Класс

Класс - некоторое множество обьектов имеющие некоторую общую структуру.

Класс - структурный тип данных, который включает в себя описание полей и функций (*методов*).

Класс - пользовательский тип данных.

```
class FileStruct{
    File *f
    int status;
  public:
      FileString(string);
      search(string);
      AddRecord(record);
      DelRecord(record);
      inOpened();
  }
```

```
 int FileStruct::opened(){
     this->f;;
    f;
```

Перегрузка операторов

```
 class data{
 private:
     int y;
     int m;
     int d;
 public:
     data();
    data(int);
    data(int,int);
    data(int,int,int);
```

Метод определяется не именем а параметрами. Создать метод можно несколькими способами:

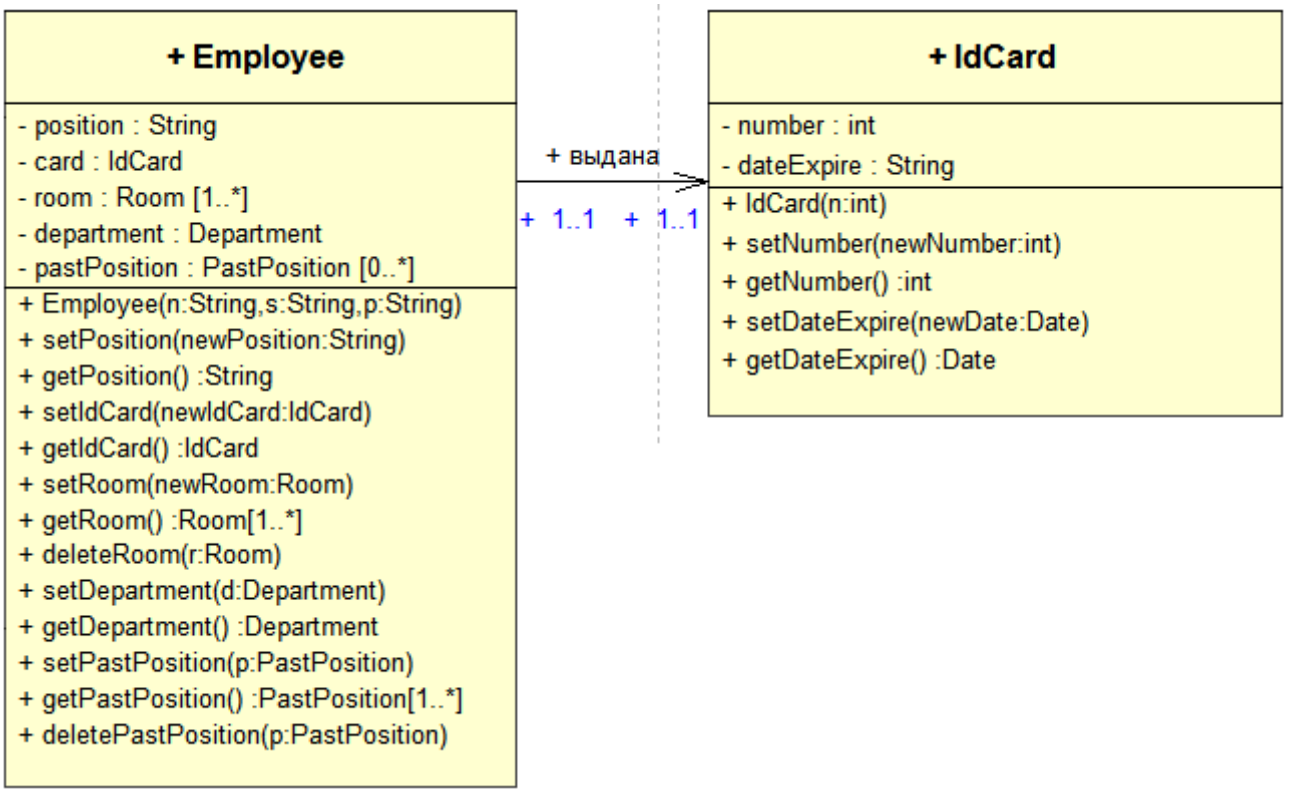
1. Вызвать его конструктор. `data d();`
2. Выделение памяти. `data *d=new(data);`
3. Приравнивание. `data dt=d;`

```
  class A{
   int a;
  public:
   A(int a1):a(a1){}
```

