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

«Библиотека классов для оконного интерфейса в графическом режиме»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

по дисциплине «Объектно-ориентированное программирование»

ЮУрГУ–01.03.02.2020.153.ПЗ КР

Руководитель,

_____ *Демидов А.К.*

«___» _____ 2020г.

Автор работы:

Студент группы: ЕТ – 212

_____ *Шерстобитов Т.С.*

«___» _____ 2020г.

Работа защищена с оценкой

_____

«___» _____2020 г.

Челябинск – 2020

АННОТАЦИЯ

Шерстобитов Т.С. Библиотека классов для оконного интерфейса в графическом режиме. – Челябинск: ЮУрГУ, ЕТ-212, 2020. – 48с, библиографический список – 1 наим., 1 прил.

В курсовой работе описывается разработка библиотеки классов для оконного интерфейса в графическом режиме с помощью объектно-ориентированного подхода. Работа содержит результаты объектно-ориентированного анализа и проектирования, инструкции по установке и использованию библиотеки.

В результате работы была разработана библиотека классов для оконного интерфейса в графическом режиме, код которой приводится в приложении.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы – Объектно-ориентированный подход является наиболее прогрессивной технологией разработки программных систем, позволяет разрабатывать более сложные системы.

Цель работы – Написать библиотеку для оконного интерфейса в графическом режиме.

Задачи работы:

–изучить приёмы объектно-ориентированного анализа;

–научиться разрабатывать программы в объектно-ориентированом 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 – Графический интерфейс Borland (BGI)

Предмет работы – применение объектно-ориентированного подхода для разработки библиотеки.

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 библиотеку классов для оконного интерфейса в графическом режиме. Должны определяться следующие классы:

1.окно
- -координаты
- -размеры
- -видимость
- -цвет фона
- -текст заголовка
- -процедура, вызываемая для нераспознанных клавиш окна
- -верхнее окно // свойство класса
- -добавить(элемент) // метод

2.метка (надпись)
- -координаты
- -видимость
- -цвет
- -текст метки

3.кнопка
- -координаты
- -размеры
- -видимость
- -цвет кнопки
- -текст кнопки
- -цвет текста
- -процедура, вызываемая при нажатии кнопки

4.ввод строки
- -координаты
- -ширина
- -видимость
- -цвет поля
- -цвет текста
- -текст

В качестве примера был рассмотрен оконный интерфейс Windows XP

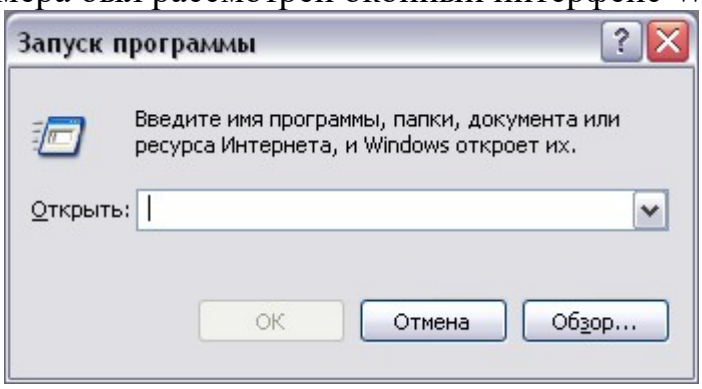

Рисунок 1.1 пример окна

Анализ предметной области выявил, что объекты интерфейса могут быть двух типов, контейнеры и компоненты. Каждый объект интерфейса имеет такие свойства как видимость, ширина, высота, позиция по x и y относительно родительского элемента, цвет фона, цвет отрисовки.

2 ОПИСАНИЕ ПРОГРАММЫ

2.1 Для разработки библиотеки были использованы:

–компилятор MinGW GNU C/C++ 7.2.

2.2 Библиотека состоит из 9 модулей:

Модуль Event (интерфейсная часть в файле .hpp, реализация в файле event.cpp) содержит следующие классы

```
class Event {//Интерфейс метка для стуктур данных событий
public:
   virtual ~Event() {};
};

//Событие мыши
class MouseEvent : public Event {
public:
   enum Type {
      MOVE, DRAG, LEFT_KEY_DOWN, LEFT_KEY_UP
   };
   MouseEvent(int, int, Type, int = 0, int = 0);//Конструктор
   int getX() const;//Возращает позицию x нажатия мыши
   int getY() const;// Возращает позицию y нажатия мыши
   int getPreX() const;// Возращает предыдущую позицию x нажатия
мыши
   int getPreY() const;// Возращает предыдущую позицию y нажатия
мыши
   Type getType() const;// Возращает тип события мыши
private:
   int x, y, preX, preY;//Текущие x,y, предыдущие x,y
   Type type;//тип события мыши
};

//Событие клавиатуры
class KeyboardEvent : public Event {
public:
   KeyboardEvent(int, int = 0);//Конструктор
   int getKeyCode() const;//Возращает код тукущей нажатой клавиши
   int getPreKeyCode() const;//Возращает предыдущей нажатой
клавишу
private:
   int keyCode, preKeyCode;//Код клавиши, код предыдущей нажатой
клавиши
};
```

Модуль Listener (интерфейсная часть в файле .hpp, реализация в файле Listener.cpp) содержит следующие классы

```
class Listener {//Слушатель
public:
    virtual ~Listener(){}//Деструктор
    virtual void onEvent(Event *event){}//Действие при событии
};

class OnClickListener : public Listener {//Слушатель клика мыши
protected:
    virtual void onClick(MouseEvent *event){};//Действие при клике
мышкой
private:
    void onEvent(Event *event);//Проверка было ли событие кликом
мыши
};

class OnPressListener : public Listener {//Слуушатель зжатия
кнопки мыши
protected:
    virtual void onPress(MouseEvent *event){};//Действие при
зажатии кнопки мыши
private:
    void onEvent(Event *event);//Проверка было ли событие зажатием
кнопки мыши
};

class OnDragListener : public Listener {//Слуушатель
перетаскивания зажатой мыши
protected:
    virtual void onDrag(MouseEvent *event){};//Действие при
перетаскивании зажатой мыши
private:
    void onEvent(Event *event);//Проверка было ли событие
перетаскиванием зажатой мыши
};

class OnMoveListener : public Listener {//Слуушатель передвижения
мыши
protected:
    virtual void onMove(MouseEvent *event){};//Действие при
передвижении мыши
private:
    void onEvent(Event *event);//Проверка было ли событие
передвижением мыши
};
```

```
class OnKeyUpListener : public Listener {//Слуушатель событий
клавиатуры
protected:
    virtual void onKeyUp(KeyboardEvent *event){};// Действие при
событии клавиатуры
private:
    void onEvent(Event *event);//Проверка является ли событие
событием клавиатуры
};
```

Модуль Component (интерфейсная часть в файле .hpp, реализация в файле Component.cpp) содержит следующие классы

```
class Component { //Компонент
public:
   virtual ~Component(); //Деструктор
   void addListener(Listener *listener); //Добавляет слушателя
   void notify(Event *event); //Сообщить слушателям компонента о
событии
   void render(int rootX, int rootY, int rootWidth, int
rootHeight);//Визуализация и обновление абсолютной позиции
   virtual void draw(int rootWidth, int rootHeight) {} //Метод
отрисовки компонента
   //Возвращает…
   bool isVisible() const; // true если компонент видимый
   int getWidth() const; // ширину компонента
   int getHeight() const; // высоту компонента
   int getX() const; // позицию x компонента в родительском
контейнере
   int getY() const; // позицию y компонента в родительском
контейнере
   int getAbsolutX() const; // абсолютную позицию x
   int getAbsolutY() const; // абсолютную позицию y
   string getTag() const; // тег элемента
   //Устанавливает
   void setVisible(bool visible); // Видимость элемента
   void setWidth(int width); // Ширину элемента
   void setHeight(int height); // Высоту элемента
   void setX(int x); // Позицию x компонента в родительском
контейнере
   void setY(int y); // Позицию y компонента в родительском
контейнере
   void setPosition(int x, int y);// Позицию x и y компонента в
родительском контейнере
   void setBgColor(int color);// Цвет фона компонента
   void setColor(int color); // Основной цвет отрисовки компонента
   void setTag(string tag); // Тег элемента
protected:
   string tag = "untag"; // Тег элемента
   bool visible = true; //Видимость объекта
```

```
   int width = 0, height = 0, x = 0, y = 0,absolutX = 0, absolutY
= 0;// ширина, высота, позиция компонента относительно
родительского контейнера, абсолютная позиция компонента
   int bgColor = LIGHTGRAY; //Цвет фона, по умолчанию светло-серый
   int color = BLACK; //Основной цвет рисования, по умолчанию
чёрный
private:
   vector<Listener*> listeners; //Динамический массив слушателей
};
```

