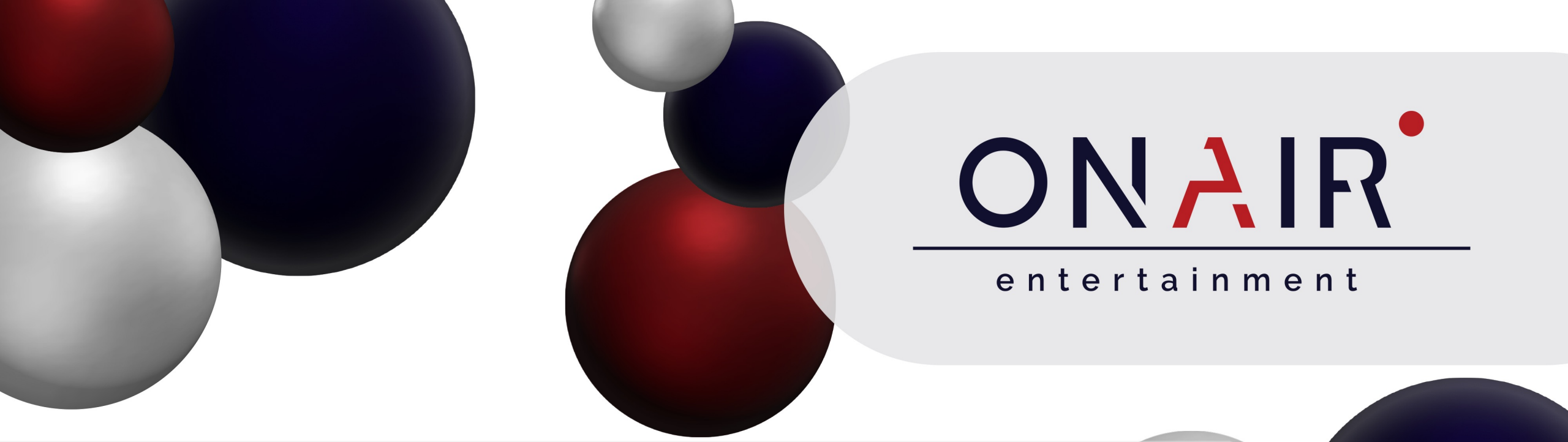

ONAIR entertainment

Gens & Lens

Making generators composable

by Aleksei Shamenev

Gens & Lens: ジェネレータを合成可能にする

Materials

発表資料です

Agenda

- Problem definition
- Tech. stack
- Composable gens

問題の提起、技術スタック
合成可能な gen

ONAIR entertainment

Problem definition

Bank accounts in Westeros

- Westeros
- Mobile bank service
- Lots of corner cases in its validation logic
- Using property-based tests for the logic

ONAIR entertainment

架空のウェスタロス大陸でモバイル・バンキングを作る
プロパティ・ベースでロジックをテストする

Validation rules

Imaginary world, imaginary regulations

- Lots of regions in the world
- Lots of different regulations for banks:
  1. Personal bank accounts are forbidden in **Westerlands**. Both fiat and crypto.
  2. **Crownlands** allows using cryptocurrencies only for business and don't for personal use.
  3. No cryptocurrencies are legal in **Stormlands**.
  4. And only **Free Cities** have no regulations for banks.

ONAIR entertainment

地域によって異なる銀行規制。岩の王国では個人口座は禁止。
王室領では、仮想通貨は商用のみ。

Domain

```
final case class Account private (
    accountType: AccountType,
    billAddress: Address,
    balance: Balance
) derives Eq

final case class Address(country: Country, city: City) derives Eq

final case class Balance(value: Long, currency: Currency) derives Eq

enum Currency derives Eq:
  case CoinCurr(fiat: Coins)
  case CryptoCurr(crypto: Crypto)
```

ドメイン

Validation rules in Scala

Imaginary world, imaginary regulations

```
object Account:
  def make(
      accountType: AccountType,
      billAddress: Address,
      balance: Balance
  ): Either[ValidationError, Account] =
    if (
      billAddress.country === Country.Westerlands &&
      accountType === AccountType.Personal
    ) Left(ValidationError.PersonalAccountsForbidden(billAddress.country))
    else if (…) // omitted
    ) Left(…) // omitted
    else if (
      billAddress.country === Country.Stormlands &&
      Currency.isCrypto(balance.currency)
    ) Left(ValidationError.CryptoForbidden(billAddress.country))
    else
      Right(Account(accountType, billAddress, balance))
```

