Министерство науки и высшего образования РФ
ФГАОУ ВО ЮЖНО-УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИУ)
Институт естественных и точных наук

Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования

«Векторный графический редактор»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

по дисциплине «Объектно-ориентированное программирование»

ЮУрГУ–01.03.02.2020.212.ПЗ КР

Руководитель,

_____*Демидов А.К.*

«___» _____ 2020г.

Автор работы:

Студент группы: ЕТ – 212

_____*Утков А.С.*

«___» _____ 2020г.

Работа защищена с оценкой

_____

«___» _____2020 г.

Челябинск – 2020

АННОТАЦИЯ

Утков А.С. Векторный графический редактор. – Челябинск: ЮУрГУ, ЕТ-212, 2020. – 31 с., 8 ил., библиографический список – 2 наим., 1 прил.

В курсовой работе описывается разработка векторного графического редактора с помощью объектно-ориентированного подхода. Работа содержит результаты объектно-ориентированного анализа и проектирования, инструкции по использованию программы.

В результате работы был разработан векторный графический редактор, код которого приводится в приложении А.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы. Объектно-ориентированный подход является наиболее прогрессивной технологией разработки программных систем, позволяет разрабатывать более сложные системы.
Цель работы – разработать векторный графический редактор.
Задачи работы:
– изучить приемы объектно-ориентированного анализа;
– научиться разрабатывать программы в объектно-ориентированном стиле;
– овладеть технологиями объектно-ориентированного анализа и проектирования;
– изучить концепции объектно-ориентированного программирования;
– изучить особенности объектной модели языка программирования C++;
– научиться самостоятельно и творчески использовать знания и полученные практические навыки;
– овладеть навыками самостоятельного получения новых знаний по теории и практике объектного подхода в программировании.

Объект работы – векторный графический редактор
Предмет работы – применение объектно-ориентированного подхода для разработки программы.
Результаты работы можно использовать в процессе последующего обучения в соответствии с учебным планом подготовки бакалавров по направлению «Прикладная математика и информатика»

1 ПОСТАНОВКА ЗАДАЧИ

Необходимо разработать упрощенный векторный графический редактор Corel Draw v0.0 со следующими возможностями:
– рисование линий;
– рисование прямоугольников (заполненных и нет);
– выбор цветов рисования и заполнения из 16 или более;
– чтение и запись рисунка в собственном формате;
– удаление объектов.
При движении мыши высвечивать координаты.

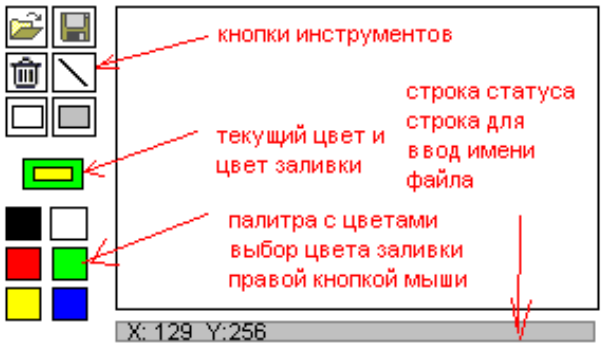

Рисунок 1.1 – Примерный интерфейс программы

Анализ предметной области выявляет следующие объекты:
– кнопки управления, имеющие квадратную форму и отличающиеся изображением или цветом;
– объект, отображающий текущие цвета рисования;
– объект, отображающий текущие координаты мыши и вводимое имя файла при загрузке и сохранении;
– объект “Линия”, позволяющий создать на поле для рисования прямую линию с возможностью её удаления;
– объект “Прямоугольник”, позволяющий создать на поле для рисования прямоугольник с возможностью его удаления;
– объект “Закрашенный прямоугольник”, позволяющий создать на поле для рисования закрашенный прямоугольник с возможностью его удаления;
– поле для рисования, в котором происходит непосредственно работа с объектами “Линия”, “Прямоугольник” и “Закрашенный прямоугольник”.
Кнопки управления по поведению можно разделить на 4 группы, различающиеся по поведению:
– 16 кнопок выбора цвета, изменяющие состояние объекта, отображающего текущие цвета рисования;
– 3 кнопки создания новой фигуры;
– 1 кнопка удаления с поля текущей выбранной фигуры;
– 2 кнопки загрузки и сохранения рисунка.

Требования к программе:
– графический интерфейс;
– работа с мышью.

2 ОПИСАНИЕ ПРОГРАММЫ

2.1 Для разработки программы были использованы:
– компилятор Visual C++ 2019

2.2 Программа состоит из 3 модулей:

1. Модуль **main** (файл main.cpp) содержит функцию int main().
2. Модуль **button** (интерфейсная часть в файле button.hpp, реализация в файле button.cpp) содержит пять классов:

```
class Button : public ControlObject { // Базовый класс для кнопок управления
public:
     // конструктор получает координаты левого верхнего угла x,y
     Button(int x, int y) :ControlObject(x, y, x + 31, y + 31) {}
};
class ColorButton : public Button { // Класс для кнопок выбора цвета
     int c; // цвет кнопки
public:
     // конструктор получает координаты x,y и номер цвета c
     ColorButton(int x, int y, int c) :Button(x, y), c(c) {}
     void draw(); // нарисовать объект
     void press(); // реакция на нажатие
};
class IconButton : public Button { // Класс для кнопок с картинкой
     IMAGE* image; // загруженная картинка для кнопки
public:
     // конструктор получает координаты x,y, имя файла с картинкой
     IconButton(int x, int y, string icon) :Button(x, y),
          image(loadBMP(icon.c_str())) {}
     ~IconButton() { freeimage(image); } // деструктор
     void draw(); // нарисовать объект
};
class ToolButton : public IconButton { // Класс для кнопок выбора инструмента
     fun_ptr tool; // инструмент
public:
     // конструктор получает координаты x,y, имя файла с картинкой
     // icon и указатель на функцию-инструмент t
     ToolButton(int x, int y, string icon, fun_ptr t)
          :IconButton(x, y, icon), tool(t) {}
     void press(); // реакция на нажатие
};
```

```
class CommandButton : public IconButton { // Класс для кнопок-
команд
     fun_ptr action; // действие
public:
     // конструктор получает координаты x,y, имя файла с картинкой
     // icon и указатель на действие a
     CommandButton(int x, int y, string icon, fun_ptr a)
          :IconButton(x, y, icon), action(a) {}
     void press(); // реакция на нажатие
};
void delete_tool(); // инструмент для удаления
void line_tool(); // инструмент линия
void rect_tool(); // инструмент прямоугольник
void colloredrect_tool(); // инструмент закрашенный прямоугольник
void load_command(); // загрузка рисунка
void save_command(); // сохранение рисунка
```

