Лабораторная работа №3.
дополнительный материал

Рассмотренная теория: виртуальные функции, абстрактные классы.

Изучить реализацию принципа полиморфизма через использование виртуальных функций при наследовании.

Виртуальные функции. В языке C++ полиморфизм реализуется посредством виртуальных функций. Виртуальная функция – это функция, объявленная с ключевым словом virtual в базовом классе и переопределенная в одном или нескольких производных этого класса. Объявление

```
virtual void print(void);
```

говорит о том, что функция print может быть различной для базового и разных производных классов. В производных классах функция может иметь список параметров, отличный от параметров виртуальной функции базового класса. В этом случае эта функция будет не виртуальной, а перегруженной. Механизм вызова виртуальных функций можно пояснить следующим образом. При создании нового объекта для него выделяется память. Для виртуальных функций (и только для них) создается указатель на таблицу функций, из которой выбирается требуемая функция в процессе выполнения. Если в некотором классе задана хотя бы одна виртуальная функция, то все объекты этого класса содержат указатель на связанную с их классом виртуальную таблицу. Эта таблица содержит адреса (указатели на первые инструкции) действительных функций, которые будут вызваны. Доступ к виртуальной функции осуществляется через этот указатель и соответствующую таблицу (т. е. осуществляется косвенный вызов функции). Если функция вызвана с использованием ее полного имени, то виртуальный механизм игнорируется. Свойство виртуальности проявляется только тогда, когда обращение к функции идет через указатель или ссылку на объект. Указатель или ссылка могут указывать как на объект базового, так и на объект производного класса.

Рассмотрим пример использования виртуальной функции

```
#include <iostream>
#include <iomanip>
#include <string.h>
using namespace std;

class Base //  базовый класс
{
public:
   virtual char *name(void)
   {
      return "noname";
   }
```

```
    virtual double area(void)
    {
        return 0;
    }
};

class Rect : public Base // производный класс «Прямоугольник»
{
    int h, s; // размеры прямоугольника
public:
    Rect(int H, int S) // конструктор
    {
        h = H;
        s = S;
    }
    char *name(void);  // вывод на экран названия фигуры
    double area(void); // вывод на экран площади фигуры
};

char *Rect::name(void) // вывод на экран названия фигуры
{
    return "прямоугольник";
}
double Rect::area(void) // вывод на экран площади фигуры
{
    return h * s;
}
class Circle : public Base // производный класс «Окружность»
{
    int r; // радиус окружности
public:
    Circle(int R) // конструктор
    {
        r = R;
    }
    char *name(void);  // вывод на экран названия фигуры
    double area(void); // вывод на экран площади фигуры
};

char *Circle::name(void) // вывод на экран названия фигуры
{
    return "круг";
}
double Circle::area(void) // вывод на экран площади фигуры
{
    return 3.14 * r * r;
}
```

```
int main()
{
   setlocale(LC_ALL, "Russian");
   Base *p[2]; //  массив указателей на базовый класс
   Rect obj1(3, 4);
   Circle obj2(5);
   p[0] = &obj1;
   p[1] = &obj2;
   for (int i = 0; i < 2; i++)
      cout << "площадь " << p[i]->name() << setw(10) << p[i]->area() << endl;
   return 0;
}
```

Массив указателей p хранит адреса объектов производных классов и необходим для вызова виртуальных функций этих классов. Если функции name() и area() в базовом классе объявлены как virtual и мы вызываем эти функции через указатель базового класса, указывающий на объекты производных классов, то программа будет динамически (т. е. во время выполнения программы) выбирать соответствующие функции name() и area() производного класса. Это называется динамическим связыванием (dynamic binding). Когда виртуальная функция вызывается путем обращения к заданному объекту по имени и при этом используется операция доступа к элементу «точка», тогда эта ссылка обрабатывается во время компиляции и это называется статическим связыванием.

Если функция была объявлена как виртуальная в некотором классе (базовом классе), то она остается виртуальной независимо от количества уровней в иерархии классов, через которые она прошла.

Приведем основные правила использования виртуальных функций:

– виртуальный механизм поддерживает полиморфизм на этапе выполнения программы. Это значит, что требуемая версия программы выбирается на этапе выполнения программы, а не компиляции;

– класс, содержащий хотя бы одну виртуальную функцию, называется полиморфным;

– виртуальные функции можно объявлять только в классах (class) и структурах (struct);

– виртуальными функциями могут быть только нестатические функции (без спецификатора static), т. к. характеристика virtual наследуется. Функция производного класса автоматически становится virtual;

– виртуальные функции можно объявлять со спецификатором friend для другого класса;

– виртуальными функциями могут быть только не глобальные функции (т. е. компоненты класса);

– если виртуальная функция объявлена в производном классе со спецификатором virtual, то можно рассматривать новые версии этой функции в классах, наследуемых из этого производного класса;
– для вызова виртуальной функции требуется больше времени, чем для не виртуальной. При этом также требуется дополнительная память для хранения виртуальной таблицы;
– при использовании полного имени при вызове виртуальной функции виртуальный механизм не поддерживается.
Приведем еще пример использования виртуальной функции.

