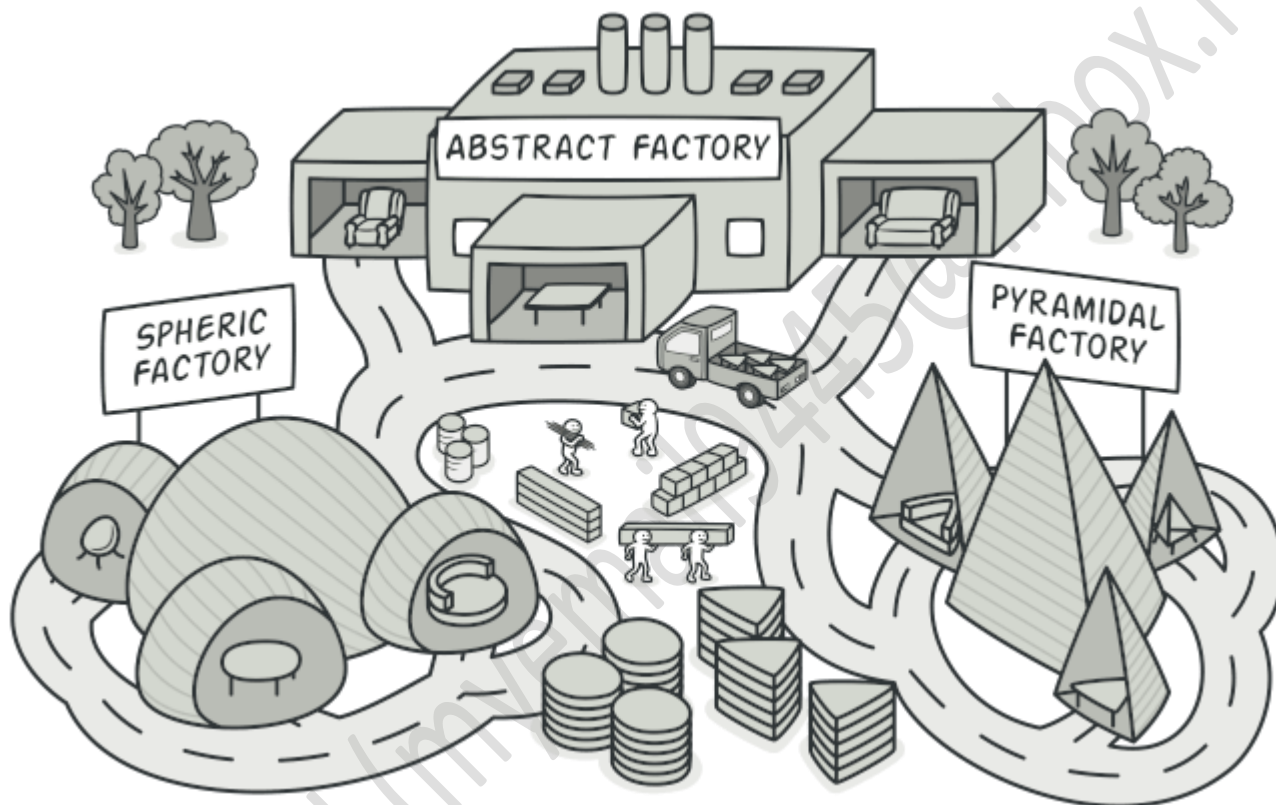


# Abstract Factory

## Maqsad:










Abstract factory – bu creational design pattern turkumiga mansub bo'lib, u sizga o'zaro bog'liq obyektlar oilasini ularning aniq klasslarini aniqlamay turib yaratish imkonini beradi.



## Muammo:

Faraz qiling, siz mebel do'koni simulyatorini yaratyapsiz. Sizning kodingiz quyidagilarni beruvchi klasslardan tashkil topgan bo'lsin deylik:

1. O'xshash mahsulotlar oilasi, aytaylik: *Chair + Sofa + CoffeeTable*
2. Shu oilaga o'xshash yana bir qancha oilalar. Masalan, *Chair + Sofa + CoffeeTable* mahsulotlari quyidagi variantlarda mavjud: *Modern, Victorian, ArtDeco*

	Chair	Sofa	Coffee Table
Art Deco			
Victorian			
Modern			

### *Mahsulot oilalari va ularning variantlari*

Sizga yakka holatdagi mebel obyektlarini yaratish yo'li kerak bo'lsin. Bunda siz yaratayotgan yakka holdagi obyektlar shu oiladagi boshqa obyektlar bilan solishtirasiz. Xaridorlar esa o'zaro mos kelmaydigan mebel olishni xohlashmaydi.

