## Разработка иерархии и интерфейсов классов

Для реализации поставленной задачи требуется написать множество похожих по поведению компонентов. Следственно все эти компоненты должны наследовать абстрактный класс Component.

Графические объекты отображаются внутри различных контейнеров, т.е. необходимо создание абстрактного класса Container. Контейнеры могут повторять поведения компонентов, поэтому они тоже являются наследниками Component.

Основные наследники класса Component:

1. Container
2. Label (Метка)
3. TextInput

Основные наследники класса Container

1. Screen (Корневой контейнер)
2. Window
3. Button

Класс Buttun является контейнером т.к. может содержать в себе самые различные компоненты, к примеру помимо текста в кнопке может быть иконка. Кнопка изначально содержащая в себе метку, наследуется от Button и имеет название TextButton.

Ниже приведена схема иерархии классов.



Схема Иерархия компонентов

Компоненты должны выполнять определённые программистом функции во время определённых событий. Чтобы добавить гибкость, оптимальным решением является использовать шаблон проектирования «Наблюдатель»(«Слушатель»). Необходимо создать интерфейс Listener. Далее наследовать от него следующих слушателей:

1. OnClickListener
2. OnDragListener
3. OnMoveListener
4. OnPressListener
5. OnKeyUpListener

Слушатели 1-4 являются слушателями действий мыши, 5 слушатель событий клавиатуры.

Ниже приведена схема иерархии классов слушателей.



Схема Иерархия слушателей

Так как слушатели должны получать данные об определённых событиях для их обработки, существует необходимость в объектах которые несут в себе эти данные. Для решения этой проблемы нужен пустой интерфейс Event, от которого наследуются следующие классы:

1. MouseEvent
2. KeyboardEvent

MouseEvent содержит в себе данные о событиях мыши, а KeyboardEvent данные о событиях клавиатуры

Ниже приведена схема иерархии классов событий



Схема Иерархия событий

Класс Screen должен иметь несколько уникальных слушателей. ScreenOnKeyUpListener наследующего OnKeyUpListener и реализующего завершение работы графического интерфейса при нажатии ESC. ScreenOnSelectListener наследующего OnClickListener и реализующего выбор «верхнего окна». Классы ScreenOnKeyUpListener и ScreenOnSelectListener являются вложенными приватными классами класса Screen.

Ниже приведена схема иерархии вложенных классов класса Screen.



Схема Иерархия вложеных классов класса Screen

Класс Window должен иметь несколько уникальных слушателей. WindowOnDragListener наследующего OnDragListener и реализующего установку флага перемещения и перемещение окна по корневому контейнеру. WindowOnClickListener наследующего OnClickListener и реализующего снятие флага перемещения. Классы WindowOnDragListener и WindowOnClickListener являются вложенными приватными классами класса Window.

Ниже приведена схема иерархии вложенных классов класса Window.



Схема Иерархия вложенных классов класса Window

Класс Button должен иметь несколько уникальных слушателей. ButtonOnMoveListener наследующего OnMoveListener и реализующего изменение цвета кнопки в случае наведения на неё мыши. ButtonOnPressListener наследующего OnPressListener и реализующего изменение цвета кнопки в случае зажатия на ней кнопки мыши. ButtonOnClickListener наследующего OnClickListener и реализующего снятие эффекта зажатия кнопки мыши. Классы ButtonOnMoveListener, ButtonOnPressListener и ButtonOnClickListener являются вложенными приватными классами класса Button.

Ниже приведена схема иерархии вложенных классов класса Button.



Схема Иерархия вложенных классов класса Button

Класс TextInput должен иметь одного уникального слушателя. InputOnKeyListener наследующего OnKeyUpListener и реализующего считывание ввода с клавиатуры. Класс InputOnKeyListener является вложенным приватным классом класса TextInput.

Ниже приведена схема иерархии вложенного класса в классе TextInput.



Схема Иерархия вложенного класса в классе TextInput

Ниже представлены объявления классов.

### Component

class Component {

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

//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

protected:

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<Listeners \*> listeners; //Динамический массив слушателей

};

### Container

class Container : public Component {

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 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

};

### Screen, ScreenOnKeyUpListener и ScreenOnSelectListener

class Screen : public Container {

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

};

};

### Window

/\*\*

\*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

bool 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 {

public:

WindowOnDragListener(Window \*window);//Конструктор

private:

Window \*window;//Окно слушателя

void onDrag(MouseEvent \*event);//Перемещение окна

};

class WindowOnClickListener : public OnClickListener {

public:

WindowOnClickListener(Window \*window); //Конструктор

private:

Window \*window;//Окно слушателя

void onClick(MouseEvent \*event);//Снятие флага перемещения

};

};

### Button

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(200,200,200); //Цвет устанавливаемый при наведении мыши на кнопку, по умолчанию светлосерый

int pressedColor = DARKGRAY; //Цвет устанавливаемый

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

};

};

### TextButton

class TextButton : public Button{

public:

TextButton(int x, int y, int width, int height, string text = " ");//Конструктор

virtual ~TextButton();//Деструктор

string getText() const;//Возвращает текст кнопки

void setText(string text);//Установка текста кнопки

private:

Label buttonLabel;//Метка текстовой кнопки

};

### Label

/\*

\*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

};

### TextInput

/\*

\*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

};

};

### Event

//Интерфейс метка для стуктур данных событий

class Event {

public:

virtual ~Event(){};//Деструктор

};

### KeyboardEvent

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

};

### MouseEvent

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

};

### Listener

class Listener {

public:

virtual ~Listener(){}//Деструктор

virtual void onEvent(Event \*event){}//Действие при событии

};

### OnClickListener

class OnClickListener : public Listener {

protected:

virtual void onClick(MouseEvent \*event){};//Действие при клике мышкой

private:

void onEvent(Event \*event);//Проверка было ли событие кликом мышки

};

### OnDragListener

class OnDragListener : public Listener {

protected:

virtual void onDrag(MouseEvent \*event){};// Действие при перетаскивании зажатой мыши

private:

void onEvent(Event \*event);// Проверка было ли событие перетаскиванием зажатой мыши

};

### OnMoveListener

class OnMoveListener : public Listener {

protected:

virtual void onMove(MouseEvent \*event){};// Действие при передвижении мыши

private:

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

};

### OnPressListener

class OnPressListener : public Listener {

protected:

virtual void onPress(MouseEvent \*event){};// Действие при зажатии кнопки мыши

private:

void onEvent(Event \*event);// Проверка было ли событие зажатием кнопки мыши

};

### OnKeyUpListener

class OnKeyUpListener : public Listener {

protected:

virtual void onKeyUp(KeyboardEvent \*event){};// Действие при событии клавиатуры

private:

void onEvent(Event \*event);//Проверка является ли событие событием клавиатуры

};