МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Уральский федеральный университет имени первого Президента России
Б.Н.Ельцина»

Институт радиоэлектроники и информационных технологий - РТФ
Кафедра Информационных технологий

Реализация принципов наследования. Иерархия классов геометрических примитивов

Методические указания к лабораторной работе
по дисциплине «Технология проектирования и тестирования
программного обеспечения»
для студентов всех форм обучения по направлению
230100 – Информатика и вычислительная техника

Екатеринбург

2012

УДК 004.432.2

Составитель С.П.Трофимов
Научный редактор доц., канд. техн. наук В.П.Битюцкий

РЕАЛИЗАЦИЯ ПРИНЦИПОВ НАСЛЕДОВАНИЯ. ИЕРАРХИЯ КЛАССОВ ГЕОМЕТРИЧЕСКИХ ПРИМИТИВОВ: Метод. разработка к лабораторной работе по дисциплине «Технология проектирования и тестирования программного обеспечения» / сост. С.П. Трофимов. Екатеринбург: УрФУ, 2012. 11 с.

Даются основные понятия, связанные с наследованием классов на языке Си++, и с организацией иерархии классов с использованием механизма виртуальных функций. Рассматривается создание интерфейса иерархии. Материал закрепляется на примере иерархии классов геометрических примитивов и реализации анимации сложных геометрических фигур.

Приводятся контрольные вопросы, указания по подготовке и выполнению лабораторных заданий.

Указания предназначены для студентов всех форм обучения направления 230100 – «Информатика и вычислительная техника».

Библиогр.: 5 назв., Табл.: 1

Подготовлено кафедрой «Информационные технологии»

СОДЕРЖАНИЕ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Парадигма ООП - наследование

Наследование - это возможность объявления нового класса на базе одного или нескольких существующих классов. При простом наследовании родительский или базовый класс единственный. При множественном родительских классов может быть два или более. Множественное наследование создает ряд проблем, поэтому в некоторых реализациях ООП оно запрещено.

Производный или дочерний класс наследует от родителей данные и методы. Данные предоставляются потомку по их именам в родителе. С методами ситуация сложнее.

Потомок может переопределить родительский метод, то есть дочерний класс может определить свой метод с родительским прототипом. Если потомок укажет только прототип метода без реализации, то вызов метода будет запрещен.

Таким образом, переопределение и перегрузка – это разные вещи. Перегруженные функции имеют одно имя, но разный список формальных параметров. Переопределенные функции имеют одно имя и один и тот же список параметров. Переопределение возможно только при наследовании.

Пример 1. Перегруженные функции

```
void msg(char *w){
```

```
 printf("%s", w);
}
void msg(char *w, char *w2){
 printf("%s %s", w, w2);
}
void main(){
     msg("Hello, world");
     msg("Hello, ", "world");
}
```

Пример 2. Переопределенные функции

```
class parent{
 public:
 void msg(char *w){
     printf("%s", w);
 }
};
class child: public parent
{
 public:
 void msg(char *w){
     printf("Message: %s", w);
 }
};
void main(){
 parent par;
 child ch;
 par.msg("Hello, world");  // Hello, world
 ch.msg("Hello, world"); // Message: Hello, world

}
```

Синтаксис наследования

class Base
{ ..... };
class Proizv : [спецификатор_доступа] Base
{ ..... };

Элементы родительского класса могут иметь три спецификатора доступа, и при наследовании возможны три спецификатора доступа. По умолчанию доступ - private. Таким образом, производный класс наследует элементы в 9 различных ситуациях.

Таблица

Спецификаторы доступа при наследовании класса

Доступ в базовом классе	Доступ при наследовании	Доступ в производном классе
public	public	public
	protected	protected
	private	private
protected	public	protected
	protected	protected

	private	private
private	public	нет доступа
	protected	нет доступа
	private	нет доступа

Пример 3.

```
class parent{
      public: int pub;
      protected: int prot;
      private: int pr;
};
class pub_child: public parent{
      //…….
      public:
      void show(){
          printf("%d", prot);
      //printf("%d", pr);  ошибка
      }
};
void main(){
      pub_child  pc;
      pc.pub=0;// допустимо
      pc.prot=0;// ошибка
      pc.pr=0;// ошибка
      pc.show();// допустимо
}
```

Виртуальные функции

Метод может быть объявлен как virtual. Ключевое слово virtual обязывает компилятор генерировать дополнительную информацию об этом методе. Назначение виртуальных функций в следующем.

Допустим имеется указатель на базовый класс, содержащий адрес существующего объекта производного класса. Тогда по этому указателю можно обратиться:

- ко всем данным из родительского класса;
- к не виртуальным методам родительского класса;
- к методы производного класса, объявленные как виртуальные в базовом классе.

