

CS314. Функциональное программирование

Лекция 9. Классы типов

В. Н. Брагилевский

Направление «Фундаментальная информатика и информационные технологии»
Институт математики, механики и компьютерных наук имени И. И. Воровича
Южный федеральный университет

8 октября 2016 г.

Содержание

- 1 Стандартные классы типов
- 2 Примеры собственных классов типов

Содержание

- 1 Стандартные классы типов
 - Основные понятия
 - Класс Show и его экземпляры
- 2 Примеры собственных классов типов

Содержание

- 1 Стандартные классы типов
 - Основные понятия
 - Класс Show и его экземпляры
- 2 Примеры собственных классов типов

Основные понятия

- Тип определяет множество возможных значений.
- Класс типов содержит сигнатуры функций, применимых к значениям некоторого типа (интерфейс).
- Экземпляр класса типов — это реализация функций из класса типов для конкретного типа.
- Говорят, что тип имеет экземпляр некоторого класса, или что тип является частью класса.

Базовый синтаксис

Объявление класса

```
class ИмяКласса типоваяПеременная where  
  сигнатуры функций  
  реализации по умолчанию
```

Определение экземпляра

```
instance ИмяКласса Тип where  
  реализации функций
```

Пример: класс Eq

Класс Eq определяет операции проверки на равенство (==) и неравенство (/=) для значений типа.

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

- Имена и типы функций
- Реализация по умолчанию
- Взаимно рекурсивная реализация (достаточно реализовать только одну из двух операций)

Экземпляр класса Eq для собственного типа

```
data TrafficLight = Red | Yellow | Green
```

```
instance Eq TrafficLight where
```

```
  Red == Red = True
```

```
  Green == Green = True
```

```
  Yellow == Yellow = True
```

```
  _ == _ = False
```

В точности такой же экземпляр можно породить автоматически с использованием механизма deriving:

```
data TrafficLight = Red | Yellow | Green
  deriving (Eq)
```

```
ghci> Red == Red
True
```

```
ghci> Red == Yellow
False
```

```
ghci> Red `elem` [Red, Yellow, Green]
True
```


Экземпляр класса Eq для Maybe a

```
instance Eq a => Eq (Maybe a) where
  Just x == Just y = x == y
  Nothing == Nothing = True
  _ == _ = False
```

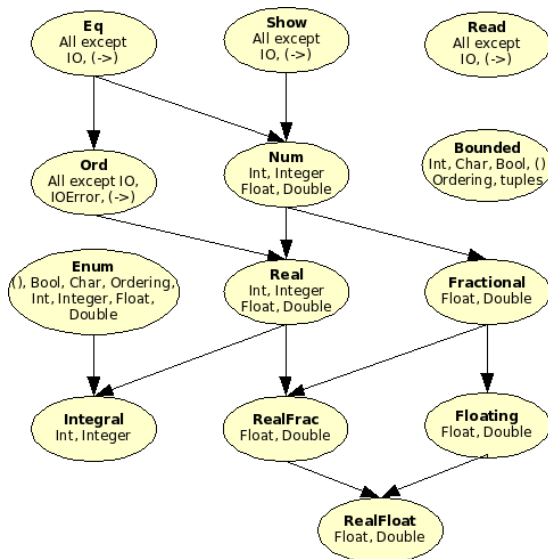
- Стандартный экземпляр
- Ограничение на тип параметра

Наследование классов

- Классы типов могут быть связаны отношением наследования (inheritance).

```
class Eq a => Ord a where  
  ...
```

Наследование в стандартной библиотеке



Содержание

- 1 Стандартные классы типов
 - Основные понятия
 - Класс Show и его экземпляры
- 2 Примеры собственных классов типов

Экземпляр класса Show: светофор

```
data TrafficLight = Red | Yellow | Green
```

```
instance Show TrafficLight where
```

```
  show Red = "Red color"
```

```
  show Yellow = "Yellow color"
```

```
  show Green = "Green color"
```

```
ghci> [Red, Yellow, Green]
```

```
[Red color,Yellow color,Green color]
```

