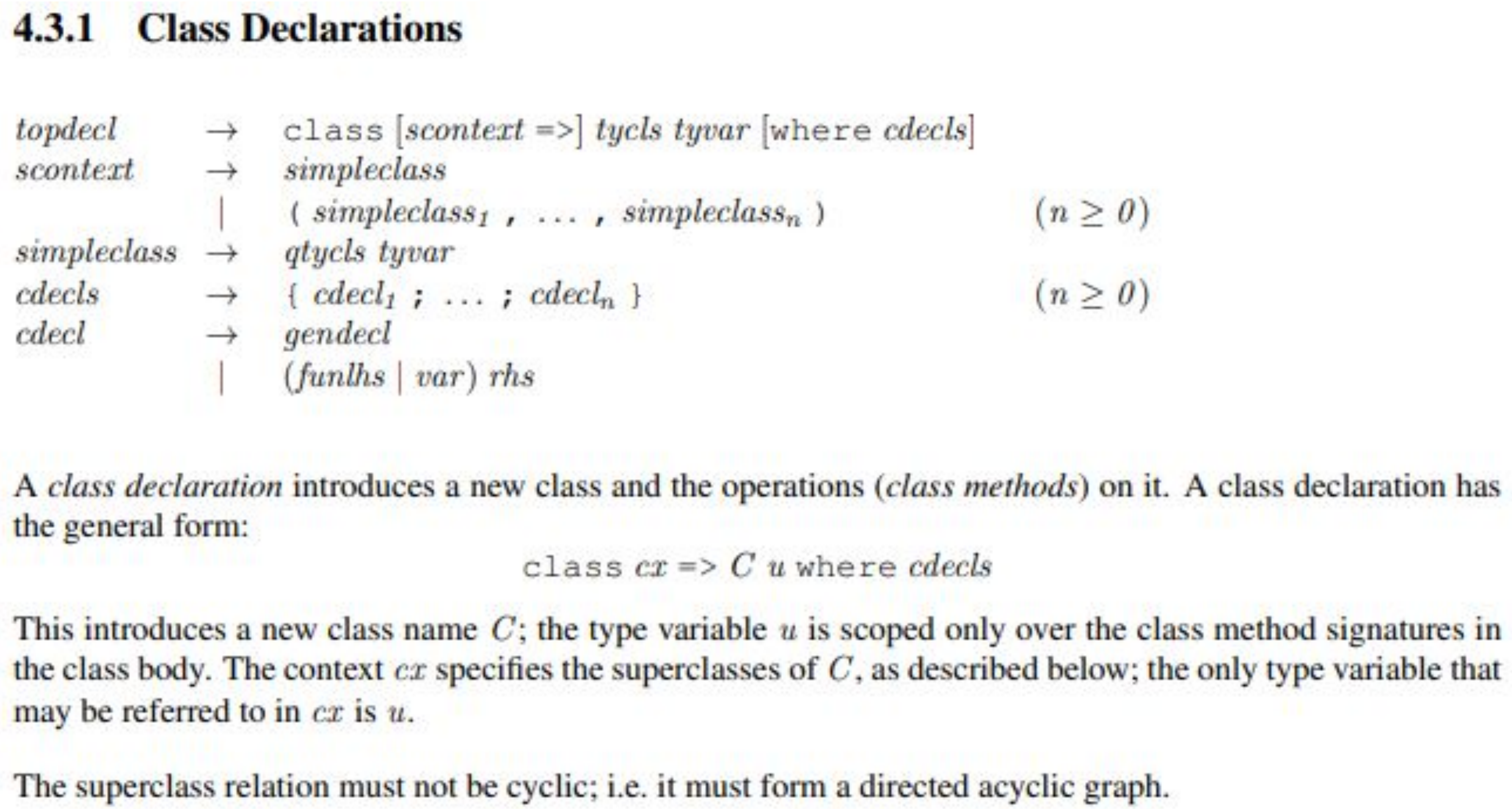
Функційне програмування мовою Haskell

Класи типів

Класи типів

4.3 Type Classes and Overloading

https://www.haskell.org/definition/haskell2010.pdf



4.3.1 Class Declarations

topdecl	→	`class` [*scontext* `=>`] *tycls tyvar* [`where` *cdecls*]	
scontext	→	*simpleclass*	
	|	`(` $simpleclass_1$ `,` … `,` $simpleclass_n$ `)`	$(n \geq 0)$
simpleclass	→	*qtycls tyvar*	
cdecls	→	`{` $cdecl_1$ `;` … `;` $cdecl_n$ `}`	$(n \geq 0)$
cdecl	→	*gendecl*	
	|	(*funlhs* | *var*) *rhs*	