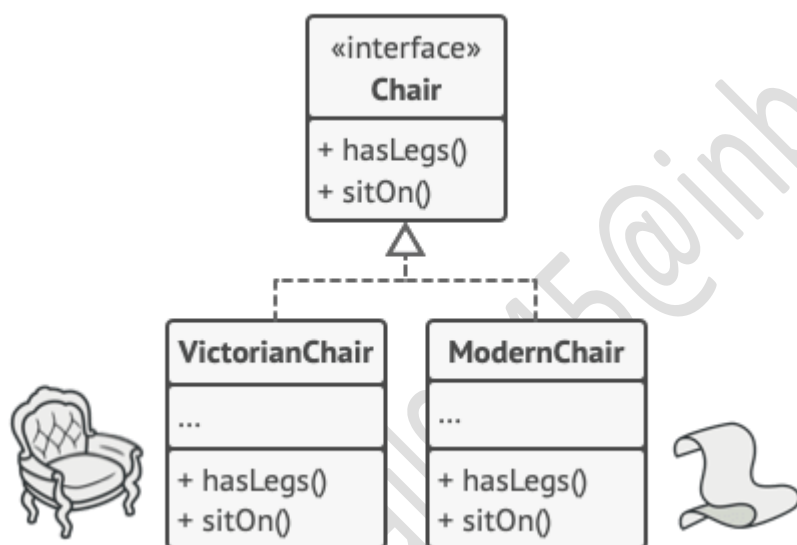


### *Modern usulidagi divan Victorian usulidagi stulga mos kelmaydi*

Shu bilan birga siz dasturingizga yangi mahsulotlar yoki mahsulotlar oilalarini qo'shish paytida kodingizni o'zgartirishni ham xohlamaysiz. Mebel sotuvchilari o'zlarining kataloglarini tez-tez yangilab turishadi va har safar siz kodingizni o'zgartirishingiz kelmaydi.

## Yechim

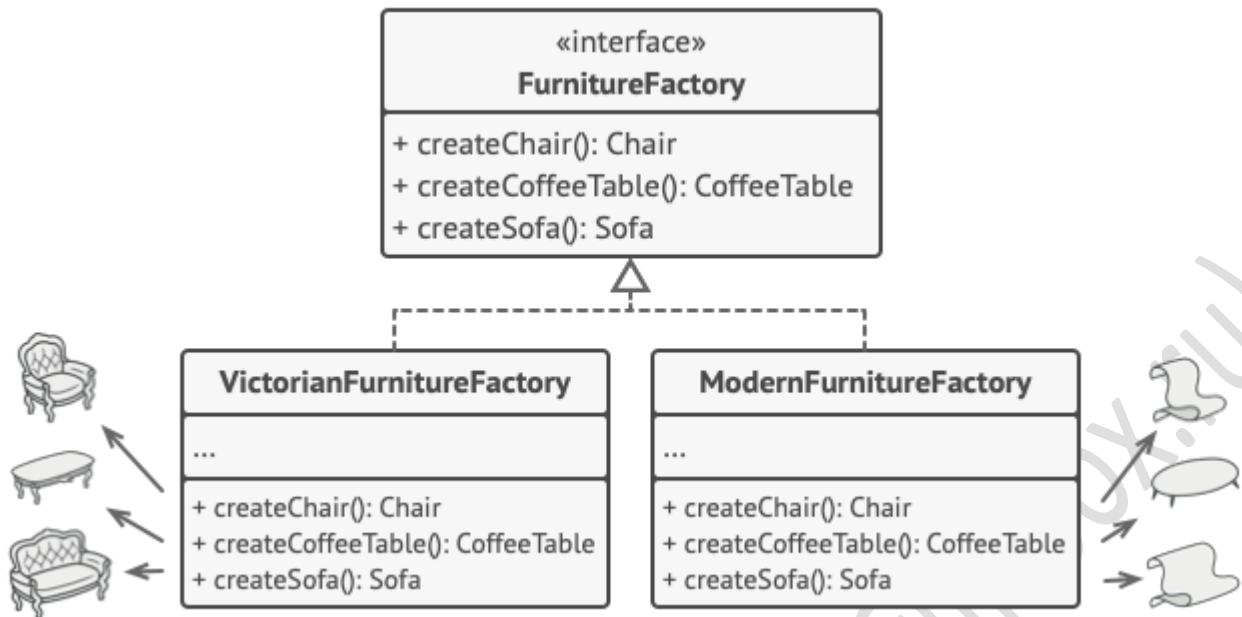
Abstract factory taklif qiladigan birinchi narsa bu mahsulot oilasidagi har bir alohida mahsulot uchun alohida interfeys e'lon qilishdir(m: stul (chair), divan (sofa) yoki kofe stoli (coffee table)). Bu holatda siz mahsulotlarning barcha variantlarini shu interfeyslar bilan ishlatishingiz mumkin bo'ladi. Masalan, barcha stul variantlari Chair interfeysini, barcha kofe stoli variantlari CoffeeTable interfeysini, va hokazo, ishlatishi mumkin.



