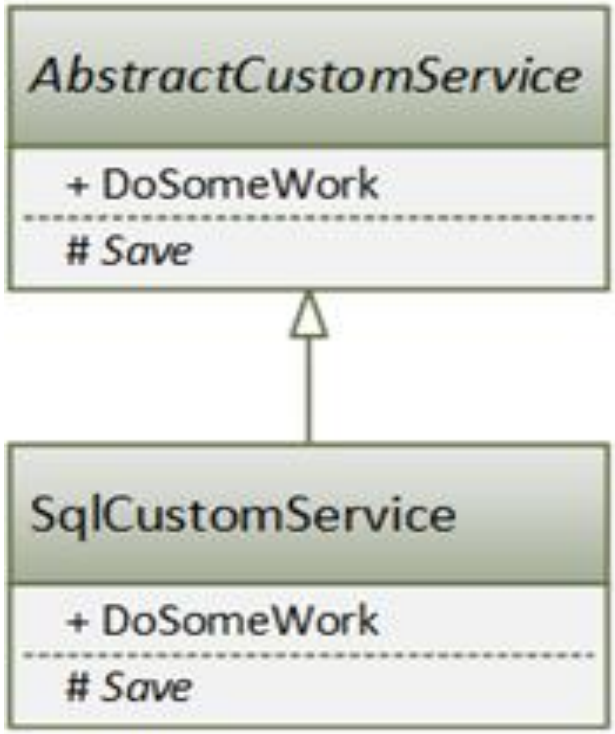
Наследование vs Композиция vs Агрегация

```
public class CustomService
{
        private readonly CustomRepository _repository;

        public CustomService()
        {
           //......
        }
        public void DoSomething()
        {
            // .....
        }
         public void SaveCustomRehjsitory()
        {
            // .....
        }
}
```

Наследование vs Композиция vs Агрегация

Открытого наследования (отношение «является», **IS A Relationship)** – является одним из типов отношений и гласит что все, что справедливо для базового класса справедливо и для его наследника.



