Московский авиационный институт
(национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №7
Тема: Проектирование структуры классов

Студент: Лагуткина Мария Сергеевна
Группа: М8О-206Б-19
Преподаватель: Чернышов Л. Н.
Дата: 11.12.2020
Оценка:

Москва, 2020

1 Постановка задачи

Спроектировать простейший «графический» векторный редактор. Требование к функционалу редактора:

- создание нового документа;
- импорт документа из файла;
- экспорт документа в файл;
- создание графического примитива (согласно варианту задания);
- удаление графического примитива;
- отображение документа на экране (печать перечня графических объектов и их характеристик в std::cout);
- реализовать операцию undo, отменяющую последнее сделанное действие.

Должно действовать для операций добавления/удаления фигур.
Требования к реализации:

- Создание графических примитивов необходимо вынести в отдельный класс – Factory;
- Сделать упор на использовании полиморфизма при работе с фигурами;
- Взаимодействие с пользователем (ввод команд) реализовать в функции main.

Вариант задания: 10.
Типы фигур: квадрат, прямоугольник, трапеция.

2 Описание программы

Для решения задачи был реализован класс Figure и классы-наследники для работы с квадратом, прямоугольником и трапецией. Нужные фигуры создаются по номеру фигуры (0 - квадрат, 1 - прямоугольник,2 - трапеция) в структуре Factory. Указатели на созданные фигуры хранятся в векторе. Каждое состояние вектора указателей кладется в стек в структуре Originator. Это позволяет выполнять операцию undo, отменяющее последнее действие: для этого нужно взять со стека последнее положенное состояние. Также реализована работа с документом:сохранение и загрузка данных. В документ сохраняется сначала количество фигур, затем тип фигур и их

координаты.
Также реализован интерфейс для взаимодействия с пользователем. Сначала печатается справка по работе с программой, в которой отражаются все возможные операции с редактором. Ввод фигуры производится по координатам, затем проверяется корректность введенных координат. Если фигура не корректна, в поток ошибок выводится: ERROR INPUT. Если при удалении фигуры по индексу, пользователь указывает некорректный индекс, в поток ошибок выводится: BORDER OVERLAY. Если пользователь вводит некорректную команду, в поток ошибок выводится: ERROR COMMAND. Если любое из действий прошло успешно, пользователю выводится: OK.

3 Набор тестов

Программа получает на ввод команду и, если это нужно, аргументы к ней. Ознакомится со списком команд можно через команду 7:

```
Enter choice:
0 -create document
1 -add
2 -remove
3 -undo
4 -save in file
5 -load out file
6 -print all figures
7 -print usage

Types of figures:
0 -square
1 -rectangle
2 -trapezoid
```

Тест №1.
В первом тесте проверяется корректность ввода всех типов фигур и печать всех фигур. Первые 3 введенные фигуры предполагаются корректными. Вводится команда, тип фигуры, затем координаты четырех вершин фигуры. Замечание: более подробно корректность ввода этих фигур тестировалась в лабораторной работе №3.

```
1
0
-1 0 0 1 1 0 0 -1
1
```

```
1
1 1 1 3 3 3 3 1
1
2
-3 -2 0 2 2.5 2 5.5 -2
1
0
1 2 2 5 5 2 2 1
6
```

Тест №2.
Во втором тесте проверяется корректность работы с программой, проверяются работоспособность всех остальных команд.

```
1
1
1 1 1 3 3 3 3 1
1
0
-1 0 0 1 1 0 0 -1
6
3
6
1
2
-3 -2 0 2 2.5 2 5.5 -2
1
0
0 0 0 1 1 1 1 0
4
test.txt
3
3
6
2
0
6
5
test.txt
6
```

4 Результаты выполнения тестов

При каждом запуске программы печатается справка по ее работе. Для улучшения читаемости в данном разделе она не будет приводиться.
Тест №1.

```
1
0
-1 0 0 1 1 0 0 -1
OK
1
2
-3 -2 0 2 2.5 2 5.5 -2
OK
1
1
1 1 1 3 3 3 3 1
OK
1
0
1 2 2 5 5 2 2 1
ERROR INPUT
6
No:1 -It's square
Figure coordinates:
A: (-1,0)
B: (0,1)
C: (1,0)
D: (0,-1)
No:2 -It's trapezoid
Figure coordinates:
A: (-3,-2)
B: (0,2)
C: (2.5,2)
D: (5.5,-2)
No:3 -It's rectangle
Figure coordinates:
A: (1,1)
B: (1,3)
C: (3,3)
D: (3,1)
```

