|  |  |
| --- | --- |
| https://studfiles.net/html/2706/219/html_4NnFGVyFmL.LWVf/img-KuTuVC.png | МИНОБРНАУКИ РОССИИ  федеральное государственное бюджетное образовательное учреждение  высшего образования  **«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»**  **(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова»)** |
| БГТУ.СМК-Ф-4.2-К5-01 |

Факультет И Информационные и управляющие системы

шифр наименование

Кафедра И8 Системы приводов, мехатроника и робототехника

шифр наименование

Дисциплина Правоведение

РЕФЕРАТ

на тему

|  |
| --- |
|  |
| Муниципальное право |
|  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил студент группы | | | | | | *И-882* |
| *Коваленко Е.М.* | | | | | | |
| Фамилия И.О. | | | | | | |
|  | | | | | | |
| **РУКОВОДИТЕЛЬ** | | | | | | |
|  | | | | | | |
| *Попова Н.П.* | | | |  |  | |
| Фамилия И.О. | | | |  | Подпись | |
|  | | | | | | |
| Оценка | | |  | | |  |
| « |  | » |  | | | 2019 г. |

САНКТ-ПЕТЕРБУРГ

2019 г.

# Component-sdl2

[![Language](data:image/svg+xml;charset=utf-8;base64,)](https://isocpp.org/) [![Standard](data:image/svg+xml;charset=utf-8;base64,)](https://en.wikipedia.org/wiki/C%2B%2B#Standardization) [![License](data:image/svg+xml;charset=utf-8;base64,)](https://opensource.org/licenses/BSD-3-Clause)

[![Dependencies](data:image/svg+xml;charset=utf-8;base64,)](https://www.libsdl.org/) [![Dependencies](data:image/svg+xml;charset=utf-8;base64,)](https://www.libsdl.org/tmp/SDL_ttf/) [![Dependencies](data:image/svg+xml;charset=utf-8;base64,)](https://www.libsdl.org/tmp/SDL_image/) [![Dependencies](data:image/svg+xml;charset=utf-8;base64,)](http://www.ferzkopp.net/wordpress/2016/01/02/sdl_gfx-sdl2_gfx/)

# Введение

Целью данной работы стало создание функциональной и простой в использовании GUI-библиотеки на базе низкоуровневой графической библиотеке SDL второй версии. Так же, вместе с основной библиотекой, были использованы библиотеки SDL\_ttf, SDL\_image и SDL\_gfx.

В основу данной работы были положены некоторые принципы фреймворка React для создания клиентской части веб-сайтов.

# Техническое задание

Разработать GUI-библиотеку (GUI — графический пользовательский интерфейс) на базе библиотеки SDL2, а так же, дополнительных к ней, библиотеках SDL\_\*. Библиотека должна предоставлять простой путь создания окон с возможностью наследования для создания собственных классов окон. Библиотека должна предоставлять базовый набор GUI компонентов (кнопка, флажок). Библиотека должна быть построена в объектно-ориентированной парадигме.

# Описание библиотеки

Библиотека написана на языке С++ с использованием графической библиотеки SDL2. В качестве IDE была использована Visual Studio 2019. В реализации библиотеки не были использованы специфичные возможности операционной системы Windows, что означает, что данная реализация является кроссплатформенной и может быть запущена на любой платформе поддерживаемой библиотекой SDL.

Для удобства разработки вся библиотека была распределена по отдельным папкам. Так вся библиотека расположена в папке kit со следующей иерархией:

