



Модули 1, 2

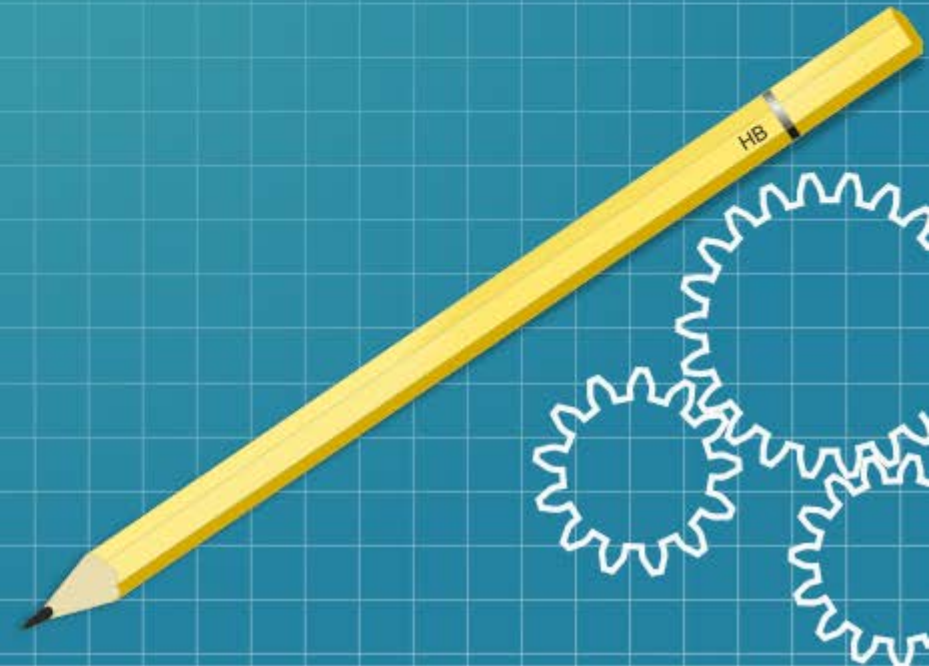
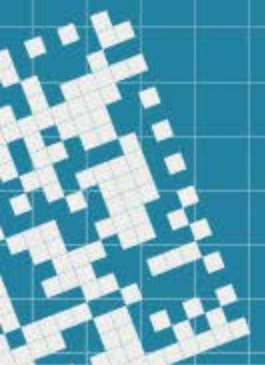
- Введение в паттерны проектирования
- Порождающие паттерны (часть 1)

Содержание (цели)



- **Модуль 1 – Введение в паттерны проектирования**
 - Понятие паттерна проектирования
 - Принципы применения, выбора и разделения паттернов проектирования
 - Использование UML при анализе паттернов проектирования
- **Модуль 2 – Порождающие паттерны (Часть 1)**
 - Понятие порождающего паттерна
 - Abstract Factory
 - Практический пример использования паттерна Abstract Factory

Модуль 1 – Введение в паттерны проектирования



Тенденции в развитии паттернов



197

- Идея паттернов пришла из архитектуры → Кристофер Александр

7

198

- Шаблоны разработки программного обеспечения для графических оболочек на языке Smalltalk - Кент Бэк, Вард Каннигем

7

198

- Докторская диссертация о паттернах в ПО - Эрих Гамма

8

199

- Приёмы объектно-ориентированного проектирования.
Паттерны проектирования (*Design Patterns: Elements of Reusable Object-Oriented Software*) - Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес – “Банда четырех” - “**Gang of Four**” - “**GoF**”

4

...

Enterprise Patterns

xUnit Test Patterns

Inversion of Control

MVC MVP MVVM

...

...

...

Понятие паттерна



Общее понятие:

Паттерн – образец решения для схожих ситуаций

Алгоритмы процедурного программирования

=

паттерны вычислений


≠

паттерны проектирования

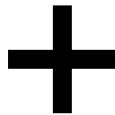
Идиомы – паттерны типичных решений на конкретном языке

Паттерны проектирования – не зависят от языка
программирования !!!

Причины возникновения паттернов проектирования



В конце 80-х годов XX века - много различных а по сути однотипных решений в ОО-проектирования



Упорядочение знаний в ОО-проектирования



Решение проблемы систематизации накопленного опыта в объектно-ориентированном проектировании

Понятие паттерна проектирования

«паттерн» (*pattern*)

«шаблон» (*template*)



«образец»

Кристофер Александр: «... любой паттерн описывает задачу, которая снова и снова возникает в нашей работе, а также принцип её решения, причем таким образом, что решение можно использовать миллион раз, ничего не изобретая заново...» [GoF95]

Понятие паттерна в ОО-проектировании




Под паттерном проектирования (design pattern) будем понимать описание взаимодействия объектов и классов, адаптированных для решения общей задачи проектирования в конкретном контексте [GoF95].

Составляющие паттерна



- **Имя** – уникальный идентификатор паттерна. Как правило общепринятое
- **Задача** – описывает ситуацию и контекст в которой можно применить паттерн
- **Решения** - определяет общие функции каждого элемента дизайна и отношения между ними
- **Результаты** - описываются преимущества и недостатки выбранного решения, его последствия, различного рода компромиссы, вариации паттерна

Принципы применения паттернов проектирования



Дизайн системы на первом месте, **ПОТОМ** паттерны !!!

