МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВЦИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЬЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №7

по курсу объектно-ориентированное программирование I семестр, 2019/20 уч. год

Студент *Артемьев Дмитрий Иванович, группа М8О-206Б-18*

Преподаватель *Журавлёв Андрей Андреевич*

Условие

Спроектировать простейший графический векторный редактор.

Требование к функционалу редактора:

- создание нового документа
- импорт документа из файла
- экспорт документа в файл
- создание графического примитива
- удаление графического примитива
- отображение документа на экране
- реализовать операцию undo

Требование к реализации:

- создание графических примитивов необходимо вынести в отдельный класс - Factory
- сделать упор на использование полиморфизма при работе с фигурами
- взаимодействие с пользователем реализовать в функции main

Описание программы

Исходный код лежит в 8 файлах:

1. src/main.cpp: основная интерактивная программа, возможность работать со стеком
2. include/figures/figure.hpp: абстрактный класс фигуры
3. include/figures/triangle.hpp: описание и реализация класса треугольника
4. include/figures/rectangle.hpp: описание и реализация класса прямоугольника
5. include/figures/square.hpp: описание и реализация класса квадрата
6. include/figures/point.hpp: описание и реализация класса точки
7. include/command.cpp: реализация методов классов команд
8. include/command.hpp: описание классов команд
9. include/document.cpp: реализация методов класса документа

10. include/document.hpp: описание класса документа
11. include/editor.cpp: реализация методов класса редактор
12. include/editor.hpp: описание класса редактор
13. include/factory.cpp: реализация методов класса фабрика
14. include/factory.hpp: описание класса фабрика

Дневник отладки

Проблемы с линковкой функции dist из файла point.hpp.

Недочёты

Не реализован графический интерфейс.

Выводы

Я получил практические навыки в проектировании структуры классов приложения.

Исходный код

src/main.cpp

```cpp
#include <iostream>

#include "editor.hpp"
#include "factory.hpp"

void help() {
    std::cout << "create <document_name>\n"
                 "add <figure_name> <points>\n"
                 "remove <figure_index>\n"
                 "undo\n"
                 "load <filename>\n"
                 "save <filename>\n"
                 "print\n"
                 "exit\n";
}

void create(bad::Editor& editor) {
    std::string name;
    std::cin >> name;
    try {
        editor.CreateDocument(name);
        std::cout << "Document created" << std::endl;
    } catch (...) {
        std::cout << "Error: Creating document";
    }
}

void add(bad::Editor& editor, bad::Factory& factory) {
    try {
        std::shared_ptr<bad::Figure> newFigure = factory.FigureCreate(std::cin);
        editor.InsertFigure(newFigure);
        std::cout << "Ok" << std::endl;
    } catch (std::logic_error& e) {
        std::cout << "Error: " << e.what();
    }
}

void remove(bad::Editor& editor) {
    int index;
```

```
    std::cin >> index;
    try {
        editor.DeleteFigure(index);
        std::cout << "Ok" << std::endl;
    } catch (std::logic_error& e) {
        std::cout << "Error: " << e.what();
    }
}

void undo(bad::Editor& editor) {
    try {
        editor.Undo();
        std::cout << "Ok" << std::endl;
    } catch (std::logic_error& e) {
        std::cout << "Error: " << e.what();
    }
}

void load(bad::Editor& editor) {
    std::string name;
    std::cin >> name;
    try {
        editor.LoadDocument(name);
        std::cout << "Document " << name << " loaded" << std::endl;
    } catch (std::runtime_error& e) {
        std::cout << "Error: " << e.what();
    }
}

void save(bad::Editor& editor) {
    std::string name;
    std::cin >> name;
    try {
        editor.SaveDocument(name);
        std::cout << "Document " << name << " saved" << std::endl;
    } catch (std::runtime_error& e) {
        std::cout << "Error: " << e.what();
    }
}

void print(bad::Editor& editor) {
    editor.PrintDocument(std::cout);
```

```
}

int main() {
    bad::Editor editor;
    bad::Factory factory;

    std::cout << "Enter help to find out how it works" << std::endl;

    while (std::cin) {
        std::string command;
        std::cin >> command;

        if (command == "create") {
            create(editor);
        }
        else if (command == "add") {
            add(editor, factory);
        }
        else if (command == "remove") {
            remove(editor);
        }
        else if (command == "undo") {
            undo(editor);
        }
        else if (command == "load") {
            load(editor);
        }
        else if (command == "save") {
            save(editor);
        }
        else if (command == "print") {
            print(editor);
        }
        else if (command == "help") {
            help();
        }
        else if (command == "exit") {
            break;
        }
        else {
            std::cout << "Wrong command" << std::endl;
        }
```

```
    }
    return 0;
}
```

include/figures/figure.hpp

```cpp
#pragma once

#include <iostream>
#include "point.hpp"

namespace bad {

struct Figure {
public:
    virtual void print(std::ostream& os) const = 0;
    virtual ~Figure() = default;
};

}; // namespace bad
```

