Класс

Класс определяет абстракцию существующего объекта. Объект существует в течение некоторого времени, а класс не существует (условно). **Класс** - это некое множество объектов, имеющих общую структуру и общее поведение. **Класс** - это структурный вид данных, который включает в себя описание полей и функций, названные методами. **Класс** - это пользовательский тип данных

```
class Window{
    private:
        int x1, y1, x2, y2;
    public:
        Window(int, int, int, int); //Конструктор
        draw();                     //Рисование
}
```

```
    class FileStat{
        File *f;
        int status = 0,1,2;
    public:
        FileStat(char *s);
        openf();
        search(char *s);
        addtofile(record r);
        delfromfile(record r);
}

FileStatus *fs = new FileStatus("filename.txt");
if (fs->status){
    ...
}
```

Отличие метода от функции в том, что первый неявный параметр метода - объект, метод которого вызывается.

Конструкторы можно перегрузить как и любой метод в языке

```
class data{
    int y;
    int m;
    int d;
public:
data(); // Внутри взять текущее системное время
data(int); // Один параметр - год
data(int,int); // Два параметра - год, месяц
```

Сигнатура метода - имя и перечень входных параметров.

Создать объект можно тремя способами.

1. Вызов его конструктора `data d()`
2. Выделением памяти под указатель и вызовом конструктора `data *d = new data()`
3. Присваивание `data d = d1;`

```
class A{
    int a;
public:
    A(int a1):a(a1){} // Инициализация параметра a, до создания объекта
```

```
class A{
    int a;
public:
    A(int a1){
        a = a1; // Присваивается после выделения памяти под объект
    }
}
```

Во втором случае конструктор вызывается два раза. Если некоторое поле задано константой, то возможна только инициализация в конструкторе.

3-й способ инициализации, добавлен в C++11 (Uniform)

```
class A{
    const int a;
public:
    A(int a1):a{a1}{}; // Uniform инициализация
}
```

Заголовок класса делится на три раздела (которые могут быть разнесены по классу). private описывает доступ для текущего класса и для всех friend функций и классов. protected описывает структуру, видимую самому классу, классу его наследников и friend классам и фукнциям public - раздел, видимый и доступныый всем без ограничений. public является интерфейсом для остальной программы.

```
class Window{
    private:
        int x1, y1; // Имплементация (и одновременно интерфейс для friend'ов)
        int x2, y2; // Имплементация
    protected:
        int color;  // Это будет интерфейсом для наследников
    public:
        Window(); // Интерфейс
        draw();   // Интерфейс
}
```

Виды отношений между классами.

1. Ассоциация. Один тип объектов ассоциируем с другим. Самый слабый вид отношения. В обоих объектах имеются указатели на ассоциируемый объект.
2. Наследование. Механизм, позволяющий создавать новые классы на основе других(родителей)
3.

```
class Unit{
int hp;
public
Unit():hp(100){};
int live(){this->hp--;}
}
class Warrior: public Unit{
int damage;
public:
Warrior():Unit(), damage(10){}; // Необходимо вызвать конструктор родственника, т.к. унаследовавший класс не имеет доступ к hp
int step(int, int);
}
```

тип наследования	поле родителя	поле наследника
public	public	public
public	protected	protected
public	private	private
protected	public	protected
protected	protected	protected
protected	private	private
private	public	private
private	protected	private
private	private	private

3. Множественное наследование
4.

```
class Technic{
int w;
public:
Technic():w(wei){};
}
```

```
class Tank: public Unit, public Technic{...}
```

4. Агрегация. Агрегация между класса связана непосредственно с агрегацией между объектами.

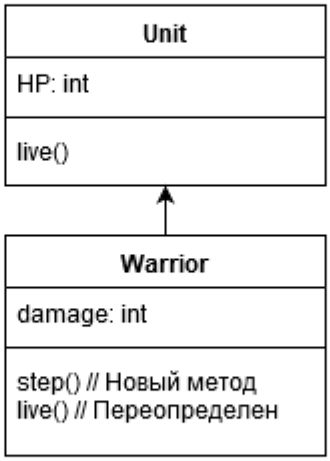
Агрегация = включение.Как правило агрегация односторонняя. Агрегация - включение по ссылке, т.е., один объект может существовать без другого. Пример - самолет без пассажира может быть, без двигателя не может.

5. Композиция. Физическое включение - один объект не может существовать без другого.

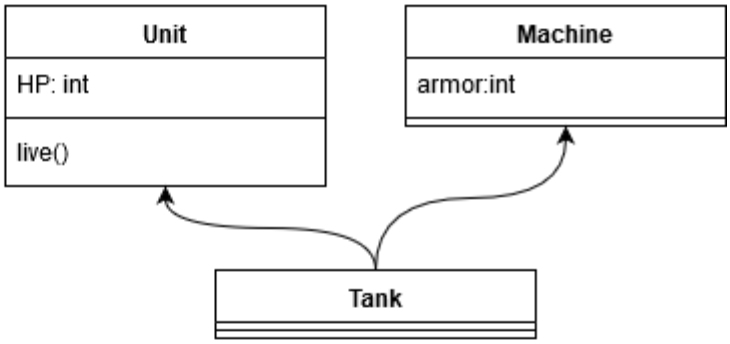
6. Зависимость (использование).

1. Ассоциация

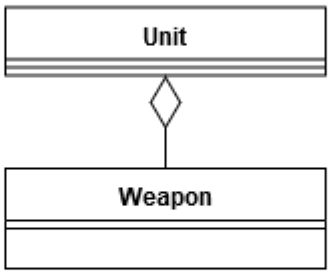
2. Наследование



3. Множественное наследование



4. Агрегация



5. Композиция