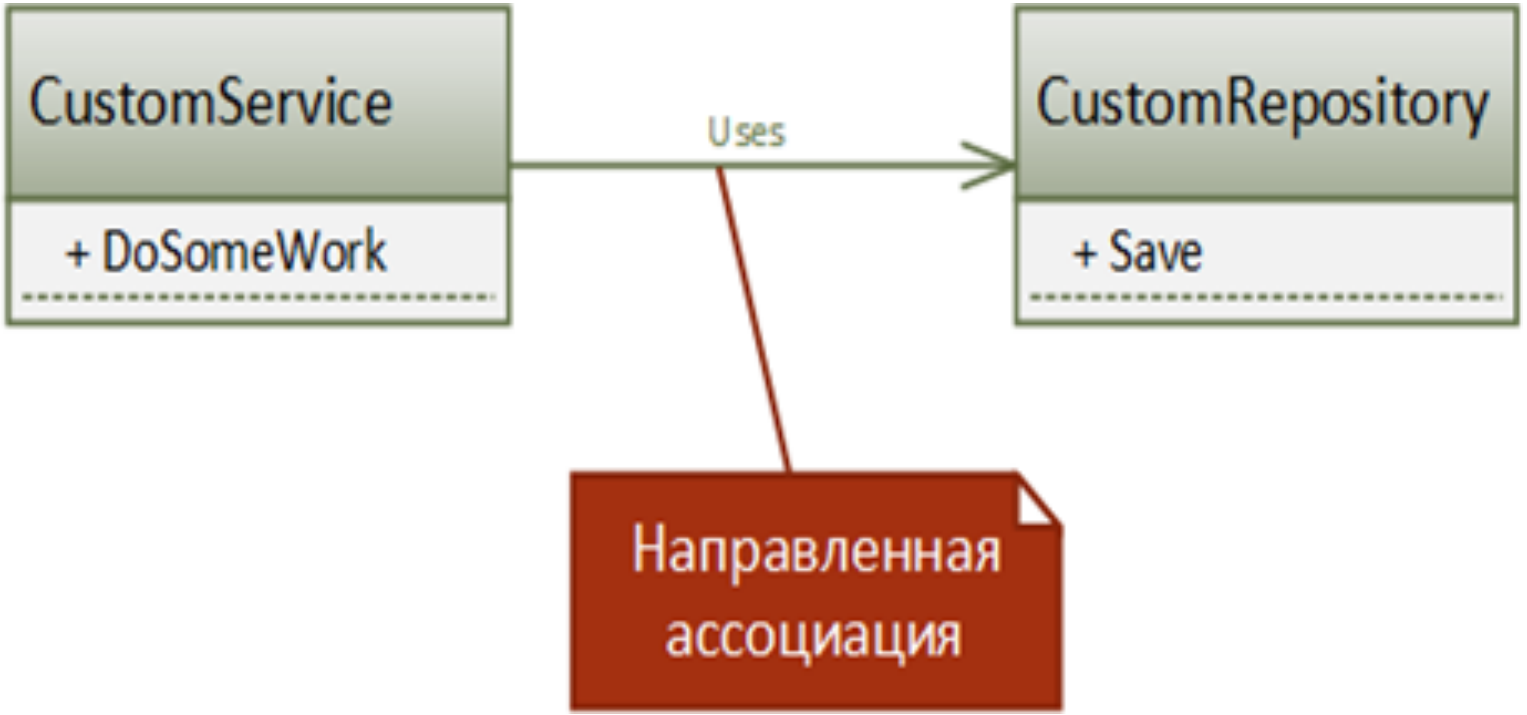
Наследование vs Композиция vs Агрегация

Наследование vs Композиция vs Агрегация

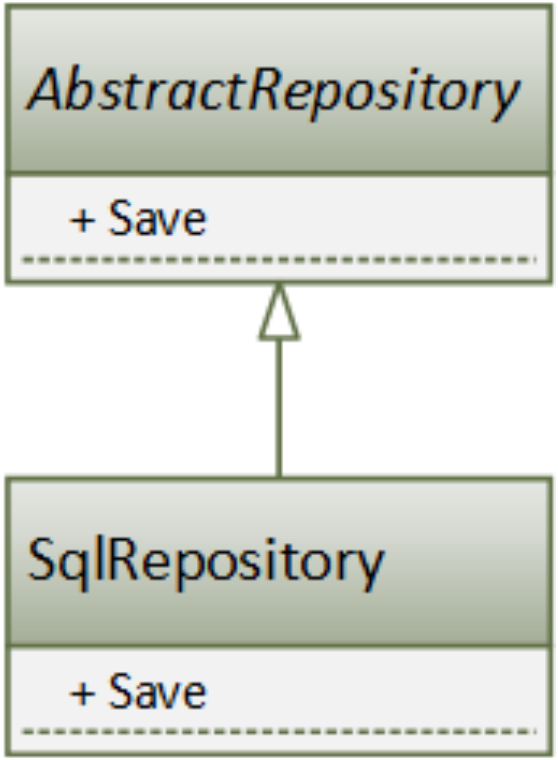
Между двумя классами/объектами существует разные типы отношений
Базовый тип отношений – *ассоциация* (association)



Отношение ассоциации

Наследование vs Композиция vs Агрегация

Более точный тип отношений – отношение открытого наследования (отношение «является», **IS A Relationship)** – все, что справедливо для базового класса справедливо и для его наследника



