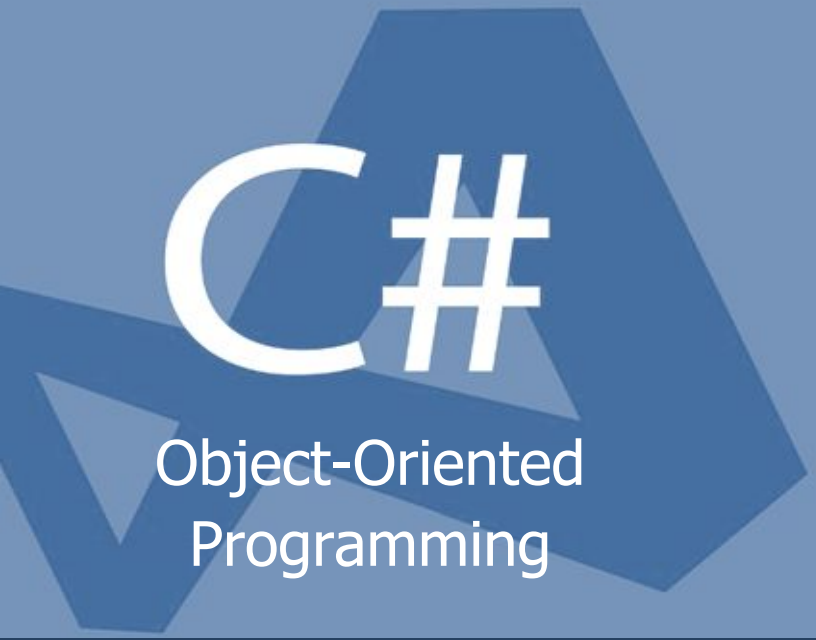


Object-Oriented Programming

3

Class relations

Емельянов Виталий Александрович

: v.yemelyanov@gmail.com

Отношения между классами

Особенности классов и объектов в ООП:

- **Классы редко бывают изолированными** друг от друга
- **Классы связаны друг с другом разными видами отношений**, которые характеризуют разные виды взаимодействия

Основными типами связей между классами являются:

- отношение наследования
- отношение реализации
- отношение ассоциации
- отношения композиции и агрегации

Отношения между классами: Наследование

- Наследование позволяет одному классу (наследнику) унаследовать функционал другого класса (родительского).
- Отношения наследования еще называют генерализацией или обобщением.
- Наследование определяет отношение **IS A**, то есть "является"

UML

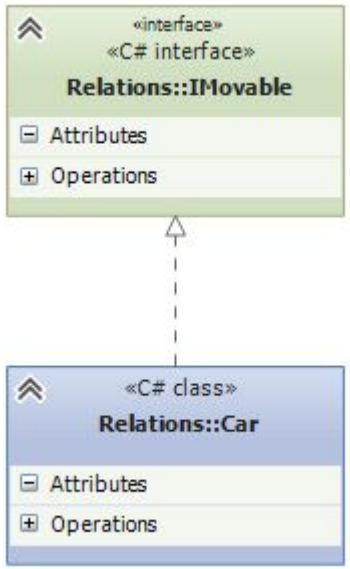
C#

```
1
2   class User
3   {
4       public int Id { get; set; }
5       public string Name { get; set; }
6   }
7
8   class Manager : User
9   {
10      public string Company { get; set; }
11  }
12
```

Отношения между классами: Реализация

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

UML



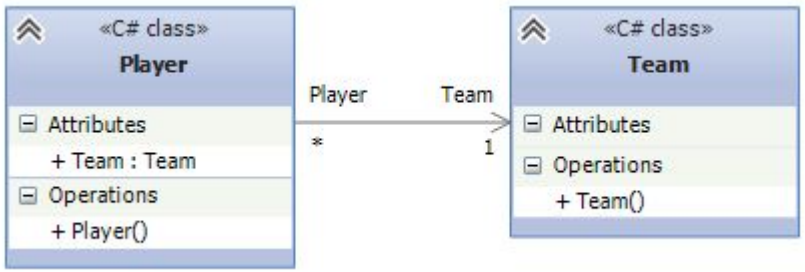
C#

```
1
2  public interface IMovable
3  {
4      void Move();
5  }
6  public class Car : IMovable
7  {
8      public void Move()
9      {
10         Console.WriteLine("Машина едет");
11     }
12 }
```

Отношения между классами: Ассоциация

- **Ассоциация** - это отношение, при котором объекты одного типа неким образом связаны или используют объекты другого типа.
- Например, игрок играет в определенной команде (класс `Player` связан отношением ассоциации с классом `Team`):

