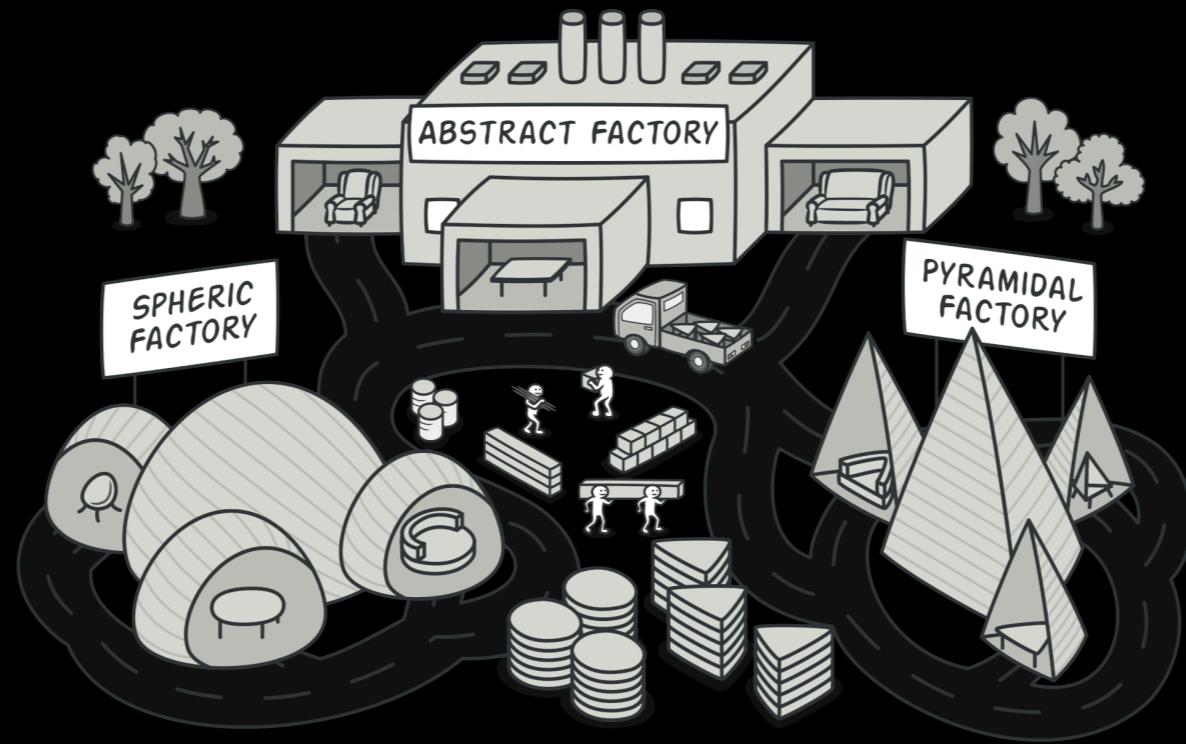


# Abstract Factory

Абстрактная Фабрика



Абстрактная фабрика (Abstract factory) – порождающий шаблон проектирования, предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов.

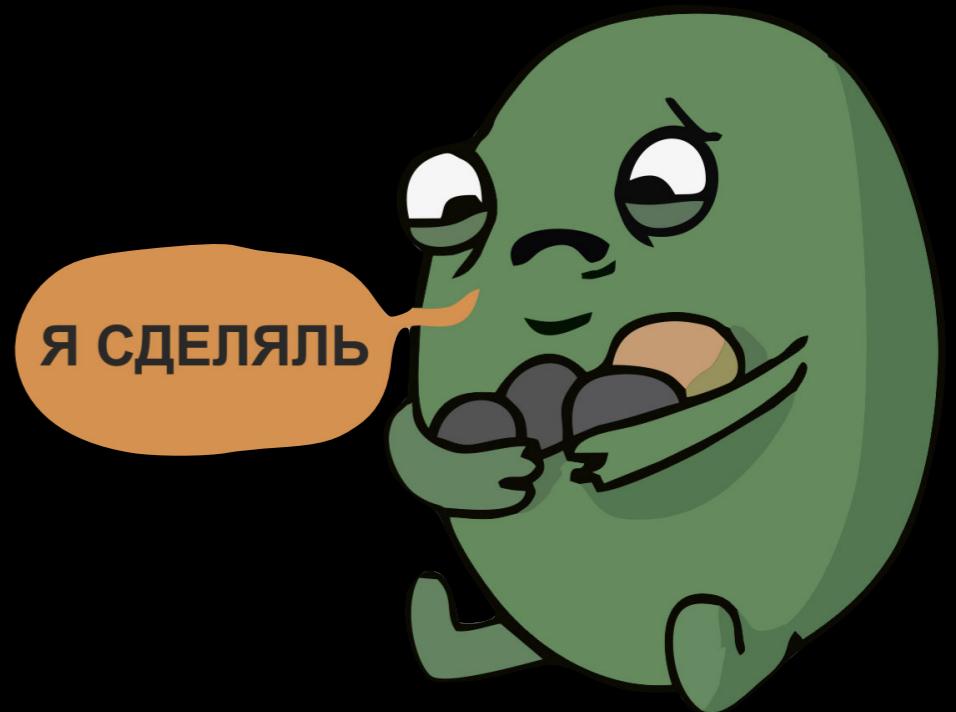
Шаблон реализуется созданием абстрактного класса `AbstractFactory`, который представляет собой интерфейс для создания компонентов системы.

