**Лабораторная работа № 5. “ОБРАБОТКА СОБЫТИЙ”**  
**Цель.** Получить практические навыки разработки объектно-ориентированной программы, управляемой событиями.  
**Основное содержание работы.**  
Написать интерактивную программу, выполняющую команды, вводимые пользователем с клавиатуры.  
**Краткие теоретические сведения.**  
**Объектно-ориентированная программа как программа, управ-**  
**ляемая событиями.**  
При использовании ООП все объекты являются в некотором смысле  
обособленными друг от друга, и возникают определенные трудности в передаче информации от объекта к объекту. В ООП для передачи информации между объектами используется механизм обработки событий.  
События лучше всего представить себе как пакеты информации, которыми обмениваются объекты и которые создаются объектно-ориентированной средой в ответ на те или иные действия пользователя.  
Нажатие на клавишу или манипуляция мышью порождают событие, которое передается по цепочке объектов, пока не найдется объект, знающий,  
как обрабатывать это событие. Для того чтобы событие могло передаваться от объекта к объекту, все объекты программы должны быть объединены  
в группу. Отсюда следует, что прикладная программа должна быть объектом-группой, в которую должны быть включены все объекты, используемые в программе.  
Таким образом, объектно-ориентированная программа – это программа, управляемая событиями. События сами по себе не производят никаких действий в программе, но в ответ на событие могут создаваться новые объекты, модифицироваться или уничтожаться существующие, что и  
приводит к изменению состояния программы. Иными словами все действия по обработке данных реализуются объектами, а события лишь управляют их работой.  
Принцип независимости обработки от процесса создания объектов приводит к появлению двух параллельных процессов в рамках одной программы: процесса создания объектов и процесса обработки данных.  
Это означает, что действия по созданию, например, интерактивных  
элементов программы (окон, меню и пр.) можно осуществлять, не заботясь  
о действиях пользователя, которые будут связаны с ними.

И наоборот, мы можем разрабатывать части программы, ответственные за обработку действий пользователя, не связывая эти части с созданием нужных интерактивных элементов.  
**Событие.**  
Событие с точки зрения языка С++ – это объект, отдельные поля которого характеризуют те или иные свойства передаваемой информации,  
например:  
*struct TEvent*  
*{int what*  
*union{*  
*MouseEventType mouse;*  
*KeyDownEvent keyDown;*  
*MessageEvent message;*  
*};*  
Объект *TEvent* состоит из двух частей. Первая (*what*) задает тип события, определяющий источник данного события. Вторая задает информацию, передаваемую с событием. Для разных типов событий содержание  
информации различно. Поле *what* может принимать следующие значения:  
***evNothing*** – это пустое событие, которое означает, что ничего делать  
не надо. Полю *what* присваивается значение *evNothing*, когда событие обработано каким-либо объектом.  
***evMouse*** – событие от мыши.  
Событие от мыши может иметь, например, такую структуру:  
*struct MouseEventType*  
*{int buttons;*  
*int doubleClick;*  
*TPoint where;*  
*};*  
где *buttons* указывает нажатую клавишу;  
*doubleClick* указывает был ли двойной щелчок;  
*where* указывает координаты мыши.  
***evKeyDown*** – событие от клавиатуры.  
Событие от клавиатуры может иметь, например, такую структуру:  
*struct KeyDownEvent*  
*{union{int keyCode;*  
*union{char charCode;*  
*char scanCode;*  
*};*  
***evMessage*** − событие-сообщение от объекта.  
Для события от объекта (*evMessage*) задаются два параметра :  
*command* – код команды, которую необходимо выполнить при появлении данного события;  
*infoPtr* – передоваемая с событием (сообщение) информация.  
*struct MessageEvent*  
*{int command;*  
*void infoPtr;*  
*};*  
**Методы обработки событий.**  
Следующие методы необходимы для организации обработки событий (названия произвольны).  
**GetEvent** – формирование события;  
**Execute** реализует главный цикл обработки событий. Он постоянно  
получает событие путем вызова GetEvent и обрабатывает их с помощью  
HandleEvent. Этот цикл завершается, когда поступит событие “конец”.  
**HandleEvent** – обработчик событий. Обрабатывает каждое событие  
нужным для него образом. Если объект должен обрабатывать определенное событие (сообщение), то его метод HandleEvent должен распознавать  
это событие и реагировать на него должным образом. Событие может распознаваться, например, по коду команды (поле command).  
**ClearEvent** очищает событие, когда оно обработано, чтобы оно не  
обрабатывалось далее.  
**Обработчик событий (метод HandleEvent).**  
Получив событие (структуру типа TEvent), обработчик событий для  
класса TDerivedClass обрабатывает его по следующей схеме:  
void TDerivedClass::HandleEvent(TEvent& event)  
{ //Вызов обработчика событий базового класса  
TBaseClass::handleEvent( event );  
if( event.what == evCommand ) // Если обработчик событий базового  
класса // событие не обработал  
{  
switch( event.message.command )  
{  
case cmCommand1:  
// Обработка команды cmCommand1  
// Очистка события