3. Модуль **interface** (интерфейсная часть в файле interface.hpp, реализация в файле interface.cpp) содержит девять классов:

```
class DrawObject { // класс для объектов на экране
protected:
     int x1, y1, x2, y2; // координаты углов прямоугольника, в
                         котором находится объект
public:
     DrawObject(int x1, int y1, int x2, int y2) :x1(x1), y1(y1),
          x2(x2), y2(y2) {} // конструктор получает координаты ле-
вого верхнего угла x1,y1 и координаты правого нижнего угла x2,y2
     virtual ~DrawObject() {} // деструктор
     bool in(int x, int y); // координаты x, y внутри объекта
     int get_top() { return y1 + 1; } // координаты
     int get_left() { return x1 + 1; } // верхнего угла
     int get_height() { return y2 - y1 - 1; } // высота
     int get_width() { return x2 - x1 - 1; } // ширина
     virtual void draw() = 0; // нарисовать объект
};
class SelectedColors : public DrawObject { // класс для
отображения текущих цветов рисования
     int fc, // цвет рисования
         bc; // цвет закраски
     // конструктор получает координаты углов x1,y1,x2,y2
     SelectedColors(int x1, int y1, int x2, int y2) :
          DrawObject(x1, y1, x2, y2), fc(BLACK), bc(WHITE) {}
public:
     static SelectedColors& instance(); // объект для текущих
цветов
     void draw(); // нарисовать объект
     void set_fc(int c); // установить цвет рисования
     int get_fc() { return fc; } // получить  цвет рисования
     void set_bc(int c); // установить цвет закраски
     int get_bc() { return bc; } // получить цвет закраски
};
```

```
class StatusLine : public DrawObject { // класс для отображения
текущих координат мыши и имени файла
     int x, y; // координаты мыши
     bool edit;
     string name; // имя файла
     // конструктор получает координаты углов x1,y1,x2,y2
     StatusLine(int x1, int y1, int x2, int y2) :
          DrawObject(x1, y1, x2, y2), x(0), y(0), name("new.vec")
          {}
public:
     static StatusLine& instance(); // строка состояния
     void draw(); // нарисовать объект
     void set_xy(int x, int y); // изменить текущие координаты
     void set_name(const string& name); // установить имя файла
     string get_name() { return name; } // получить имя файла
     void edit_name(); // ввести имя файла
};
class ControlObject : public DrawObject { // класс для объектов,
реагирующих на мышь
public:
     ControlObject(int x1, int y1, int x2, int y2)
          :DrawObject(x1, y1, x2, y2) {} // конструктор получает
     координаты углов x1,y1,x2,y2
     virtual void press() = 0; // реакция на нажатие
};
typedef void (*fun_ptr)(); // вспомогательный тип - указатель на
                                   функцию
class Figure {
     bool visible = 0; // видимость
protected:
     int x_1, y_1, //точки для построения
         x_2, y_2;
     int c; //цвет
public:
     Figure(int x_1, int y_1, int x_2, int y_2, int c) :
          x_1(x_1), y_1(y_1), x_2(x_2), y_2(y_2), c(c) {}
     virtual ~Figure() {}
     int get_color() const { return c; } //получить цвет
     void hide(); //спрятать
     void show(); //показать фигуру
     bool isvisible() const { return visible; } //видима?
     virtual void get_data(int& x1, int& y1, int& x2, int& y2,
          int& c); // получение точек и цветов фигуры
     virtual void draw() = 0; // метод для отрисовки
};
```

```
class PaintArea : public ControlObject { // класс для поля
рисования
     vector <Figure*> storage; // хранилище фигур
     fun_ptr tool; // текущий инструмент
     // конструктор получает координаты углов x1,y1,x2,y2
     PaintArea(int x1, int y1, int x2, int y2) :
          ControlObject(x1, y1, x2, y2), tool(nullptr) {}
public:
     static PaintArea& instance(); // поле для рисования
     void draw(); // нарисовать само поле
     void create_figure(string type, int x_1, int y_1, int x_2,
          int y_2, int c, int inner_c = -1); // создание фигуры
     void press(); // реакция на нажатие
     void set_tool(fun_ptr t) { tool = t; }; // установить
инструмент рисования
     fun_ptr get_tool() { return tool; }; // получить текущий
инструмент рисования
     void save(const string& name); // сохранить изображение
     void load(const string& name); // загрузить изображение
     Figure* get_figure(int i) { return storage[i]; }; // доступ к
i-ой фигуре
     int figure_amount() { return storage.size(); }; // возвращает
кол-во фигур
     void delete_figure(int i) { storage.erase(storage.begin() +
          i); }; // удалить фигуру из вектора
     void refresh(); // обновить содержимое экрана
};
class Rect : public Figure {
public:
     Rect(int x_1, int y_1, int x_2, int y_2, int c)
          :Figure(x_1, y_1, x_2, y_2, c) {}
     ~Rect() { hide(); }
     void draw(); // метод для отрисовки
};
class ColoredRect : public Rect {
protected:
     int inner_c; // цвет заливки
public:
     ColoredRect(int x_1, int y_1, int x_2, int y_2, int c,
          int inner_c)
          :Rect(x_1, y_1, x_2, y_2, c), inner_c(inner_c) {}
     ~ColoredRect() { hide(); }
     void get_data(int& x_1, int& y_1, int& x_2, int& y_2, int& c,
          int& inner_c); // получение точек и цветов закрашенного
          прямоугольника
     void draw(); // метод для отрисовки
};
```

```
class Line : public Figure { // класс для отрезка
public:
     Line(int x_1, int y_1, int x_2, int y_2, int c)
          :Figure(x_1, y_1, x_2, y_2, c) {}
     ~Line() { hide(); }
     void draw(); // метод для отрисовки
};
```

Иерархия классов показана на рисунке 2.1:

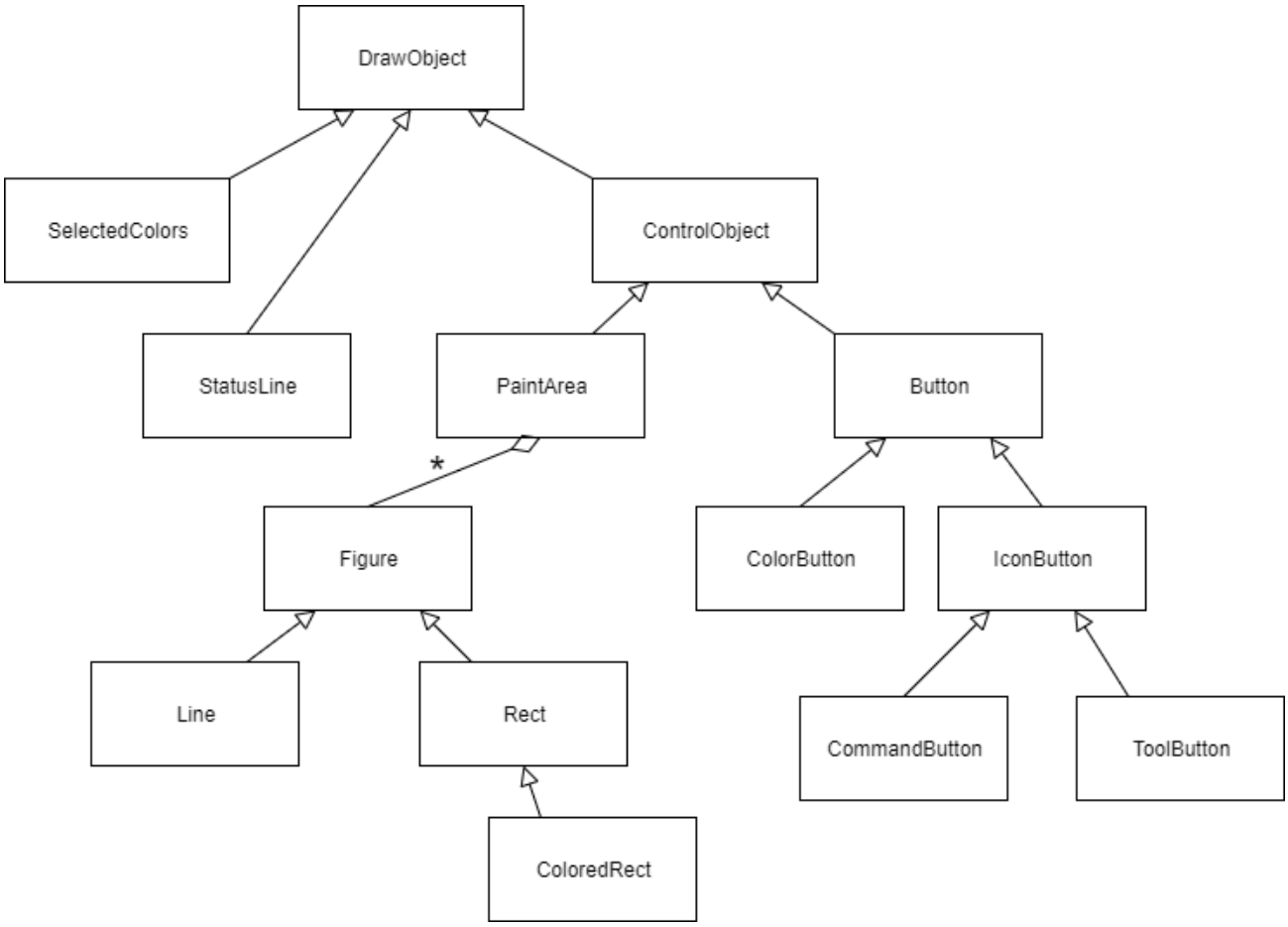

Рисунок 2.1 – Иерархия классов

2.3 Используемые внешние файлы

Программа использует изображения из папки res для отрисовки кнопок. Папка должна находиться в папке с программой.

3 ИНСТРУКЦИЯ ПО УСТАНОВКЕ И ТРЕБОВАНИЯ К СИСТЕМЕ

3.1 Требования к компьютеру:

Процессор: Intel Pentium 4 1ГГц или выше
Память: 512+ МБ
Видеокарта: разрешение 800x600 или выше, 24-битный цвет
Свободное место на диске: 10 Мб
Операционная система: Windows 10
Дополнительное устройство: мышь

3.2 Для установки скопировать на диск файл vector.exe и папку res со следующим содержимым: delete.bmp, frect.bmp, line.bmp, load.bmp, rect.bmp, save.bmp.

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

После запуска программы открывается рабочее окно программы (Рисунок 4.1).

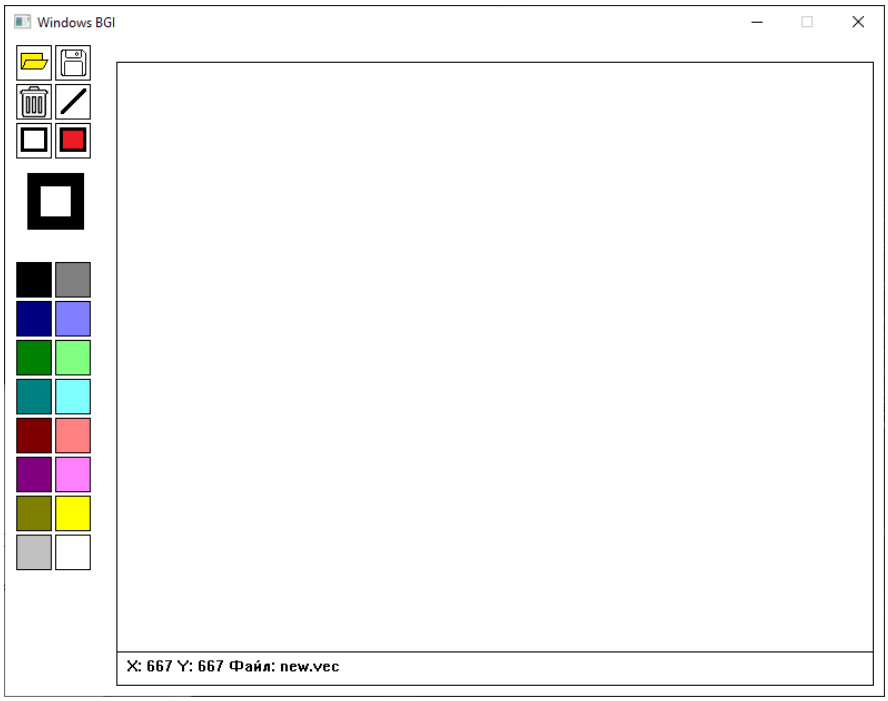

Рисунок 4.1 – Окно редактора.

При нажатии левой или правой кнопкой мыши на кнопки с цветами, этот цвет будет установлен как основной или дополнительный соответственно (Рисунок 4.2).

