Classes

João Marcelo Uchôa de Alencar

Universidade Federal do Ceará - Quixadá

11 de Maio de 2017

Classes

- As classes representam um conceito de hierarquia de tipos e métodos que são particulares a um conjunto de funções.
- Os objetos da hierarquia são funções.
- Sub-classe, superclasse, herança, polimorfismo, etc.
- Um conjunto de funções que apresentam propriedades comuns definem uma classe.

Polimorfismo

- As listas [1,2,3], ['h', 'e', 'l', 'l', 'o'] e [[2], [4], [6]] são membros de uma mesma família, apesar do tipo de cada elemento ser diferente.
- Posso usar uma mesma variável para representar qualquer tipo de lista: (a:x).

```
length [] = 0
length (a:x) = 1 + length x
```

Qual é o tipo da função *length*?

Polimorfismo *ad hoc*

```haskell
soma_lista :: [Int] -> Int
soma_lista [] = 0
soma_lista (x:xs) = x + soma_lista xs
```

A função *soma_lista* não é polimórfica. Aceita apenas uma lista de inteiros.

- A função de adição (+) funciona entre inteiros e reais (polimórfica), mas não entre caracteres (não polimórfica).
- A função de adição (+) tem polimorfismo restrito, ou polimorfismo *ad hoc*.
- A definição de uma classe em Haskell define quais funções podem ser aplicadas em um tipo.
- Uma classe estabelece um conjunto de assinaturas de funções cujos tipos de membros dessa classe são chamados de instâncias.

Polimorfismo *ad hoc*

```
class Eq a where
(==), (/=) :: a -> a -> Bool
x /= y = not (x == y)
```

- A classe Eq é o conjunto de tipos em que os operadores $==$ e $/=$ são definidos.
- Logo, posso criar instâncias da classe Eq como os tipos Int, [Char], String, etc.

Polimorfismo *ad hoc*

```
data NomeCompleto = Nome String
(Nome "Marcio") == (Nome "Marcio") -- dá certo?
iguala :: NomeCompleto -> NomeCompleto -> Bool
iguala (Nome a) (Nome b) = a == b
iguala _ _ = False
-- Com a função iguala definida,
-- posso fazer que NomeCompleto faça
-- parte da classe Eq.
instance Eq NomeCompleto where
(==) = iguala
(Nome "Marcio") == (Nome "Marcio") -- dá certo?
```

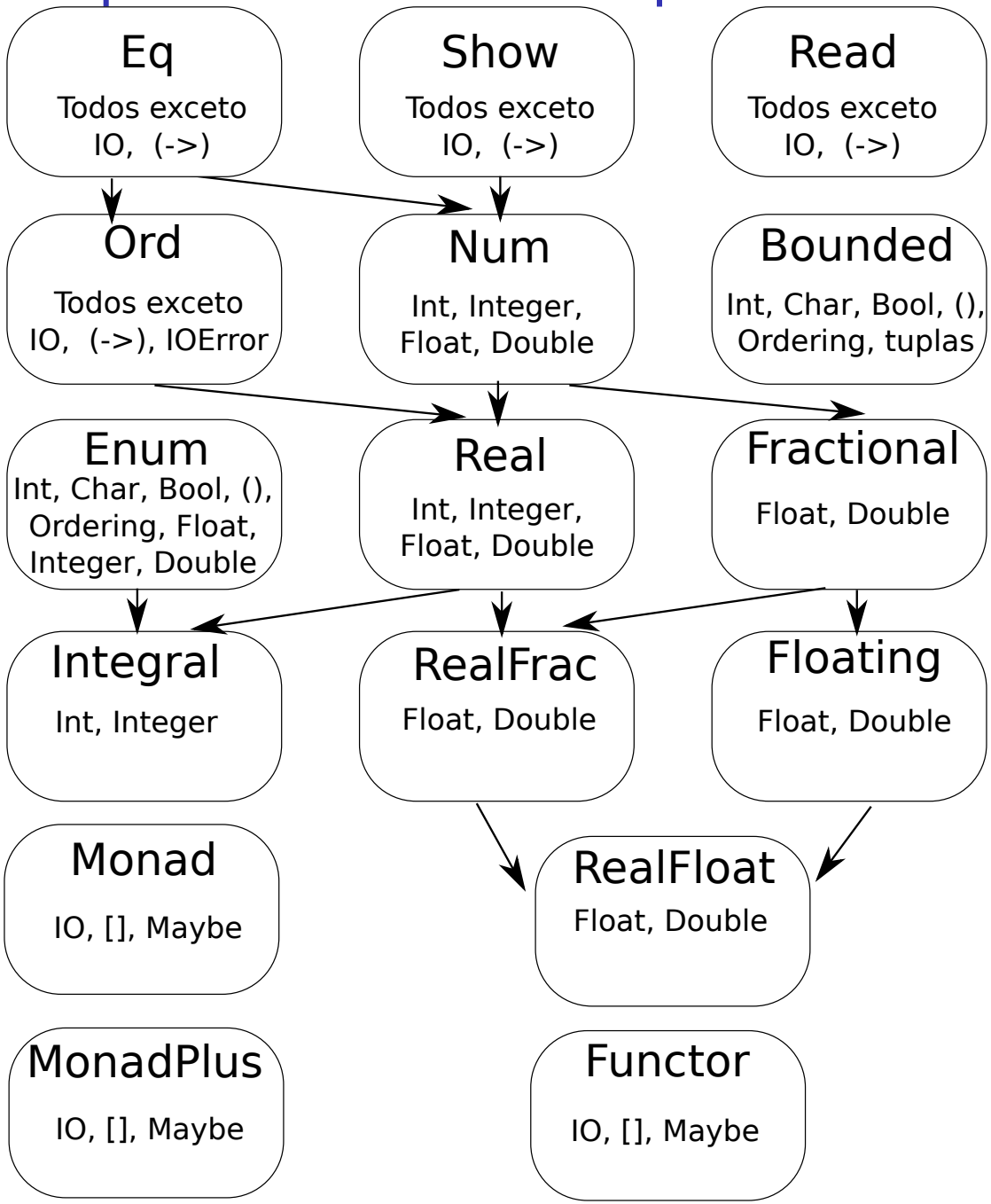
- Outra maneira de incluir um tipo na classe é indicar derivação na definição do tipo.
- Basta colocar *deriving(Eq)* ao final da definição do tipo.
- No caso de tipos estruturados simples, as comparações padrões podem ser inferidas.

Classes Derivadas

```
class (Eq a) => Ord a where
(<), (<=), (>=), (>) :: a -> a -> Bool
max, min :: a -> a -> a
```

- Aqui, a classe Ord é subclasse de Eq. Todos os tipos que pertencem a Ord também podem ser usados nas funções definidas por Eq.
- Ord implementa outras funções. Então nem todo tipo que serve para as funções de Eq serve também para as funções Ord.
- Exemplos de tipos instâncias de Ord: Integer, Int, Char, Array (vamos ver), Bool e Tuplas.
- Haskell também suporta herança múltipla através da palavra reserva *Multi*.

Hierarquia de Classes e Tipos



Classe Enum

```
class (Ord a) => Enum a where
    enumFrom :: a -> [a] -- [n..]
enumFromThen :: a -> a -> [a] -- [n, m..]
enumFromTo :: a -> a -> [a] -- [n..m]
enumFromThenTo :: a -> a -> a -> [a] --[n, n' ..m]
```

- Representa os tipos cujos elementos tem sucessores e predecessores.
- A funções começando em *enum* permitem a definição de listas através de intervalos.