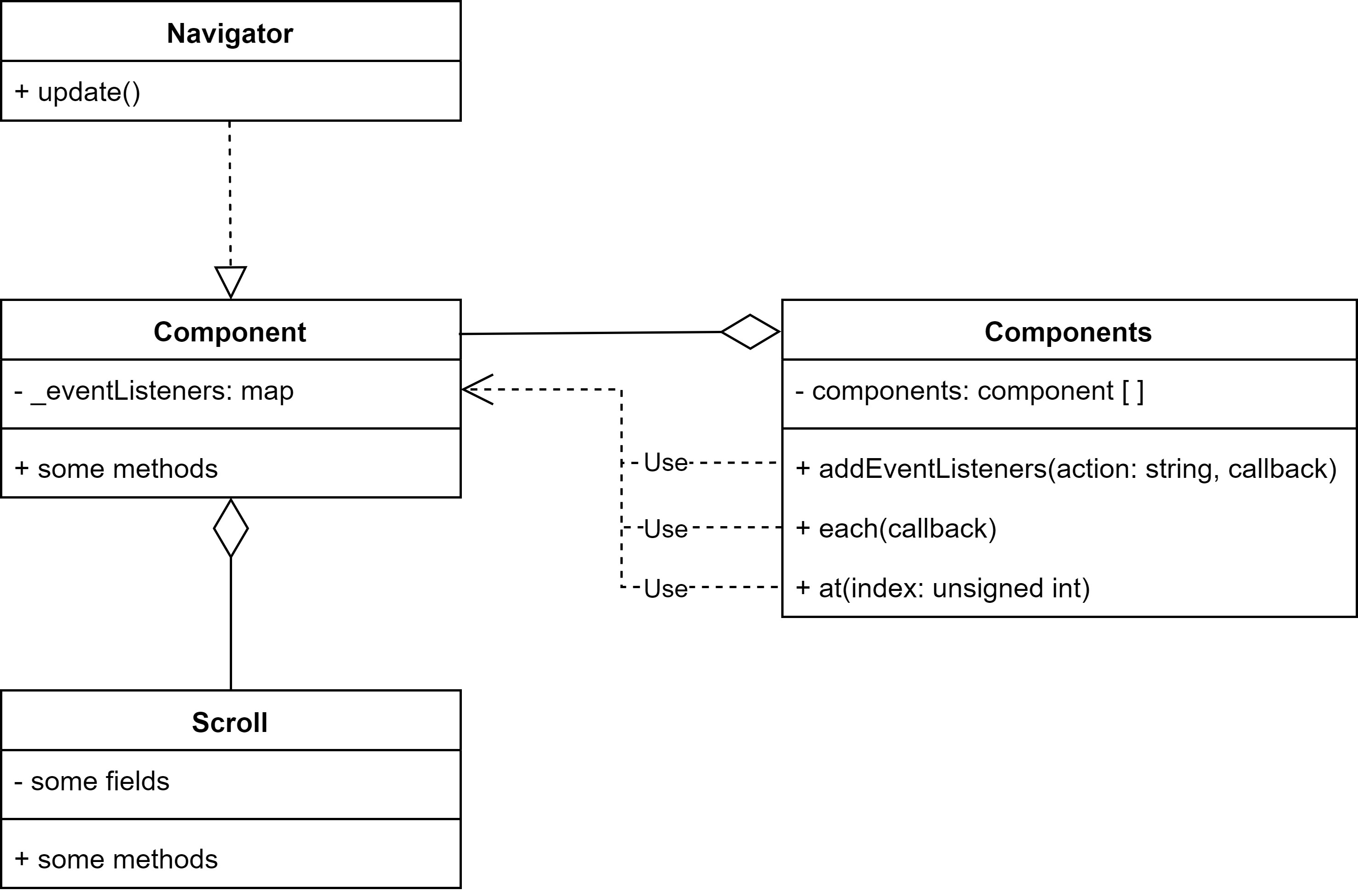
---  
 - component  
 - components  
 - components.h  
 - components.cpp   
 - navigator  
 - navigator.h  
 - navigator.cpp   
 - scroll  
 - scroll.h  
 - scroll.cpp   
 - component.h  
 - component.cpp  
 - component-header.h  
 - event  
 - event.h  
 - tools  
 - css  
 - color  
 - color.h  
 - color.cpp   
 - utils  
 - css\_utils.cpp  
 - utils.h  
 - css.h  
 - css.cpp  
 - css\_block.h  
 - css\_block.cpp  
 - css\_block\_state.h  
 - css\_block\_state.cpp  
 - css-attributes.h  
 - css-attributes.cpp  
 - css-parse.h  
 - css-parse.cpp  
 - font  
 - font.h  
 - font.cpp  
 - image  
 - image.h  
 - image.cpp  
 - point   
 - simple-point  
 - simple-point.h  
 - extended-point  
 - extended-point.h  
 - extended-point.cpp  
 - rect  
 - simple-rect  
 - simple-rect.h  
 - extended-rect  
 - extended-rect.h  
 - extended-rect.cpp  
 - sdl\_gfx  
 - SDL2\_gfxPromitives.c  
 - SDL2\_rotozoom.cpp  
 - size  
 - simple-size  
 - simple-size.h  
 - extended-size.cpp  
 - extended-size  
 - extended-size.h  
 - extended-size.cpp  
 - text  
 - text.h  
 - text.cpp  
 - text-line.h  
 - text-line.cpp  
 - utils  
 - utils.h  
 - utils.cpp  
  
