Лабораторна робота №4

СТАТИЧНІ МЕТОДИ,ПЕРЕВАНТАЖЕННЯ ОПЕРАТОРІВ

ТА МЕТОДІВ

***Мета роботи***: Навчитись доречно використовувати статичні методи, а також використовувати перевантаження методів та операторів.

**1 ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

Додати програмі асинхронність. Визначити уView2 статичний метод OnTimerAction().Перевантажити оператори > =для Data1,= для Data2 .

**2 ОПИС ПРОГРАМИ**

**2.1 Ієрархія та структура класів**

**class button –** клас, який містить атрибути **data\_field1 data\_field2.**

**Class View –** клас, який містить метод, що виводить значення полів **data\_field1 data\_field2**.

**class View2** – клас, який мітить методи info() ,Square() та атрибути **name, surname, num**.

**class button2 -** клас, який характеризується атрибутами матеріал та тип кришки

**class BaseView -** клас, який виводить на екран **data\_field1 data\_field2.**

**class View3 -** клас, який виводить на екран об’єкт класу **button2**

**2.2 Опис програми**

На рисунку 2.1 наведена структура розробленої програми

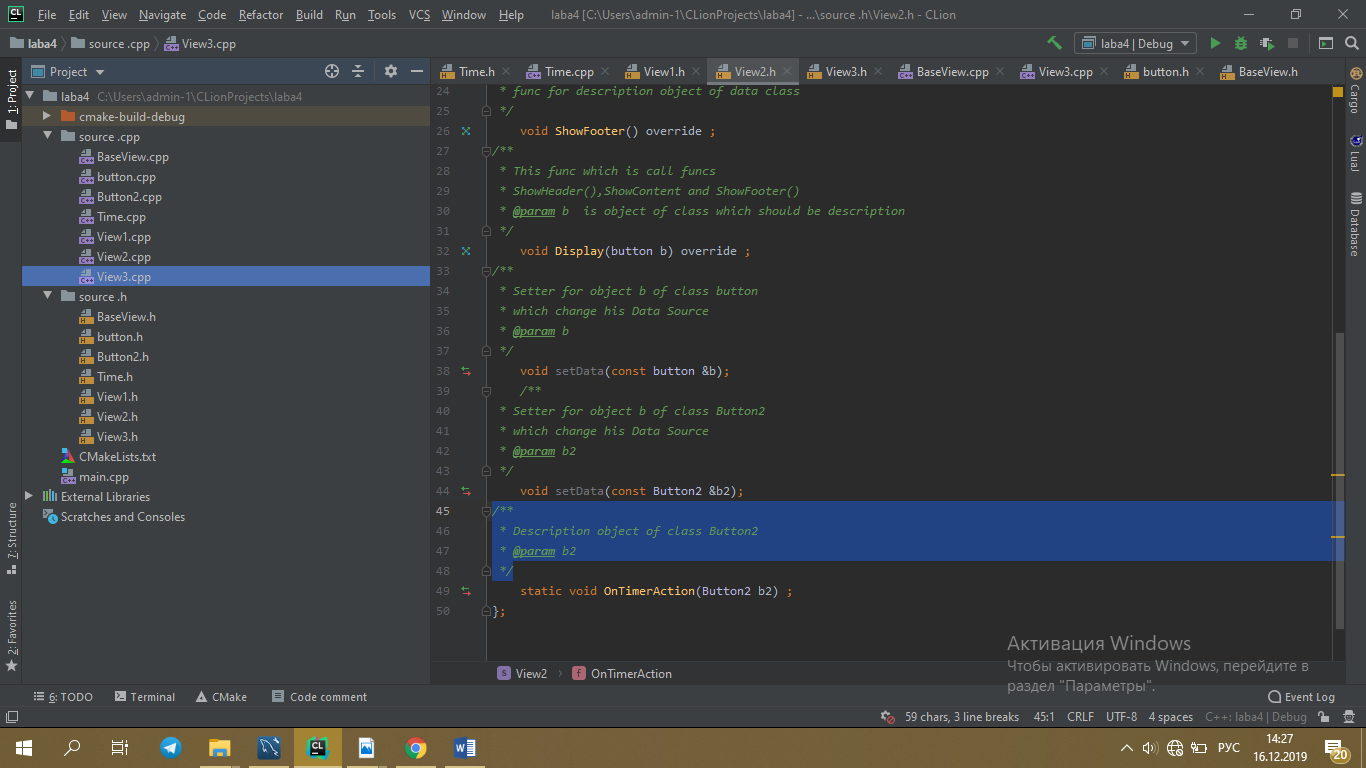


Рис 2.1 – структура програми

**2.3 Важливі фрагменти програми**

2.3.1 Файл*main.cpp*

/\*\*  
 \* @autor Гущин Данил  
 \* @Lava 4.0  
 \* @since 1.1  
 \*  
 \*/  
#include <iostream>  
#include "source .cpp/button.cpp"  
#include "source .cpp/Button2.cpp"  
#include "source .h/button.h"  
#include "source .h/Button2.h"  
#include "source .cpp/View1.cpp"  
#include "source .cpp/View2.cpp"  
#include "source .cpp/View3.cpp"  
#include "source .cpp/BaseView.cpp"  
#include "source .h/BaseView.h"  
#include "source .cpp/Time.cpp"  
#include "source .h/Time.h"  
#include <chrono>  
  
using namespace std;  
/\*\*  
 \* the main function in the program  
 \* @return 0  
 \*/  
int main() {  
Time t;  
 View2 view2;  
 Button2 button2;  
 button2.setSkan\_code(123);  
 button2.setLabel("ESC");  
 button2.setButtonState(*OFF*);  