Экземпляр класса Show: комплексные числа

```
data Complex = Complex Double Double
```

```
instance Show Complex where
```

```
  show (Complex re im) = show re ++ " + i*" ++ show im
```

```
ghci> Complex 4 6  
4.0 + i*6.0
```

Экземпляр класса Show: комплексные числа

```
data Complex a = Complex a a
```

```
instance (Show a) => Show (Complex a) where  
  show (Complex re im) = show re ++ " + i*" ++ show im
```

```
ghci> Complex 4 6
```

```
4 + i*6
```

```
ghci> Complex 4.1 6.2
```

```
4.1 + i*6.2
```

Содержание

1 Стандартные классы типов

2 Примеры собственных классов типов

- Класс типов «Да—Нет»
- Циклическое перечисление

Содержание

- 1 Стандартные классы типов
- 2 Примеры собственных классов типов
 - Класс типов «Да—Нет»
 - Циклическое перечисление

Javascript — слабо типизированный язык

```
var a = if (0) "ДА!" else "НЕТ!"  
var b = if ("") "ДА!" else "НЕТ!"  
var c = if (false) "ДА!" else "НЕТ!"
```

```
var a = if (1024) "ДА!" else "НЕТ!"  
var b = if ("КУ") "ДА!" else "НЕТ!"  
var c = if (true) "ДА!" else "НЕТ!"
```

Реализовать подобное поведение в языке Haskell невозможно из-за требований строгой типизации, но можно написать функцию-преобразователь к Bool.

Класс типов «Да—Нет»

Объявление класса типов

```
class YesNo a where  
  yesno :: a -> Bool
```

```
instance YesNo Int where  
  yesno 0 = False  
  yesno _ = True
```

```
instance YesNo (Maybe a) where  
  yesno (Just _) = True  
  yesno Nothing = False
```

```
instance YesNo [a] where  
  yesno [] = False  
  yesno _ = True
```

```
instance YesNo Bool where  
  yesno = id
```

Примеры использования

```
ghci> yesno $ length []
```

```
False
```

```
ghci> yesno "!!!"
```

```
True
```

```
ghci> yesno ""
```

```
False
```

```
ghci> yesno []
```

```
False
```

```
ghci> yesno [0,0,0]
```

```
True
```

```
ghci> yesno $ Just 0
```

```
True
```

```
ghci> yesno True
```

```
True
```

Слаботипизированный аналог конструкции if/then/else

```
yesnoIf :: (YesNo y) => y -> a -> a -> a
yesnoIf yesnoVal yesResult noResult =
    if yesno yesnoVal
      then yesResult
      else noResult
```

NB!

Эта функция будет работать с новым типом при наличии для него экземпляра класса YesNo.

```
ghci> yesnoIf [ ] "ДА!" "НЕТ!"
"НЕТ!"
ghci> yesnoIf [2,3,4] "ДА!" "НЕТ!"
"ДА!"
ghci> yesnoIf (Just 500) "ДА!" "НЕТ!"
"ДА!"
ghci> yesnoIf Nothing "ДА!" "НЕТ!"
"НЕТ!"
```

Содержание

1 Стандартные классы типов

2 Примеры собственных классов типов

- Класс типов «Да—Нет»
- Циклическое перечисление

Задача об ориентировании локатора

Постановка задачи

Локатор ориентирован на одну из сторон света (север, запад, юг, восток) и может принимать три команды поворота: поворот налево, поворот направо, поворот на 180° . Определить ориентацию локатора после выполнения заданной команды, если локатор находится в заданной ориентации.

Класс типов циклического перечисления

```
class (Eq a, Enum a, Bounded a) => CEnum a where
  cpred :: a -> a
  cpred d
    | d == minBound = maxBound
    | otherwise = pred d

  csucc :: a -> a
  csucc d
    | d == maxBound = minBound
    | otherwise = succ d
```


Задача об ориентировании локатора: решение

```
data Direction = North | East | South | West
  deriving (Eq, Ord, Enum, Bounded, Show)
```

```
instance CEnum Direction
```

```
data Turn = TLeft | TRight | TAround
```

```
orient :: Turn -> Direction -> Direction
```

```
orient TLeft = cpred
```

```
orient TRight = csucc
```

```
orient TAround = cpred . cpred
```