



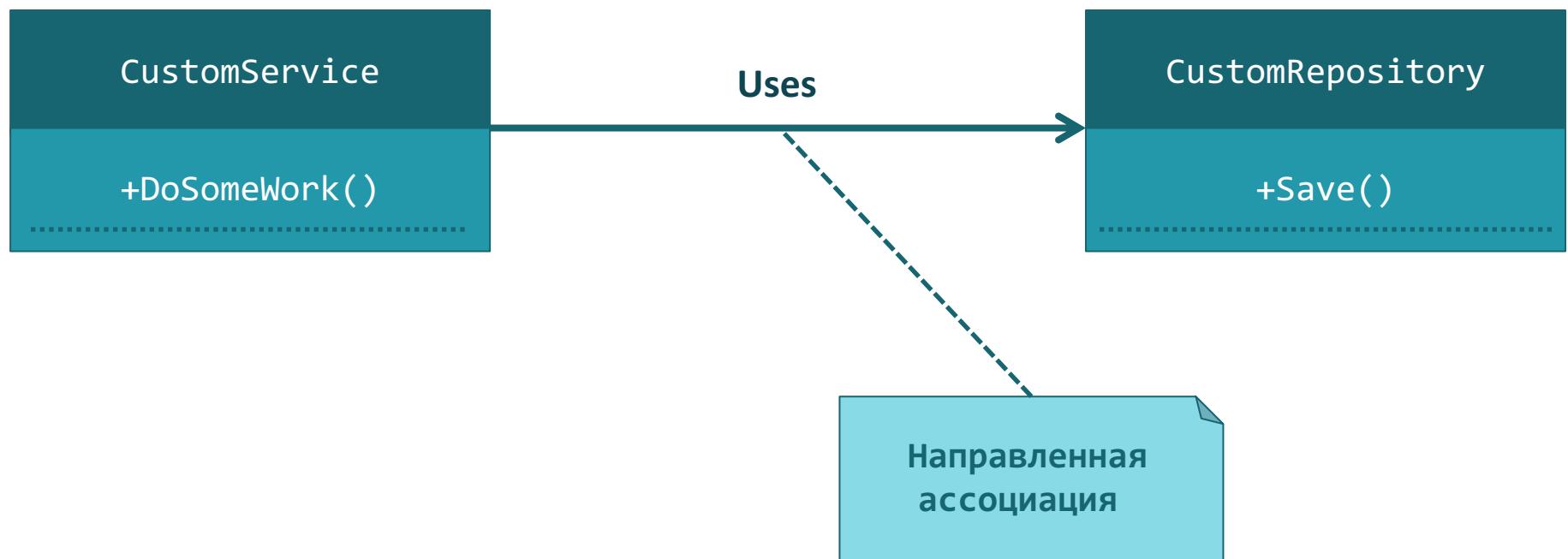
# НЕКОТОРЫЕ АСПЕКТЫ НАСЛЕДОВАНИЯ

.NET & JS LAB

Anzhelika KRAVCHUK

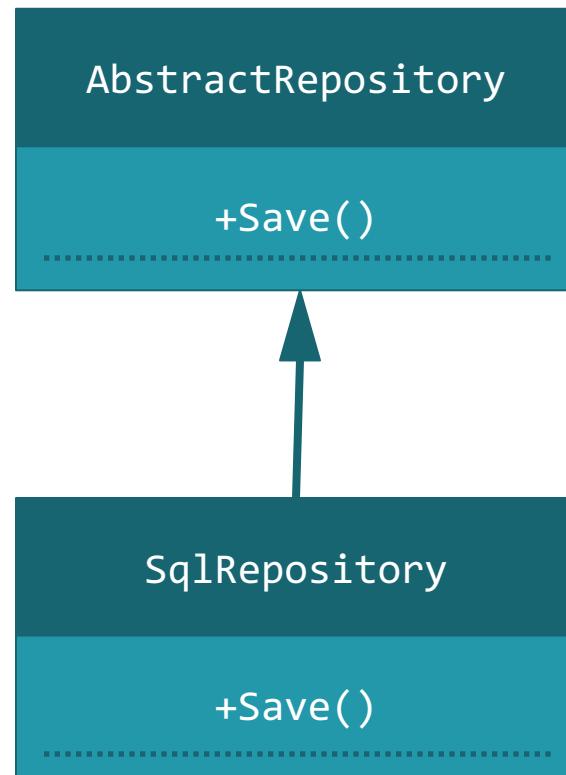
# Наследование vs Композиция vs Агрегация