*Bir xildagi obyektlarning barcha variantlari yagona klass iyerarxiyasiga almashtiriladi.*

Keyingi qadamda mahsulot oilasining bir qismi bo'lgan barcha mahsulotlar uchun yaratish metodlari(creation methods) ro'yxatiga ega bo'lgan Abstract Factory interfeysi

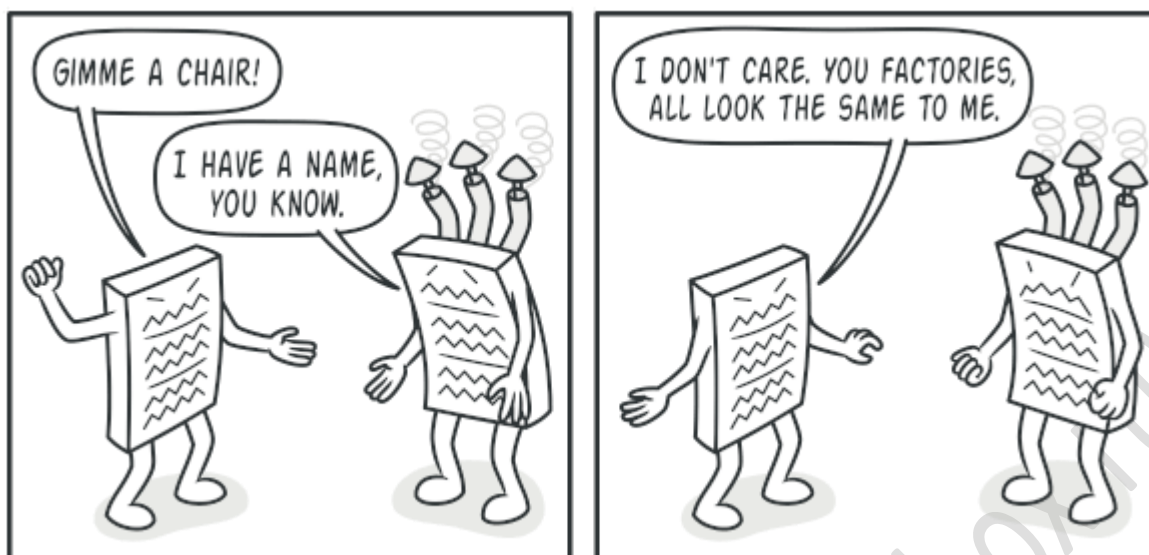
e'lon qilinadi(m: createChair, createSofa va createCoffeeTable).



*Har bir factory aniq bir mahsulot variantiga mos keladi.*

Endi, mahsulot varianlari haqida nima deyish mumkin? Har bir mahsulot oilasining varianti uchun AbstractFactory interfeysiga asosan alohida factory klassini yaratib chiqamiz. Bu yerda factory maxsus turdagi mahsulotlarni qaytaruvchi klass. Masalan, ModernFurnitureFactory faqat ModernChair, ModernSofa va ModernCoffeeTable obyektlarini yarata oladi xolos.

Mijoz kodi factorylar bilan ham, mahsulotlar bilan ham o'ziga mos bo'lgan interfays orqali ishlay olishi shart. Bu sizga mijozga uzatuvchi factory turini, shuningdek, mijoz kodi qabul qiluvchi mahsulotni, mavjud kodni buzmasdan o'zgartirish imkonini beradi.

