Лабораторная работа №9

Программа, управляемая событиями

1. Цель задания:

1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2) Разработка программы, управляемой событиями.

2. Теоретические сведения

При ОО подходе к проектированию и программированию часто бывает удобно рассматривать программу как набор независимых объектов, обменивающихся сообщениями, которые называют событиями. Особенно это удобно при организации обмена сообщениями с пользователем. В этом случае источниками сообщений могут служить внешние устройства: мышь, клавиатура. Такая организация программы может оказаться удобной и в других ситуациях. Чтобы объекты обменивались сообщениями необходимо, чтобы в программе присутствовал специальный объект – диспетчер сообщений, который будет контролировать работу всей системы и передавать сообщения от одних объектов другим. Именно таким образом построены многие современные ОС, например MS Windows, где одни объекты – окна посылают сообщения другим окнам, используя системные вызовы. В роли диспетчера сообщений выступает сама ОС.

2.1. Класс-группа

Группа – это объект, в который включены другие объекты. Объекты, входящие в группу, называются элементами группы. Элементы группы, в свою очередь, могут быть группой.

Примеры групп:

1. Окно в интерактивной программе, которое владеет такими элементами, как поля ввода и редактирования данных, кнопки, списки выбора, диалоговые окна и т.д. Примерами таких окон являются объекты классов, порожденных от абстрактного класса TGroup(TDeskTop, TWindow, TDialog) в иерархии классов библиотеки Turbo Vision, и объекты классов, порожденных от TWindowObject в иерархии классов библиотеки OWL.

2. Агрегат, состоящий из более мелких узлов.

3. Огород, состоящий из растений, системы полива и плана выращивания.

4. Некая организационная структура (например, ФАКУЛЬТЕТ, КАФЕДРА, СТУДЕНЧЕСКАЯ ГРУППА).

В отличие от контейнера мы понимаем группу как класс, который не только хранит объекты других классов, но и обладает собственными свойствами, не вытекающими из свойств его элементов. Контейнер используется только для хранения других данных. Примеры контейнеров - объекты контейнерных классов библиотеки STL в C++ (массивы, списки, очереди).

Группа дает второй вид иерархии - иерархию объектов(иерархию типа целое/часть), построенную на основе агрегации, первый вид – иерархия классов, построенная на основе наследования.

Реализовать группу можно разными способами.

2.2. Событие

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.

Сообщение передаваемое от одних объектов другим имеет, как правило, следующие характеристики:

- код класса сообщения, отличающий сообщения объектов одного класса от объектов другого класса;
- адрес объекта, которому предназначено сообщение (м. б. не задан, тогда сообщение могут прочитать все объекты);
- информационное поле.

Событие с точки зрения языка C++ – это объект, отдельные поля которого характеризуют те или иные свойства передаваемой информации, например:

```
struct TEvent
{
      int what;//тип события
      union
      {
            int command;//код команды
            struct

            {
                  int message;
                  int a;//параметр команды
            };
      };
};
```

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.
- ***evMessage*** – событие-сообщение от объекта.

Для события от объекта (*evMessage*) задаются два параметра :

- *command* – код команды, которую необходимо выполнить при появлении данного события;
- передаваемая с событием информация (сообщение).

2.3. 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
             ClearEvent( event );
              break;
     case cmCommand2:
          // Обработка команды cmCommand2
          ClearEvent( event );
           break;
        …
     case cmCommandN:
          // Обработка команды cmCommandN
          ClearEvent( 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
/* просмотрены не все элементы */
       while((event.what != evNothing)
       {
       //вызвать HandleEvent текущего элемента
       //перейти к следующему элементу группы
       }
        break;
    }
}
```

Метод ClearEvent-очистка события.

ClearEvent очищает событие, присваивая полю event.What значение evNothing.

2.4. 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
//событие осталось не обработано
          if(event.what!=evNothing)
               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.

3. Постановка задачи

1. Определить иерархию пользовательских классов (см. лабораторную работу №5). Во главе иерархии должен стоять абстрактный класс с чисто виртуальными методами для ввода и вывода информации об атрибутах объектов.
2. Реализовать конструкторы, деструктор, операцию присваивания, селекторы и модификаторы.
3. Определить класс-группу на основе структуры, указанной в варианте.
4. Для группы реализовать конструкторы, деструктор, методы для добавления и удаления элементов в группу, метод для просмотра группы, перегрузить операцию для получения информации о размере группы.
5. Определить класс Диалог – наследника группы, в котором реализовать методы для обработки событий.
6. Добавить методы для обработки событий группой и объектами пользовательских классов.
7. Написать тестирующую программу.
8. Нарисовать диаграмму классов и диаграмму объектов.

4. Ход работы

Задача

Класс- группа ВЕКТОР с элементами типа int.

Иерархия пользовательских классов:

МАШИНА

- торговая_марка - string
- число_цилиндров - int
- мощность - int

Производный класс ГРУЗОВИК

- характеристика грузоподъемности кузова - int.

События, обрабатываемые группой:

- Создать группу (формат команды: m количество элементов группы).
- Добавить элемент в группу (формат команды: +)
- Удалить элемент из группы (формат команды -)
- Вывести информацию о группе (количество элементов в группе и информацию о каждом объекте) формат команды (?)
- Вывести информацию о значении атрибута Торговая марка для каждого объекта, используя обработчик событий объекта (формат команды /)
- Конец работы (формат команды: q)

1. Создать пустой проект.
2. Добавить в него абстрактный класс Object.
3. В файл Object.h добавить описание класса Object:

