Type Classes in Scala and Haskell

Table of Contents

Example: List.sorted + List.sum

- class List[+A] {
  - …
  - def sorted[B >: A](implicit ord: math.Ordering[B]): List[A]
  - def sum[B >: A](implicit num: Numeric[B]): B
  - …
- }

Some Type Classes (Scala)

- scala.math.Ordering
- scala.math.Numeric
- cats.Monoid
- cats.Functor
- cats.Monad
- etc.

How to use the Type Class Pattern

- Define a type class (as a trait)
- Define a type class instance for each type that should support the type class (as an implicit val)
- Use the type class instance implicitly (= as an implicit parameter to another method or function)

Define a type class

- trait Printable[A] {
-   def format(value: A): String
- }

Define type class instances (1)

- implicit val intPrintable: Printable[Int] = new Printable[Int] {
-   override def format(value: Int): String =
    - "How many cats? " + value.toString
- }

- implicit val datePrintable: Printable[Date] = new Printable[Date] {
-   override def format(value: Date): String =
    - "Date of meeting: " + value.toString
- }

Use the type class instance (1)

- def myPrint[A](value: A)(implicit printable: Printable[A]): Unit =
-         println(printable.format(value))

- myPrint(2)
- myPrint(new Date)

Define type class instances (2)

```scala
final case class Cat(name: String, age: Int, color: String)
object Cat {
  implicit val catPrintable: Printable[Cat] = new Printable[Cat] {
    override def format(cat: Cat): String = {
      val name  = Printable.format(cat.name)
      val age   = Printable.format(cat.age)
      val color = Printable.format(cat.color)
      s"$name is a $age year-old $color cat."
    }
  }
}
```

Use the type class instance (2)

- def myPrint[A](value: A)(implicit printable: Printable[A]): Unit =
- println(printable.format(value))

- val mizzi = Cat("Mizzi", 1, "black")
- val garfield = Cat("Garfield", 38, "ginger and black")

- myPrint(mizzi)
- myPrint(garfield)

Better Design

- Move the print method into a singleton object (e.g. the companion object of the type class).
- Use extension methods (= type enrichment) by defining an implicit class. (The implicit class must be parameterized with the same type as the type class.)

Better Design (1)

- Move the print method into a singleton object (e.g. the companion object of the type class).
- object Printable {
-   def format[A](value: A)(implicit printable: Printable[A]): String =
    - printable.format(value)
-   def print[A](value: A)(implicit printable: Printable[A]): Unit =
    - println(printable.format(value))
- }
- Printable.print(mizzi)

Better Design (2)

- Use extension methods (= type enrichment) by defining an implicit class. (The implicit class must be parameterized with the same type as the type class.)
- `implicit class PrintableOps[A](value: A) {`
- `  def format(implicit printable: Printable[A]): String =`
    - `printable.format(value)`
- `  def print(implicit printable: Printable[A]) = println(format)`
- `}`
- `mizzi.print`

Where to keep the type class instances?

- Type class instances for standard types (String, Int, Date etc.) should be stored in the same package as the type class itself.
- Type class instances for your own types like domain classes (Cat, Person, Order etc.) should be stored in the same package as the respective domain class.

Benefit of type classes

- The type class (Printable) and the domain class (Cat) are completely decoupled.
- You can extend and enrich not only your own types but also sealed types from libraries which you do not own.

Type classes in Haskell

- Define a type class.
- Define a type class instance for each type that should support the type class. This enriches each type with the methods of the type class.
- Use the type class methods for the types that have an instance.

Define a type class

- class Printable a where
- format :: a -> String
-
- pprintt :: a -> IO ()
- pprintt x = putStrLn $ format x

Define type class instances (1)

- instance Printable Int where
-     format = show

- instance Printable UTCTime where
-     format time = "The exact date is: " ++ formatTime defaultTimeLocale "%F, %T (%Z)" time

Define type class instances (2)

- data Cat = Cat
-   { name :: String
-   , age   :: Int
-   , color :: String
-   }

- instance Printable Cat where
-     format cat = "Cat {name=" ++ name cat ++ ", age=" ++ show (age cat) ++ ", color=" ++ color cat ++ "}"

Use the type class methods with the instance types.

- putStrLn $ format $ utcTime 2018 3 8 16 38 19
- pprintt $ utcTime 2018 3 8 16 38 19

- let mizzi = Cat "Mizzi" 1 "black"
-     garfield = Cat "Garfield" 38 "ginger and black"
- putStrLn $ format mizzi
- pprintt mizzi
- putStrLn $ format garfield
- pprintt garfield

Comparison

- Haskell has its own type class syntax.
- Scala uses implicits to provide type classes.
- In Scala (using implicit val ...) you need to create an object for each type class instance.
- In Haskell object creation is avoided.

Resources

- Source code and slides – https://github.com/hermannhueck/typeclasses
- „Scala with Cats“ by Noel Welsh and Dave Gurnell – https://gumroad.com/discover?query=scala+cats
- „Haskell Programming from first principles“ by Christoper Allen and Julie Moronuki – https://gumroad.com/discover?query=allen+haskell

Thank you!

Q & A