

# Семинар #1

Вспоминаем ООП

## Формула\*

$$\text{НАКОП\_М2} = 0,3 * \text{АУД\_М2} + 0,4 * \text{ДЗ\_М2} + 0,3 * \text{КР\_М2}$$

$$\text{НАКОП\_М3} = 0,4 * \text{АУД\_М3} + 0,6 * \text{ДЗ\_М3}$$

$$\text{НАКОП\_СР\_М2\_3} = (\text{НАКОП\_М2} + \text{НАКОП\_М3}) / 2$$

ПРОМ\_ЭКЗ\_М3 — оценка за промежуточный экзамен в модуле 3

$$\text{ПРОМ\_ОЦЕН\_М3} = 0,65 * \text{НАКОП\_СР\_М2\_3} + 0,35 * \text{ПРОМ\_ЭКЗ\_М3}$$

$$\text{НАКОП\_М4} = 0,4 * \text{АУД\_М4} + 0,6 * \text{ДЗ\_М4}$$

$$\text{СР\_НАКОП} = (\text{ПРОМ\_ОЦЕН\_М3} + \text{НАКОП\_М4}) / 2$$

$$\text{ФИН\_ОЦЕН} = 0,7 * \text{СР\_НАКОП} + 0,3 * \text{ФИН\_ЭКЗ\_М4}$$

ПРОМ\_ЭКЗ\_М3 и ФИН\_ЭКЗ\_М4 – блокирующие



## Формула\*\*

Если все три контрольные (большие ДЗ) были сданы, то студент получает оценку за экзамен «автоматом» и его оценка вычисляется по формуле (но право сдать экзамен сохраняется):

$$access\_estimation = round((0.15 * seminars\_estimation + 0.15 * homework\_estimation + 0.6 * tasks\_estimation) / 0.9)$$

Если контрольные не сданы (или студент решил пойти на экзамен), то оценка вычисляется по формуле:

$$result\_estimation = round(0.15 * seminars\_estimation + 0.15 * homework\_estimation + 0.6 * tasks\_estimation + 0.1 * exam\_estimation)$$

- *seminars\_estimation* – в конце семинаров будут даваться мини-тестики на 3-5 вопросов по материалам лекций/семинаров.
- *homework\_estimation* – это оценки за небольшие домашние задания (или доделку семинарских заданий) на закрепление материала. Принимаются они в течении недели после семинара на котором были выданы.
- *tasks\_estimation* – это те самые большие ДЗ. Вот тут будут заранее известные критерии и достаточно жесткая проверка на списывание. Всех людей мучить на коллоквиуме мы в этом году не станем, но претендующие на отл или авторы подозрительных работ будут их защищать.

## Формула (актуальная)

Автомат.

Нужно решить все ДЗ на  $\geq 4$ . Тогда итог рассчитывается с учетом приведения к единице:

$$\text{result\_estimation} = \text{math\_round} ( ( 0.1 * \text{seminars\_estimation} + 0.7 * \text{tasks\_estimation} ) / 0.8 )$$

Полуавтомат.

Если не все ДЗ сделаны, то оценка, вычисляется по формуле:

$$\text{low\_estimation} = ( 0.1 * \text{seminars\_estimation} + 0.7 * \text{tasks\_estimation} )$$

- `seminars_estimation` – математическое среднее всех оценок за семинары. В конце семинаров будут мини-тесты на 3-5 вопросов по материалам лекций/семинаров. Тест не сдан или дано 0 правильных ответов - 0,  $\leq$  половины правильных ответов - 5,  $>$  половины правильных ответов - 8, все ответы верны - 10.
- `tasks_estimation` – математическое среднее оценок за ДЗ. Тут будут заранее известные критерии и достаточно жесткая проверка на списывание. Претендующие на отл или авторы подозрительных работ будут их защищать. Примерная развесовка домашек: ДЗ-1 = 0.2, ДЗ-2 = 0.2, ДЗ-3 = 0.3, ДЗ-4 = 0.3

Итоговая оценка меньше 4 округляется вниз.