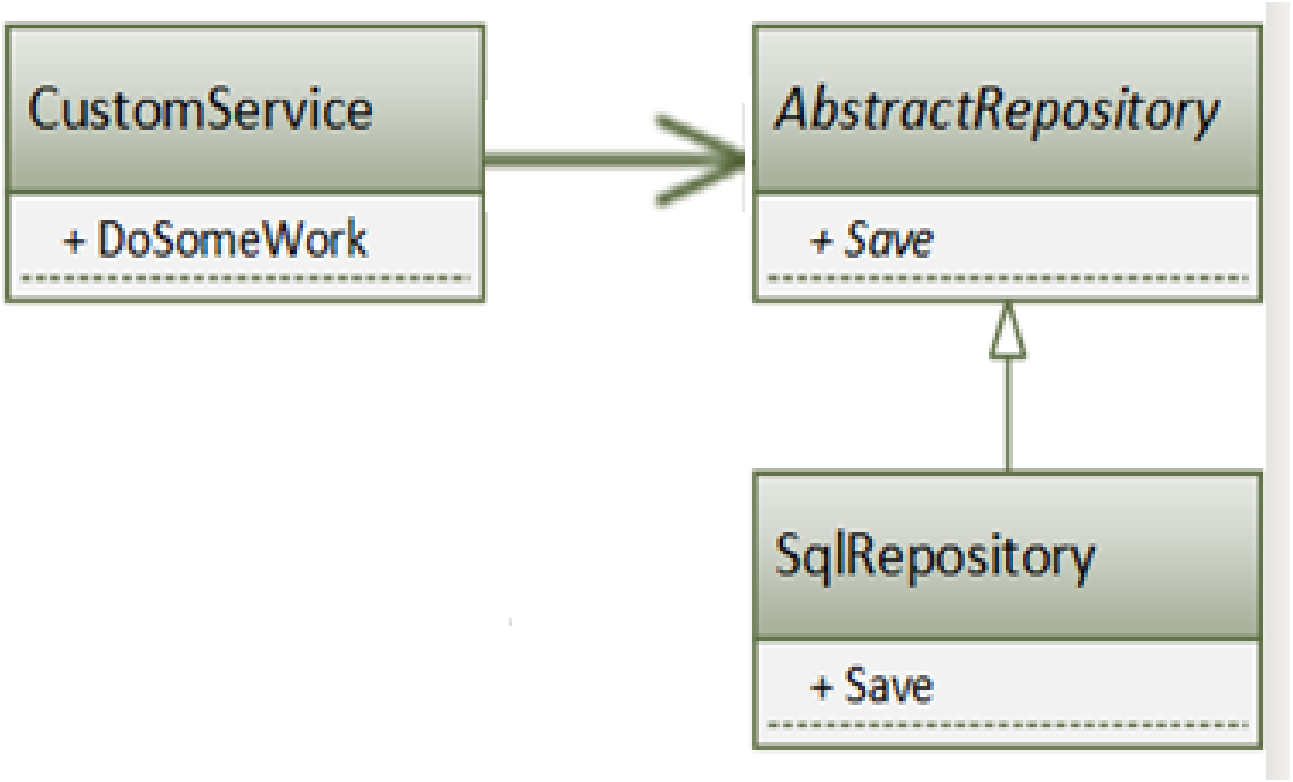
Наследование vs Композиция vs Агрегация

- 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)