Пример проявления тропинки в парке, которую ПОТОМ покрывают тротуарной плиткой :)

Базовые принципы при разработке дизайна:

- всегда формировать простой дизайн
- слабая зависимость между фрагментами системы

По мере проектирования логической структуры, можно увидеть типичные задачи решаемые с помощью паттернов

Классификация паттернов в ООАП



- **Архитектурные паттерны** – описывают фундаментальные способы структурирования программных систем
- **Паттерны анализа** – описывают общие схемы организации процесса объектно-ориентированного моделирования
- **Паттерны проектирования** – описывают структуру программных систем в терминах классов
- **Паттерны тестирования** – описывают общие схемы организации процесса тестирования программных систем
- **Паттерны реализации** – описывают шаблоны, которые используются при написании программного кода

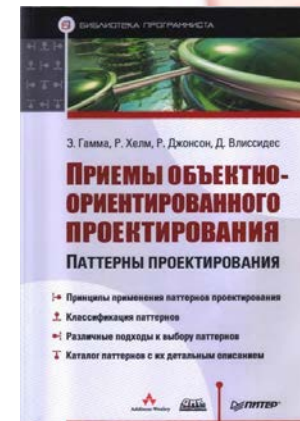
ООАП – объектно-ориентированный анализ и проектирование

Классификация паттернов проектирования GoF

Порождающие паттерны (Creational Patterns) – Абстрагируют процесс инстанцирования; делают систему независимой от того, как в ней создаются, компонуются и представляются объекты

Структурные паттерны (Structural Patterns) – Решают вопрос о создании из классов и объектов более крупных структур

Поведенческие паттерны (Behavioral Patterns) – Распределяют обязанности между объектами; описывают способы их взаимодействия.



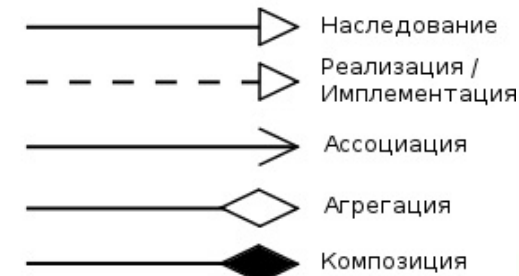
Цель Уровень	Порождающие паттерны	Структурные паттерны	Паттерны поведения
Класс	Фабричный метод	Адаптер (класса)	Интерпретатор Шаблонный метод
Объект	Абстрактная фабрика Одиночка Прототип Строитель	Адаптер (объекта) Декоратор Заместитель Компоновщик Мост Приспособленец Фасад	Итератор Команда Наблюдатель Посетитель Посредник Состояние Стратегия Хранитель Цепочка обязанностей

Паттерны уровня классов

описывают отношения между классами и их подклассами. Эти отношения выражаются с помощью статических связей – **наследования и реализации**.

Паттерны уровня объектов

описывают взаимодействия между объектами. Эти отношения выражаются с помощью динамических связей – **ассоциации, агрегации и композиции**.



Пример паттерна проектирования

Название паттерна:

Abstract Factory/Абстрактная фабрика

Также известный под именем: Toolkit/Инструментарий

Цель паттерна:

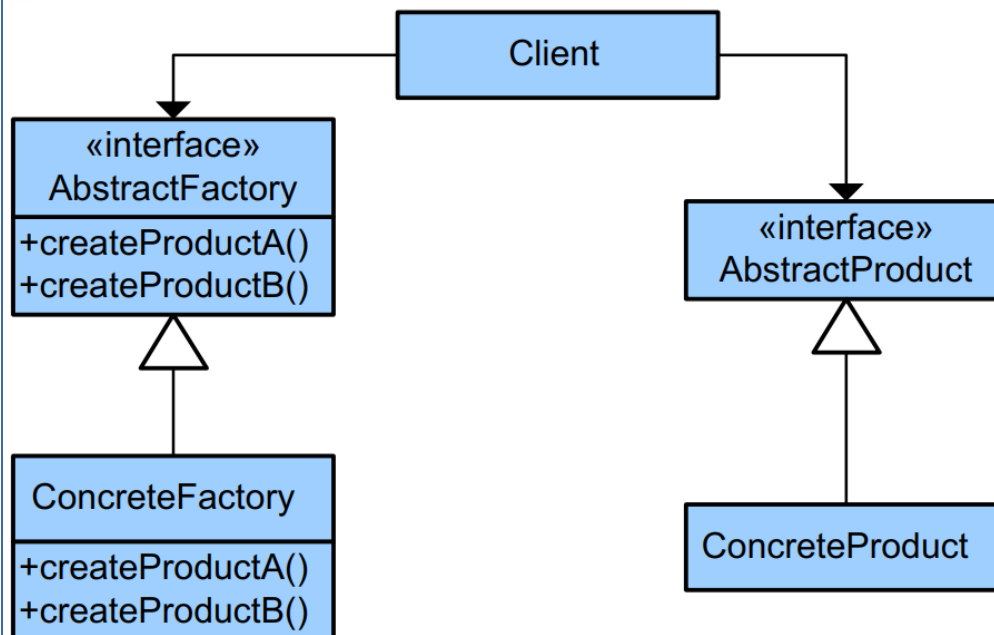
Предоставляет интерфейс для создания семейства взаимосвязанных и взаимозависимых объектов, не специфицируя конкретных классов, объекты которых будут создаваться. [GoF95]

Паттерн следует использовать когда...