Модуль Container (интерфейсная часть в файле .hpp, реализация в файле Container.cpp) содержит следующие классы

```
class Container : public Component {//Контейнер
public:
   Container(); //Конструктор
   virtual ~Container(); //Деструктор
   void addComponent(Component *component); //Добавить компонент в
контейнер
   void notifyAll(Event *event); //Вызывает свой notify, notify
компонентов в контейнере и notifyAll контейнеров в контейнере
   void setVisible(bool visible);//Устанавливает видимость
контейнера и дочерних элементов
   void renderAll(int rootX, int rootY, int rootWidth, int
rootHeight); //Вызывает свой render, render компонентов в
контейнере и renderAll контейнеров в контейнере
   vector<Component *> *getComponents() const; //Возращает
динамический массив компонентов
protected:
   void draw(int rootWidth, int rootHeight) {} //Метод отрисовки
контейнера
private:
   vector<Component *> *components; //Динамический массив
компонентов контейнера
};
```

Модуль Screen (интерфейсная часть в файле .hpp, реализация в файле Screen.cpp) содержит следующие классы

```
class Screen : public Container {//Экран, основа отображения
элементов
public:
   Screen();//Конструктор
   ~Screen() {}; //Деструктор
   void start();//Инициализация графического интерфейса
private:
   void draw(int rootWidth, int rootHeight);//Метод отрисовки
   bool run;//Флаг, при false приложение завершает свою работу
   void checkMouse();//Проверка событий мыши, уведомляет о них
слушателей
   void checkKeyboard();//Проверка событий клавиатуры, уведомляет
о них слушателей
```

```
   class ScreenOnKeyUpListener : public OnKeyUpListener {
   public:
      ScreenOnKeyUpListener(Screen *screen);//Конструктор
   private:
      Screen *screen;//Родительский экран
      void onKeyUp(KeyboardEvent *event);//Реализация завершения
работы Screen  при нажатии клавиши ESC
   };
   class ScreenOnSelectListner : public OnClickListener {
   public:
      ScreenOnSelectListner(Screen *screen);//Конструктор
   private:
      Screen *screen;//Родительский экран
      void replaceSelectables(int pos);//Перемещение окна в верх
массива
      void onClick(MouseEvent *event);//Выбор текущего активного
окна
   };
   class AboutButton : public Button {
   public:
      AboutButton(Screen *screen);//Конструктор
      virtual ~AboutButton() {}; //Деструктор
   private:
      class AboutButtonOnClickListener : public OnClickListener {
      public:
         AboutButtonOnClickListener(AboutButton *button,Screen
*screen);//Конструктор
      private:
         Screen *screen; //Кореной элемент
         AboutButton *button;//Родительская кнопка
         void onClick(MouseEvent *event); //Открытие окна "о
библиотеке"
      };
   };
};
```

Модуль Window (интерфейсная часть в файле .hpp, реализация в файле Window.cpp) содержит следующие классы

```
/**
*Окно, является контейнером для других компонентов.
*Может иметь заголовок.
*/
class Window : public Container {
public:
   Window(string title = " ", int width = 250, int height =
200);//Конструктор
   virtual ~Window(){};//Деструктор
   bool isSelected() const;//Возвращает true, если окно является
активным
   void setSelected(bool selected);//Устанавливает статус
активности окна
```

```
   string getTitle() const;//возвращает текст заголовка окна
protected:
   void draw(int rootWidth, int rootHeight);//отрисовывает окно,
унаследовано Container
private:
   bool selected = false;//статус активности окна
   bool dragedNow = false;//флаг перемещения окна
   string title;//текст заголовка окна
   class WindowOnDragListener : public OnDragListener { //вложеный
класс, реализующий слушатель перетаскивания мышью
   public:
      WindowOnDragListener(Window *window);//Конструктор
   private:
      Window *window;//окно слушателя
      void onDrag(MouseEvent *event);//перемещение окна
   };
   class WindowOnClickListener : public OnClickListener {
//вложеный класс, реализующий слушатель шелчка левой кнопки мыши
   public:
      WindowOnClickListener(Window *window);//Конструктор
   private:
      Window *window;//окно слушателя
      void onClick(MouseEvent *event);//снятие флага перемещения
   };
};
```

Модуль Button (интерфейсная часть в файле .hpp, реализация в файле Button.cpp) содержит следующие классы

```
class Button : public Container {
public:
   Button(int x = 0, int y = 0, int width = 100, int height =
25);// Конструктор
   virtual ~Button() {}; // Деструктор
   void setCheckColor(int color); // Установка цвета
устанавливаемого при наведении мыши на кнопку
   void setPressedColor(int color); // Установка цвета
устанавливаемого при зажатии левой кнопки мыши на кнопке
   protected:
   void draw(int rootWidth, int rootHeight); // Отрисовка
private:
   int checkColor = RGB(220,220,220); //Цвет устанавливаемый при
наведении мыши на кнопку, по умолчанию светлосерый
   int pressedColor = DARKGRAY; //Цвет устанавливаемый прри
зажатии кнопки
   enum Stage {
      UP, CHECK, DOWN
   };
```

```
   Stage stage = UP;//Стадия кнопки, по умолчанию UP
   class onMoveInButtonListener : public OnMoveListener {
   public:
      onMoveInButtonListener(Button *button); //Конструктор
   private:
      Button *button;//Родительская кнопка
      void onMove(MouseEvent *event); //Установка стадии на CHECK
при движении на кнопке
   };
   class onPressedButtonListener : public OnPressListener {
   public:
      onPressedButtonListener(Button *button);//Конструктор
   private:
      Button *button;//Родительская кнопка
      void onPress(MouseEvent *event);//Изменение стадии кнопки на
DOWN когда кнопка зажата
   };
   class ButtonOnClickListener : public OnClickListener {
   public:
      ButtonOnClickListener(Button *button);//Конструктор
   private:
      Button *button;//Родительская кнопка
      void onClick(MouseEvent *event); //Установка стадии на CHECK
при клике на кнопке
   };
};

class TextButton : public Button{
public:
   TextButton(string text = " " ,int x = 0, int y = 0, int width
=100, int height = 25);//Конструктор
   virtual ~TextButton(){};//Деструктор
   string getText() const;//Возвращает текст кнопки
   void setText(string text);//Установка текста кнопки
private:
   Label *buttonLabel;//Метка текстовой кнопки
};
```

Модуль Image (интерфейсная часть в файле .hpp, реализация в файле Image.cpp) содержит следующие классы

```
/**
*Метка (текст)
*/
class Image : public Component {
public:
   Image(string path, int x = 0, int y = 0, bool useMask =
false);//Конструктор
   virtual ~Image();//Деструктор
   void draw(int rootWidth, int rootHeight);//Отрисовка
изображения
```

```
private:
   bool useMask;//флаг использования маски (альтернатива альфа
канала)
   IMAGE *img, *mask;//изображение и маска
   IMAGE *createmask(IMAGE *p);//создание маски
};
```

Модуль TextInputField (интерфейсная часть в файле .hpp, реализация в файле TextInputField.cpp) содержит следующие классы

```
/*
*Поле ввода текста
*/
class TextInputField : public Component {
public:
   TextInputField(string text = " ", int x = 0, int y =
0);//Конструктор
   virtual ~TextInputField() {}; //Деструктор
   void setText(string text);//Устанавливает текст поля
   string getText() const;//Возвращает текст поля
protected:
   void draw(int rootWidth, int rootHeight);//Отрисовка
private:
   string text;//Текст поля
   class InputOnKeyListener : public OnKeyUpListener {
   public:
      InputOnKeyListener(TextInputField *field);//Конструктор
   private:
      TextInputField *field;//Родительское поле ввода
      void onKeyUp(KeyboardEvent *event);//Обработка события ввода
   };
};
```