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

class Grup // базовый класс
{
protected:
   char *spec; // название специальности
   long gr;    // номер группы
public:
   Grup(char *SPEC, long GR) // конструктор
   {
      spec = new char[40];
      strcpy_s(spec, 40 * sizeof(char), SPEC);
      gr = GR;
   }
   ~Grup() // деструктор
   {
      cout << "деструктор класса grup" << endl;
      delete[] spec;
   }
   virtual void print(void); // объявление виртуальной функции
};

class Asp : public Grup // производный класс
{
   char *fam; // фамилия
   int oc[4]; // массив оценок
public:
   // конструктор производного класса
   Asp(char *SPEC, long GR, char *FAM, int OC[]) : Grup(SPEC, GR)
```

```
    {
        fam = new char[30];
        strcpy_s(fam, 30 * sizeof(char), FAM);
        for (int i = 0; i < 4; i++)
            oc[i] = OC[i];
    }
    ~Asp()
    {
        cout << "Деструктор класса asp" << endl;
        delete[] fam;
    }
    void print(void);
};

void Grup::print(void) // определение виртуальной функции
{
    cout << setw(10) << "Специальность" << setw(10) << "Группа" << endl;
    cout << setw(10) << spec << setw(10) << gr << endl;
}

void Asp::print(void)
{
    Grup::print(); // вызов функции базового класса
    cout << setw(10) << "ФИО" << setw(10) << "Оценки" << endl;
    cout << setw(10) << fam;
    for (int i = 0; i < 4; i++)
        cout << setw(4) << oc[i];
    cout << endl;
}

int main()
{
    setlocale(LC_ALL, "Russian");
    int OC[] = {4, 5, 5, 4}; // массив оценок
    char SP[40];
    long GR;
    char FAM[40];
    cout << "Введите название специальности" << endl;
    cin >> SP;
    cout << "Введите номер группы" << endl;
    cin >> GR;
    cout << "Введите фамилию" << endl;
```

```
    cin >> FAM;
    Grup ob1(SP, GR), *p;
    Asp ob2(SP, GR, FAM, OC);
    cout << "Результат" << endl;
    p = &ob1;   // указатель на объект базового класса
    p->print(); // вызов функции базового класса
    p = &ob2;
    p->print(); // вызов функции производного класса
}
```

Абстрактные классы. Базовый класс обычно содержит ряд виртуальных функций, которые часто фиктивны и имеют пустое тело. Эти функции существуют как некоторая абстракция, конкретное значение им придается в производных классах. Такие функции называются чисто виртуальными (pure virtual function), т. е. такими, тело которых не определено. Общая форма записи чисто виртуальной функции имеет вид

virtual прототип функции = 0;

Если класс является производным класса с чисто виртуальной функцией и эта функция в нем не описана, тогда функция остается чисто виртуальной и в этом производном классе. Следовательно, такой производный класс является абстрактным. Хотя иерархия классов не требует обязательного включения в нее каких-либо абстрактных классов, однако программы, использующие объектно-ориентированное программирование, все же имеют иерархию, порожденную абстрактным базовым классом. Абстрактные классы могут составлять несколько уровней иерархии. В качестве примера можно привести иерархию форм (рис. 5).

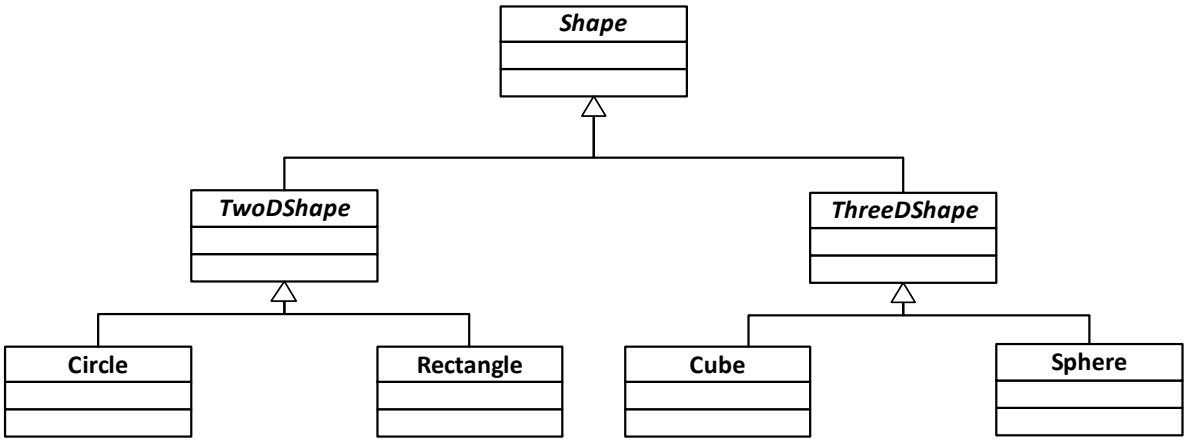

Рис. 5. Иерархия наследования классов

Иерархия может порождаться абстрактным базовым классом *Shape*. На уровень ниже можно получить два абстрактных класса *TwoDShape* и *ThreeDShape*. При переходе еще на один уровень ниже можно определить конкретные классы для

двухмерных (Circle, Rectangle) и трехмерных (Cube, Sphere) форм. Согласно языку UML имя абстрактного класса пишется курсивом.

Рассмотрим пример программы приведенной иерархии классов, в которой будет выводиться название фигуры, площадь двухмерных и объем трехмерных фигур:

```
#include<iostream>
#include<iomanip>
using namespace std;
#include<string.h>
const double pi=3.14159;
class Shape                        // абстрактный базовый класс
{
  public:                                  // чисто виртуальные функции
   virtual void print()=0;                 // печать названия фигуры
   virtual void area() = 0;               // вычисление площади фигуры
   virtual void volume()=0;      // вычисление объема фигуры
};
class TwoDShape: public Sha         // абстрактный производный класс
{
  protected:
     float r;
  public:
     TwoDShape(float r1)                // конструктор с параметрами
     {
       r=r1;
     }
     virtual void area()=0;              // вычисление площади фигуры
     void volume(){}                     // определение функции вычисления
                                    // объема фигуры
};
class ThreeDShape: public Shape // абстрактный производный класс
{
  protected:
     float h;
  public:
     ThreeDShape(float h1)
     {
         h=h1;
     }
     virtual void volume()=0;          // вычисление объема фигуры
     void area(){}                     // определение функции вычисления
                                  // площади фигуры
};
class Circle: public TwoDShape   // класс «Окружность»
{
  public:
      Circle(float r):TwoDShape(r) {}
      void print()
```

```
        {
            cout << "Окружность" << endl;
        }
        void area()
        {
            cout << "Площадь окружности" << setw(10) << pi*r*r << endl;
        }
};
class Rectangle: public TwoDShape     // класс «Квадрат»
{
   public:
      Rectangle(float r):TwoDShape(r)  {}
      void print()
      {      cout << "Квадрат" << endl;    }
      void area()
      {
          cout << "Площадь квадрата" << setw(7) << r*r << endl;
      }
};
class Sphere: public ThreeDShape         // класс «Сфера»
{
   public:
      Sphere(float h): ThreeDShape(h)   {}
      void print()
      {
        cout << "Сфера" << endl;
      }
      void volume()
              {
          cout << "Объем сферы" << setw(10) << (4.0*pi*h*h*h)/3.0 << endl;
      }
};
class Cube: public ThreeDShape            // класс «Куб»
{
   public:
      Cube(float h): ThreeDShape(h)  {}
      void print()
      {
          cout << "Куб" << endl;
      }
      void volume()
      {
          cout << "Объем куба" << setw(5) << h*h*h << endl;
      }
};
int main()
{
   setlocale(LC_ALL,"Russian");
```

```
    Shape *ptr[4];            //  массив указателей на абстрактный базовый класс
    Circle okr(5);                  // объект класса «Окружность»
    Rectangle pr(5);      // объект класса «Прямоугольник»
    Sphere sf(5);         // объект класса «Сфера»
    Cube kb(5);           // объект класса «Куб»
    ptr[0]=&okr;          // инициализация массива указателей ptr
    ptr[1]=&pr;
    ptr[2]=&sf;
    ptr[3]=&kb;
    for(int i=0; i < 4; i++)
    {
        ptr[i]->print();        // вывод названия фигуры
        ptr[i]->area();         // вывод площади фигуры
        ptr[i]->volume();    // вывод объема фигуры
    }
    return 0;
}
```