```
#pragma once
class Object
{
public:
	Object(void);
	virtual void Show()=0;
	virtual void Input()=0;
	virtual ~Object(void);
};
```

4. В файл Object.cpp добавить определение методов класса Object:

```
#include "Object.h"

Object::Object(void)
{
}

Object::~Object(void)
{
}
```

5. Добавить класс Car, унаследованный от Object. Для этого в файле Car.h разместить описание класса:

```
class Car :
	public Object
{
public:
	Car(void);//конструктор без параметров
public:
	virtual ~Car(void);//деструктор
	void Show();//функция для просмотра атрибутов класса с помощью указателя
	void Input();//функция для ввода значений атрибутов
	Car(string,int,int);//конструктор с параметрами
	Car(const Car&);//конструктор копирования
//селекторы
	string Get_mark(){return mark;}
	int Get_cyl(){return cyl;}
	int Get_power(){return power;}
//модификаторы
	void Set_mark(string);
	void Set_cyl(int);
	void Set_power(int);
	Car& operator=(const Car&);//перегрузка операции присваивания
protected:
	string mark;
	int cyl;
	int power;
};
```

В файле Car.cpp разместить определение методов класса Car.

```
//конструктор без параметров
Car::Car(void)
{
	mark="";
	cyl=0;
	power=0;
}
//деструктор
Car::~Car(void)
{
}
//констрктор с параметрами
Car::Car(string M,int C,int P)
{
	mark=M;
	cyl=C;
	power=P;
}
//конструктор копирования
Car::Car(const Car& car)
{
	mark=car.mark;
	cyl=car.cyl;
	power=car.power;
}
```

```
//селекторы
void Car::Set_cyl(int C)
{
	cyl=C;
}
void Car::Set_mark(string M)
{
	mark=M;
}
void Car::Set_power(int P)
{
	power=P;
}
//оператор присваивания
Car& Car::operator=(const Car&c)
{
	if(&c==this)return *this;
	mark=c.mark;
	power=c.power;
	cyl=c.cyl;
	return *this;
}
//метод для просмотра атрибутов
	 void Car::Show()
 {
	cout<<"\nMARK : "<<mark;
	cout<<"\nCYL : "<<cyl;
	cout<<"\nPOWER : "<<power;
	cout<<"\n";
 }
//метод для ввода значений атрибутов
	 void Car::Input()
{
	 cout<<"\nMark:"; cin>>mark;
	cout<<"\nPower:";cin>>power;
	cout<<"\nCyl:";cin>>cyl;
}
```

6. Добавить класс Lorry, унаследованный от Car. Для этого в файле Lorry.h разместить описание класса:

```
class Lorry :
	public Car
{
public:
	Lorry(void);
public:
	~Lorry(void);
	void Show();
	void Input();
	Lorry(string,int,int,int);
	Lorry(const Lorry & );
	int Get_gruz(){return gruz;}
	void Set_Gruz(int);
	Lorry& operator=(const Lorry&);
protected:
	int gruz;
};
```

В файле Lorry.cpp разместить определение методов класса Lorry