 - window  
 - window.h  
 - window.cpp  
  
 - kit.h  
 - kit-main.h  
 - kit-main.cpp  
 - kit-enter-point

## Описание некоторых составных блоков

### Класс Component и его приложения

Класс Component предоставляет универсальный строительный блок интерфейса. На базе данного класса можно построить любой необходимый элемент интерфейса. Пример создания приведен в главе N. В дополнение к нему, имеются класс Components, который является оберткой над контейнером объектов класса Component для удобной работы с выборкой элементов, класс Navigator представляющий из себя класс-наследник для Component и использующийся в окне, как главный компонент, а также класс Scroll реализующий в себе логику работы скроллинга класса Component.

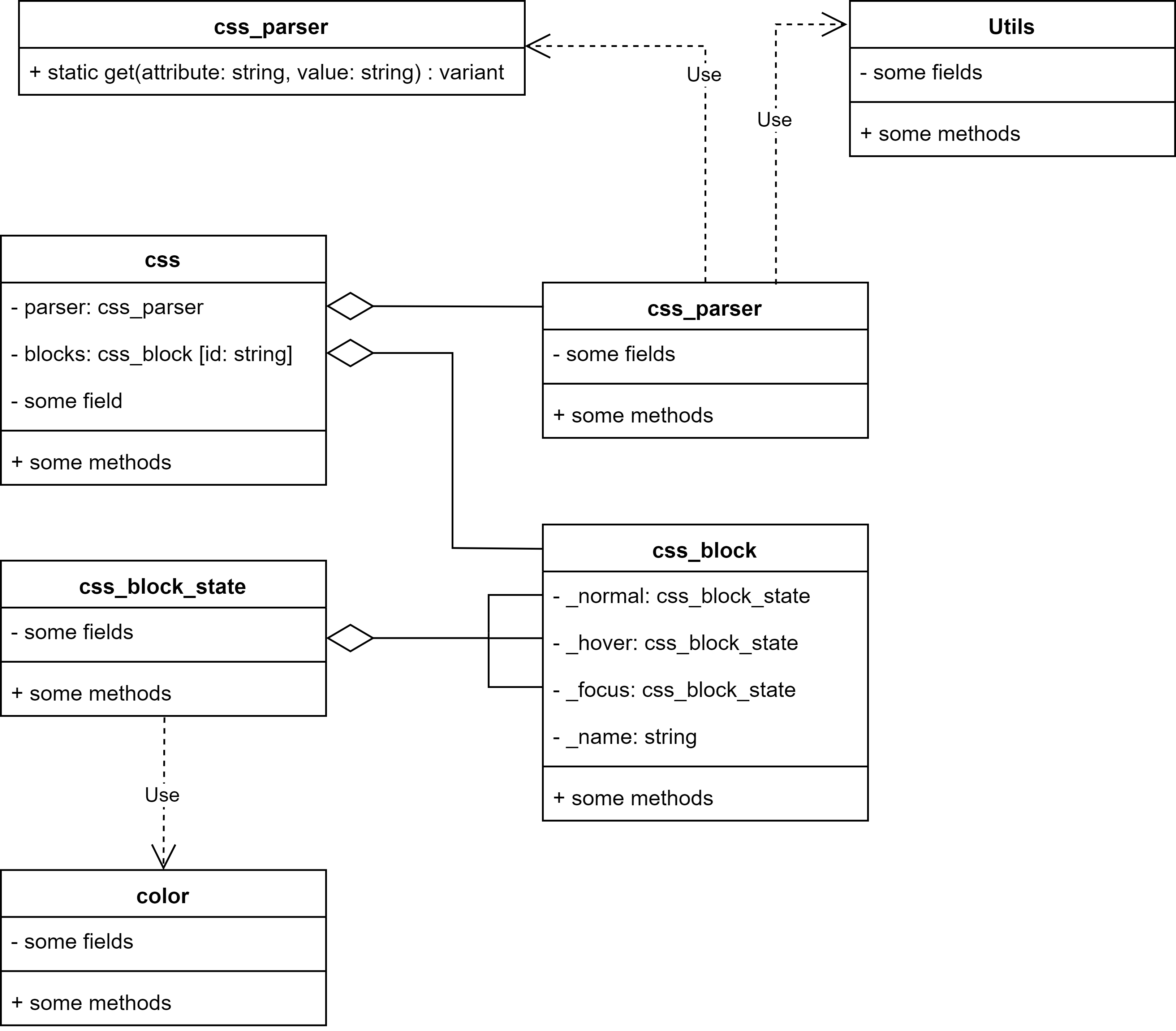
Диаграмму взаимодействий данных классов можно представить следующим образом:



### Класс css и его приложения

Класс css реализует логику хранения и обработки стилей для каждого из окон. Класс css включает в себя класс css\_parser реализующий логику разбора css файлов со стилями. Так же включает в себя ассоциативную коллекцию объектов класса css\_block реализующий логику хранения стилей для каждого из блоков в окне. Класс css\_block хранит в себе три возможных состояния, такие как нормальное состояние, состояния при наведении курсора мыши и состояния нажатия на элемент. Эти состояния описываются классом css\_block\_state. В дополнение для этих классов есть еще класс css\_attribute реализующий определение и возврат нужного типа для каждого значения по его атрибуту.

Диаграмму их взаимодействия можно представить следующим образом:



### Класс Window

Класс Window реализует логику создания окон, а так же их наполнения компонентами интерфейса. Класс является базовым, от него можно унаследоваться для создания более комплексного класса окна, или для создания собственных окон с настраиваемым макетом. Подробнее о создании собственных классов Окна на базе класса Window в главе N.

### Дополнительный инструментарий

Для реализации тех или иных методов были созданы следующие вспомогательные классы:

1. Класс Font — реализует логику работы со шрифтами;
2. Класс Image — реализует логику работы с картинками;
3. Класс Point — реализует хранение точки в программе;
4. Класс Size — реализует хранение размеров в программе;
5. Класс Rect — реализует хранение прямоугольника в программе;
6. Класс Text — реализует логику работы с текстом;
7. Класс Utils — реализует дополнительные функции.

# Использование библиотеки

## Начало работы

Для использования библиотеки необходимо подключить заголовочный файл kit.h.

#include "kit/kit.h"

Вся библиотека расположена в пространстве имен Kit, чтобы каждый раз не писать Kit::\* можно прописать следующую строку

using namespace Kit;

Важно, функция main должна принимать две переменных: int argc, char\*\* argv.

Для удобства, после подключения библиотеки, доступ к объекту библиотеки сразу возможен через короткое имя $.

Для запуска библиотеки необходимо вызвать у $ функцию-член run сразу после оператора return.

#include "kit/kit.h"  
using namespace Lib;  
  
int main(int argc, char\*\* argv)  
{  
 return $.run();  
}

Рассмотрим базовый функционал.

Метод (здесь и далее будет употреблен термин “метод”, как более короткий и понятный):

Window\* addWindow(Window\* window);

добавляет новое окно в текущее приложение.

Класс Window является базовым классом для любых окон приложения. Имеет единственный конструктор:

Window(string title, SimpleRect size);

где первый параметр это заголовок окна, а второй его размеры и положение.

Рассмотрим пример добавление окна:

#include "kit/kit.h"  
using namespace Lib;  
  
int main(int argc, char\*\* argv)  
{  
 $.addWindow(new Window("new window", { 100, 100, 200, 200 }));  
 return $.run();  
}

Данная программа выведет пустое окно размерами 200 на 200 пикселей с координатами 100, 100. Базовый класс окна не имеет никаких компонентов и нужен исключительно для наследования.

