ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
ДИМИТРОГРАДСКИЙ ИНСТИТУТ ТЕХНОЛОГИЙ,
УПРАВЛЕНИЯ И ДИЗАЙНА

Лабораторная работа №5
по курсу: “ООП”
“*Композиция и иерархия*”
Вариант №6

Выполнил студент гр. ВТ-31: Потеренко А.Г.
Проверил преподаватель: Наскальнюк А.Н.

Димитровград 2006 г.

Содержание.

Задание для работы.

Создать карту и используя композицию – колоду карт. Конструкторы должны инициализировать колоду упорядочено и случайным образом. Создать производный класс от колоды – пасьянс, в котором выбираются по 3 карты и, если 2 крайние одного цвета, то их выбирают. Всю колоду проходят 3 раза.

Теория к заданиям.

Механизм наследования классов позволяет строить иерархии, в которых производные классы получают элементы родительских, или базовых, классов и могут дополнять их или изменять их свойства. При большом количестве никак не связанных классов управлять ими становится невозможным. Наследование позволяет справиться с этой проблемой путем упорядочивания и ранжирования классов, то есть объединения общих для нескольких классов свойств в одном классе и использования его в качестве базового.

Классы, находящиеся ближе к началу иерархии, объединяют в себе наиболее общие черты для всех нижележащих классов. По мере продвижения вниз по иерархии классы приобретают все больше конкретных черт. Множественное наследование позволяет одному классу обладать свойствами двух и более родительских классов.

Ключи доступа

Ключ доступа	Спецификатор в базовом классе	Доступ в производном классе
private	private protected public	нет private private
protected	private protected public	нет protected protected
public	private protected public	нет protected public

При описании класса в его заголовке перечисляются все классы, являющиеся для него базовыми. Возможность обращения к элементам этих классов регулируется с помощью ключей доступа private, protected, public:

```
class имя : [private | protected | public] базовый класс

 { тело класса };
```

Если базовых классов несколько, они перечисляются через запятую. Ключ доступа может стоять перед каждым классом, например:

```
class A {…};

class B {…};

class C {…};

class D: A, protected B, public C {…};
```

По умолчанию для классов используется ключ доступа private, а для структур public.

До сих пор мы рассматривали только применяемые к элементам класса спецификаторы доступа private и public. Для любого элемента класса может также использоваться спецификатор

protected, который для одиночных классов, не входящих в иерархию, равносилен private. Разница между ними проявляется при наследовании, что можно видеть из приведенной таблицы:

Как видно из таблицы, private элементы базового класса в производном классе недоступны вне зависимости от ключа. Обращение к ним может осуществляться только через методы базового класса.

Элементы protected при наследовании с ключом private становятся в производном классе private, в остальных случаях права доступа к ним не изменяются.

Доступ к элементам public при наследовании становится соответствующим ключу доступа.

Если базовый класс наследуется с ключом private, можно выборочно сделать некоторые его элементы доступными в производном классе, объявив их в секции public с производного класса с помощью операции доступа к области видимости:

```
class Base
  {
   …
   public: void f();
  };
class Derived : private Base
  {
   …
   public:
     Base::void f();
  };
```

Простое наследование

Простым называется наследование, при котором производный класс имеет одного родителя. Для различных методов класса существуют разные правила наследования — например, конструкторы и операция присваивания в производном классе не наследуются, а деструкторы наследуются.

Рассмотрим правила наследования различных методов.

Конструкторы не наследуются, поэтому производный класс должен иметь собственные конструкторы. Порядок вызова конструкторов определяется приведенными ниже правилами.

? Если в конструкторе производного класса явный вызов конструктора базового класса

  отсутствует, автоматически вызывается конструктор базового класса по умолчанию (то есть

  тот, который можно вызвать без параметров).

? Для иерархии, состоящей из нескольких уровней, конструкторы базовых классов вызываются

  начиная с самого верхнего уровня. После этого выполняются конструкторы тех элементов

  класса, которые являются объектами, в порядке их объявления в классе, а затем

  исполняется конструктор класса.

? В случае нескольких базовых классов их конструкторы вызываются в порядке объявления.

ВНИМАНИЕ

Если конструктор базового класса требует указания параметров, он должен быть явным образом вызван в конструкторе производного класса в списке инициализации (это продемонстрировано в трех последних конструкторах).

? Вызов функций базового класса предпочтительнее копирования фрагментов кода из функций базового класса в функции производного. Кроме сокращения объема кода, этим достигается упрощение модификации программы: изменения требуется вносить только в одну точку программы, что сокращает количество возможных ошибок.

Ниже перечислены правила наследования деструкторов.

? Деструкторы не наследуются, и если программист не описал в производном классе деструктор, он формируется по умолчанию и вызывает деструкторы всех базовых классов.

? В отличие от конструкторов, при написании деструктора производного класса в нем не требуется явно вызывать деструкторы базовых классов, поскольку это будет сделано автоматически.