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



# 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



# Наследование vs Композиция vs Агрегация

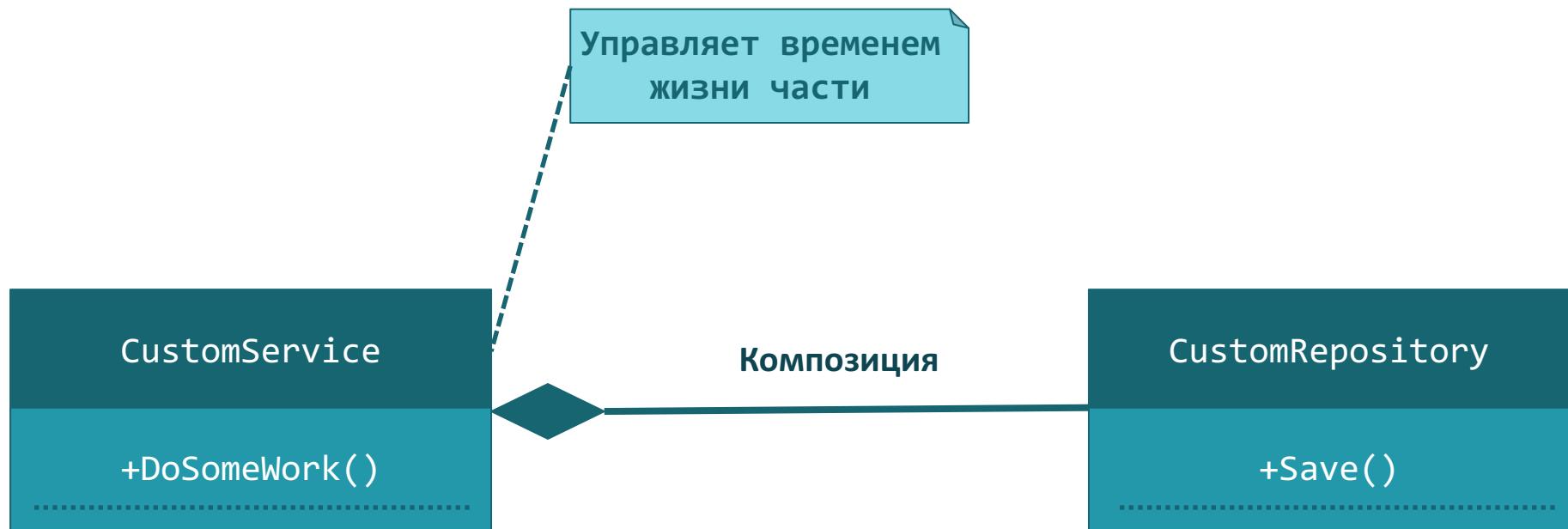
- полиморфное поведение
- абстрагирование от конкретной реализации классов
- работа с абстракциями (интерфейсами или базовыми классами)
- не обращаем внимание на детали реализации

- 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)