Рассмотрим создание нового окна на основе данного класса.

Создадим папку MyWindow рядом с папкой kit. И создадим MyWindow.h.

Для наследования необходимо подключить заголовочный файл базового класса окна:

#include "../kit/window/window.h"

Далее создаем класс MyWindow и наследуем его от Window перегружаем конструктор и добавляем один метод setup, который вызываем в конструкторе. Здесь, пока класс небольшой, напишем реализацию в заголовочном файле.

#pragma once  
  
#include "../kit/window/window.h"  
using namespace Lib;  
  
class MyWindow : public Window  
{  
public:  
 MyWindow(string title, SimpleRect size)  
 : Window(title, size)   
 {  
 setup();  
 };  
  
public:  
 void setup()  
 {  
  
 }  
  
};

Подключим заголовочный файл класса в main.cpp и создадим экземпляр. Так как класс MyWindow — это дочерний класс для Window, то использование указателя на MyWindow разрешено.

#include "kit/kit.h"  
  
#include "MyWindow/MyWindow.h"  
using namespace Lib;  
  
int main(int argc, char\*\* argv)  
{  
 $.addWindow(new MyWindow("new window", { 100, 100, 200, 200 }));  
 return $.run();  
}

Пока что, если запустить программу, ничего не поменялось, экран все еще пуст. Перейдем к добавлению компонентов в интерфейс.

## Добавление компонентов в интерфейс

Любые компоненты в окне строятся на базе класса Component. Возможностей одного объекта класса вполне хватит для создания простейших элементов интерфейса, таких как, кнопка или надпись, но для создания более комплексных компонентов таких как, например, флажок с надписью уже нужно использование нескольких экземпляров класса Component, специально для таких случаев создаются собственные компоненты на базе Component, но об это чуть позже.

Для того, чтобы добавить новый компонент в окно используется метод append у специального компонента Navigator по имени navigator или по короткому имени $$.

Добавим один компонент в только что созданное окно MyWindow. В классе MyWindow был создан метод setup именно в нем добавляются все компоненты окна.

Для краткости рассмотрим только сам метод

void setup()  
{  
 $$->append(new Component("#comp-id", { 10, 10, 50, 50 }, ".class1 .class2");  
}

Функция append имеет следующий прототип:

Component\* append(Component\* component);

Класс Component имеет два конструктора. Рассмотрим первый.

Component(string id, Rect size, string classes);  
Component(string id, Rect size, string classes, vector<Component\*> childrens);

Первым параметром конструктор принимает строку-идентификатор компонента, по которому его в дальнейшем можно будет найти в окне.

**Важно!** В окне не может быть двух компонентов с одинаковым идентификатором.

Вторым параметром идут размеры компонента. Размеры можно указывать, как только числами, так и строками вида 20px или 20%, при этом размер в процентах будет рассчитываться относительно родительского. Так же поддерживаются записи вида x + y или x - y, например 100% - 20px или 50% + 23px.

Третьим параметром идет строка с набором классовых идентификаторов через пробел. Данные классовые идентификаторы могут повторяться у разных элементов. Они используются для стилизации компонентов.

Данная библиотека использует для стилизации небольшую часть языка стилей CSS. Больше о стилизации и поддержки CSS далее.

### Настройка стилей компонента

Для стилизации компонентов библиотека использует язык CSS. Любое окно может подключать файл CSS с помощью функции include

void include(string path);

Добавим компонент:

$$->append(new Component("#button", { 50, 50, 75, 25 }, ".button");

Создадим папку css в папке MyWindow и создадим в ней файл style.css.

Подключим данный файл в наше окно.

void setup()  
{  
 include("css/style.css");  
  
 $$->append(new Component("#button", { 50, 50, 75, 25 }, ".button");  
}

и пропишем в style.css следующие:

.button  
{  
 background-color: #263238;  
 border-color: #0F1518;  
}

теперь наш компонент будет иметь цвета фона и обводки такие, какие были заданны в файле style.css.

