Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»
Дисциплина «Объектно-ориентированное программирование»

Лабораторная работа №7

Тема: Проектирование структуры классов

Студент: Инютин М. А.
Группа: М8О-207Б-19
Преподаватель: Чернышев Л. Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

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

- создание нового документа;
- импорт документа из файла;
- экпорт 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.

Требование 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 2. Квадрат, прямоугольник, трапеция.

2. Описание программы

Создадим абстрактный класс *IFigure*, чтобы наследовать от него классы дгугих фигур и иметь возможность хранить разные фигуры вместе. Каждая фигура — шаблонный полиморфный класс, поддерживающий методы родителя. Шаблон фигуры — скалярный тип. Из входных данных трудно определить, какую фигуру вводит пользователь, поэтому каждая фигура будет иметь свой уникальный идентификатор. Класс *TFactory* читает аргументы фигур и создаёт их, возвращает указатели на эти фигуры. Шаблонный класс *TDocument* хранит список указателей на фигуры и стек с действиями. Действия — наследники абстракного класса *IAction*, поддерживающие общий метод *PerformAction*. Шаблон документа — тоже скалярный тип, то есть можно задать этот тип сразу для всех фигур документа. Для чтения из файла и записи в файл используются системные вызовы fread и fwrite.

3. Набор тестов

Программа обрабатывает команды пользователя до окончания ввода и поддерживает следующие команды:

- *«n»* — создание нового документа;
- *«o имя_файла»* — загрузить документ из файла;
- *«s имя_файла»* — сохранить документ в файл;
- *«+ позиция тип_фигуры аргументы_фигуры»* — добавить фигуру с заданными аргументами на заданную позицию;
- *«- позиция»* — удалить фигуру на заданной позиции;
- *«p»* — вывести весь документ;
- *«u»* — отменить предыдущее изменение;
- *«h»* — вывести справочную информацию.

В случае ввода несуществующей команды ничего не происходит.

Тест 1. Проверка функций вставки, удаления и отмены предыдущего действия.

```
+ 0 3 0 0 10 6 3
+ 1 1 1 1 4
+ 1 2 5 5 2 3
p
+ 2 3 -1 6 4 8 1
+ 3 1 -1 -1 2
+ 0 2 -2 -6 6 2
p
- 6
- 4
- 2
p
- 1
- 0
- 0
p
u
u
u
p
u
u
u
p
```

Тест 2. Проверка функций вставки, удаления, чтения из файла и записи в файл.

```
+ 0 3 2 2 10 4 2
+ 0 1 -3 -4 5
+ 2 2 -6 0 4 4
p
s MyDoc.v
- 1
- 1
p
o MyDoc.v
p
+ 4 1 0 0 7
p
s MyDoc.v
- 3
- 2
p
o MyDoc.v
p
```

Тест 3. Комбинация тестов 1 и 2.

```
+ 0 3 -4 -2 4 2 2
p
+ 0 1 4 4 1
p
+ 1 2 -4 4 2 4
p
s MyDoc.v
u
u
+ 0 2 -4 4 2 4
p
o MyDoc.v
u
+ 4 2 8 9 1 2
p
u
- 2
p
u
p
```

4. Результат выполнения программы

Программа выводит результат обработки каждой введённой команды. В случае ошибки будет явно выведено, что пошло не так (удаление из пустого документа, ошибка при открытии файла на чтение или запись).

