Отчет по лабораторной работе № 3
по курсу «Функциональное программирование»

Студент группы 8о-306Б МАИ *Макаров Никита*, №11 по списку
Контакты: quizbeat@gmail.com
Работа выполнена: 17.04.2015

Преподаватель: Иванов Дмитрий Анатольевич, доц. каф. 806
Отчет сдан:
Итоговая оценка:
Подпись преподавателя:

1. Тема работы

Обобщённые функции, методы и классы объектов.

2. Цель работы

Научиться определять простейшие классы, порождать экземпляры классов, считывать и изменять значения слотов, научиться определять обобщённые функции и методы.

3. Задание

Определить обычную функцию `on-single-line-p` - предикат,

- принимающий в качестве аргумента список точек (радиус-векторов),
- возвращающий `T`, если все указанные точки лежат на одной прямой (вычислять с допустимым отклонением `tolerance`).

Точки могут быть заданы как декартовыми координатами (экземплярами `cart`), так и полярными (экземплярами `polar`).

```
(defvar *tolerance* 0.001)

(defun on-single-line-p (vertices)
  ...)
```

4. Оборудование студента

Процессор Intel Core i5 2 @ 1.3GHz, память: 4Gb, разрядность системы: 64.

5. Программное обеспечение

ОС Mac OS X 10.9, среда Clozure CL 1.10

6. Идея, метод, алгоритм

Очевидно, что точки лежат на одной прямой, если их углы в полярном представлении совпадают. Поэтому будем рекурсивно идти по списку точек, преобразовывать точки в полярную систему координат и сравнивать их углы с учетом погрешности `tolerance`. `on-single-line-p` принимает список точек и возвращает `T`, если функция `on-single-line` от этого же списка точек вернула число. `on-single-line` непосредственно сравнивает точки с помощью функции `approx-eq`, и если на каком-либо сравнении разница углов окажется больше `tolerance`, функция будет возвращать вверх по рекурсии `nil`, и следовательно число возвращено не будет.

7. Сценарий выполнения работы

8. Распечатка программы и её результаты

8.1. Исходный код

```
(defun square (x) (* x x))

(defclass cart()
    ((x :initarg :x :accessor cart-x)
     (y :initarg :y :accessor cart-y)
    )
)

(defclass polar()
    ((radius :initarg :radius :accessor radius)
     (angle  :initarg :angle  :accessor angle)
    )
)

(defgeneric get-angle (arg)
    (:documentation "returns angle of arg point")
)

(defmethod get-angle ((c cart))
    (atan (cart-y c) (cart-x c))
)

(defmethod get-angle ((p polar))
    (angle p)
)

(defgeneric get-radius (arg)
    (:documentation "returns radius of arg point")
)
```

```
(defmethod get-radius ((c cart))
    (sqrt (+ (square (cart-x c))
             (square (cart-y c)))
    )
)

(defmethod get-radius ((p polar))
    (radius p)
)

(defgeneric to-polar (arg)
    (:documentation "transformation to polar coordinate system")
)

(defmethod to-polar ((p polar)) p)

(defmethod to-polar ((c cart))
    (make-instance 'polar
        :radius (get-radius c)
        :angle (get-angle c)
    )
)

(defmethod print-object ((c cart) stream)
    (format stream "[CART x:~d y:~d]"
        (cart-x c) (cart-y c)
    )
)

(defmethod print-object ((p polar) stream)
    (format stream "[POLAR radius:~d angle:~d]"
        (radius p) (angle p)
    )
)

(defvar tolerance 0.001)

(defun approx-eq (x y)
    (if (and (numberp x) (numberp y))
        (<= (abs (- x y)) tolerance)
        nil
    )
)

(defun on-single-line (vertices)
```

```
    (if (= (length vertices) 1)
        (get-angle (first vertices))
        (if (approx-eq (get-angle (first vertices))
                       (on-single-line (rest vertices)))
            (get-angle (first vertices))
            nil
        )
    )
)

(defun on-single-line-p (vertices)
    (numberp (on-single-line vertices))
)

(defvar p1 (make-instance 'cart :x 1 :y 2))
(defvar p2 (make-instance 'cart :x 2 :y 4))
(defvar p3 (to-polar (make-instance 'cart :x 3 :y 6)))
(defvar p4 (make-instance 'cart :x 2 :y 2))
(defvar p5 (make-instance 'polar :radius 42 :angle 1.107111))
(defvar l1 (list p1 p2 p3))
(defvar l2 (list p1 p2 p4))
(defvar l3 (list p1 p2 p3 p5))
```

8.2. Результаты работы

```
> (on-single-line-p l1)
T
> (on-single-line-p l2)
NIL
> (on-single-line-p l3)
T
```

9. Дневник отладки

10. Замечания, выводы

С помощью данной лабораторной работы я ознакомился с классами и обобщенными функциями в языке Common Lisp. Синтаксис для создания классов довольно прост, с ним было нетрудно разобраться. Обощенные функции играют немаловажную роль. Они позволяют не создавать несколько похожих функций для разных классов, а просто написать реализации одной функции, которая будет применима для нескольких классов.