## Что можно использовать

- Visual Studio – оптимально для windows
- Rider (JetBrains) – оптимально для mac, linux
- VS Code – легковесный редактор

### Оптимально

- C#
- Java
- Kotlin

### Тоже можно

- Golang
- Python\*

- Версии языков используйте актуальные
- Нейросети использовать можно, но... Я бы подумал дважды :)

# ООП

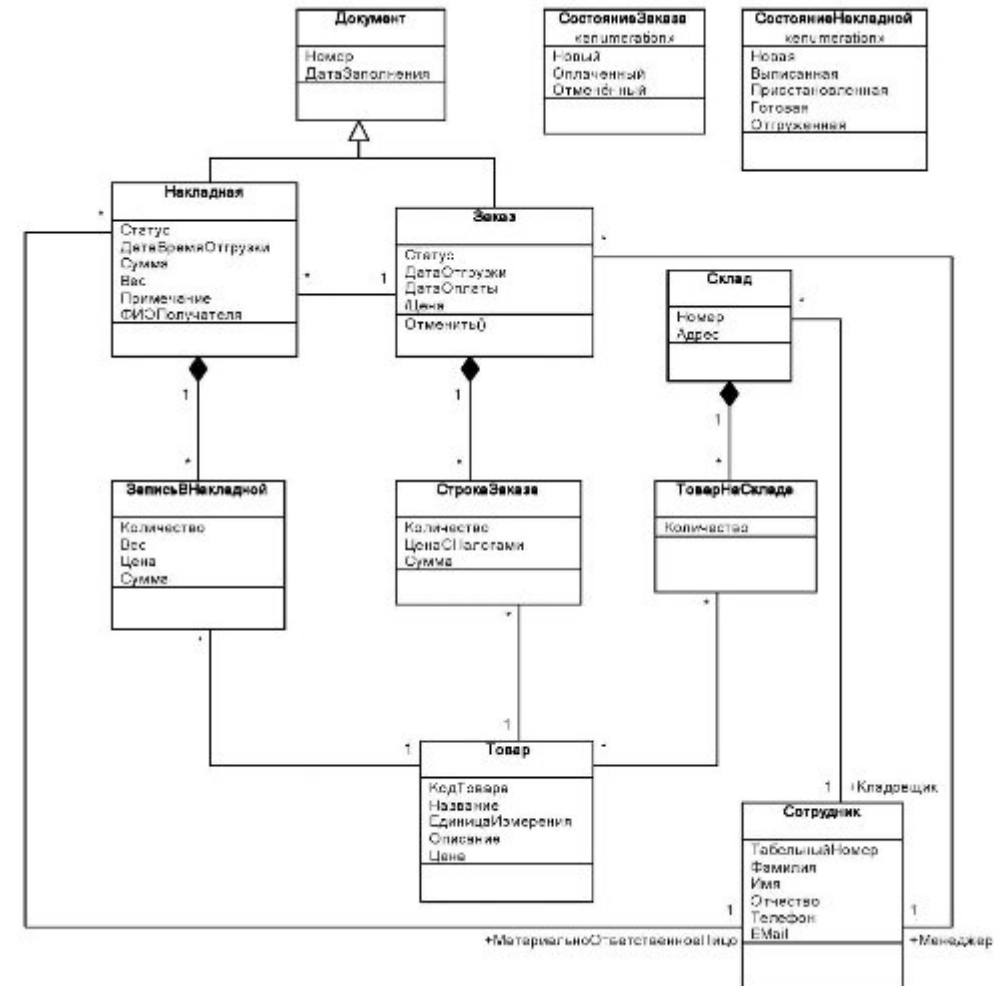
**Объектно-ориентированное программирование** - методология программирования, основанная на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования. Прежде чем начать писать инструкции для решения задачи, в задаче выделяются объекты и описываются с помощью классов. В классе прописывается поведение объектов с помощью методов и характеристики или свойства объекта с помощью переменных класса.