A *class declaration* introduces a new class and the operations (*class methods*) on it. A class declaration has the general form:

$$\texttt{class}\ cx\ \texttt{=>}\ C\ u\ \texttt{where}\ cdecls$$

This introduces a new class name C; the type variable u is scoped only over the class method signatures in the class body. The context cx specifies the superclasses of C, as described below; the only type variable that may be referred to in cx is u.

The superclass relation must not be cyclic; i.e. it must form a directed acyclic graph.

Оголошення класів

4.3.1 Class Declarations

https://www.haskell.org/definition/haskell2010.pdf   стор. 40/60

class [*context* **=>**] *Class_name type_var* [**where** cdecls]

Декларація класу представляє *новий клас* і операції (методи класу).

Визначаються:

- сигнатури операцій (методів)
- infix – оголошення
- означення методів-за-замо́вчуванням

Оголошення класів

4.3.1 Class Declarations

https://www.haskell.org/definition/haskell2010.pdf стор. 40/60

class [*context* **=>**] *Class_name type_var* [**where** cdecls]

class cx **=>** *C u* **where** cdecls

Декларація класу представляє *новий клас* і операції (методи класу). Визначаються:

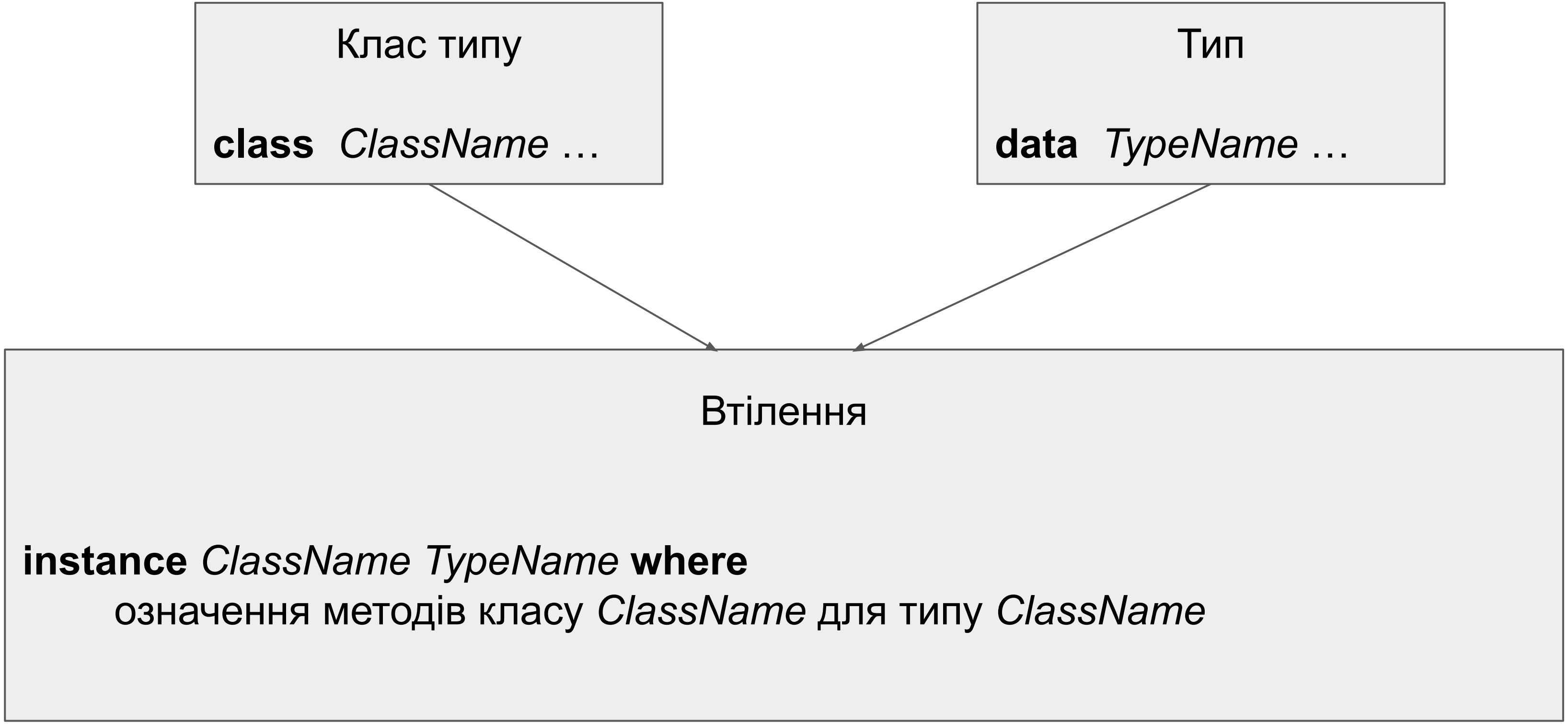
- сигнатури операцій (методів)
- infix – оголошення
- означення методів-за-замо́вчуванням

```
Prelude> :i Num

class Num a where
  (+) :: a -> a -> a
  (-) :: a -> a -> a
  (*) :: a -> a -> a
  negate :: a -> a
  abs :: a -> a
  signum :: a -> a
  fromInteger :: Integer -> a
  {-# MINIMAL (+), (*), abs, signum, fromInteger, (negate | (-)) #-}
        -- Defined in `GHC.Num'
instance Num Word -- Defined in `GHC.Num'
instance Num Integer -- Defined in `GHC.Num'
instance Num Int -- Defined in `GHC.Num'
instance Num Float -- Defined in `GHC.Float'
instance Num Double -- Defined in `GHC.Float'
```