 button2.setButtonShape(*Oval*);  
  
  
 t.add(std::chrono::milliseconds(1000),view2,button2,true);  
 t.add(std::chrono::milliseconds(5000),view2,button2);  
 t.add(std::chrono::milliseconds(10000),view2,button2,false);  
 t.add(std::chrono::milliseconds(10000),view2,button2,true);  
  
}

2.3.1 Файл*button.cpp*

//  
// Created by admin-1 on 06.09.2019.  
//  
#pragma once  
#include "../source .h/button.h"  
#include <iostream>  
/\*\*  
 \* Setter  
 \* @param b Value for initialization field bt  
 \*/  
 void button::setButtonState(ButtonState b) {  
 bt = b;  
 }  
  
  
  
/\*\*  
 \*  
 \* @return info when button is round pressed true  
 \* or false if button is don't pressed  
 \*/  
bool button::IsRoundPressed() {  
 return getButtonState() == 0;  
}  
/\*\*  
 \* Setter  
 \* @param b Value for initialization field bs  
 \*/  
void button::setButtonShape(ButtonShape b) {  
bs=b;  
}  
/\*\*  
 \* Getter  
 \* @return bs  
 \*/  
ButtonShape button::getButtonShape() const {  
 return this->bs;  
}  
/\*\*  
 \* show work button or doesn't work  
 \*/  
void button::ButtonOperation() {  
 if(IsRoundPressed()){  
 cout<<"System block works";  
 }  
 else  
 {  
 cout<<"System block doesn't work";  
 }  
  
}  
/\*\*  
 \* Getter  
 \* @return bt  
 \*/  
ButtonState button::getButtonState() const {  
 return bt;  
}

2.3.1 Файл*button.h*

#pragma once  
#include <string>  
  
using namespace std;  
/\*\*  
 \* Created enum for class <Data>  
 \* which description state of the button  
 \*/  
enum ButtonState  
{  
 *ON*,  
 *OFF*};  
/\*\*  
 \* Created enum for class <Data>  
 \* which description shape of the button  
 \*/  
enum ButtonShape  
 {  
*Oval*,  
*Rectangular* };  
/\*\*  
 \*Class button(<Data>)  
 \* the main class data  
 \* which have fields shape and state of button  
 \* and his setters and getters  
 \*/  
 class button{  
 private:  
 ButtonState bt;  
 ButtonShape bs;  
 int sa;  
  
  
 public:  
  
