Эссе

На тему «Сравнение синтаксиса Java и Haskell»

Существуют различные парадигмы языков программирования. ООП, функциональное программирование, эзотерические языки и др. В объектно-ориентированных языках (ООП) основе концепции лежит понятие *объекта* — некой сущности, которая объединяет в себе поля (данные) и методы (выполняемые объектом действия). В программах происходит взаимодействие таких объектов. Java относится к ООП. С версии 8.0 Java поддерживает некоторые особенности функциональных языков, таких как лямбда-выражений, функциональные интерфейсы (предикаты, поставщики) В функциональных языках, а в частности в Haskell, основной управляющей структурой является функция. Программа состоит из последовательного вызова функций с определенными аргументами.

Рассмотрим синтаксис языков.

В Java для описания объектов нужно объявить класс с полями и методами:

public class MyClass {

int myField;

String myString;

public void myMethod (int intArgument, String stringArgument) {

System.out.println(stringArgument + “ ” + Integer.toString(intArgument));

}

}

В описанном коде объявляется класс с двумя полями (числовым и строковым) и метод, который выводит на экран значения переданных аргументов.

Довольно громоздко, однако такая структура позволяет программисту легче понять, что делает программа, как написать программу для определенной задачи.

Рассмотрим синтаксис языка Haskell. Так как язык функциональный, то никакие классы объявлять не нужно. Нужно описывать функции:

module Functions where

factorial :: Int -> Int

factorial 0 = 1

factorial n | n > 0 = n \* factorial (n - 1)

Этот код реализовывает функцию факториала. Такой код сложнее понять рядовому программисту, однако для человека, знакомого с Haskell, код очень компактен и лаконичен.

Так как в Haskell нет циклов, для перебора элементов списка необходимо использовать рекурсию.

Рассмотрим реализацию быстрой сортировки в Java и Haskell.

Java:

**int** partition (**int**[] array, **int** start, **int** end)

{

**int** marker = start;

**for** ( **int** i = start; i <= end; i++ )

{

**if** ( array[i] <= array[end] )

{

**int** temp = array[marker]; *// swap*

array[marker] = array[i];

array[i] = temp;

marker += 1;

}

}

**return** marker - 1;

}

**void** quicksort (**int**[] array, **int** start, **int** end)

{

**if** ( start >= end )

{

**return**;

}

**int** pivot = partition (array, start, end);

quicksort (array, start, pivot-1);

quicksort (array, pivot+1, end);

}

Теперь на Haskell:

qsort [] **=** []

qsort (x**:**xs) **=** qsort (**filter** (**<**x) xs) **++** [x] **++** qsort (**filter** (**>=**x) xs)

Как видно, код на Haskell значительно короче. При чем в обеих реализациях используется рекурсия. Очевидно, что код на Haskell исполняется быстрее. Но это обусловлено не только компактностью кода, но и такой особенностью языка, как ленивые вычисления (отложенные вычисления). Вычисления производятся только тогда, когда пользователю нужно увидеть результат. Это позволяет описывать бесконечные списки, что невозможно в ООП языках. Только когда пользователь попросит показать какую-то часть этого списка, компилятор вычислит именно требуемые элементы и покажет их.

Однако такая простота Haskell работает на негромоздких задачах. В задачах масштабов компаний, сложных технологий, вроде машин, или самолетов, вперед выходит ООП языки.

Таким образом, у каждого языка есть и плюсы и минусы.