Тест №2.

```
1
1
1 1 1 3 3 3 3 1
OK
1
0
-1 0 0 1 1 0 0 -1
OK
6
No:1 -It's rectangle
Figure coordinates:
A: (1,1)
B: (1,3)
C: (3,3)
D: (3,1)

No:2 -It's square
Figure coordinates:
A: (-1,0)
B: (0,1)
C: (1,0)
D: (0,-1)
3
OK
6
No:1 -It's rectangle
Figure coordinates:
A: (1,1)
B: (1,3)
C: (3,3)
D: (3,1)
1
2
-3 -2 0 2 2.5 2 5.5 -2
OK
1
0
0 0 0 1 1 1 1 0
OK
```

```
4
Enter path: c:7.txt
Ok
3
OK
3
OK
6
No:1 -It's rectangle
Figure coordinates:
A: (1,1)
B: (1,3)
C: (3,3)
D: (3,1)
2
Enter index: 0
OK
6
5
Enter path: c:7.txt
OK
6
No:1 -It's rectangle
Figure coordinates:
A: (1,1)
B: (1,3)
C: (3,3)
D: (3,1)
No:2 -It's trapezoid
Figure coordinates:
A: (-3,-2)
B: (0,2)
C: (2.5,2)
D: (5.5,-2)
No:3 -It's square
Figure coordinates:
A: (0,0)
B: (0,1)
C: (1,1)
D: (1,0)
```