 /\*\*  
 \* Setter  
 \* @param b Value for initialization field bt  
 \*/  
 void setButtonState(ButtonState b);  
 /\*\*  
 \* Setter  
 \* @param b Value for initialization field bt  
 \*/  
 void setButtonShape(ButtonShape b);  
 /\*\*  
 \* Getter  
 \* @return bs  
 \*/  
 ButtonShape getButtonShape() const;  
 /\*\*  
 \* Getter  
 \* @return bt  
 \*/  
 ButtonState getButtonState() const;  
/\*\*  
 \*  
 \* @return info when button is round pressed true  
 \* or false if button is don't pressed  
 \*/  
 bool IsRoundPressed();  
/\*\*  
 \* show work button or doesn't work  
 \*/  
 virtual void ButtonOperation();  
  
  
};

2.3.2 Файл*view3.cpp*

//  
// Created by admin-1 on 07.09.2019.  
//  
#pragma once  
#include "../source .h/View3.h"  
/\*\*  
 \* Func for show info about object of class Button2(Data)  
 \* @param b2 object of class Button2(Data2)  
 \*/  
void View3::InfoButton2(Button2 b2) {  
cout<<"ButtonState() = "<<b2.getButtonState()<<endl;  
cout<<"ButtonShape = "<<b2.getButtonShape()<<endl;  
cout<<"Label = "<<b2.getLabel()<<endl;  
cout<<"skan\_code = "<<b2.getSkan\_code()<<endl;  
b2.ButtonOperation();  
}

2.3.3 Файл*view3.h*

//  
// Created by admin-1 on 07.09.2019.  
//  
#pragma once  
  
  
/\*\*  
 \* Class which show info about object  
 \* of class Button2(Data2)  
 \*/  
class View3 : public BaseView {  
public:  
 /\*\*  
 \* Func for show info about object of class Button2(Data)  
 \* @param b2 object of class Button2(Data2)  
 \*/  
 void InfoButton2(Button2 b2);  
};

2.3.4 Файл*View2.cpp*

//  
// Created by admin-1 on 07.09.2019.  
//  
#pragma once  
#include "../source .h/View2.h"  
#include <iostream>  
#include "../source .h/button.h"  
#include "../source .cpp/button.cpp"  
/\*\*  
 \* Setter for object b of class button  
 \* which change his Data Source  
 \* @param b  
 \*/  
void View2::setData(const button &b)  
{  
 c=b;  
}  
/\*\*  
 \* func for description object of data class  
 \*/  
void View2::ShowContent(button b ) {  
  
 int len = 31;  
 int len\_str1 = 25 + strlen("Kit-27B");  
 int len\_str2 = strlen("Gushin") + strlen("Danil") + strlen("Sergeyevich") + 6;  
 int len\_str3 = strlen("ButtonState = ")+5;  
 int len\_str4 = strlen("ButtonShape = ")+5;  
  
  
 cout << "\n\t\t\t";  
 for (int counter = 0; counter < len; counter++ )  
 cout << "\*";  
  
 cout << "\n\t\t\t\* Laba #2 ";  
 for (int counter = 0; counter < len - 25; counter++ )  
 cout << " ";  
  
 cout << "\*\n\t\t\t\* Completed by: st. gr." << "Kit-27B";  
 for (int counter = 0; counter <= (len - len\_str1); counter++ )  
 cout << " ";  
  
 cout << "\*\n\t\t\t\* " << "Gushin" << " " << "Danil" << " " << "Sergeyevich";  
 for (int counter = 0; counter <= (len - len\_str2); counter++ )  
 cout << " ";  
  
 cout << "\*\n\t\t\t\* " << "ButtonState = " <<b.getButtonState();  
 for (int counter = 0; counter <= (len - len\_str3); counter++ )  
 cout << " ";  
  
 cout << "\*\n\t\t\t\* " << "ButtonShape = " << b.getButtonShape() ;  
 for (int counter = 0; counter <= (len - len\_str4); counter++ )  
 cout << " ";  
  
 cout << "\*\n\t\t\t";  
 for (int counter = 0; counter < len; counter++ )  
 cout << "\*";  
 cout << endl;  
  
  
  