### Доступ к компонентам по их id

Для доступа к добавленным компонентам в окне используется функция getElementById, где единственным параметром передается идентификатор компонента:

Component\* getElementById(string id);

### Добавление текста в компонент

Класс Component имеет метод setText для установки текста. Добавим текст недавно созданному компоненту:

void setup()  
{  
 include("css/style.css");  
  
 $$->append(new Component("#button", { 50, 50, 75, 25 }, ".button");  
  
 Window::getElementById("#button")->setText("ok");  
}

Для стилизации текста можно использовать следующие свойства:

color: цвет текста (HEX полная запись из шести 16-ных чисел и решетка в начале);  
font-size: размер текста (number + px);  
line-height: междустрочный интервал (double);  
text-align: выравнивание текста по горизонтале (left center right);  
vertical-align: выравнивание текста по вертикале (top center bottom);  
margin-top: сдвиг текста сверху (number + px);  
margin-bottom: сдвиг текста снизу (number + px);  
margin-left: сдвиг текста слева (number + px);  
margin-right: сдвиг текста справа (number + px);

Например, добавим нашему компоненту следующие стили, чтобы текст не сливался с фоном:

.button  
{  
 background-color: #263238;  
 border-color: #0F1518;  
 color: #ffffff;  
  
 font-size: 12px;  
 line-height: 1.2;  
  
 text-align: center;  
 vertical-align: center;  
}

### Расширенная настройка стилей компонента

Библиотека позволяет использовать 2 псевдокласса CSS такие как hover и active. Псевдоклассы в CSS задаются следующим образом:

.name-class:pseudoclass-name  
{  
  
}

Псевдокласс hover задает стили компонента, если на него наведен курсор мыши. А active, когда на объект только была нажата кнопка мыши, но еще не была отпущена.

Добавим нашему компоненту стили при наведении, для этого добавим в файл стилей следующее:

.button:hover  
{  
 background-color: #0D1012;  
}

Это все, что нужно, чтобы при наведении кнопка меняла свой цвет. При этом можно прописывать любые доступные в данной библиотеке свойства, так же как и при задании базовых стилей элемента.

### Добавление прослушивателей для событий

Любые компоненты могут отслеживать следующие 7 событий:

1. Нажатие кнопки мыши (onmousedown);
2. Отпускание кнопки мыши (onmouseup);
3. Движение курсора мыши по элементу (mousemotion);
4. Попадание курсора мыши на элемент (onmouseover);
5. Выход курсора мыши из элемента (onmouseout);
6. Наведение на элемент курсора (hover);
7. Клик по элементу (click).

Для добавления прослушивателя используется метод addEventListener, где первым аргументом идет имя события, которое мы прослушиваем, а вторым функция, которая будет вызвана при возникновении данного события:

void addEventListener(string name\_event, function<void(Component\* sender, Event\* e) callback>);

Добавим прослушиватель события клика на наш элемент, используя лямбда-функцию для более краткой записи:

void setup()  
{  
 include("css/style.css");  
  
 $$->append(new Component("#button", { 50, 50, 75, 25 }, ".button")->setText("text");  
  
  
 Window::getElementById("#button")->addEventListener("click",   
 [](Component\* sender, Event\* e)  
 {  
 std::cout << "Component clicked" << std::endl;  
 });  
}

Для вывода также подключим библиотеку iostream.

Теперь при клике на компонент, в консоли будут появляться сообщения о том, что компонент был нажат. Получилась простая кнопка в несколько строк.

Но, это выглядит очень громоздко для всего-лишь одной кнопки, далее будет рассмотрено создание своих компонентов на базе класса Component.

### Удаление прослушивателя для события

Так как можно установить прослушиватель, его можно и удалить с помощью функции removeEventListener:

void removeEventListener(string action);

### Немедленный вызов установленного прослушивателя

Может случиться такая ситуация, что необходимо вызвать метод привязанный к какому-либо событию, для этого используется метод callEventListener, где вторым параметром передается событие:

void callEventListener(string action, Event\* e);

## Создание собственных компонентов на базе класса Component

Перейдем к созданию собственных компонентов.

Создадим папку Button рядом с папкой MyWindow и добавим Button.h.

Для создания собственных компонентов подключим заголовочный файл класса Component. Создадим класс и унаследуем его от Component, перегрузим конструктор и добавим функцию setup, которую нужно вызвать в конструкторе.