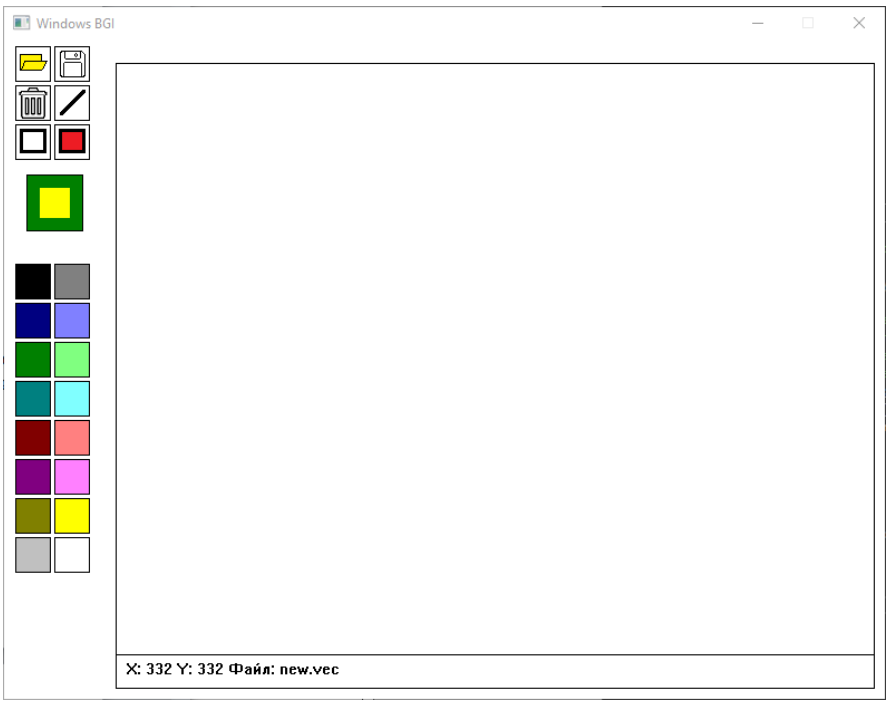

Рисунок 4.2 – В качестве основного цвета выбран зеленый, а в качестве дополнительного - желтый.

При нажатии левой кнопкой мыши на панель инструментов будет выбран один из них. Если выбрать инструмент “Отрезок”, “Прямоугольник” или “Закрашенный прямоугольник”, то появится возможность рисовать соответствующие фигуры. Для этого необходимо нажать левой кнопкой мыши в одной точке поля для рисования и, не отпуская левую кнопку мыши, переместить курсор в другую точку поля, а затем отпустить мышь (Рисунок 4.3).

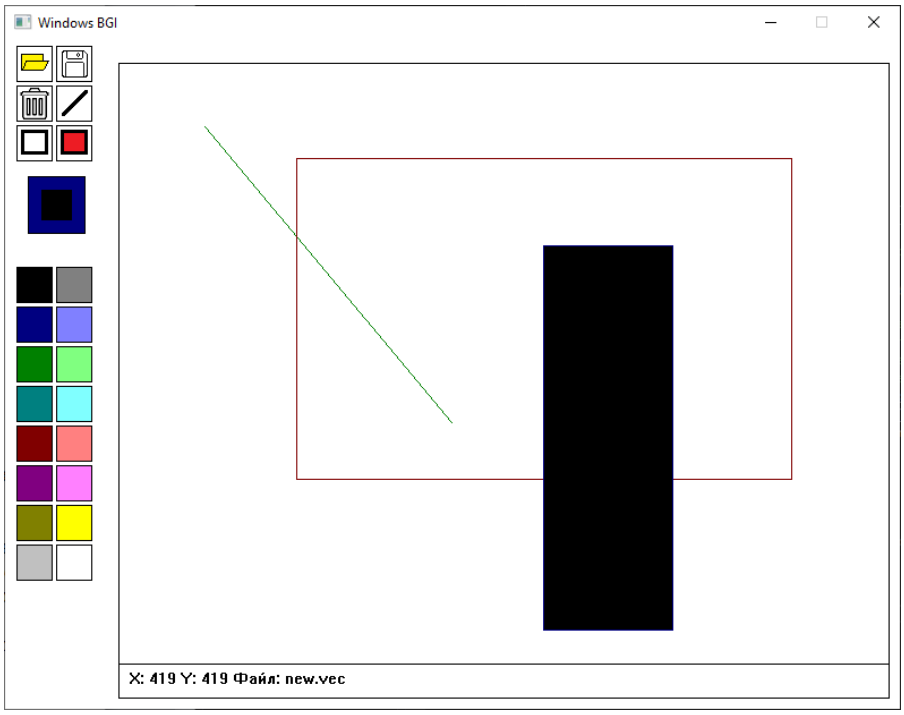

Рисунок 4.3 – Нарисованый отрезок, прямоугольник, и закрашенный прямоугольник.

Если нажать левой кнопкой мыши на отрезок, одну из сторон прямоугольника или внутри закрашенного прямоугольника с выбранным инструментом “Удаление”, то выбранная фигура будет удалена (Рисунок 4.4).

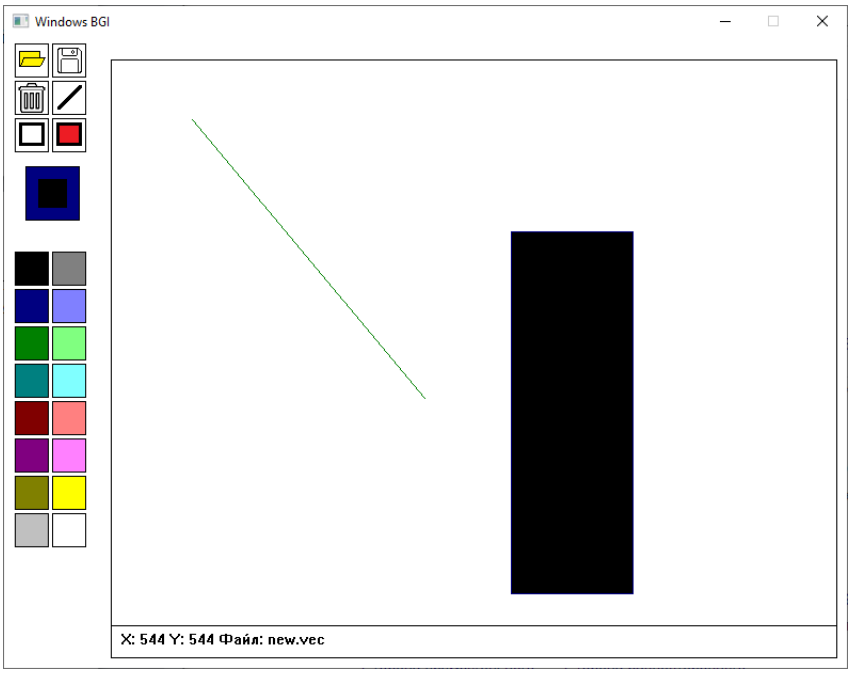

Рисунок 4.4 – Прямоугольник был удален.

При необходимости можно сохранить или загрузить ранее созданное изображение. Для этого необходимо выбрать инструмент “Сохранение” или “Загрузка” соответственно, ввести имя файла, и нажать “Enter” (Рисунок 4.5). При попытке загрузить несуществующий файл будет выведено сообщение “Файл не найден” (Рисунок 4.6).