```
Lorry::Lorry(void):Car()
{
	gruz=0;
}
Lorry::~Lorry(void)
```

```
{
}
Lorry::Lorry(string M, int C, int P, int G):Car(M,C,P)
{
	gruz=G;
}
Lorry::Lorry(const Lorry &L)
{
	mark=L.mark;
	cyl=L.cyl;
	power=L.power;
	gruz=L.gruz;
}
void Lorry::Set_Gruz(int G)
{
	gruz=G;
}
Lorry& Lorry::operator=(const Lorry&l)
{
	if(&l==this)return *this;
	mark=l.mark;
	power=l.power;
	cyl=l.cyl;
	return *this;
}
  void Lorry::Show()
{
	cout<<"\nMARK : "<<mark;
	cout<<"\nCYL : "<<cyl;
	cout<<"\nPOWER : "<<power;
	cout<<"\nGRUZ : "<<gruz;
	cout<<"\n";
}
void Lorry::Input()
{
	cout<<"\nMark:"; cin>>mark;
	cout<<"\nPower:";cin>>power;
	cout<<"\nCyl:";cin>>cyl;
	cout<<"\nGRUZ : ";cin>>gruz;
}
```

7. Выполнить тестирование созданных классов. Для этого добавить в проект файл lab8_main.cpp. В файл записать функцию main(), создающую объекты классов Car и Lorry.

```
void main()
{
	Car *a=new Car;
	a->Input();
	a->Show()

	Lorry *b=new Lorry;
	a->Input();
	a->Show()
}
```

8. Добавить класс-группу Vector. Группа содержит указатель на динамический массив указателей типа Object (Object **beg), количество элементов, под которые выделена память (int size), номер последнего элемента, добавленного в группу (int cur). В файл Vector.h записать описание класса:

```
class Vector
{
public:
	Vector(int);//конструктор с параметрами
```

```
public:
	~Vector(void);//деструктор
	void Add();//добавление элемента в вектор
	void Del();
	void Show();
	int operator()();//размер вектора

protected:
	Object**beg;//указатель на первый элемент вектора
	int size;//размер
	int cur;//текущая позиция
};
```

9. В файл Vector.cpp записать реализацию методов класса Vector.

```
//деструктор
Vector::~Vector(void)
{
	if(beg!=0)delete [] beg;
	beg=0;
}
//конструктор с параметрами
Vector::Vector(int n)
{
	beg=new Object*[n];
	cur=0;
	size=n;
}
//добавление объекта, на который указывает указатель p в вектор
void Vector::Add()
{
	Object*p;
//выбор из объектов двух возможных классов
	cout<<"1.Car"<<endl;
	cout<<"2.Lorry"<<endl;
	int y;
	cin>>y;
	if(y==1)//добавление объекта класса Car
	{
	Car*a=new (Car);
	a->Input();//ввод значений атрибутов
	p=a;

	if(cur<size)
	{
		beg[cur]=p;//добавление в вектор
		cur++;
	}
	}
	else
		if(y==2) //добавление объекта класса Lorry
		{
			Lorry *b=new Lorry;
			b->Input();
			p=b;
			if(cur<size)
	{
		beg[cur]=p;
		cur++;
	}
		}
		else return;


}
//просмотр вектора
```

```
void Vector::Show()
{
	if(cur==0) cout<<"Empty"<<endl;
	Object **p=beg;//указатель на указатель типа Object
	for(int i=0;i<cur;i++)
		{
			(*p)->Show();//вызов метода Show() (позднее связывание)
			p++;//передвигаем указатель на следующий объект
	}


}
//операция, которая возвращает размер вектора
 int Vector::operator ()()
 {
	 return cur;
 }
//удаление элемента из вектора, память не освобождается!
 void Vector::Del()
 {
	if(cur==0)return;//пустой
	 cur--;
 }
```

10. Выполнить тестирование созданных классов. В файл lab8_main() записать функцию main(), создающую объекты классов Car , Lorry и Vector и записывающую объекты в вектор.

```
void main()
{
	Car *a=new Car;//создаем объект класса Car
	a->Input();
	Lorry *b=new Lorry; //создаем объект класса Lorry
	b->Input();

	Vector v(10);//Создаем вектор
	Object*p=a;//ставим указатель на объект класса Car
	v.Add(p);//добавляем объект в вектор
	p=b;//ставим указатель на объект класса Lorry
	v.Add(p); //добавляем объект в вектор
	v.Show();//вывод вектора
	v.Del();//удаление элемента
	cout<< "\nVector size="<<v();
}
```

11. Добавим в проект классы и методы, необходимые для обработки событий. Для этого создадим файл Event.h, в который запишем константы для обозначения обрабатываемых событий и структурированный тип, реализующий событие.