...

Причины возникновения паттерна

...



```
class Client
{
    public void Main()
    {
        // Клиентский код может работать с любым конкретным классом
        // фабрики.
        Console.WriteLine("Client: Testing client code with the first
        ClientMethod(new ConcreteFactory1());
        Console.WriteLine();

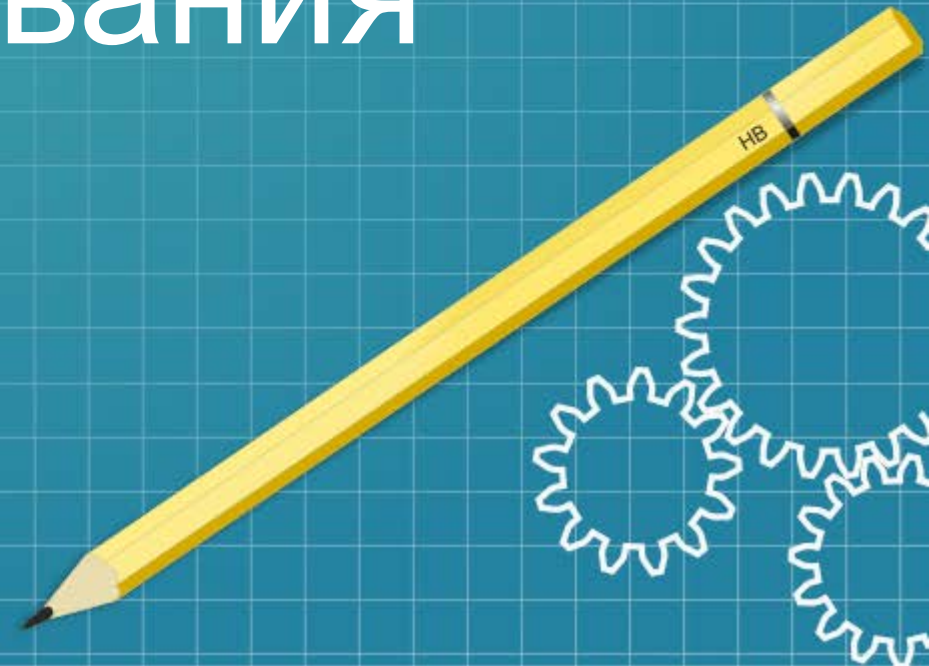
        Console.WriteLine("Client: Testing the same client code with the
        ClientMethod(new ConcreteFactory2());
    }

    public void ClientMethod(IAbstractFactory factory)
    {
        IAbstractProductA productA = factory.CreateProductA();
        IAbstractProductB productB = factory.CreateProductB();

        Console.WriteLine(productB.UsefulFunctionB());
        Console.WriteLine(productB.AnotherUsefulFunctionB(productA));
    }
}

class Program
{
    static void Main(string[] args)
    {
        new Client().Main();
    }
}
```

Использование UML в анализе паттернов проектирования



Унифицированный язык моделирования (Unified Modelling Language, UML)

Появился с 1989 по 1997 год.

является графическим языком для визуального представления, составления спецификаций, проектирования и документирования систем, в которых большая роль принадлежит программному обеспечению.

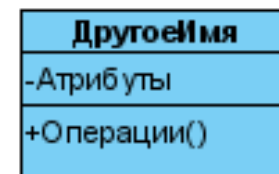
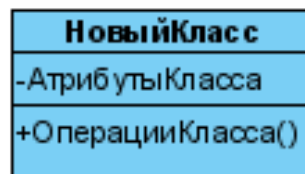
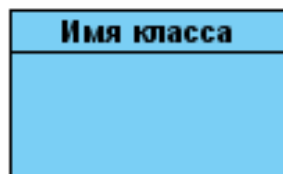
Гради БУЧ.

Диаграмма классов

*предназначена для представления
статической структуры системы в
терминах классов объектно-
ориентированного программирования*

Класс (class) в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов.

Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции:



Секции:

- 1) имя класса
- 2) атрибуты (поля, члены данных) класса (могут отсутствовать)
- 3) операции (методы, функции-члены) класса (могут отсутствовать)

Аттрибуты классов



[visibility] **name** [multiplicity] [: type] [= initial value] [{ property }]

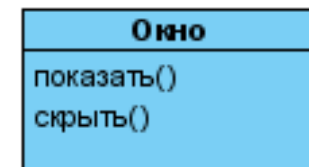
- **visibility** — область видимости, которая может получать следующие значения:
- (+) — Public
- (#) — Protected
- (−) — Private
- (~) — Package
- **name** — имя атрибута (единственный обязательный параметр);
- **multiplicity** — кратность, т.е. количество экземпляров атрибута;
- **type** — тип, к которому принадлежит атрибут;
- **initial value** — начальное значение;
- **property** — другие свойства, например, readonly.