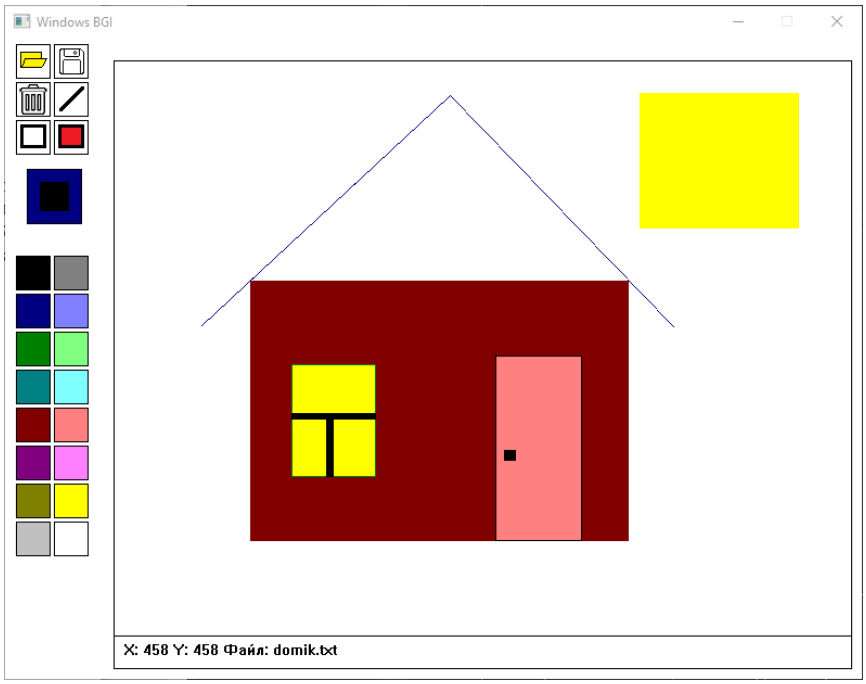

Рисунок 4.5 – Изображение успешно загружено.

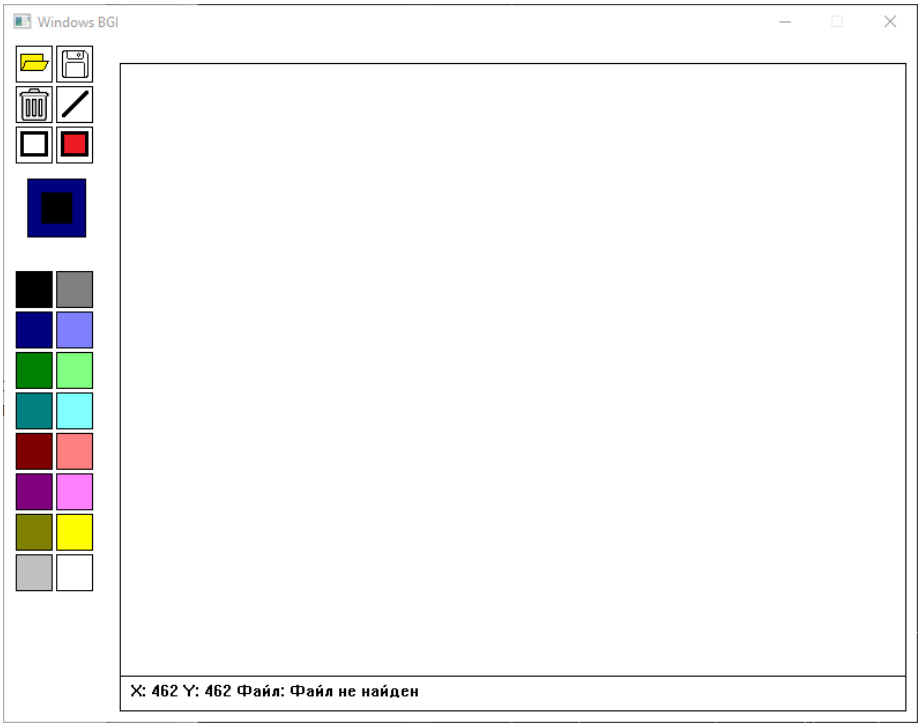

Рисунок 4.6 – Ошибка при открытии несуществующего файла.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были выявлены объекты предметной области и определена система классов для них, разработан интерфейс программы. После объектно-ориентированного проектирования классы были реализованы на языке C++. Разработанный код был проверен на контрольных тестах и в код были внесены необходимые исправления. Для программы была разработана документация, описывающая её установку и использование. Таким образом, цель работы была достигнута, задачи – решены.

Результаты работы можно использовать в процессе последующего обучения в форме навыков практического применения объектно-ориентированного подхода для разработки сложных программных систем, понимания порядка этапов разработки программного обеспечения и достигаемых на каждом этапе результатов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Гамма, Э. Приемы объектно ориентированного проектирования. Паттерны проектирования. [Электронный ресурс] / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. — Электрон. дан. — М. : ДМК Пресс, 2007. — 368 с. — Режим доступа: http://e.lanbook.com/book/1220

2 Липман, С. Язык программирования C++. Полное руководство. [Электронный ресурс] / С. Липман, Ж. Лажойе. — Электрон. дан. — М. : ДМК Пресс, 2006. — 1105 с. — Режим доступа: http://e.lanbook.com/book/1216

ПРИЛОЖЕНИЕ А

А.1 Файл main.cpp

```
#include "graphics.h"
#include "interface.hpp"
#include "button.hpp"

Button* buttons[22];

void init_controls() {
     for (int i = 0; i < 16; ++i) {
          buttons[i] = new ColorButton(10 + (i / 8) * 35, 200 +
               (i % 8) * 35, i);
     }
     buttons[16] = new CommandButton(10, 5, "res/load.bmp",
          load_command);
     buttons[17] = new CommandButton(45, 5, "res/save.bmp",
          save_command);
     buttons[18] = new ToolButton(10, 40, "res/delete.bmp",
          delete_tool);
     buttons[19] = new ToolButton(45, 40, "res/line.bmp",
          line_tool);
     buttons[20] = new ToolButton(10, 75, "res/rect.bmp",
          rect_tool);
     buttons[21] = new ToolButton(45, 75, "res/frect.bmp",
          colloredrect_tool);
     PaintArea::instance().set_tool(line_tool);
}
void draw_controls() {
     setfillstyle(SOLID_FILL, WHITE);
     bar(0, 0, getmaxx(), getmaxy());
     for (int i = 0; i < 22; ++i) {
          buttons[i]->draw();
     }
     PaintArea::instance().draw();
     SelectedColors::instance().draw();
     StatusLine::instance().draw();
}
int main() {
     initwindow(800, 600);
     init_controls();
     draw_controls();
     int x, y;
     while (1) {
          if (mousebuttons()) {
               x = mousex();
               y = mousey();
```

```
                if (PaintArea::instance().in(x, y)) {
                     PaintArea::instance().press();
                }
                else {
                     for (int i = 0; i < 22; ++i) {
                          if (buttons[i]->in(x, y)) {
                               buttons[i]->press();
                          }
                     }
                }
           }
           else {
                x = mousex();
                y = mousey();
                if (PaintArea::instance().in(x, y)) {
                     StatusLine::instance().set_xy(x -
                          PaintArea::instance().get_left(), y -
                          PaintArea::instance().get_top());
                }
           }
     }
     closegraph();
}
```