5 Листинг программы

```
1  //Лагуткина Мария СергеевнаМОБ,8-206-19
2  //Вариант 10: квадрат, прямоугольник, трапеция
3
4  #include <iostream>
5  #include <fstream>
6  #include <memory>
7  #include <stack>
8  #include <vector>
9  #include <string>
10
11 using namespace std;
12 using vertex_t = pair<double, double>;
13
14 istream& operator>> (istream& input, vertex_t & v) {
15     input >> v.first >> v.second;
16     return input;
17 }
18 ostream& operator<< (ostream& output, const vertex_t v) {
19     output << " ("<< v.first <<"," << v.second << ")" << '\n';
20     return output;
21 }
22 double lenght(vertex_t a, vertex_t b) {
23     return sqrt((b.first - a.first)*(b.first - a.first) + ((b.second - a.second)*(b.
       second - a.second)));
24 }
25 int CorrectInput(vertex_t& a, vertex_t& b, vertex_t& c, vertex_t& d, int fig) {
26     if (((abs(a.first - c.first) < numeric_limits<double>::epsilon()) &&
27         (abs(a.second - c.second) < numeric_limits<double>::epsilon())) ||
28         ((abs(b.first - d.first) < numeric_limits<double>::epsilon()) &&
29         (abs(b.second - d.second) < numeric_limits<double>::epsilon()))) {
30         return 1;
31     }
32     double len_ab = lenght(a, b);
33     double len_bc = lenght(b, c);
34     double len_cd = lenght(c, d);
35     double len_da = lenght(d, a);
36     double len_ac = lenght(a, c);
37     double len_bd = lenght(b, d);
38     if (fig == 1) {
39         if (abs(len_cd - len_ab) < numeric_limits<double>::epsilon() &&
40             abs(len_cd - len_bc) < numeric_limits<double>::epsilon() &&
41             abs(len_cd - len_da) < numeric_limits<double>::epsilon() &&
42             abs(len_bd - len_ac) < numeric_limits<double>::epsilon()) {
43             return 0;
44         }
45         return 1;
46     }
```

```
47      //проверка коректности прямоугольника: равенсво противоположных сторон и
        диагоналей
48      if (fig == 2) {
49          if (abs(len_ab - len_cd) < numeric_limits<double>::epsilon() && abs(len_bc -
        len_da) < numeric_limits<double>::epsilon() &&
50              abs(len_ac - len_bd) < numeric_limits<double>::epsilon()) {
51              return 0;
52          }
53          return 1;
54      }
55      //проверка коректности равнобедренной трапеции: равенсво боковых строн и
        диагоналей
56      if (fig == 3) {
57          if (abs(len_ab - len_cd) < numeric_limits<double>::epsilon() &&
58              abs(len_ac - len_bd) < numeric_limits<double>::epsilon()) {
59              return 0;
60          }
61          return 1;
62      }
63      return 0;
64  }
65  class Figure {
66  public:
67      virtual void Set(istream& is) = 0;
68      virtual ostream& Print(ostream& output) = 0;
69      virtual ~Figure() {};
70      virtual void Save(ofstream& os) const = 0;
71      virtual void Load(ifstream& is) = 0;
72      virtual size_t GetId() = 0;
73  };
74  enum figure_t {
75      square = 0,
76      rectangle = 1,
77      trapezoid = 2
78  };
79
80  class Square : public Figure {
81  public:
82      Square() {};
83      Square(vertex_t A, vertex_t B, vertex_t C, vertex_t D) {
84          a = A; b = B; c = C; d = D;
85      };
86      void Set(istream& is) override{
87         is >> a >> b >> c >> d;
88         if (CorrectInput(a, b, c, d, 1) == 1) {
89             throw logic_error("It‘s not a square");
90         }
91         id = 0;
92      }
```

```
93      size_t GetId() override{
94          return id;
95      }
96      ostream& Print(ostream& output) {
97          //output << "It's square\n";
98          output << "A:" << a << '\n' << "B:" << b <<'\n'
99                  << "C:" << c << '\n' << "D:" << d << '\n';
100         return output;
101     }
102     void Save(ofstream& output) const {
103         output << id << " ";
104         output << a.first << ' ' << a.second << ' ' <<
105                     b.first << ' ' << b.second << ' ' <<
106                     c.first << ' ' << c.second << ' ' <<
107                     d.first << ' ' << d.second << '\n';
108     }
109     void Load(ifstream& input) override{
110         input >> a.first >> a.second >> b.first >> b.second >>
111             c.first >> c.second >> d.first >> d.second;
112         id = 0;
113     }
114 private:
115     size_t id;
116     vertex_t a;
117     vertex_t b;
118     vertex_t c;
119     vertex_t d;
120 };
121 class Rectangle : public Figure {
122 public:
123     Rectangle() {};
124     Rectangle(vertex_t A, vertex_t B, vertex_t C, vertex_t D) {
125         a = A; b = B; c = C; d = D;
126     };
127     void Set(istream& is) override {
128         is >> a >> b >> c >> d;
129         if (CorrectInput(a, b, c, d, 2) == 1) {
130             throw logic_error("It‘s not a square");
131         }
132         id = 1;
133     }
134     size_t GetId() {
135         return id;
136     }
137     ostream& Print(ostream& output) {
138         output << "A:" << a << '\n' << "B:" << b << '\n' << "C:" << c << '\n' << "D:"
    << d << '\n';
139         return output;
140     }
```

```
141     void Save(ofstream& output) const {
142         output << id << "   ";
143         output << a.first <<' ' << a.second << ' ' <<
144             b.first << ' ' << b.second << ' ' <<
145             c.first << ' ' << c.second  << ' ' <<
146             d.first << ' ' << d.second << '\n';
147     }
148     void Load(ifstream& input) override {
149         input >> a.first >> a.second >> b.first >> b.second >>
150             c.first >> c.second >> d.first >> d.second;
151         id = 1;
152     }
153 private:
154     size_t id;
155     vertex_t a;
156     vertex_t b;
157     vertex_t c;
158     vertex_t d;
159 };
160 class Trapezoid : public Figure {
161 public:
162     Trapezoid() {};
163     Trapezoid(vertex_t A, vertex_t B, vertex_t C, vertex_t D) {
164         a = A; b = B; c = C; d = D;
165     };
166     void Set(istream& is) override {
167         is >> a >> b >> c >> d;
168         if (CorrectInput(a, b, c, d, 3) == 1) {
169             throw logic_error("It's not a square");
170         }
171         id = 2;
172     }
173     size_t GetId() {
174         return id;
175     }
176     ostream& Print(ostream& output) {
177         //output << "It's trapezoid\n";
178         output << "A:" << a << '\n' << "B:" << b << '\n' << "C:" << c << '\n' << "D:"