}  
/\*\*  
 \* func for description object of data class  
 \*/  
void View2::ShowHeader() {  
 cout<<"it's full info about button"<<endl;  
  
}  
/\*\*  
 \* func for description object of data class  
 \*/  
void View2::ShowFooter() {  
 cout<<"Version 2.0";  
  
}  
/\*\*  
 \* This func which is call funcs  
 \* ShowHeader(),ShowContent and ShowFooter()  
 \* @param b is object of class which should be description  
 \*/  
void View2::Display(button b) {  
  
 ShowHeader();  
 cout<<endl;  
 ShowContent(b);  
 cout<<endl;  
 ShowFooter();  
 cout<<endl;  
}  
/\*\*  
 \* Description object of class Button2  
 \* @param b2  
 \*/  
 void View2::OnTimerAction(Button2 b2) {  
 cout<<"ButtonState() = "<<b2.getButtonState()<<endl;  
 cout<<"ButtonShape = "<<b2.getButtonShape()<<endl;  
 cout<<"Label = "<<b2.getLabel()<<endl;  
 cout<<"skan\_code = "<<b2.getSkan\_code()<<endl;  
 b2.ButtonOperation();  
}  
  
/\*\*  
\* Setter for object b of class Button2  
\* which change his Data Source  
\* @param b2  
\*/  
void View2::setData(const Button2 &b2) {  
b=b2;  
}

#include <iostream>  
#include "BaseView.h"  
  
using namespace std;  
  
/\*\*  
 \* Class View2 which show more detailed info about  
 \* data class(<Data>) with graphical interface  
 \*/  
class View2: public BaseView  
{  
  
  
public:  
 /\*\*  
 \* func for description object of data class  
 \*/  
 void ShowHeader() override ;  
 /\*\*  
 \* func for description object of data class  
 \*/  
 void ShowContent(button b) override ;  
 /\*\*  
 \* func for description object of data class  
 \*/  
 void ShowFooter() override ;  
/\*\*  
 \* This func which is call funcs  
 \* ShowHeader(),ShowContent and ShowFooter()  
 \* @param b is object of class which should be description  
 \*/  
 void Display(button b) override ;  
/\*\*  
 \* Setter for object b of class button  
 \* which change his Data Source  
 \* @param b  
 \*/  
 void setData(const button &b);  
 /\*\*  
 \* Setter for object b of class Button2  
 \* which change his Data Source  
 \* @param b2  
 \*/  
 void setData(const Button2 &b2);  
/\*\*  
 \* Description object of class Button2  
 \* @param b2  
 \*/  
 static void OnTimerAction(Button2 b2) ;  
};

2.3.6 Файл*button2.cpp*

/  
// Created by admin-1 on 07.09.2019.  
//  
#pragma once  
#include "../source .h/Button2.h"  
/\*\*  
 \* Setter  
 \* @param sc  
 \*/  
void Button2::setSkan\_code(int sc) {  
skan\_code=sc;  
}  
/\*\*  
 \* Setter  
 \* @param l  
 \*/  
void Button2::setLabel(string l) {  
label=l;  
}  
/\*\*  
 \* Getter  
 \* @return label  
 \*/  
string Button2::getLabel() {  
 return label;  
}  
/\*\*  
 \* Getter  
 \* @return skan\_code  
 \*/  
int Button2::getSkan\_code() {  
 return skan\_code;  
}  
/\*\*  
 \* This func about state of button  
 \* work button or doesn't work  
 \*/  
void Button2::ButtonOperation() {  
 if(IsRoundPressed())  
 {  
 cout<<"Monitor Work";  
 }  
 else  
 {  
 cout<<"Monitor doesn't work";  
 }  
}  
/\*\*  
 \*overloaded comparison operator for skan\_code  
 \* @param skan\_code  
 \* @return this->skan\_code==skan\_code(@param )  
 \*/  
bool Button2::operator==(int skan\_code) {  
 {  
 return this->skan\_code==skan\_code;  
 }  
}