2.3 Диаграмма UML

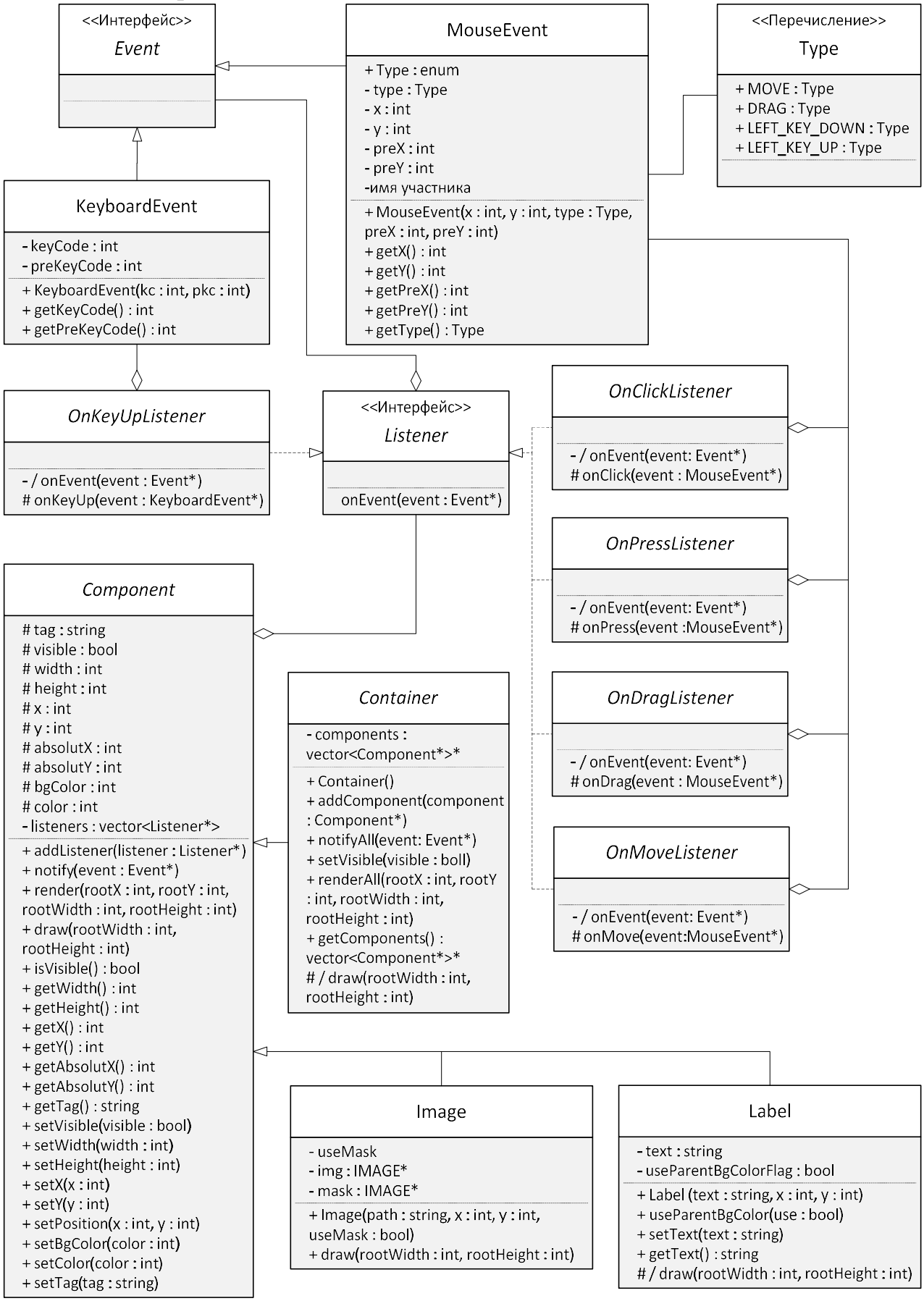

Рисунок 2.1 Иерархия классов

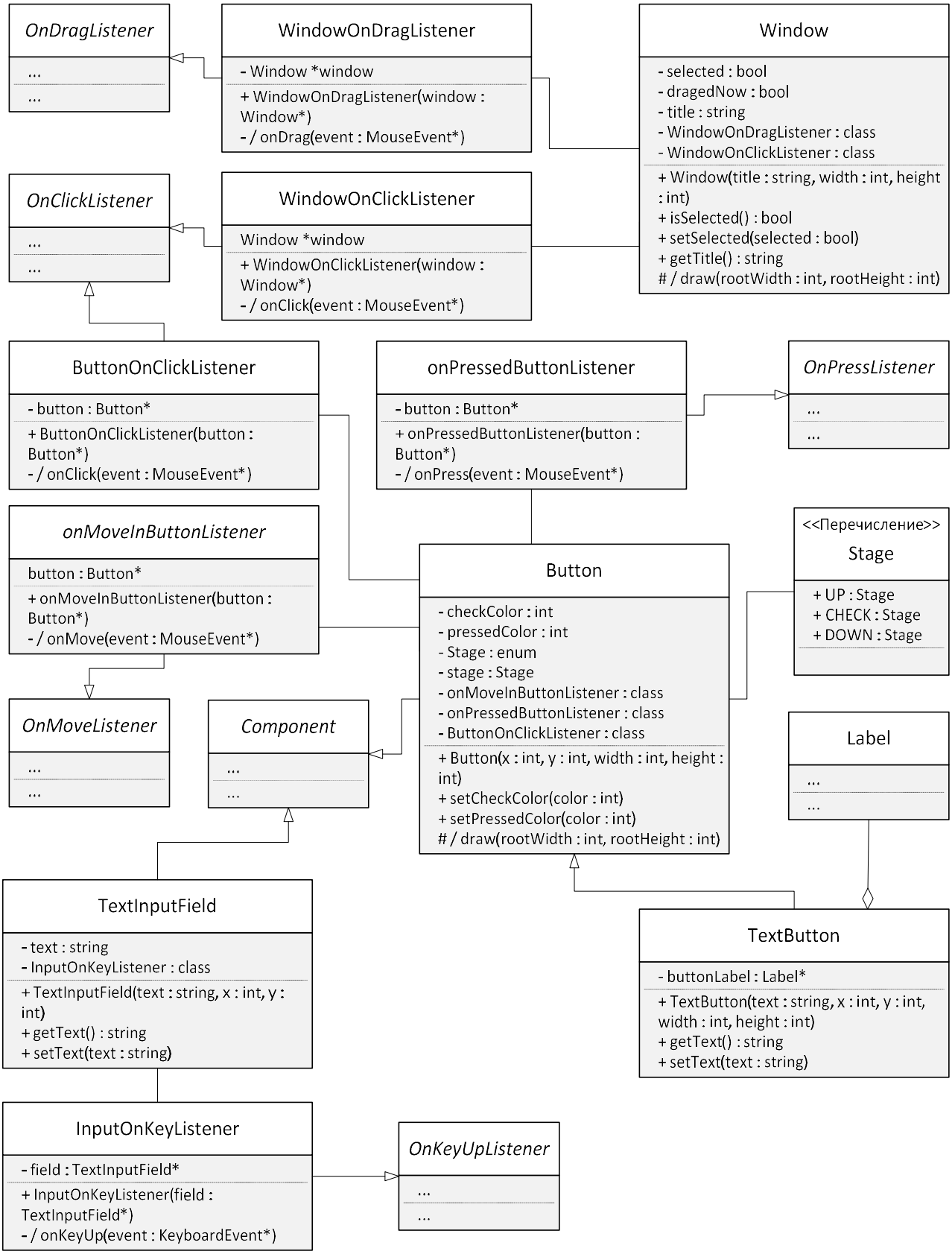

Рисунок 2.2 Иерархия классов

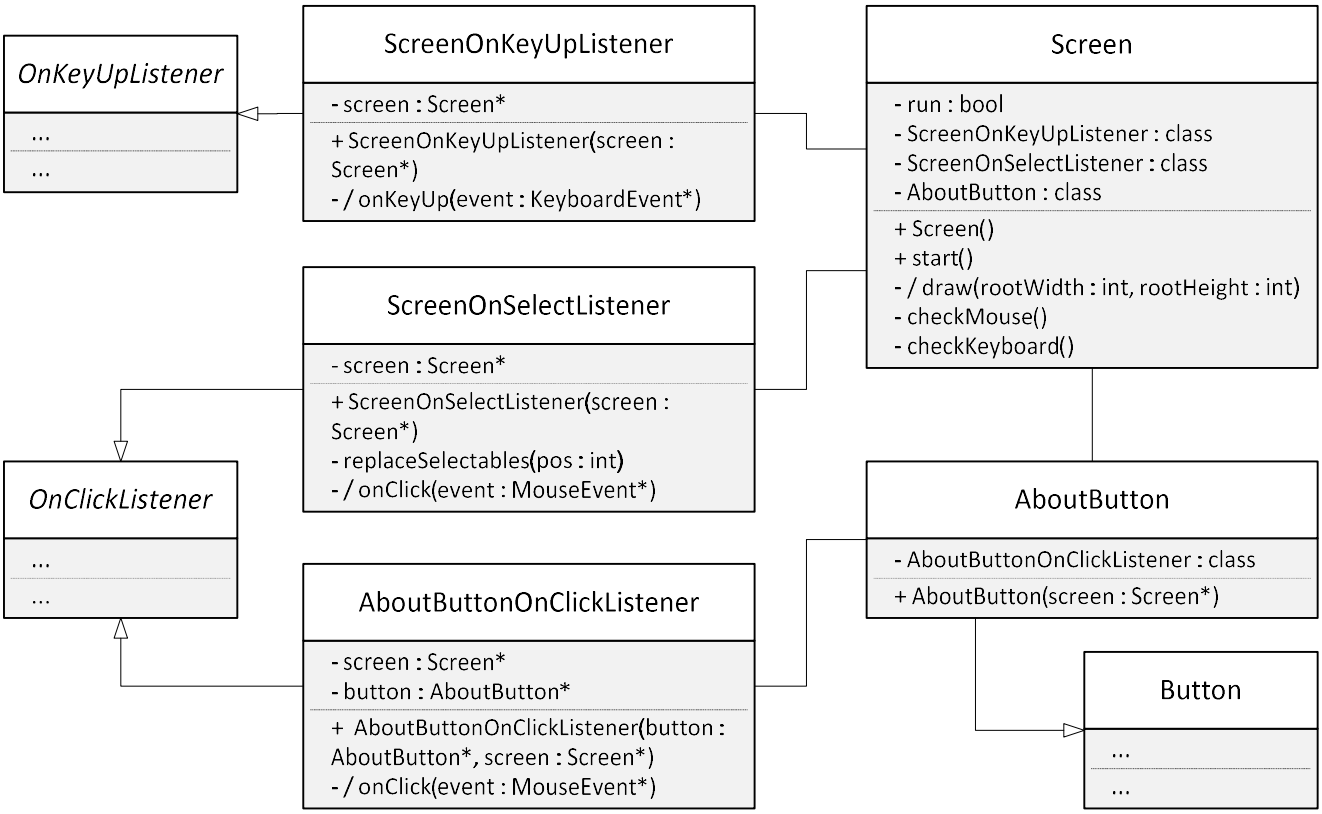

Рисунок 2.3 Иерархия классов

2.4 Пояснения по алгоритму и особенностям реализации

Для объектов наследников класса Component использован шаблон проектирования «Наблюдатель». В случае порождения, какого либо отслеживаемого события, главный контейнер Screen уведомляет все хранящиеся в нём компоненты. Контейнеры, которые были уведомлены, также уведомляют свои компоненты и т.д. Вследствие уведомления компонентов, слушатели выполняют то или иное действие или уведомление игнорируется.