Тест 1.

```
Printing document:
[1] Trapeze {(0, 0), (2, 3), (8, 3), (10, 0)}
[2] Rectangle {(5, 5), (5, 7), (8, 7), (8, 5)}
[3] Square {(1, 1), (1, 5), (5, 5), (5, 1)}
Printing document:
[1] Rectangle {(-2, -6), (-2, 0), (0, 0), (0, -6)}
[2] Trapeze {(0, 0), (2, 3), (8, 3), (10, 0)}
[3] Rectangle {(5, 5), (5, 7), (8, 7), (8, 5)}
[4] Trapeze {(-1, 6), (1, 7), (5, 7), (7, 6)}
[5] Square {(-1, -1), (-1, 1), (1, 1), (1, -1)}
[6] Square {(1, 1), (1, 5), (5, 5), (5, 1)}
Printing document:
[1] Rectangle {(-2, -6), (-2, 0), (0, 0), (0, -6)}
[2] Rectangle {(5, 5), (5, 7), (8, 7), (8, 5)}
[3] Square {(-1, -1), (-1, 1), (1, 1), (1, -1)}
Printing document:
Printing document:
[1] Rectangle {(-2, -6), (-2, 0), (0, 0), (0, -6)}
[2] Rectangle {(5, 5), (5, 7), (8, 7), (8, 5)}
[3] Square {(-1, -1), (-1, 1), (1, 1), (1, -1)}
Printing document:
[1] Rectangle {(-2, -6), (-2, 0), (0, 0), (0, -6)}
[2] Trapeze {(0, 0), (2, 3), (8, 3), (10, 0)}
[3] Rectangle {(5, 5), (5, 7), (8, 7), (8, 5)}
[4] Trapeze {(-1, 6), (1, 7), (5, 7), (7, 6)}
[5] Square {(-1, -1), (-1, 1), (1, 1), (1, -1)}
[6] Square {(1, 1), (1, 5), (5, 5), (5, 1)}
```

Тест 2.

```
Printing document:
[1] Square {(-3, -4), (-3, 1), (2, 1), (2, -4)}
[2] Trapeze {(2, 2), (5, 4), (9, 4), (12, 2)}
[3] Rectangle {(-6, 0), (-6, 4), (-2, 4), (-2, 0)}
Saved document to MyDoc.v
Printing document:
[1] Rectangle {(-6, 0), (-6, 4), (-2, 4), (-2, 0)}
Loaded document from MyDoc.v
Printing document:
[1] Square {(-3, -4), (-3, 1), (2, 1), (2, -4)}
[2] Trapeze {(2, 2), (5, 4), (9, 4), (12, 2)}
[3] Rectangle {(-6, 0), (-6, 4), (-2, 4), (-2, 0)}
Printing document:
[1] Square {(-3, -4), (-3, 1), (2, 1), (2, -4)}
[2] Trapeze {(2, 2), (5, 4), (9, 4), (12, 2)}
[3] Rectangle {(-6, 0), (-6, 4), (-2, 4), (-2, 0)}
[4] Square {(0, 0), (0, 7), (7, 7), (7, 0)}
Saved document to MyDoc.v
Printing document:
[1] Square {(-3, -4), (-3, 1), (2, 1), (2, -4)}
[2] Square {(0, 0), (0, 7), (7, 7), (7, 0)}
Loaded document from MyDoc.v
Printing document:
[1] Square {(-3, -4), (-3, 1), (2, 1), (2, -4)}
[2] Trapeze {(2, 2), (5, 4), (9, 4), (12, 2)}
[3] Rectangle {(-6, 0), (-6, 4), (-2, 4), (-2, 0)}
[4] Square {(0, 0), (0, 7), (7, 7), (7, 0)}
```