К данным из производного класса обратиться нельзя.

Пример 4.

```
#include <stdio.h>
#include <conio.h>
class Base{
public:
 virtual void virt(){
     cout << "Hello from Base::virt" << endl;
};
void nonvirt(){
     cout << "Hello from Base::nonvirt" << endl;
```

```
};
};
class Proiz : public Base{
public:
virtual void virt(){
     cout << "Hello from Proiz::virt" << endl;
};
void nonvirt(){
     cout << "Hello from Proiz::nonvirt" << endl;
};
};
void main(){
Base *bp = new Proiz;
bp->virt();
bp->nonvirt();
}
```

Будет напечатано

Hello from Proiz::virt

Hello from Base::nonvirt

Чтобы использовать механизм виртуальных функций необходимо применять указатели иди ссылки. Только указатель на класс может ссылаться на объект производного класса без явного переопределения типа. Спецификатор virtual необязателен при переопределении функции в производном классе. Можно обойти механизм виртуальных функций, если указать при вызове имя родительского класса с операцией разрешения видимости.

bp->Base::virt(); // Обходит виртуальный механизм

У базового класса может быть несколько производных классов. Пусть имеется указатель на базовый класс, а в базовом классе - виртуальная функция, переопределенная во всех производных классах. И вот мы вызываем виртуальную функцию. Из какого производного класса она реально будет вызвана? Из того класса, на объект которого реально указывает указатель. Это можно определить только в ходе выполнения программы. Подобное явление называется поздним связыванием. По аналогии с поздним связыванием существует раннее связывание, которое осуществляется на этапе линковки при преобразовании объектного файла в исполняемый файл.

Абстрактные классы и чистые виртуальные функции

Чистая виртуальная функция является виртуальным методом класса, тело которого определяется как =0. Для чистой виртуальной функции не нужно приводить действительное определение в своем классе. Предполагается, что он переопределяется в производных классах.

Абстрактным классом называется класс, содержащий хотя бы одну чистую виртуальную функцию. К абстрактным классам применимы следующие правила:

- нельзя определить объект абстрактного класса,

- можно объявить указатель или ссылку на абстрактный класс,
- если класс, производный от абстрактного, не определяет все чистые виртуальные функции своего абстрактного родителя, он также является абстрактным.

Абстрактные классы, как правило, являются базовыми классами в иерархиях классов. Такие классы содержат интерфейс иерархии, то есть перечень методов, обязательных для всех производных классов.

Примерами иерархии классов являются потоковые классы, объявленные в файле <iostream.h>.

Проектная организация иерархии классов

Для каждого класса, например Pixel, создают два файла. Первый файл Pixel.h называется интерфейсом класса. Он содержит объявление класса с прототипами методов. Определения методов помещают в файл Pixel.cpp, который называют реализацией класса. После компиляции файла Pixel.cpp получают объектный код Pixel.obj.

При передаче класса другому программисту или при переносе его в другой проект, "продажной" частью являются обычно интерфейсный и объектный файлы.

В других файлах проекта с помощью директивы include подключается интерфейс класса. Чтобы предотвратить повторную компиляцию класса, используют механизм условной прекомпиляции.

Пример. Интерфейсный файл Pixel.h

```
#ifndef PIXEL_SECOND
#define PIXEL_SECOND
class Pixel
{…….};
#else
class Pixel;  // предъобъявление класса
#endif
```

Иерархия классов геометрических примитивов

Построим иерархию классов геометрических примитивов, таких как точка, отрезок, окружность, прямоугольник и т.д. Эти примитивы будут использоваться также в производном классе фигуры. Минимальные требования к интерфейсу иерархии включают в себя реализацию прямолинейного движения. Это означает, что каждый примитив должен содержать методы Show, Hide, Move.

Базовый класс

```
class Base{
 public:
 virtual void Hide() = 0;
 virtual void Show() = 0;
 virtual void Move( int dx, int dy) = 0;
};//Base
```

Класс точки

```
class Pixel: public Base
{
 public:
 int x, y, col;
 Pixel(): Base(){};
 Pixel( int _x, int _y, int _col):x(_x), y(_y), col(_col)
 {}
 virtual void Hide(){
     putpixel ( x, y, 0);
 }
 virtual void Show(){
     putpixel ( x, y, col);
 }
 virtual void Move( float dx, float dy){
     Hide();
     x += dx;
     y += dy;
     Show();
 }
};//Pixel
```