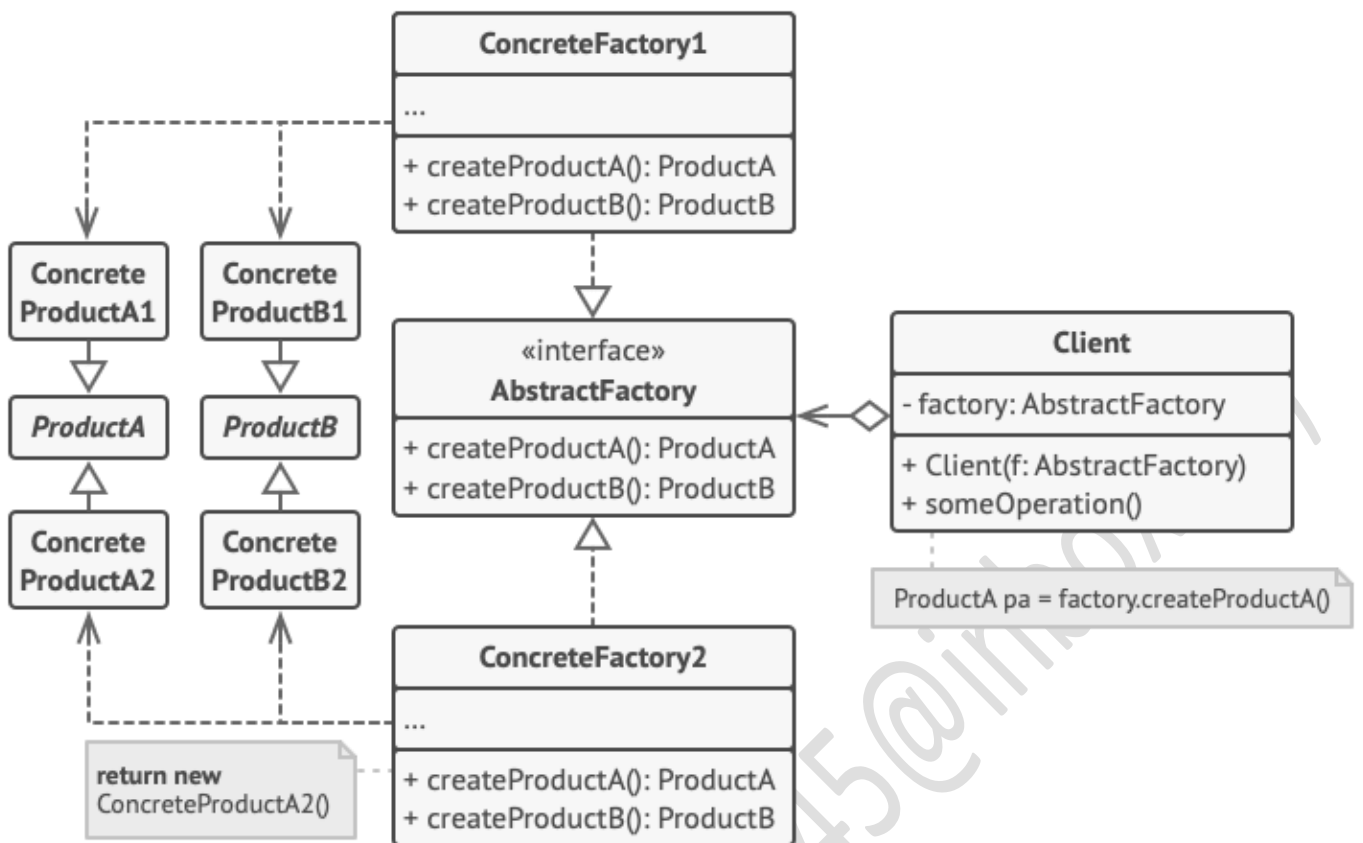


*Mijoz o'zi ishlaydigan aniq factoryning aniq klassi haqida qayg'urishi kerak emas.*

Aytaylik, mijoz stul yaratib beradigan factory xohlayapti. Mijoz factoryning klassi haqida xabari ham bo'lmaydi, qanaqa turdagi stul olishning farqi ham yo'q. Stul turi Modernmi yoki Victorian, abstract Chair interfeysidan foydalanib, mijoz ularga bir xil tarzda murojaat qilishi kerak. Bu yondashuv orqali, mijoz faqatgina o'zi olayotgan chair obyektini sitOn metodini ishlatishini biladi. Shu bilan birga, qanaqa turdagi stul qaytarilishidan qat'i nazar, u doimo o'zining factory obyektini tomonidan yaratilgan divan(sofa) va kofe stoli(coffee table) turlari bilan solishtiriladi.