ONAIR entertainment

Scala での検証ルール

Tech. stack

- Scala 3
- ScalaCheck
- Monocle

ONAIR entertainment

ScalaCheck

- Property-based tests
- Gen[T]

プロパティ・ベース・テスト

ONAIR entertainment

Monocle

Access to and transformation of immutable structures

- Lens
- Optional

ONAIR entertainment

不変データ構造のアクセスを補助するライブラリ

Manual composition

Lots of boilerplate code with .copy(..)

```
property("Crypto is forbidden in Stormlands") {
  forAll { (accountType: AccountType, address: Address, balance: Balance, anyCrypto: Crypto) =>
    val stormlands    = address.copy(country = Country.Stormlands)
    val cryptoCurr    = Currency.CryptoCurr(anyCrypto)
    val cryptoBalance = balance.copy(currency = cryptoCurr)
    val result        = Account.make(accountType, stormlands, cryptoBalance)

    (result === Left(ValidationError.CryptoForbidden(Country.Stormlands))) :| s"result = $result"
  }
}
```

ONAIR entertainment

.copy(...) を何度も手で書くのが面倒

Using Monocle

Optics definition

```
object DomainLens:
  val AddressCountryLens: Lens[Address, Country]    = GenLens[Address](_.country)
  val AddressCityLens: Lens[Address, City]          = GenLens[Address](_.city)
  val BalanceCurrencyLens: Lens[Balance, Currency] = GenLens[Balance](_.currency)
  val CurrencyCryptoOptional: Optional[Currency, Crypto] = ??? // omitted
  val BalanceCryptoOptional: Optional[Balance, Crypto] =
    BalanceCurrencyLens.andThen(CurrencyCryptoOptional)
```

ONAIR entertainment

Monocle を使うときれいに書ける

Using Monocle

DSL

```
object LensDSL:
  final case class BySetterStep[F[_], A, B](fa: F[A], setter: Setter[A, B]):
    infix def byF(fb: F[B])(using Monad[F]): F[A] = for {
      a <- fa
      b <- fb
    } yield setter.replace(b)(a)
    infix def by(b: B)(using Monad[F]): F[A] = for {
      a <- fa
    } yield setter.replace(b)(a)
```

ONAIR entertainment

Using Monocle

Test case

```
property("Crypto is forbidden in Stormlands") {
  val patchedAddressGen = addressGen replace AddressCountryLens by Country.Stormlands
  val patchedBalanceGen = balanceGen replace BalanceCryptoOptional byF cryptoGen

  forAll(accountTypeGen, patchedAddressGen, patchedBalanceGen) { (accountType, address, balance) =>
    val result = Account.make(accountType, address, balance)
    (result === Left(ValidationError.CryptoForbidden(Country.Stormlands))) :| s"result = $result"
  }
}
```

ONAIR entertainment

Implicit optics

Define optics using `given`

```
object GivenDomainLens:
  given Lens[Address, Country]     = DomainLens.AddressCountryLens
  given Lens[Address, City]        = DomainLens.AddressCityLens
  given Lens[Balance, Currency]    = DomainLens.BalanceCurrencyLens
  given Optional[Currency, Crypto] = DomainLens.CurrencyCryptoOptional
  given Optional[Balance, Crypto]  = DomainLens.BalanceCryptoOptional
```

ONAIR entertainment

given を使ったオプティクスの定義

Implicit optics

DSL

```
object GivenLensDSL:
  extension [F[_], A](fa: F[A])
    infix def by[B](b: B)(using setter: Setter[A, B], m: Monad[F]): F[A] =
      for a <- fa
    yield setter.replace(b)(a)

    infix def byF[B](fb: F[B])(using opti: Optional[A, B], m: Monad[F]): F[A] = for
      a <- fa
      b <- fb
    yield opti.replace(b)(a)
```

ONAIR entertainment

Implcit optics

Test case

```
property("Crypto is forbidden in Stormlands") {
  val patchedAddressGen = addressGen by Country.Stormlands
  val patchedBalanceGen = balanceGen byF cryptoGen

  forAll(accountTypeGen, patchedAddressGen, patchedBalanceGen) { (accountType, address, balance) =>
    val result = Account.make(accountType, address, balance)
    (result === Left(ValidationError.CryptoForbidden(Country.Stormlands))) :| s"result = $result"
  }
}
```

ONAIR entertainment

Auto Derivation

Further work

- Optics autoderivation
- No need to explicitly define optics
- More requirements to the structures
  - Unique types for fields within the structure of cases classes
  - Or explicit markup of fields

ONAIR entertainment

今後の課題はオプティクスの自動導出

ONAIR
entertainment

The End

ご清聴ありがとうございました