# Семинар #1

Вспоминаем ООП

# Формула*

НАКОП_М2 = 0,3 * АУД_М2 + 0,4 * ДЗ_М2 + 0,3 * КР_М2
НАКОП_М3 = 0,4 * АУД_М3 + 0,6 * ДЗ_М3

НАКОП_СР_М2_3 = (НАКОП_М2 + НАКОП_М3) / 2

ПРОМ_ЭКЗ_М3 — оценка за промежуточный экзамен в модуле 3
ПРОМ_ОЦЕН_М3 = 0,65 * НАКОП_СР_М2_3 + 0,35 * ПРОМ_ЭКЗ_М3

НАКОП_М4 = 0,4 * АУД_М4 + 0,6 * ДЗ_М4
СР_НАКОП = (ПРОМ_ОЦЕН_М3 + НАКОП_М4) / 2

ФИН_ОЦЕН = 0,7 * СР_НАКОП + 0,3 * ФИН_ЭКЗ_М4

ПРОМ_ЭКЗ_М3 и ФИН_ЭКЗ_М4 – блокирующие

* - формула 2023-2024 учебного года

# Формула (2025)

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. Тот кто не сдал или не дал ни одного правильного ответа получает за семинар 0, студенты набравшие <= половины правильных ответов получают 5, а те кто набрал > половины правильных ответов получают 10. Средняя всех этих оценок пойдет в *seminars_estimation*.
- *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. Четких критериев оценивания или сроков проверки тут не будет, но и обижать никого тоже не стану. Средняя всех оценок пойдет соответственно в *homework_estimation*.
- *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. По сути сдадут мини-коллок :)

# ООП

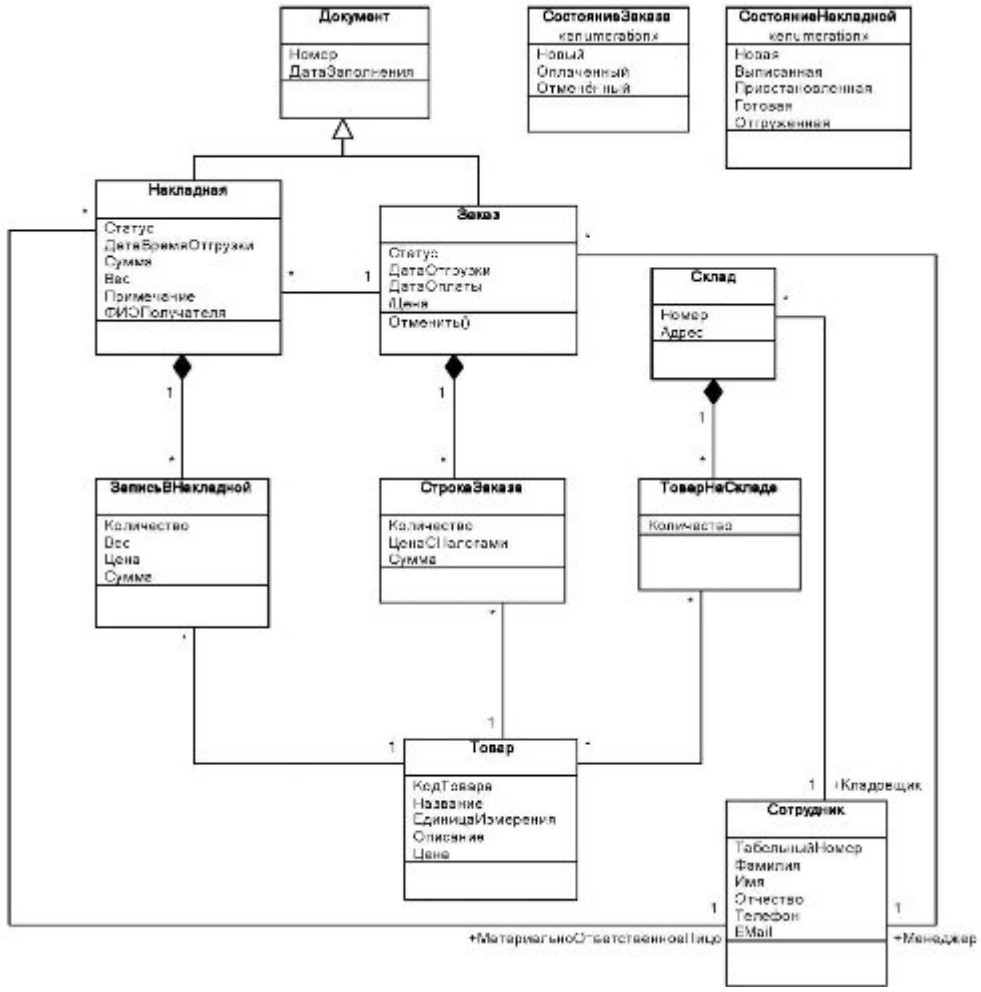
**О**бъектно-**о**риентированное **п**рограммирование - методология программирования, основанная на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования. Прежде чем начать писать инструкции 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

