Что мы узнали полезного?

1. https://metanit.com/sharp/patterns/1.2.php

Наследование

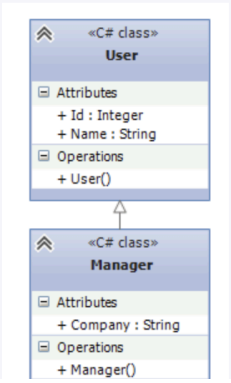
Наследование является базовым принципом ООП и позволяет одному классу (наследнику) унаследовать функционал другого класса (родительского). Нередко отношения наследования еще называют генерализацией или обобщением. Наследование определяет отношение **IS A**, то есть "является". Например:

```
class User
{
    public int Id { get; set; }
    public string Name { get; set; }
}

class Manager : User
{
    public string Company{ get; set; }
}
```

В данном случае используется наследование, а объекты класса Manager также **являются** и объектами класса User.

С помощью диаграмм UML отношение между классами выражается в незакрашенной стрелочке от класса-наследника к классу-родителю:

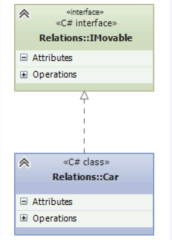


Реализация

Реализация предполагает определение интерфейса и его реализация в классах. Например, имеется интерфейс IMovable с методом Move, который реализуется в классе Car:

```
public interface IMovable
{
    void Move();
}
public class Car : IMovable
{
    public void Move()
    {
        Console.WriteLine("Машина едет");
    }
}
```

С помощью диаграмм UML отношение реализации также выражается в незакрашенной стрелочке от класса к интерфейсу, только линия теперь пунктирная:



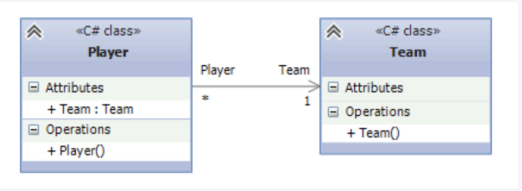
Ассоциация

Ассоциация - это отношение, при котором объекты одного типа неким образом связаны с объектами другого типа. Например, объект одного типа содержит или использует объект другого типа. Например, игрок играет в определенной команде:

```
class Team
{

}
class Player
{
    public Team Team { get; set; }
}
```

Класс Player связан отношением ассоциации с класом Team. На схемах UML ассоциация обозначается в виде обычно стрелки:



Композиция

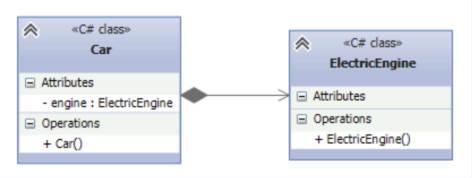
Композиция определяет отношение **HAS A**, то есть отношение "имеет". Например, в класс автомобиля содержит объект класса электрического двигателя:

```
public class ElectricEngine
{ }

public class Car
{
    ElectricEngine engine;
    public Car()
    {
        engine = new ElectricEngine();
    }
}
```

При этом класс автомобиля полностью управляет жизненным циклом объекта двигателя. При уничтожении объекта автомобиля в области памяти вместе с ним будет уничтожен и объект двигателя. И в этом плане объект автомобиля является главным, а объект двигателя - зависимой.

На диаграммах UML отношение композиции проявляется в обычной стрелке от главной сущности к зависимой, при этом со стороны главной сущности, которая содержит, объект второй сущности, располагается закрашенный ромбик:



Агрегация

От композиции следует отличать агрегацию. Она также предполагает отношение **HAS A**, но реализуется она иначе:

```
public abstract class Engine
{ }

public class Car
{
    Engine engine;
    public Car(Engine eng)
    {
        engine = eng;
    }
}
```

При агрегации реализуется слабая связь, то есть в данном случае объекты Car и Engine будут равноправны. В конструктор Car передается ссылка на уже имеющийся объект Engine. И, как правило, определяется ссылка не на конкретный класс, а на абстрактный класс или интерфейс, что увеличивает гибкость программы.

Отношение агрегации на диаграммах UML отображается также, как и отношение композиции, только теперь ромбик будет незакрашенным: