Mutable the immutable - introduction to Optics

Andrzej Ressel

12.07.2018

.copy

.mapIndexed

.mapValues

Basic example

```
data class Street(val number: Int, val name: String)
data class Address(val city: String, val street: Street)
```

Basic example

```
data class Street(val number: Int, val name: String)
data class Address(val city: String, val street: Street)

val address = Address("Google", Street(1600, "Amphitheatre
    Parkway"))

// address = Address(city=Google, street=Street(number=1600,
    name=Amphitheatre Parkway))
```

Basic example

```
data class Street(val number: Int, val name: String)
data class Address(val city: String, val street: Street)

val address = Address("Google", Street(1600, "Amphitheatre
    Parkway"))

// address = Address(city=Google, street=Street(number=1600,
    name=Amphitheatre Parkway))

val address2 = address.copy(
        street = address.street.copy(
                name = address.street.name.toUpperCase()
        )
)

// address2 = Address(city=Google, street=Street(number=1600,
    name=AMPHITHEATRE PARKWAY))
```

Advanced example

```
data class Street(val number: Int, val name: String)
data class Address(val city: String, val street: Street)
data class Company(val name: String,
                   val addresses: List<Address>)
data class Employee(val name: String, val company: Company)
```

Advanced example

```
data class Street(val number: Int, val name: String)
data class Address(val city: String, val street: Street)
data class Company(val name: String,
                   val addresses: List<Address>)
data class Employee(val name: String, val company: Company)

val employee = Employee("John Smith", Company("Google", listOf(
    Address("Mountain View", Street(1600, "Amphitheatre Parkway
    ")), Address("San Francisco", Street(1455, "Market Street")
    ))))

// employee = Employee(name=John Smith, company=Company(name=
    Google, addresses=[Address(city=Mountain View, street=
    Street(number=1600, name=Amphitheatre Parkway)), Address(
    city=San Francisco, street=Street(number=1455, name=Market
    Street))]))
```

Advanced example

```
val n = 1
val employee2 = employee.copy(
        company = employee.company.copy(
            addresses = employee.company.addresses.mapIndexed {
                index, address ->
                if (index != n) address
                else address.copy(
                    street = address.street.copy(
                        name = address.street.name.toUpperCase()
                    )
                )
            }
        )
    )

// employee2 = Employee(name=John Smith, company=Company(name=
    Google, addresses=[Address(city=Mountain View, street=
    Street(number=1600, name=Amphitheatre Parkway)), Address(
    city=San Francisco, street=Street(number=1455, name=MARKET
    STREET))]))
```

```
address.copy(
    street = address.street.copy(
        name = address.street.name.toUpperCase()
    )
)
```

```
address.copy(
    street = address.street.copy(
        name = address.street.name.toUpperCase()
    )
)


Address.street.name.modify(address, String::toUpperCase)
```

```
employee.copy(
    company = employee.company?.copy(
        addresses = employee.company.addresses.mapIndexed {
            index, address ->
            if (index != n) address
            else address.copy(
                street = address.street.copy(
                    name = address.street.name.toUpperCase()
                )
            )
        }
    )
)
```

```
employee.copy(
    company = employee.company?.copy(
        addresses = employee.company.addresses.mapIndexed {
            index, address ->
            if (index != n) address
            else address.copy(
                street = address.street.copy(
                    name = address.street.name.toUpperCase()
                )
            )
        }
    )
)


Employee.company.addresses[n].street.name.modify(employee,
    String::toUpperCase)
```

Optics

Lens<S,A>

Definition

```
class Lens<S, A> {

    companion object {

        operator fun <S, A> invoke(
            get: (S) -> A,
            set: (A) -> (S) -> S
        ) : Lens<S, A>

    }

    fun modify(a: S, f: (A) -> A): S

}
```

Lens

```
data class Street(val number: Int, val name: String)
```

Lens

```
data class Street(val number: Int, val name: String)

val streetNameLens: Lens<Street, String>
```

Lens

```
data class Street(val number: Int, val name: String)

val streetNameLens: Lens<Street, String> = Lens(
    get = { it.name },
```

Lens

```
data class Street(val number: Int, val name: String)

val streetNameLens: Lens<Street, String> = Lens(
    get = { it.name },
    set = { name -> { street -> street.copy(name = name) } }
)
```

Lens

```
data class Street(val number: Int, val name: String)

val streetNameLens: Lens<Street, String> = Lens(
    get = { it.name },
    set = { name -> { street -> street.copy(name = name) } }
)

val street = Street(1600, "Amphitheatre Parkway")
// street = Street(number=1600, name=Amphitheatre Parkway)
```

Lens

```
data class Street(val number: Int, val name: String)

val streetNameLens: Lens<Street, String> = Lens(
    get = { it.name },
    set = { name -> { street -> street.copy(name = name) } }
)

val street = Street(1600, "Amphitheatre Parkway")
// street = Street(number=1600, name=Amphitheatre Parkway)

val street2 = streetNameLens.modify(street, String::toUpperCase)
// street2 = Street(number=1600, name=AMPHITHEATRE PARKWAY)
```

A new challenger approaches

Either<A,B>

Optional<S,A>

Optional

```
class Optional<S, A> {

    companion object {

        operator fun <S, A> invoke(
            getOrModify: (S) -> Either<S, A>,
            set: (A) -> (S) -> S
        ): Optional<S, A>
    }

    fun modify(a: S, f: (A) -> A): S

}
```

Optional

```
data class OptionString(val string: String?)
```

Optional

```
data class OptionString(val string: String?)

val optional: Optional<OptionString, String>
```

Optional

```
data class OptionString(val string: String?)

val optional: Optional<OptionString, String> = Optional(
  getOrModify = { it.string?.right() ?: it.left() },
```

Optional

```
data class OptionString(val string: String?)

val optional: Optional<OptionString, String> = Optional(
  getOrModify = { it.string?.right() ?: it.left() },
  set = { string -> { option -> option.copy(string = string) } }
)
```

Optional

```
data class OptionString(val string: String?)

val optional: Optional<OptionString, String> = Optional(
  getOrModify = { it.string?.right() ?: it.left() },
  set = { string -> { option -> option.copy(string = string) } }
)

val fullOption = OptionString("abc")
// fullOption = OptionString(string=abc)
```

Optional

```
data class OptionString(val string: String?)

val optional: Optional<OptionString, String> = Optional(
  getOrModify = { it.string?.right() ?: it.left() },
  set = { string -> { option -> option.copy(string = string) } }
)

val fullOption = OptionString("abc")
// fullOption = OptionString(string=abc)

val fullOption2 = optional.modify(fullOption, String::
    toUpperCase)
// fullOption2 = OptionString(string=ABC)
```

Optional

```
data class OptionString(val string: String?)

val optional: Optional<OptionString, String> = Optional(
  getOrModify = { it.string?.right() ?: it.left() },
  set = { string -> { option -> option.copy(string = string) } }
)

val fullOption = OptionString("abc")
// fullOption = OptionString(string=abc)

val fullOption2 = optional.modify(fullOption, String::
    toUpperCase)
// fullOption2 = OptionString(string=ABC)

val emptyOption = OptionString(null)
// emptyOption = OptionString(string=null)
```

Optional

```
data class OptionString(val string: String?)

val optional: Optional<OptionString, String> = Optional(
  getOrModify = { it.string?.right() ?: it.left() },
  set = { string -> { option -> option.copy(string = string) } }
)

val fullOption = OptionString("abc")
// fullOption = OptionString(string=abc)

val fullOption2 = optional.modify(fullOption, String::
    toUpperCase)
// fullOption2 = OptionString(string=ABC)

val emptyOption = OptionString(null)
// emptyOption = OptionString(string=null)

val emptyOption2 = optional.modify(emptyOption, String::
    toUpperCase)
// emptyOption2 = OptionString(string=null)
```

$$\text{Optic1<A,B>} + \text{Optic2<B, C>} = \text{Optic3<A, C>}$$

DSL

```
Employee.company.addresses[n].street.name.modify(employee,
    String::toUpperCase)
```

DSL

```
Employee.company .addresses [n] .street .name .modify(employee,
    String::toUpperCase)
```

DSL

```
Employee.company .addresses [n] .street .name .modify(employee,
    String::toUpperCase)
```

- `val Employee.Companion.company: Lens<Employee, Company>`

DSL

```
Employee.company .addresses [n] .street .name .modify(employee,
    String::toUpperCase)
```

- `val Employee.Companion.company: Lens<Employee, Company>`
- `val <S> Lens<S, Company>.addresses: Lens<S, List<Address>>`

DSL

```
Employee.company .addresses [n] .street .name .modify(employee,
    String::toUpperCase)
```

- `val Employee.Companion.company: Lens<Employee, Company>`
- `val <S> Lens<S, Company>.addresses: Lens<S, List<Address>>`
- `val <S, E> Lens<S, List<E>>.get(i: Int): Optional<S, E>`

DSL

```
Employee.company .addresses [n] .street .name .modify(employee,
    String::toUpperCase)
```

- `val Employee.Companion.company: Lens<Employee, Company>`
- `val <S> Lens<S, Company>.addresses: Lens<S, List<Address>>`
- `val <S, E> Lens<S, List<E>>.get(i: Int): Optional<S, E>`
- `val <S> Optional<S, Address>.street: Optional<S, Street>`

DSL

```
Employee.company .addresses [n] .street .name .modify(employee,
    String::toUpperCase)
```

- `val Employee.Companion.company: Lens<Employee, Company>`
- `val <S> Lens<S, Company>.addresses: Lens<S, List<Address>>`
- `val <S, E> Lens<S, List<E>>.get(i: Int): Optional<S, E>`
- `val <S> Optional<S, Address>.street: Optional<S, Street>`
- `val <S> Optional<S, Street>.name: Optional<S, String>`

DSL

```
Employee.company .addresses [n] .street .name .modify(employee,
    String::toUpperCase)
```

- ~~val Employee.Companion.company: Lens<Employee, Company>~~
- ~~val <S> Lens<S, Company>.addresses: Lens<S, List<Address>>~~
- val <S, E> Lens<S, List<E>>.get(i: Int): Optional<S, E>
- ~~val <S> Optional<S, Address>.street: Optional<S, Street>~~
- ~~val <S> Optional<S, Street>.name: Optional<S, String>~~

Annotation processor

```
@optics
data class Employee(val name: String, val company: Company) {
    companion object }
```

Annotation processor

```
@optics
data class Employee(val name: String, val company: Company) {
    companion object }

val Employee.Companion.name: Lens<Employee, String>
val <S> Lens<S, Employee>.name: Lens<S, String>
val <S> Optional<S, Employee>.name: Optional<S, String>

val Employee.Companion.company: Lens<Employee, Company>
val <S> Lens<S, Employee>.company: Lens<S, Company>
val <S> Optional<S, Employee>.company: Optional<S, Company>
```

Λrrow

arrow-kt.io

The end