

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Наследование в языке C++

ВОПРОСЫ

- ⊙ Какой основной метод борьбы со сложностью в программировании?
- ⊙ Какие виды декомпозиции вы знаете?
- ⊙ Каким образом можно получить доступ к элементам объекта?
- ⊙ Каким образом можно получить доступ к статическим элементам класса?
- ⊙ Какие операции применимы к стандартным типам C++:
 - ⊙ $(a = b) = c;$
 - ⊙ $a = b = c = 5;$
 - ⊙ $(a + b) = c;$

НАСЛЕДОВАНИЕ

НАСЛЕДОВАНИЕ

- ⊙ Наследование упорядочивает и ранжирует классы
- ⊙ Механизм наследования классов позволяет строить иерархии, в которых производные классы получают элементы базовых классов и могут дополнять их или изменять их свойства
- ⊙ Классы, находящиеся ближе к началу иерархии, объединяют в себе наиболее общие черты для всех нижележащих классов
- ⊙ Задача: сформировать иерархию наследования, содержащую классы треугольника, четырехугольника, круга, овала, прямоугольника, квадрата

ВИДЫ НАСЛЕДОВАНИЯ

- ⊙ При описании класса в его заголовке перечисляются все классы, являющиеся для него базовыми.
- ⊙ Возможность обращения к элементам этих классов регулируется с помощью модификаторов наследования `private`, `protected` и `public`.

```
class B: public A { /*...*/ };
```

- ⊙ Если базовых классов несколько, то они перечисляются через запятую
- ⊙ По умолчанию для классов установлен модификатор `private`, а для структур - `public`.

PUBLIC-НАСЛЕДОВАНИЕ

- ⊙ `public`: открытое наследование

```
class B: public A { /*...*/ };
```

- ⊙ класс **B** является наследником **A**. Это также означает, что любой объект типа **B** также *является* объектом (разновидностью) типа **A** (но не наоборот!)
- ⊙ `protected` и `public` элементы базового класса остаются, соответственно, `protected` и `public` в классе-наследнике
- ⊙ Мы можем использовать указатель на базовый класс **A** для инициализации объекта класса-наследника **B**:
A *some = new B; // все хорошо: A – базовый класс,
// произойдет неявное преобразование

```

// ---- Класс monster ----
class monster
{
// ----- Скрытые поля класса:
private:
    int health, ammo;
    color skin;
    char *name;
public:
// ----- Конструкторы:
monster(int he = 100, int am = 10);
monster(color sk);
monster(char * nam);
monster(monster &M);
// ----- Деструктор:
~monster() {delete [] name;}
// ----- Операции:
...
int get_health()const
int get_ammo()const

void set_health(int he)
void draw(int x, int y,
          int scale, int position);
};

```

```

// ----- Класс daemon -----
class daemon : public monster
{
private:
    int brain;

public:
// ----- Конструкторы:
daemon(int br = 10){brain = br;};
daemon(color sk) : monster (sk) {brain = 10;}
daemon(char * nam) : monster (nam) {brain = 10;}
daemon(daemon &M) : monster (M) {brain = M.brain;}
...
void draw(int x, int y, int scale, int position);
void think();
};

```

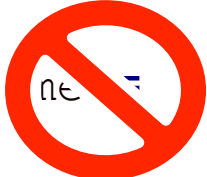
PRIVATE-НАСЛЕДОВАНИЕ

- ⊙ `private`: закрытое наследование

```
class B: private A { /*...*/ };
```

- ⊙ Таким образом, мы говорим, что класс **B** *реализован при помощи* класса **A**, но при этом он не является разновидностью этого класса.
- ⊙ Т.е. на этапе проектирования `private`-наследование никоим образом не фигурирует, а появляется только на этапе написания программного кода.

A *s ~~не~~ =w B;



PRIVATE-НАСЛЕДОВАНИЕ

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

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

class Derived : private Base{
    public: using Base::f;
};

int main(void) {
    Derived *d = new Derived;
    d->f();
}
```

PROTECTED-НАСЛЕДОВАНИЕ

- ⊙ `protected`: защищенное наследование

```
class B: protected A { /*...*/ };
```

- ⊙ Применяется редко. Означает что все `public` элементы базового класса становятся `protected` элементами дочернего класса.

ПРАВИЛА НАСЛЕДОВАНИЯ МЕТОДОВ

КОНСТРУКТОРЫ

- ⊙ *Конструкторы* не наследуются, поэтому производный класс должен иметь собственные конструкторы.
- ⊙ Если в конструкторе производного класса явный вызов конструктора базового класса отсутствует, автоматически вызывается конструктор базового класса по умолчанию
- ⊙ Для иерархии, состоящей из нескольких уровней, конструкторы базовых классов вызываются начиная с самого верхнего уровня.
- ⊙ В случае нескольких базовых классов их конструкторы вызываются в порядке объявления.
- ⊙ Если конструктор базового класса требует указания параметров, он должен быть явным образом вызван в конструкторе производного класса в списке инициализации.

ПРИМЕР НАСЛЕДОВАНИЯ

```
// ----- Класс monster -----  
class monster  
{  
  //...  
};  
  
// ----- Класс daemon -----  
class daemon : public monster  
{  
  int brain;  
  public:  
  // ----- Конструкторы:  
  daemon(int br = 10) {brain = br;};  
  daemon(color sk) : monster (sk) {brain = 10;}  
  daemon(char * nam) : monster (nam) {brain = 10;}  
  daemon(daemon &M) : monster (M) {brain = M.brain;}  
  void think();  
};  
  
void daemon::think(){ /* ... */ }
```

ОПЕРАЦИЯ ПРИСВАИВАНИЯ

- ⊙ *Операция присваивания не наследуется, поэтому ее также требуется явно определить в дочернем классе. При ее реализации можно явно вызывать функции-операции присваивания из базового класса.*
- ⊙ Вызов функций базового класса обычно предпочтительнее копирования фрагментов кода из функций базового класса в функции производного.