```
#pragma once
const int evNothing=0;//пустое событие
const int evMessage=100;//непустое событие
const int cmAdd=1;//добавить объект в группу
const int cmDel=2;//удалить объект из группы
const int cmGet=3;//вывести атрибут всех объектов
const int cmShow=4;//вывести всю группу
const int cmMake=6;//создать группу
const int cmQuit=101;//выход
//класс событие
struct TEvent
{
	int what;//тип события
	union
	{
		int command;//код команды
```

```
            struct

            {
                  int message;
                  int a;//параметр команды
            };
      };
};
```

Этот файл с помощью директивы #include "event.h" будем поключать к тем заголовочным файлам, в которых будут использоваться константы и переменная е типа Event.

12. Добавим в проект класс Dialog и методы, необходимые для обработки событий. В файл Dialog.h добавим описание класса

```
class Dialog :
      public Vector
{
public:
Dialog(void);//контруктор
public:
virtual ~Dialog(void);//деструктор
virtual void GetEvent (TEvent &event);//получить событие
virtual int Execute();//главный цикл обработки событий
virtual void HandleEvent (TEvent& event); //обработчик
virtual void ClearEvent (TEvent& event);//очистить событие
int Valid();//проверка атрибута EndState
void EndExec();//обработка события «конец работы»
protected:
      int EndState;
};
```

13. В файл Dialog.cpp добавим реализацию методов

```
//конструктор
Dialog::Dialog(void):Vector()
{
      EndState=0;
}
//деструктор
Dialog::~Dialog(void)
{
}
//получение события
void Dialog::GetEvent(TEvent &event)
{
string OpInt = "+-?/qam"; //строка содержит коды операций
string s;
string param;

char code;
cout<<'>';
cin>>s; code = s[0];//первый символ команды
if(OpInt.find(code)>=0)//является ли символ кодом операции
{
      event.what = evMessage;
         switch(code)
        {
       case 'm':event.command=cmMake;break;//создать группу
       case '+': event.command=cmAdd;break;//добавить объект в группу
       case '-':  event.command=cmDel;break;//удалить объект из группы
       case '?':  event.command=cmShow;break;//просмотр группы
       case'q': event.command = cmQuit;break; //конец работы
              }
//выделяем параметры команды, если они есть
      if(s.length()>1)
       {
       param=s.substr(1,s.length()-1);
```

```
       int A=atoi(param.c_str());//преобразуем парметр в число
       event.a=A;//записываем в сообщение
       }
}
else event.what=evNothing;//пустое событие
}

int Dialog::Execute()
{ TEvent event;
      do{
            EndState=0;
      GetEvent(event); //получить событие
      HandleEvent(event); //обработать событие

      }
  while(!Valid());
  return EndState;
}

int Dialog::Valid()
{ if (EndState == 0) return 0;
   else return 1;
}
void Dialog::ClearEvent(TEvent& event)
{
   event. what= evNothing;//пустое событие
}

void Dialog::EndExec()
{
   EndState= 1;
}
//обработчик событий
void Dialog::HandleEvent(TEvent& event)
   {
     if( event.what == evMessage)
        {
        switch( event.command )
            {
            case cmMake://создание группы
                  size=event.a; //размер группы
            beg=new Object*[size];//выделяем память под массив указателей
                  cur=0; //текущая позиция
                  ClearEvent(event);//очищаем событие
                  break;
             case cmAdd://добавление
                        Add();
                   ClearEvent( event );
                   break;
                  case cmDel:Del(); //удаление
                   ClearEvent( event );
                   break;
                  case cmShow:Show(); //просмотр
                   ClearEvent( event );
                   break;

            case cmQuit:EndExec(); //выход
        ClearEvent( event );
                    break;
            };
         };
    }
```

14. Выполнить тестирование созданных классов. В файл lab8_main() записать функцию main(), выполняющую обработку команд с помощью класса Dialog.

```
void main()
{
      Dialog D;
      D.Execute();
}
```

15. Для вывода информации о значении атрибута Торговая марка для каждого объекта будем использовать обработчики событий объектов (формат команды /). Для этого добавляем в класс Object чисто виртуальную функцию virtual void HandleEvent(const Event&e)=0.

```
class Object
{
. . . . . .
virtual void HandleEvent(const TEvent &e)=0;
}
```

16. Добавляем в класс Car виртуальную функцию virtual void HandleEvent(const Event&e).

