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

Воспользуемся уравнением прямой, проходящей через 2 точки. Если 3 точки лежат на одной прямой, то для них выполняется равенство:

$$\frac{x_1 - x_2}{x_3 - x_2} = \frac{y_1 - y_2}{y_3 - y_2}.$$

Чтобы избежать погрешностей при делении, заменим частные на произведения:

$$(x_1 - x_2) * (y_3 - y_2) = (x_3 - x_2) * (y_1 - y_2).$$

Функция `on-single-line-p` рекурсивно проверяет выполнение вышеприведенного равенства для всех элементов списка с помощью функции `on-single-line`, если его длина больше трех. Если длина равна трем, то функция `on-single-line` вызывается один раз. Для длины списка равной двум или одному ответ, очевидно, `T`.

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)
    )
)

(defmethod cart-x ((p polar))
    (* (radius p) (cos (angle p)))
)

(defmethod cart-y ((p polar))
    (* (radius p) (sin (angle p)))
)

(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)
    (<= (abs (- x y)) tolerance)
)

(defun on-single-line (v)
    (approx-eq (* (- (cart-x (first v)) (cart-x (second v)))
                  (- (cart-y (third v)) (cart-y (second v))))
               (* (- (cart-x (third v)) (cart-x (second v)))
                  (- (cart-y (first v)) (cart-y (second v))))
    )
)

(defun on-single-line-p (vertices)
    (cond
        ((< (length vertices) 3) T)
        ((= (length vertices) 3) (on-single-line vertices))
        ((> (length vertices) 3)
            (if (on-single-line-p (rest vertices))
                (on-single-line (list (first vertices)
                                      (second vertices)
                                      (third vertices)))
            )
        )
    )
)

(defun t (v) (on-single-line-p v)) ; fast testing

; tests
(defvar p1 (make-instance 'cart :x 1 :y 2))
(defvar p2 (make-instance 'cart :x 2 :y 4))
(defvar p3 (make-instance 'cart :x 3 :y 6))

(defvar p4 (make-instance 'cart :x 0 :y 1))
```

```
(defvar p5 (make-instance 'cart :x 1 :y 1))
(defvar p6 (make-instance 'cart :x 2 :y 1))

(defvar p7 (make-instance 'cart :x 4 :y 8.0002))
(defvar p8 (make-instance 'cart :x 5 :y 10.00001))
(defvar p9 (make-instance 'cart :x 5.0005 :y 10.0001))

(defvar p10 (make-instance 'polar :radius 1 :angle 1))
(defvar p11 (make-instance 'polar :radius 2 :angle 1))
(defvar p12 (make-instance 'polar :radius 3 :angle 1))
(defvar p13 (make-instance 'polar :radius 4 :angle 1.000001))

(defvar l1 (list p1 p2 p3)) ; T
(defvar l2 (list p1 p2 p4)) ; NIL
(defvar l3 (list p1 p2 p3 p2 p3 p1)) ; T

(defvar l4 (list p4 p5 p6)) ; T
(defvar l5 (list p7 p8 p9)) ; T
(defvar l6 (list p1 p2 p3 p7 p8 p9)) ; T

(defvar l7 (list p10 p11 p12)) ; T
(defvar l8 (list p10 p11 p12 p13)) ; T
(defvar l9 (list p10 p11 p12 p13 p1)) ; NIL
```

8.2. Результаты работы

```
> (on-single-line-p l1)
T
> (on-single-line-p l2)
NIL
> (on-single-line-p l3)
T
> (on-single-line-p l4)
T
> (on-single-line-p l5)
T
> (on-single-line-p l6)
T
> (on-single-line-p l7)
T
> (on-single-line-p l8)
T
> (on-single-line-p l9)
NIL
```

9. Дневник отладки

`15.05.15` - Исправлен алгоритм.

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.