- **абстракция** - придание объекту характеристик, которые отличают его от всех объектов, четко определяя его концептуальные границы
- **инкапсуляция** - свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе.
- **наследование** - свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствованной функциональностью.
- **полиморфизм** - свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

# Диаграмма классов

Диаграмма классов (class diagram) предназначена для представления внутренней структуры объектно-ориентированных систем в виде классов и связей между ними.

Все сущности реального мира, с которыми собирается работать программист, должны быть представлены объектами классов в программе. При этом у каждого класса должно быть только одно назначение и уникально осмысленное имя, которое будет связано с этой целью.



# Структура класса

## 1. Модификаторы доступа:

- Private (-)
- Public (+)
- Protected (#)
- Internal (~)

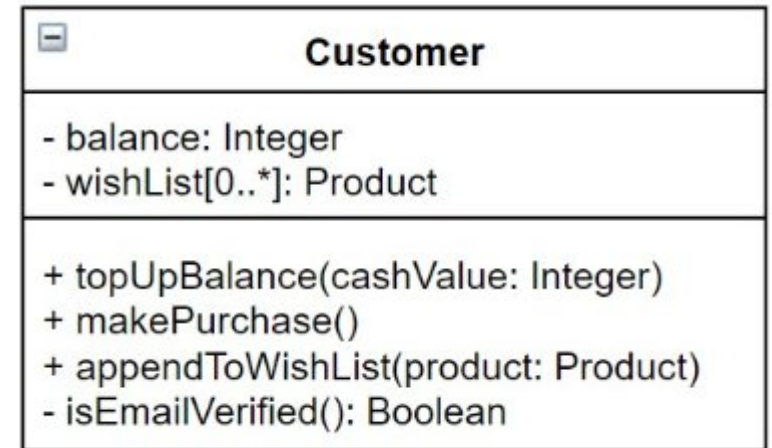
## 2. Статические члены подчеркиваются

## 3. Кратность указывается в квадратных скобках:

- [0..1] - 0 или 1 объект
- [0..\*] - 0 или неограниченное количество
- [1..\*] - 1 или неограниченное количество

## 4. Указание типа поля/аргумента/возвращаемого значения

- <Имя>:<Тип>





# Отношения между классами

- **Отношение ассоциации** используют, чтобы показать, что между классами существует некоторая связь. Обычно с помощью него на диаграмме классов показывают, что один класс пользуется функционалом другого класса.
- **Отношение зависимости** используют, чтобы показать, что изменение одного класса требует изменение другого класса. Стрелка отношения зависимости направлена от зависимого класса к независимому.
- **Отношение наследования** используется, чтобы показать, что один класс является базовым для другого. Стрелка от производного к базовому.
- **Отношение агрегации** между двумя классами показывает, что один из них включает в себя другой класс в качестве составной части. С отношением агрегации можно использовать кратность. Стрелка направлена к агрегату.
- **Отношение композиции** является частным случаем отношения агрегации, но есть нюанс – включаемые классы, не могут существовать обособленно. С отношением композиции можно использовать кратность. Стрелка направлена к целому.



Отношение ассоциации (от англ. "association relationship")



Отношение зависимости (от англ. "dependency relationship")



Отношение обобщения (от англ. "generalization relationship")

Отношение наследования (от англ. "inheritance relationship")



