МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Государственное образовательное учреждение
высшего профессионального образования
"Донской государственный технический университет"
ДГТУ

Кафедра "Программное обеспечение вычислительной техники и автоматизированных систем"

Отношения между классами: агрегирование и наследование

Методические указания к лабораторной работе по дисциплине "Объектно-ориентированное программирование"

Ростов-на-Дону
20 г.

Составитель: к.ф.-м.н., доц. Габрельян Б.В.

УДК 512.3

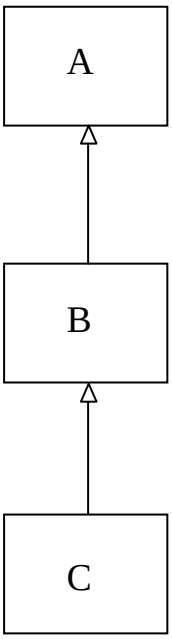
Отношения между классами: агрегирование и наследование: методические указания – Ростов н/Д: Издательский центр ДГТУ, 20 . – с.

В методической разработке рассматриваются особенности наиболее известных отношений между классами: агрегирования и расширения (наследования) в C++. Даны задания по выполнению лабораторной работы. Методические указания предназначены для студентов направления 231000 "Программная инженерия".

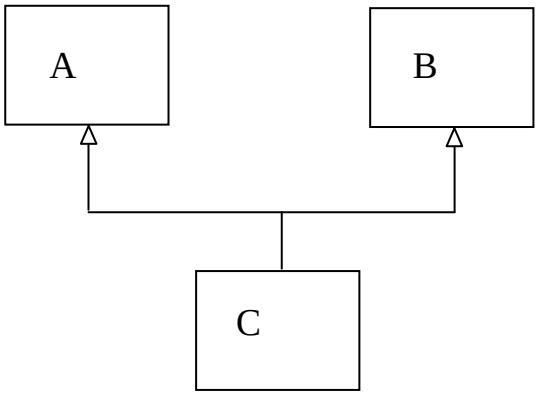
I. Отношение расширения (наследования).

Между классами можно установить отношение, в котором некоторые классы будут выступать в качестве базовых для построения других, производных от них классов. При этом производные классы получают все состояния и все поведение базовых классов и могут добавить свои состояния и свое поведение. Поэтому отношение и называется отношением расширения. Связь между классами становится родственной, поэтому базовые классы называют также родительскими или классами предками, а производные наследниками, дочерними ил классами потомками. На UML-диаграмме классов отношение наследования задается стрелками, направленными от производных к базовым классам. Например,

1)



2)



На обеих диаграммах у класса C есть два базовых класса A и B. Но на схеме 1) класс B является прямым базовым классом для C, а класс A непрямым. На схеме 2) оба класса, и B и A прямые базовые классы для C. поэтому схема 1) задает так называемое единичное, а схема 2) множественное наследование.

II. Виды наследования в C++.

В C++ общий синтаксис объявления производного класса при единичном наследовании таков:

class *Имя_производного_класса* : [*вид_наследования*] *Имя_базового_класса* {

описания полей и методов производного класса

};

Если вид наследования не задан явно, то, в случае, если класс создавался с помощью ключевого слова class, по умолчанию задается закрытое наследование, а если с помощью ключевого слова struct, то открытое. Можно задать открытое (public), закрытое (private) или защищенное (protected) наследование. Вид наследования не влияет на доступ методов производного класса к полям и методам унаследованным от базового класса. Он определяет доступ к этим полям и методам из других классов или внешних функций. При открытом наследовании доступ определяется описанием базового класса. При закрытом наследовании все описания базового класса становятся закрытыми для тех классов и функций, которые используют объекты производного класса. При защищенном наследовании закрытая часть базового класса остается закрытой, а остальные становятся защищенными. Например,

```
class Base {
  private:    int privB;
  protected:  int protB;
  public:     int pubB;
};

class Derived : protected Base {
  public:
      void test() {
      x       privB = 1;   // Ошибка. Нет доступа к закрытому полю
              protB = 2;   // Есть доступ, как у наследника
              pubB = 3;   // Есть доступ, как у всех
      }
};
```

```
class Derived2 : protected Derived {
  public:
        void test() {
        x       privB = 1;   // Ошибка. Нет доступа к закрытому полю
                protB = 2;   // Есть доступ, как у наследника
                pubB = 3;   // Есть доступ, как у всех
        }
};

int main() {
        Derived d;
x       d.privB = 1;   // Ошибка. Нет доступа к закрытому полю
x       d.protB = 2;   // Ошибка. Нет доступа т.к. не наследник
x       d.pubB = 3;   // Ошибка. Нет доступа т.к. не наследник
        return 0;
}
```