Оголошення втілень (екземплярів) класу

4.3.2 Instance Declarations



Клас типу

class *ClassName* …

Тип

data *TypeName* …

Втілення

instance *ClassName* *TypeName* **where**

означення методів класу *ClassName* для типу *ClassName*

Оголошення втілень (екземплярів) класу

```
-- декларація класу
class MyClass a where
  eq   :: a -> a -> Bool
  mul :: a -> a -> Int

-- декларація типу
newtype MyInt = ConMyInt Int

-- втілення (класу в типі)
instance MyClass MyInt where
  (ConMyInt x) `eq` (ConMyInt y) = abs(x-y) < 3
  (ConMyInt 2) `mul` (ConMyInt 2) = 5
  (ConMyInt i) `mul` (ConMyInt j) = i * j
```

Оголошення втілень (екземплярів) класу

```
> ConMyInt 2 `mul` ConMyInt 3
6

> :t ConMyInt 2 `mul` ConMyInt 3
ConMyInt 2 `mul` ConMyInt 3 :: Int

> ConMyInt 3 `mul` ConMyInt 3
9

> ConMyInt 3 `mul` ConMyInt 2
6

> ConMyInt 2 `mul` ConMyInt 2
5
```

Оголошення втілень (екземплярів) класу

```
> ConMyInt 2 `eq` ConMyInt 2
True

> ConMyInt 2 `eq` ConMyInt 3
True

> ConMyInt 2 `eq` ConMyInt 4
True

> ConMyInt 2 `eq` ConMyInt 5
False
```

Оголошення втілень (екземплярів) класу

```
> ConMyInt 2

<interactive>:12:1: error:
    * No instance for (Show MyInt) arising from a use of `print'
    * In a stmt of an interactive GHCi command: print it
```

Оголошення втілень (екземплярів) класу

```
> ConMyInt 2

<interactive>:12:1: error:
    * No instance for (Show MyInt) arising from a use of `print'
    * In a stmt of an interactive GHCi command: print it
```

```
newtype MyInt = ConMyInt Int
  deriving Show
```