include/figures/point.hpp

```cpp
#pragma once

#include <iostream>

namespace bad {

struct Point {
public:
    void Read(std::istream& is) {
        is >> x >> y;
    }

    void Print(std::ostream& os) const {
        os << x << ' ' << y << ' ';
    }

    int x, y;
};

inline int dist(const Point& a, const Point& b) {
    return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
}

}; // namespace bad
```

include/figures/rectangle.hpp

```cpp
#pragma once

#include <iostream>
#include "figure.hpp"

namespace bad {

struct Rectangle : Figure{
public:
    Rectangle() = default;
    Rectangle(std::istream& is) {
        for (int i = 0; i < NUM; i++) {
            pts[i].Read(is);
        }
        if (dist(pts[0], pts[2]) == dist(pts[1], pts[3]) &&
            dist(pts[0], pts[1]) == dist(pts[2], pts[3]) &&
            dist(pts[1], pts[2]) == dist(pts[0], pts[3])) ;
        else
            throw std::logic_error("It is NOT rectangle\n");
    }

    void print(std::ostream& os) const override {
        os << "Rectangle\n";
        for (int i = 0; i < NUM; i++) {
            pts[i].Print(os);
        }
        os << std::endl;
    }
private:
    static const int NUM = 4;
    Point pts[NUM];
};

}; // namespace bad
```

include/figures/square.hpp

```cpp
#pragma once

#include <iostream>
#include "figure.hpp"

namespace bad {

struct Square : Figure{
public:
    Square() = default;
    Square(std::istream& is) {
        for (int i = 0; i < NUM; i++) {
            pts[i].Read(is);
        }
        if (dist(pts[0], pts[2]) == dist(pts[1], pts[3]) &&
            dist(pts[0], pts[1]) == dist(pts[2], pts[3]) &&
            dist(pts[1], pts[2]) == dist(pts[0], pts[3]) &&
            dist(pts[0], pts[1]) == dist(pts[1], pts[2])) ;
        else
            throw std::logic_error("It is NOT square\n");
    }

    void print(std::ostream& os) const override {
        os << "Square\n";
        for (int i = 0; i < NUM; i++) {
            pts[i].Print(os);
        }
        os << std::endl;
    }
private:
    static const int NUM = 4;
    Point pts[NUM];
};

}; // namespace bad
```

include/figures/triangle.hpp

```cpp
#pragma once

#include <iostream>
#include "figure.hpp"

namespace bad {

struct Triangle : Figure {
public:
    Triangle() = default;
    Triangle(std::istream& is) {
        for (int i = 0; i < NUM; i++) {
            pts[i].Read(is);
        }
    }

    void print(std::ostream& os) const override {
        os << "Triangle\n";
        for (int i = 0; i < NUM; i++) {
            pts[i].Print(os);
        }
        os << std::endl;
    }
private:
    static const int NUM = 3;
    Point pts[NUM];
};

}; // namespace bad
```

include/command.hpp

```
#pragma once

namespace bad {
struct Command;
}

#include "document.hpp"
#include "figure.hpp"

namespace bad {

struct Command {
public:
    virtual void Undo() = 0;
    virtual ~Command() = default;
protected:
    Document* document_;
};

struct InsertCommand : public Command{
public:
    InsertCommand(Document* document) {
        document_ = document;
    };

    void Undo() override;
};

struct DeleteCommand : public Command {
public:
    DeleteCommand(std::shared_ptr<Figure>& figure, int index, Document* document) :
        figure_(figure),
        index_(index) {
        document_ = document;
    };


    void Undo() override;
private:
    std::shared_ptr<Figure> figure_;
    int index_;
```

```
};

}; // namespace bad
```

include/command.cpp

```cpp
#include "command.hpp"

namespace bad{

void InsertCommand::Undo() {
    document_->Erase();
}

void DeleteCommand::Undo() {
    document_->Insert(figure_, index_);
}

}; //namespace bad
```

include/document.hpp

```cpp
#pragma once

#include <iostream>
#include <stack>
#include <vector>
#include <memory>

namespace bad {
struct Document;
}

#include "command.hpp"
#include "factory.hpp"
#include "figure.hpp"

namespace bad {

struct Document {
public:

    Document(const std::string& name) :
        documentName_(name)
    {}

    void Insert(std::shared_ptr<Figure>& ptr);
    void Insert(std::shared_ptr<Figure>& ptr, int index);

    void Erase(int index);
    void Erase();

    void Save(const std::string& name);
    void Load(const std::string& name);
    void Print(std::ostream& os) const;

    void Undo();

    int Size() {
        return buffer_.size();
    }

private:
```

```
    std::string documentName_;
    Factory factory_;
    std::vector<std::shared_ptr<Figure>> buffer_;
    std::stack<std::unique_ptr<Command>> history_;
};

}; // namespace bad
```