Данная библиотека может быть легко расширена и использована для написания различных программ с графическим интерфейсом. Для добавления своего функционала пользователь может, как использовать уже готовые компоненты, создавая только пользовательских слушателей, так и создавать свои компоненты и контейнеры.

В случае необходимости, кнопка «О библиотеке» может быть легко удалена. Для этого нужно удалить её код из класса Screen.

2.5 Используемые внешние файлы

res/about.bmp – Изображение используемое в качестве кнопки «О библиотеке». Изображение имеет разрешение 48x48 и 8 битный цвет

Папка res находится в текущей папке библиотеки.

3 ИНСТРУКЦИЯ ПО УСТАНОВКЕ И ТРЕБОВАНИЯ К СИСТЕМЕ

Для сборки библиотеки необходим компилятор, поддерживающий C++14 и подключённая библиотека «graphics.h».

Установка и настройки не требуются. Для работы необходимо добавить файлы библиотеки в проект подключить заголовочный файл Screen.hpp

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для правильной работы библиотеки рекомендуется использовать в качестве главного контейнера класс Screen.

Пользователь может добавлять в контейнер любого наследника абстрактного класса Component. Классы наследники Container могут содержать в себе другие компоненты, в том числе контейнеры. Контейнеры, в том числе Screen, также являются наследниками абстрактного класса Component.

Component – абстрактный класс родитель для визуализируемых компонентов.

–С помощью метода addListener(Listener *listener) можно добавить компоненту пользовательского слушателя;

–Метод notify(Event *event) сообщает слушателям компонента о событии;

–Метод render(int rootX, int rootY, int rootWidth, int rootHeight) визуализирует компонент и обновляет его данные;

–Виртуальный метод draw(int rootWidth, int rootHeight) определяет как компонент будет визуализирован;

–Методы возвращающие некоторые данные компонента.
isVisible() – видимость компонента,
getWidth() – ширину, getHeight() – высоту,
getX() – позицию по оси x относительно контейнера компонента,
getY() – позицию по оси y относительно контейнера компонента,
getAbsolutX() – абсолютную позицию x, getAbsolutY() – абсолютную позицию y,
getTag() – тег компонента;

–Методы устанавливающие некоторые данные компонента.
setVisible(bool visible) – видимость компонента,
setWidth(int width) - ширину, setHeight(int height) - высоту,
setX(int x) – позицию по оси x относительно контейнера компонента,
setY(int y) - позицию по оси y относительно контейнера компонента,
setPosition(int x, int y) – координаты (x;y) относительно контейнера компонента,
setBgColor(int color) – цвет фона компонента,
setColor(int color) – цвет компонента
setTag() – тег компонента.

Классы компонентов и их использование:

1.Labe – метка.

Используется для простого вывода текста.
Создание метки производится с помощью конструктора
Label(string text = " ", int x = 0, int y = 0).
Метка может переключаться между режимами фона с помощью метода useParentBgColor(bool use), где при передаче методу true в качестве цвета фона

метки берётся цвет фона контейнера в котором находится метка, иначе используется цвет фона метки.
Текст метки можно изменить с помощью метода setText(string text), а получить текущий текст (string) метки можно с помощью метода getText()

2.Image – изображение.

Используется для простого вывода изображения.
Создание изображения производится с помощью конструктора Image(string path, int x = 0, int y = 0, bool useMask = false), где если useMask будет иметь значение true, то при визуализации изображения будет использована маска (Прозрачный фон).

3.TextInputField поле ввода текста.

Используется для ввода данных с клавиатуры.
Размер вводимых данных ограничен размером поля ввода.
Создание метки производится с помощью конструктора TextInputField(string text = " ", int x = 0, int y = 0)
Можно установить текст который находится в поле ввода с помощью метода setText(string text), а также получить текущий текст (string) поля можно с помощью метода getText().

Container – абстрактный класс родитель для контейнеров, может содержать в себе любое количество компонентов и других контейнеров.

–С помощью метода addComponent(Component *component); можно добавить компонент в контейнер;

–Метод notifyAll(Event *event) сообщает слушателям контейнера и слушателям компонентов в контейнере о событии;

–Метод setVisible(bool visible) устанавливает видимость контейнера и дочерних элементов;

–Метод renderAll(int rootX, int rootY, int rootWidth, int rootHeight) вызывает метод rende() контейнера и компонентов в контейнере;

–Метод getComponents() возвращает динамический массив компонентов в контейнере (vector<Component *> *).

Классы контейнеров и их использование:

1.Screen – экран.

Основной контейнер, запускает графический интерфейс и считывает события мыши и клавиатуры. Содержит слушателей работы с окнами.
Создать экран можно с помощью конструктора Screen().

2.Window – окно.

Контейнер с заголовком.
Может быть создан с помощью конструктора
Window(string title = " ", int width = 250, int height = 200).
Метод isSelected() возвращает true если окно является верхним и активным, а метод setSelected(bool selected) устанавливает статус окна.
Метод getTitle() возвращает текст(string) заголовка окна.

3.Button – кнопка.

Контейнер, реагирующий на действия мыши.

Может быть создан с помощью конструктора
Button(int x = 0, int y = 0, int width = 100, int height = 25)
С помощью метода setCheckColor(int color) можно установить цвет устанавливаемого при наведении мыши на кнопку.
С помощью метода setPressedColor(ште сщдщк) можно установить цвет устанавливаемого при зажатии левой кнопки мыши на кнопке.

4.TextButton – кнопка с текстом.

Наследник Button. Содержит в себе метку.
Создать кнопку с текстом можно с помощью конструктора
TextButton(string text = " " ,int x = 0, int y = 0, int width=100, int height=25)
Можно установить текст метки кнопки с помощью метода setText(string text), а также получить текущий текст (string) метки кнопки можно с помощью метода getText().

Для добавления своего функционала к компонентам пользователь должен реализовать класс слушатель.

Listener – интерфейс содержит виртуальный метод onEvent(Event *event).

В библиотеке присутствуют некоторые абстрактные классы слушатели слушающие определённые события:

1.OnClickListener – Слушатель клика мыши. Содержит виртуальный метод onClick(MouseEvent *event).

2.OnPressListener – Слушатель зажатия левой кнопки мыши. Содержит виртуальный метод onPress(MouseEvent *event).

3.OnDragListener – Слушатель перетаскивания мыши с зажатой левой клавишей. Содержит виртуальный метод onDrag(MouseEvent *event).

4.OnMoveListener – Слушатель перетаскивания мыши. Содержит виртуальный метод onMove(MouseEvent *event).

5.OnKeyUpListener – Слушатель событий клавиатуры, нажатия клавиш. Содержит виртуальный метод onKeyUp(KeyboardEvent *event).

Слушатели содержат информацию в классах реализующих интерфейс метку Event. В библиотеке присутствуют две реализации этого интерфейса.

MouseEvent – событие мыши. Содержит в себе данные о текущей и предыдущей позиции мыши, которые могут быть получены с помощью методов getX(),getY(),getPreX(),getPreY(). Также событие мыши имеет свой тип указанный с помощью перечисления Type и может принимать значения: MOVE, DRAG, LEFT_KEY_DOWN, LEFT_KEY_UP. Чтобы получить текущий тип события можно воспользоваться методом getType().

KeyboardEvent – событие клавиатуры. Содержит в себе код нажатой клавиши и предыдущей нажатой клавиши. Изначально код предыдущей клавиши равен 0. Коды клавиш могут быть получены с помощью методов getKeyCode() и getPreKeyCode().