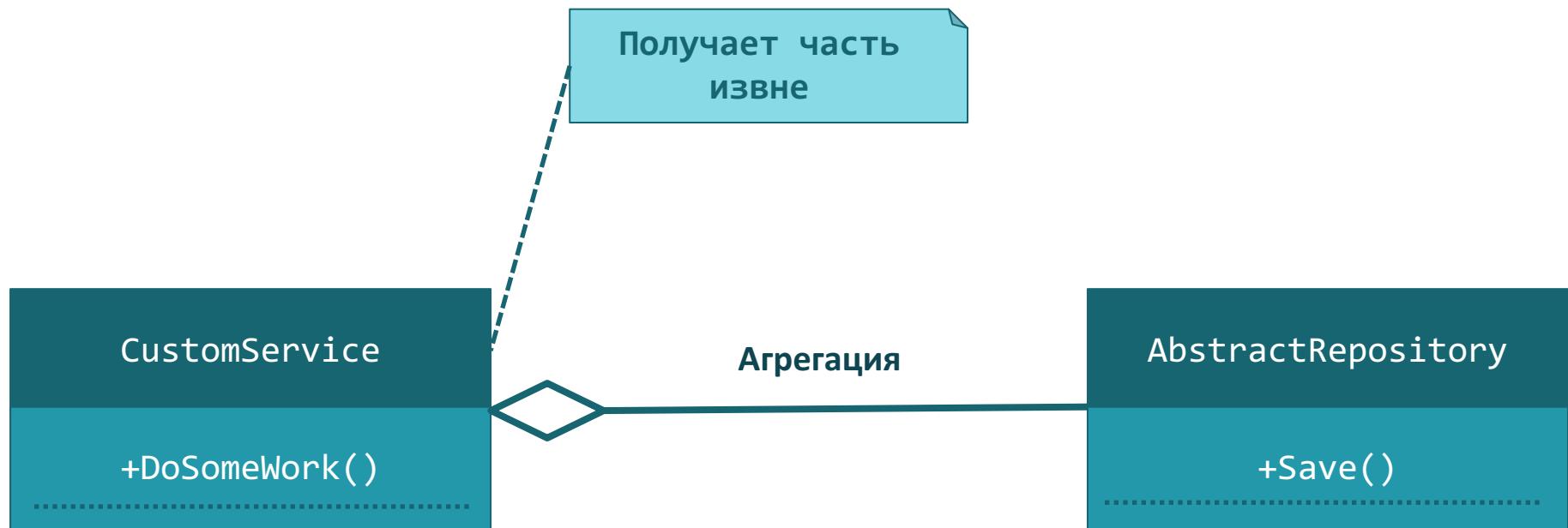
Моделируют отношение «является частью» (**HAS-A Relationship**) и обычно выражаются в том, что класс целого содержит поля (или свойства) своих составных частей



# 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



# 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)
    {
        this.repository = repository;
    }

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

# 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

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

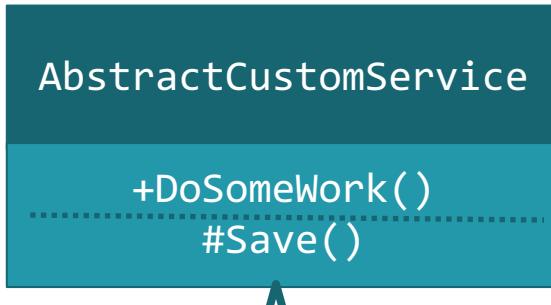
```
public interface IRepositoryFactory
{
    AbstractRepository Create();
}

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

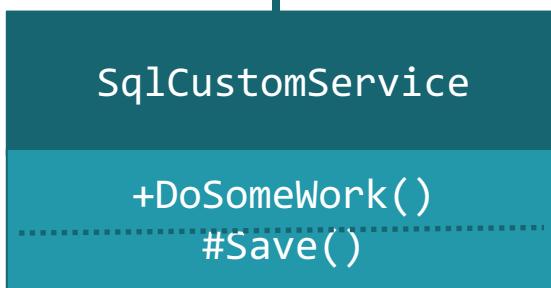
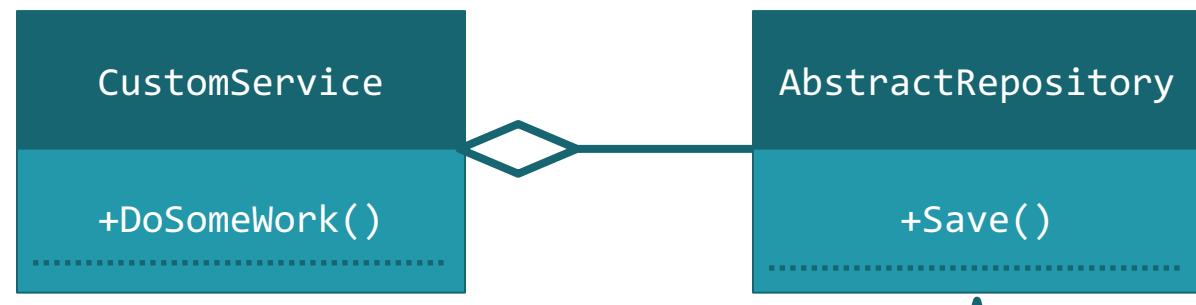
# Наследование vs Композиция vs Агрегация

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



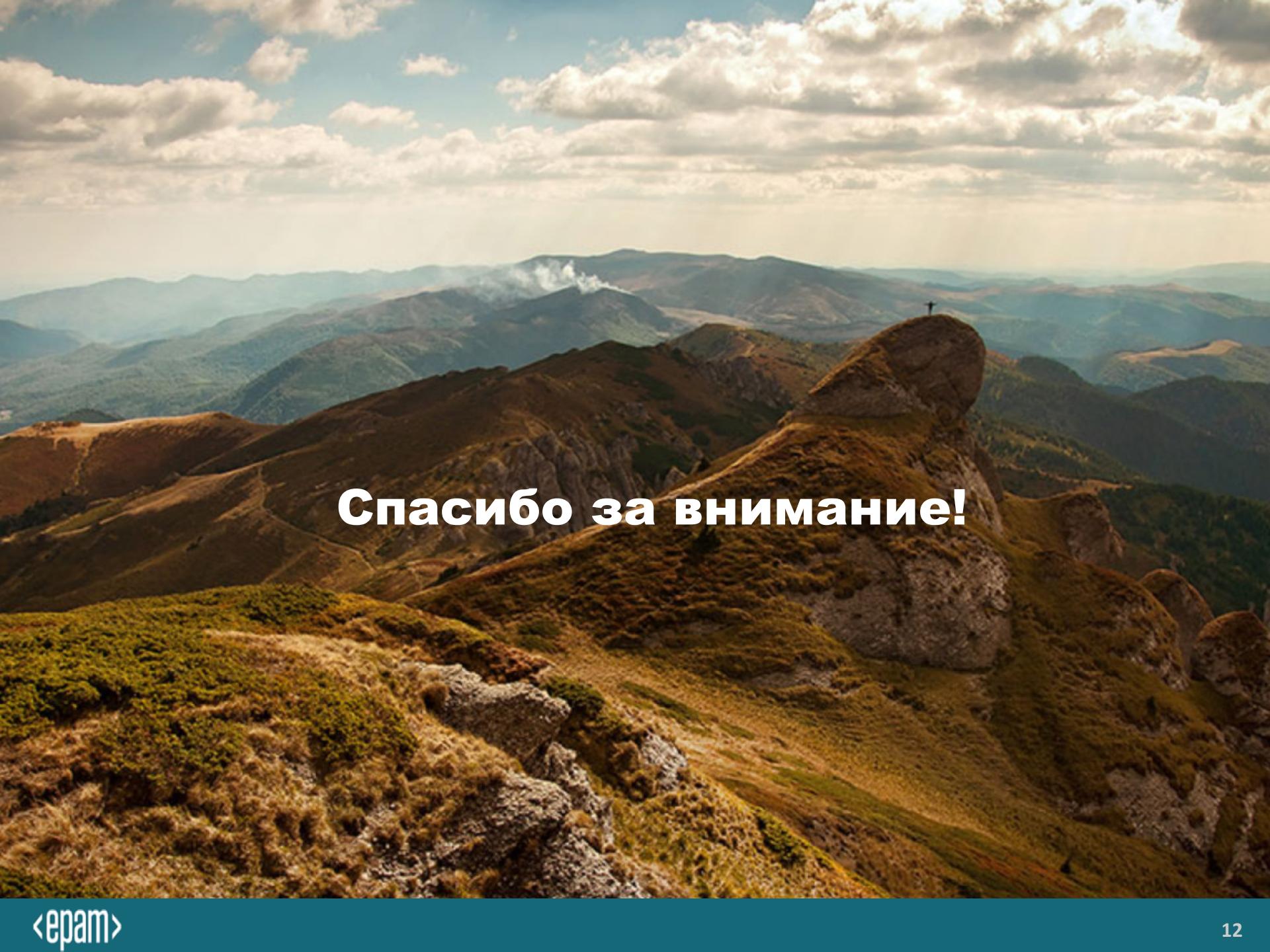
## Агрегация



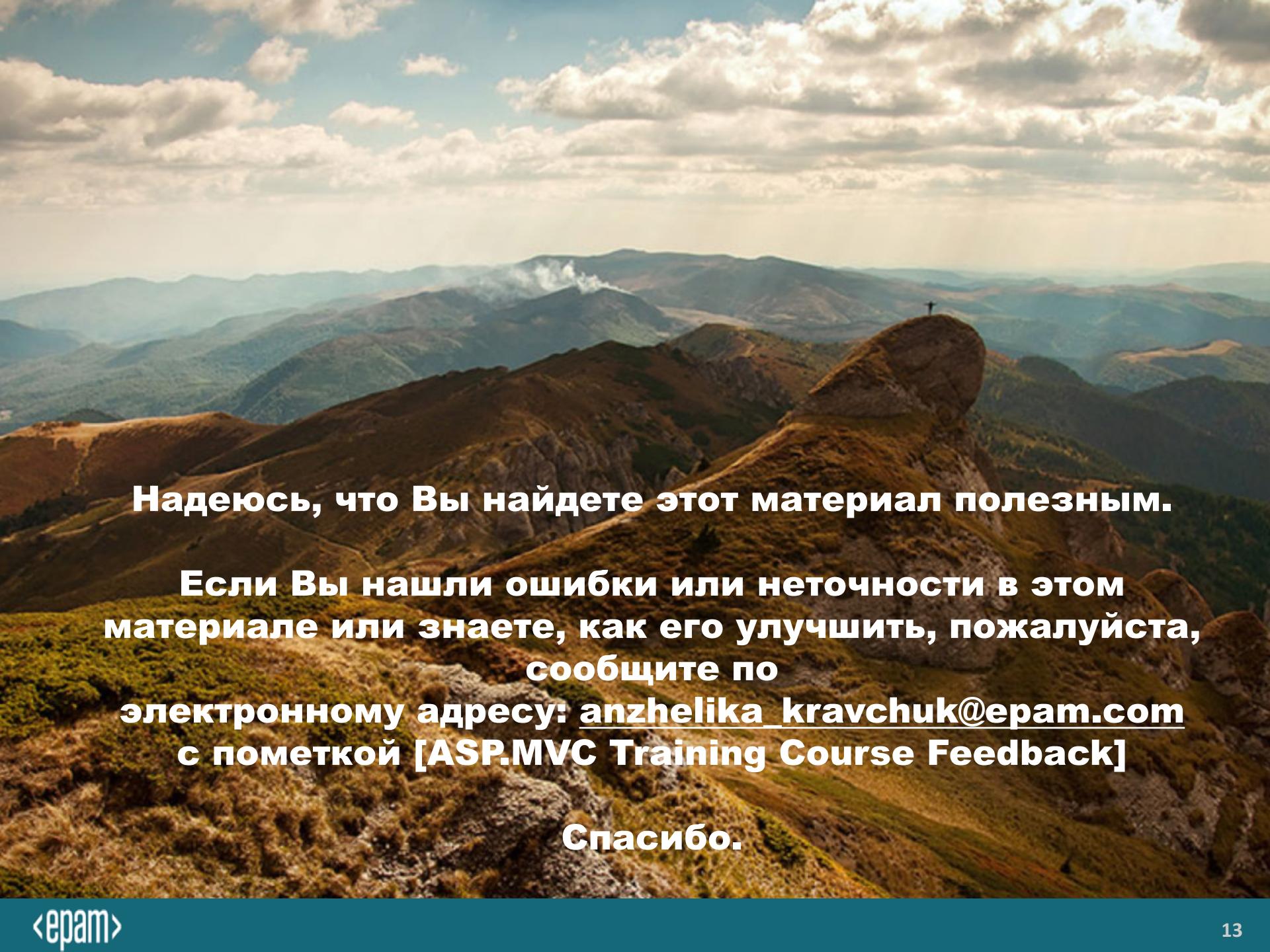
# Наследование vs Композиция vs Агрегация

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

A wide-angle photograph of a mountain range under a dramatic sky. In the foreground, rocky terrain and green slopes are visible. A lone figure stands on a prominent peak in the middle ground. The background features multiple layers of mountains, with a plume of white smoke or steam rising from one of the peaks in the distance.

**Спасибо за внимание!**



**Надеюсь, что Вы найдете этот материал полезным.**

**Если Вы нашли ошибки или неточности в этом  
материале или знаете, как его улучшить, пожалуйста,  
сообщите по**

**электронному адресу: [anzhelika\\_kravchuk@epam.com](mailto:anzhelika_kravchuk@epam.com)  
с пометкой [ASP.MVC Training Course Feedback]**

**Спасибо.**