III. Множественное наследование.

При множественном наследовании у класса наследника есть не менее двух прямых предков. При объявлении наследования вид наследования задается индивидуально для каждого базового класса. Например,

```
class Base1 {};
class Base2 {};
class Derived : public Base1, protected Base2 {};
```

При создании объекта производного класса, прежде всего вызываются конструкторы базовых классов в том порядке, в котором они заданы в описании класса наследника, потом конструктор производного класса. При уничтожениии объекта производного класса деструкторы вызываются в обратном порядке. Чтобы организовать ромбовидное наследование необходимо использовать виртуальные базовые классы:

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

IV. Создание семейства полиморфных классов.

Рассмотрим задачу создания графического редактора. Пусть это будет программная система, оперирующая плоскими графическими примитивами: треугольниками, квадратами, окружностями и т.д. Каждый примитив может быть закрашен некоторым цветом или не закрашен. У каждого примитива есть некоторая точка, позволяющая привязать его к некоторой области экрана и размеры. Тогда рисунок можно рассматривать как совокупность примитивов, имеющих нужные размеры и выводимых на экран в нужном порядке. При сохранении рисунка на внешнем устройстве можно создавать метафайл команд, т.е. текстовый файл, содержащий описания примитивов (тип, точка привязки, размеры, цвет). Для отображения сохраненного изображения нужно прочитать команды из файла, преобразовать их в объекты и нарисовать каждый объект на экране в том порядке, в котором команды следовали в файле. Представлением для рисунка в программе будет некоторая единая структура данных (массив, список), содержащая информацию обо всех примитивах. Но массив и, обычно, список содержат элементы одинакового типа, а примитивы представлены разными классами. Как объединить их в массив или список? Можно воспользоваться тем, что указатель на базовый класс может содержать адрес объекта любого производного от него класса. Тогда рисунок можно представить массивом указателей на базовый класс. Тем самым нам нужно создать некоторый базовый класс и следить, что каждый примитив обязательно был его наследником. Необходимо в базовом классе задать общее поведение, характерное для всех примитивов, т.к. через указатель на базовый класс можно вызвать только те методы, которые объявлены в этом классе. С другой стороны, каждый класс наследник отображает себя на экране по-своему, поэтому нужно, чтобы через

указатель на базовый класс вызывался переопределенный метод конкретного производного класса. Такое поведение обеспечивают только полиморфные классы. Например,

```
class Shape {   // базовый класс
  protected:
      int x, y;  // точка привязки примитива к некоторой точке на экране
  public:
      virtual void show() {}  /* общее поведение – умение отобразить себя на экране, полиморфизм */
};

class Rectangle : public Shape {  // конкретный примитив – прямоугольник
      int width, height;   // ширина и высота
  public:
      Rectangle(int _x = 0, _y = 0, _w = 0, _h= 0) {   /* координаты левого верхнего угла, ширина и высота */
            x = _x;        y = _y;
            width = _w; height = _h;
      }
      virtual void show() {   // переопределение метода базового класса
            cout<<"Rectangle: ("<<x<<','<<y<<','
                <<width<<','<<height<<')'<<endl;
      }
};

class Circle : public Shape {  // конкретный примитив – окружность
      int r;   // радиус
  public:
      Circle(int _x = 0, _y = 0, _r = 0) {  /* координаты левого верхнего угла, радиус */
```

```
        x = _x;        y = _y;
        r = _r;
    }
    virtual void show() {   // переопределение метода базового класса
        cout<<" Circle: ("<<x<<','<<y<<','
            <<r<<')'<<endl;
    }
};

int main() {
    int count = 3;
    Shape *figure[] = {   // создаем рисунок
        new Circle(100,100,80),
        new Rectangle(120,120,40,30),
        new Circle(150,170,30)
    };
    // Вывод рисунка на экран
    for(int i = 0; i < count; ++i )
        figure[i]->show();
    // Удаление примитивов из памяти
    for(int i = 0; i < count; ++i )
        delete figure[i];
    return 0;
}
```

V. Абстрактный класс как корень дерева наследования.

Класс Shape позволяет организовать иерархию наследования, но сам представляет некоторое абстрактное понятие. Не имеет смысла ни отображение на экране, ни даже само создание объектов этого класса. Поэтому класс Shape должен быть абстрактным классом. По определению, нельзя создавать объекты