Код пример использования библиотеки

```
#include <cstdlib>
#include "Screen.hpp"

using namespace std;
```

```
//Простой тест, создание окна с меткой, "Привет мир!" и заголовком
"Окно теста 1"
void test1(Screen *screen);
//Тест в виде программы калькулятора
void test2(Screen *screen);

int main() {
   initwindow(1280,720);
   Screen *screen = new Screen();
   test1(screen);
   test2(screen);
   screen->start();
   delete screen;
   return 0;
}

void test1(Screen *screen) {
   Window *window = new Window("Окно теста 1",150,150);
   window->setTag("Test 1 Window");
   window->setPosition(screen->getWidth()/2 - window-
>getWidth()/2, screen->getHeight()/2 - window->getHeight()/2);
   Label *label = new Label("Привет мир!");
   label->setTag("Hello World Label  ");
   label->setPosition(150/2 - label->getWidth()/2, 150/2 - label-
>getHeight()/2);
   window->addComponent(label);
   screen->addComponent(window);
}

class TestTextButtonListener : public OnClickListener {
public:
   TestTextButtonListener(Label *l, TextInputField *i, Window *w)
{
      this->l = l;
      this->i = i;
      this->w = w;
   }
   Label *l;
   TextInputField *i;
   Window *w;
   void onClick(MouseEvent *event) {
      l->setText(i->getText());
      l->setPosition(w->getWidth()/2 - l->getWidth()/2, w-
>getHeight()/2 + l->getHeight()*2);
   }
};

void test2(Screen *screen) {
   Window *window = new Window("Окно теста 2");
```

```
   window->setPosition(screen->getWidth()/2 - window-
>getWidth()/2, screen->getHeight()/2 - window->getHeight()/2);
   Label *label1 = new Label("Введённый текст:");
   label1->setPosition(window->getWidth()/2 - label1-
>getWidth()/2, window->getHeight()/2 - label1->getHeight()/2);
   Label *label2 = new Label();
   label2->setPosition(window->getWidth()/2 - label2-
>getWidth()/2, window->getHeight()/2 + label2->getHeight()*2);
   TextInputField *in = new TextInputField();
   in->setWidth(100);
   in->setPosition(window->getWidth()/2 - in->getWidth()/2,8);
   TextButton *button = new TextButton("Принять текст");
   button->setPosition(window->getWidth()/2 - button-
>getWidth()/2,window->getHeight() - button->getHeight() - 8);

   button->addListener(new
TestTextButtonListener(label2,in,window));
   window->addComponent(in);
   window->addComponent(label1);
   window->addComponent(label2);
   window->addComponent(button);
   screen->addComponent(window);
}
```

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были выявлены объекты предметной области и определена система классов для них. 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 библиотеки 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.Приемы объектно-ориентированного проектирования. Паттерны проектирования /Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес; пер. с англ.: А. Слинкин, науч. ред.: Н. Шалаев. — Санкт-Петербург: Питер, 2014. — 366с.

ПРИЛОЖЕНИЕ А

А.1 COMPONENT.HPP

```
#include<vector>
#include"Listener.hpp"
#include"graphics.h"

#ifndef COMPONENT_HPP
#define COMPONENT_HPP

using namespace std;

class Component { //Компонент
public:
   virtual ~Component(); //Деструктор
   void addListener(Listener *listener); //Добавляет слушателя
   void notify(Event *event); //Сообщить слушателям компонента о
событии
   void render(int rootX, int rootY, int rootWidth, int
rootHeight);//Визуализация и обновление абсолютной позиции
   virtual void draw(int rootWidth, int rootHeight) {} //Метод
отрисовки компонента
   //Возвращает…
   bool isVisible() const; // true если компонент видимый
   int getWidth() const; // ширину компонента
   int getHeight() const; // высоту компонента
   int getX() const; // позицию x компонента в родительском
контейнере
   int getY() const; // позицию y компонента в родительском
контейнере
   int getAbsolutX() const; // абсолютную позицию x
   int getAbsolutY() const; // абсолютную позицию y
   string getTag() const; // тег элемента
   //Устанавливает
   void setVisible(bool visible); // Видимость элемента
   void setWidth(int width); // Ширину элемента
   void setHeight(int height); // Высоту элемента
   void setX(int x); // Позицию x компонента в родительском
контейнере
   void setY(int y); // Позицию y компонента в родительском
контейнере
   void setPosition(int x, int y);// Позицию x и y компонента в
родительском контейнере
   void setBgColor(int color);// Цвет фона компонента
   void setColor(int color); // Основной цвет отрисовки компонента
   void setTag(string tag); // Тег элемента
protected:
   string tag = "untag"; // Тег элемента
   bool visible = true; //Видимость объекта
```

```
   int width = 0, height = 0, x = 0, y = 0,absolutX = 0, absolutY
= 0;// ширина, высота, позиция компонента относительно
родительского контейнера, абсолютная позиция компонента
   int bgColor = LIGHTGRAY; //Цвет фона, по умолчанию светло-серый
   int color = BLACK; //Основной цвет рисования, по умолчанию
чёрный
private:
   vector<Listener*> listeners; //Динамический массив слушателей
};

#endif
```

A.2 COMPONENT.CPP

```
#include "Component.hpp"
#include <iostream>

using namespace std;

Component::~Component() {
   cout << "delete " << tag << "...\n";
   for (Listener *listener : listeners) {
      delete listener;
   }
}

void Component::addListener(Listener *listener) {
   listeners.push_back(listener);
}


void Component::notify(Event *event) {
   for (Listener *listener : listeners) {
      listener->onEvent(event);
   }
}

bool Component::isVisible() const {
   return visible;
}

void Component::render(int rootX, int rootY, int rootWidth, int
rootHeight) {
   absolutX = rootX + x;
   absolutY = rootY + y;
   if (visible == false)
      return;
   draw(rootWidth, rootHeight);
}

int Component::getWidth() const {
   return width;
```

```
}

int Component::getHeight() const {
   return height;
}

int Component::getX() const {
   return x;
}

int Component::getY() const {
   return y;
}

int Component::getAbsolutX() const {
   return absolutX;
}

int Component::getAbsolutY() const {
   return absolutY;
}

string Component::getTag() const {
   return tag;
}

void Component::setVisible(bool visible) {
   this->visible = visible;
}

void Component::setWidth(int width) {
   this->width = width;
}
void Component::setHeight(int height) {
   this->height = height;
}

void Component::setX(int x) {
   this->x = x;
}

void Component::setY(int y) {
   this->y = y;
}

void Component::setPosition(int x, int y) {
   this->x = x;
   this->y = y;
}

void Component::setBgColor(int color) {
```

```
   bgColor = color;
}

void Component::setColor(int color) {
   this->color = color;
}

void Component::setTag(string tag) {
   this->tag = tag;
}
```

A.3 CONTAINER.HPP

```
#include "Component.hpp"

#ifndef CONTAINER_HPP
#define CONTAINER_HPP

using namespace std;

class Container : public Component {//Контейнер
public:
   Container(); //Конструктор
   virtual ~Container(); //Деструктор
   void addComponent(Component *component); //Добавить компонент в
контейнер
   void notifyAll(Event *event); //Вызывает свой notify, notify
компонентов в контейнере и notifyAll контейнеров в контейнере
   void setVisible(bool visible);//Устанавливает видимость
контейнера и дочерних элементов
   void renderAll(int rootX, int rootY, int rootWidth, int
rootHeight); //Вызывает свой render, render компонентов в
контейнере и renderAll контейнеров в контейнере
   vector<Component *> *getComponents() const; //Возращает
динамический массив компонентов
protected:
   void draw(int rootWidth, int rootHeight) {} //Метод отрисовки
контейнера
private:
   vector<Component *> *components; //Динамический массив
компонентов контейнера
};

#endif
```

A.4 CONTAINER.CPP

```
#include "Container.hpp"
#include "Window.hpp"

Container::Container(){
components = new vector<Component*>();
}

Container::~Container(){
   for (int i = 0; i < components->size(); i++) {
      delete components->at(i);
   }
   delete components;
}

void Container::notifyAll(Event *event) {
   notify(event);
   if (Window *container = dynamic_cast<Window*>(this))
      if(!container->isSelected())
         return;
   for (int i = 0; i < components->size(); i++) {
      Component* component = components->at(i);
      if (Container *container = dynamic_cast<Container
*>(component))
         container->notifyAll(event);
      else
         component->notify(event);
   }
}

void Container::setVisible(bool visible){
   this->visible = visible;
   for (int i = 0; i < components->size(); i++) {
      components->at(i)->setVisible(visible);
   }
}

void Container::renderAll(int rootX, int rootY, int rootWidth, int
rootHeight) {
   render(rootX,rootY,rootWidth,rootHeight);
   for (int i = 0; i < components->size(); i++) {
      Component* component = components->at(i);
      if (Container *container =
dynamic_cast<Container*>(component)){
         container->renderAll(absolutX, absolutY, width, height);
      }else
         component->render(absolutX, absolutY, width, height);
   }
}

void Container::addComponent(Component* component){
```

```
   if(!visible)
      component->setVisible(false);
   components->push_back(component);
}

vector<Component*> *Container::getComponents() const{
   return components;
}
```