В конструктор сразу добавим поле для текста кнопки и запишем его в поле text.

#pragma once  
#include "../kit/component/component.h"  
  
using namespace Lib;  
  
class Button : public Component  
{  
private:  
 string text;  
  
public:  
 Button(string id, Rect size, string classes, string text)  
 : Component(id, size, classes)  
 {  
 this->text = text;  
 setup();  
 }  
   
public:  
 void setup()  
 {  
  
 }  
};

Метод setup используется для настройки компонента, в нем можно вызывать все методы класса Component. В предыдущем примере мы использовали метод setText для установки текста компоненту. Вызовем этот метод в setup.

void setup()  
{  
 setText(this->text);  
}

Таким образом мы инкапсулировали логику базового класса Component в обертке Button.

Подключим данный компонент в окно и добавим его:

#pragma once  
  
#include "../kit/window/window.h"  
using namespace Lib;  
  
class MyWindow : public Window  
{  
public:  
 MyWindow(string title, SimpleRect size)  
 : Window(title, size)   
 {  
 setup();  
 };  
  
public:  
 void setup()  
 {  
 include("css/style.css");  
  
 $$->append(new Button("#button", { 50, 50, 75, 25 }, ".button", "Ok");  
 }  
  
};

Это все та же кнопка, но теперь не нужно вызывать метод setText для установки текста, так как компонент сам знает, что его надо установить.

Таким образом можно создать любой сложный компонент интерфейса.

## Подключение стилей в компоненте

До этого все стили подключались глобально для всего окна, но каждый компонент может также содержать в себе необходимые стили, тем самым, этот компонент будет полностью обособлен от окна. Для добавления стилей для компонента используется функция include:

void include(string path);

Например вынесем стили кнопок в стили отдельного компонента. Создадим в папке Button папку css и добавим в нее файл button.css и пропишем в нем стили из style.css:

.button:hover  
{  
 background-color: #0D1012;  
}  
  
.button  
{  
 background-color: #263238;  
 border-color: #0F1518;  
 color: #ffffff;  
  
 font-size: 12px;  
 line-height: 1.2;  
  
 text-align: center;  
 vertical-align: center;  
}

а затем подключим его:

void setup()  
{  
 include("css/button.css");  
  
 setText(this->text);  
}

И удалим стили кнопки из style.css.

Теперь компонент Button полностью обособлен. При его добавлении стили подключаться автоматически.

## Вложенность компонентов

Любой компонент может включать в себя множество других компонентов. Для добавления компонента используется метод append.

Component\* append(Component\* component);

Это тот же метод, что и для добавления компонента в окно. Все потому, что Navigator это производный класс от Component, созданный, как компонент кнопка выше.

Таким образом вы можете вкладывать в компоненты другие компоненты множество раз, создавая при этом сложные элементы, такие как, например, списки.

В случае если дочерние компоненты выходят за рамки родительского компонента по высоте, родительскому компоненту добавляется вертикальный скролл.

Данное поведение можно предотвратить добавим для родительского компонента стиль overflow: hidden, тогда все дочерние компоненты будут обрезаться по размерам родительского.

## Дополнительная информация в компоненте

Любой компонент может хранить в себе дополнительную информацию. Для добавления используется метод addUserData:

void addUserData(string key, void\* data);

А для получения информации по ключу — метод userData:

void\* userData(string key);

## Расширенная работа с классовыми идентификаторами

Классовые идентификаторы используются для стилизации компонентов, для работ с ними есть несколько методов. Названия методов говорят сами за себя.

bool hasClass(string className) const;  
Component\* removeClass(string className);  
Component\* addClass(string className);  
Component\* toggleClass(string className);

Добавление/удаление классовых идентификатором может понадобится, например, при создании Checkbox для изменения класса с unchecked на checked.

## Получение компонентов по классовому идентификатору

Как уже было описано для доступа к компоненту используется его идентификатор. Но доступ к элементу можно получить и по классовому идентификатору, но с некоторыми оговорками.

Так как один и тот же классовый идентификатор может быть у нескольких компонентов, то по классовому идентификатору будет получена коллекция компонентов.

