Статичні методи,перевантаження операторів та методів

Мета

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

1. Індивідуальне завдання

Необхідно визначити у класі GraphScreen статичний метод OnTimerAction(). Цей метод відображатиме на екрані заданий нащадок Manipulator.
Обрати для Win32-таймера власний інтервал повторних викликів.
Встановити реалізований метод GraphScreen::OnTimerAction() на виклик у таймері.
Таймер повинен спрацювати лише 4 рази. Метод повинен виводити на екран дані про поточний асоційований об’єкт даних.

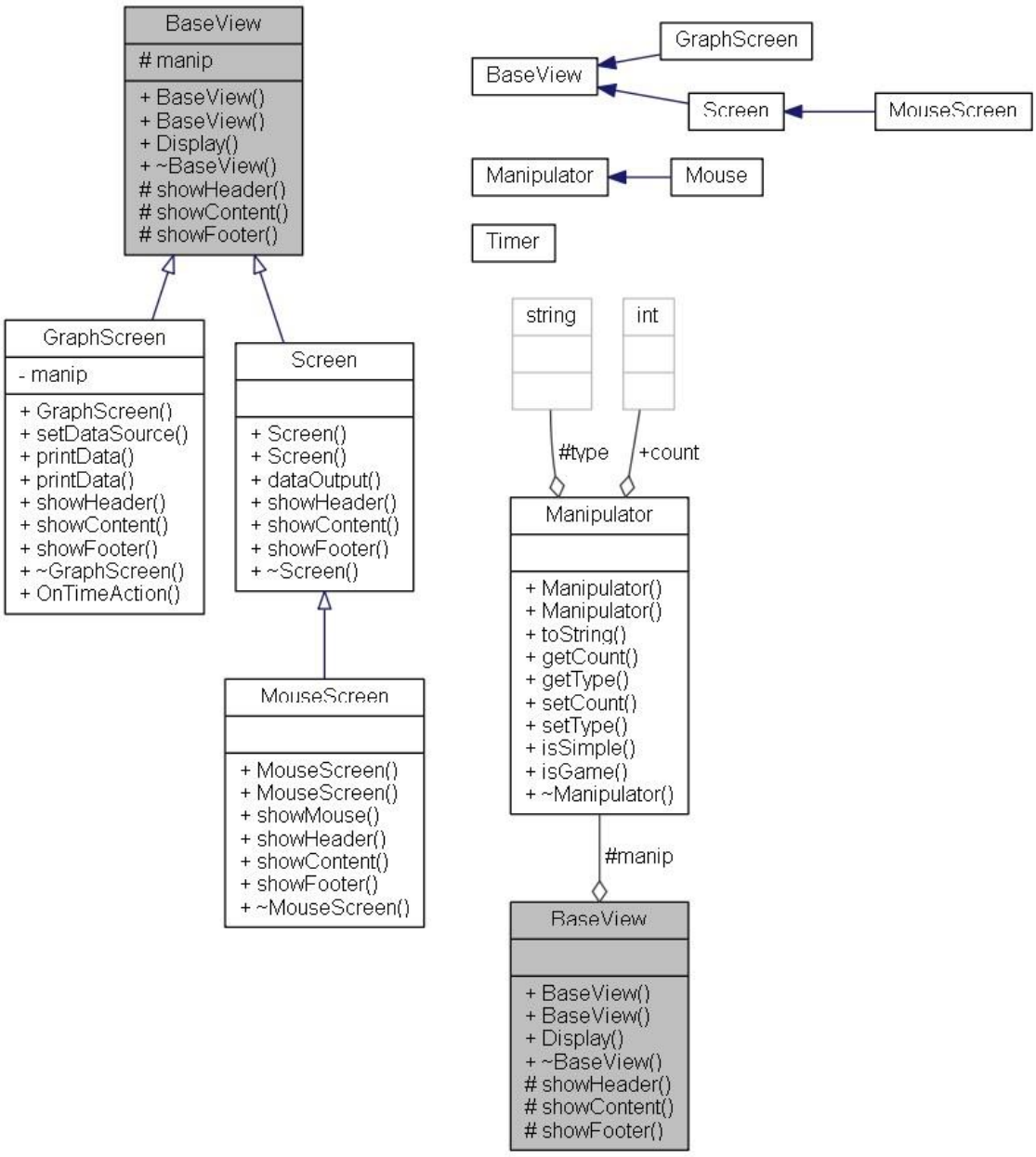
2. Розробка програми

2.1 Засоби ООП

В ході розробки програми були використані так засоби ООП:
- Абсракція – кожен об’єкт описує свою сутність, яка визначається його полями.
- Спадкування - механізм утворення нових класів на основі використання вже існуючих
- Інкапсуляція - поля об'єктів закриті для користувача,натомість ми даємо доступ до даних за допомогою геттерів та сеттерів, так користувач має можливість отримати готові дані, а не обробляти їх, для подпльшого вікористання.
- Поліморфізм - властивість, яка дозволяє одне і те саме ім’я використовувати для вирішення декількох технічно різних задач, тобто основною метою поліморфізму є використання одного імені для задання загальних класу дій.