Classe Show

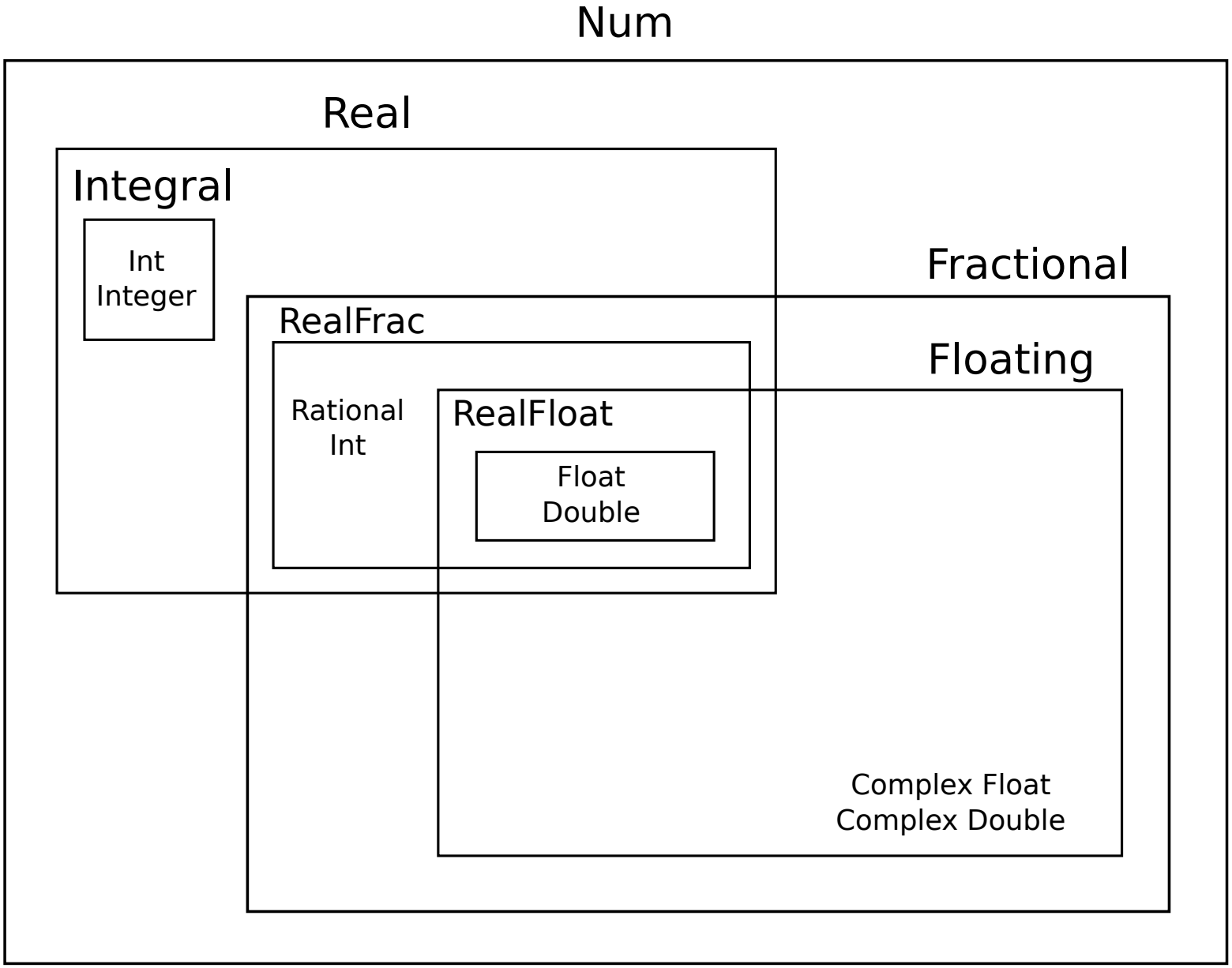
```
instance (Show a) => Show (Talvez a) where
    show Nada = "Nada"
show (Somente x) = "(Somente " ++ (show x) ++ ")"
```

- Representa os tipos básicos ou pré-definidos que podem ser exibidos na tela na forma de texto.

Classe Num

- A classe Num opera com os tipos que são aceitos pelas funções (+), (*) e (-).
- (+) :: Num a => $a- > a- > a$
- O símbolo => indica o contexto à sua esquerda, ou seja, a restrição do tipo dos parâmetros.
- Qual o tipo da função f x y z = $(x + 1, y + +z)$?

Classe Num


Num
Real
Integral
Int
Integer
Fractional
RealFrac
Floating
Rational
Int
RealFloat
Float
Double
Complex Float
Complex Double

Classe Num

```
even 9.8876 -- dá certo?
mod 8.987 3 -- dá certo?

f_um_int :: Int
f_um_int = 77
f_dois_integer :: Integer
f_dois_integer = 99
(/) 789 f_um_int
(/) 789 f_dois_integer
(/) 789 77
```

- Quando se fixa o tipo na definição, não há dedução (coerção) implícita.
- Com o valor literal, o compilador faz *coerção implícita*.

Classe Num

Função	Descrição da classe da função
fromInteger	Num a => Integer -> a
fromIntegral	(Integral a, Num b) => a -> b
fromRational	Fractional a => Rational -> a
toInteger	Integral a => a -> Integer
toRational	Real a => a -> Rational