```
daemon& operator = (daemon &M) {  
    if (&M == this) return *this;  
    brain = M.brain;  
    monster::operator = (M);  
    return *this;  
}
```

ДЕСТРУКТОРЫ

- ⊙ *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

НАСЛЕДОВАНИЕ МЕТОДОВ

- ⊙ Класс-потомок наследует все методы базового класса, кроме конструкторов, деструктора и операции присваивания. Не наследуются ни дружественные функции, ни дружественные отношения классов.
- ⊙ В классе-наследнике можно определять *новые методы*. В них разрешается вызывать любые доступные методы базового класса.
- ⊙ Если имя метода в наследнике совпадает с именем метода базового класса, то метод производного класса скрывает все методы базового класса с таким именем.

ВИРТУАЛЬНЫЕ МЕТОДЫ

НАСЛЕДОВАНИЕ И УКАЗАТЕЛИ

```
monster *p;  
p = new daemon;  
p -> draw(1, 1, 1, 1);
```

- ⊙ Какой метод будет вызван:
 - ⊙ `monster::draw` или `daemon::draw` ?
 - ⊙ Ссылки на методы разрешаются во время компоновки программы. Этот процесс называется *ранним связыванием*.



VIRTUAL - РЕАЛИЗАЦИЯ МЕХАНИЗМА ПОЗДНЕГО СВЯЗЫВАНИЯ

- ⊙ В C++ реализован механизм *позднего связывания*, когда разрешение ссылок на функцию происходит на этапе выполнения программы.
- ⊙ Этот механизм реализован с помощью виртуальных методов.
- ⊙ Для определения *виртуального метода* используется спецификатор `virtual`:

```
virtual void draw(int x, int y, int position);
```

ПРАВИЛА ИСПОЛЬЗОВАНИЯ ВИРТУАЛЬНЫХ МЕТОДОВ

- ⊙ Если в базовом классе метод определен как виртуальный, метод, определенный в производном классе *с тем же именем и набором параметров*, автоматически становится виртуальным.
- ⊙ Виртуальные методы *наследуются*.
- ⊙ Если виртуальный метод переопределен в производном классе, объекты этого класса могут получить доступ к методу базового класса с помощью операции доступа к области видимости.
- ⊙ Виртуальный метод не может объявляться с модификатором `static`, но может быть объявлен как дружественная функция.
- ⊙ Если производный класс содержит виртуальные методы, они должны быть определены в базовом классе хотя бы как чисто виртуальные.

ЧИСТО ВИРТУАЛЬНЫЙ МЕТОД, АБСТРАКТНЫЕ КЛАССЫ

- ◎ Чисто виртуальный метод содержит признак = 0 вместо тела, например:

```
virtual void f(int) = 0;
```

- ◎ Чисто виртуальный метод должен переопределяться в производном классе (возможно, опять как чисто виртуальный).
- ◎ Класс, содержащий хотя бы один чисто виртуальный метод, называется *абстрактным*.
- ◎ Абстрактный класс может использоваться *только в качестве базового* для других классов - **объекты абстрактного класса создавать нельзя**, поскольку прямой или косвенный вызов чисто виртуального метода приводит к ошибке при выполнении.

МЕХАНИЗМ ПОЗДНЕГО СВЯЗЫВАНИЯ

- ⊙ Для каждого класса (не объекта!), содержащего хотя бы один виртуальный метод, компилятор создает *таблицу виртуальных методов* (`vtbl`), в которой для каждого виртуального метода записан его адрес в памяти.
- ⊙ Каждый объект содержит скрытое дополнительное *поле ссылки* на `vtbl`, называемое `vptr`.
- ⊙ На этапе компиляции ссылки на виртуальные методы заменяются на обращения к `vtbl` через `vptr` объекта.

ДЕСТРУКТОРЫ И НАСЛЕДОВАНИЕ

- ⊙ Деструкторы следует объявлять виртуальными тогда, когда в классе есть хотя бы одна виртуальная функция.
- ⊙ В стандарте C++ указывается, что *«когда объект производного класса уничтожается через указатель на базовый класс с невиртуальным деструктором, то результат не определен»*.
- ⊙ Во время исполнения это обычно приводит к тому, что часть объекта, принадлежащая производному классу, никогда не будет уничтожена.
- ⊙ Если же класс не имеет виртуальных функций, это часто означает, что он не предназначен быть базовым.

- ◎ `public` наследование – «В **является** А»
- ◎ `private` наследование – «В **реализуется посредством** А»
- ◎ Класс-потомок наследует все методы базового класса, кроме конструкторов, деструктора и операции присваивания. Не наследуются ни дружественные функции, ни дружественные отношения классов.
- ◎ Виртуальные функции позволяют реализовать механизм позднего связывания в C++
- ◎ Абстрактный класс – класс, содержащий хотя бы 1 чисто виртуальный метод
- ◎ Если класс содержит хотя бы один виртуальный метод – необходимо реализовать виртуальный деструктор