Тест 3.

```
Printing document:
[1] Trapeze {(-4, -2), (-3, 0), (-1, 0), (0, -2)}
Printing document:
[1] Square {(4, 4), (4, 5), (5, 5), (5, 4)}
[2] Trapeze {(-4, -2), (-3, 0), (-1, 0), (0, -2)}
Printing document:
[1] Square {(4, 4), (4, 5), (5, 5), (5, 4)}
[2] Rectangle {(-4, 4), (-4, 6), (0, 6), (0, 4)}
[3] Trapeze {(-4, -2), (-3, 0), (-1, 0), (0, -2)}
Saved document to MyDoc.v
Printing document:
[1] Rectangle {(-4, 4), (-4, 6), (0, 6), (0, 4)}
[2] Trapeze {(-4, -2), (-3, 0), (-1, 0), (0, -2)}
Loaded document from MyDoc.v
Nothing to undo!
Printing document:
[1] Square {(4, 4), (4, 5), (5, 5), (5, 4)}
[2] Rectangle {(-4, 4), (-4, 6), (0, 6), (0, 4)}
[3] Trapeze {(-4, -2), (-3, 0), (-1, 0), (0, -2)}
[4] Rectangle {(8, 9), (8, 10), (10, 10), (10, 9)}
Printing document:
[1] Square {(4, 4), (4, 5), (5, 5), (5, 4)}
[2] Trapeze {(-4, -2), (-3, 0), (-1, 0), (0, -2)}
Printing document:
[1] Square {(4, 4), (4, 5), (5, 5), (5, 4)}
[2] Rectangle {(-4, 4), (-4, 6), (0, 6), (0, 4)}
[3] Trapeze {(-4, -2), (-3, 0), (-1, 0), (0, -2)}
```

5. Листинг программы

Программа разделена на файлы: figure.hpp, square.hpp, rectangle.hpp, trapeze.hpp, factory.hpp, document.hpp, main.cpp. В каждом файле находится реализация соответствующего класса, а в main.cpp обработка команд.

figure.hpp

```
#ifndef FIGURE_HPP
#define FIGURE_HPP

#include <iostream>
#include <tuple>

class IFigure {
public:
  virtual void Print(std::ostream & of) = 0;
  virtual void Write(FILE* out) = 0;
  virtual ~IFigure() {}
};

template<class T1, class T2>
std::ostream & operator << (std::ostream & out, const
std::pair<T1, T2> & p) {
  out << "(" << p.first << ", " << p.second << ")";
  return out;
}

#endif /* FIGURE_HPP */
```

square.hpp

```
#ifndef SQUARE_HPP
#define SQUARE_HPP

#include "figure.hpp"

const unsigned int SQUARE_TYPE_ID = 1;

template<class T>
class TSquare : public IFigure {
private:
  /* Cords of left bottom corner and square side length */
  std::pair<T, T> Cords;
  T Side;
public:
  TSquare() : Cords(), Side() {}
  TSquare(const std::pair<T, T> & xy, const T & l) : Cords(xy),
Side(l) {}

  void Write(FILE* out) override {
     fwrite(&SQUARE_TYPE_ID, sizeof(unsigned int), 1, out);
     fwrite(&Cords.first, sizeof(T), 1, out);
     fwrite(&Cords.second, sizeof(T), 1, out);
     fwrite(&Side, sizeof(T), 1, out);
  }

  void Print(std::ostream & of) override {
     of << *this;
  }

  template<class U>
  friend std::ostream & operator << (std::ostream & of, const
TSquare<U> & sq) {
     of << "Square {";
     of << std::pair<U, U>(sq.Cords.first, sq.Cords.second) << ",
";
     of << std::pair<U, U>(sq.Cords.first, sq.Cords.second +
sq.Side) << ", ";
     of << std::pair<U, U>(sq.Cords.first + sq.Side,
sq.Cords.second + sq.Side) << ", ";
     of << std::pair<U, U>(sq.Cords.first + sq.Side,
sq.Cords.second);
     of << "}";
     return of;
  }
};

#endif /* SQUARE_HPP */
```

