applicative.lhs

```
> {-# LANGUAGE InstanceSigs #-}

> module Applicative where
```

Класс `Applicative`

Класс

(source)

```
class Functor f => Applicative f where
  pure  :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
```

Законы

1. identity

```
pure id <*> v == v
```

2. composition

```
pure (.) <*> u <*> v <*> w == u <*> (v <*> w)
```

3. homomorphism

```
pure f <*> pure x = pure (f x)
```

4. interchange

```
u <*> pure y == pure ($ y) <*> u
```

Пара инстансов для примера

```
instance Applicative Maybe where
  pure = Just
  Just f <*> Just x = Just (f x)
  _ <*> _ = Nothing

instance Applicative [] where
  pure x = [x]
  fs <*> xs = concat (map (\f -> map f xs) fs)
```

Пример: DSL для валидации

```
> data ValidationResult a
>   = Errors [String] -- список ошибок
>   | Ok a [String]   -- значение и список предупреждений
>   deriving (Show)

> newtype Validation src dest = Validation
>   { validate :: src -> ValidationResult dest
>   }
```

Инстансы

```
> instance Semigroup (Validation a b) where
>   (Validation v1) <> (Validation v2) =
>     Validation $ \x -> case (v1 x, v2 x) of
>       (Errors e1, Errors e2) -> Errors (e1 ++ e2)
>       (Ok _   w1, Ok v   w2) -> Ok v   (w1 ++ w2)
>       (Errors e1, _)         -> Errors e1
>       (_        , Errors e2) -> Errors e2

> instance Functor (Validation s) where
>   fmap :: (a -> b) -> Validation c a -> Validation c b
>   fmap f (Validation g) = Validation $ \x -> case g x of
>     Errors es -> Errors es
>     Ok v ws   -> Ok (f v) ws
```

```
> instance Applicative (Validation s) where
>   pure :: a -> Validation b a
>   pure x = Validation $ \_ -> Ok x []

>   (<*>) :: Validation c (a -> b) -> Validation c a -> Validation c b
>   Validation vf <*> Validation vx = Validation $ \x ->
>     case (vf x, vx x) of
>       (Errors e1, Errors e2) -> Errors (e1 ++ e2)
>       (Ok f   w1, Ok v   w2) -> Ok (f v) (w1 ++ w2)
>       (Errors e1, _)         -> Errors e1
>       (_        , Errors e2) -> Errors e2
```

Примитивы и комбинаторы

```
> check :: String -> (a -> Bool) -> Validation a a
> check err test = Validation $ \x ->
>   if test x
>   then Ok x []
>   else Errors [err]

> note :: String -> (a -> Bool) -> Validation a a
> note warning test = Validation $ \x ->
>   Ok x $ if test x
>          then []
>          else [warning]

> field :: (a -> b) -> Validation b c -> Validation a c
> field f (Validation g) = Validation $ \x -> g (f x)
```

Пример использования

```
> data User = User
>   { userName :: String
>   , userAge  :: Int
>   , userPet  :: Pet
>   }
>   deriving (Show)

> data Pet = Pet
>   { petName :: String
>   }
>   deriving (Show)

> ageV :: Validation Int Int
> ageV =
>   check "Negative age!" (> 0)
>   <> note "Maybe too young!" (> 20)
>   <> note "Maybe too old!" (< 80)

> nameV :: Validation String String
> nameV = check "Empty name!" (not . null)

> userV :: Validation User User
> userV = User
>   <$> field userName nameV
>   <*> field userAge ageV
>   <*> field userPet (Pet
>      <$> field petName nameV
>      )
```