Класс окружности

```
class Okr: public Pixel{
 public:
 int r;
 Okr(): Pixel(){};
 Okr(int _x, int _y, int _r) :x(_x),y(_y),r(_r), col(WHITE)
 {}
 virtual void Hide(){
     setcolor ( 0 );
     circle ( x, y, r);
     setcolor ( WHITE);
 }
 virtual void Show(){
     setcolor (WHITE);
     circle ( x, y, r);
 }
 virtual void Move( int dx, int dy){
     Hide();
     x += dx;
     y += dy;
     Show();
 }
};//Okr
```

Класс фигуры

```
class Figure: public Base
{
 public:
 int N;
 Base *A[100];
 Figure(){
```

```
        N = 0;
    }
    void Hide(){
        for ( int i=0; i<N; i++)
        A[i]->Hide();
    }
    void Show(){
        for ( int i=0; i<N; i++)
        A[i]->Show();
    }
    void Move( int dx, int dy){
        for ( int i=0; i<N; i++)
        A[i]->Move(dx,dy);
    }
    void Add ( Base *Ptr){
        if(N<100)
        A[N++] = Ptr;
    }
};//Figure
```

Использование иерархии примитивов

```
void main(){
 int gdriver = DETECT, gmode;
 initgraph(&gdriver, &gmode, "c:\\BORLANDC\\BGI");
 Pixel A(50,50, WHITE), B(150, 50, YELLOW);
 Okr S(50,70, 50), T(150, 70, 50));
 Figure F;
 F.Add((Base *)&A);
 F.Add((Base *)&B);
 F.Add((Base *)&S);
 F.Add((Base *)&T);
 F.Show();
 for (int i=0; i<400; i++){
     F.Move(1, 0);
     delay(100);
 }
 closegraph();
}
```

Контрольные вопросы

1. Что такое файл интерфейса класса и файл реализации класса?
2. Объясните назначение интерфейса иерархии классов.
3. Какова роль чистых функций?
4. Объясните работу механизма виртуальных функций.
5. Как выглядел бы класса Figure, если бы не было механизма виртуальных функций?
6. В чем отличие между ранним и поздним связыванием?
7. Может ли обычный класс иметь абстрактного потомка?

ЛАБОРАТОРНЫЕ ЗАДАНИЯ

1. Иерархия примитивов

Реализуйте иерархию классов геометрических примитивов, содержащую базовый класс, классы точки, отрезка, окружности, фигуры. Интерфейсные методы иерархии должны включать: Show, Hide, Move.
Все классы нужно объявлять в отдельных файлах с разделением на файл реализации класса и файл интерфейса класса.

2. Добавление новых примитивов

Добавьте новые примитивы (по вариантам): дуга окружности, кольцо, треугольник, ромб, отрезок, стрелка и др.

3. Запись примитивов в файл

Добавьте к интерфейсным методам возможность записи примитива в поток.
Замечание. Использовать для этой цели перегрузку операции вывода в поток operator<< нельзя. Эта операция представляет собой дружественную функцию класса. Например, для класса Pixel ее прототип имеет вид

```
friend ostream& operator<<(ostream& os, Pixel P);
```

Но дружественные функции не поддерживаются механизмом виртуальных функций и потому не могут быть интерфейсными методами. Выход состоит в создании обычного метода с прототипом

```
void output(ostream& os);
```

Этот метод включается в абстрактный базовый класс Base и переопределяется во всех производных классах.

4. Чтение фигуры из файла

Добавьте в класс Figure метод read чтения данных из текстового файла. Метод имеет прототип

```
void read(istream& is);
```

Файл может состоять из строк вида
"Окружность: 100 200 30"
"Точка: 100 100 15"
Метод read читает строки, анализирует первое слово и создает объект соответствующего примитива с заданными параметрами инициализации.

ДОПОЛНИТЕЛЬНЫЕ ЗАДАНИЯ

5. Перемещение фигуры по кривой

Произведите из класса фигуры новый класс Fly. Добавьте в данный класс параметрическое задание плоской кривой (x(t), y(t)),t >=0. Переопределите в этом классе метод

void Move(int dt1, int dt2);

так, чтобы фигура перемещалась по заданной линии путем параллельного переноса.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Бруно Бабэ. Просто и ясно о Borland C++. - М.: Бином, 1995, 395с.
2. Страуструп Б. Язык программирования Си++. - М.: Радио и связь, 1991, 348с.
3. Подбельский В. В., Фомин С. С. Программирование на языке Си. Учеб. Пособие. – 2-е доп. изд. М.: Финансы и статистика, 2004, 600 с.
4. Программирование в Си. Организация ввода–вывода: метод.указания / сост. С.П. Трофимов. Екатеринбург: УГТУ, 1998. 14 с.
5. Программирование в Си. Динамическое распределение памяти: метод.указания / сост. С.П. Трофимов. Екатеринбург: УГТУ, 1998. 13с.