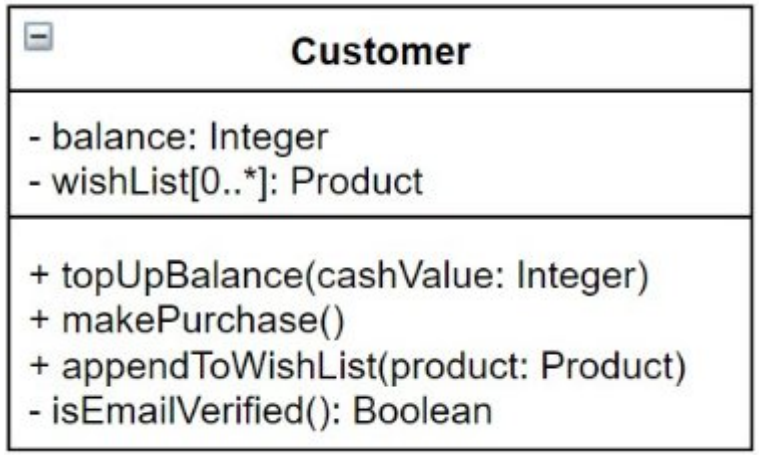
Диаграмма классов (class diagram) предназначена для представления внутренней структуры объектно-ориентированных систем в виде классов и связей между ними.

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>

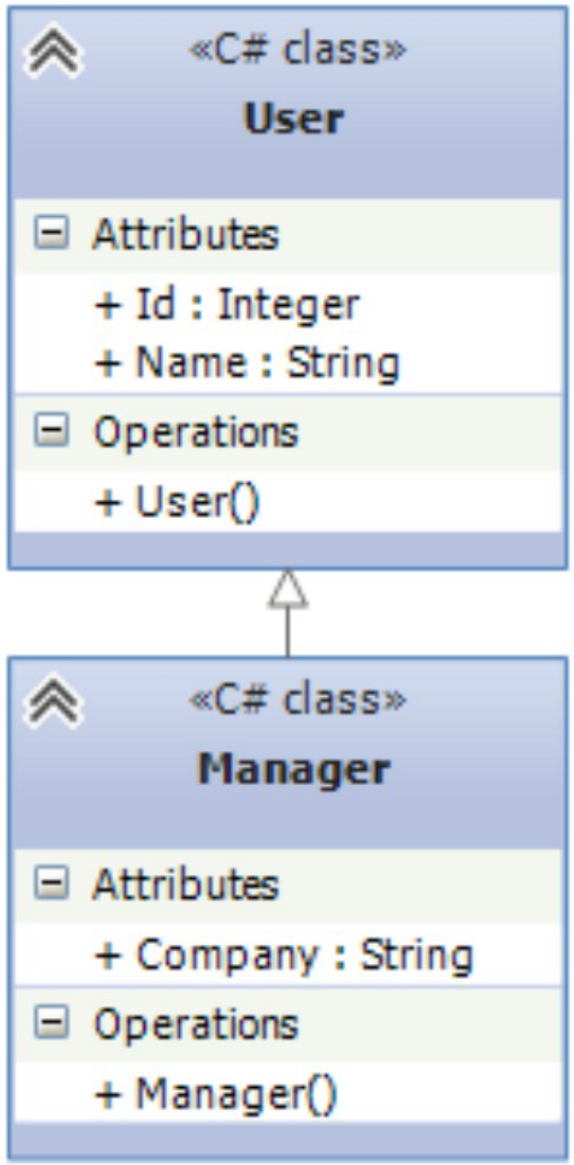


# Отношения между классами

- **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.

