# Monocle

/DEV
/ Lyon

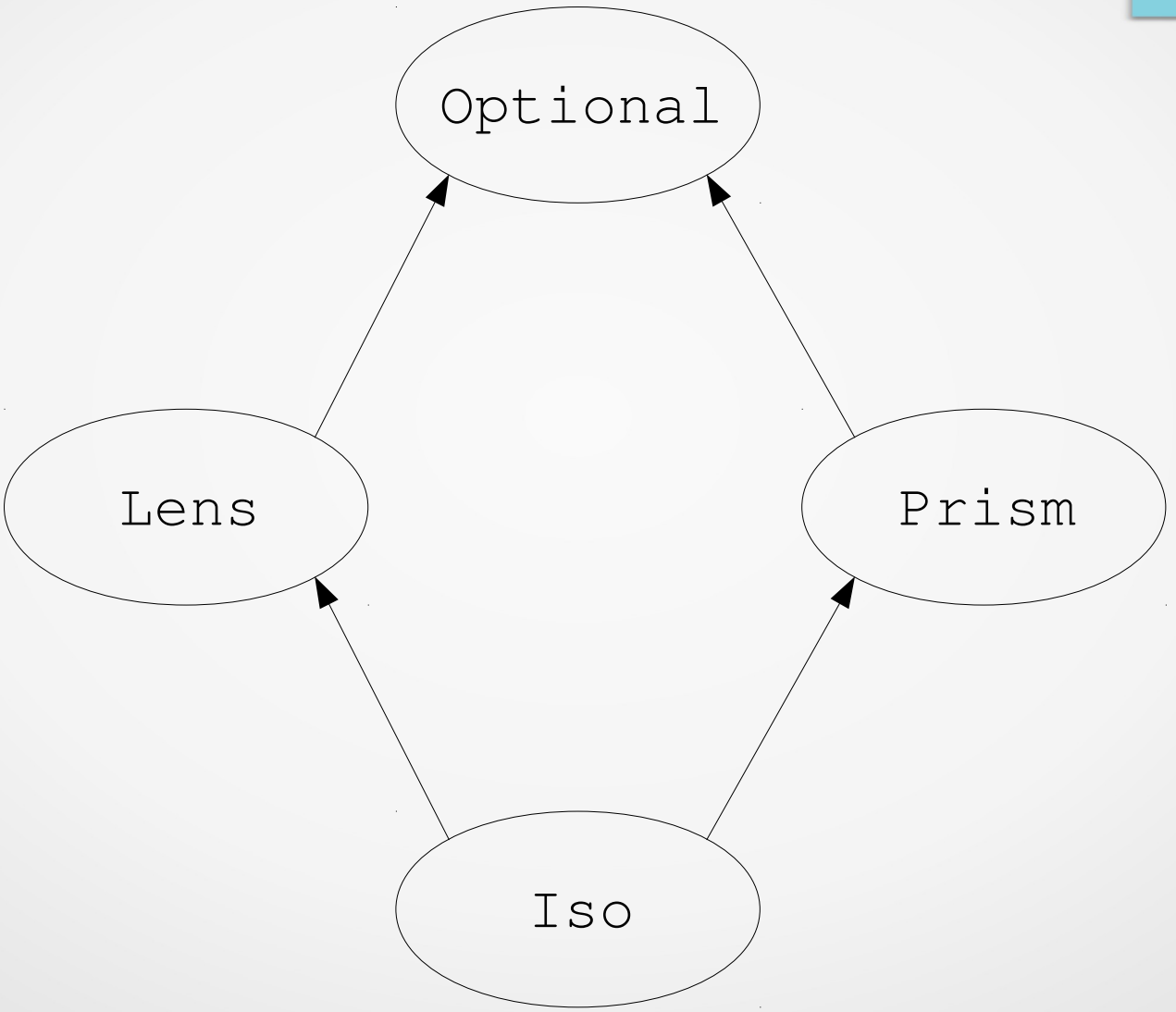
@karolchmist

# Pourquoi

- Modifier des données immuables

```
facture.copy(
  client = facture.client.copy(
    address = facture.client.address.copy(
      city = facture.client.address.city.toUpperCase
)))
```

# Optiques de base



# Iso

```
case class Iso[S,A] (
  get :        S => A,
  reverseGet : A => S
)

reverseGet(get(s)) == s
get(reverseGet(a)) == a
```

# Iso - exemple

```
case class Kilogram(v: double)
case class Pound(v: double)

val kgToPound = Iso[Kilogram, Pound](
  get =        k => Pound(k.v / 0.45359237),
  reverseGet = p => Kilogram(p.v * 0.45359237)
)

kgToPound.get(Kilogram(23)) == Pound(50.70632030252184)

kgToPound.reverseGet(Pound(50.70632030252184))
                    == Kilogram(23)
```