Отношение наследование

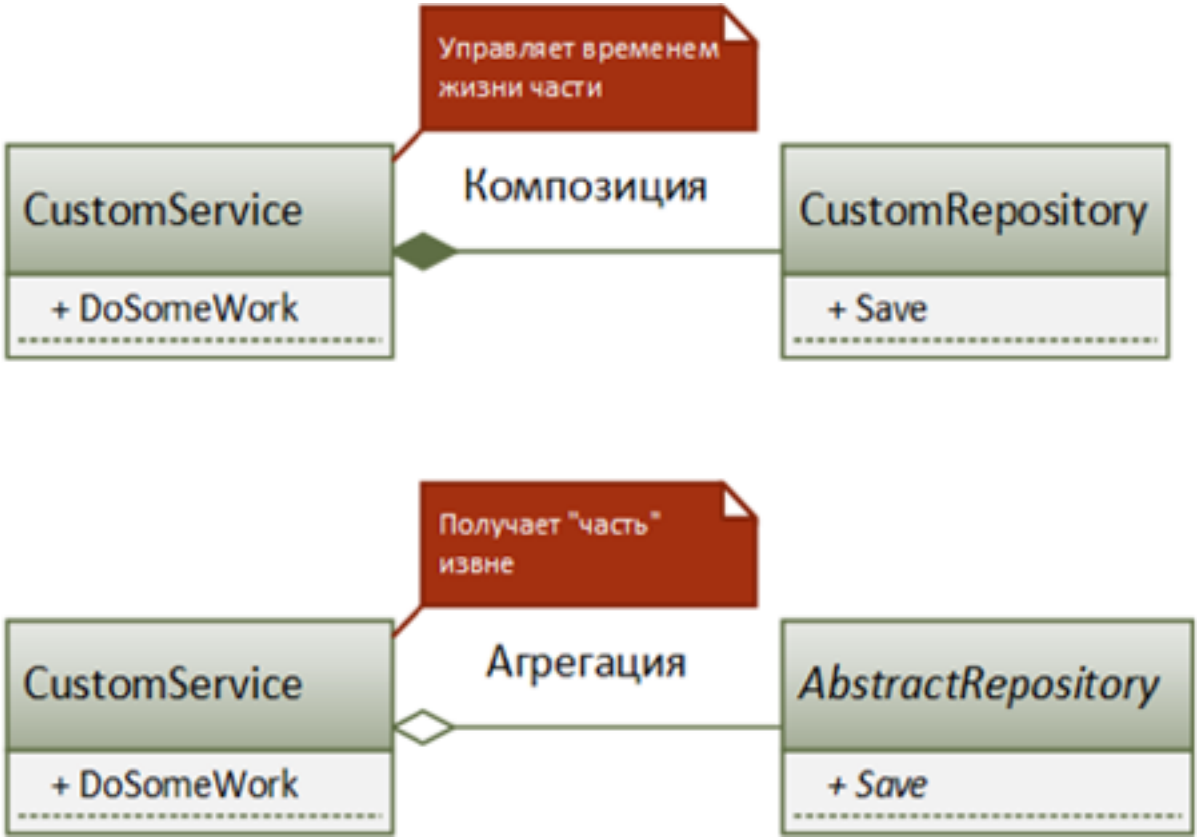
Наследование vs Композиция vs Агрегация

- полиморфное поведение
- абстрагирование от конкретной реализации классов
- работа с абстракциями (интерфейсами или базовыми классами)
- не обращаем внимание на детали реализации

- не все отношения между классами определяются отношением «является»
- наследование является самой сильной связью между двумя классами, которую невозможно разорвать во время исполнения (это отношение является статическим и, в строго типизированных языках определяется во время компиляции)

Наследование vs Композиция vs Агрегация

Отношения: **композиция** (composition) и **агрегация** (aggregation)
Моделируют отношение «является частью» (**HAS-A Relationship**) и обычно выражаются в том, что класс целого содержит поля (или свойства) своих составных частей



Отношения композиции и агрегации

Наследование vs Композиция vs Агрегация

Разница между композицией и агрегацией заключается в том, что **в случае композиции целое явно контролирует время жизни своей составной части** (часть не существует без целого), а **в случае агрегации целое хоть и содержит свою составную часть, время их жизни не связано** (например, составная часть передается через параметры конструктора

```
class CompositeCustomService
{
    private readonly CustomRepository _repository = new CustomRepository();

    public void DoSomething()
    {
        // Используем _repository
    }
}
```

Явный контроль времени жизни обычно приводит к более высокой связанности между целым и частью, поскольку используется конкретный тип, тесно связывающий участников между собой

Наследование vs Композиция vs Агрегация

Разница между композицией и агрегацией заключается в том, что **в случае композиции целое явно контролирует время жизни своей составной части** (часть не существует без целого), а **в случае агрегации целое хоть и содержит свою составную часть, время их жизни не связано**(например, составная часть передается через параметры конструктора

```
class AggregatedCustomService
{
    private readonly AbstractRepository _repository;

    public AggregatedCustomService(AbstractRepository repository)
    {
        _repository = repository;
    }

    public void DoSomething()
    {
        // Используем _repository
    }
}
```