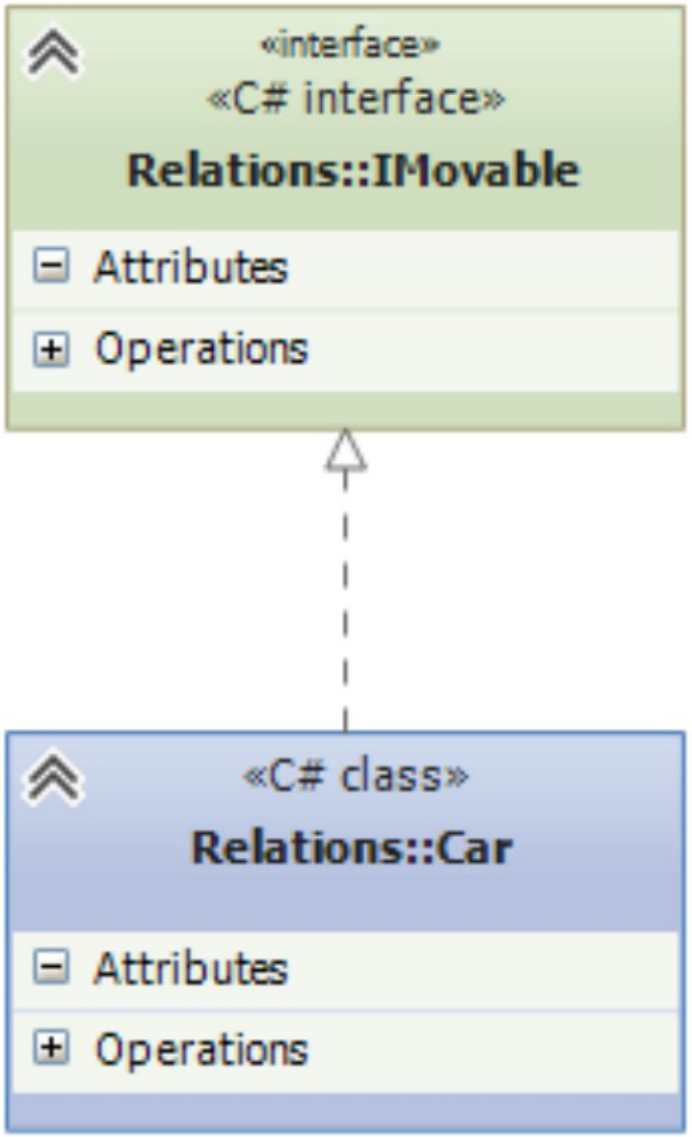
# Наследование

```
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 }
```



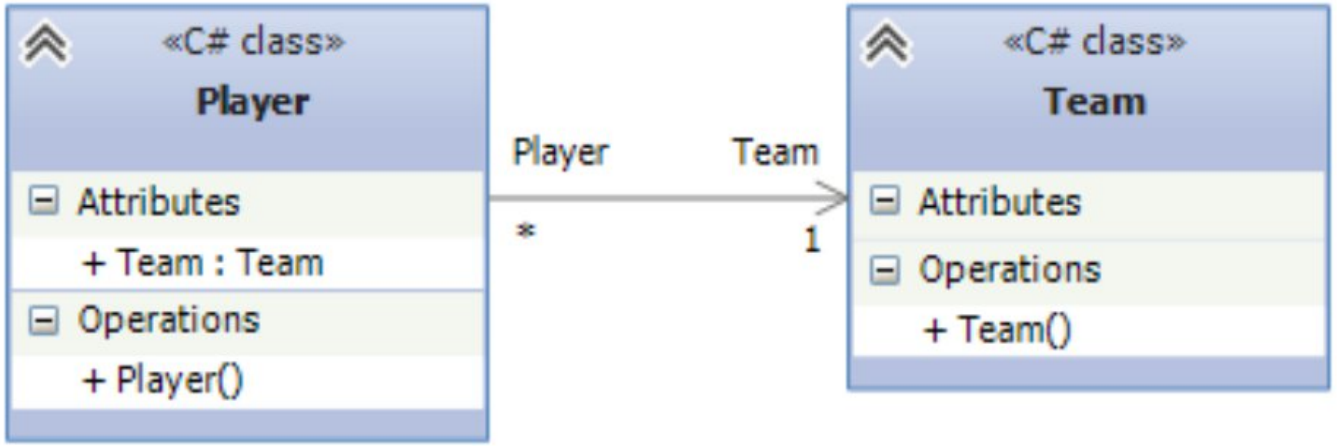
# Реализация

```
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 }
```