# Iso - exemple

```
case class Kilogram(v: double)
case class Pound(v: double)
case class Stone(v: double)

val poundToStone = Iso[Pound, Stone](
  get =        p => Stone(p.v / 14),
  reverseGet = s => Pound(s.v * 14)
)

val kgToStone = kgToPound composeIso poundToStone

kgToStone.get(Kilogram(23)) == Stone(3.621880021608703)
```

# Iso - Lois

```
reverseGet(get(s)) == s
get(reverseGet(a)) == a
```


```
kgToPound.get(
  kgToPound.reverseGet(
    Pound(50.706320302521840000001)))
 == Pound(50.70632030252184)
```

# Prism

- Iso avec getOption au lieu de get

```
case class Prism[S,A](
  getOption  : S => Option[A],
  reverseGet : A => S
)

getOption(s) map reverseGet == Some(s) || None
getOption(reverseGet(a))    == Some(a)
```

# Prism – exemple

```
sealed trait Character
case class Hero (level: Int)    extends Character
case class Enemy(name : String) extends Character

object Character {
  val _hero:  Prism[Character, Hero]  = GenPrism[Character, Hero]
  val _enemy: Prism[Character, Enemy] = GenPrism[Character, Enemy]
}

val hero:  Character = Hero(level = 12)
val enemy: Character = Enemy(name = "Dragon")

Character._hero.getOption(hero)  == Some(Hero(level = 12))
Character._hero.getOption(enemy) == None
```

# Lens

- Iso avec set au lieu de reverseGet

```
case class Lens[S,A] (
  get : S      => A,
  set : (A,S) => S
)

set(get(s), s) == s
get(set(a, s)) == a
```

# Lens - création

```
case class Facture(client: Client, id: String)
case class Client(name: String, address: Address)
case class Address(street: Street, city: String)
case class Street(name: String, number: Int)

val _client1 : Lens[Facture, Client] =
  Lens[Facture, Client](_.client,
                        c => f => f.copy(client = c))

val _client2 : Lens[Facture, Client] =
                       GenLens[Facture](_.client)

@Lenses("_") case class Facture(client: Client, id: String)
```

# Lens - modifications

```
val lensFacture2StreetName: Lens[Facture, String] =
        (Facture._client
          composeLens Client._address
          composeLens Address._street
          composeLens Street._name)

val factureModified = lensFacture2StreetName
                      .modify(_.toUpperCase)(facture)

factureModified.client.address.street.name
                      == "RUE MERCIÈRE"
```

# Lens - DSL

```
val lensFactureToStreetNameDSL =
        (Facture._client
          ^|-> Client._address
          ^|-> Address._street
          ^|-> Street._name)
```

# Lens - modifyF

```
val address: Address =
        Address(Street("rue Maronier", 33), "Lyon")

val lensAddressToStreetNumber =
     (Address._street composeLens Street._number)

val neighbours: List[Address] =
     lensAddressToStreetNumber
     .modifyF(n => List(n - 1, n + 1))(address)

neighbours == List(
  Address(Street("rue Maronier", 32), "Lyon"),
  Address(Street("rue Maronier", 34), "Lyon"))
```

# Optional

- Prism + Lens

```
case class Optional[S,A] (
  getOptional: S      => Option[A],
  set:          (A, S) => S
)

getOption(s) map set(_, s) == Some(s)
getOption(set(a, s)) == Some(a) || None
```

# Optional – exemple

```
val adrsToStrtNameFirstLetter: Optional[Address, Char]
    = Address._street
      composeLens Street._name
      composeOptional headOption

val address = Address(Street("rue Servient"))

adrsToStrtNameFirstLetter.modify(_.toUpper)(address)
         == Address(Street("Rue Servient"))
```