Прямоугольник
p1 : Point
p2 : Point

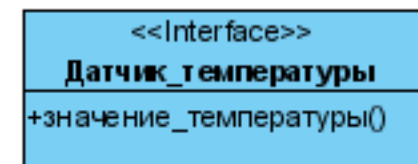
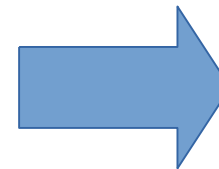
Операции классов и интерфейсы

[visibility] **name** ([parameter list]) [: return type] [{ property }]

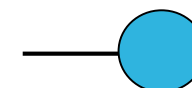
- **visibility** — область видимости, которая может получать следующие значения:
 - (+) — Public
 - (#) — Protected
 - (−) — Private
 - (~) — Package
- **name** — имя атрибута (вместе с “(“ и “)” единственный обязательный параметр)
- **parameter list** — список параметров
- **return type** — тип возвращаемого значения
- **property** — другие свойства, например, readonly.



Если обозначение класса содержит **только операции**, то такой класс является интерфейсом, обозначиться как `<<interface>>`



или



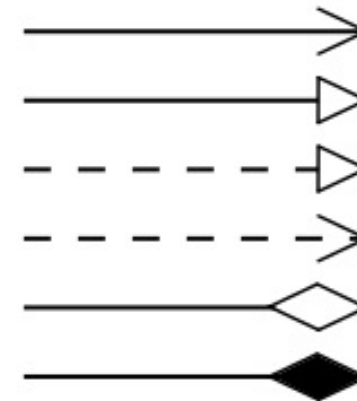
Interface name

Отношения между классами

Взаимосвязь — это особый тип логических отношений между сущностями, показанных на диаграмме классов.

В UML представлены следующие виды отношений

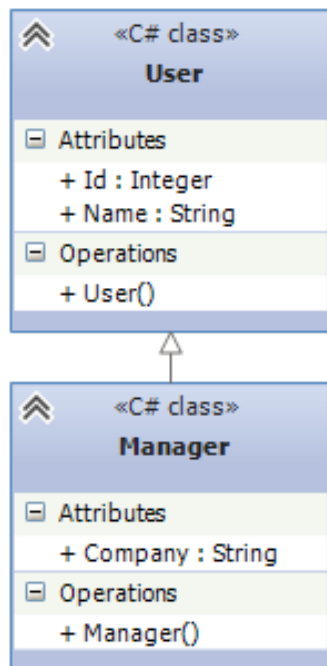
- Ассоциация
- Обобщение (наследование)
- Реализация
- Зависимость
- Агрегация
- Композиция



Отношение наследования

Наследование (генерализация или обобщение) - позволяет одному классу (наследнику) унаследовать функционал другого класса (родительского). Определяет отношение **IS A** - “является”

Пример реализации отношения



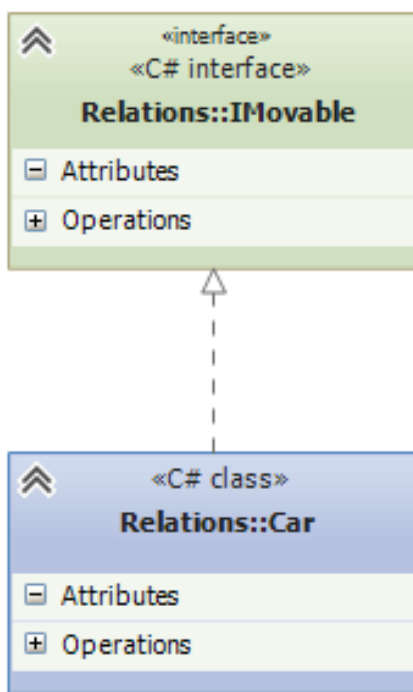
```
class User
{
    public int Id { get; set; }
    public string Name { get; set; }
}

class Manager : User
{
    public string Company { get; set; }
}
```


Отношение реализации

Реализация предполагает определение интерфейса и его реализация в классах.

Пример реализации отношения



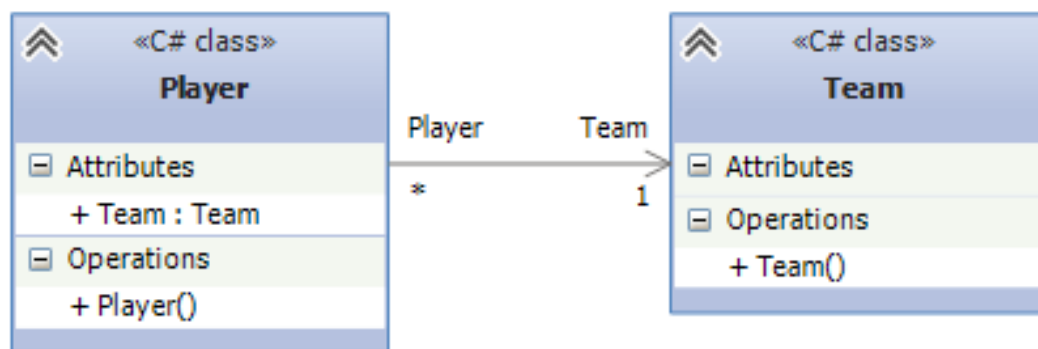
```
public interface IMovable
{
    void Move();
}

public class Car : IMovable
{
    public void Move()
    {
        Console.WriteLine("Машина едет");
    }
}
```

Отношение ассоциации

Ассоциация - это отношение, при котором объекты одного типа неким образом связаны с объектами другого типа

Пример реализации отношения

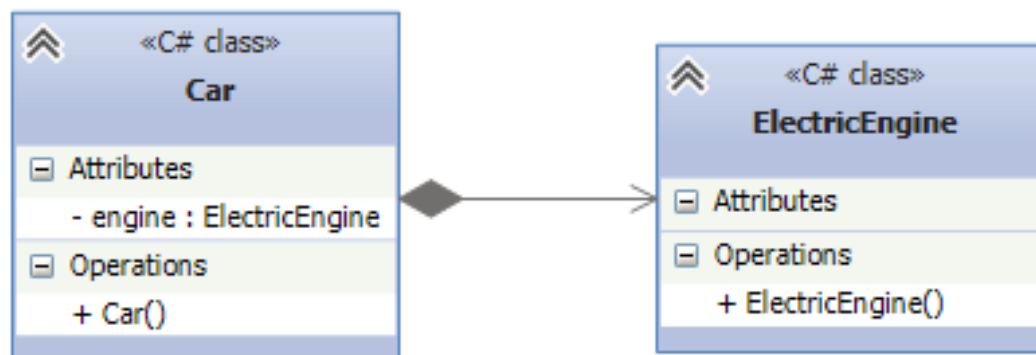


```
class Team
{
}
class Player
{
    public Team Team { get; set; }
}
```

Отношение композиции

Композиция определяет отношение **HAS A** (имеет), то есть отношение "имеет"

Пример реализации отношения



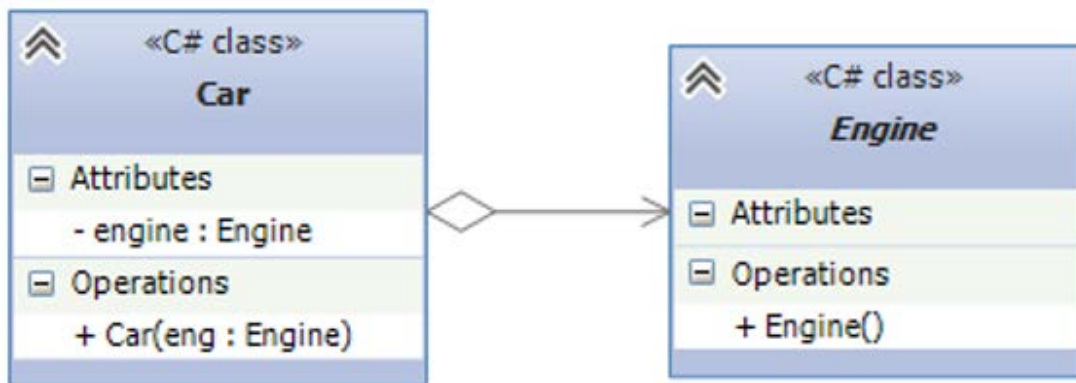
```
public class ElectricEngine
{ }

public class Car
{
    ElectricEngine engine;
    public Car()
    {
        engine = new ElectricEngine();
    }
}
```

Отношение агрегации

От композиции следует отличать **агрегацию**. Она также предполагает отношение **HAS A**, но реализуется она иначе

Пример реализации отношения



```
public abstract class Engine
{ }

public class Car
{
    Engine engine;
    public Car(Engine eng)
    {
        engine = eng;
    }
}
```

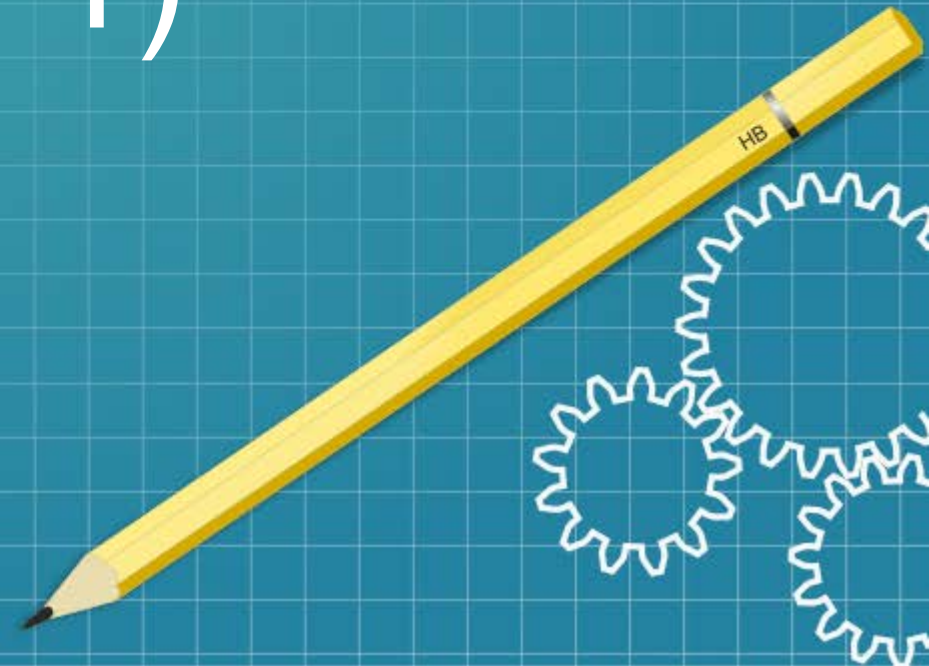

Введение в UML

Практические задания

- Написать программный код реализующий отношения
 - Наследования
 - Реализации
 - Ассоциации
 - Композиции
 - Агрегации
- Для примера можно реализовать следующие классы:
- Автомобиль – Легковой Автомобиль – интерфейс IMovable
- Растение – Цветок – интерфейс IPlant
- Компьютер – Ноутбук – интерфейс IComputable)