Оголошення втілень (екземплярів) класу

```
> ConMyInt 2

<interactive>:12:1: error:
    * No instance for (Show MyInt) arising from a use of `print'
    * In a stmt of an interactive GHCi command: print it

newtype MyInt = ConMyInt Int
  deriving Show

> ConMyInt 2
ConMyInt 2
```

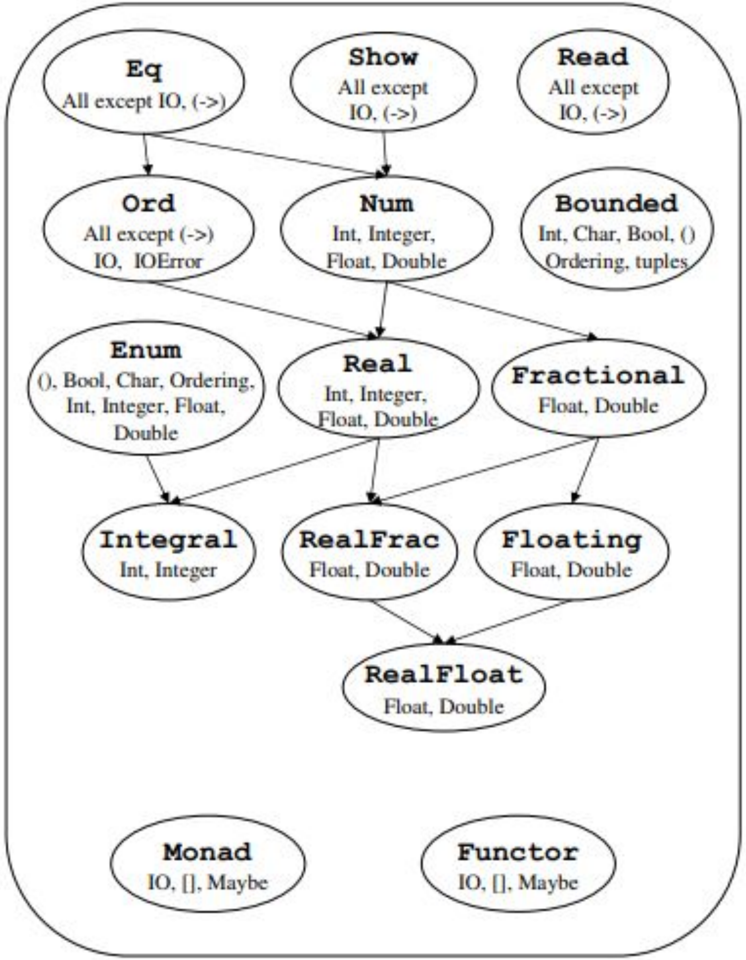


Figure 6.1: Standard Haskell Classes

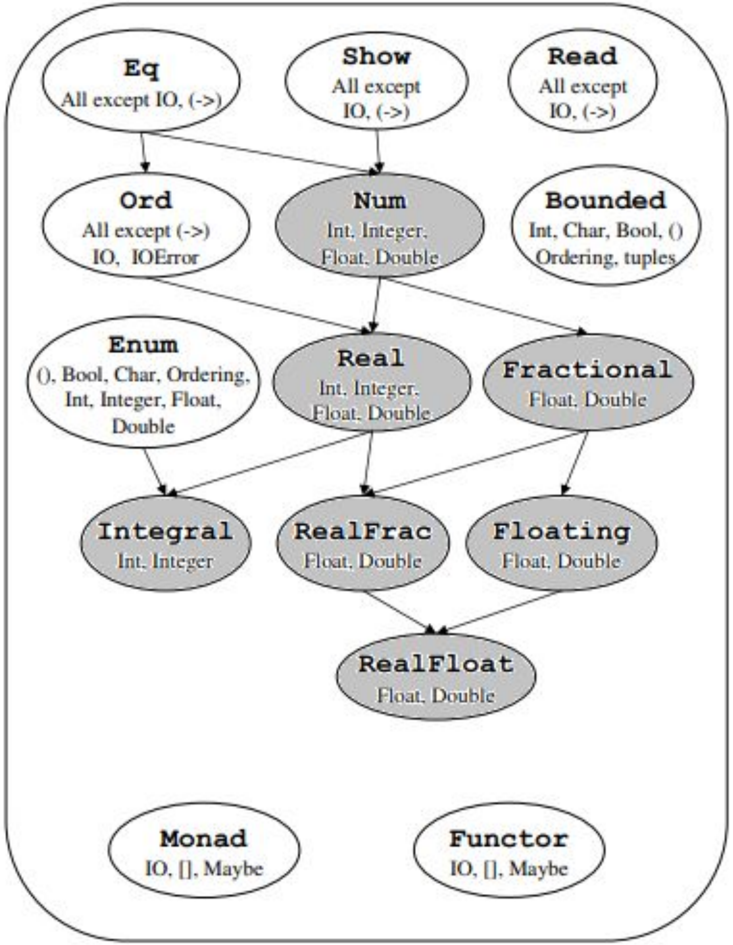


Figure 6.1: Standard Haskell Classes

```
Prelude> :i Num

class Num a where
  (+) :: a -> a -> a
  (-) :: a -> a -> a
  (*) :: a -> a -> a
  negate :: a -> a
  abs :: a -> a
  signum :: a -> a
  fromInteger :: Integer -> a
  {-# MINIMAL (+), (*), abs, signum, fromInteger, (negate | (-))
#-}
        -- Defined in `GHC.Num'
instance Num Word -- Defined in `GHC.Num'
instance Num Integer -- Defined in `GHC.Num'
instance Num Int -- Defined in `GHC.Num'
instance Num Float -- Defined in `GHC.Float'
instance Num Double -- Defined in `GHC.Float'
```

```
Prelude> :i Real

class (Num a, Ord a) => Real a where
  toRational :: a -> Rational
  {-# MINIMAL toRational #-}
        -- Defined in `GHC.Real'
instance Real Word -- Defined in `GHC.Real'
instance Real Integer -- Defined in `GHC.Real'
instance Real Int -- Defined in `GHC.Real'
instance Real Float -- Defined in `GHC.Float'
instance Real Double -- Defined in `GHC.Float'
```

```
Prelude> :i Floating

class Fractional a => Floating a where
  pi :: a
  exp :: a -> a
  log :: a -> a
  sqrt :: a -> a
  (**) :: a -> a -> a
  logBase :: a -> a -> a
  sin :: a -> a
  cos :: a -> a
  tan :: a -> a
  asin :: a -> a