Наследование vs Композиция vs Агрегация

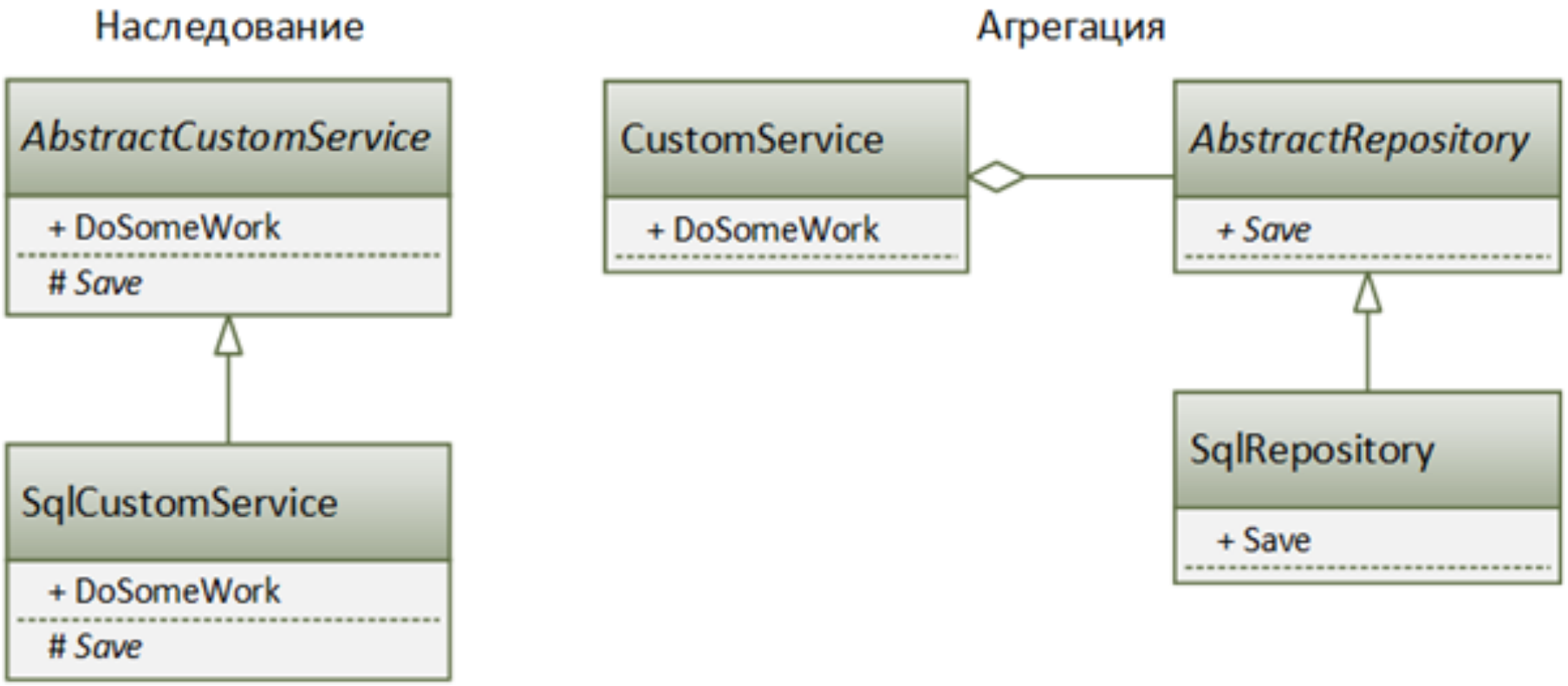
Можно использовать композицию и контролировать время жизни объекта, не завязываясь на конкретные типы (абстрактная фабрика)

```
internal interface IRepositoryFactory
{
    AbstractRepository Create();
}

class CustomService
{
    // Композиция
    private readonly IRepositoryFactory _repositoryFactory;
    public CustomService(IRepositoryFactory repositoryFactory)
    {
        _repositoryFactory = repositoryFactory;
    }
    public void DoSomething()
    {
        var repository = _repositoryFactory.Create();
        // Используем созданный AbstractRepository
    }
}
```

Наследование vs Композиция vs Агрегация

Одну и ту же задачу можно решить десятком разных способов, при этом в одном случае мы получим сильно связанный дизайн с большим количеством наследования и композиции, а в другом случае – эта же задача будет разбита на более автономные строительные блоки, объединяемые между собой с помощью агрегации



Наследование vs Композиция vs Агрегация

Объективные критерии для определения связности дизайна по диаграмме классов:

- большие иерархии наследования (глубокие или широкие иерархии)
- повсеместное использование композиции, а не агрегации скорее всего говорит о сильно связанном дизайне

Отношения на уровне классов.

Generalization. Наследование.

Самое обычное наследование: **class A extends B { }**

Implementation. Имплементация.

Реализация интерфейса: **class A implements I { }**

Отношения на уровне объектов

Association. Ассоциация.

Семейство отношения между объектами классов. "Студент" - "Преподаватель", "Покупатель" - "Продавец" и т.д.
Может обозначаться вообще без стрелки.
Агрегация и Композиция - подтипы ассоциации.

Aggregation. Агрегация.

Подтип ассоциации. Например один класс содержит (агрегируют) объекты другого класса.

Composition. Композиция.

Похоже на агрегацию только более сильная связь. Поэтому закрашенный ромб. Например: если уничтожается композитор, то его объекты классов на которые он ссылается также перестают существовать.

Просто отношение

Dependency. Зависимость.

Классы "каким либо образом" зависят друг от друга. Например, если у одного класса меняются методы, конструкторы или поля, и поэтому приходится переписывать другой класс, то значит они зависимы. Одна из самых слабеньких связей. Например объекты одного класса передаются как параметр в методы другого класса и т.д.