Aniqlashtirilishi kerak bo'lgan yana bitta narsa qoldi: agar mijoz faqat abstract interfeyslar bilan ishlasa, unda haqiqiy factory obyektlari nima yaratadi. Odatda, dastur initsializatsiya bosqichida aniq factory obyektlarini yaratadi. Shundan oldinroq esa, dastur konfiguratsiyalarga yoki muhit sozlamalariga(environment settings) bog'liq bo'lgan factory turini tanlashi kerak.

## **Tuzilishi**



1. Abstract mahsulotlar (abstract products) alohida ammo bog'liq bo'lgan mahsulotlarning to'plami uchun interfeyslarni e'lon qiladi. Bu mahsulotlar to'plami mahsulot oilasini tashkil qiladi.
2. Aniq mahsulotlar (concrete products) – variantlar orqali guruhlanadigan abstract mahsulotlarning turli holda ishlatilishi. Har bir abstract mahsulot (chair/sofa) berilgan barcha variantlar (Victorian/Modern) da ishlatilishi kerak.
3. Abstract Factory interfeysi har bir abstract mahsulotni yaratish uchun metodlar to'plamini e'lon qiladi.
4. Aniq factorylar (concrete factory) abstract factoryning yaratish metodlarini ishlatadi. Har bir aniq factory mahsulotlarning aniq bir variantiga mos keladi va faqat shu variantdagi mahsulotni yartadi.
5. Garchi aniq factorylar aniq productlardan obyekt olsada, ularning yaratish metodlari belgilari (signatures) mos keluvchi abstract mahsulotlarni qaytarishi kerak. ...

## Qo'llanilishi

- Abstract factorydan turli xildagi o'zaro bog'liq mahsulot oilalari bilan ishlaganda foyadalaniladi. Ammo, bu mahsulotlarning aniq klasslariga bog'liq bo'lishni xohlamaysiz. Bunda siz mahsulotlar haqida oldindan hech narsa bilmaysiz yoki shunchaki kelajakdagi kengaytirishlarni hisobga olib qo'yasiz.

## Qanday ishlatiladi

1. Mahsulotlarning variantlariga qarab alohida mahsulot turlarining matritsasini chizing
2. Barcha mahsulot turlari uchun abstract mahsulot interfeyslarini e'lon qilish. Keyin barcha aniq mahsulot klasslari shu interfeyslarni ishlatishi kerak.
3. Barcha abstract mahsulotlar uchun yaratish metodlariga ega bo'lgan abstract factory interfeysini e'lon qilish kerak.
4. Aniq factory klasslar to'plamini bittalab har bir mahsulot variant uchun ishlatish kerak.
5. Dasturning kerakli joyida factoryni initsializatsiya qilish kodini yarating. U konfiguratsiya yoki muhit sozlamalariga qarab aniq factory klasslardan birini ishga tushiradi. Bu factory obyektini mahsulotlarni yartuvchi barcha klasslarga uzatish kerak.
6. Kodni ko'rib chiqib, barcha mahsulot konstruktorlarini to'g'ridan to'g'ri chaqirilgan joylarni toping. Ularni factory obyektidagi mos keluvchi yaratish metodlari chaqirishlari bilan almashtiring.

## Afzallik va kamchiliklari

### Afzalliklari

Factorydan olayotgan mahsulotlaringiz bir biri bilan mosligi haqida o'ylash shart bo'lmaydi.

### Kamchiliklari

Odatiy holatdagi koddan ko'ra murakkabroq bo'ladi. Chunki bu patternda ko'plab yangi interfeys va

klasslar qo'shiladi.

Mijoz kodi va aniq mahsulotlar orasidagi kuchli bog'lanish(tight coupling)ni chetlab o'tiladi.

Yagona Javobgarlik Tamoyili.

Mahsulotning yaratish kodini boshqa alohida joyga chiqarib olasiz.

Ochiq/Yopiqlik Tamoyili. Mavjud kodni buzmasdan turib yangi variantdagi mahsulotlarni qo'shishingiz mumkin bo'ladi.