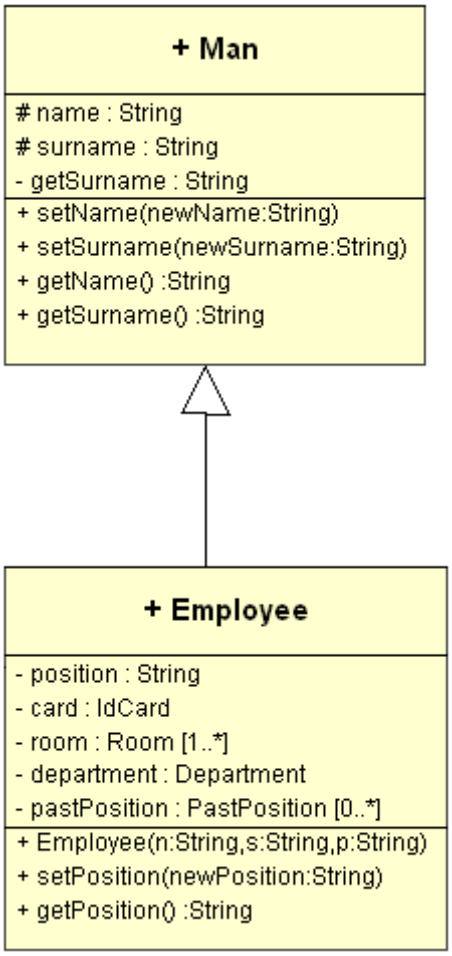
```
class A{
 int a;
public:
 A(int a1){
    a=a1;
}
```

Виды отношений между классами:

- **Ассоциация** - один тип объектов ассоциируем с другим.



- **Наследование** - механизм, позволяющий создавать новые классы на основе других(*родителей*).




```
class Unit{
  u_int hp;
public:
  Unit():hp(100){}
  live(){tis->hp--;}
}
```

Родитель

```
class Warior : public Unit{
  int dmg;
public:
  Warior():Unit(),dmg(10){};
}
```

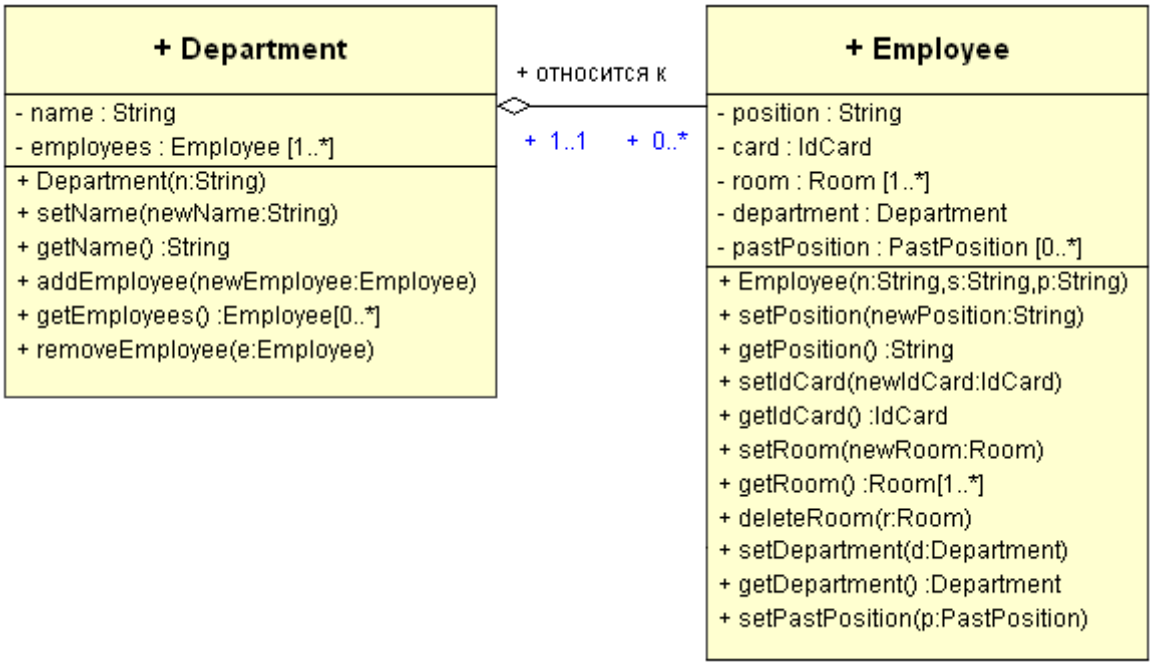
Наследник

Таблица видимости при наследовании.