A.5 LABEL.HPP

```
#include "Container.hpp"

#ifndef LABEL_HPP
#define LABEL_HPP

/*
*Метка (текст)
*/
class Label : public Component{
   public:
      Label(string text = " ", int x = 0, int y = 0);//Конструктор
      virtual ~Label(){};//Деструктор
      void useParentBgColor(bool use);//Использовать цвет фона
родительского элемента
      void setText(string text);//Устанавливает текст метки
      string getText() const;//Возвращает текст метки
   protected:
      void draw(int rootWidth, int rootHeight);//Отрисовывает
метку
   private:
      string text;//текст метки
      bool useParentBgColorFlag = true;//флаг использования цвета
фона родительского элемента, вместо своего
};

#endif
```

A.6 LABEL.CPP

```
#include "Label.hpp"

Label::Label(string text, int x, int y){
   setText(text);
   this->x = x;
   this->y = y;
   tag = "Label";
}

void Label::useParentBgColor(bool use){
   useParentBgColorFlag = use;
}
```

```
void Label::setText(string text){
   this->text = text;
   height = textheight(text.c_str());
   width = textwidth(text.c_str());
}

string Label::getText() const{
   return text;
}

void Label::draw(int rootWidth, int rootHeight){
   setbkcolor(useParentBgColorFlag ? getpixel(absolutX +
width/2,absolutY + height/2) : bgColor);
   setcolor(color);
   outtextxy(absolutX, absolutY, text.c_str());
}
```

A.7 IMAGE.HPP

```
#include"Component.hpp"

#ifndef IMAGE_HPP
#define IMAGE_HPP

class Image : public Component {
public:
   Image(string path, int x = 0, int y = 0, bool useMask =
false);//Конструктор
   virtual ~Image();//Деструктор
   void draw(int rootWidth, int rootHeight);//Отрисовка
изображения
private:
   bool useMask;//флаг использования маски (альтернатива альфа
канала)
   IMAGE *img, *mask;//изображение и маска
   IMAGE *createmask(IMAGE *p);//создание маски
};

#endif
```

A.8 IMAGE.CPP

```
#include "Image.hpp"

Image::Image(string path, int x, int y, bool useMask) {
   this->x = x;
   this->y = y;
   this->useMask = useMask;
   img = loadBMP(path.c_str());
   width = imagewidth(img);
   height = imageheight(img);
   if (useMask)
      mask = createmask(img);
```

```
  tag = "Image";
}

Image::~Image() {
   freeimage(img);
   if (useMask)
      freeimage(mask);
}

IMAGE *Image::createmask(IMAGE *p) {
   IMAGE *m=createimage(width,height);
   int c=imagegetpixel(p,0,0);
   for (int x=0; x<width; ++x)
      for (int y=0; y<height; ++y)
      {  int d=imagegetpixel(p,x,y);
         if (c==d)
         {  imageputpixel(m,x,y,WHITE);
            imageputpixel(p,x,y,BLACK);
         }
         else
            imageputpixel(m,x,y,BLACK);
      }
   return m;
}

void Image::draw(int rootWidth, int rootHeight) {
   putimage(absolutX,absolutY,mask,AND_PUT);
   putimage(absolutX,absolutY,img,OR_PUT);
}
```

A.9 TEXTINPUTFIELD.HPP

```
#include "Container.hpp"

#ifndef TEXTINPUTFIELD_HPP
#define TEXTINPUTFIELD_HPP

/*
*Поле ввода текста
*/
class TextInputField : public Component {
public:
   TextInputField(string text = " ", int x = 0, int y =
0);//Конструктор
   virtual ~TextInputField() {}; //Деструктор
   void setText(string text);//Устанавливает текст поля
   string getText() const;//Возвращает текст поля
protected:
   void draw(int rootWidth, int rootHeight);//Отрисовка
private:
   string text;//Текст поля
   class InputOnKeyListener : public OnKeyUpListener {
```

```
   public:
      InputOnKeyListener(TextInputField *field);//Конструктор
   private:
      TextInputField *field;//Родительское поле ввода
      void onKeyUp(KeyboardEvent *event);//Обработка события ввода
   };
};

#endif
```

A.10 TEXTINPUTFIELD.CPP

```
#include "TextInputField.hpp"

TextInputField::TextInputField(string text, int x, int y){
   this->text = text;
   this->x = x;
   this->y = y;
   width = 150;
   height = 25;
   bgColor = WHITE;
   InputOnKeyListener *iokl = new InputOnKeyListener(this);
   addListener(iokl);
   tag = "TextInputField";
}

void TextInputField::setText(string text){
   this->text = text;
}

string TextInputField::getText() const{
   return text;
}

void TextInputField::draw(int rootWidth, int rootHeight){
   setcolor(color);
   setfillstyle(SOLID_FILL,bgColor);
   setbkcolor(bgColor);

   bar(absolutX,absolutY,absolutX + width, absolutY + height);
   rectangle(absolutX,absolutY,absolutX + width, absolutY +
height);
   outtextxy(absolutX+2,absolutY+height/2 -
textheight(text.c_str())/2,text.c_str());
}

TextInputField::InputOnKeyListener::InputOnKeyListener(TextInputFi
eld *field){
   this->field = field;
}
```

```
void TextInputField::InputOnKeyListener::onKeyUp(KeyboardEvent
*event){
   if (event->getKeyCode() == BACKSPACE){
      field->text.erase(field->text.end()-1);
   }
   if (event->getKeyCode() > MIN_CHAR && event->getKeyCode() <
MAX_CHAR){
      field->text+= event->getKeyCode();
      if(textwidth((field->text).c_str()) > field->getWidth())
         field->text.erase(field->text.end()-1);
   }
}
```

A.11 SCREEN.HPP

```
#include "Container.hpp"
#include "Button.hpp"
#include "Window.hpp"
#include "TextInputField.hpp"
#include "Image.hpp"

#ifndef SCREEN_HPP
#define SCREEN_HPP

class Screen : public Container {//Экран, основа отображения
элементов
public:
   Screen();//Конструктор
   ~Screen() {}; //Деструктор
   void start();//Инициализация графического интерфейса
private:
   void draw(int rootWidth, int rootHeight);//Метод отрисовки
   bool run;//Флаг, при false приложение завершает свою работу
   void checkMouse();//Проверка событий мыши, уведомляет о них
слушателей
   void checkKeyboard();//Проверка событий клавиатуры, уведомляет
о них слушателей
   class ScreenOnKeyUpListener : public OnKeyUpListener {
   public:
      ScreenOnKeyUpListener(Screen *screen);//Конструктор
   private:
      Screen *screen;//Родительский экран
      void onKeyUp(KeyboardEvent *event);//Реализация завершения
работы Screen  при нажатии клавиши ESC
   };
   class ScreenOnSelectListner : public OnClickListener {
   public:
      ScreenOnSelectListner(Screen *screen);//Конструктор
   private:
      Screen *screen;//Родительский экран
      void replaceSelectables(int pos);//Перемещение окна в верх
массива
```

```
      void onClick(MouseEvent *event);//Выбор текущего активного
окна
   };
   class AboutButton : public Button {
   public:
      AboutButton(Screen *screen);//Конструктор
      virtual ~AboutButton() {}; //Деструктор
   private:
      class AboutButtonOnClickListener : public OnClickListener {
      public:
         AboutButtonOnClickListener(AboutButton *button,Screen
*screen);//Конструктор
      private:
         Screen *screen; //Кореной элемент
         AboutButton *button;//Родительская кнопка
         void onClick(MouseEvent *event); //Открытие окна "о
библиотеке"
      };
   };
};

#endif
```

