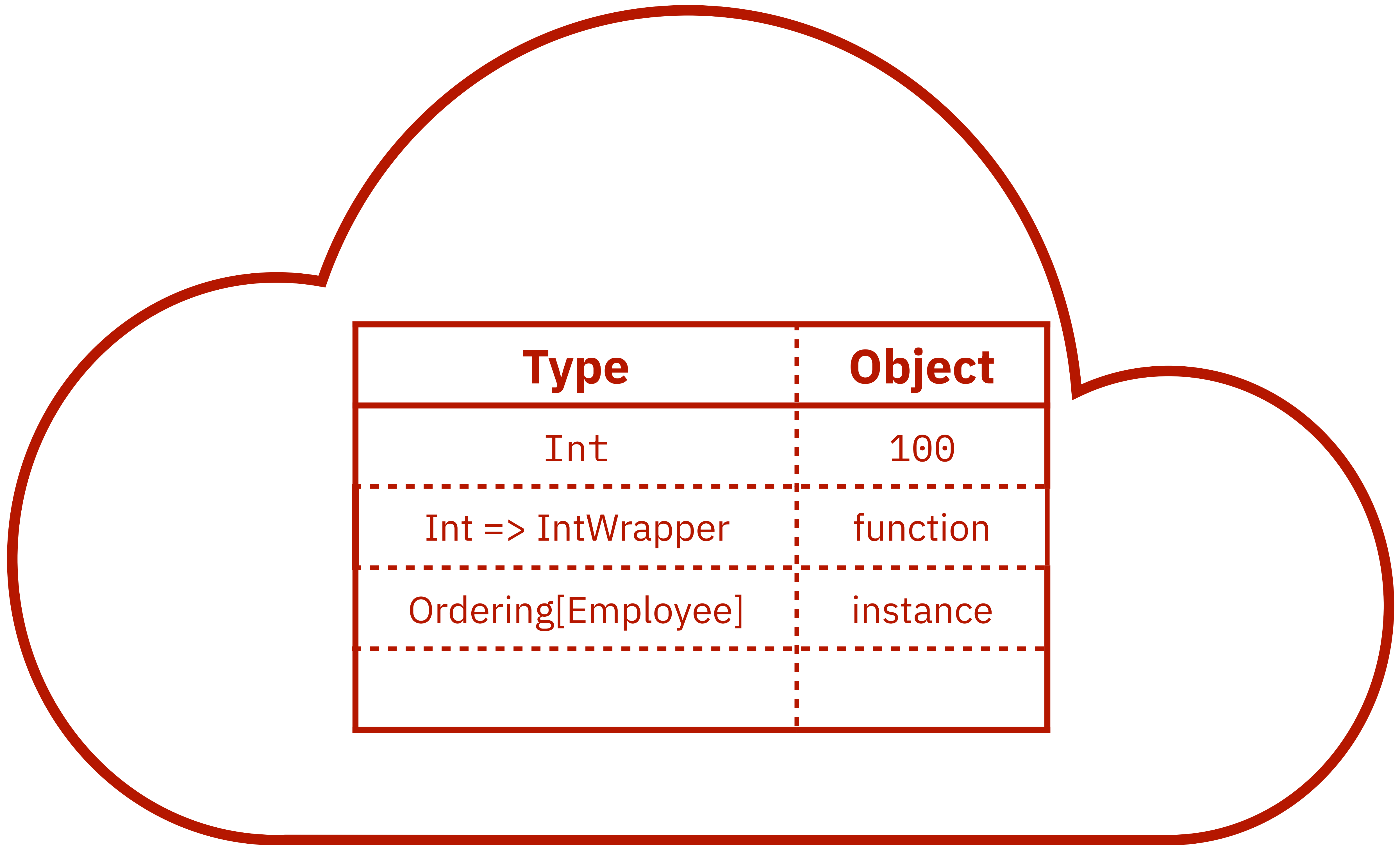
Type	Object
`Int`	`100`
Int => IntWrapper	function
Ordering[Employee]	instance