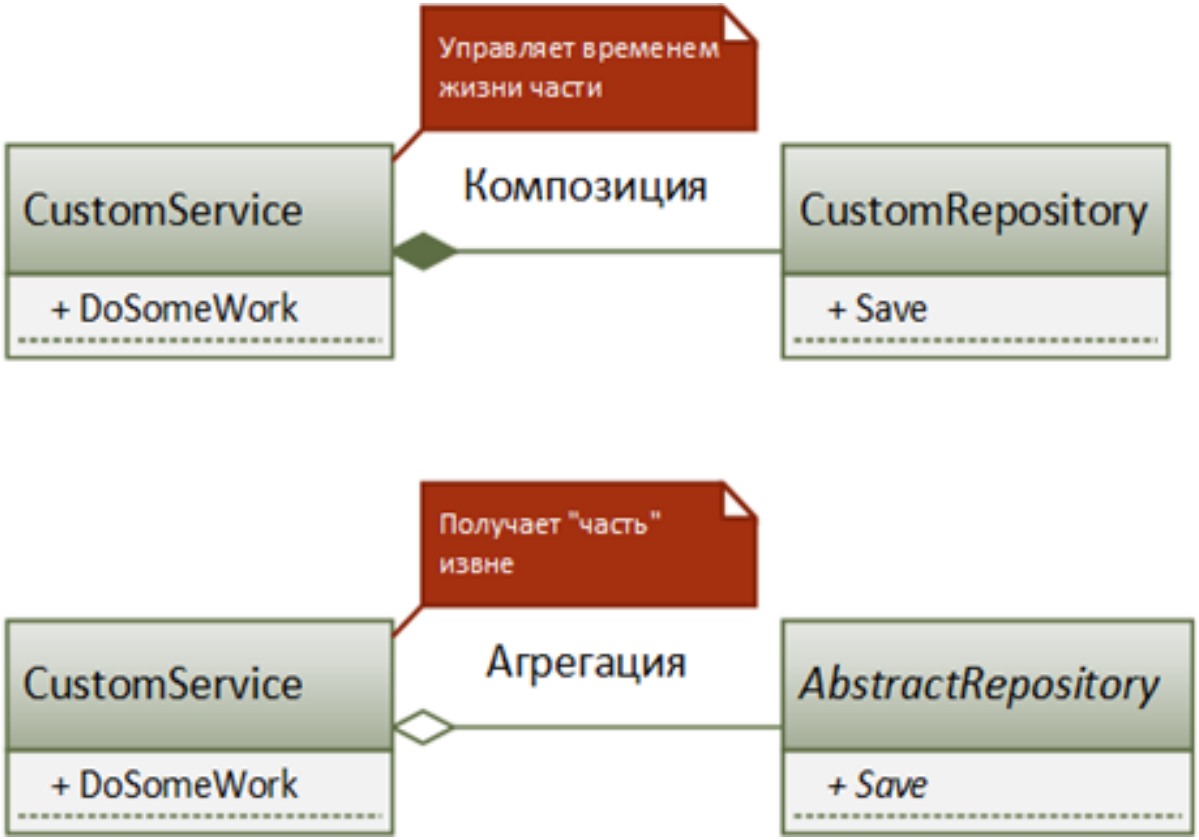
Наследование vs Композиция vs Агрегация

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



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

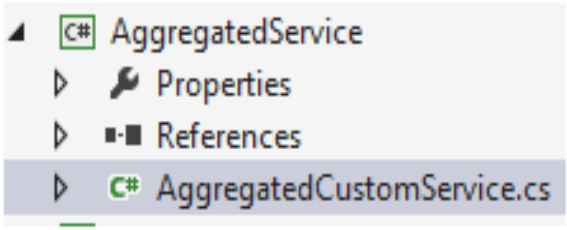


Наследование vs Композиция vs Агрегация

В случае агрегации целое хоть и содержит свою составную часть, время их жизни не связано(например, составная часть передается через параметры конструктора)

```
namespace AggregatedService
{
    ссылок 3
    public class AggregatedCustomService
    {
        private readonly AbstractRepository _repository;
        ссылка 1
        public AggregatedCustomService(AbstractRepository repository)
        { _repository = repository; }

        ссылка 1
        public void DoSomething()
        {
            // Используем _repository
            _repository.Save();
        }
    }
}
```




```
static void Main(string[] args)
{
    AggregatedCustomService custom =
        new AggregatedCustomService(new SqlRepository());

    custom.DoSomething();
}
```

Наследование vs Композиция vs Агрегация

В случае композиции целое явно контролирует время жизни своей составной части (часть не существует без целого)

