Классы в Haskell

Лекция 6.

«Столпы» ООП.

Наследование
Инкапсуляция
Полиморфизм
Абстракция

Полиморфизм

Полиморфизм – способность программы автоматически выбирать правильный метод (функцию) для использования в зависимости от типа данных, полученных для обработки.

При этом на этапе трансляции тип данных может быть вовсе неизвестен.

*можно спать

Полиморфизм в Haskell

Поддерживаются два вида полиморфизма:

- Параметрический (чистый)
- Специальный (на основе классов)

*можно просыпаться

Параметрический полиморфизм

Функция реализуется для всех типов одинаково.

Таким образом, она реализуется для любого типа данных.

*можно просыпаться

Параметрический полиморфизм

Пример:
Рассмотрим тип функции tail

```
tail :: [a] - > [a]
```

a – переменная типа, в каждом конкретном случае ей присваивается конкретное значение.

Если функции на вход поступил список целых чисел, а принимает значение Integer.

*нужно проснуться

Специальный полиморфизм

Идея: перегрузка имён функций и прочих объектов.

Разные типы имеют различную структуру и различное представление (в т.ч. и в памяти ЭВМ)

Для работы с различными типами необходимы методы, специфичные для каждого из них.

*важно

Специальный полиморфизм

Каноничный пример специального полиморфизма в Haskell:

```
elem x [] = False
elem x (h:t) = ( x == h ) || ( elem x t )
```

При сравнении разных типов используются разные функции сравнения.

*важно

Специальный полиморфизм

```
class Eq a where
(==) :: [a] -> [a] -> Bool
```

Читать как: «Тип a является экземпляром класса Eq <=> для этого типа перегружена операция сравнения соответствующего типа»

*важно

Определение классов

Определение классов может содержать:

- Перечисления методов с указанием их типов
- Определение приоритетов инфиксных методов класса
- Определение способа вычисления для любого метода класса, которое используется по умолчанию.

*можно спать

Определение классов

Общий вид определения классов:

```
class [<context> =>] <class_name> <type_var>
[where
{<method_name> :: <method_type>}
{infix[r|l] <value> <method_name>}
{<method_name_i> = <definition_i>}]
```

*можно спать

Контексты

Ограничения, накладываемые на тип параметрической переменной в типе функции называется «контекстом».

```
Eq a => a -> a -> a
```

Следует понимать как «любой тип, принадлежащий классу Eq, может быть использован в выражении типа `a -> a -> a`»

Наследование

Классы можно наследовать от других классов.

Описание наследования – контексты.

Пример:

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

*basic syntax

Наследование

Определение класса Ord можно прочитать так:

«Класс Ord, параметризующий тип a,
наследует все методы класса Eq, но при этом
определяет свои собственные, а именно...»

*можно спать

Наследование

Наследование может быть множественным:

```
class (Read a, Show a) => Textual a
```

Реализация

Классы не являются типами данных.

Невозможно создать объект, чей тип был бы некоторым классом.

Для типов данных необходимо определять, экземплярами каких классов они являются.

*to sum it up

Реализация

Пример:

```
data MyType a = MyType a
instance (Eq a)=>Eq (MyType a) where
  (MyType a) == (MyType b) = a == b
```

«Тип MyType a принадлежит классу Eq если a принадлежит Eq и операция сравнения реализована след. образом...»

*what is the type of MyType a?

Реализация

Классы и типы данных независимы друг от друга, они могут быть определены без всякой связи друг с другом.

Только ключевое слово instance позволяет связывать классы и типы данных.

*важно