2.2 Ієрархія та структура класів

На рис 2.1 дивись ієрархію класів.


BaseView
manip
+ BaseView()
+ BaseView()
+ Display()
+ ~BaseView()
# showHeader()
# showContent()
# showFooter()
GraphScreen
BaseView
Screen
MouseScreen
Manipulator
Mouse
Timer
GraphScreen
- manip
+ GraphScreen()
+ setDataSource()
+ printData()
+ printData()
+ showHeader()
+ showContent()
+ showFooter()
+ ~GraphScreen()
+ OnTimeAction()
Screen
+ Screen()
+ Screen()
+ dataOutput()
+ showHeader()
+ showContent()
+ showFooter()
+ ~Screen()
MouseScreen
+ MouseScreen()
+ MouseScreen()
+ showMouse()
+ showHeader()
+ showContent()
+ showFooter()
+ ~MouseScreen()
string
int
#type
+count
Manipulator
+ Manipulator()
+ Manipulator()
+ toString()
+ getCount()
+ getType()
+ setCount()
+ setType()
+ isSimple()
+ isGame()
+ ~Manipulator()
#manip
BaseView
+ BaseView()
+ BaseView()
+ Display()
+ ~BaseView()
# showHeader()
# showContent()
# showFooter()

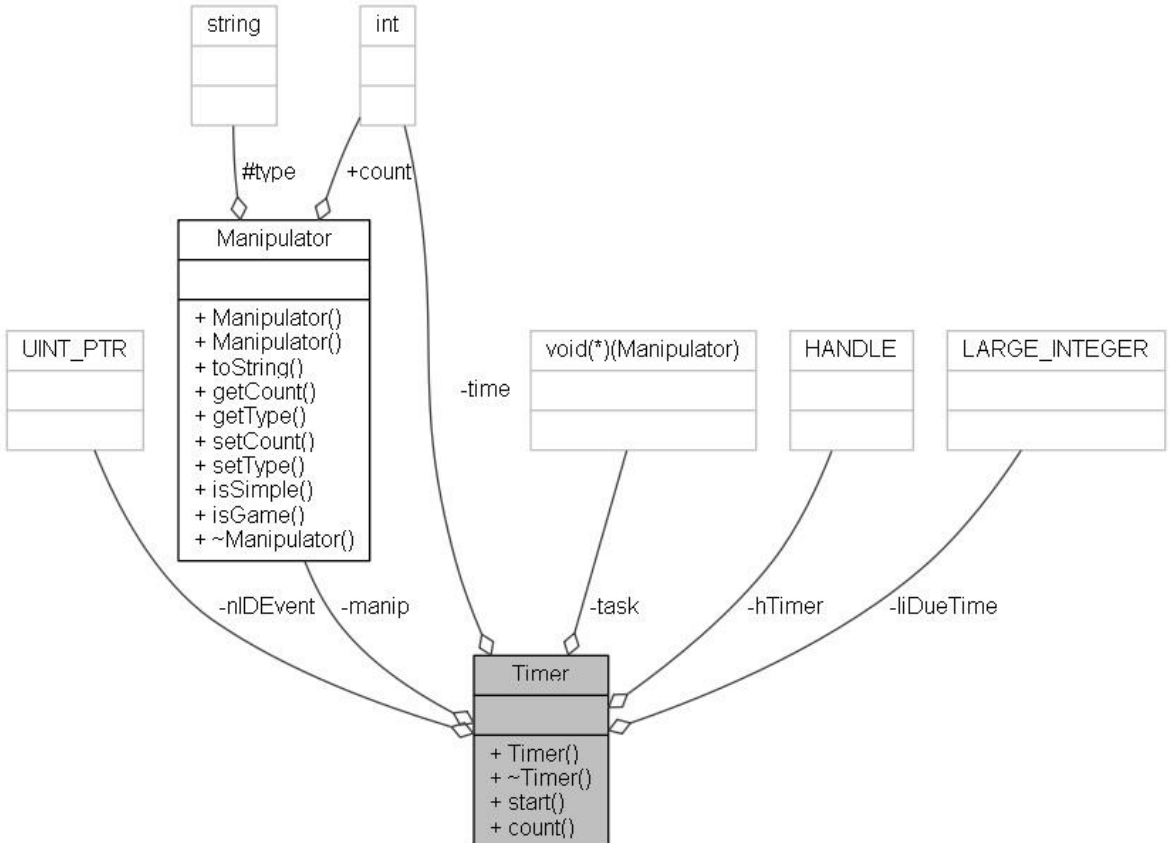

string
int
#type
+count
Manipulator
+ Manipulator()
+ Manipulator()
+ toString()
+ getCount()
+ getType()
+ setCount()
+ setType()
+ isSimple()
+ isGame()
+ ~Manipulator()
UINT_PTR
-time
void(*)(Manipulator)
HANDLE
LARGE_INTEGER
-nIDEvent
-manip
-task
-hTimer
-liDueTime
Timer
+ Timer()
+ ~Timer()
+ start()
+ count()

Рисунок 2.1 – Ієрархія класів

Опис програми
На рис. 2.2 дивись структуру проекту.