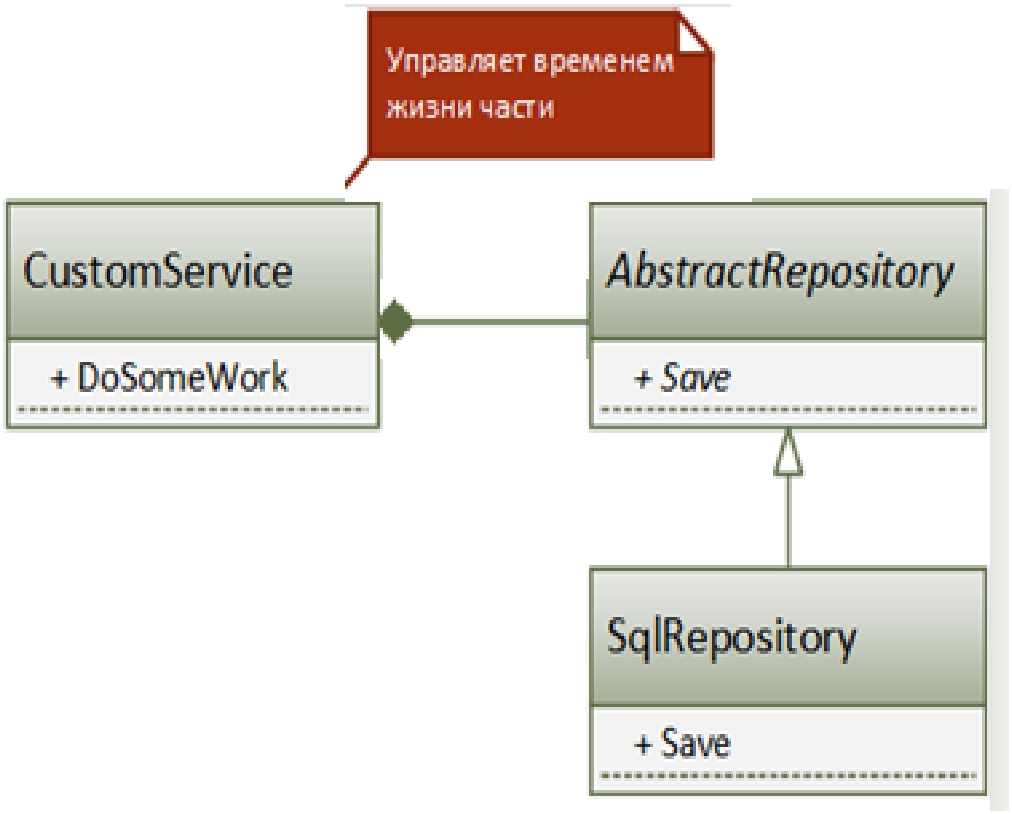
```
class CompositeCustomService
{
    private readonly CustomRepository _repository = new CustomRepository();

    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 (фабричный метод или абстрактная фабрика)



Наследование vs Композиция vs Агрегация

Factory Method

Фабричный метод - паттерн, порождающий классы. *Известен также под именем* виртуальный конструктор (Virtual Constructor).

Определяет интерфейс для создания объектов, при этом объект класса создается с помощью методов подклассов.

Используется при решении следующих задач:

1. проект должен быть ***расширяемым***, иметь возможность добавления объектов новых типов или замены объектов одного типа на другой, и ***независимым*** как от самого процесса порождения объектов, так и от их типов.

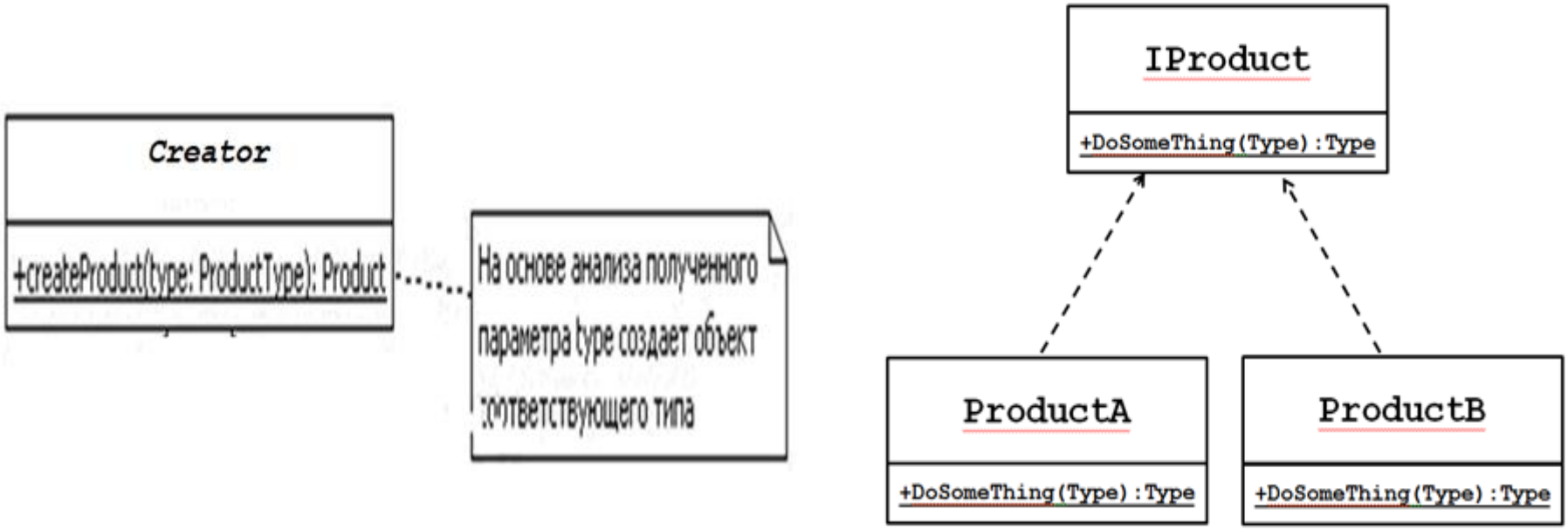
2. **заранее известно, когда нужно создавать объект, но неизвестен его тип**.

Наследование vs Композиция vs Агрегация

Два способа описания фабричного метода
1. с помощью ***статического метода***, который получает идентификатор типа создаваемого объекта.
2. с помощью ***интерфейса или абстрактного класса в котором содержится абстрактный метод***, и ***классы которые реализуют данный интерфейс*** или наследуются от абстрактного класса, должны реализовать данный метод.

Наследование vs Композиция vs Агрегация

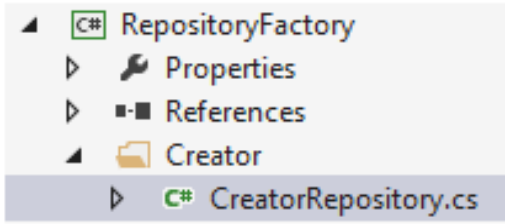
Описания фабричного метода с помощью ***статического метода***, который получает идентификатор типа создаваемого объекта.

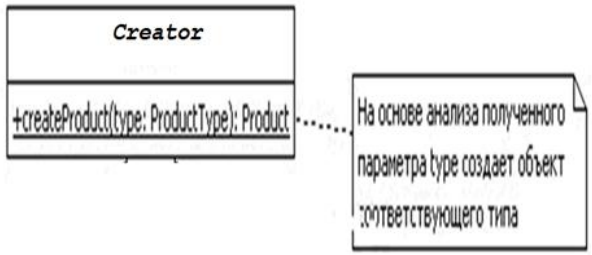


Наследование vs Композиция vs Агрегация

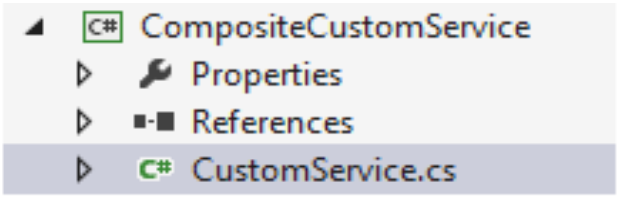
```
namespace RepositoryFactory.Creator
{
    ссылок 3
    public enum RepositoryEnum
     {
         SqlRepository, XmlRepository, CollectionRepository
     }