Затем пишутся классы, реализующие этот интерфейс

# Основные случаи использования:

- Входящие в семейство взаимосвязанные объекты должны использоваться вместе и вам необходимо обеспечить выполнение этого ограничения
- Требуется предоставить библиотеку объектов, раскрывая только их интерфейсы, но не классы
- Необходимость слабой связанности классов
- Сложная логика создания связанных объектов

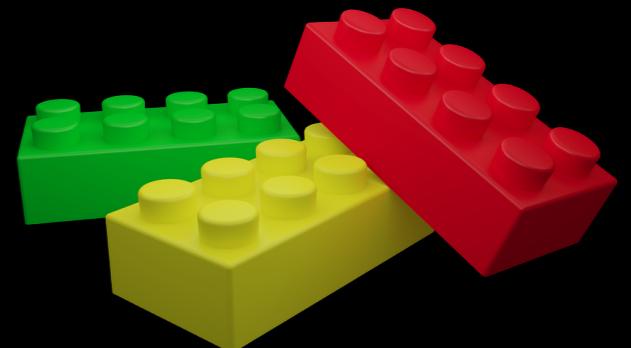
Abstract Factory - порождающий шаблон проектирования



# ПЛЮСЫ И МИНУСЫ

## Плюсы

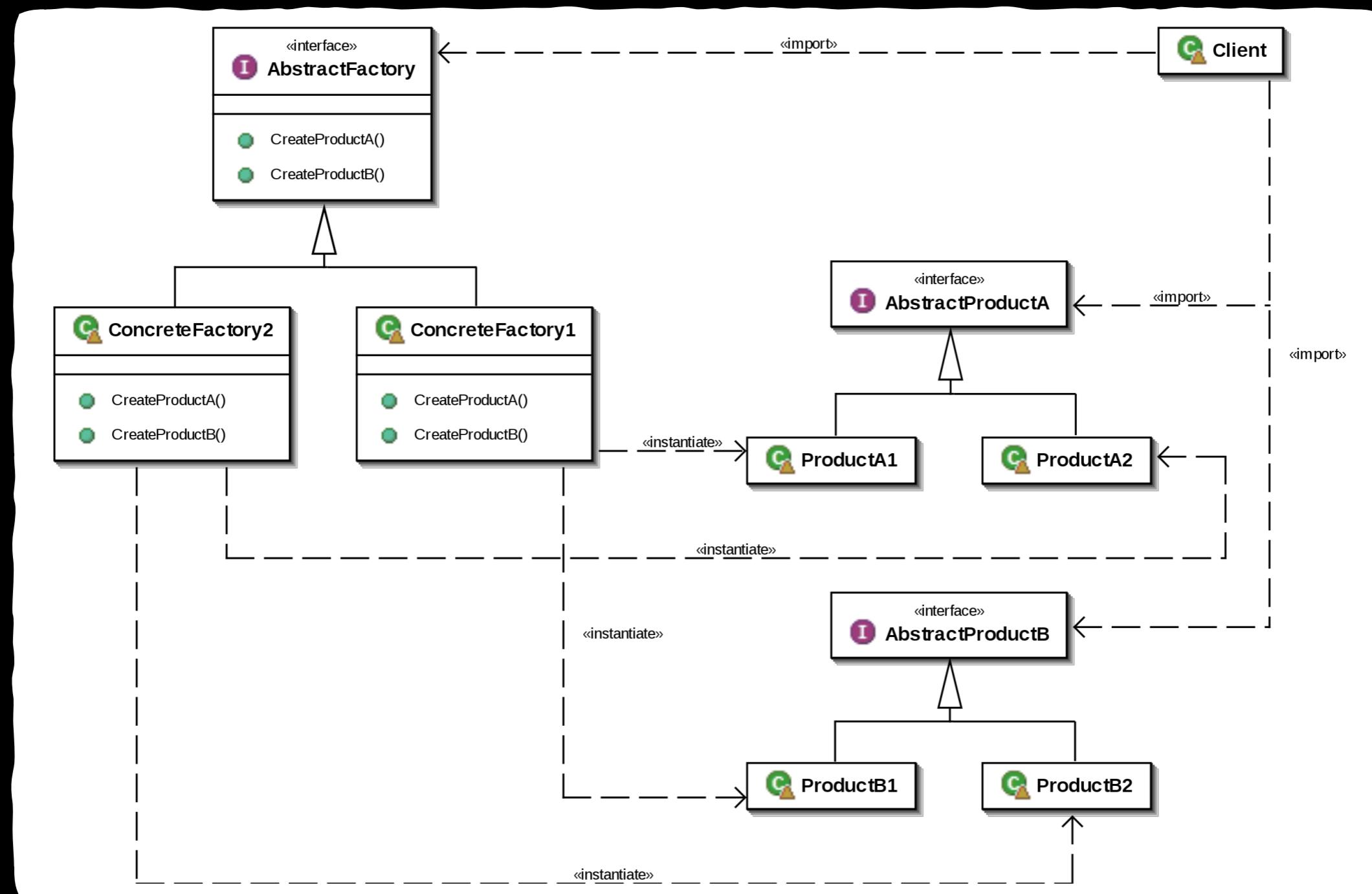
- Сильная изоляция конкретных классов
- Упрощение замены реализации
- Гарантия совместимости продуктов фабрики



