Т.2.7.1. Принципы ООП. Определение классов и их методов.

Образец подзаголовка

Основные принципы ООП

Объектно-ориентированное программирование основано на «трех китах» - трех важнейших принципах, придающих объектам новые свойства. Этими принципами являются **инкапсуляция, наследование и полиморфизм.**

Инкапсуляция

Инкапсуляция есть объединение в единое целое данных и алгоритмов обработки этих данных.

Инкапсуляция позволяет в максимальной степени изолировать объект от внешнего окружения. Она существенно повышает надежность разрабатываемых программ, т.к. локализованные в объекте алгоритмы обмениваются с программой сравнительно небольшими объемами данных, причем количество и тип этих данных обычно тщательно контролируются. В результате замена или модификация алгоритмов и данных, инкапсулированных в объект, как правило, не влечет за собой плохо прослеживаемых последствий для программы в целом (в целях повышения защищенности программ в ООП почти не используются глобальные переменные).

Наследование

Наследование есть свойство объектов порождать своих потомков. Объект-потомок автоматически наследует от родителя все поля и методы, может дополнять объекты новыми полями и заменять (перекрывать) методы родителя или дополнять их.

Принцип наследования решает проблему модификации свойств объекта и придает ООП в целом исключительную гибкость. При работе с объектами программист обычно подбирает объект, наиболее близкий по своим свойствам для решения конкретной задачи, и создает одного или нескольких потомков от него, которые «умеют» делать то, что не реализовано в родителе.

Полиморфизм

Полиморфизм - это свойство родственных объектов (т.е. объектов, имеющих одного общего родителя) решать схожие по смыслу проблемы разными способами.

В рамках ООП поведенческие свойства объекта определяются набором входящих в него методов. Изменяя алгоритм того или иного метода в потомках объекта, программист может придавать этим потомкам отсутствующие у родителя специфические свойства. Для изменения метода необходимо перекрыть его в потомке, т.е. объявить в потомке одноименный метод и реализовать в нем нужные действия. В результате в объекте-родителе и объекте-потомке будут действовать два одноименных метода, имеющие разную алгоритмическую основу и, следовательно, придающие объектам разные свойства. Это и называется полиморфизмом объектов.

В C++ полиморфизм достигается не только описанным выше механизмом наследования и перекрытия методов родителя, но и их

Классы

Класс есть тип.

Класс – это некоторая идея еще не существующего объекта, в которой воедино собраны все детали, все свойства и все нужные действия необходимые для этого объекта

Классы в программировании состоят из **свойств и методов.**

Свойства — это любые данные, которыми можно характеризовать объект класса.

Методы — это функции, которые могут выполнять какие-либо действия над данными (свойствами) класса.

Методы класса — это его функции.

Свойства класса — его переменные.

Объект это экземпляр класса.

Классы

```
class Car
{ auto mark; // свойство марка
 auto model; // свойство модель
 auto color; // свойство цвет
void go(); // метод ехать
void turn(); // метод повернуть
void stop(); // метод остановиться };
```

Поля (атбибуты) класса:

- могут быть простыми переменными любого типа, указателями, массивами и ссылками (т.е. могут иметь практически любой тип, кроме типа этого же класса, но могут быть указателями или ссылками на этот класс);
- могут быть константами (описаны с модификатором *const*), при этом они инициализируются только один раз (с помощью *конструктора*) и не могут изменяться;

Инициализация полей при описании не допускается.

Доступ к членам класса

Все свойства и методы классов имеют права доступа. По умолчанию, все содержимое класса является доступным для чтения и записи только для него самого.

Закрытые данные класса размещаются после модификатора доступа **private.** Если отсутствует модификатор public, то все функции и переменные, по умолчанию являются закрытыми.

Для того, чтобы разрешить доступ к данным класса извне, используют модификатор доступа **public.** Все функции и переменные, которые находятся после модификатора public, становятся доступными из всех частей программы.

То что описано внутри public доступно для всей программы, а то что написано внутри private доступно только внутри класса

Protected — доступ открыт классам, производным от данного.

Описание функций

Компонентные функции или методы могут быть описаны как внутри, так и вне определения класса.

В последнем случае определение класса должно содержать прототипы этих функций, а заголовок описываемой функции должен включать квалификатор видимости, который состоит из имени класса и знака «::».

Таким образом, компилятору сообщается, что определяемой функции доступны внутренние поля класса: :: () { }

Описание функций внутри класса

```
class First
{ public:
char c;
int x,y;
void print(void)
 { cout<<c<<x<<y; }
void set(char ach, int ax, int ay)
{ c=ach; x=ax; y=ay; }
};
```

Описание функций вне класса

```
class First
 { public:
char c;
int x,y;
 void print(void);
 void set(char ach,int ax,int ay); };
/* компонентные функции, описанные вне класса */
 void First::print(void)
 {cout<<c<<x<<y; }
 void First::set (char ach,int ax,int ay)
{ c=ach; x=ax; y=ay; }
```

Обращение к общедоступным полям и методам объекта

Например:

a.First::set('A',3,4); // статический объект

b->First::set('B',3,4); // динамический объект

c[i].First::set('C',3,4); // массив объектов

Однако обычно доступ к компонентам объекта обеспечивается с помощью укороченного имени, в котором квалификатор доступа опущен, тогда принадлежность к классу определяется по типу объекта: . -> [].

Например:

a.set('A',3,4)

b->set('B',3,4)

c[i].set('C',3,4)

Пример

Например:

a.First::set('A',3,4); // статический объект

b->First::set('B',3,4); // динамический объект

c[i].First::set('C',3,4); // массив объектов

Однако обычно доступ к компонентам объекта обеспечивается с помощью укороченного имени, в котором квалификатор доступа опущен, тогда принадлежность к классу определяется по типу объекта: . -> [].

Например:

a.set('A',3,4)

b->set('B',3,4)

c[i].set('C',3,4)

Создание класса комплексных чисел

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

class Complex
{public:
 int real;
 int image;
 void show();
};

void Complex::show()
{
   cout<<real<<" + "<<image<<"  *i "<<endl;
}
```

Создание класса комплексных чисел

```
int main()
{
   Complex a;
   a.real=-6;
   a.image=8;
   a.show();
    return 0;
}
```

Добавить функцию сложения

```
class Complex
{public:
 int real;
 int image;
 void add(Complex x);
 void show();
};

void Complex::add(Complex x)
{real=real+x.real;
 image=image+x.image;
}
```

Добавить функцию сложения

```
int main()
{
   Complex a,b;
   a.real=-6;
   a.image=8;
   b.real=5;
   b.image=90;
   a.show();
   b.show();
    a.add(b);
   a.show();
    return 0;
```