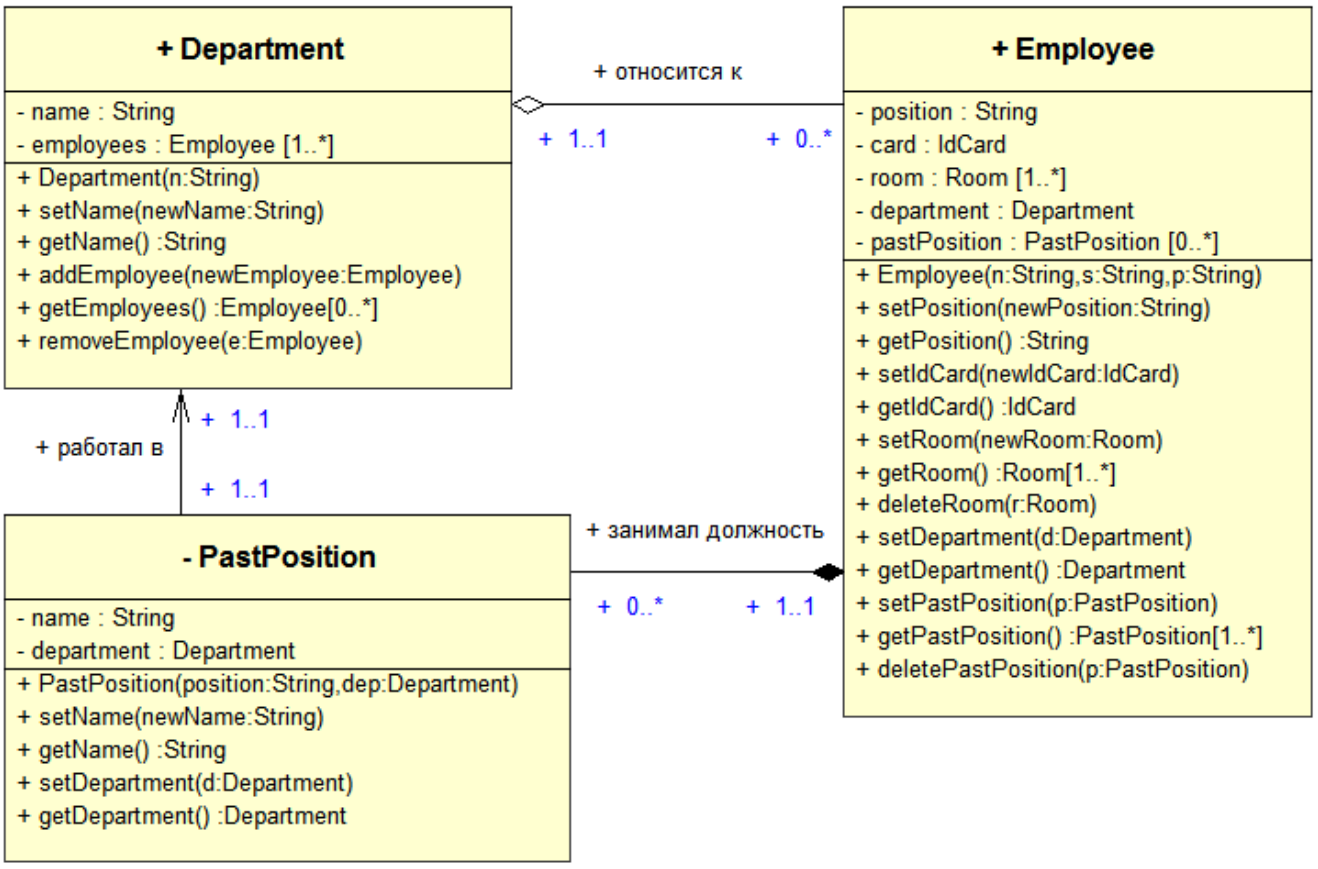
	Поле родителя	**Поле наследника**
public	• public • protected • private	• public • protected • private

protected	• public • protected • private	• protected • protected • private
private	• public • protected • private	• private • private • private

