Отчет по лабораторной работе №5
по курсу «Функциональное программирование»

Студент группы 8о-306Б Пивницкий Даниэль Сергеевич, № по списку 13.

Контакты: pivnitskiydaniel@gmail.com
Работа выполнена: 06.06.2022
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 (вариант № 5.23)

Определить обобщённую функцию и методы on-single-line3-p - предикат,

- принимающий в качестве аргументов три точки (радиус-вектора) и необязательный параметр tolerance (допуск),
- возвращающий T, если три 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).

```
(defgeneric on-single-line3-p (v1 v2 v2 &optional tolerance))
(defmethod on-single-line3-p ((v1 cart) (v2 cart) (v3 cart) &optional (tolerance 0.001))
  ...)
```

4. Оборудование студента

Ноутбук Lenovo Legion 5 Pro, процессор Ryzen 5 5600h, память 16ГБ, 64-разрядная система.

5. Программное обеспечение

ОС Windows 10, программа LispWorks Personal Edition 7.1.2

6. Идея, метод, алгоритм

Три точки A, B, C лежат на одной прямой, если угол между векторами AB и AC равен 0 (т.е. модуль косинуса угла между векторами равен 1). При наличии параметра с отклонением модуль косинуса угла должен лежать в промежутке [1 - tolerance; 1].

Если точки заданы в декартовых координатах, то косинус угла можем найти как отношение скалярного произведения к произведению длин векторов.

Если точки заданы в полярных координатах, то будем искать косинус разности угла векторов AB и AC.

7. Сценарий выполнения работы

1. Изучение обобщенных функций в языке Коммон Лисп.
2. Изучение классов для точек в декартовой и полярной системах координат в языке Коммон Лисп, а также обобщенных функций add2 и mul2.
3. Составление математического алгоритма для определения, находятся ли заданные три точки на прямой.
4. Реализация методов обобщенной функции для решения этой задачи на Коммон Лиспе.

8. Распечатка программы и её результаты

Программа

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

; Объявим класс точек на плоскости в декартовой системе координат
(defclass cart ()
  ((x :initarg :x :reader cart-x)
   (y :initarg :y :reader cart-y)))

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

; Объявим класс точек на плоскости в полярной системе координат
(defclass polar ()
  ((radius :initarg :radius :accessor radius)
   (angle  :initarg :angle  :accessor angle)))

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

; Перевод из декартовой системы в полярную
(defmethod radius ((c cart))
  (sqrt (+ (square (cart-x c))
           (square (cart-y c)))))

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

(defgeneric to-polar (arg)
 (:documentation "Преобразование аргумента в полярную систему.")
 (:method ((p polar))
  p)
 (:method ((c cart))
  (make-instance 'polar
                 :radius (radius c)
                 :angle (angle c))))