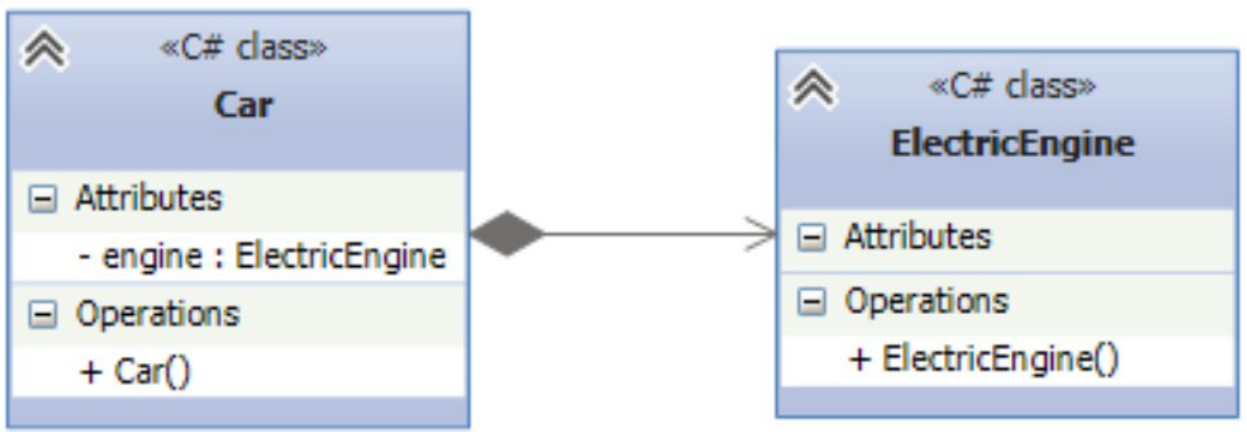
# Ассоциация

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



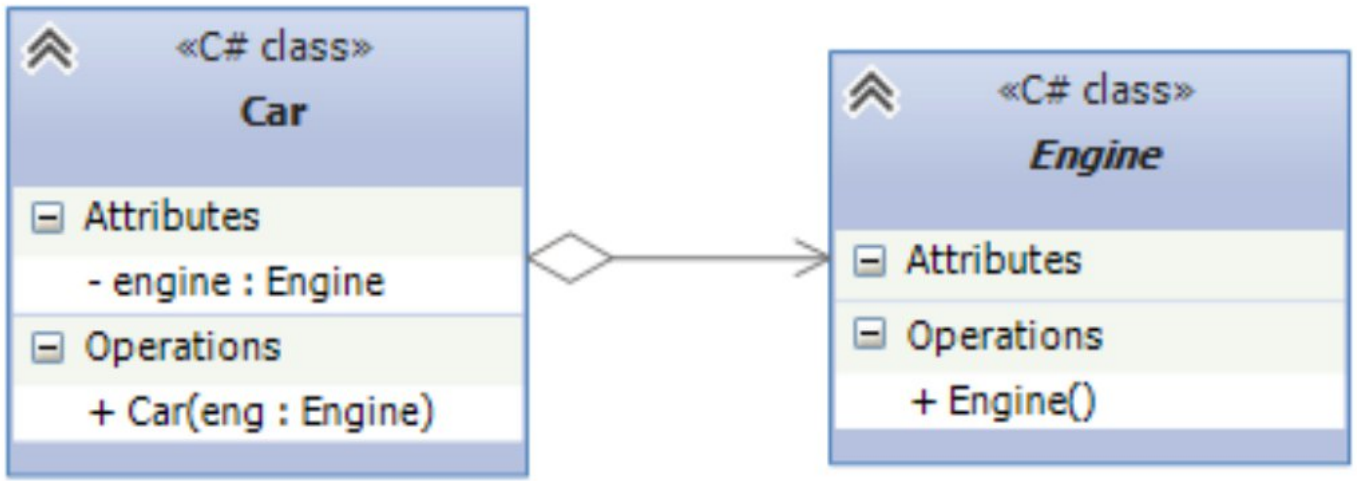
# Композиция

```
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#, проглядите первые главы руководства по языку на метаните (там есть и про ООП в 4 главе):

- 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