New types as a solution to primitive obsession

The Art of Scala

POWERED BY Evolution

Rafał Piotrowski

Rafał Piotrowski

ITERATORS

https://twitter.com/r_piotrow

https://github.com/rpiotrow

Primitive obsession

```
case class Person(firstName: String, lastName: String)


val johnDoe: Person = Person("John", "Doe")


val doeJohn: Person = Person("Doe", "John")
```

New types

```
case class Person(firstName: FirstName, lastName: LastName)
```

```
val johnDoe: Person = Person(FirstName("John"), LastName("Doe"))
```

- less error-prone
- possibility to enrich types with methods

Value classes: code

```
case class FirstName(value: String) extends AnyVal

case class LastName(value: String) extends AnyVal
```

Value classes: pros and cons

PROS

- simple
- not much additional code
- works in Scala 3 and Scala 2

CONS

- sometimes not cost-free
- serialization/codecs require additional code

Tagged types: code

```
trait FirstNameTag
type FirstName = String with FirstNameTag
object FirstName:
  def apply(value: String): FirstName =
    value.asInstanceOf[FirstName]

trait LastNameTag
type LastName = String with LastNameTag
object LastName:
  def apply(value: String): LastName =
    value.asInstanceOf[LastName]
```

Tagged types: pros and cons

PROS

- cost-free
- works in Scala 3 and Scala 2

CONS

- complex
- boilerplate code
- serialization/codecs require additional code

Opaque types: code

```
opaque type FirstName = String
object FirstName:
  def apply(value: String): FirstName = value

opaque type LastName = String
object LastName:
  def apply(value: String): LastName = value
```

Opaque types: pros and cons

PROS

- cost-free

CONS

- boilerplate code
- serialization/codecs require additional code
- only Scala 3

Libraries

Monix’s Newtypes

https://newtypes.monix.io/

Monix's Newtypes: simple type

```
import monix.newtypes._

type FirstName = FirstName.Type
object FirstName extends NewtypeWrapped[String]

type LastName = LastName.Type
object LastName extends NewtypeWrapped[String]
```

Monix's Newtypes: validated type (1)

```
type Title = Title.Type
object Title extends NewtypeValidated[String]:
  def apply(value: String): Either[BuildFailure[Title], Title] =
    if !value.isBlank then
      Right(unsafeCoerce(value))
    else
      Left(BuildFailure("empty title"))
```

Monix's Newtypes: validated type (2)

```
type ISBN = ISBN.Type
object ISBN extends NewtypeValidated[String]:
  def apply(value: String): Either[BuildFailure[ISBN], ISBN] =
    if IsbnValidator.validate(value) then
      Right(unsafeCoerce(value))
    else
      Left(BuildFailure("invalid ISBN"))
```

[Demo]

Monix's Newtypes: pros and cons

PROS

- cost-free
- same code for Scala 3 and Scala 2
- built-in support for implementing custom validations (but only runtime)
- no dependencies
- integration with circe

CONS

- serialization/codec (except circe) require additional code
- need to use `unsafeCoerce` in `NewtypeValidated`

ZIO Prelude

https://zio.github.io/zio-prelude/

ZIO Prelude: simple type

```
import zio.prelude.Newtype

type FirstName = FirstName.Type
object FirstName extends Newtype[String]

type LastName = LastName.Type
object LastName extends Newtype[String]
```

ZIO Prelude: validated type (1)

```
type Title = Title.Type
object Title extends Newtype[String]:
  override inline def assertion: Assertion[String] =
    matches(".*\\S+.*".r)
```

ZIO Prelude: validated type (2)

```
type ISBN = ISBN.Type
object ISBN extends Subtype[String]:
  override inline def assertion: Assertion[String] =
    matches("\\d{9}[0-9X]|\\d{13}".r)
  extension (self: ISBN)
    def validated: Validation[String, ISBN] =
      Validation.fromPredicateWith("Invalid ISBN")(self)(
        IsbnValidator.validate
      )
```