Для получения коллекции используется метод getElementsByClassName

Components getElementsByClassName(string className);

Класс Components является оберткой над коллекцией компонентов. Над коллекцией можно производить следующие действия:

1. Установить каждому элементу коллекции одинаковый прослушиватель с помощью метода addEventListener, который полностью повторяет метод класса Component;
2. Вызвать для каждого элемента коллекции некоторую функцию обратного вызова с передачей первым параметром указателя на текущий элемент коллекции с помощью метода each cpp void each(function< void(Component\* sender)> callback);
3. Получить компонент по номеру с помощью метода at или оператора []: cpp Component\* at(size\_t index); Component\* operator[](size\_t index);

## Возможности в стилизации компонентов

Для стилизации компонентов доступны следующие свойства:

1. background-color — цвет фона [hex color (#XXXXXX)];
2. border-color — цвет обводки [hex color (#XXXXXX)];
3. color — цвет текста [hex color (#XXXXXX)];
4. background-image — фоновое изображение [url(путь\_к\_изображению\_без\_кавычек)];
5. background-position-x — сдвиг изображения по X [number + px];
6. background-position-y — сдвиг изображения по Y [number + px];
7. background-size — размер изображения по ширине [number + px];
8. font-size — размер шрифта [number + px];
9. line-height — междустрочный интервал [double];
10. text-align — выравнивание текста по горизонтали [left|center|right];
11. vertical-align — выравнивание текста по вертикали [top|center|bottom];
12. margin-[top|bottom|left|right] — сдвиг текста сверху [number + px];
13. border-radius — радиус закругления обводки [number + px];
14. border-[top|bottom|left|right] — обводка [number + px|hex color (#XXXXXX)|solid];
15. border-[top|bottom|left|right]-size — ширина обводки [number + px];
16. border-[top|bottom|left|right]-color — цвет обводки hex color (#XXXXXX);
17. border-[top|bottom|left|right]-type — тип обводки solid;
18. overflow — показ скролла или сокрытие [hidden|unset].

Для каждых трех состояний.

## Добавление компонентов с большим уровнем вложенности

В самом начале у класса Component был рассмотрен только первый конструктор. Второй конструктор отличается только тем, что последним параметром принимает список дочерних объектов для данного.

Component(string id, Rect size, string classes, vector<Component\*> childrens);

Рассмотрим пример, когда нам нужно добавить компонент, в этот компонент еще один и в него еще один.

Создавать по отдельности компоненты и добавлять их с помощью метода append может стать довольно неприятной ситуацией, да и читаемость такой вложенности будет маленькой. С помощью второго конструктора код превращается в такой:

$$->append(  
 new Component("main", { "45px", "30px", "100% - 65px", "100% - 50px" }, ".main",   
 {  
 new Component("#left-side", { "0px", "30px", "45px", "100% - 50px" }, ".left-side",   
 {  
 new Component("#settings", { "8px", "100% - 50px", "30px", "30px" }, ".settings")  
 })  
 })  
);

В нем прекрасно просматривается вложенность одних компонентов в другие.

# Заключение

Целью данной работы было создание GUI библиотеки на базе низкоуровневой библиотеки SDL2. Данная библиотека включает в себя все необходимое по техническому заданию. Во время работы вылетов не замечено. Все работает так, как и было задумано.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Б.И. Березин. Начальный курс С и С++. – М.:Издательство Диалог-МИФИ, 2005 г. – 248 с.
2. Р. Лафоре. Объектно-ориентированное программирование в С++. 4-е издание. – Спб.: Издательство ПИТЕР, 2004 г. – 902 с.
3. Б. Страуструп. Язык программирования С++. Специальное издание. Пер. с англ. – М.: Издательство Бином, 2011 г. – 1136 с.
4. Лафоре, Р. Объектно-ориентированное программирование в C++: Пер. с англ./ Р. Лафоре; Пер. А. Кузнецов, Пер. М. Назаров, Пер. В. Шрага. - 4-е изд. - СПб.: Питер, 2003. - 923 с.
5. Официальный сайт графической библиотеки SDL [Электронный ресурс] 2019. URL: https://www.libsdl.org (Дата обращения 13.12.2019)