Модуль 2 – Порождающие паттерны (Часть 1)



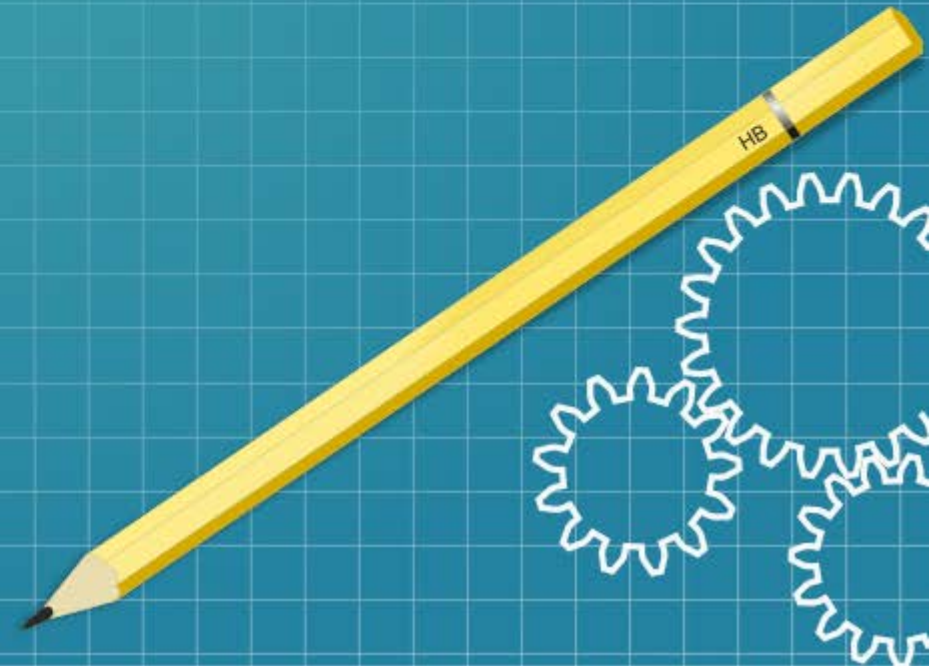
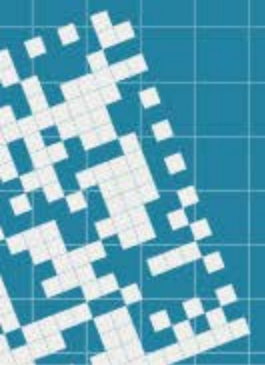
Понятие порождающего паттерна



Порождающие паттерны проектирования:

- абстрагируют процесс инстанцирования
- помогают сделать систему независимой от способа создания, композиции, и представления объектов
- позволяют ответить на вопрос: кто, когда и как создает объекты в системе.

Abstract Factory



Abstract Factory



Название паттерна - Abstract Factory/Абстрактная фабрика
Также известный под именем: Toolkit/Инструментарий.
Описан в работе [GoF95].

Цель паттерна

Предоставляет интерфейс для создания семейства взаимосвязанных и взаимозависимых объектов, не специфицируя конкретных классов, объекты которых будут создаваться.
[GoF95]

Паттерн следует использовать когда...

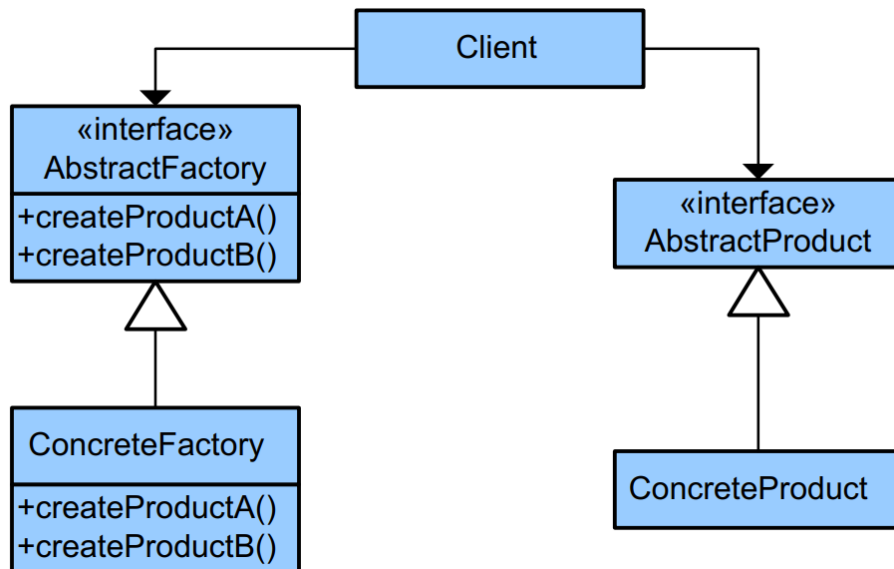
- ◆ Система не должна зависеть от того, как в ней создаются и компонуются объекты
- ◆ Объекты, входящие в семейство, должны использоваться вместе
- ◆ Система должна конфигурироваться одним из семейств объектов
- ◆ Надо предоставить интерфейс библиотеки, не раскрывая её внутренней реализации