  acos :: a -> a
  atan :: a -> a
  sinh :: a -> a
  cosh :: a -> a
  tanh :: a -> a
  asinh :: a -> a
  acosh :: a -> a
  atanh :: a -> a
  GHC.Float.log1p :: a -> a
  GHC.Float.expm1 :: a -> a
  GHC.Float.log1pexp :: a -> a
  GHC.Float.log1mexp :: a -> a
  {-# MINIMAL pi, exp, log, sin, cos, asin, acos, atan, sinh, cosh,
              asinh, acosh, atanh #-}
        -- Defined in `GHC.Float'
instance Floating Float -- Defined in `GHC.Float'
instance Floating Double -- Defined in `GHC.Float'
```

```
> :i Real

class (Num a, Ord a) => Real a where
  toRational :: a -> Rational
  {-# MINIMAL toRational #-}
        -- Defined in `GHC.Real'
instance Real Word -- Defined in `GHC.Real'
instance Real Integer -- Defined in `GHC.Real'
instance Real Int -- Defined in `GHC.Real'
instance Real Float -- Defined in `GHC.Float'
instance Real Double -- Defined in `GHC.Float'
```

```
> toRational 24
24 % 1

> toRational 24/18
4 % 3

> toRational 1.25
5 % 4

> :t it
it :: Rational
```

```
Prelude> :i Fractional

class Num a => Fractional a where
  (/) :: a -> a -> a
  recip :: a -> a
  fromRational :: Rational -> a
  {-# MINIMAL fromRational, (recip | (/)) #-}
        -- Defined in `GHC.Real'
instance Fractional Float -- Defined in `GHC.Float'
instance Fractional Double -- Defined in `GHC.Float'
```

```
> :i Fractional

class Num a => Fractional a where
  (/) :: a -> a -> a
  recip :: a -> a
  fromRational :: Rational -> a
  {-# MINIMAL fromRational, (recip | (/)) #-}
        -- Defined in `GHC.Real'
instance Fractional Float -- Defined in `GHC.Float'