    ссылок 0
    public static class CreatorRepository
     {
         ссылок 0
         public static AbstractRepository Create(RepositoryEnum repository)
         {
             switch (repository)
             {
                 case RepositoryEnum.SqlRepository:
                     return new SqlRepository();
                 // case RepositoryEnum.CollectionRepository:
                //     return new CollectionRepository();
                 // case RepositoryEnum.XmlRepository:
                //     return new XmlRepository();
                   default: return new SqlRepository();
             }
         }
     }
}
```





Наследование vs Композиция vs Агрегация




```
namespace CompositeCustomService
{
    ссылок 3
    public class CustomService
    {
        private readonly AbstractRepository _abstractRepository;

        ссылка 1
        public CustomService(RepositoryEnum repositoryEnum)
        {
            _abstractRepository = CreatorRepository.Create(repositoryEnum);
        }
        ссылка 1
        public void DoSomething()
        {
            _abstractRepository.Save();
        }

    }
}
```

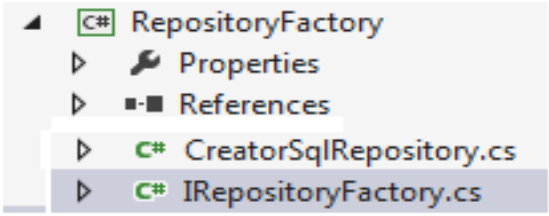
```
CustomService custom =
    new CustomService(RepositoryEnum.SqlRepository);
custom.DoSomething();
```

Наследование vs Композиция vs Агрегация

Классическая реализация фабричного метода с помощью ***интерфейса*** *и классы реализующие данный интерфейс.*

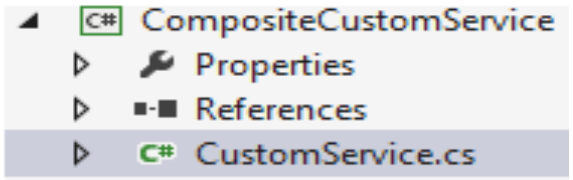
```
namespace RepositoryFactory
{
    ссылок 3
    public interface IRepositoryFactory
    {
        ссылок 2
        AbstractRepository Create();
    }

}
```




```
namespace RepositoryFactory
{
    ссылок 0
    public class CreatorSqlRepository : IRepositoryFactory
    {
        ссылок 2
        public virtual AbstractRepository Create()
        {
            return new SqlRepository();
        }
    }
}
```

Наследование vs Композиция vs Агрегация




```
namespace CompositeCustomService
{
    ссылок 3
    public class CustomService
    {
        private readonly IRepositoryFactory _repositoryFactory;

        ссылка 1
        public CustomService(IRepositoryFactory repositoryFactory)
        {
            _repositoryFactory = repositoryFactory;
        }
        ссылка 1
        public void DoSomething()
        {
            var repository = _repositoryFactory.Create();
            // Используем созданный AbstractRepository
            repository.Save();
        }
    }
}
```

```
CustomService custom =
    new CustomService(new CreatorSqlRepository());
custom.DoSomething();
```

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

СПАСИБО ЗА ВНИМАНИЕ!

ВОПРОСЫ?

NET.C#.08

Наследование vs Композиция vs Агрегация и SOLID

Author: **Саркисян Гаяне Феликсовна**
gayane.f.sarkisyan@gmail.com