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

Студент группы М8О-307 МАИ *Безлуцкая Елизавета*, №2 по списку
Контакты: lizabezlutskaya@gmail.com
Работа выполнена: 30.05.2019

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

Тема работы

Обобщённые функции, методы и классы объектов.

Цель работы

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

Задание (вариант №36)

Объемлющий прямоугольник для геометрической фигуры - это такой, который

- полностью заключает в себе фигуру,
- имеет стороны, параллельные осям координат.

Дан экземпляр класса `line` (отрезок), причём концы отрезка могут задаваться как в декартовых (экземплярами `cart`), так и в полярных координатах (экземплярами `polar`).

Задание: Написать обобщённую функцию и метод, возвращающий список из четырех вершин объемлющего четырёхугольника для отрезка.

```
(defgeneric containing-rect (shape))
(defmethod containing-rect ((l line))
  ...)
```

Оборудование студента

Процессор Intel Core i5-3230M 4 @ 2.6GHz, память: 16Gb, разрядность системы: 64.

Программное обеспечение

Ubuntu 16.04 LTS, clisp compiler

Идея, метод, алгоритм

Метод `containing-rect` работает следующим образом:

- если отрезок расположен по диагонали ($x_1 \neq x_2$ и $y_1 \neq y_2$), то выводятся вершины $(x_1, y_1), (x_1, y_2), (x_2, y_2), (x_2, y_1)$ (Рис.1)
- если отрезок расположен по вертикали($x_1 = x_2$), то выводятся кооррдинаты прямоугольника со смещением по x влево и вправо – $(x_1 - 1, y_1), (x_2 - 1, y_2), (x_2 + 1, y_2), (x_1 + 1, y_1)$(Рис.2)
- если отрезок расположен по горизонтали($y_1 = y_2$), то выводятся кооррдинаты прямоугольника со смещением по x влево и вправо – $(x_1, y_1 - 1), (x_2, y_2 - 1), (x_2, y_2 + 1), (x_1, y_1 + 1)$(Рис.3)

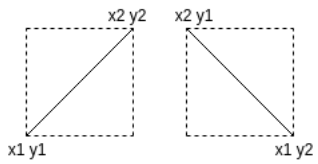

Рис.1

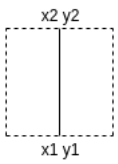

Рис.2

Рис.3

Сценарий выполнения работы

Распечатка программы и её результаты

Исходный код

```
(defclass line ()
 ((start :initarg :start :accessor line-start)
  (end   :initarg :end   :accessor line-end)))

(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)))

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

```
16
17 (defmethod cart-x ((p polar))
18   (* (radius p) (cos (angle p))))
19
20 (defmethod cart-y ((p polar))
21   (* (radius p) (sin (angle p))))
22
23 (defgeneric containing-rect (shape)
24     )
25
26 (defmethod containing-rect ((l line))
27     (let ((x1 (cart-x (line-start l)))
28           (y1 (cart-y (line-start l)))
29           (x2 (cart-x (line-end l)))
30           (y2 (cart-y (line-end l))))
31         (cond ((= x1 x2)
32                (list (make-instance 'cart :x (1- x1) :y y1)
33                      (make-instance 'cart :x (1+ x1) :y y1)
34                      (make-instance 'cart :x (1- x2) :y y2)
35                      (make-instance 'cart :x (1+ x2) :y y2))
36                )
37               ((= y1 y2)
38                (list (make-instance 'cart :x x1 :y (1- y1))
39                      (make-instance 'cart :x x1 :y (1+ y1))
40                      (make-instance 'cart :x x2 :y (1- y2))
41                      (make-instance 'cart :x x2 :y (1+ y2))))
42               (t
43                (list (make-instance 'cart :x x1 :y y1)
44                      (make-instance 'cart :x x1 :y y2)
45                      (make-instance 'cart :x x2 :y y2)
46                      (make-instance 'cart :x x2 :y y1))))))
```

Результаты работы

```
1 (print (containing-rect (make-instance 'line
2           :start (make-instance 'cart :x 1 :y 3)
3           :end (make-instance 'cart :x 4 :y 1))))
4 ([CART x 1 y 3] [CART x 1 y 1] [CART x 4 y 1] [CART x 4 y 3])
5
6
7 (print (containing-rect (make-instance 'line
8           :start (make-instance 'cart :x 2 :y 2)
9           :end (make-instance 'cart :x 6 :y 5))))
10 ([CART x 2 y 2] [CART x 2 y 5] [CART x 6 y 5] [CART x 6 y 2])
11
12
13 (print (containing-rect (make-instance 'line
14           :start (make-instance 'cart :x 2 :y 1)
15           :end (make-instance 'cart :x 2 :y 5))))
16 ([CART x 1 y 1] [CART x 3 y 1] [CART x 1 y 5] [CART x 3 y 5])
17
```

```
(print (containing-rect (make-instance 'line
          :start (make-instance 'cart :x 0 :y 1)
          :end (make-instance 'polar :radius 5 :angle (/ pi 4)))))
([CART x 0 y 1] [CART x 0 y 3.5355339059327376221L0]
 [CART x 3.5355339059327376221L0 y 3.5355339059327376221L0]
 [CART x 3.5355339059327376221L0 y 1])
```

Дневник отладки

Дата	Событие	Действие по исправлению	Примечание
4.06.2019	Концы отрезка могут быть заданы и в полярных координатах, не учитывается данный случай	Добавлены методы – декартовы координаты по полярным	

Замечания автора по существу работы

Замечаний не имею.

Выводы

В данной лабораторной работе я встретила новую для себя конструкцию – generic. Считаю, что написание для нее методов в отдельном блоке кода усложняет читаемость программы. Возможно, такое неудобство обосновано для некоторых задач.