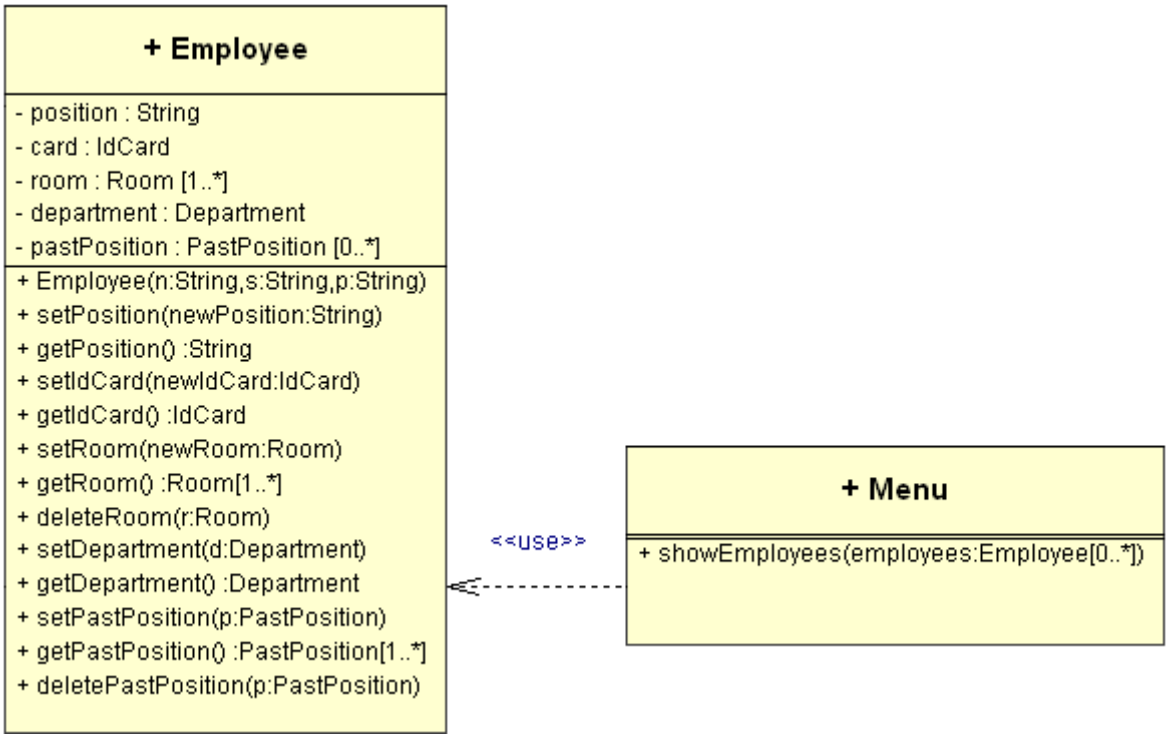
- **Агрегация** - включение, исключение включаемого элемента не влияет на исходный объект.



- **Композиция** - включение по ссылке, исходный объект не может существовать без включаемого.



- **Зависимость** (использование) - отношение между классами при которых 1 класс использует сервисы другого.




```
class Unit{
   int x,y;
  public:
       Unit()ix(1),y(1){};
       void print();
       static void printS(){}
}
```

```
class Field{
   public:
   void createUnit(Unit &a);
  Unit createUnit(){
      return new Unit();
  }
  void printUnit(){
      Unit.printS();
  }
}
```

- **Использование метаклассов** - использование класса как объекта.

Полиморфизм

Появление этого механизма объяснено желанием дать методам разные имена тем самым увеличив гибкость программы. Статический и реальный полиморфизм.

Полиморфными объектами называются объекты в процессе выполнения программы можно присвоить переменные тип которых отличается от предыдущих.

Полиморфизм (как механизм) - возможность задания различных реализаций некоторого единого по названию метода для классов различного уровня иерархии.

```
class A{
    int a;
    public:
        A(int a1),a(a1){};
        void print(){
            std:cout<< "print A";
        }
        void show(){
            std:cout<< "show A";
            print();
        }
}
```

```
class B : public A{
    int b;
    public:
        B(int b1,int a1),A(a1),b(b1){};
        void print(){
            std:cout<< "print B";
        }
}
```

```
class C : public B{
    int c;
    public:
        C(int c1,int tb1,int ta1),B(b1,a1),A(a1),c(c1){};
        void print(){
```

```
            std:cout<< "print C";
        }
}
```