2.3.7 Файл*button2.h*

//  
// Created by admin-1 on 07.09.2019.  
//  
#pragma once  
#include "button.h"  
/\*\*  
 \*Class Button2(<Data2>)  
 \* his base class is button  
 \* which have fields shape and state of button  
 \* alse skan\_code and label  
 \* and his setters and getters  
 \*/  
class Button2 :public button{  
  
private:  
 int skan\_code;  
 string label;  
  
public:  
 /\*\*  
 \*overloaded comparison operator for skan\_code  
 \* @param skan\_code  
 \* @return this->skan\_code==skan\_code(@param )  
 \*/  
 bool operator == (int skan\_code);  
  
 /\*\*  
 \* Setter  
 \* @param sc  
 \*/  
 void setSkan\_code(int sc);  
 /\*\*  
 \* Setter  
 \* @param l  
 \*/  
 void setLabel(string l);  
 /\*\*  
 \* Getter  
 \* @return skan\_code  
 \*/  
 int getSkan\_code();  
 /\*\*  
 \* Getter  
 \* @return label  
 \*/  
 string getLabel();  
 /\*\*  
 \* This func about state of button  
 \* work button or doesn't work  
 \*/  
 void ButtonOperation() override ;  
};

2.3.8 Файл*view3.h*

//  
// Created by admin-1 on 07.09.2019.  
//  
#pragma once  
  
  
/\*\*  
 \* Class which show info about object  
 \* of class Button2(Data2)  
 \*/  
class View3 : public BaseView {  
public:  
 /\*\*  
 \* Func for show info about object of class Button2(Data)  
 \* @param b2 object of class Button2(Data2)  
 \*/  
 void InfoButton2(Button2 b2);  
};

2.3.9 Файл*View3.срр*

//  
// Created by admin-1 on 07.09.2019.  
//  
#pragma once  
#include "../source .h/View3.h"  
/\*\*  
 \* Func for show info about object of class Button2(Data)  
 \* @param b2 object of class Button2(Data2)  
 \*/  
void View3::InfoButton2(Button2 b2) {  
cout<<"ButtonState() = "<<b2.getButtonState()<<endl;  
cout<<"ButtonShape = "<<b2.getButtonShape()<<endl;  
cout<<"Label = "<<b2.getLabel()<<endl;  
cout<<"skan\_code = "<<b2.getSkan\_code()<<endl;  
b2.ButtonOperation();  
}

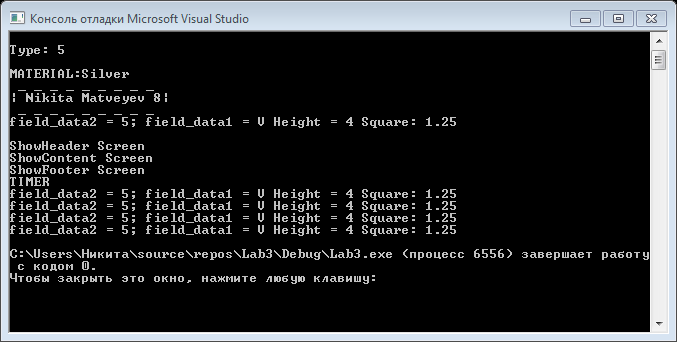
2.3.10 Файл*BaseView.h*

//  
// Created by admin-1 on 07.09.2019.  
//  
#pragma once  
/\*\*  
 \* this is the base class for all classes View  
 \*  
 \*/  
class BaseView {  
public:  
  
 /\*\*  
 \* This function give info about object b  
 \* of class button  
 \* @param b  
 \*/  
 void INFO(button b);  
 button c;  
 Button2 b;  
 /\*\*  
 \* This function give info about object b  
 \* of class button  
 \* @param b  
 \*/  
 void PrintData(const button& b );  
  
/\*\*  
 \* This func which is call funcs  
 \* ShowHeader(),ShowContent and ShowFooter()  
 \* @param b is object of class which should be description  
 \*/  
 virtual void Display(button b) ;  
  
/\*\*  
 \* func for description object  
 \*/  
 virtual void ShowHeader() ;  
 /\*\*  
 \* func for description object  
 \*/  
 virtual void ShowContent(button b) ;  
 /\*\*  
 \* func for description object  
 \*/  
 virtual void ShowFooter();  
 void CheckFuncOfTheButton( button\* px);  
};