```
class Car :
      public Object
{
public:
. . . .
      void HandleEvent(const TEvent &e);
protected:
      string mark;
      int cyl;
      int power;
};
```

17. Добавляем реализацию этой функции в файл Car.cpp

```
void Car::HandleEvent(const TEvent&e)
 {
  if(e.what==evMessage)//событие-сообщение
   {
       switch(e.command)
       {
            case cmGet:cout<<"mark="<<Get_mark()<<endl;
            break;
       }
   }
 }
```

18. Добавляем функцию функцию virtual void HandleEvent(const Event&e) в класс Vector. Реализация функции будет выгдядеть следующим образом:

```
void Vector::HandleEvent(const TEvent&e)
 {
       if(e.what==evMessage)
       {
            Object**p=beg;
            for(int i=0;i<cur;i++)
           {
                 (*p)->HandleEvent(e);//вызов метода  (позднее связывание)
                 p++;//передвигаем указатель на следующий объект
           }
       }
 }
```

19. В функцию void Dialog::GetEvent(TEvent &event) добавляем операторы

```
  switch(code)
```

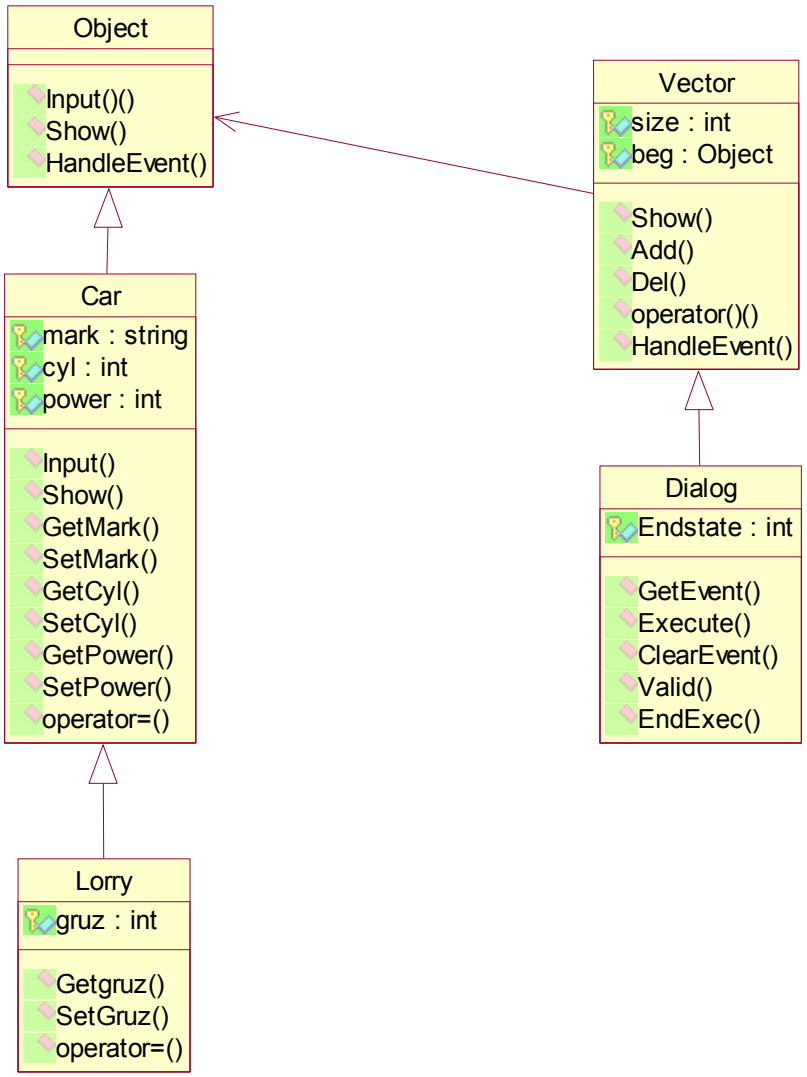
```
        {
            . . . . . .

            case '/':  event.command=cmGet;break;