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

`No implicit ordering found for implicit`

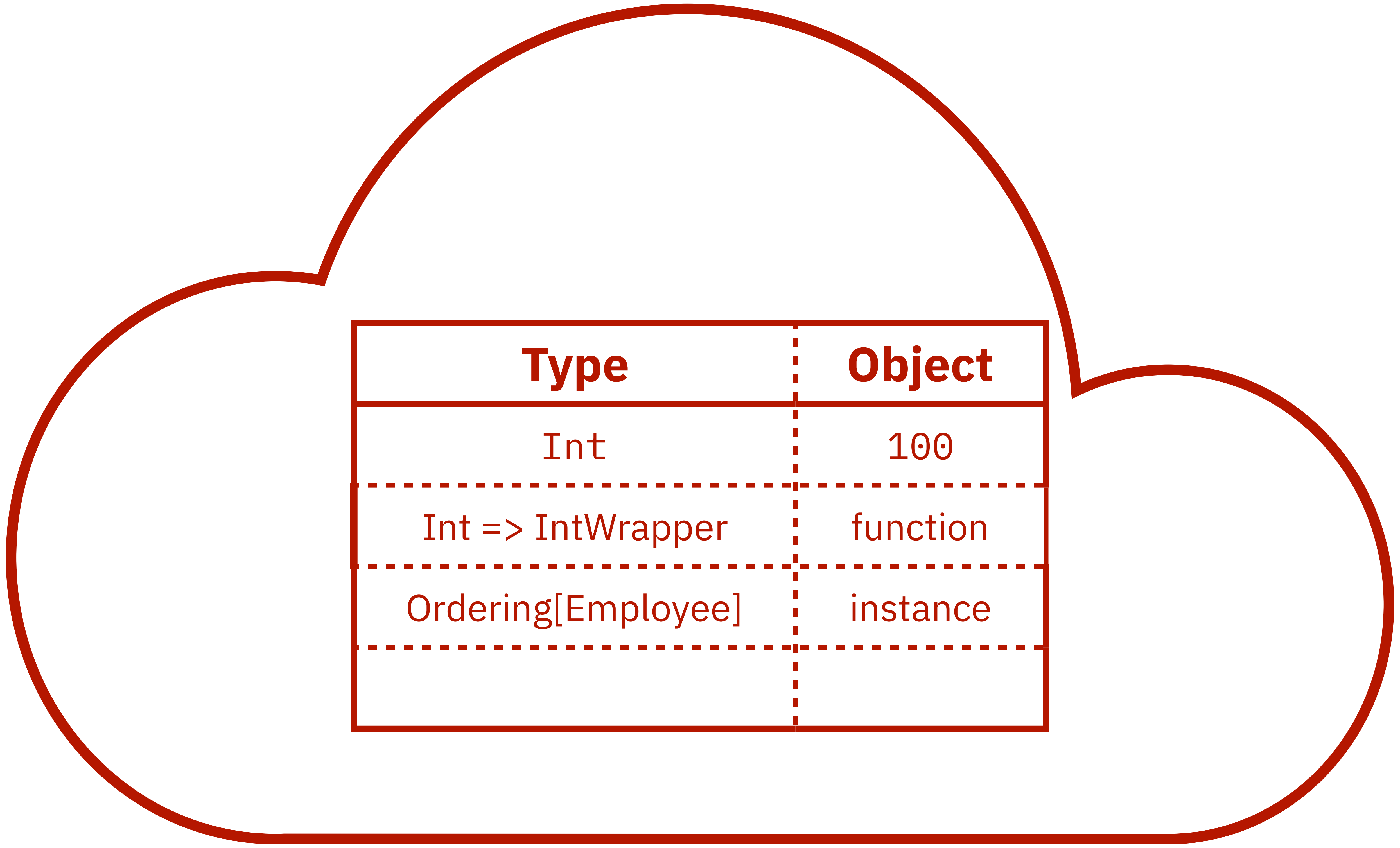
Scala 3

Delegates & Given



Type	Object
Int	100
Int => IntWrapper	function
Ordering[Employee]	instance

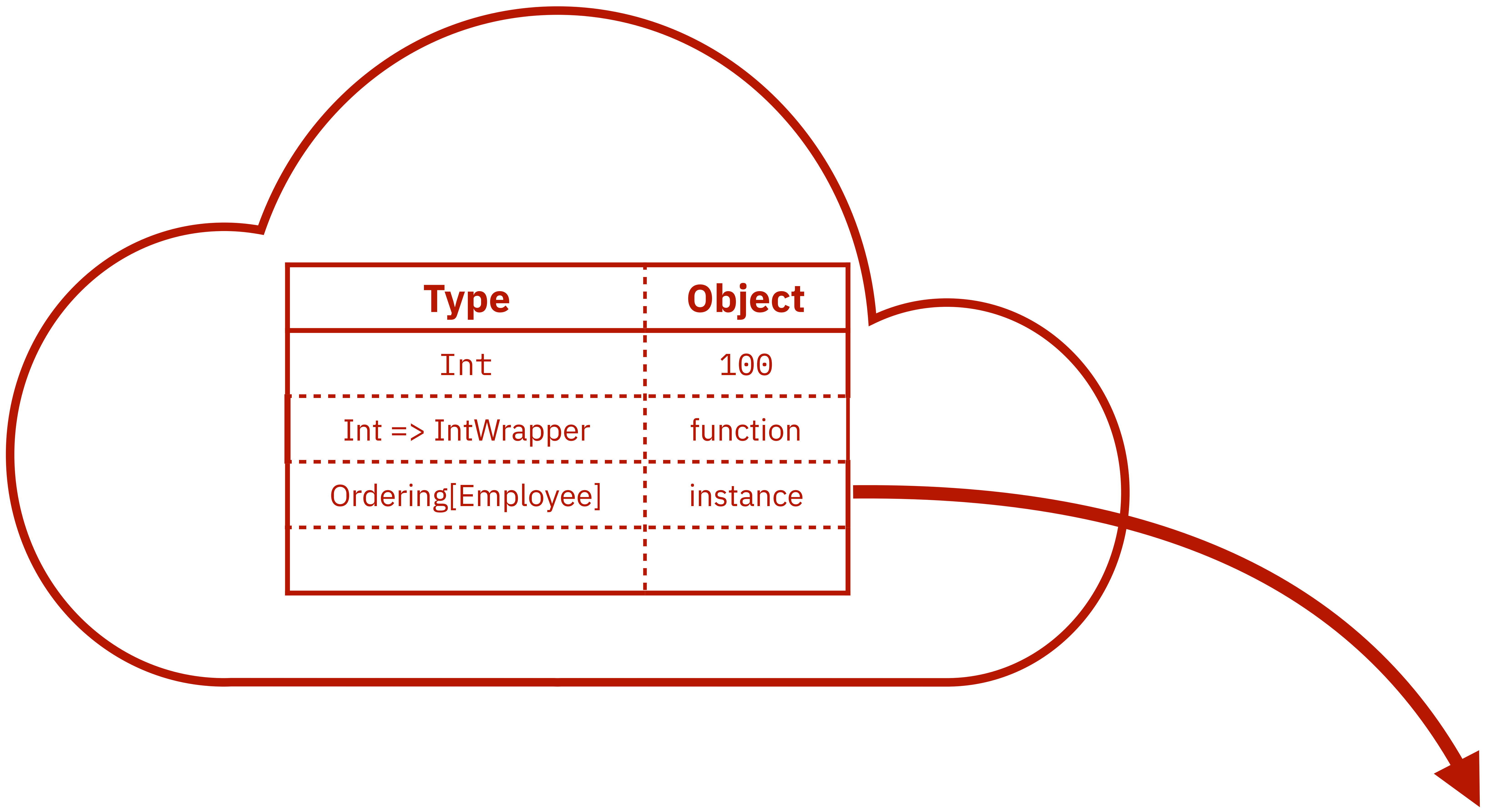
```
delegate o for Ordering[Employee] = ...
```



Type	Object
Int	100
Int => IntWrapper	function
Ordering[Employee]	instance

?

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



Type	Object
Int	100
Int => IntWrapper	function
Ordering[Employee]	instance

```
def mySortingMethod(list:List[Employee] given (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]
```

Scala 3

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:
com.evolutionnext.givendelegates.ExecutionContextDelegate
```

Scala 3

Extension Methods

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

```
Demo:
com.evolutionnext.extensionmethods.ExtensionMethods
```

Scala 3

TypeClass Implementations

Scala 3

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:
com.evolutionnext.multiversalequality.MultiversalEquality
```

Scala 3

Implicit 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`

```
Demo:
com.evolutionnext.conversions.Conversions
```

Scala 3

Opaque Types

- Provides a completely new type based on the previous one
  - They are not synonymous or an alias
  - Information hiding

```
Demo:
com.evolutionnext.opaquetypes.OpaqueTypes
```

Scala 3

Parameter Untupling

- One pain point with Scala is use 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:
com.evolutionnext.parameteruntupling.ParameterUntupling
```

Scala 3

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:
com.evolutionnext.tupledfunction.TupledFunction
```

Scala 3

Creator Applications

- Instantiate a class without a `new`
- Makes the language more unified as to how to instantiate or create objects
- Compliments `apply`.
- Internally there is a 4th rule that if there is a stable identifier, then call it with `new`

```
Demo:
com.evolutionnext.creatorapplications.CreatorApplications
com.evolutionnext.creatorapplications.CreatorApplicationsUsingJava
```

Scala 3

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:
`com.evolutionnext.structuraltypes.StructuralTypes`

Scala 3

Type Lambdas

- First a note about 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 strict functional programming

- Defines a function of types to types
- They may carry bounds and variances
- Allows us to express a higher kinded type

Scala 3

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

Thank you

@dhinojosa
dhinojosa@evolutionnext.com