[Demo]

ZIO Prelude: pros and cons

PROS

- cost-free
- new types and sub-types
- same code for Scala 3 and Scala 2 (if assertions are not used)
- built-in support for compile time assertions
- built-in support for implementing custom validations

CONS

- different code for assertions in Scala 3 and Scala 2
- serialization/codec require additional code
- adds ZIO to dependencies

Kebs

https://github.com/theiterators/kebs

Kebs opaque: simple type

```
import pl.iterators.kebs.opaque._

opaque type FirstName = String
object FirstName extends Opaque[FirstName, String]

opaque type LastName = String
object LastName extends Opaque[LastName, String]
```

Kebs opaque: validated type (1)

```
opaque type Title = String
object Title extends Opaque[Title, String]:
  override protected def validate(value: String):
      Either[String, Title] =
    if !value.isBlank then
      Right(value)
    else
      Left("empty title")
```

Kebs opaque: validated type (2)

```
opaque type ISBN = String
object ISBN extends Opaque[ISBN, String]:
  override protected def validate(value: String):
      Either[String, ISBN] =
    if IsbnValidator.validate(value) then
      Right(value)
    else
      Left("invalid ISBN")
```

[Demo]

Kebs opaque: pros and cons

PROS

- cost-free
- built-in support for implementing custom validations
- no dependencies
- built-in codecs for:
  - circe [COMING SOON]
  - play-json [PROBABLY COMING SOON]
  - akka-http [PROBABLY COMING SOON]
  - scalacheck [PROBABLY COMING SOON]

CONS

- only Scala 3

Kebs tagged: simple type

```
import pl.iterators.kebs.tagged._
import pl.iterators.kebs.tag.meta.tagged

@tagged object PersonDomain {

  trait FirstNameTag
  type FirstName = String @@ FirstNameTag

  trait LastNameTag
  type LastName = String @@ LastNameTag
}
```

Kebs tagged: validated type (1)

```
trait TitleTag
type Title = String @@ TitleTag
object Title {
  def validate(value: String): Either[String, String] =
    if (!value.isBlank)
      Right(value)
    else
      Left("empty title")
}
```

Kebs tagged: validated type (2)

```
trait ISBNTag
type ISBN = String @@ ISBNTag
object ISBN {
  def validate(value: String): Either[String, String] =
    if (IsbnValidator.validate(value))
      Right(value)
    else
      Left("invalid ISBN")
}
```

[Demo]

Kebs tagged: pros and cons

PROS

- cost-free
- built-in support for implementing custom validations
- no dependencies
- built-in codecs for:
  - circe
  - slick
  - play-json
  - spray-json
  - akka-http
  - jsonschema
  - scalacheck
- plugin for IntelliJ

CONS

- only Scala 2

Other

- Refined + scala-newtypes in Scala 2
  - https://github.com/fthomas/refined
  - https://github.com/estatico/scala-newtype
- Refined in Scala 3
- Mazboot (Scala 3)
  - https://github.com/tabdulradi/mazboot

Summary

Final conclusions

- new types can be created using Scala language features:
  - value classes
  - tagged types
  - opaque types
- or libraries:
  - Monix's Newtypes
  - ZIO Prelude
  - kebs

Which should I choose?

- **if** ZIO **then** use ZIO Prelude
- **if** scala2 **then** use kebs or Refined+scala-newtypes or Monix
- **else** use kebs or Monix

Links

- Libraries
  - https://newtypes.monix.io/
  - https://zio.github.io/zio-prelude/
  - https://github.com/theiterators/kebs
  - https://github.com/fthomas/refined
  - https://github.com/estatico/scala-newtype
  - https://github.com/tabdulradi/mazboot
- Sample code from this presentation
  - https://github.com/rpiotrow/scala-love-2022

Thank you!