2.3.11 Файл*BaseView.cpp*

//  
// Created by admin-1 on 07.09.2019.  
//  
#pragma once  
#include "../source .h/BaseView.h"  
  
  
  
/\*\*  
 \* This func which is call funcs  
 \* ShowHeader(),ShowContent and ShowFooter()  
 \* @param b is object of class which should be description  
 \*/  
void BaseView::Display(button b) {  
 ShowHeader();  
 cout<<endl;  
 ShowContent(b);  
 cout<<endl;  
 ShowFooter();  
 cout<<endl;  
  
}  
/\*\*  
 \* func for description object  
 \*/  
void BaseView::ShowHeader() {  
 cout<<"Info about button and her work";  
}  
/\*\*  
 \* func for description object  
 \*/  
void BaseView::ShowContent(button b) {  
 cout<<"it's content of button in the future when i create first version 1.0 View";  
  
}  
/\*\*  
 \* func for description object  
 \*/  
void BaseView::ShowFooter() {  
 cout<<"Warnings: don't press this button if you installing any programs on your laptop";  
}  
/\*\*  
 \* This function give info about object b  
 \* of class button  
 \* @param b  
 \*/  
void BaseView::INFO(button b) {  
  
 cout <<"ButtonState = "<<b.getButtonState()<<endl;  
  
  
 cout <<"ButtonState = " <<b.getButtonShape()<<endl;  
  
  
}  
void BaseView::CheckFuncOfTheButton(button \*px){  
 px->ButtonOperation();  
  
}  
/\*\*  
\* This function give info about object b  
\* of class button  
\* @param b  
\*/  
void BaseView::PrintData(const button& b)  
{  
  
  
  
  
/\*\*  
 \* Made length for 5 columns of graphical table  
 \*/  
 int len = 31;  
 int len\_str1 = 25 + strlen("Kit-27B");  
 int len\_str2 = strlen("Gushin") + strlen("Danil") + strlen("Sergeyevich") + 6;  
 int len\_str3 = strlen("ButtonState = ")+5;  
 int len\_str4 = strlen("ButtonShape = ")+5;  
  
/\*\*  
 \* This is an algorithm for description info about object  
 \* of class button in graphical interface (table)  
 \*/  
  
  
/\*\*  
 \* the top bord of the table  
 \*/  
 cout << "\n\t\t\t";  
 for (int counter = 0; counter < len; counter++ )  
 cout << "\*";  
/\*\*  
 \* first column  
 \*/  
 cout << "\n\t\t\t\* Laba #2 ";  
 for (int counter = 0; counter < len - 25; counter++ )  
 cout << " ";  
/\*\*  
 \* second column  
 \*/  
 cout << "\*\n\t\t\t\* Completed by: st. gr." << "Kit-27B";  
 for (int counter = 0; counter <= (len - len\_str1); counter++ )  
 cout << " ";  
/\*\*  
 \* third column  
 \*/  
 cout << "\*\n\t\t\t\* " << "Gushin" << " " << "Danil" << " " << "Sergeyevich";  
 for (int counter = 0; counter <= (len - len\_str2); counter++ )  
 cout << " ";  
/\*\*  
 \* fourth column  
 \*/  
 cout << "\*\n\t\t\t\* " << "ButtonState = " <<b.getButtonState();  
 for (int counter = 0; counter <= (len - len\_str3); counter++ )  
 cout << " ";  
/\*\*  
 \* fifth column  
 \*/  
 cout << "\*\n\t\t\t\* " << "ButtonShape = " << b.getButtonShape() ;  
 for (int counter = 0; counter <= (len - len\_str4); counter++ )  
 cout << " ";  
/\*\*  
 \* the foot bord of the table  
 \*/  
 cout << "\*\n\t\t\t";  
 for (int counter = 0; counter < len; counter++ )  
 cout << "\*";  
 cout << endl;  
  
  
  
}

**3 РЕЗУЛЬТАТИ**

****

**ВИСНОВКИ**

Навчились доречно використовувати статичні методи, а також використовувати перевантаження методів та операторів.