A.12 SCREEN.CPP

```
#include "Screen.hpp"

Screen::Screen() {
   width = getmaxx();
   height = getmaxy();
   ScreenOnKeyUpListener *sokul = new ScreenOnKeyUpListener(this);
   ScreenOnSelectListner *sosl = new ScreenOnSelectListner(this);
   addListener(sokul);
   addListener(sosl);
   x = 0;
   y = 0;
   tag = "Screen";
}

void Screen::start() {
   AboutButton *ab = new AboutButton(this);
   addComponent(ab);
   run = true;
   int p=0;
   while (run) {
      checkMouse();
      checkKeyboard();
      p=1-p;
      setactivepage(p);
      setbkcolor(LIGHTBLUE);
      renderAll(x,y,width,height);
      setvisualpage(p);
```

```
      delay(1);
   }
}

void Screen::checkMouse() {
   static int preX, preY;
   static MouseEvent::Type preType;
   MouseEvent::Type type;

   if (mousebuttons() == 0) {
      if (preType == MouseEvent::LEFT_KEY_DOWN || preType ==
MouseEvent::DRAG)
         type = MouseEvent::LEFT_KEY_UP;
      else
         type = MouseEvent::MOVE;
   } else {
      if (preType == MouseEvent::MOVE)
         type = MouseEvent::LEFT_KEY_DOWN;
      else {
         type = MouseEvent::DRAG;
      }
   }

   MouseEvent *event = new
MouseEvent(mousex(),mousey(),type,preX,preY);
   notifyAll(event);
   delete event;
   preX = mousex();
   preY = mousey();
   preType = type;
}

void Screen::checkKeyboard() {
   static int preKeyCode;
   if (kbhit() == 0)
      return;
   int keyCode = getch();
   KeyboardEvent *event = new KeyboardEvent(keyCode, preKeyCode);
   notifyAll(event);
   delete event;
   preKeyCode = keyCode;
}

void Screen::draw(int rootWidth, int rootHeight) {
   clearviewport();
}

Screen::ScreenOnKeyUpListener::ScreenOnKeyUpListener(Screen
*screen) {
   this->screen = screen;
}
```

```
void Screen::ScreenOnKeyUpListener::onKeyUp(KeyboardEvent *event)
{
   if (event->getKeyCode() == ESC) {
      for (int i = 0; i < screen->getComponents()->size(); i++)
         if (Window *w = dynamic_cast<Window *>(screen-
>getComponents()->at(i)))
            if (w->isSelected()) {
               screen->getComponents()->erase(screen-
>getComponents()->begin() + i);
               delete w;
               return;
            }
      screen->run = false;
   }
}

Screen::ScreenOnSelectListner::ScreenOnSelectListner(Screen
*screen) {
   this->screen = screen;
}

void Screen::ScreenOnSelectListner::replaceSelectables(int pos) {
   for (int i = pos; i < screen->getComponents()->size() - 1; i++)
{
      auto buffer = screen->getComponents()->at(i+1);
      screen->getComponents()->at(i+1) = screen->getComponents()-
>at(i);
      screen->getComponents()->at(i) = buffer;
   }
}

void Screen::ScreenOnSelectListner::onClick(MouseEvent *event) {
   bool selectFlag = true;
   int n = -1;
   for (int i = screen->getComponents()->size() - 1; i >= 0; i--)
{
      Component *component = screen->getComponents()->at(i);
      if (Window *w = dynamic_cast<Window *>(component)) {
         w->setSelected(false);
         if (component->isVisible() && selectFlag && component-
>getX() < event->getX() &&
               component->getWidth() + component->getX() > event-
>getX() &&
               component->getY() - textheight(w-
>getTitle().c_str())< event->getY() &&
               component->getHeight() + component->getY() > event-
>getY()) {
            selectFlag = false;
            if (!w->isSelected()) {
               w->setSelected(true);
```

```
                n = i;
            }
         }
      }
   }
   if (n != -1) {
      replaceSelectables(n);
   }
}

Screen::AboutButton::AboutButton(Screen *screen) : Button() {
   Image *img = new Image("res/about.bmp",4,4,true);
   addComponent(img);
   AboutButtonOnClickListener *abocl = new
AboutButtonOnClickListener(this, screen);
   addListener(abocl);

   x = screen->getWidth() - img->getWidth() - 8;
   y = screen->getHeight() - img->getHeight() - 8;
   width = screen->getWidth() - x;
   height = screen->getHeight() - y;
}

Screen::AboutButton::AboutButtonOnClickListener::AboutButtonOnClic
kListener(AboutButton *button, Screen *screen) {
   this->button = button;
   this->screen = screen;
}

void
Screen::AboutButton::AboutButtonOnClickListener::onClick(MouseEven
t *event) {
   if (button->absolutX < event->getX() && event->getX() < button-
>absolutX + button->width &&
         button->absolutY < event->getY() && event->getY() <
button->absolutY + button->height) {

      Window *helpWindow = new Window("Информация о библиотеке",
400, 100);//Синевато(black) бледная куртка, розовая шапка,
разноцветный портфель
      helpWindow->setPosition(screen->getWidth()/2 - helpWindow-
>getWidth()/2,
                              screen->getHeight()/2 - helpWindow-
>getHeight()/2);

      Label *l = new Label("Данная библиотека является курсовой
работой", 8, 8);
      helpWindow->addComponent(l);
      helpWindow->addComponent(new Label("студента ЮУрГУ, ИЕТН,
группы ЕТ-212", 8, l->getHeight() + 8));
```

```
      helpWindow->addComponent(new Label("Шерстобитова Тимофея
Сергеевича", 8, l->getHeight()*2 + 8));
      helpWindow->addComponent(new Label("Email:
woodgoldfilm@gmail.com", 8, l->getHeight()*3 + 8));
      helpWindow->addComponent(new Label("VK:
https://vk.com/ozymand", 8, l->getHeight()*4 + 8));
      screen->addComponent(helpWindow);
   }
}
```

A.13 WINDOW.HPP

```
#include "Container.hpp"

#ifndef WINDOW_HPP
#define WINDOW_HPP
/**
*Окно, является контейнером для других компонентов.
*Может иметь заголовок.
*/
class Window : public Container {
public:
   Window(string title = " ", int width = 250, int height =
200);//Конструктор
   virtual ~Window(){};//Деструктор
   bool isSelected() const;//Возвращает true, если окно является
активным
   void setSelected(bool selected);//Устанавливает статус
активности окна
   string getTitle() const;//возвращает текст заголовка окна
protected:
   void draw(int rootWidth, int rootHeight);//отрисовывает окно,
унаследовано Container
private:
   bool selected = false;//статус активности окна
   bool dragedNow = false;//флаг перемещения окна
   string title;//текст заголовка окна
   class WindowOnDragListener : public OnDragListener { //вложеный
класс, реализующий слушатель перетаскивания мышью
   public:
      WindowOnDragListener(Window *window);//Конструктор
   private:
      Window *window;//окно слушателя
      void onDrag(MouseEvent *event);//перемещение окна
   };
   class WindowOnClickListener : public OnClickListener {
//вложеный класс, реализующий слушатель шелчка левой кнопки мыши
   public:
      WindowOnClickListener(Window *window);//Конструктор
   private:
      Window *window;//окно слушателя
      void onClick(MouseEvent *event);//снятие флага перемещения
```

```
   };
};

#endif
```

A.14 WINDOW.CPP

```
#include "Window.hpp"

Window::Window(string title, int width, int height) {
   this->title = title;
   this->width = width;
   this->height = height;
   WindowOnDragListener *wodl = new WindowOnDragListener(this);
   WindowOnClickListener *wocl = new WindowOnClickListener(this);
   addListener(wodl);
   addListener(wocl);
   tag = "Window";
}

bool Window::isSelected() const {
   return selected;
}

void Window::setSelected(bool selected) {
   this->selected = selected;
}

string Window::getTitle() const {
   return title;
}

void Window::draw(int rootWidth, int rootHeight) {
   setcolor(color);
   if (selected) {
      setfillstyle(SOLID_FILL, WHITE);
      setbkcolor(WHITE);
   } else {
      setfillstyle(SOLID_FILL, DARKGRAY);
      setbkcolor(DARKGRAY);
   }

   int w = rootWidth > width ? width : rootWidth, h = rootHeight >
height ? height : rootHeight;

   bar(absolutX, absolutY - textheight(title.c_str()), absolutX +
w, absolutY);
   outtextxy(absolutX + 2, absolutY - textheight(title.c_str()),
title.c_str());
   rectangle(absolutX, absolutY - textheight(title.c_str()),
absolutX + w, absolutY);
```

```
   setfillstyle(SOLID_FILL, bgColor);
   bar(absolutX, absolutY,absolutX + w,
       absolutY + h);
   rectangle(absolutX, absolutY,absolutX + w,
             absolutY + h);

}

Window::WindowOnDragListener::WindowOnDragListener(Window *window)
{
   this->window = window;
}

void Window::WindowOnDragListener::onDrag(MouseEvent *event) {
   if (!window->selected)
      return;
   if (window->x < event->getX() && event->getX() < window->x +
window->width &&
         window->y+2 > event->getY() && event->getY() > window->y
- textheight(window->title.c_str()))
      window->dragedNow = true;
   if (window->dragedNow)
      window->setPosition(event->getX() - window->width/2, event-
>getY());
}

Window::WindowOnClickListener::WindowOnClickListener(Window
*window) {
   this->window = window;
}

void Window::WindowOnClickListener::onClick(MouseEvent *event) {
   window->dragedNow = false;
}
```

A.15 BUTTON.HPP

```
#include "Container.hpp"
#include "Label.hpp"
#ifndef BUTTON_HPP
#define BUTTON_HPP

class Button : public Container {
public:
   Button(int x = 0, int y = 0, int width = 100, int height =
25);// Конструктор
   virtual ~Button() {}; // Деструктор
   void setCheckColor(int color); // Установка цвета
устанавливаемого при наведении мыши на кнопку
   void setPressedColor(int color); // Установка цвета
устанавливаемого при зажатии левой кнопки мыши на кнопке
   protected:
```

```
   void draw(int rootWidth, int rootHeight); // Отрисовка
private:
   int checkColor = RGB(220,220,220); //Цвет устанавливаемый при
наведении мыши на кнопку, по умолчанию светлосерый
   int pressedColor = DARKGRAY; //Цвет устанавливаемый прри
зажатии кнопки
   enum Stage {
      UP, CHECK, DOWN
   };
   Stage stage = UP;//Стадия кнопки, по умолчанию UP
   class onMoveInButtonListener : public OnMoveListener {
   public:
      onMoveInButtonListener(Button *button); //Конструктор
   private:
      Button *button;//Родительская кнопка
      void onMove(MouseEvent *event); //Установка стадии на CHECK
при движении на кнопке
   };
   class onPressedButtonListener : public OnPressListener {
   public:
      onPressedButtonListener(Button *button);//Конструктор
   private:
      Button *button;//Родительская кнопка
      void onPress(MouseEvent *event);//Изменение стадии кнопки на
DOWN когда кнопка зажата
   };
   class ButtonOnClickListener : public OnClickListener {
   public:
      ButtonOnClickListener(Button *button);//Конструктор
   private:
      Button *button;//Родительская кнопка
      void onClick(MouseEvent *event); //Установка стадии на CHECK
при клике на кнопке
   };
};

class TextButton : public Button{
public:
   TextButton(string text = " " ,int x = 0, int y = 0, int width
=100, int height = 25);//Конструктор
   virtual ~TextButton(){};//Деструктор
   string getText() const;//Возвращает текст кнопки
   void setText(string text);//Установка текста кнопки
private:
   Label *buttonLabel;//Метка текстовой кнопки
};
#endif
```