            . . . . . .
        }
}
else event.what=evNothing;
}
```

20. В функцию void Dialog::HandleEvent(TEvent& event) добавляем оператор

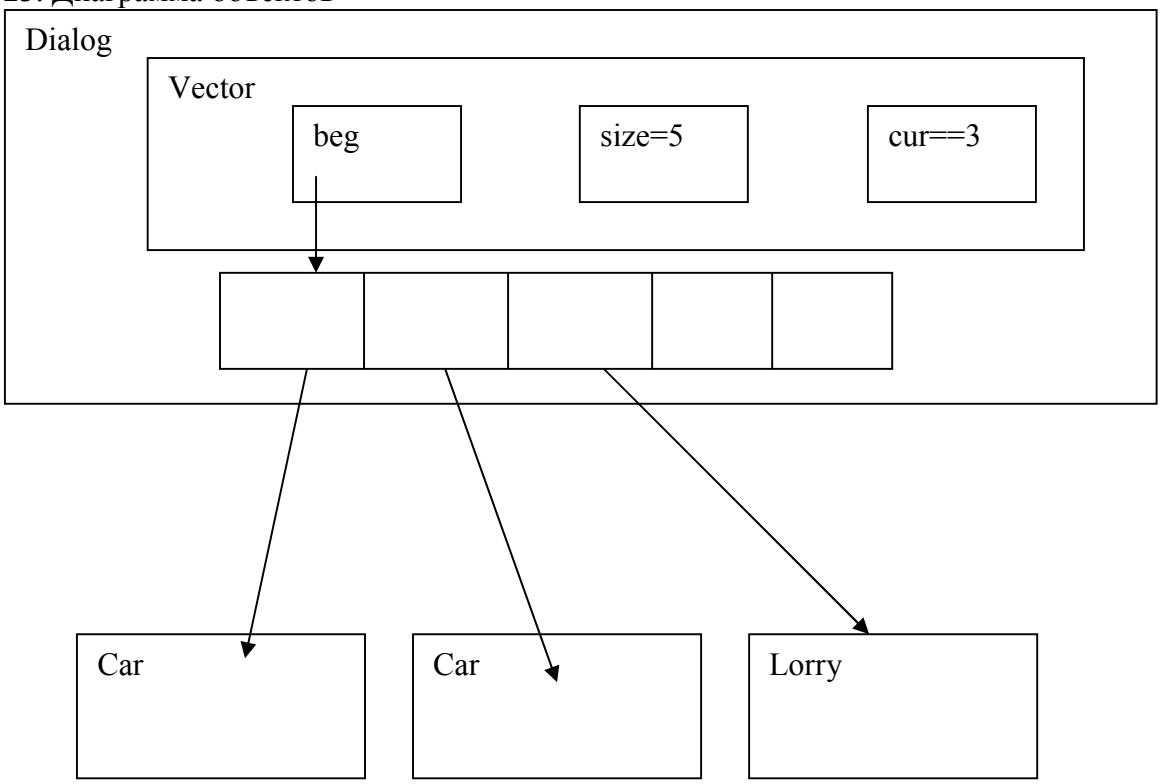
```
switch( event.command )
            {
            . . . . . .
                  default:Vector::HandleEvent(event);
            };
         };
    }
```

21. Запускаем программу на выполнение и тестируем ее работу.

22. Диаграмма классов



23. Диаграмма объектов



5. Варианты

№	Задание
1	Базовый класс: ЧЕЛОВЕК (Person) Имя – string Возраст – int Производный класс СТУДЕНТ (Student) Рейтинг - float Группа – Вектор (Vector). Команды: • Создать группу (формат команды: m количество элементов группы). • Добавить элемент в группу (формат команды: +) • Удалить элемент из группы (формат команды -) • Вывести информацию об элементах группы (формат команды: s) • Вывести информацию об элементе группы (формат команды :? номер объекта в группе) • Конец работы (формат команды: q)

2	Базовый класс: ЧЕЛОВЕК (Person) Имя – string Возраст – int Производный класс СОТРУДНИК (Emloyee) Заработная плата – float Должность - string Группа – Вектор (Vector). Команды: • Создать группу (формат команды: m количество элементов группы). • Добавить элемент в группу (формат команды: +) • Удалить элемент из группы (формат команды -) • Вывести информацию об элементах группы (формат команды: s) • Вывести информацию об имени элемента с номером k (формат команды : z k, где k – целое число) • Конец работы (формат команды: q)
3	Базовый класс: ЧЕЛОВЕК (Person) Имя – string Возраст – int Производный класс АБИТУРИЕНТ(ABITURIENT) Количество баллов - int Специальность - string Группа – Вектор (Vector). Команды: • Создать группу (формат команды: m количество элементов группы). • Добавить элемент в группу (формат команды: +) • Удалить элемент из группы (формат команды -) • Вывести информацию об элементах группы (формат команды: s) • Вывести информацию о среднем возрасте (формат команды : z) • Конец работы (формат команды: q)
4	Базовый класс: ПЕЧАТНОЕ_ИЗДАНИЕ(PRINT) Название– string Автор – string Производный класс КНИГА (BOOK) Количество страниц - int Издательство - string Группа – Вектор (Vector). Команды: • Создать группу (формат команды: m количество элементов группы). • Добавить элемент в группу (формат команды: +) • Удалить элемент из группы (формат команды -) • Вывести информацию об элементах группы (формат команды: s) • Вывести информацию о названии элемента группы с номером k (формат команды : z k, где k – целое число)

	• Конец работы (формат команды: q)
5	Базовый класс: ПЕЧАТНОЕ_ИЗДАНИЕ(PRINT) Название– string Автор – string Производный класс ЖУРНАЛ (MAGAZIN) Количество страниц - int Группа – Вектор (Vector). Команды: • Создать группу (формат команды: m количество элементов группы). • Добавить элемент в группу (формат команды: +) • Удалить элемент из группы (формат команды -) • Вывести информацию об элементах группы (формат команды: s) • Вывести информацию о названиях всех элементов группы (формат команды : z) Конец работы (формат команды: q)
6	Базовый класс: ЧЕЛОВЕК (Person) Имя – string Возраст – int Производный класс СТУДЕНТ (Student) Рейтинг - float Группа – Список (List). Команды: • Создать группу (формат команды: m количество элементов группы). • Добавить элемент в группу (формат команды: +) • Удалить элемент из группы (формат команды -) • Вывести информацию об элементах группы (формат команды: s) • Вывести информацию об элементе группы (формат команды :? номер объекта в группе) • Конец работы (формат команды: q)
7	Базовый класс: ЧЕЛОВЕК (Person) Имя – string Возраст – int Производный класс СОТРУДНИК (Emloyee) Заработная плата – float Должность - string Группа – Список (List). Команды: • Создать группу (формат команды: m количество элементов группы). • Добавить элемент в группу (формат команды: +) • Удалить элемент из группы (формат команды -) • Вывести информацию об элементах группы (формат команды: s) • Вывести информацию об имени элемента с номером k (формат команды : z k, где k – целое число) • Конец работы (формат команды: q)

8	Базовый класс: ЧЕЛОВЕК (Person) Имя – string Возраст – int Производный класс АБИТУРИЕНТ(ABITURIENT) Количество баллов - int Специальность - string Группа – Список (List). Команды: • Создать группу (формат команды: m количество элементов группы). • Добавить элемент в группу (формат команды: +) • Удалить элемент из группы (формат команды -) • Вывести информацию об элементах группы (формат команды: s) • Вывести информацию о среднем возрасте (формат команды : z) • Конец работы (формат команды: q)