rectangle.hpp

```
#ifndef RECTANGLE_HPP
#define RECTANGLE_HPP

#include "figure.hpp"

const unsigned int RECTANGLE_TYPE_ID = 2;

template<class T>
class TRectangle : public IFigure {
private:
  /* Cords of left bottom corner, height and width */
  std::pair<T, T> Cords;
  T Height, Width;
public:
  TRectangle() : Cords(), Height(), Width() {}
  TRectangle(const std::pair<T, T> & xy, const T & h, const T & w)
: Cords(xy), Height(h), Width(w) {}

  void Print(std::ostream & of) override {
     of << *this;
  }

  void Write(FILE* out) override {
     fwrite(&RECTANGLE_TYPE_ID, sizeof(unsigned int), 1, out);
     fwrite(&Cords.first, sizeof(T), 1, out);
     fwrite(&Cords.second, sizeof(T), 1, out);
     fwrite(&Height, sizeof(T), 1, out);
     fwrite(&Width, sizeof(T), 1, out);
  }

  template<class U>
  friend std::ostream & operator << (std::ostream & of, const
TRectangle<U> & rect) {
     of << "Rectangle {";
     of << std::pair<U, U>(rect.Cords.first, rect.Cords.second) <<
", ";
     of << std::pair<U, U>(rect.Cords.first, rect.Cords.second +
rect.Height) << ", ";
     of << std::pair<U, U>(rect.Cords.first + rect.Width,
rect.Cords.second + rect.Height) << ", ";
     of << std::pair<U, U>(rect.Cords.first + rect.Width,
rect.Cords.second);
     of << "}";
     return of;
  }
};

#endif /* RECTANGLE_HPP */
```

trapeze.hpp

```
#ifndef TRAPEZE_HPP
#define TRAPEZE_HPP

#include "figure.hpp"

const unsigned int TRAPEZE_TYPE_ID = 3;

template<class T>
class TTrapeze : public IFigure {
private:
  /* Cords of left bottom corner, greater and smaller base,
trapeze heigth */
  std::pair<T, T> Cords;
  T GreaterBase, SmallerBase, Height;
public:
  TTrapeze() : Cords(), GreaterBase(), SmallerBase(), Height() {}
  TTrapeze(const std::pair<T, T> & xy, const T & gb, const T & sb,
const T & h) : Cords(xy), GreaterBase(gb), SmallerBase(sb),
Height(h) {
     if (SmallerBase > GreaterBase) {
          std::swap(SmallerBase, GreaterBase);
     }
  }

  void Print(std::ostream & of) override {
     of << *this;
  }

  void Write(FILE* out) override {
     fwrite(&TRAPEZE_TYPE_ID, sizeof(unsigned int), 1, out);
     fwrite(&Cords.first, sizeof(T), 1, out);
     fwrite(&Cords.second, sizeof(T), 1, out);
     fwrite(&SmallerBase, sizeof(T), 1, out);
     fwrite(&GreaterBase, sizeof(T), 1, out);
     fwrite(&Height, sizeof(T), 1, out);
  }

  template<class U>
  friend std::ostream & operator << (std::ostream & out, const
TTrapeze<U> & trapeze) {
     T d = (trapeze.GreaterBase - trapeze.SmallerBase) / 2.0;
     out << "Trapeze {";
     out << std::pair<T, T>(trapeze.Cords.first,
trapeze.Cords.second) << ", ";
     out << std::pair<T, T>(trapeze.Cords.first + d,
trapeze.Cords.second + trapeze.Height) << ", ";
     out << std::pair<T, T>(trapeze.Cords.first +
trapeze.SmallerBase + d, trapeze.Cords.second + trapeze.Height) <<
", ";
```

```
        out << std::pair<T, T>(trapeze.Cords.first +
trapeze.GreaterBase, trapeze.Cords.second);
        out << "}";
        return out;
    }
};

#endif /* TRAPEZE_HPP */
```