Результаты использования паттерна

- ◆ Позволяет изолировать конкретные классы продуктов
- ◆ Упрощает замену семейств продуктов
- ◆ Дает гарантию сочетаемости продуктов
- ◆ Серьезным недостатком паттерна есть трудность поддержки нового вида продуктов

Abstract Factory

Структура
паттерна



```
class Client
{
    public void Main()
    {
        Console.WriteLine("Client: Testing client code with the first  
ClientMethod(new ConcreteFactory1());  
Console.WriteLine();

        Console.WriteLine("Client: Testing the same client code with  
ClientMethod(new ConcreteFactory2());
    }

    public void ClientMethod(IAbstractFactory factory)
    {
        IAbstractProductA productA = factory.CreateProductA();
        IAbstractProductB productB = factory.CreateProductB();

        Console.WriteLine(productB.UsefulFunctionB());
        Console.WriteLine(productB.AnotherUsefulFunctionB(productA));
    }
}

class Program
{
    static void Main(string[] args)
    {
        new Client().Main();
    }
}
```

Abstract Factory

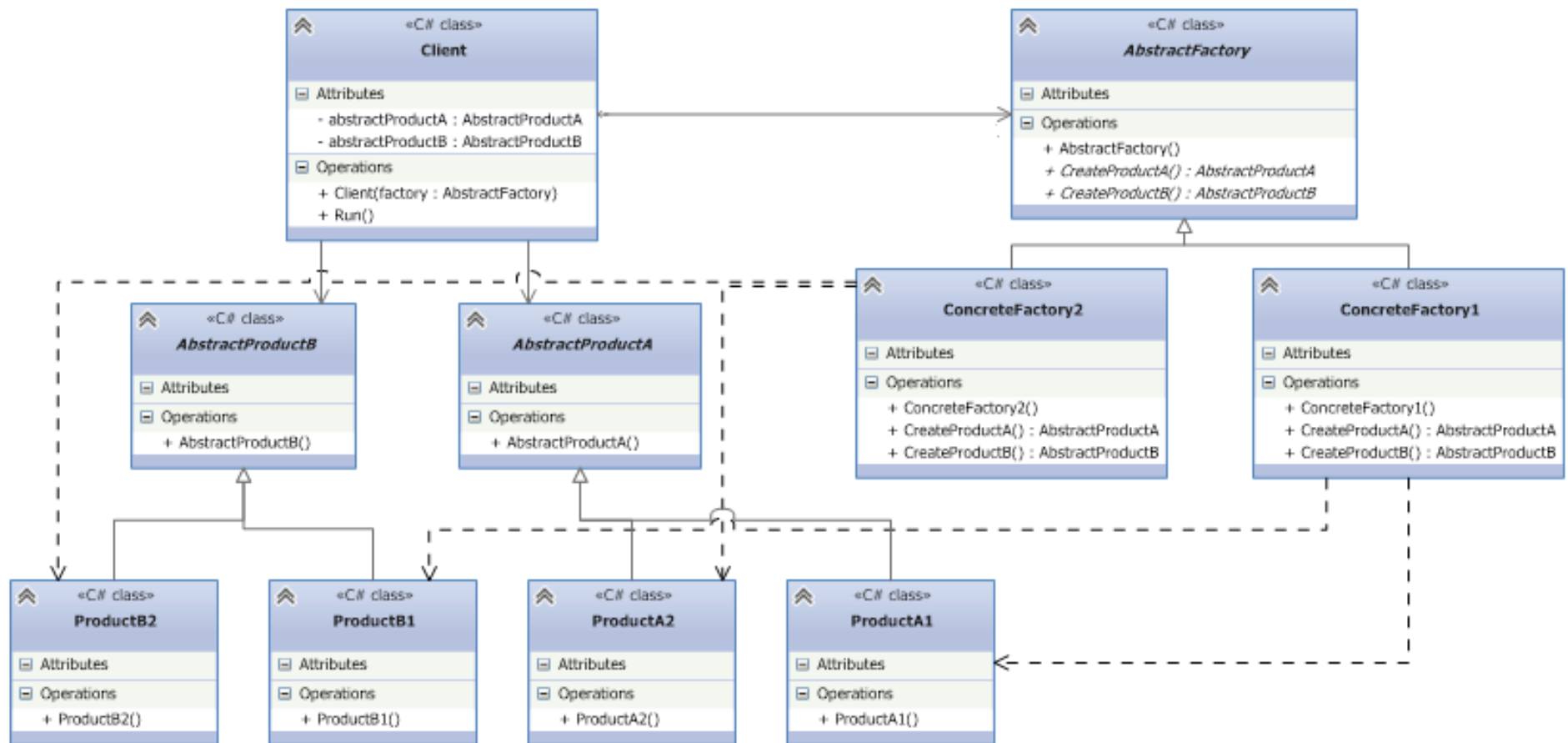


Участники

- **AbstractProduct** - Абстрактный продукт
 - представляет интерфейс абстрактного продукта, ссылку на который возвращают методы фабрик.
- **AbstractFactory** - Абстрактная фабрика
 - представляет общий интерфейс для создания семейства продуктов.
- **Client** – Клиент
 - создает и использует продукты, пользуясь исключительно интерфейсом абстрактных классов AbstractFactory и AbstractProduct и ему ничего не известно о конкретных классах фабрик и продуктов.
- **ConcreteProduct** - Конкретный продукт
 - реализует конкретный тип продукта, который создается конкретной фабрикой.
- **ConcreteFactory** - Конкретная фабрика
 - реализует интерфейс AbstractFactory и создает семейство конкретных продуктов.
 - Пример:

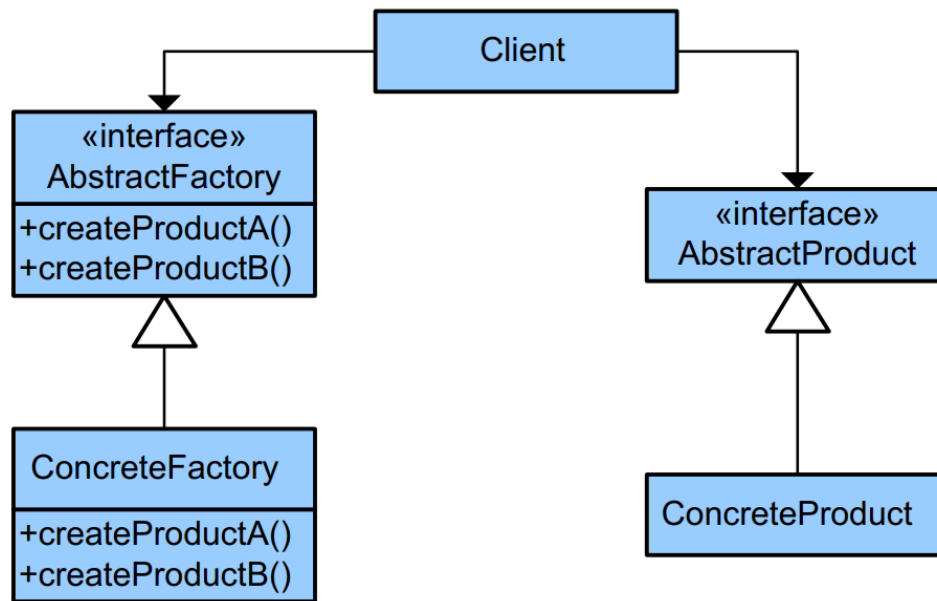
Abstract Factory

практическая схема



Лабораторная работа

Цель работы: создать работающий программный код используя данные с этого слайда



C:\Windows\system32\cmd.exe

```
Client: Testing client code with the first factory type...
The result of the product B1.
The result of the B1 collaborating with the (The result of the product A1.)

Client: Testing the same client code with the second factory type...
The result of the product B2.
The result of the B2 collaborating with the (The result of the product A2.)
Press any key to continue . . .
```

```
class Client
{
    public void Main()
    {
        Console.WriteLine("Client: Testing client code with the first
        // ClientMethod(new ConcreteFactory1());
        Console.WriteLine();

        Console.WriteLine("Client: Testing the same client code with t
        // ClientMethod(new ConcreteFactory2());
    }

    /*
    public void ClientMethod(IAbstractFactory factory)
    {
        IAbstractProductA productA = factory.CreateProductA();
        IAbstractProductB productB = factory.CreateProductB();

        Console.WriteLine(productB.UsefulFunctionB());
        Console.WriteLine(productB.AnotherUsefulFunctionB(productA));
    }
    */
}

class Program
{
    static void Main(string[] args)
    {
        new Client().Main();
    }
}
```

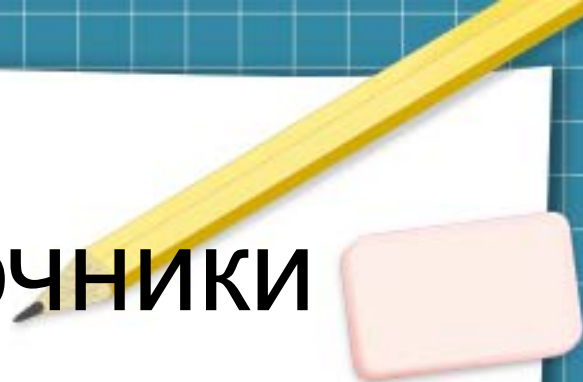
Abstract Factory

Практические задания



- Telephone
 - Производитель: Samsung, Nokia
 - Компоненты: Display, Accumulator
 - В каждом компоненте имеется метод возвращающий его описание
 - В экране реализован метод, показывающий тип установленного аккумулятора
- Car
 - Производитель: Ford, Toyota
 - Компоненты: Engine, Body
 - В каждом компоненте имеется метод возвращающий его описание
 - В кузове реализован метод показывающий тип установленного двигателя
- Computer
 - Производитель: Dell, Sony
 - Компоненты: Mainboard, Processor
 - В каждом компоненте имеется метод возвращающий его описание
 - В материнской плате реализован метод показывающий тип установленного в нее процессора

Использованные информационные источники



- 1) [GoF95] - Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. «Приемы объектно-ориентированного проектирования. Паттерны проектирования». — Спб: Питер, 2001. — 386 с.
- 2) [Grand2004] - Гранд М. «Шаблоны проектирования в Java» / М. Гранд; Пер. с англ. С. Беликовой. — М.: Новое знание, 2004. — 559 с.
- 3) [DPWiki] - [Шаблон проектирования](#)
- 4) [DPOverview] - [Обзор паттернов проектирования](#)
- 5) [DPMetanit] - [Паттерны проектирования в C# и .NET](#)