А.2 Файл button.hpp

```
#ifndef BUTTON_H
#define BUTTON_H

#include <cmath>
#include "graphics.h"
#include "interface.hpp"

// Базовый класс для кнопок управления
class Button : public ControlObject {
public:
     // конструктор получает координаты левого верхнего угла x,y
     Button(int x, int y) :ControlObject(x, y, x + 31, y + 31) {}
};
// Класс для кнопок выбора цвета
class ColorButton : public Button {
     int c; // цвет кнопки
public:
     // конструктор получает координаты x,y и номер цвета c
     ColorButton(int x, int y, int c) :Button(x, y), c(c) {}
     void draw(); // нарисовать объект
     void press(); // реакция на нажатие
};
```

```
// Класс для кнопок с картинкой
class IconButton : public Button {
     IMAGE* image; // загруженная картинка для кнопки
public:
     // конструктор получает координаты x,y, имя файла с картинкой
     IconButton(int x, int y, string icon) :Button(x, y),
          image(loadBMP(icon.c_str())) {}
     ~IconButton() { freeimage(image); } // деструктор
     void draw(); // нарисовать объект
};
// Класс для кнопок выбора инструмента
class ToolButton : public IconButton {
     fun_ptr tool; // инструмент
public:
     // конструктор получает координаты x,y, имя файла с картинкой
     // icon и указатель на функцию-инструмент t
     ToolButton(int x, int y, string icon, fun_ptr t)
          :IconButton(x, y, icon), tool(t) {}
     void press(); // реакция на нажатие
};
// Класс для кнопок-команд
class CommandButton : public IconButton {
     fun_ptr action; // действие
public:
     // конструктор получает координаты x,y, имя файла с картинкой
     // icon и указатель на действие a
     CommandButton(int x, int y, string icon, fun_ptr a)
          :IconButton(x, y, icon), action(a) {}
     void press(); // реакция на нажатие
};

void delete_tool(); // инструмент для удаления
void line_tool(); // инструмент линия
void rect_tool(); // инструмент прямоугольник
void colloredrect_tool(); // инструмент закрашенный прямоугольник
void load_command(); // загрузка рисунка
void save_command(); // сохранение рисунка

#endif
```

А.3 Файл button.cpp

```
#include "button.hpp"

double count_distance(int x, int y, int x1, int y1, int x2, int y2)
{
     double t = ((x - x1) * (x2 - x1) + (y - y1) * (y2 - y1)) /
          (pow(x2 - x1, 2) + pow(y2 - y1, 2));
```

```
    if (t < 0){
        t = 0;
    }
    if (t > 1){
        t = 1;
    }
    double dist = sqrt(pow(x1 - x + (x2 - x1) * t, 2) + pow(y1 -
        y + (y2 - y1) * t, 2));
    return dist;
}
void ColorButton::draw() {
    setfillstyle(SOLID_FILL, c);
    bar(x1, y1, x2, y2);
    setcolor(BLACK);
    rectangle(x1, y1, x2, y2);
}
void ColorButton::press() {
    if (mousebuttons() == 1){
        SelectedColors::instance().set_fc(c);
    }
    else{
        SelectedColors::instance().set_bc(c);
    }
}
void IconButton::draw() {
    setfillstyle(SOLID_FILL, WHITE);
    bar(x1, y1, x2, y2);
    putimage(x1 + 1, y1 + 1, image, COPY_PUT);
    setcolor(BLACK);
    rectangle(x1, y1, x2, y2);
}
void CommandButton::press() {
    action();
}
void ToolButton::press() {
    PaintArea::instance().set_tool(tool);
}
void delete_tool() {
    const int max_dist = 4;
    int x = mousex(),
        y = mousey();
    Figure* curr_fig;
    StatusLine::instance().set_xy(x -
        PaintArea::instance().get_left(), y -
        PaintArea::instance().get_top());
```

```
     for (int i = PaintArea::instance().figure_amount() - 1;
          i >= 0; i--) {
          curr_fig = PaintArea::instance().get_figure(i);
          if (typeid(*curr_fig) == typeid(Line)) {
               int x1, y1, x2, y2, c;
               curr_fig->get_data(x1, y1, x2, y2, c);
               double dist = count_distance(x, y, x1, y1, x2, y2);
               if (dist <= max_dist) {
                    PaintArea::instance().delete_figure(i);
                    PaintArea::instance().refresh();
                    delay(200);
                    break;
               }
          }
          if (typeid(*curr_fig) == typeid(Rect)) {
               int x1, y1, x2, y2, c;
               curr_fig->get_data(x1, y1, x2, y2, c);
              double dist1 = count_distance(x, y, x1, y1, x2, y1);
              double dist2 = count_distance(x, y, x2, y1, x2, y2);
              double dist3 = count_distance(x, y, x2, y2, x1, y2);
              double dist4 = count_distance(x, y, x1, y2, x1, y1);
               if (dist1 <= max_dist || dist2 <= max_dist ||
                    dist3 <= max_dist || dist4 <= max_dist) {
                    PaintArea::instance().delete_figure(i);
                    PaintArea::instance().refresh();
                    delay(200);
                    break;
               }
          }
          if (typeid(*curr_fig) == typeid(ColoredRect)) {
               int x1, y1, x2, y2, c, inner_c;
               ColoredRect* temp =
                    dynamic_cast<ColoredRect*>(curr_fig);
               temp->get_data(x1, y1, x2, y2, c, inner_c);
               if (x >= x1 && x <= x2 && y >= y1 && y <= y2) {
                    PaintArea::instance().delete_figure(i);
                    PaintArea::instance().refresh();
                    delay(200);
                    break;
               }
          }
     }
}
void rect_shape(int x1, int y1, int x2, int y2) {
     setwritemode(XOR_PUT);
     setcolor(WHITE);
     rectangle(x1, y1, x2, y2);
     setwritemode(COPY_PUT);
}
```

```
void line_shape(int x1, int y1, int x2, int y2) {
     setwritemode(XOR_PUT);
     setcolor(WHITE);
     line(x1, y1, x2, y2);
     setwritemode(COPY_PUT);
}
bool elastic_tool(int& x1, int& y1, int& x2, int& y2,
     void (*shape)(int x1, int y1, int x2, int y2)) {
     x1 = mousex();
     y1 = mousey();
     x2 = x1;
     y2 = y1;
     shape(x1, y1, x2, y2);
     while (1) {
          int b = mousebuttons();
          if (!b) break;
          int xn = mousex();
          int yn = mousey();
          if (!PaintArea::instance().in(xn, yn)) {
               shape(x1, y1, x2, y2);
               return 0;
          }
          if (xn != x2 || yn != y2) {
               shape(x1, y1, x2, y2);
               x2 = xn;
               y2 = yn;
               shape(x1, y1, x2, y2);
               StatusLine::instance().set_xy(x2 -
                    PaintArea::instance().get_left(), y2 -
                    PaintArea::instance().get_top());
          }
     }
     shape(x1, y1, x2, y2);
     return 1;
}
void line_tool() {
     int x1, y1, x2, y2;
     if (elastic_tool(x1, y1, x2, y2, line_shape)) {
          PaintArea::instance().create_figure("Line", x1, y1, x2,
               y2, SelectedColors::instance().get_fc());
     }
}
void rect_tool() {
     int x1, y1, x2, y2;
     if (elastic_tool(x1, y1, x2, y2, rect_shape)) {
          PaintArea::instance().create_figure("Rect", x1, y1, x2,
               y2, SelectedColors::instance().get_fc());
     }
}
```

```
void colloredrect_tool() {
     int x1, y1, x2, y2;
     if (elastic_tool(x1, y1, x2, y2, rect_shape)) {
          PaintArea::instance().create_figure("ColoredRect", x1,
               y1, x2, y2, SelectedColors::instance().get_fc(),
               SelectedColors::instance().get_bc());
     }
}
void load_command() {
     StatusLine::instance().edit_name();
     PaintArea::instance().load(StatusLine::instance().get_name());
}
void save_command() {
     StatusLine::instance().edit_name();
     PaintArea::instance().save(StatusLine::instance().get_name());
}
```

A.4 Файл interface.hpp

```
#ifndef INTERFACE_H
#define INTERFACE_H

#include "graphics.h"
#include <string>
#include <vector>
#include <fstream>

using namespace std;


class DrawObject { // класс для объектов на экране
protected:
     int x1, y1, x2, y2; // координаты углов прямоугольника, в ко-
     тором находится объект
public:
     DrawObject(int x1, int y1, int x2, int y2) :x1(x1), y1(y1),
          x2(x2), y2(y2) {} // конструктор получает координаты ле-
вого верхнего угла x1,y1 и координаты правого нижнего угла x2,y2
     virtual ~DrawObject() {} // деструктор
     bool in(int x, int y); // координаты x, y внутри объекта
     int get_top() { return y1 + 1; } // координаты
     int get_left() { return x1 + 1; } // верхнего угла
     int get_height() { return y2 - y1 - 1; } // высота
     int get_width() { return x2 - x1 - 1; } // ширина
     virtual void draw() = 0; // нарисовать объект
};
```

```
class SelectedColors : public DrawObject { // класс для
отображения текущих цветов рисования
     int fc, // цвет рисования
         bc; // цвет закраски
     // конструктор получает координаты углов x1,y1,x2,y2
     SelectedColors(int x1, int y1, int x2, int y2) :
          DrawObject(x1, y1, x2, y2), fc(BLACK), bc(WHITE) {}
public:
     static SelectedColors& instance(); // объект для текущих
цветов
     void draw(); // нарисовать объект
     void set_fc(int c); // установить цвет рисования
     int get_fc() { return fc; } // получить  цвет рисования
     void set_bc(int c); // установить цвет закраски
     int get_bc() { return bc; } // получить цвет закраски
};

class StatusLine : public DrawObject { // класс для отображения
текущих координат мыши и имени файла
     int x, y; // координаты мыши
     bool edit;
     string name; // имя файла
     // конструктор получает координаты углов x1,y1,x2,y2
     StatusLine(int x1, int y1, int x2, int y2) :
          DrawObject(x1, y1, x2, y2), x(0), y(0), name("new.vec")
          {}
public:
     static StatusLine& instance(); // строка состояния
     void draw(); // нарисовать объект
     void set_xy(int x, int y); // изменить текущие координаты
     void set_name(const string& name); // установить имя файла
     string get_name() { return name; } // получить имя файла
     void edit_name(); // ввести имя файла
};

class ControlObject : public DrawObject { // класс для объектов,
реагирующих на мышь
public:
     ControlObject(int x1, int y1, int x2, int y2)
          :DrawObject(x1, y1, x2, y2) {} // конструктор получает
     координаты углов x1,y1,x2,y2
     virtual void press() = 0; // реакция на нажатие
};

typedef void (*fun_ptr)(); // вспомогательный тип - указатель на
функцию
```

```
class Figure {
     bool visible = 0; // видимость
protected:
     int x_1, y_1, //точки для построения
         x_2, y_2;
     int c; //цвет
public:
     Figure(int x_1, int y_1, int x_2, int y_2, int c) :
          x_1(x_1), y_1(y_1), x_2(x_2), y_2(y_2), c(c) {}
     virtual ~Figure() {}
     int get_color() const { return c; } //получить цвет
     void hide(); //спрятать
     void show(); //показать фигуру
     bool isvisible() const { return visible; } //видима?
     virtual void get_data(int& x1, int& y1, int& x2, int& y2,
          int& c); // получение точек и цветов фигуры
     virtual void draw() = 0; // метод для отрисовки
};

class Rect : public Figure {
public:
     Rect(int x_1, int y_1, int x_2, int y_2, int c)
          :Figure(x_1, y_1, x_2, y_2, c) {}
     ~Rect() { hide(); }
     void draw(); // метод для отрисовки
};

class ColoredRect : public Rect {
protected:
     int inner_c; // цвет заливки
public:
     ColoredRect(int x_1, int y_1, int x_2, int y_2, int c,
          int inner_c)
          :Rect(x_1, y_1, x_2, y_2, c), inner_c(inner_c) {}
     ~ColoredRect() { hide(); }
     void get_data(int& x_1, int& y_1, int& x_2, int& y_2, int& c,
          int& inner_c); // получение точек и цветов закрашенного
          прямоугольника
     void draw(); // метод для отрисовки
};

class Line : public Figure { // класс для отрезка
public:
     Line(int x_1, int y_1, int x_2, int y_2, int c)
          :Figure(x_1, y_1, x_2, y_2, c) {}
     ~Line() { hide(); }
     void draw(); // метод для отрисовки
};
```

```
class PaintArea : public ControlObject { // класс для поля
рисования
     vector <Figure*> storage; // хранилище фигур
     fun_ptr tool; // текущий инструмент
     // конструктор получает координаты углов x1,y1,x2,y2
     PaintArea(int x1, int y1, int x2, int y2) :
          ControlObject(x1, y1, x2, y2), tool(nullptr) {}
public:
     static PaintArea& instance(); // поле для рисования
     void draw(); // нарисовать само поле
     void create_figure(string type, int x_1, int y_1, int x_2,
          int y_2, int c, int inner_c = -1); // создание фигуры
     void press(); // реакция на нажатие
     void set_tool(fun_ptr t) { tool = t; }; // установить
инструмент рисования
     fun_ptr get_tool() { return tool; }; // получить текущий
инструмент рисования
     void save(const string& name); // сохранить изображение
     void load(const string& name); // загрузить изображение
     Figure* get_figure(int i) { return storage[i]; }; // доступ к
i-ой фигуре
     int figure_amount() { return storage.size(); }; // возвращает
кол-во фигур
     void delete_figure(int i) { storage.erase(storage.begin() +
          i); }; // удалить фигуру из вектора
     void refresh(); // обновить содержимое экрана
};

#endif
```

A.5 Файл interface.cpp

```
#include "interface.hpp"
bool DrawObject::in(int x, int y) {
     return x > x1&& y > y1&& x < x2 && y < y2;
}
SelectedColors& SelectedColors::instance() {
     static SelectedColors sc(20, 120, 70, 170);
     return sc;
}
void SelectedColors::draw() {
     setfillstyle(SOLID_FILL, fc);
     bar(x1, y1, x2, y2);
     setfillstyle(SOLID_FILL, bc);
     int dx = (x2 - x1) / 4;
     int dy = (y2 - y1) / 4;
     bar(x1 + dx, y1 + dx, x2 - dx, y2 - dy);
     setcolor(BLACK);
     rectangle(x1, y1, x2, y2);
}
```

```
void SelectedColors::set_fc(int c) {
     fc = c;
     draw();
}
void SelectedColors::set_bc(int c) {
     bc = c;
     draw();
}
StatusLine& StatusLine::instance() {
     static StatusLine sl(100, 550, 780, 580);
     return sl;
}
void StatusLine::draw() {
     setfillstyle(SOLID_FILL, WHITE);
     bar(x1, y1, x2, y2);
     setcolor(BLACK);
     rectangle(x1, y1, x2, y2);
     string s = "X: " + to_string(x) + " Y: " + to_string(x) +
          (edit ? " Введите имя файла: " : " Файл: ") + name;
     setbkcolor(WHITE);
     outtextxy(x1 + 10, y1 + 5, s.c_str());
}
void StatusLine::set_xy(int x, int y) {
     if (x == this->x && y == this->y) return;
     this->x = x;
     this->y = y;
     draw();
}
void StatusLine::set_name(const string& n) {
     name = n;
     draw();
}
void StatusLine::edit_name() {
     int ch;
     edit = 1;
     while (1) {
          draw();
          ch = getch();
          if (ch == KEY_ENTER || ch == KEY_ESC) break;
          else if (ch == KEY_BACKSPACE && name.size() > 0)
               name.pop_back();
          else if (ch > ' ' && ch <= 'z') name += (char)ch;
     }
     edit = 0;
     draw();
}
PaintArea& PaintArea::instance() {
     static PaintArea pa(100, 20, 780, 550);
     return pa;
}
```

```
void PaintArea::draw() {
     setfillstyle(SOLID_FILL, WHITE);
     bar(x1, y1, x2, y2);
     setcolor(BLACK);
     rectangle(x1, y1, x2, y2);
}
void PaintArea::press() {
     tool();
}
void PaintArea::save(const string& name) {
     ofstream file(name);
     for (int i = 0; i < storage.size(); i++) {
          if (typeid(*storage[i]) == typeid(ColoredRect)) {
               int x1, y1, x2, y2, c, inner_c;
               ColoredRect* temp =
                    dynamic_cast<ColoredRect*>(storage[i]);
               temp->get_data(x1, y1, x2, y2, c, inner_c);
               file << "ColoredRect " << x1 << " " << y1 << " "
                    << x2 << " " << y2 << " " << c << " "
                    << inner_c << endl;
          }
          else {
               int x1, y1, x2, y2, c;
               storage[i]->get_data(x1, y1, x2, y2, c);
               if (typeid(*storage[i]) == typeid(Rect)) {
                    file << "Rect ";
               }
               else {
                    file << "Line ";
               }
               file << x1 << " " << y1 << " " << x2 << " "
                    << y2 << " " << c << endl;
          }
     }
     file.close();
}
void PaintArea::load(const string& name) {
     ifstream file(name);
     string str;
     int x_1, y_1, x_2, y_2, c, inner_c;
     if (!file.is_open()) {
          StatusLine::instance().set_name("Файл не найден");
     }
     storage.clear(); // очистка вектора от старых фигур
     draw(); // очистка экрана
     while (file >> str >> x_1 >> y_1 >> x_2 >> y_2 >> c) {
          if (str == "ColoredRect") {
               file >> inner_c;
          }
```

```
            create_figure(str, x_1, y_1, x_2, y_2, c, inner_c);
        }
        file.close();
}
void PaintArea::refresh() {
        draw();
        for (int i = 0; i < storage.size(); i++) {
            storage[i]->draw();
        }
}
void PaintArea::create_figure(string type, int x_1, int y_1,
        int x_2, int y_2, int c, int inner_c) {
        if (type == "Line") {
            storage.push_back(new Line(x_1, y_1, x_2, y_2, c));
        }
        else {
            if (type == "Rect") {
                storage.push_back(new Rect(min(x_1, x_2), min(y_1,
                    y_2), max(x_1, x_2), max(y_1, y_2), c));
            }
            else {
                storage.push_back(new ColoredRect(min(x_1, x_2),
                    min(y_1, y_2), max(x_1, x_2), max(y_1, y_2),
                    c, inner_c));
            }
        }
        storage.back()->show();
}
void Figure::hide() {
        if (!visible) return;
        setfillstyle(SOLID_FILL, WHITE);
        bar(x_1, y_1, x_2, y_2);
        visible = 0;
}
void Figure::show() {
        if (visible) return;
        visible = 1;
        draw();
}
void Figure::get_data(int& x_1, int& y_1, int& x_2, int& y_2, int&
c) {
        x_1 = this->x_1;
        y_1 = this->y_1;
        x_2 = this->x_2;
        y_2 = this->y_2;
        c = this->c;
}
```

```
void Line::draw() {
     setcolor(get_color());
     line(x_1, y_1, x_2, y_2);
}
void Rect::draw() {
     setcolor(get_color());
     rectangle(x_1, y_1, x_2, y_2);
}
void ColoredRect::draw() {
     setfillstyle(SOLID_FILL, inner_c);
     bar(x_1, y_1, x_2, y_2);
     setcolor(get_color());
     rectangle(x_1, y_1, x_2, y_2);
}
void ColoredRect::get_data(int& x_1, int& y_1, int& x_2, int& y_2,
int& c, int& inner_c) {
     x_1 = this->x_1;
     y_1 = this->y_1;
     x_2 = this->x_2;
     y_2 = this->y_2;
     c = this->c;
     inner_c = this->inner_c;
}
```