# Optional - exemple

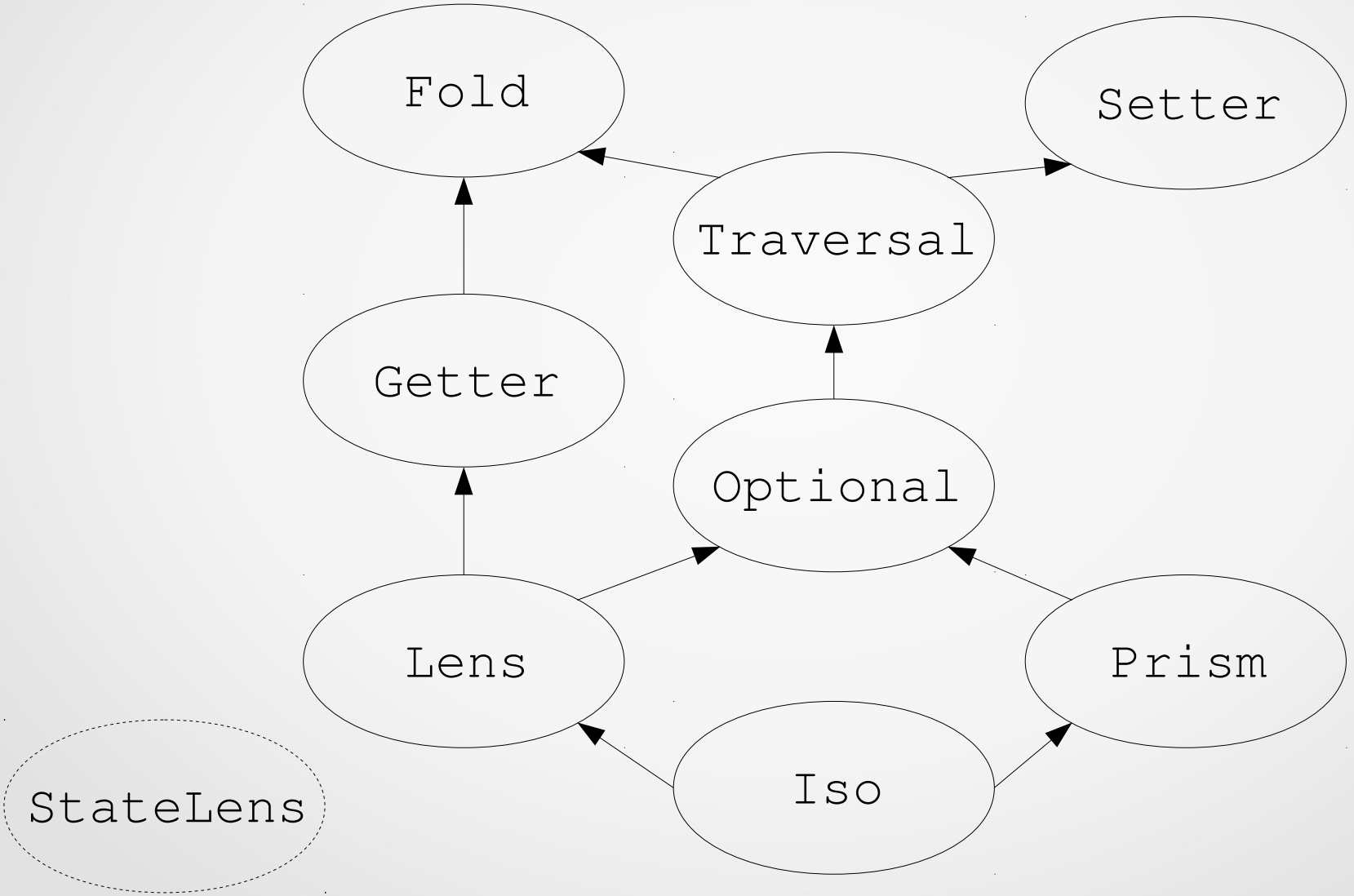
```
@Lenses("_") case class Carte(streets: List[Street])

val carte = Carte(List(Street("rue Servient"),
                       Street("cours gambetta")))

val optionalCarteToFirstStreetName: Optional[Carte, String]
        = Carte._streets composeOptional index(1)
                         composeLens Street._name

optionalCarteToFirstStreetName.modify(_.toUpperCase)(carte)
               == Carte(List(Street("rue Servient"),
                             Street("COURS GAMBETTA")))
```

# Hiérarchie


Fold
Setter
Traversal
Getter
Optional
Lens
Prism
Iso
StateLens

# Exemples

```
case class Artist(
        name: String,
        albums: List[Album])


case class Album(
        title: String,
        songs: List[Song],
        year: Year)


case class Song(
        title: String,
        length: Duration,
        singleReleaseYear: Option[Year])
```

# Exemple - each

```
val albumToSongsDuration: Traversal[Album, Duration]
        = Album._songs composeTraversal each
                       composeLens Song._length
```

```
val longerAlbum: Album = traversalAlbumToSongsLength
                           .modify(_.plusMinutes(1))(album)
```

# Exemple - fold

```
val durationToSeconds =
    Iso[Duration,Long](_.getSeconds)
```

→ (Duration.ofSeconds)

```
val albumToSongsLength: Traversal[Album, Long] =
albumToSongsDuration composeIso durationToSeconds
```

```
val allSongsLength: Long =
     albumToSongsLength.fold(album)
```

```
// standard Scala

album.songs.map(_.length.getSeconds).sum
```

# Librairies Lenses

- Monocle
- Scalaz
- QuickLens
- Shapeless
- ...

# Mes conclusions

Avantages

- Lois – garantie de sûreté
- Composition – réutilisation

Désavantages

- Code plus complexe potentiellement

# Sources

- Julien Truffaut – Beyond Scala Lenses

  http://www.slideshare.net/JulienTruffaut/beyond-scala-lens

- Ilan Godik – Optics with Monocle

  http://www.slideshare.net/IlanGodik/optics-with-monocle-modeling-the-part-and-the-whole

Questions ?

# Merci !

https://github.com/karolchmist/slug-monocle

@karolchmist