UML



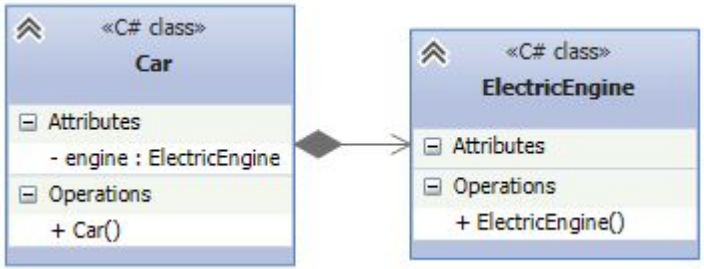
C#

```
1
2   class Team
3   {
4
5   }
6
7   class Player
8   {
9       public Team Team
10      { get; set; }
11  }
12
```

Отношения между классами: Композиция

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

UML



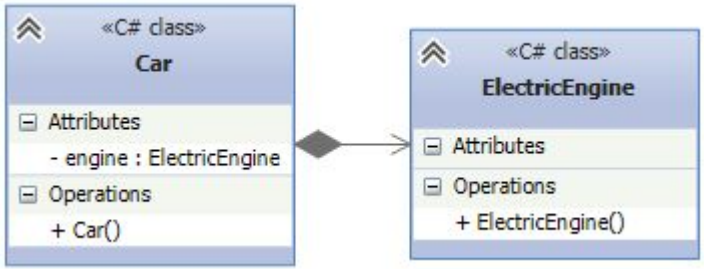
C#

```
1
2  public class ElectricEngine
3  { }
4
5  public class Car
6  {
7      ElectricEngine engine;
8      public Car()
9      {
10         engine = new ElectricEngine();
11     }
12 }
```

Отношения между классами: Композиция

- 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 - зависимым.

UML



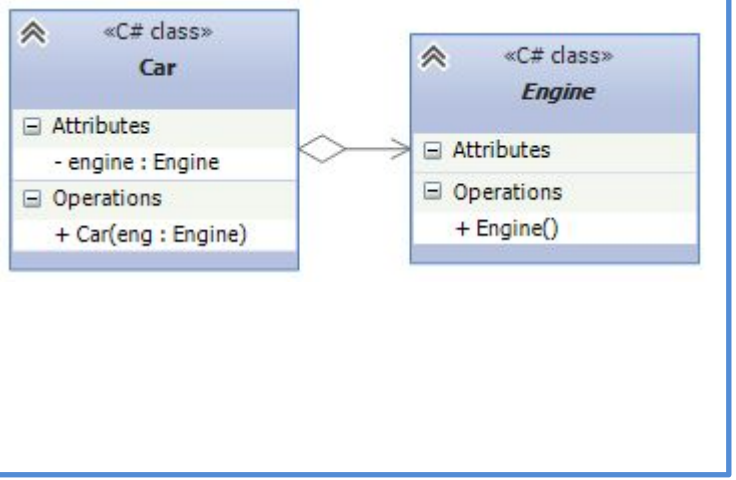
C#

```
1
2  public class ElectricEngine
3  { }
4
5  public class Car
6  {
7      ElectricEngine engine;
8      public Car()
9      {
10         engine = new ElectricEngine();
11     }
12 }
```

Отношения между классами: Агрегация

- Агрегация также предполагает отношение **HAS A**, но реализуется она иначе:

UML



C#

```
1   public abstract class Engine
2   { }
3
4   public class Car
5   {
6       Engine autoEngine;
7
8       public Car(Engine engine1)
9       {
10          autoEngine = engine1;
11      }
12  }
```

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.

Рекомендации при проектировании отношений между классами

- **Вместо наследования следует предпочитать композицию**

  При наследовании весь функционал класса-наследника жестко определен на этапе компиляции. И во время выполнения программы мы не можем его динамически переопределить. А класс-наследник не всегда может переопределить код, который определен в родительском классе. Композиция же позволяет динамически определять поведение объекта во время выполнения, и поэтому является более гибкой.

- **Вместо композиции следует предпочитать агрегацию, как более гибкий способ связи компонентов**

  НО не всегда агрегация уместна. Например, есть класс человека, который содержит объект нервной системы. В реальности невозможно вовне определить нервную систему и внедрить ее в человека. То есть в данном случае человек будет главным компонентом, а нервная система - зависимым, подчиненным, и **их**