; Перевод из полярной системы в декартову
(defmethod cart-x ((p polar))
  (* (radius p) (cos (angle p))))
```

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

(defgeneric to-cart (arg)
  (:documentation "Преобразование аргумента в декартову систему.")
  (:method ((c cart))
   c)
  (:method ((p polar))
   (make-instance 'cart
                  :x (cart-x p)
                  :y (cart-y p))))

; Обобщенная функция сложения
(defgeneric add2 (arg1 arg2)
 (:method ((n1 number) (n2 number))
  (+ n1 n2)))

(defmethod add2 ((c1 cart) (c2 cart))
  (make-instance 'cart
                 :x (+ (cart-x c1) (cart-x c2))
                 :y (+ (cart-y c1) (cart-y c2))))

(defmethod add2 ((p1 polar) (p2 polar))
  (to-polar (add2 (to-cart p1)
                  (to-cart p2))))

(defmethod add2 ((c cart) (p polar))
  (add2 c (to-cart p)))

; Обобщенная функция умножения
(defgeneric mul2 (arg1 arg2)
  (:method ((n1 number) (n2 number))
   (* n1 n2)))

(defmethod mul2 ((n number) (c cart))
  (make-instance 'cart
                 :x (* n (cart-x c))
                 :y (* n (cart-y c))))

(defun normalize-angle (a)
  ;; Привести полярный угол a в диапазон (-pi, pi]
  (let* ((2pi (* 2 pi))
         (rem (rem a 2pi)))
    (cond ((< pi rem)
           (- rem 2pi))
          ((<= rem (- pi))
           (+ 2pi rem))
          (t rem))))

(defmethod mul2 ((n number) (p polar))
  (let ((a (angle p)))
    (make-instance 'polar
                   :radius (abs (* n (radius p)))
                   :angle (if (< n 0)
                              (normalize-angle (+ a pi))
                              a))))

; Нахождение скалярного произведения радиус-векторов
(defgeneric scalar-product (arg1 arg2)
  (:method ((c1 cart) (c2 cart))
    (+ (* (cart-x c1) (cart-x c2)) (* (cart-y c1) (cart-y c2)))))

; Обобщенная функция определения, лежат ли три точки на одной прямой
; (возможно с некоторым допустимым отклонением)
(defgeneric on-single-line3-p (v1 v2 v3 &optional tolerance))
```

```
(defmethod on-single-line3-p ((v1 cart) (v2 cart) (v3 cart) &optional (tolerance
0.001))
  (let ((v12 (add2 v2 (mul2 -1 v1))) ; v2 - v1
        (v13 (add2 v3 (mul2 -1 v1))) ; v3 - v1
       )
    (setq cos_v12_v13 (/ (scalar-product v12 v13) (* (radius v12) (radius
v13))))
    (> (abs cos_v12_v13) (- 1 tolerance))))

(defmethod on-single-line3-p ((v1 polar) (v2 polar) (v3 polar) &optional
(tolerance 0.001))
  (let ((v12 (add2 v2 (mul2 -1 v1))) ; v2 - v1
        (v13 (add2 v3 (mul2 -1 v1))) ; v3 - v1
       )
    (setq cos_v12_v13 (cos (- (angle v12) (angle v13))))
    (> (abs cos_v12_v13) (- 1 tolerance))))
```

Результаты

```
CL-USER 24 > (setq a (make-instance 'cart :x 0 :y 0))
[CART x 0 y 0]

CL-USER 25 > (setq b (make-instance 'cart :x 1.2 :y -5.3))
[CART x 1.2 y -5.3]

CL-USER 26 > (setq c (make-instance 'cart :x -1.2 :y 5.3))
[CART x -1.2 y 5.3]

CL-USER 27 > (setq d (make-instance 'cart :x 122 :y 191))
[CART x 122 y 191]

CL-USER 28 > (on-single-line3-p a b c)
T

CL-USER 29 > (on-single-line3-p b c a)
T

CL-USER 30 > (on-single-line3-p a b d)
NIL

CL-USER 31 > (setq ap (to-polar a))
[POLAR radius 0.0 angle 0.0]

CL-USER 32 > (setq bp (to-polar b))
[POLAR radius 5.4341517 angle -1.3481354]

CL-USER 33 > (setq cp (to-polar c))
[POLAR radius 5.4341517 angle 1.7934573]

CL-USER 34 > (setq dp (to-polar d))
[POLAR radius 226.63848 angle 1.0023751]

CL-USER 35 > (on-single-line3-p ap bp cp)
T

CL-USER 36 > (on-single-line3-p bp cp ap)
T

CL-USER 37 > (on-single-line3-p ap bp dp)
NIL
```

9. Дневник отладки

№	Дата, время	Событие	Действие по исправлению	Примечание

1				

10. Замечания автора по существу работы

Для меня было удивлением, что на языке Коммон Лисп можно описывать классы и реализовывать обобщенные функции (полиморфизм).

11. Выводы

Выполнив данную работу, я изучил классы и обобщенные функции в языке Коммон Лисп и самостоятельно реализовал простейшие математические обобщенные функции. При выполнении работы очень полезными оказались некоторые обобщенные функции и классы, данные на лекциях.