СlearEvent( event );  
break;  
case cmCommand2:

// Обработка команды cmCommand2  
СlearEvent( event );  
break;  
…  
case cmCommandN:  
// Обработка команды cmCommandN  
СlearEvent( event );  
break;  
default: // событие не обработано  
break;  
}  
};  
}

Обработчик событий группы вначале обрабатывает команды группы,  
а затем, если событие не обработано, передает его своим элементам, вызывая их обработчики событий.  
void TGroup::HandleEvent(TEvent& event)  
{ if( event.what == evCommand )  
{switch( event.message.command )  
// обработка событий объекта-группы  
default: // событие не группой обработано  
//получить доступ к первому элементу группы  
while((event.what != evNothing)!!( /\* просмотрены не все элементы \*/)  
{  
//вызвать HandleEvent текущего элемента  
//перейти к следующему элементу группы  
}  
break;  
}  
}  
**Метод ClearEvent-очистка события.**  
ClearEvent очищает событие, присваивая полю event.What значение  
evNothing.  
**Главный цикл обработки событий (метод Execute)**  
Главный цикл обработки событий реализуется в методе Execute  
главной группы-объекта “прикладная программа” по следующей схеме:  
int TMyApp::Execute()  
{do{endState=0;

GetEvent(event); //получить событие  
HandleEvent(event); //обработать событие  
if(event.what!=evNothing) //событие осталось не обработано  
EventError(event);  
}  
while(!Valid());  
return endState;  
}  
Метод HandleEvent программы обрабатывает событие “конец работы”, вызывая метод EndExec. EndExec изменяет значение private – переменной EndState. Значение этой переменной проверяет метод–функция  
Valid, возвращающая значение true, если “конец работы”. Такой несколько  
сложный способ завершения работы программы связан с тем, что в активном состоянии могут находиться несколько элементов группы. Тогда метод Valid группы, вызывая методы Valid своих подэлементов, возвратит true, если все они возвратят true. Это гарантирует, что программа завершит свою работу, когда завершат работу все ее элементы.  
Если событие осталось не обработанным, то вызывается метод EventError, которая в простейшем случае может просто выдать сообщение.  
**Пример обработки событий**.  
Рассмотрим простейший калькулятор, воспринимающий команды в  
командной строке. Здесь приводится упрощенный вариант. Вариант, по  
схеме которого следует выполнить лабораторную работу, приведен в **приложении.**  
Формат команды:  
знак параметр  
Знаки +, –, \*, /, =, ?, q  
Параметр – целое число  
Константы-команды:  
сonst int evNothing = 0;  
сonst int evMessage = 100;  
сonst int cmSet = 1; //занести число  
сonst int cmGet = 2; //посмотреть значение  
сonst int cmAdd = 3; //добавить  
и т.д.  
сonst int cmQuit = 101; //выход  
Класс-событие  
*struct TEvent*  
*{int what*  
*union{*