document.hpp

```
#ifndef DOCUMENT_HPP
#define DOCUMENT_HPP

#include <list>
#include <stack>

#include "factory.hpp"

template<class SCALAR_TYPE>
class TDocument {
private:
    struct IAction;

    using figure_pointer = std::shared_ptr<IFigure>;
    using action_pointer = std::shared_ptr<IAction>;
    using const_iterator = std::list< figure_pointer
>::const_iterator;

    std::list< figure_pointer > FiguresList;
    std::stack< action_pointer > ActionStack;

    struct IAction {
        virtual void PerformAction(TDocument & fact) = 0;
        virtual ~IAction() {}
    };

    class TDeleteAction : public IAction {
    private:
        size_t DeletePos;
    public:
        TDeleteAction(const size_t & pos) : DeletePos(pos) {}
        void PerformAction(TDocument & fact) override {
            fact.Delete(DeletePos);
        }
    };
```

```
  class TAddAction : public IAction {
  private:
     size_t AddPos;
     figure_pointer AddFigure;
  public:
     TAddAction(const size_t & pos, const figure_pointer & fig) :
AddPos(pos), AddFigure(fig) {}
     void PerformAction(TDocument & fact) override {
          fact.AddFigure(AddPos, AddFigure);
     }
  };
public:
  void CreateNew() {
     while (!ActionStack.empty()) {
          ActionStack.pop();
     }
     FiguresList.clear();
  }

  void LoadFromFile(FILE* in) {
     CreateNew();
     size_t n;
     fread(&n, sizeof(size_t), 1, in);
     for (size_t i = 0; i < n; ++i) {
          unsigned int type;
          fread(&type, sizeof(unsigned int), 1, in);
          if (type == SQUARE_TYPE_ID) {
               FiguresList.push_back(TFactory< SCALAR_TYPE,
TSquare<SCALAR_TYPE> >::Read(in));
          } else if (type == RECTANGLE_TYPE_ID) {
               FiguresList.push_back(TFactory< SCALAR_TYPE,
TRectangle<SCALAR_TYPE> >::Read(in));
          } else if (type == TRAPEZE_TYPE_ID) {
               FiguresList.push_back(TFactory< SCALAR_TYPE,
TTrapeze<SCALAR_TYPE> >::Read(in));
          }
     }
  }

  void SaveToFile(FILE* out) {
     size_t n = FiguresList.size();
     fwrite(&n, sizeof(size_t), 1, out);
     for (const_iterator it = FiguresList.begin(); it !=
FiguresList.end(); ++it) {
          (*it)->Write(out);
     }
  }

  void Add(const size_t & pos, const unsigned int & figureID) {
     if (figureID == SQUARE_TYPE_ID) {
```

```
            AddFigure(pos, TFactory<SCALAR_TYPE, TSquare<SCALAR_TYPE>
>::CreateFigure());
        } else if (figureID == RECTANGLE_TYPE_ID) {
            AddFigure(pos, TFactory<SCALAR_TYPE,
TRectangle<SCALAR_TYPE> >::CreateFigure());
        } else if (figureID == TRAPEZE_TYPE_ID) {
            AddFigure(pos, TFactory<SCALAR_TYPE,
TTrapeze<SCALAR_TYPE> >::CreateFigure());
        }
    }

    void AddFigure(const size_t & pos, const figure_pointer & fig) {
        if (pos > FiguresList.size()) {
            FiguresList.push_back(fig);
            TDeleteAction* delAct = new
TDeleteAction(FiguresList.size());
            ActionStack.push(action_pointer(delAct));
        } else {
            size_t cur = 0;
            const_iterator it = FiguresList.begin();
            while (cur < pos) {
                ++cur;
                ++it;
            }
            FiguresList.insert(it, fig);
            TDeleteAction* delAct = new TDeleteAction(pos + 1);
            ActionStack.push(action_pointer(delAct));
        }
    }

    void Delete(const size_t & pos) {
        if (FiguresList.empty()) {
            std::cout << "Nothing to delete!" << std::endl;
            return;
        }
        if (pos > FiguresList.size()) {
            TAddAction* addAct = new TAddAction(FiguresList.size() -
1, FiguresList.back());
            ActionStack.push(action_pointer(addAct));
            FiguresList.pop_back();
        } else {
            size_t cur = 1;
            const_iterator it = FiguresList.begin();
            while (cur < pos) {
                ++cur;
                ++it;
            }
            TAddAction* addAct = new TAddAction(cur - 1, *it);
            ActionStack.push(action_pointer(addAct));
            FiguresList.erase(it);
        }
```

```
    }

    void Undo() {
        if (ActionStack.empty()) {
            std::cout << "Nothing to undo!" << std::endl;
        } else {
            ActionStack.top()->PerformAction(*this);
            ActionStack.pop();
            ActionStack.pop();
        }
    }

    template<class U>
    friend std::ostream & operator << (std::ostream & of, const
TDocument<U> & fact) {
        TDocument::const_iterator it = fact.FiguresList.begin();
        for (size_t i = 0; i < fact.FiguresList.size(); ++i) {
            of << "[" << i + 1 << "] ";
            (*it)->Print(of);
            of << std::endl;
            ++it;
        }
        return of;
    }
};

#endif /* DOCUMENT_HPP */
```