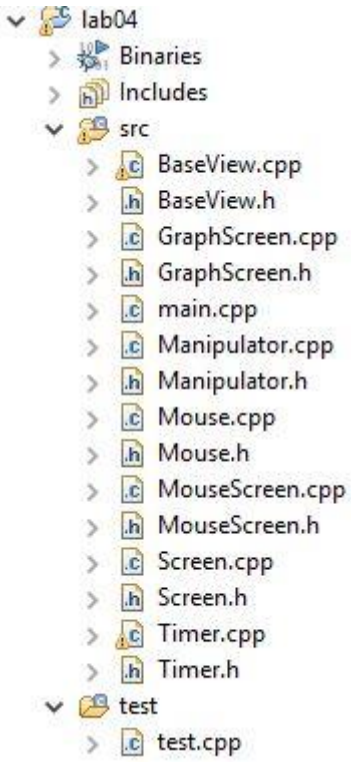


Рисунок 2.2 – Структура проекту

На рис. 2.3 дивись призначення класів.

Класи, структури, об'єднання та інтерфейси з коротким описом.

BaseView	Базовий клас для Manipulator & Mouse
GraphScreen	Клас для відображення псевдографіки
Manipulator	Клас опису маніпулятору
Mouse	Клас опису мишки
MouseScreen	Клас для відображення данних мишки
Screen	Класс для відображення даних
Timer	Клас таймер

Рисунок 2.3 – Призначення класів

Класс Timer описує Win32 timer. Метод GraphScreen::OnTimerAction() відображає 4 рази інформацію про Manipulator.

2.4 Важливі фрагменті програми

У программі слід зауважити увагу на таких моментах:

Клас `Timer.cpp`:

```
pTimer::Timer(void (*task)(Manipulator), Manipulator manip) :
```

```
            task(task), manip(manip) {
      this->time = 3;
      this->liDueTime.QuadPart = -10000000LL;
      this->hTimer = NULL;
}

void Timer::start(){
      hTimer = CreateWaitableTimer(NULL, TRUE, "Time");
            if(hTimer == NULL){
                  printf("Create timer is faled (%d)\n", GetLastError());
            }

            for(int i = 0; i < Timer::time; i++){
                  count();
            }
}

void Timer::count(){
      if(!SetWaitableTimer(hTimer, &liDueTime, 0, NULL, NULL, 0)){
            printf("SetTimer failed (%d)\n", GetLastError());
      }

      if (WaitForSingleObject(hTimer, INFINITE) != WAIT_OBJECT_0)
                  printf("WaitForSingleObject failed (%d)\n",
GetLastError());
            else
                  task(manip);
}
```

OnTimerAction():

```
void GraphScreen::OnTimeAction(Manipulator manip) {
      cout << "В OnTimeAction:";
      cout << "\n Тип девайса: " << manip.getType() << endl;
      cout << "Количество кнопок: " << manip.getCount() << endl;
      cout << "" << endl;
}
```

3. Результати роботи

Результати роботи показано на рис.3.1

Console | Problems | Tasks | Properties

<terminated> (exit value: 0) lab04.exe [C/C++ Application] C:\workspace\lab04\Debug\lab04.exe (01.11.17, 10:53)

```
B OnTimeAction:
 Тип девайса: Mouse
Количество кнопок: 4

B OnTimeAction:
 Тип девайса: Mouse
Количество кнопок: 4

B OnTimeAction:
 Тип девайса: Mouse
Количество кнопок: 4

From Screen
manip1:

Screen constructor
Screen::showHeader()
Количество кнопок: 5
Тип устройства : Joystick

 Screen::showFooter()
manip2:

Screen constructor
Screen::showHeader()
Количество кнопок: 2
Тип устройства : Mouse

 Screen::showFooter()

GraphScreen constructor
From Graph Screen
GraphScreen::showHeader()
```

```
 Тип девайса: Joystick
Количество кнопок: 5


 GraphScreen::showFooter()

Screen constructor
From mouse:
```

Рисунок 3.1 – Результати роботи

Висновки

Отримав основні навики перевантаження операторів, методів та ознайомився зі статичними методами