*int evNothing;*  
*union{int command;*  
*int a;*}  
*}*  
*}*

**Объект-калькулятор, работающий с целыми числами.**  
class TInt{  
int EndState;  
public  
int x;  
Int(int x1);  
virtual ~Int();  
virtual void GetEvent (TEvent &event);  
virtual int Exicute();  
virtual void HandleEvent (TEvent& event);  
virtual void ClearEvent (TEvent& event);  
int Valid();  
void EndExec();  
int GetX();  
void SetX (int newX);  
void AddY (int Y);  
…  
};

Рассмотрим возможную реализацию основных методов.  
void TInit::GetEvent(TEvent &event)  
{char\* OpInt = “+-\*/=?q”; //строка содержит коды операций  
char s[20];  
char code;  
cout<<‘>’;  
cin>>s; code = s[1];  
if(Test(char code,char\*OpInt) // Функции Test проверяет, входит ли  
символ // code в строку OpInt  
{event.what = evMessage;  
swith(code)  
{case ‘+’: event.command=cmAdd;  
break;  
…  
case‘q’: event.command = cmQuit;  
break;

}  
//выделить второй параметр, перевести его в тип int и присвоить  
полю A  
};  
else event.what= evNothing  
};  
int TMyApp::Execute()  
{do{endState=0;  
GetEvent(event); //получить событие  
HandleEvent(event); //обработать событие  
if(event.what!=evNothing) //событие осталось не обработано  
while(!Valid());  
return endState;  
}  
void TInt::HandleEvent(TEvent& event)  
{  
if( event.what == evMessage)  
{  
switch( event.message.command )  
{  
case cmAdd:AddY(event.A);  
СlearEvent( event );  
break;  
…  
case cmQuit:EndExec();  
СlearEvent( event );  
break;  
};  
};  
}

int TInt::Valid();  
{ if (EndState == 0) return 0;  
else return 1;  
}  
void TInt::ClearEvent(TEvent& event)  
{  
Event. what:= evNothing;  
}

void TInt::EndExec()  
{  
EndState= 1;  
}  
void TInt::AddY(int Y)  
{  
x+=Y;  
и т.д.  
void main()  
{  
TInt MyApp;  
MyApp.Execute();  
}

**Порядок выполнения работы.**  
1. Разобрать пример, представленный в приложении. Написать собственный калькулятор, на одном из языков (С++, С#, Java). Добавить собственные функции (возведение в степень, вычитание, деление). Предусмотреть возможность работы с дробными числами. (Пример работы с событиями на языке С# можно найти в приложении А).  
2. Выбрать группу объектов, которые будут обрабатывать события  
(это не могут быть объекты, приведенные в приложении).  
3.Для выбранной группы объектов определить перечень операций,  
которые должны выполняться по командам пользователя.  
4.Определить вид командной строки <код\_операции><параметры>.  
Решить вопросы:  
как кодируются операции? какие передаются параметры?  
5. Определить иерархию объектов. Если необходимо, добавить новые объекты (группы объектов).

6. Определить иерархию классов. Если необходимо, добавить новые  
классы.

7. Определить, какой объект в программе играет роль приложения. В  
случае необходимости добавить в иерархию классов класс ***TApp.*** Решить, в каком классе будет метод *E****xecute***, организующий главный цикл обработки  
событий.  
8. Определить и реализовать необходимые для обработки событий  
методы.  
9. Написать основную функцию (main).  
**Методические указания.**  
1. В качестве группы, для которой организуется обработка событий,  
выбрать группу из лабораторной работы № 2.  
2. Количество различных обрабатываемых команд должно быть не  
менее 5.  
3. Определение классов поместить в файл \*.h. Определение функций-членов класса поместить в файл \*.cpp.  
4. Для констант, связанных с командами, использовать мнемонические имена *cmXXXX.*

5. Дополнительно выполнить задание на языке C# либо Java.

Приложение А.