## Минусы

- Сложность добавление новых видов продуктов
- Сложность добавления нового свойства в продукт фабрики

# Графическое представление



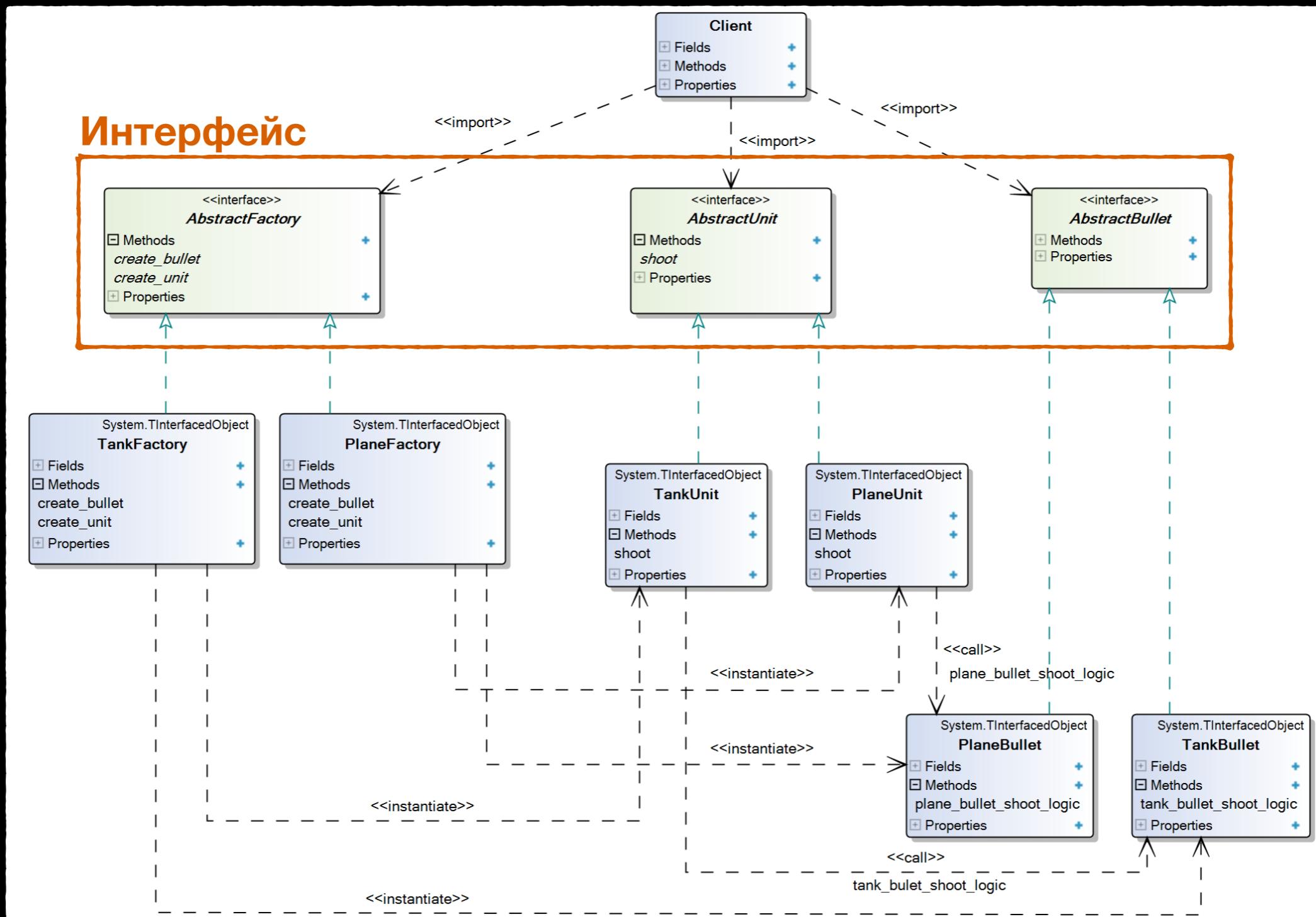
# Пример использования



# Задача

- Создать систему классов, позволяющую с помощью **общего** абстрактного интерфейса создавать объекты одного вида.
- Объекты одного вида должны иметь возможность взаимодействовать друг с другом.
- Необходима низкая связанность модулей

# UML диаграмма



# Абстрактные интерфейсы

## **abstract\_factory.py**

```
# класс абстрактной фабрики, создает технику и совместимые пули/снаряды
class AbstractFactory:
    # создание техники
    def create_unit(self):
        pass

    # создание пули/снаряда
    def create_bullet(self):
        pass

# класс абстрактной пули/снаряда
class AbstractBullet:
    pass

# класс абстрактной техники
class AbstractUnit:
    def shoot(self, bullet):
        pass
```

# Реализация фабрики танков

## tank\_factory.py

```
from abstract_factory import *

class TankBullet(AbstractBullet):
    def tank_bullet_shoot_logic(self):
        print('>> TankBullet.tank_bullet_shoot_logic: shoot logic')

class TankUnit(AbstractUnit):
    def shoot(self, bullet):
        print('>> TankUnit: shoot')
        bullet.tank_bullet_shoot_logic()

class TankFactory(AbstractFactory):
    def create_unit(self):
        print('>> TankFactory: new TankUnit')
        return TankUnit()

    def create_bullet(self):
        print('>> TankFactory: new TankBullet')
        return TankBullet()
```

# Реализация фабрики самолетов

## plane\_factory.py

```
from abstract_factory import *

class PlaneBullet(AbstractBullet):
    def plane_bullet_shoot_logic(self, power):
        print('>> PlaneBullet.plane_bullet_shoot_logic: shoot'
logic with power = ' + str(power))

class PlaneUnit(AbstractUnit):
    def shoot(self, bullet):
        print('>> PlaneUnit: shoot')
        bullet.plane_bullet_shoot_logic(42)

class PlaneFactory(AbstractFactory):
    def create_unit(self):
        print('>> PlaneFactory: new PlaneUnit')
        return PlaneUnit()

    def create_bullet(self):
        print('>> PlaneFactory: new PlaneBullet')
        return PlaneBullet()
```

# Использование

```
from abstract_factory import *
from tank_factory import *
from plane_factory import *

def new_unit_with_shoot(factory):
    # создаем юнит
    unit = factory.create_unit()
    # создаем пулю/снаряд
    bullet = factory.create_bullet()
    # стреляем
    unit.shoot(bullet)

    print('*** Tank ***')
    # создаем фабрику танков и etc.
    factory_for_tank = TankFactory()
    # создаем танк, снаряд, и стреляем
    new_unit_with_shoot(factory_for_tank)

    print('*** Plane ***')
    # создаем фабрику самолетов и etc.
    factory_for_plane = PlaneFactory()
    # создаем самолет, пулю, и стреляем
    new_unit_with_shoot(factory_for_plane)
```

\*\*\* Tank \*\*\*  
>> TankFactory: new TankUnit  
>> TankFactory: new TankBullet  
>> TankUnit: shoot  
>> TankBullet.tank\_bullet\_shoot\_logic: shoot logic  
\*\*\* Plane \*\*\*  
>> PlaneFactory: new PlaneUnit  
>> PlaneFactory: new PlaneBullet  
>> PlaneUnit: shoot  
>> PlaneBullet.plane\_bullet\_shoot\_logic: shoot logic with power = 42

QA