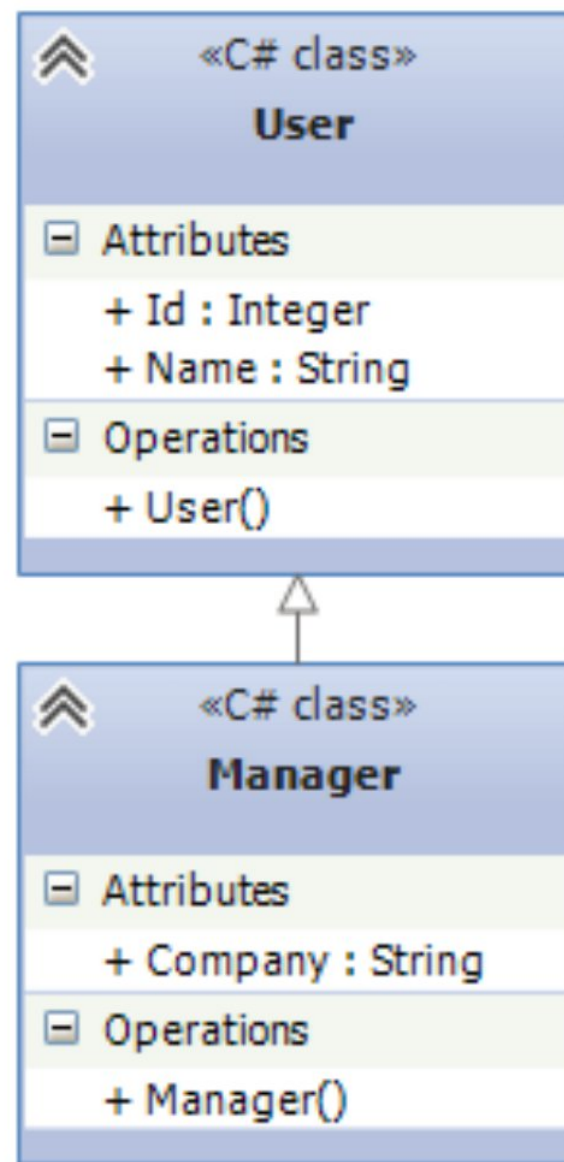
Отношение агрегации (от англ. "aggregation relationship")



Отношение композиции (англ. "composition relationship")

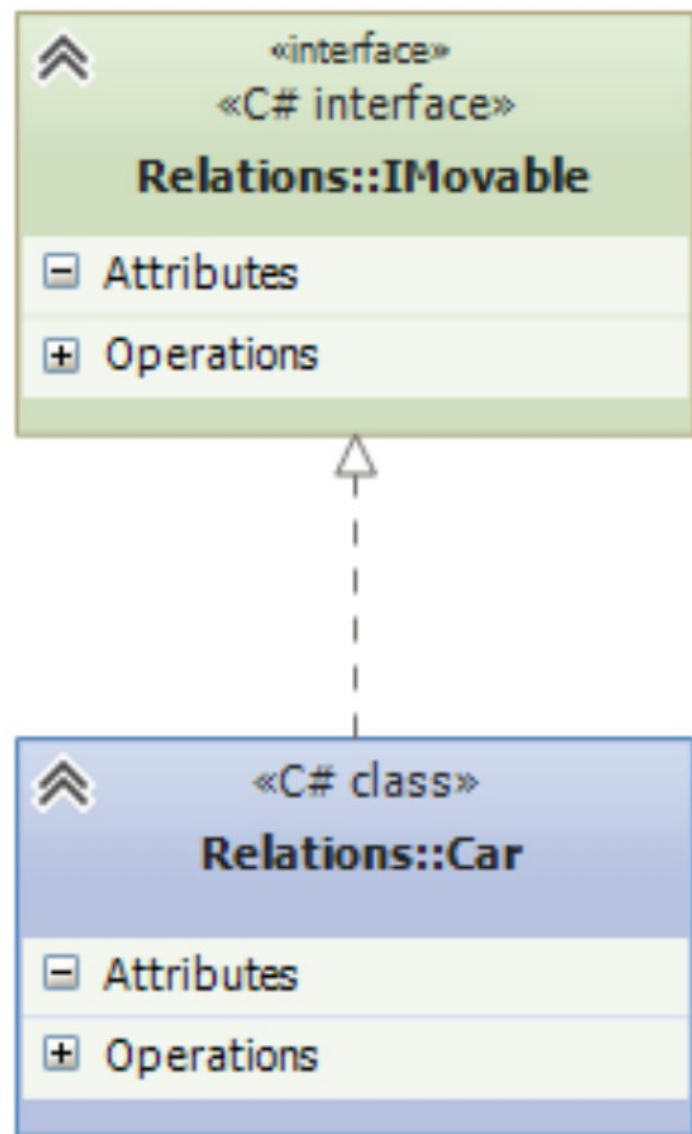
# Наследование

```
1 class User
2 {
3     public int Id { get; set; }
4     public string Name { get; set; }
5 }
6
7 class Manager : User
8 {
9     public string Company { get; set; }
10 }
```