Далее представлен пример использования событий в языке C#.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace lab5

{

public class CalculatorEvent

{

public int a;

public int b;

public int r;

public void onCalculaterSetA(object sender, EventArgs e)

{

Console.Clear();

Console.WriteLine("Введите A:");

a = int.Parse(Console.ReadLine());

}

public void onCalculaterSetB(object sender, EventArgs e)

{

Console.Clear();

Console.WriteLine("Введите B:");

b = int.Parse(Console.ReadLine());

}

public void onCalculaterSetPlus(object sender, EventArgs e)

{

r = a + b;

}

public void onCalculaterSetMul(object sender, EventArgs e)

{

r = a \* b;

}

public void onCalculaterSetCount(object sender, EventArgs e)

{

a = r;

r = 0;

b = 0;

}

public void onCalculatorClear(object sender, EventArgs e)

{

a = 0;

b = 0;

r = 0;

}

public void onCalculatorMenu(object sender, EventArgs e)

{

Console.Clear();

Console.WriteLine("=============================================================");

Console.WriteLine("Регистр A = {0}", a);

Console.WriteLine("Регистр B = {0}", b);

Console.WriteLine("Регистр R = {0}", r);

Console.WriteLine("=============================================================");

Console.WriteLine();

Console.WriteLine("1 - Очистить регистры");

Console.WriteLine("2 - Ввести регистр А");

Console.WriteLine("3 - Ввести регистр B");

Console.WriteLine();

Console.WriteLine("=============================================================");

Console.WriteLine();

Console.WriteLine("+ - Сложить");

Console.WriteLine("\* - Сложить");

Console.WriteLine("= - Перенести результат из R в регистр A и очистить R, B");

Console.WriteLine();

Console.WriteLine("=============================================================");

Console.WriteLine();

Console.WriteLine();

Console.WriteLine("ESC - Выход");

}

}

public class Calculator

{

public delegate void eventDelegate(object sender, EventArgs e);

public event eventDelegate calculatorIsEvent;

public void activate\_calculatorEvent()

{

// задан ли обработчик события?

if (calculatorIsEvent != null)

{

// если содержится ссылка на обработчик

// то происходит генерация события и вызов обработчика

calculatorIsEvent(this, new EventArgs());

}

}

}

class Program

{

static void Main(string[] args)

{

CalculatorEvent calcEv = new CalculatorEvent();

Calculator calc = new Calculator();

System.ConsoleKeyInfo c = Console.ReadKey();

while (c.Key != ConsoleKey.Escape)

{

calc.calculatorIsEvent += new Calculator.eventDelegate(calcEv.onCalculatorMenu);

calc.activate\_calculatorEvent();

calc.calculatorIsEvent -= new Calculator.eventDelegate(calcEv.onCalculatorMenu);

c = Console.ReadKey();

if (c.KeyChar == '1')

{

calc.calculatorIsEvent += new Calculator.eventDelegate(calcEv.onCalculatorClear);

calc.activate\_calculatorEvent();

calc.calculatorIsEvent -= new Calculator.eventDelegate(calcEv.onCalculatorClear);

}

if (c.KeyChar == '2')

{

calc.calculatorIsEvent += new Calculator.eventDelegate(calcEv.onCalculaterSetA);

calc.activate\_calculatorEvent();

calc.calculatorIsEvent -= new Calculator.eventDelegate(calcEv.onCalculaterSetA);

}

if (c.KeyChar == '3')

{

calc.calculatorIsEvent += new Calculator.eventDelegate(calcEv.onCalculaterSetB);

calc.activate\_calculatorEvent();

calc.calculatorIsEvent -= new Calculator.eventDelegate(calcEv.onCalculaterSetB);

}

if (c.KeyChar == '+')

{

calc.calculatorIsEvent += new Calculator.eventDelegate(calcEv.onCalculaterSetPlus);

calc.activate\_calculatorEvent();

calc.calculatorIsEvent -= new Calculator.eventDelegate(calcEv.onCalculaterSetPlus);

}

if (c.KeyChar == '\*')

{

calc.calculatorIsEvent += new Calculator.eventDelegate(calcEv.onCalculaterSetMul);

calc.activate\_calculatorEvent();

calc.calculatorIsEvent -= new Calculator.eventDelegate(calcEv.onCalculaterSetMul);

}

if (c.KeyChar == '=')

{

calc.calculatorIsEvent += new Calculator.eventDelegate(calcEv.onCalculaterSetCount);

calc.activate\_calculatorEvent();

calc.calculatorIsEvent -= new Calculator.eventDelegate(calcEv.onCalculaterSetCount);

}

}

}

}

}