factory.hpp

```
#ifndef FACTORY_HPP
#define FACTORY_HPP
#include <memory>
#include "rectangle.hpp"
#include "square.hpp"
#include "trapeze.hpp"

template<class T, class FIGURE>
class TFactory;

template<class T>
class TFactory< T, TSquare<T> > {
public:
  static std::shared_ptr<IFigure> CreateFigure() {
     std::pair<T, T> curCords;
     T curSide;
     std::cout << "Input square as follows: x y a" << std::endl;
     std::cout << "x, y is a left bottom corner cords" <<
std::endl;
     std::cout << "a is square side" << std::endl;
     std::cin >> curCords.first >> curCords.second >> curSide;
     TSquare<T>* sq = new TSquare<T>(curCords, curSide);
     return std::shared_ptr<IFigure>(sq);
  }

  static std::shared_ptr<IFigure> Read(FILE* in) {
     std::pair<T, T> curCords;
     T curSide;
     fread(&curCords.first, sizeof(T), 1, in);
     fread(&curCords.second, sizeof(T), 1, in);
     fread(&curSide, sizeof(T), 1, in);
     TSquare<T>* sq = new TSquare<T>(curCords, curSide);
     return std::shared_ptr<IFigure>(sq);
  }
};

template<class T>
class TFactory< T, TRectangle<T> > {
public:
  static std::shared_ptr<IFigure> CreateFigure() {
     std::pair<T, T> curCords;
     T curHeight, curWidth;
     std::cout << "Input rectangle as follows: x y a b" <<
std::endl;
     std::cout << "x, y is a left bottom corner cords" <<
std::endl;
     std::cout << "a and b are width and heigth" << std::endl;
     std::cin >> curCords.first >> curCords.second >> curHeight >>
curWidth;
```

```
      TRectangle<T>* rect = new TRectangle<T>(curCords, curHeight,
curWidth);
      return std::shared_ptr<IFigure>(rect);
    }

    static std::shared_ptr<IFigure> Read(FILE* in) {
      std::pair<T, T> curCords;
      T curHeight, curWidth;
      fread(&curCords.first, sizeof(T), 1, in);
      fread(&curCords.second, sizeof(T), 1, in);
      fread(&curHeight, sizeof(T), 1, in);
      fread(&curWidth, sizeof(T), 1, in);
      TRectangle<T>* rect = new TRectangle<T>(curCords, curHeight,
curWidth);
      return std::shared_ptr<IFigure>(rect);
    }
};

template<class T>
class TFactory< T, TTrapeze<T> > {
public:
    static std::shared_ptr<IFigure> CreateFigure() {
      std::pair<T, T> curCords;
      T curGreaterBase, curSmallerBase, curHeight;
      std::cout << "Input trapeze as follows: x y a b c" <<
std::endl;
      std::cout << "x, y is a left bottom corner cords" <<
std::endl;
      std::cout << "a, b and c are larger, smaller base and height"
<< std::endl;
      std::cin >> curCords.first >> curCords.second >>
curGreaterBase >> curSmallerBase >> curHeight;
      TTrapeze<T>* trap = new TTrapeze<T>(curCords, curGreaterBase,
curSmallerBase, curHeight);
      return std::shared_ptr<IFigure>(trap);
    }

    static std::shared_ptr<IFigure> Read(FILE* in) {
      std::pair<T, T> curCords;
      T curGreaterBase, curSmallerBase, curHeight;
      fread(&curCords.first, sizeof(T), 1, in);
      fread(&curCords.second, sizeof(T), 1, in);
      fread(&curGreaterBase, sizeof(T), 1, in);
      fread(&curSmallerBase, sizeof(T), 1, in);
      fread(&curHeight, sizeof(T), 1, in);
      TTrapeze<T>* trap = new TTrapeze<T>(curCords, curGreaterBase,
curSmallerBase, curHeight);
      return std::shared_ptr<IFigure>(trap);
    }
};
#endif /* FACTORY_HPP */
```

main.cpp

```
#include "document.hpp"

/*
 * Инютин М А М8О-207Б-19
 * 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.
 */

int main() {
	TDocument<int> doc;
	std::string s;
	while (std::cin >> s) {
		if (s == "n") {
			doc.CreateNew();
			std::cout << "Created new document" << std::endl;
		} else if (s == "o") {
			std::cin >> s;
			FILE* in = fopen(s.c_str(), "rb");
			if (in == NULL) {
				std::cout << "No such file on directory" <<
std::endl;
			} else {
				doc.LoadFromFile(in);
				std::cout << "Loaded document from " << s <<
std::endl;
				fclose(in);
			}
		} else if (s == "s") {
			std::cin >> s;
			FILE* out = fopen(s.c_str(), "wb");
			if (out == NULL) {
				std::cout << "No such file on directory" <<
std::endl;
```

```
                } else {
                    doc.SaveToFile(out);
                    std::cout << "Saved document to " << s <<
std::endl;
                    fclose(out);
                }
            } else if (s == "+") {
                size_t pos;
                unsigned short type;
                std::cin >> pos >> type;
                doc.Add(pos, type);
            } else if (s == "-") {
                size_t pos;
                std::cin >> pos;
                doc.Delete(pos);
            } else if (s == "p") {
                std::cout << "Printing document:" << std::endl;
                std::cout << doc;
            } else if (s == "u") {
                doc.Undo();
            } else if (s == "h") {
                std::cout << "\'n\' - create new document" <<
std::endl;
                std::cout << "\'o\' - open document" << std::endl;
                std::cout << "\'s\' - save document" << std::endl;
                std::cout << "\'+\' - add a figure" << std::endl;
                std::cout << "\'-\' - remove a figure" << std::endl;
                std::cout << "\'p\' - print document" << std::endl;
                std::cout << "\'u\' - undo changes" << std::endl;
                std::cout << "\'h\' - show this message" <<
std::endl;
            } else {
                std::cout << "Unknown command. Type \'h\' to show
help" << std::endl;
            }
        }
        return 0;
}
```

6. Выводы

В ходе выполнения лабораторной работы я спроектировал систему классов для графического редактора, изучил различные способы записи структуры классов в файл и реализовал один из них. В жизни важно организовать рабочее пространство, это касается и программирования. Правильная структура программы и классов улучшает читаемость кода и облгегчает его дальнейшую поддержку.

Список литературы

1. fread — C++ Reference — Cplusplus.com
   URL: http://www.cplusplus.com/reference/cstdio/fread/
   (дата обращения 05.12.2020)
2. fwrite — C++ Reference — Cplusplus.com
   URL: http://www.cplusplus.com/reference/cstdio/fwrite/
   (дата обращения 05.12.2020)