9	Базовый класс: ПЕЧАТНОЕ_ИЗДАНИЕ(PRINT) Название– string Автор – string Производный класс КНИГА (BOOK) Количество страниц - int Издательство - string Группа – Список (List). Команды: • Создать группу (формат команды: m количество элементов группы). • Добавить элемент в группу (формат команды: +) • Удалить элемент из группы (формат команды -) • Вывести информацию об элементах группы (формат команды: s) • Вывести информацию о названии элемента группы с номером k (формат команды: z k, где k – целое число) • Конец работы (формат команды: q)
10	Базовый класс: ПЕЧАТНОЕ_ИЗДАНИЕ(PRINT) Название– string Автор – string Производный класс ЖУРНАЛ (MAGAZIN) Количество страниц - int Группа – Список (List). Команды: • Создать группу (формат команды: m количество элементов группы). • Добавить элемент в группу (формат команды: +) • Удалить элемент из группы (формат команды -) • Вывести информацию об элементах группы (формат команды: s) • Вывести информацию о названиях всех элементов группы (формат команды : z) • Конец работы (формат команды: q)

11	Базовый класс: ЧЕЛОВЕК (Person) Имя – string Возраст – int Производный класс СТУДЕНТ (Student) Рейтинг - float Группа – Дерево (Tree). Команды: • Создать группу (формат команды: m количество элементов группы). • Добавить элемент в группу (формат команды: +) • Удалить элемент из группы (формат команды -) • Вывести информацию об элементах группы (формат команды: s) • Вывести информацию об элементе группы (формат команды :? номер объекта в группе) • Конец работы (формат команды: q)
12	Базовый класс: ЧЕЛОВЕК (Person) Имя – string Возраст – int Производный класс СОТРУДНИК (Emloyee) Заработная плата – float Должность - string Группа – Дерево (Tree). Команды: • Создать группу (формат команды: m количество элементов группы). • Добавить элемент в группу (формат команды: +) • Удалить элемент из группы (формат команды -) • Вывести информацию об элементах группы (формат команды: s) • Вывести информацию об имени элемента с номером k (формат команды : z k, где k – целое число) • Конец работы (формат команды: q)
13	Базовый класс: ЧЕЛОВЕК (Person) Имя – string Возраст – int Производный класс АБИТУРИЕНТ(ABITURIENT) Количество баллов - int Специальность - string Группа – Дерево (Tree). Команды: • Создать группу (формат команды: m количество элементов группы). • Добавить элемент в группу (формат команды: +) • Удалить элемент из группы (формат команды -) • Вывести информацию об элементах группы (формат команды: s) • Вывести информацию о среднем возрасте (формат команды : z)

	• Конец работы (формат команды: q)
14	Базовый класс: ПЕЧАТНОЕ_ИЗДАНИЕ(PRINT) Название– string Автор – string Производный класс КНИГА (BOOK) Количество страниц - int Издательство - string Группа – Дерево (Tree). Команды: • Создать группу (формат команды: m количество элементов группы). • Добавить элемент в группу (формат команды: +) • Удалить элемент из группы (формат команды -) • Вывести информацию об элементах группы (формат команды: s) • Вывести информацию о названии элемента группы с номером k (формат команды : z k, где k – целое число) • Конец работы (формат команды: q)
15	Базовый класс: ПЕЧАТНОЕ_ИЗДАНИЕ(PRINT) Название– string Автор – string Производный класс ЖУРНАЛ (MAGAZIN) Количество страниц - int Группа – Дерево (Tree). Команды: • Создать группу (формат команды: m количество элементов группы). • Добавить элемент в группу (формат команды: +) • Удалить элемент из группы (формат команды -) • Вывести информацию об элементах группы (формат команды: s) • Вывести информацию о названиях всех элементов группы (формат команды : z) Конец работы (формат команды: q)

6. Контрольные вопросы

1. Что такое класс-группа? Привести примеры таких классов.
2. Привести пример описания класса-группы Список (List).
3. Привести пример конструктора (с параметром, без параметров, копирования) для класса-группы Список.
4. Привести пример деструктора для класса-группы Список.
5. Привести пример метода для просмотра элементов для класса-группы Список.
6. Какой вид иерархии дает группа?
7. Почему во главе иерархии классов, содержащихся в группе объектов должен находиться абстрактный класс?
8. Что такое событие? Для чего используются события?
9. Какие характеристики должно иметь событие-сообщение?
10. Привести пример структуры, описывающей событие.
11. Задана структура события

