Интерфейсы и композиция

Как вы помните, композиция – это строгий вариант агрегирования, когда включаемый объект может существовать только как часть контейнера. Если контейнер будет уничтожен, то и включённый объект тоже будет уничтожен (при агрегации, включенный объект может жить отдельно от контейнера).

Подход, основанный на наследовании типов, подвергается критике со следующими аргументами:

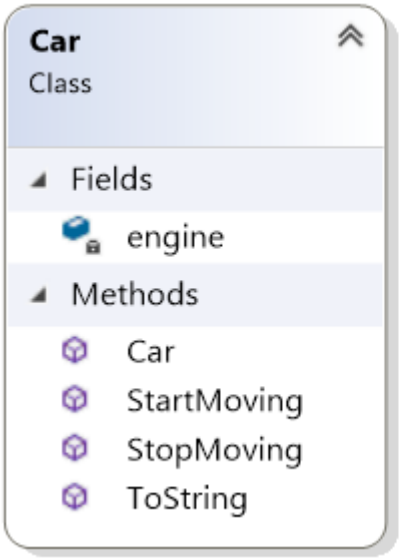
- дочерний класс наследует все данные и поведение родительского, что нужно не всегда;
- виртуальные методы менее производительные;
- концепция наследования считается довольно сложной;

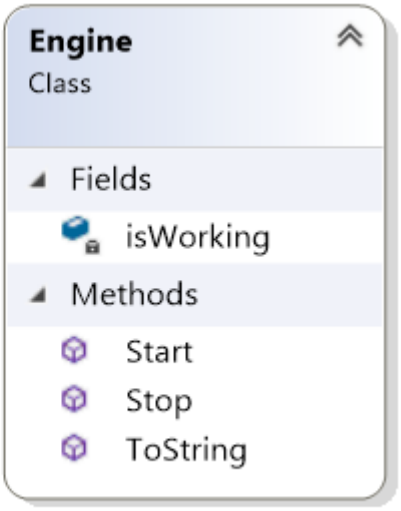
Сложности наследования реализации и тесной связи между типами при композиции решают с помощью интерфейсов, предоставляющих функциональные контракты. Вся логика действий (функциональность) разбивается на отдельные интерфейсы (чем проще интерфейс – тем лучше) и каждый тип может реализовать свой набор действий (для выполнения контрактов).

Задача состоит в том, чтобы реализовать один и тот же функционал – взаимосвязанную работу двух типов разными способами: композицией и через интерфейс.

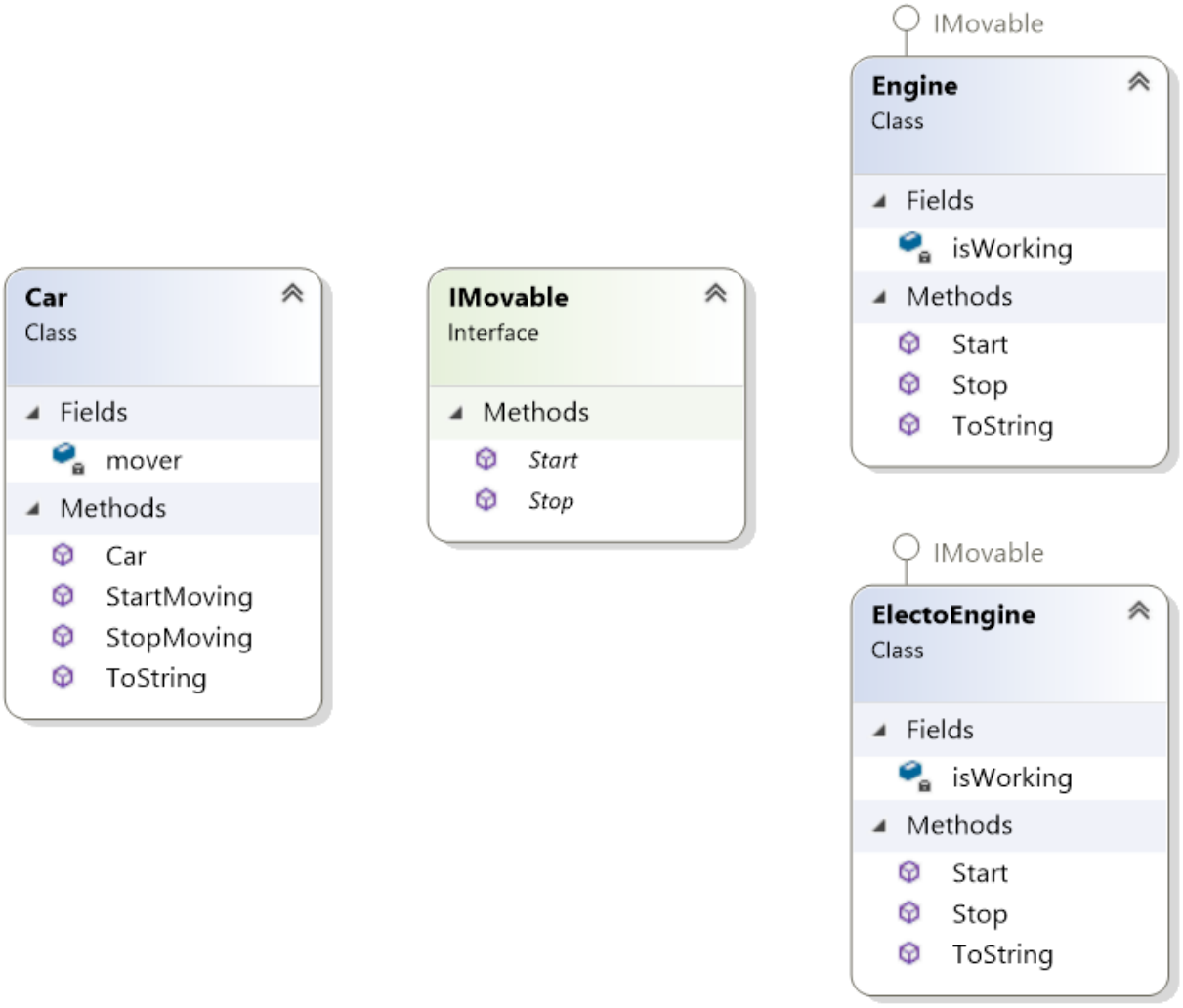
Для этого создайте два отдельных проекта.

1. В первом проекте нужно разработать класс Car, который будет композиционно включать в себя экземпляр класса Engine с возможностью передвижения (см. диаграмму классов). В основной программе создать объект типа Car и вызвать его методы StartMoving и StopMoving (после каждого вызова выводить информацию об объекте Car).





2. Во втором проекте нужно разработать класс Car, который будет агрегировать интерфейс IMovable и использовать его. Сам интерфейс должен быть реализован в классе Engine / ElectroEngine (см. диаграмму классов). В основной программе создать два объекта типа Car (в конструктор первого объекта передать ссылку на экземпляр Engine, а в конструктор второго - ссылку на экземпляр ElectroEngine) и вызвать их методы StartMoving и StopMoving (после каждого вызова выводить информацию об объектах Car).



3.