include/document.cpp

```cpp
#include "document.hpp"
#include <fstream>

namespace bad {

void Document::Insert(std::shared_ptr<Figure>& ptr) {
    std::unique_ptr<Command> command = std::unique_ptr<Command>
        (new InsertCommand(this));
    history_.push(std::move(command));

    buffer_.push_back(ptr);
}

void Document::Insert(std::shared_ptr<Figure>& ptr, int index) {
    buffer_.insert(buffer_.begin() + index, ptr);
}

void Document::Erase(int index) {
    std::shared_ptr<Figure> tmp = buffer_[index];
    std::unique_ptr<Command> command = std::unique_ptr<Command>
        (new DeleteCommand(tmp, index, this));
    //std::cout << "PUSHED ";
    //tmp->print(std::cout);
    history_.push(std::move(command));

    buffer_.erase(buffer_.begin() + index);
}

void Document::Erase() {
    buffer_.pop_back();
}

void Document::Load(const std::string& name) {
    std::ifstream fis(name);
    if (!fis.is_open()) {
        throw std::runtime_error("File is not opened\n");
    }
    documentName_ = name;
    int size_buf;
    fis >> size_buf;
    buffer_.resize(size_buf);
```

```
    for (int i = 0; i < buffer_.size(); i++) {
        buffer_[i] = factory_.FigureCreate(fis);
    }
}

void Document::Save(const std::string& name) {
    std::ofstream fos(name);
    if (!fos.is_open()) {
        throw std::runtime_error("File is not opened\n");
    }
    fos << buffer_.size() << std::endl;
    for (auto elem : buffer_) {
        elem->print(fos);
    }
}

void Document::Print(std::ostream& os) const {
    if (buffer_.empty()) {
        os << "Buffer is empty" << std::endl;
        return;
    }
    for (const auto& figure : buffer_) {
        figure->print(os);
    }
}

void Document::Undo() {
    if (history_.empty()) {
        throw std::logic_error("No further undo information\n");
    }
    std::unique_ptr<Command> last = std::move(history_.top());
    last->Undo();
    history_.pop();
}

}; // namespace bad
```

include/editor.hpp

```cpp
#pragma once

#include <iostream>
#include <memory>

#include "document.hpp"

namespace bad {

struct Editor {
public:
    Editor() :
        document_(nullptr)
    {};

    void CreateDocument(const std::string& name);
    void LoadDocument(const std::string& name);
    void SaveDocument(const std::string& name);
    void PrintDocument(std::ostream& os) const;

    void InsertFigure(std::shared_ptr<Figure>& newFigure);
    void DeleteFigure(int index);

    void Undo();

    bool DocumentExists();

    ~Editor() = default;
private:
    std::unique_ptr<Document> document_;
};

}; // namespace bad
```

include/editor.cpp

```cpp
#include "editor.hpp"

namespace bad {

void Editor::CreateDocument(const std::string& name) {
    document_ = std::unique_ptr<Document>( new Document(name));
}

void Editor::LoadDocument(const std::string& name) {
    try {
        document_ = std::unique_ptr<Document>( new Document(name));
        document_->Load(name);
    }
    catch (std::logic_error& e) {
        throw e;
    }
}

void Editor::SaveDocument(const std::string& name) {
    if (document_ == nullptr) {
        throw std::logic_error("There is no document");
        return;
    }
    document_->Save(name);
}

void Editor::PrintDocument(std::ostream& os) const {
    if (document_ == nullptr) {
        throw std::logic_error("There is no document");
        return;
    }
    document_->Print(os);
}

void Editor::InsertFigure(std::shared_ptr<Figure>& newFigure) {
    if (document_ == nullptr) {
        throw std::logic_error("There is no document");
        return;
    }
    document_->Insert(newFigure);
}
```

```
void Editor::DeleteFigure(int index) {
    if (document_ == nullptr) {
        throw std::logic_error("There is no document");
        return;
    }
    if (index > document_->Size() || index < 0) {
        throw std::logic_error("Out of bound");
    }
    document_->Erase(index);
}

void Editor::Undo() {
    try {
        document_->Undo();
    } catch(std::logic_error& e) {
        throw e;
    }
}

bool Editor::DocumentExists() {
    return document_ != nullptr;
}

}; // namespace bad
```

include/factory.hpp

```cpp
#pragma once

#include <iostream>
#include <memory>

#include "figure.hpp"
#include "triangle.hpp"
#include "rectangle.hpp"
#include "square.hpp"

namespace bad {

class Factory {
public:
    std::shared_ptr<Figure> FigureCreate(std::istream& is);
};

}; // namespace bad
```

include/factory.cpp

```cpp
#include "factory.hpp"
#include <algorithm>

namespace bad {

std::shared_ptr<Figure> Factory::FigureCreate(std::istream& is) {
    std::string figureName;
    is >> figureName;
    std::transform(figureName.begin(), figureName.end(), figureName.begin(), ::tolower);
    std::shared_ptr<Figure> figure;
    if (figureName == "triangle") {
        figure = std::shared_ptr<Figure>(new Triangle(is));
    }
    else if (figureName == "rectangle") {
        try {
            figure = std::shared_ptr<Figure>(new Rectangle(is));
        } catch (std::logic_error& e) {
            throw e;
        }
    }
    else if (figureName == "square") {
        try {
            figure =  std::shared_ptr<Figure>(new Square(is));
        } catch (std::logic_error& e) {
            throw e;
        }
    }
    else {
        throw std::logic_error("There is no such figure\n");
    }
    return figure;
}

}; // namespace bad
```