```
 struct TEvent
{
```

```
int what;
union
     {
          MouseEventType  mouse;
          KeyDownEvent  keyDown;
          MessageEvent  message;
     }
};
```

Какие значения, и в каких случаях присваиваются полю what?

12. Задана структура события

```
struct TEvent
{
     int what;//тип события
     union
     {
          int command;//код комманды
          struct//параметры команды
          {
               int message;
               int a;
          };
     };
};
```

Какие значения, и в каких случаях присваиваются полю command?

13. Задана структура события

```
struct TEvent
{
     int what;//тип события
     union
     {
          int command;//код комманды
          struct//параметры команды
          {
               int message;
               int a;
          };
     };
};
```

Для чего используются поля a и message?

14. Какие методы необходимы для организации обработки сообщений?
15. Какой вид имеет главный цикл обработки событий-сообщений?
16. Какую функцию выполняет метод ClearEvent()? Каким образом?
17. Какую функцию выполняет метод HandleEvent ()?Каким образом?
18. Какую функцию выполняет метод GetEvent ()?
19. Для чего используется поле EndState? Какой класс (объект) содержит это поле?
20. Для чего используется функция Valid()?

7. Содержание отчета

1) Постановка задачи (общая и конкретного варианта).
2) Описание класса-контейнера.
3) Определение компонентных функций.
4) Описание класса-итератора и его компонентных функций
5) Функция main().

6) Объяснение результатов работы программы.
7) Ответы на контрольные вопросы.