## **Boshqa Patternlar bilan bog'liqligi**

- Ko'pgina dizaynlar Factory Methodni qo'llashdan boshlanadi (bola klasslar yordamida kamroq murakkab bo'ladi va o'zgartirish osonroq) va Abstract Factory, Prototype yoki Builder tomonga qarab rivojlanib boradi(ancha moslashuvchan, lekin biroz murakkabroq).
- Builder qadam baqadam murakkab obyektlarni qurishga e'tibor qaratadi. Abstract Factory o'zaro bog'liq bo'lgan obyektlar oilasini yaratishga mo'ljallangan. Abstract Factory mahsulotni darhol qaytaradi, Builder esa mahsulotni olishdan avval ba'zi qo'shimcha konstruksiya qadamlarini ishga tushiradi.
- Abstract Factory klasslari ko'p hollarda Factory Methodlar to'plamiga asoslanadi, ammo, siz yana bu klasslarda metodlarni yaratish uchun Prototype patterndan ham foydalanishingiz mumkin.
- Mijoz kodi tomonidan yaratilgan osttizimlarni yashirmoqchi bo'lganingizda Abstract Factory Façade ga alternative sifatida xizmat qilishi mumkin.
- Abstract Factoryni Bridge bilan birgalikda foydalanishingiz mumkin. Bunday ishlatish Bridge tomonidan e'lon qilingan ba'zi abstraksiyalar faqat maxsus



qo'llanishlar(implementation) bilan ishlaganda foydali bo'ladi. Bu holatda, Abstract Factory bu bog'lanishlarni inkapsulyatsiyalaydi va mijozdan murakkabliklarni yashiradi.

- Abstract Factory, Builders va Prototypes lar Singleton sifatida ishlatosh mumkin.

## Namuna kod

```
<?php

namespace RefactoringGuru\AbstractFactory\Conceptual;

interface AbstractFactory
{
    public function createProductA(): AbstractProductA;

    public function createProductB(): AbstractProductB;
}

class ConcreteFactory1 implements AbstractFactory
{
    public function createProductA(): AbstractProductA
    {
        return new ConcreteProductA1;
    }

    public function createProductB(): AbstractProductB
    {
        return new ConcreteProductB1;
    }
}

class ConcreteFactory2 implements AbstractFactory
{
    public function createProductA(): AbstractProductA
    {
        return new ConcreteProductA2;
    }

    public function createProductB(): AbstractProductB
```

```

    {
        return new ConcreteProductB2;
    }
}

interface AbstractProductA
{
    public function usefulFunctionA(): string;
}

class ConcreteProductA1 implements AbstractProductA
{
    public function usefulFunctionA(): string
    {
        return "The result of the product A1.";
    }
}

class ConcreteProductA2 implements AbstractProductA
{
    public function usefulFunctionA(): string
    {
        return "The result of the product A2.";
    }
}

interface AbstractProductB
{
    public function usefulFunctionB(): string;

    public function anotherUsefulFunctionB(AbstractProductA $collaborator): string;
}

class ConcreteProductB1 implements AbstractProductB
{
    public function usefulFunctionB(): string
    {
        return "The result of the product B1.";
    }

    public function anotherUsefulFunctionB(AbstractProductA $collaborator): string

```

```

    {
        $result = $collaborator->usefulFunctionA();

        return "The result of the B1 collaborating with the ({ $result })";
    }
}

class ConcreteProductB2 implements AbstractProductB
{
    public function usefulFunctionB(): string
    {
        return "The result of the product B2.";
    }

    public function anotherUsefulFunctionB(AbstractProductA $collaborator): string
    {
        $result = $collaborator->usefulFunctionA();

        return "The result of the B2 collaborating with the ({ $result })";
    }
}

function clientCode(AbstractFactory $factory)
{
    $productA = $factory->createProductA();
    $productB = $factory->createProductB();

    echo $productB->usefulFunctionB() . "\n";
    echo $productB->anotherUsefulFunctionB($productA) . "\n";
}

echo "Client: Testing client code with the first factory type:\n";
clientCode(new ConcreteFactory1);

echo "\n";

echo "Client: Testing the same client code with the second factory type:\n";
clientCode(new ConcreteFactory2);

```

Natija:

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.)

AlisherN (myemail9445@inbox.ru)