A.16 BUTTON.CPP

```
#include "Button.hpp"

Button::Button(int x, int y, int width, int height) {
   this->x = x;
   this->y= y;
   this->width = width;
   this->height = height;
   onMoveInButtonListener *omibl = new
onMoveInButtonListener(this);
   onPressedButtonListener *opbl = new
onPressedButtonListener(this);
   ButtonOnClickListener *bocl = new ButtonOnClickListener(this);
   addListener(omibl);
   addListener(opbl);
   addListener(bocl);
   tag = "Button";
}

void Button::setCheckColor(int color) {
   checkColor = color;
}

void Button::setPressedColor(int color) {
   pressedColor = color;
}

void Button::draw(int rootWidth, int rootHeight) {
   int c;
   switch (stage) {
   case UP:
      c = bgColor;
      break;
   case CHECK:
      c = checkColor;
      break;
   case DOWN:
      c = pressedColor;
      break;
   }

   setcolor(color);
   setfillstyle(SOLID_FILL, c);
   bar(absolutX,absolutY,absolutX + width, absolutY + height);
   rectangle(absolutX,absolutY,absolutX + width, absolutY +
height);
}

Button::onMoveInButtonListener::onMoveInButtonListener(Button
*button) {
   this->button = button;
```

```
}

void Button::onMoveInButtonListener::onMove(MouseEvent *event) {
   if (button->stage == DOWN)
      return;
   if (button->absolutX < event->getX() && event->getX() < button-
>absolutX + button->width &&
         button->absolutY < event->getY() && event->getY() <
button->absolutY + button->height)
      button->stage = CHECK;
   else
      button->stage = UP;
}

Button::onPressedButtonListener::onPressedButtonListener(Button
*button) {
   this->button = button;
}

void Button::onPressedButtonListener::onPress(MouseEvent *event) {
   if (button->absolutX < event->getX() && event->getX() < button-
>absolutX + button->width &&
         button->absolutY < event->getY() && event->getY() <
button->absolutY + button->height)
      button->stage = DOWN;
}

Button::ButtonOnClickListener::ButtonOnClickListener(Button
*button) {
   this->button = button;
}

void Button::ButtonOnClickListener::onClick(MouseEvent *event) {
      button->stage = UP;
}

TextButton::TextButton(string text, int x, int y, int width, int
height) : Button(x,y,width,height) {
   buttonLabel = new Label();
   setText(text);
   addComponent(buttonLabel);
   tag = "TextButton";
}

string TextButton::getText() const {
   return buttonLabel->getText();
}

void TextButton::setText(string text) {
   buttonLabel->setText(text);
```

```
   buttonLabel->setPosition(width/2 - buttonLabel->getWidth()/2,
height/2 - buttonLabel->getHeight()/2);
}
```

A.17 LISTENER.HPP

```
#include "Event.hpp"
#ifndef Listener_HPP
#define Listener_HPP

class Listener {//Слушатель
public:
    virtual ~Listener(){}//Деструктор
    virtual void onEvent(Event *event){}//Действие при событии
};

class OnClickListener : public Listener {//Слушатель клика мыши
protected:
    virtual void onClick(MouseEvent *event){};//Действие при клике
мышкой
private:
    void onEvent(Event *event);//Проверка было ли событие кликом
мыши
};

class OnPressListener : public Listener {//Слуушатель зжатия
кнопки мыши
protected:
    virtual void onPress(MouseEvent *event){};//Действие при
зажатии кнопки мыши
private:
    void onEvent(Event *event);//Проверка было ли событие зажатием
кнопки мыши
};

class OnDragListener : public Listener {//Слуушатель
перетаскивания зажатой мыши
protected:
    virtual void onDrag(MouseEvent *event){};//Действие при
перетаскивании зажатой мыши
private:
    void onEvent(Event *event);//Проверка было ли событие
перетаскиванием зажатой мыши
};

class OnMoveListener : public Listener {//Слуушатель передвижения
мыши
protected:
    virtual void onMove(MouseEvent *event){};//Действие при
передвижении мыши
private:
```

```
    void onEvent(Event *event);//Проверка было ли событие
передвижением мыши
};

class OnKeyUpListener : public Listener {//Слуушатель событий
клавиатуры
protected:
    virtual void onKeyUp(KeyboardEvent *event){};// Действие при
событии клавиатуры
private:
    void onEvent(Event *event);//Проверка является ли событие
событием клавиатуры
};

#endif
```

A.18 LISTENER.CPP

```
#include "Listener.hpp"

void OnClickListener::onEvent(Event *event) {
   if (MouseEvent *me = dynamic_cast<MouseEvent *>(event))
      if (me->getType() == MouseEvent::LEFT_KEY_UP)
         onClick(me);
}

void OnDragListener::onEvent(Event *event) {
   if (MouseEvent *me = dynamic_cast<MouseEvent *>(event))
      if (me->getType() == MouseEvent::DRAG)
         onDrag(me);
}

void OnKeyUpListener::onEvent(Event *event){
   if(KeyboardEvent* ke = dynamic_cast<KeyboardEvent*>(event)){
      onKeyUp(ke);
   }
}

void OnMoveListener::onEvent(Event *event) {
   if (MouseEvent *me = dynamic_cast<MouseEvent *>(event))
      if (me->getType() == MouseEvent::MOVE)
         onMove(me);
}

void OnPressListener::onEvent(Event *event) {
   if (MouseEvent *me = dynamic_cast<MouseEvent *>(event))
      if (me->getType() == MouseEvent::LEFT_KEY_DOWN)
         onPress(me);
}
```

A.19 EVENT.HPP

```
//Коды клавиш клавиатуры
#define ESC 27
#define BACKSPACE 8
#define SPACE 32
#define ENTER 13
#define SHIFT 16
//Код минимального/максимального возможного для отображения
символа
#define MIN_CHAR 32
#define MAX_CHAR 255
#ifndef EVENT_HPP
#define EVENT_HPP

//Интерфейс метка для стуктур данных событий
class Event {
public:
   virtual ~Event() {};
};

//Событие мыши
class MouseEvent : public Event {
public:
   enum Type {
      MOVE, DRAG, LEFT_KEY_DOWN, LEFT_KEY_UP
   };
   MouseEvent(int, int, Type, int = 0, int = 0);//Конструктор
   int getX() const;//Возращает позицию x нажатия мыши
   int getY() const;// Возращает позицию y нажатия мыши
   int getPreX() const;// Возращает предыдущую позицию x нажатия
мыши
   int getPreY() const;// Возращает предыдущую позицию y нажатия
мыши
   Type getType() const;// Возращает тип события мыши
private:
   int x, y, preX, preY;//Текущие x,y, предыдущие x,y
   Type type;//тип события мыши
};

//Событие клавиатуры
class KeyboardEvent : public Event {
public:
   KeyboardEvent(int, int = 0);//Конструктор
   int getKeyCode() const;//Возращает код тукущей нажатой клавиши
   int getPreKeyCode() const;//Возращает предыдущей нажатой
клавишу
private:
   int keyCode, preKeyCode;//Код клавиши, код предыдущей нажатой
клавиши
};
#endif
```

A.20 EVENT.CPP

```
#include "Event.hpp"

KeyboardEvent::KeyboardEvent(int keyCode, int preKeyCode){
   this->keyCode = keyCode;
   this->preKeyCode = preKeyCode;
}

int KeyboardEvent::getKeyCode() const{
   return keyCode;
}

int KeyboardEvent::getPreKeyCode() const{
   return preKeyCode;
}

MouseEvent::MouseEvent(int x, int y, Type type, int preX, int
preY) {
    this->x = x;
    this->y = y;
    this->type = type;
    this->preX = preX;
    this->preY = preY;
}

int MouseEvent::getX() const{
   return x;
}

int MouseEvent::getY() const{
   return y;
}

int MouseEvent::getPreX() const{
   return preX;
}

int MouseEvent::getPreY() const{
   return preY;
}

MouseEvent::Type MouseEvent::getType() const{
   return type;
}
```