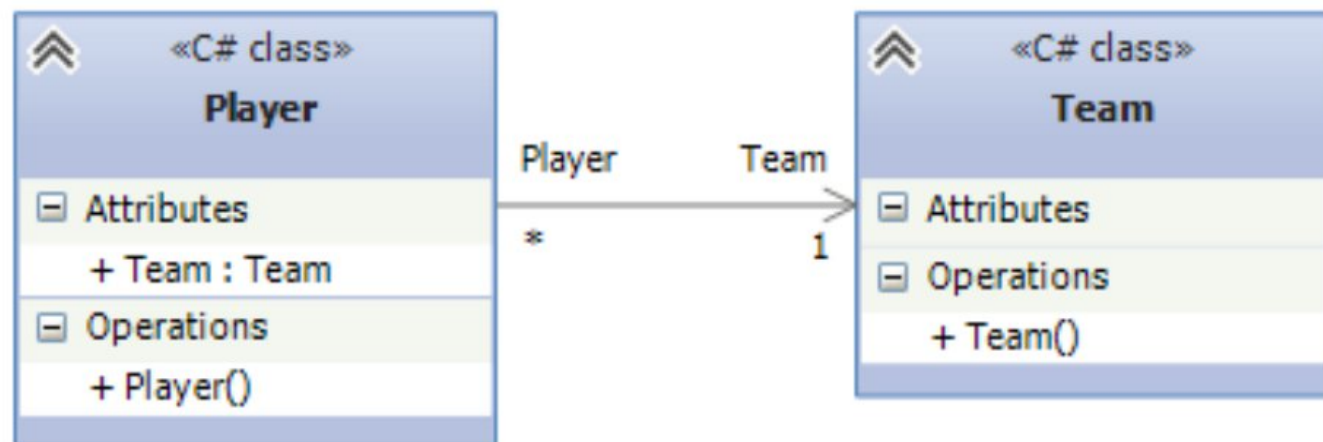
## Реализация

```
1 public interface IMovable
2 {
3     void Move();
4 }
5 public class Car : IMovable
6 {
7     public void Move()
8     {
9         Console.WriteLine("Машина едет");
10    }
11 }
```