instance Fractional Double -- Defined in `GHC.Float'


> toRational 1.25
5 % 4

> :t it
it :: Rational

> fromRational $ toRational 1.25
1.25

> :t it
it :: Fractional a => a
```

```
> :i Fractional

class Num a => Fractional a where
  (/) :: a -> a -> a
  recip :: a -> a
  fromRational :: Rational -> a
  {-# MINIMAL fromRational, (recip | (/)) #-}
        -- Defined in `GHC.Real'
instance Fractional Float -- Defined in `GHC.Float'
instance Fractional Double -- Defined in `GHC.Float'


> toRational 1.25
5 % 4

> :t it
it :: Rational

> fromRational $ toRational 1.25
1.25

> :t it
it :: Fractional a => a
```

```
> recip 2
0.5

> :t it
it :: Fractional a => a
```

```
Prelude> :i RealFrac

class (Real a, Fractional a) => RealFrac a where
  properFraction :: Integral b => a -> (b, a)
  truncate :: Integral b => a -> b
  round :: Integral b => a -> b
  ceiling :: Integral b => a -> b
  floor :: Integral b => a -> b
  {-# MINIMAL properFraction #-}
        -- Defined in `GHC.Real'
instance RealFrac Float -- Defined in `GHC.Float'
instance RealFrac Double -- Defined in `GHC.Float'
```

```
> :i RealFrac

class (Real a, Fractional a) => RealFrac a where
  properFraction :: Integral b => a -> (b, a)
  truncate :: Integral b => a -> b
  round :: Integral b => a -> b
  ceiling :: Integral b => a -> b
  floor :: Integral b => a -> b
  {-# MINIMAL properFraction #-}
        -- Defined in `GHC.Real'
instance RealFrac Float -- Defined in `GHC.Float'
instance RealFrac Double -- Defined in `GHC.Float'
```

```
properFraction :: Integral b => a -> (b, a)Source#
```

The function properFraction takes a real fractional number x and returns a pair (n,f) such that x = n+f, and:

- n is an integral number with the same sign as x; and
- f is a fraction with the same type and sign as x, and with absolute value less than 1.

The default definitions of the ceiling, floor, truncate and round functions are in terms of properFraction.

```
Prelude> :i Integral

class (Real a, Enum a) => Integral a where
  quot :: a -> a -> a
  rem :: a -> a -> a
  div :: a -> a -> a
  mod :: a -> a -> a
  quotRem :: a -> a -> (a, a)
  divMod :: a -> a -> (a, a)
  toInteger :: a -> Integer
  {-# MINIMAL quotRem, toInteger #-}
        -- Defined in `GHC.Real'
instance Integral Word -- Defined in `GHC.Real'
instance Integral Integer -- Defined in `GHC.Real'
instance Integral Int -- Defined in `GHC.Real'
```

Integral numbers, supporting integer division

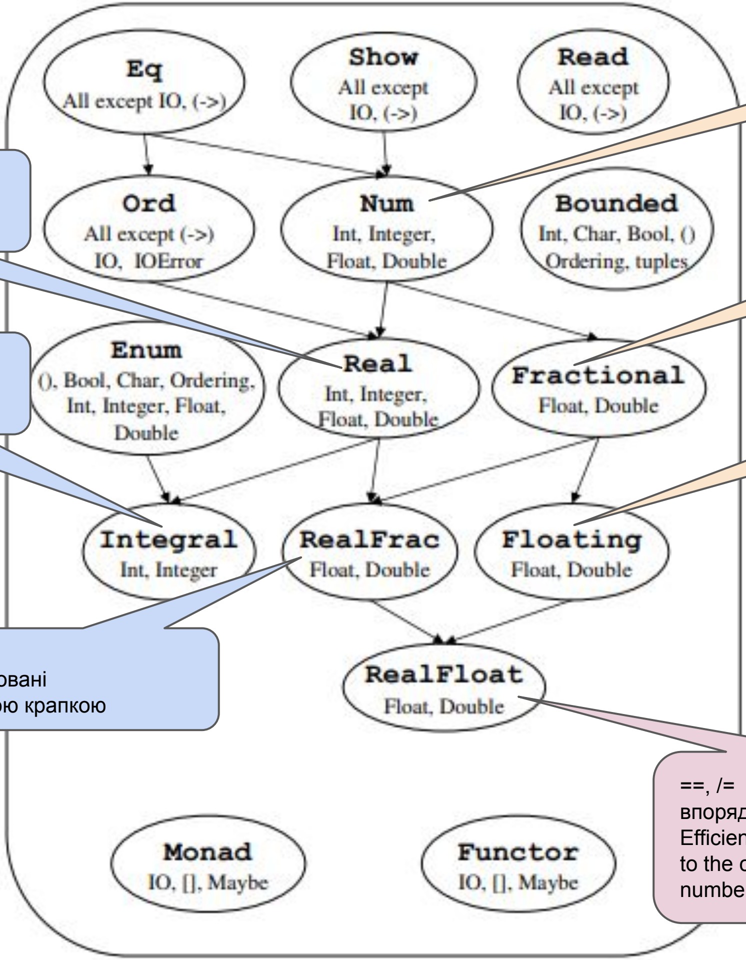


Figure 6.1: Standard Haskell Classes

==, /=
крім ділення
fromInteger :: Integer -> a

==, /=
впорядковані
toRational :: a -> Rational

==, /=
з діленням
fromRational :: Rational -> a

==, /=
впорядковані
перелічні
цілі

==, /=
з рухомою крапкою
у т.ч. комплексні

==, /=
впорядковані
з рухомою крапкою

==, /=
впорядковані
Efficient, machine-independent access to the components of a floating-point number.

Infix-оголошення

4.4.2 Fixity Declarations

infix[l|r] пріоритет ім’я_оператора

infix - неасоціативний
infix**l**- лівоасоціативний
infix**r**- правоасоціативний

пріоритет 0-9

Infix-оголошення

4.4.2 Fixity Declarations

infixr 6 |-|

(|-|) :: Num a => a->a->a

(|-|) x y = x-y

Infix-оголошення

4.4.2 Fixity Declarations

```
infixr 6 |-|
(|-|) :: Num a => a->a->a
(|-|) x y = x-y
```

```
> 3 |-| 1 |-| 1
3
```

Infix-оголошення

4.4.2 Fixity Declarations

```
infixr 6 |-|
(|-|) :: Num a => a->a->a
(|-|) x y = x-y
```

```
> 3 |-| 1 |-| 1
3
```

```
infix 5 ~=
a ~= b = a-b<h && b-a<h
     where h=0.001
```

Infix-оголошення

4.4.2 Fixity Declarations

```
infixr 6 |-|
(|-|) :: Num a => a->a->a
(|-|) x y = x-y
```

```
> 3 |-| 1 |-| 1
3
```

```
infix 5 ~=
a ~= b = a-b<h && b-a<h
    where h=0.001
```

```
> 1 ~= 1
True

> 1 ~= 2
False

> 1 ~= 1.0005
True

> 1 ~= 1.005
False

> 1 ~= 1 ~= 1    ?
```

Infix-оголошення

4.4.2 Fixity Declarations

```
infixr 6 |-|
(|-|) :: Num a => a->a->a
(|-|) x y = x-y
```

```
> 3 |-| 1 |-| 1
3
```

```
infix 5 ~=
a ~= b = a-b<h && b-a<h
      where h=0.001
```

```
> 1 ~= 1 ~= 1

<interactive>:45:1: error:
    Precedence parsing error
        cannot mix `~=' [infix 5] and `~=' [infix 5] in the same infix expression
```