абстрактного класса. Чтобы класс в C++ стал абстрактным, в нем должна быть объявлена хотя бы одна чистая виртуальная функция. В нашем примере такой функцией должен быть метод show(). Другие классы иерархии никак не изменятся, а объявление класса Shape будет выглядеть так:

```
class Shape {
  protected:
      int x, y;
  public:
      virtual void show() = 0;  /* чистая виртуальная функция */
};
```

IV. Задания.

1. Измените реализацию класса List из предыдущей лабораторной работы так, чтобы каждый узел содержал только целое значение (не обязательно уникальное). Создайте методы для добавления нового элемента в конец списка и перед указанным элементом списка, считая, что узлы списка пронумерованы, начиная с нуля, метод удаления узла с заданным индексом, получения значения элемента с заданным индексом, проверки на пустоту списка.
2. Создайте новый класс Queue1 на основе класса List с помощью агрегирования.
3. Создайте новый класс Queue2 на основе класса List с помощью наследования.
4. Модифицируйте класс Menu из предыдущей лабораторной работы следующим образом. Строки, представляющие пункты меню должны сохраняться в соответствующем поле (массив строк). В классе должен быть задан метод, например, int addMenuItem(char *item), позволяющий добавить новый пункт меню и возвращающий номер этого пункта, метод, отображающий меню на экране и предлагающий пользователю сделать свой выбор (например, void processMenu()). Если выбор неправильный меню выводится снова. Если выбор

правильный, вызывается другой метод класса Menu, например, void processMenuItem(int itemNumber) через указатель this. Пункт меню Quit (выход) добавляется и обрабатывается автоматически (пользователь не должен задавать его явно). Создайте классы ListMenu и QueueMenu для работы соответственно со списками или очередями через меню, как классы, наследники Menu. Классы-наследники должны переопределять только метод void processMenuItem(int itemNumber), всю остальную функциональность они получают от базового класса.

5. (Только для повышенной оценки). Создать иерархию классов, представляющих плоские фигуры, как описано выше в разделах IV и V. При этом в базовом классе (Shape) должен задаваться цвет фигуры, координаты характерной точки, метод для отображения фигуры на экране (абстрактный), метод для вывода фигуры в файл (абстрактный). В файл выводится описание примитива, например, для прямоугольника, RECTANGLE 10 20 40 50 1 (координата верхнего левого угла прямоугольника, его ширина, высота и цвет, например, 0 – черный, 1 – красный, 2 – зеленый, 3 – синий, 15 - белый). Предусмотреть, по крайней мере, три конкретных примитива: отрезок прямой (Line), прямоугольник (Rectangle), эллипс (Oval). Для замкнутых фигур должно быть два варианта (типа), незаполненные и заполненные цветом (например, окружность и круг).

6. (Только для повышенной оценки). Создайте класс Picture, представляющий понятие рисунок, составленный из двумерных примитивов. В классе должны быть реализованы методы для добавления примитива к рисунку, отображения рисунка на экране, записи рисунка в файл, чтения рисунка из файла. Протестируйте работоспособность классов из заданий 5 и 6.

7. (Только для повышенной оценки и если есть желание). Используйте классы и описание из папки appendix для графического отображения объектов класса.

V. Контрольные вопросы.

1. К каким полям, унаследованным от базового класса, нет доступа в производном классе?
2. Какие поля производного класса обязательно нужно инициализировать с помощью конструктора базового класса?
3. На что влияет вид наследования?
4. Как восстановить уровень доступа к компонентам базового класса для пользователей производного класса, если наследование закрытое?
5. Почему в языках программирования Java и C# наследование может быть только открытым?
6. Как отличить единичное наследование от множественного?
7. Могут ли при множественном наследовании отличаться виды наследования для разных базовых классов?
8. Как организовать ромбовидное наследование?
9. Чем удобны полиморфные классы?
10. Какие задачи помогает решать система родственных классов?
11. Можно ли получить полиморфное поведение, используя объекты классов? Указатели на объекты? Ссылки на объекты?
12. Что такое чистая виртуальная функция?
13. Что такое абстрактный класс?
14. Как в C++ объявить абстрактный класс?
15. В чем смысл использования абстрактного класса в качестве корня иерархии наследования?

Литература

1. Стэнли Б. Липпман. "Язык программирования C++. Вводный курс". – М.: Вильямс. – 2007, 896 с.

2. Г. Шилдт. "Самоучитель C++". – СПб.: БХВ-Петербург. – 2001, 688 с.

3. Стивен Пратта. "Язык программирования C++. Лекции и упражнения", 6-е изд. – М.: Вильямс. – 2012, 1248 с.