    << d << '\n';
179         return output;
180     }
181     void Save(ofstream& output) const {
182         output << id << " ";
183         output  << a.first <<' ' << a.second << ' ' <<
184             b.first << ' ' << b.second << ' ' <<
185             c.first << ' ' << c.second  << ' ' <<
186             d.first << ' ' << d.second << '\n';
187     }
188     void Load(ifstream& input) override {
```

```
        input >> a.first >> a.second >> b.first >> b.second >>
            c.first >> c.second >> d.first >> d.second;
        id = 2;
    }
private:
    size_t id;
    vertex_t a;
    vertex_t b;
    vertex_t c;
    vertex_t d;
};
struct Factory {
    static shared_ptr<Figure> create(figure_t t) {
        switch (t) {
        case figure_t::square:
            return make_shared<Square>();
        case figure_t::rectangle:
            return make_shared<Rectangle>();
        case figure_t::trapezoid:
            return make_shared<Trapezoid>();
        }
    }
};
struct Memento {
    vector<shared_ptr<Figure>> state;   //переписать на указателях
    vector<size_t> id;
    Memento() {};
    Memento(vector<shared_ptr<Figure>> other){
        state = other;
    };
};
struct Originator {  //структура, хранящая стек состояний
    stack<Memento> mementos;
    void createMemento(vector<shared_ptr<Figure>> state) {
        mementos.emplace(state);
    }
    vector<shared_ptr<Figure>> restore() {
        if (!mementos.empty()) {
            vector<shared_ptr<Figure>> result = move(mementos.top().state);
            mementos.pop();
            return result;
        }
        else throw logic_error("undo stack empty");
    }
};
class Document {
public:
    void DSave(vector<shared_ptr<Figure>> &f,string& file) {
        ofstream os(file);
```

```
238         if (!os) {
239             cerr << "INCORRECT PATH\n";
240             return;
241         }
242         os << f.size() << ' ';
243         for (int j = 0; j < f.size(); ++j) {
244             f[j]->Save(os);
245         }
246     }
247     void DLoad(vector<shared_ptr<Figure>> &figures, string& file) {
248         ifstream is(file);
249         int i;
250         shared_ptr<Figure> fig = nullptr;
251         if (!is) {
252             cerr << "INCORRECT PATH\n";
253             return;
254         }
255         int s;
256         is >> s;
257         for (int j = 0; j < s; ++j) {
258             is >> i;
259             fig = Factory::create((figure_t)i);
260             fig->Load(is);
261             figures.push_back(move(fig));
262         }
263     }
264 };
265 void Usage() {
266     cout << "Enter choice:\n"
267         "0 - create document\n"
268         "1 - add\n"
269         "2 - remove\n"
270         "3 - undo\n"
271         "4 - save in file\n"
272         "5 - load out file\n"
273         "6 - print all figures\n"
274         "7 - print usage\n\n";
275     cout << "Types of figures:\n"
276         "0 - square\n"
277         "1 - rectangle\n"
278         "2 - trapezoid\n";
279 }
280 int main(){
281     Originator origin;
282     char choice;
283     int index;
284     int i;
285     int idx;
286     string path ="";
```

```
287     Document doc;
288     vector<shared_ptr<Figure>> figures;
289     shared_ptr<Figure> fig = nullptr;
290     Usage();
291     while (true) {
292         cin >> choice;
293         switch (choice) {
294         case '1':
295             cin >> i;
296             fig = Factory::create((figure_t)i);
297             try {
298                 fig->Set(cin);
299             }
300             catch (logic_error) { cerr << "ERROR INPUT\n"; break; }
301             origin.createMemento(figures);
302             figures.push_back(move(fig));
303             cout << "OK\n";
304             break;
305         case '2':
306             cout << "Enter index: ";
307             cin >> index;
308             if (index >= 0 && index < figures.size()) {
309                 origin.createMemento(figures);
310                 figures.erase(figures.begin() + index);
311                 cout << "OK\n";
312                 break;
313             }
314             else { cerr << "BORDER OVERLAY\n"; break; }
315         case '3':
316             figures = origin.restore();
317             cout << "OK\n";
318             break;
319         case '4':
320             cout << "Enter path: ";
321             cin >> path;
322             doc.DSave(figures, path);
323             cout << "Ok\n";
324             break;
325         case '5':
326             cout << "Enter path: ";
327             cin >> path;
328             figures.clear();
329             doc.DLoad(figures, path);
330             origin.createMemento(figures);
331             cout << "OK\n";
332             break;
333         case '6':
334             idx = 0;
335             for (int j = 0; j < figures.size();++j) {
```

```
336                 cout << "No:" << ++idx << " - ";
337                 if (figures[j]->GetId() == 0) { cout << "It's square\n"; }
338                 if (figures[j]->GetId() == 1) { cout << "It's rectangle\n"; }
339                 if (figures[j]->GetId() == 2) { cout << "It's trapezoid\n"; }
340                 cout << "Figure coordinates:\n";
341                 figures[j]->Print(cout);
342             }
343             break;
344         case '7':
345             Usage();
346             break;
347         default:
348             cerr << "ERROR COMMAND\n";
349             break;
350         }
351     }
352     return 0;
353 }
```

6 GitHub

https://github.com/marianelia/MAI/tree/main/OOP/oop_exercise_07

7 Вывод

Выполняя лабораторную работу, я попыталась написать чистый код. Не скажу, что получилось соблюдать все правила, но написанный код все равно отличается в лучшую сторону от кода, например, в прошлых лабораторных. Также я узнала как просто реализовать откат к предыдущим действиям с помощью стека состояний. Такой подход удобен в реализации и понимании, но требует больших затрат по памяти.

Список литературы

[1] *Курс «Основы разработки на C++: белый пояс».* [Электронный ресурс] URL: https://www.coursera.org/learn/c-plus-plus-white (дата обращения 05.12.2020).

[2] *Документация Microsoft по C++.* [Электронный ресурс] URL: https://docs.microsoft.com/ru-ru/?view=msvc-16 (дата обращения 05.12.2020).