**Класс** – это абстрактный тип данных. С помощью класса описывается некоторая сущность (ее характеристики и возможные действия). Например, класс может описывать студента, автомобиль и т.д. Описав класс, мы можем создать его экземпляр –**объект**. Объект – это уже конкретный представитель класса.

**Инкапсуляция** – позволяет скрывать внутреннюю реализацию. В классе могут быть реализованы внутренние вспомогательные методы, поля, к которым доступ для пользователя необходимо запретить, тут и используется инкапсуляция. Больше об инкапсуляции читайте в уроке [Инкапсуляция в Си-шарп. Модификаторы доступа](http://mycsharp.ru/post/38/2014_02_23_inkapsulyaciya_v_si-sharp_modifikatory_dostupa.html).  
  
**Наследования** – позволяет создавать новый класс на базе другого. Класс, на базе которого создается новый класс, называется базовым, а базирующийся новый класс – наследником. Например, есть базовый класс животное. В нем описаны общие характеристики для всех животных (класс животного, вес). На базе этого класса можно создать классы наследники Собака, Слон со своими специфическими свойствами. Все свойства и методы базового класса при наследовании переходят в класс наследник. Больше о наследовании читайте в уроке [Наследование в Си-шарп](http://mycsharp.ru/post/28/2013_07_21_nasledovanie_v_si-sharp_konstruktor_bazovogo_klassa.html).  
  
**Полиморфизм** – это способность объектов с одним интерфейсом иметь различную реализацию. Например, есть два класса, Круг и Квадрат. У обоих классов есть метод GetSquare(), который считает и возвращает площадь. Но площадь круга и квадрата вычисляется по-разному, соответственно, реализация одного и того же метода различная. Больше о полиморфизме читайте в уроке [Полиморфизм в Си-шарп](http://mycsharp.ru/post/32/2013_08_27_polimorfizm_v_si-sharp_chto_eto_takoe_.html).

**Абстракция** – позволяет выделять из некоторой сущности только необходимые характеристики и методы, которые в полной мере (для поставленной задачи) описывают объект. Например, создавая класс для описания студента, мы выделяем только необходимые его характеристики, такие как ФИО, номер зачетной книжки, группа. Здесь нет смысла добавлять поле вес или имя его кота/собаки и т.д.

Конструктор копирования нужен нам для того, чтобы создавать «реальные» копии объектов класса, а не побитовую копию объекта. Иногда это принципиально важно. Такую «реальную» копию объекта надо создавать в нескольких случаях:

* когда мы передаем объект в какую-либо функцию в виде параметра;
* когда какая-либо функция должна вернуть объект класса в результате своей работы;
* когда мы в главной функции один объект класса инициализируем другим объектом класса.
* имяКласса (const имяКласса & object)
* {
* //код конструктора копирования
* }

Перегрузка операторов

С++ поддерживает перегрузку операторов (operator overloading). За небольшими исключениями большинство операторов С++ могут быть перегружены, в результате чего они получат специаль­ное значение по отношению к определенным классам. Например, класс, определяющий связан­ный список, может использовать оператор + для того, чтобы добавлять объект к списку. Другой класс может использовать оператор + совершенно иным способом. Когда оператор перегружен, ни одно из его исходных значений не теряет смысла. Просто для определенного класса объектов определен новый оператор. Поэтому перегрузка оператора + для того, чтобы обрабатывать свя­занный список, не изменяет его действия по отношению к целым числам.

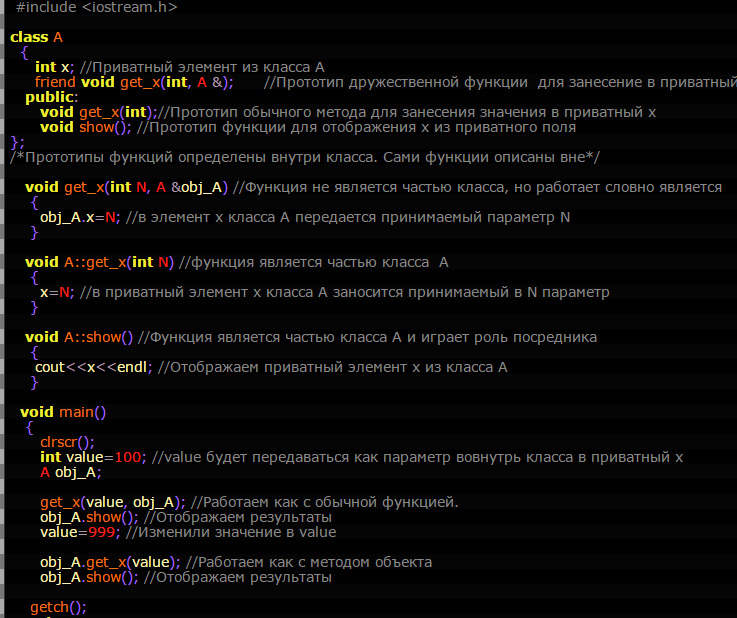
тип имя\_класса::operator#(список\_аргументов){// действия, определенные применительно к классу}

const Integer operator+(const Integer& rv) const { return (value + rv.value); }

Дружественая функция

Дружественная функция – это функция, которая не являясь частью класса имеет доступ ко всем элементам из дружественного себе класса.

Дружественная функция объявляется внутри класса с модификатором friend



Виртуальные функции

Полиморфизм времени исполнения обеспечивается за счет использования производных классов и виртуальных функций. Виртуальная функция — это функция, объявленная с ключевым словом virtual в базовом классе и переопределенная в одном или в нескольких производных классах. Виртуальные функции являются особыми функциями, потому что при вызове объекта производ­ного класса с помощью указателя или ссылки на него С++ определяет во время исполнения про­граммы, какую функцию вызвать, основываясь на типе объекта. Для разных объектов вызываются разные версии одной и той же виртуальной функции. Класс, содержащий одну или более вир­туальных функций, называется полиморфным классом (polymorphic class).

Виртуальная функция объявляется в базовом классе с использованием ключевого слова virtual. Когда же она переопределяется в производном классе, повторять ключевое слово virtual нет не­обходимости, хотя и в случае его повторного использования ошибки не возникнет.

В качестве первого примера виртуальной функции рассмотрим следующую короткую программу:

#include <iostream.h>  
class Base {  
public:  
virtual void who() { // определение виртуальной функции  
cout << \*Base\n";  
}  
};  
class first\_d: public Base {  
public:  
void who() { // определение who() применительно к first\_d  
cout << "First derivation\n";  
}  
};  
class seconded: public Base {  
public:  
void who() { // определение who() применительно к second\_d  
cout << "Second derivation\n\*";  
}  
};  
int main()  
{  
Base base\_obj;  
Base \*p;  
first\_d first\_obj;  
second\_d second\_obj;  
p = &base\_obj;  
p->who(); // доступ к who класса Base  
p = &first\_obj;  
p->who(); // доступ к who класса first\_d  
p = &second\_ob;  
p->who(); // доступ к who класса second\_d  
return 0;  
}