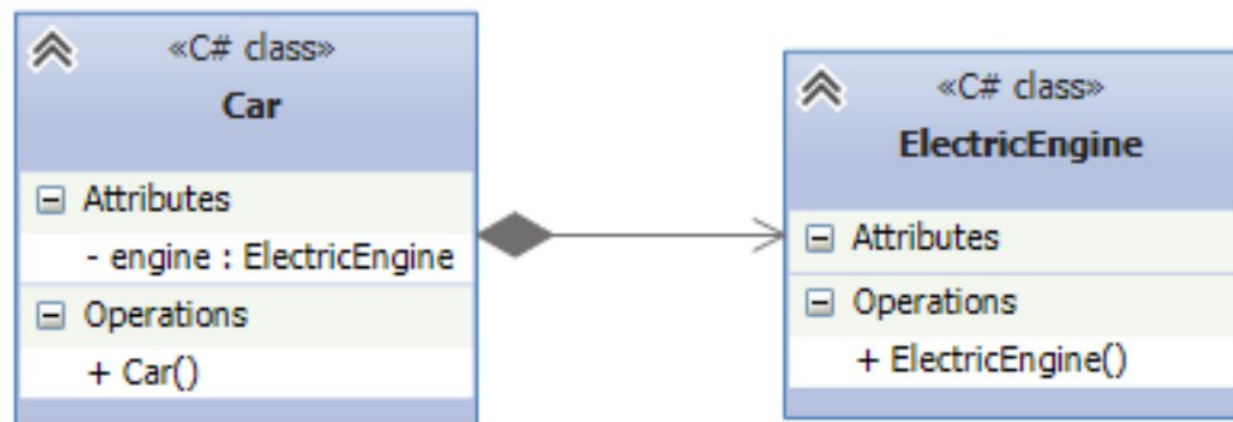
## Ассоциация

```
1 class Team
2 {
3
4 }
5 class Player
6 {
7     public Team Team { get; set; }
8 }
```



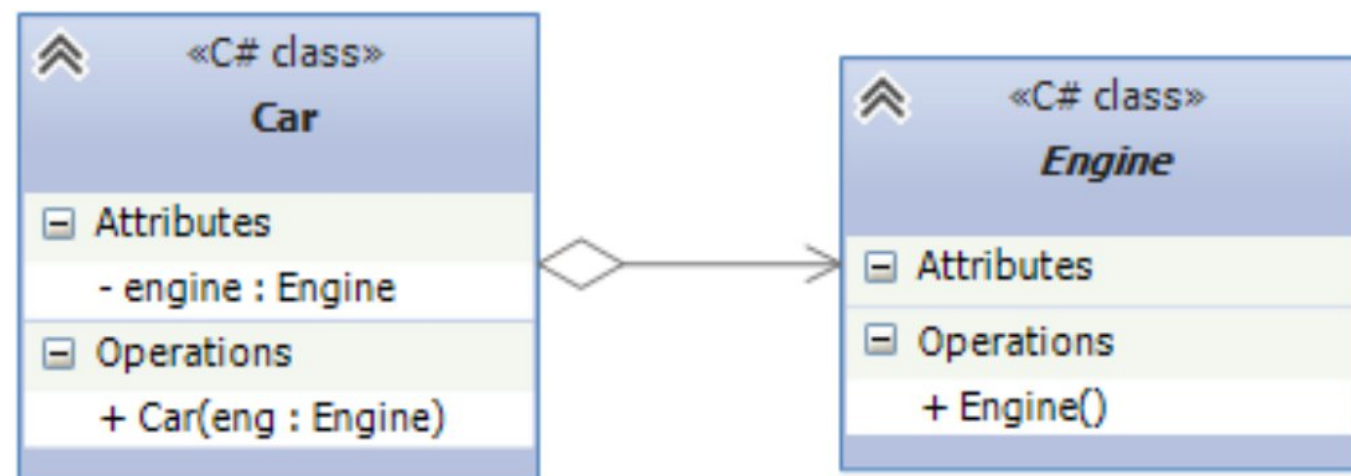
# Композиция

```
1 public class ElectricEngine
2 { }
3
4 public class Car
5 {
6     ElectricEngine engine;
7     public Car()
8     {
9         engine = new ElectricEngine();
10    }
11 }
```



## Агрегация

```
1 public abstract class Engine
2 { }
3
4 public class Car
5 {
6     Engine engine;
7     public Car(Engine eng)
8     {
9         engine = eng;
10    }
11 }
```



## Дополнительные материалы

Чтобы быстро вспомнить C#, прогляните первые главы руководства по языку на метаните:

- <https://metanit.com/sharp/tutorial/>

Также можете почитать руководства от Майкрософт:

- <https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/object-oriented/>
- <https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/tutorials/oop>