? Для иерархии классов, состоящей из нескольких уровней, деструкторы вызываются в порядке, строго обратном вызову конструкторов: сначала вызывается деструктор класса, затем — деструкторы элементов класса, а потом деструктор базового класса.

Множественное наследование

Множественное наследование означает, что класс имеет несколько базовых классов. Если в базовых классах есть одноименные элементы, при этом может произойти конфликт идентификаторов, который устраняется с помощью операции доступа к области видимости:

```
class monstr
  {
    public: int get_health();
    …
  };
class hero
  {
    public:  int get_health();
    …
  };
class ostrich: public monstr, public hero{
…
};
int main()
  {
    ostrich A;
    cout « A.monstr::get_health();
    cout « A.hero::get_health();
}
```

Как видно из примера, для вызова метода get_health требуется явно указать класс, в котором он описан. Использование обычной для вызова метода класса конструкции

```
A.get_health();
```

приведет к ошибке, поскольку компилятор не в состоянии разобраться, к методу какого из базовых классов требуется обратиться.

Если у базовых классов есть общий предок, это приведет к тому, что производный от этих базовых класс унаследует два экземпляра полей предка, что чаще всего является нежелательным. Чтобы избежать такой ситуации, требуется при наследовании общего предка определить его как виртуальный класс:

```
class monstr
{
…
};
class daemon: virtual public monstr
{
…
};
class lady: virtual public monstr
{
…
};
class baby: public daemon , public lady
{
…
};
```

Класс baby содержит только один экземпляр полей класса monstr. Если класс наследуется и как виртуальный, и обычным образом, в производном классе присутствовать отдельные экземпляры для каждого невиртуального вхождения и еще один экземпляр для виртуального.

Множественное наследование применяется для того, чтобы обеспечить производный класс свойствами двух или более базовых. Чаще всего один из этих классов является основным, а другие обеспечивают некоторые дополнительные свойства, поэтому они называются классами подмешивания. По возможности классы подмешивания должны быть виртуальными и создаваться с помощью конструкторов без параметров, что позволяет избежать многих проблем, возникающих при ромбовидном наследовании (когда у базовых классов есть общий предок).

Алгоритм работы программ.

Создаются 3 класса – карты и ее свойства, класс колоды – ее инициализация, класс пасьянс – демонстрация самого расклада карт.

Листинги.

```
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <OleCtnrs.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
class KARTA
  {
    protected:
     bool FLAG; //Уничтожать при пасьянсе или нет
     TShape * SH;
     TLabel * LB;
     int ID;
    public:
     KARTA (TScrollBox * SC, int X1, int X2, int X3, int X4, int IDS, bool flag, TColor CO)
       {
         ID=IDS;
         FLAG=false;
         LB = new TLabel (NULL);
         LB->Left=X2+25;
         LB->Top=X1+25;
         LB->Caption=ID;
         LB->Color=clBtnShadow;
         LB->Parent=SC;
         //---------------------------------------------------------------------------
```

```
         if (flag==false)
          {//При создании колоды нужно определить цвет - при пасьянсе - только вывод
           int m=random(2);
           SH = new TShape (NULL);
           if (m==0)
             SH->Brush->Color=clTeal;
           else
             SH->Brush->Color=clGrayText;
          }
         else
          {
             SH = new TShape (NULL);
             SH->Brush->Color=CO;
          };
         SH->Top=X1;
         SH->Left=X2;
         SH->Height=X3;
         SH->Width=X4;
         SH->Shape=stRoundRect;
         SH->Pen->Mode=pmXor;
         SH->Parent=SC;
       };
     TShape * SHOW1()
      {
       return SH;
      }
     TLabel * SHOW2()
      {
       return LB;
      };
     int SHOW3()
      {
       return ID;
      };
  };
//////////////////////////////////////////////////////////////////////////////
class KOLODA
  {
    //////////////////////////////////////////////////////////////////////////
    protected:
      int YY;
      TScrollBox * SCROLL;
      struct STRUCT_KART
        {
          bool FLAG;
          TShape * SH;
          TLabel * LB;
          int ID;
          STRUCT_KART * NEXT;
        };
      STRUCT_KART * FIRST, * TEC, * BUF1, * BUF2, * BUF3;
    //////////////////////////////////////////////////////////////////////////
    public:
     void VIEW(TScrollBox * S)
       {
         SCROLL=S;
         randomize();
         FIRST=NULL;
         TEC=NULL;
         ////////////////////Цикл создания всей колоды//////////////////////
         int TOP=10;
         int STET=1;
         for (int m=0;m<12;m++)
           {
              int LEFT=20;
              for (int n=0;n<3;n++)
                {
                   if (FIRST==NULL)
                     {
                       FIRST=new(STRUCT_KART);
                       TEC=FIRST;
                       //clWhite - значение здесь не играет - он здесь не исп.
                       KARTA K(SCROLL,TOP,LEFT,89,65,STET,false,clWhite);
                       TEC->SH=K.SHOW1();
                       TEC->LB=K.SHOW2();
                       TEC->ID=K.SHOW3();
                       TEC->FLAG=false;
```

```
                        TEC->NEXT=NULL;
                      }
                    else
                      {
                        TEC->NEXT=new(STRUCT_KART);
                        TEC=TEC->NEXT;
                        KARTA K(SCROLL,TOP,LEFT,89,65,STET,false,clWhite);
                        TEC->SH=K.SHOW1();
                        TEC->LB=K.SHOW2();
                        TEC->ID=K.SHOW3();
                        TEC->FLAG=false;
                        TEC->NEXT=NULL;
                      };
                    LEFT=LEFT+80;
                    STET++;
                    for (int i=0;i<1000;i++)
                      for (int i=0;i<1000;i++)
                       {Application->ProcessMessages();};
                  }
                TOP=TOP+100;
            };
          YY=TOP;  //Запоминаем конечную высоту при выводе 36 карт
        };
  };
////////////////////////////////////////////////////////////////////////////
class PASNS : public KOLODA
 {
   protected:
    TLabel * LAB;
   public:
     void RASKLAD (TScrollBox * S) //Раскладываем пасьянс
       {
         TColor COLORS1;
         TColor COLORS2;
         //////////////////Проходим 3 раз//////////////////////////////////////
         int TOP;
         for (int ZZ=0;ZZ<3;ZZ++)
         {
           ////////////////////Смотрим 3 крайние одного цвета//////////////////
           TEC=FIRST;
           while (TEC!=NULL)
              {
                bool fl=false; //узнать в конце 3 карты или 1 или 2
                int IDK=0;
                while (TEC!=NULL)
                  {
                    if (IDK==0) { BUF1=TEC; COLORS1=TEC->SH->Brush->Color; };
                    if (IDK==2) { BUF2=TEC; COLORS2=TEC->SH->Brush->Color; };
                    IDK++;
                    if (TEC==NULL && IDK==3) fl=true;
                    TEC=TEC->NEXT;
                    if (IDK==3) break;
                  }
                if (COLORS1==COLORS2)
                 {
                  ///////////////Помечаем для удаления//////////////////////////
                  BUF3=TEC;
                  TEC=BUF1;
                  TEC->FLAG=true;
                  TEC=BUF2;
                  TEC->FLAG=true;
                  TEC=BUF3;
                 };
              }
           ////////////////////Удаляем лишние/////////////////////////////////////
           TEC=FIRST;
            while (TEC!=NULL)
              {
                if (TEC->FLAG==true)
                  {
                    if (TEC==FIRST)
                      {
                        //удаляемый элемент первый
                        FIRST=TEC->NEXT;
                        delete TEC;
                        TEC=FIRST;
                      }
```

```
                    else
                    {
                    if (TEC->NEXT==NULL)
                     {
                       //удаляемый элемент последний
                       delete TEC;
                       TEC=BUF2;
                       TEC->NEXT=NULL;
                     }
                    else
                     {
                       //удаляемый элемент ни первый, ни последний
                       BUF2->NEXT=TEC->NEXT;
                       delete TEC;
                       TEC=BUF2;
                     };
                    };
                  };
                BUF2=TEC; //указатель на предпоследний
                TEC=TEC->NEXT;
              };
           ///////////////////////Вывод/////////////////////////////////////////
           TOP=YY+20;
           LAB = new TLabel (NULL);
           LAB->Left=100;
           LAB->Top=TOP;
           LAB->Caption="Проход №";LAB->Caption=LAB->Caption+(ZZ+1);
           LAB->Parent=SCROLL;
           TOP=TOP+20;
           ////////////////////////////////////////////////////////////////////
           TEC=FIRST;
           while (TEC!=NULL)
            {
             int IDK=0;
             int LEFT=20;
             while (TEC!=NULL)
               {
                  KARTA K(SCROLL,TOP,LEFT,89,65,TEC->ID,true,TEC->SH->Brush->Color);
                  IDK++;
                  TEC=TEC->NEXT;
                  LEFT=LEFT+80;
                  if (IDK==3) break;
                  for (int i=0;i<1000;i++)
                      for (int i=0;i<1000;i++)
                       {Application->ProcessMessages();};
               };
             TOP=TOP+100;
            }
           YY=TOP;
         }
       };
 };
//////////////////////////////////////////////////////////////////////////////
class TForm1 : public TForm
{
__published:   // IDE-managed Components
        TButton *Button1;
        TButton *Button2;
        TScrollBox *ScrollBox1;
        void __fastcall Button2Click(TObject *Sender);
        void __fastcall Button1Click(TObject *Sender);
private:       // User declarations
public:        // User declarations
        __fastcall TForm1(TComponent* Owner);
        KOLODA KOL;
        PASNS PAS;
};
extern PACKAGE TForm1 *Form1;
```

```
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include <stdlib.h>
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner){}
//////////////////////////Создать колоду//////////////////////////////////////
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  PAS.VIEW(ScrollBox1);
}
//////////////////////Разложить пасьянс/////////////////////////////////////
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PAS.RASKLAD(ScrollBox1);
  Button2->Enabled=false;
  Button1->Enabled=false;
}
```