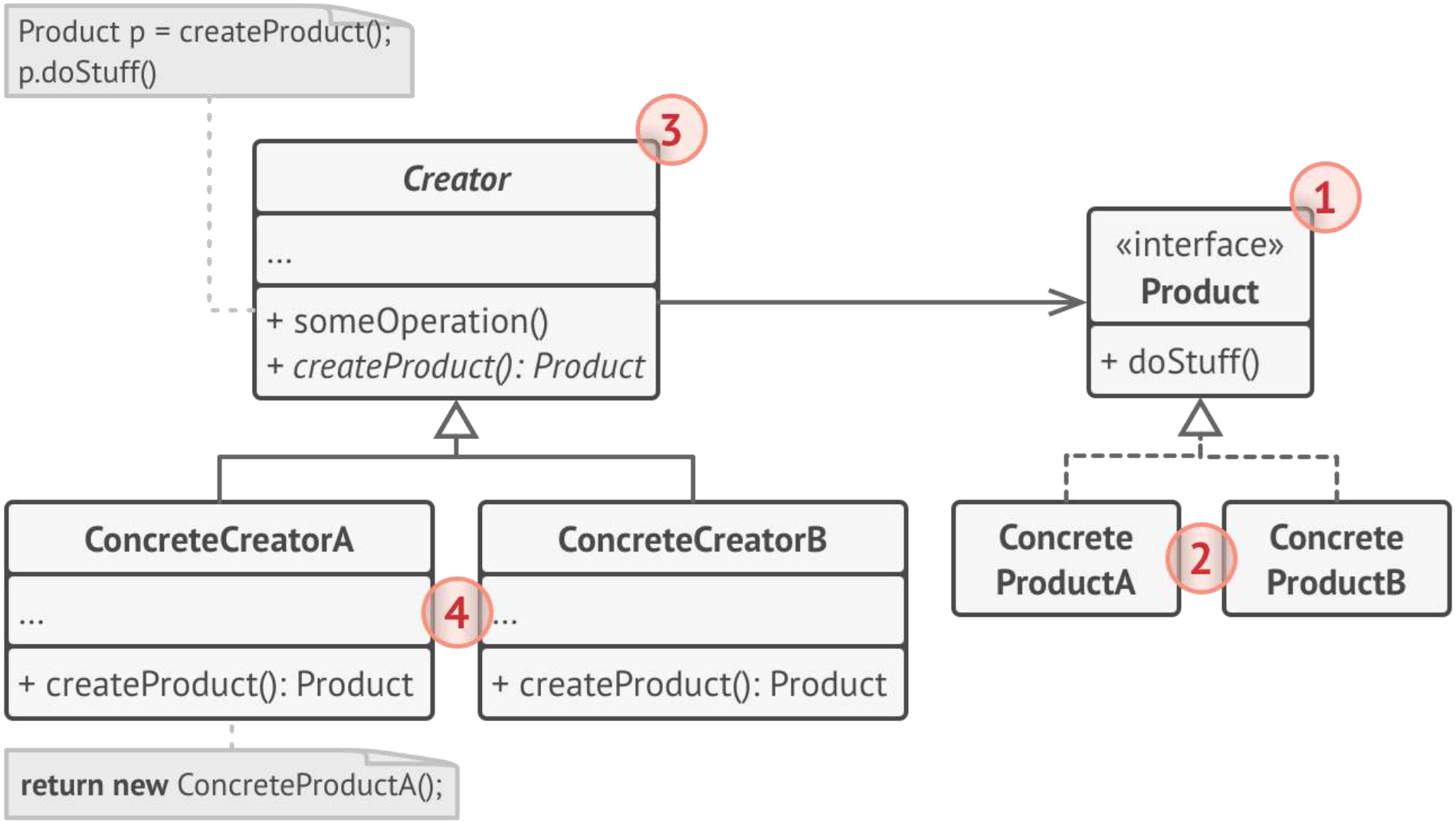
Порождающие паттерны. Часть 1

Фабричный метод

Определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.
Фабричный метод позволяет классу делегировать создание объектов подклассам

Паттерн Фабричный метод известен также под именем Virtual Constructor

Структура


Product p = createProduct();
p.doStuff()
Creator
...
+ someOperation()
+ createProduct(): Product
3
«interface»
Product
+ doStuff()
1
ConcreteCreatorA
...
+ createProduct(): Product
ConcreteCreatorB
...
+ createProduct(): Product
4
Concrete
ProductA
2
Concrete
ProductB
return new ConcreteProductA();

Когда применять?

- Когда заранее неизвестно, объекты каких типов необходимо создавать.
- Когда система должна быть независимой от процесса создания новых объектов и расширяемой: в нее можно легко вводить новые классы, объекты которых система должна создавать.
- Когда создание новых объектов необходимо делегировать из базового класса классам наследникам.

Фабричный метод. Пример

Наше приложение будет работать с документами, состоящими из разных страниц – однако в процессе разработки могут появляться новые документы, которые состоят из другого набора страниц

Фабричный метод. Пример

```
/// <summary>
/// The 'Creator' abstract class
/// </summary>
public abstract class Document
{
    List<Page> pages = new List<Page>();

    public List<Page> Pages
    {
        get { return pages; }
    }

    // Factory Method
    public abstract void CreatePages();
    // Override
    public override string ToString() => GetType().Name;
}
```

```
/// <summary>
/// The 'Product' abstract class
/// </summary>
public abstract class Page
{
    // Override. Display class name
    public override string ToString() => GetType().Name;
}
```

Фабричный метод. Пример

```
/// <summary>
/// A 'ConcreteProduct' class
/// </summary>
public class SkillsPage : Page
{
}
```

```
/// <summary>
/// A 'ConcreteProduct' class
/// </summary>
public class EducationPage : Page
{
}
```

```
/// <summary>
/// A 'ConcreteProduct' class
/// </summary>
public class ExperiencePage : Page
{
}
```

```
/// <summary>
/// A 'ConcreteProduct' class
/// </summary>
public class IntroductionPage : Page
{
}
```

```
/// <summary>
/// A 'ConcreteProduct' class
/// </summary>
class ResultsPage : Page
{
}
```

```
/// <summary>
/// A 'ConcreteProduct' class
/// </summary>
public class ConclusionPage : Page
{
}
```

```
/// <summary>
/// A 'ConcreteProduct' class
/// </summary>
public class SummaryPage : Page
{
}
```

```
/// <summary>
/// A 'ConcreteProduct' class
/// </summary>
public class BibliographyPage : Page
{
}
```

Фабричный метод. Пример

```
/// <summary>
/// A 'ConcreteCreator' class
/// </summary>
public class Resume : Document
{
    // Factory Method implementation
    public override void CreatePages()
    {
        Pages.Add(new SkillsPage());
        Pages.Add(new EducationPage());
        Pages.Add(new ExperiencePage());
    }
}
```

```
/// <summary>
/// A 'ConcreteCreator' class
/// </summary>
public class Report : Document
{
    // Factory Method implementation
    public override void CreatePages()
    {
        Pages.Add(new IntroductionPage());
        Pages.Add(new ResultsPage());
        Pages.Add(new ConclusionPage());
        Pages.Add(new SummaryPage());
        Pages.Add(new BibliographyPage());
    }
}
```

Фабричный метод. Пример

```
public class Program
{
    public static void Main()
    {
        // Document constructors call Factory Method
        var documents = new List<Document> { new Resume(), new Report() };

        // Display document pages
        foreach (var document in documents)
        {
            document.CreatePages();  // Factory method

            WriteLine($"{document} --");
            foreach (var page in document.Pages) WriteLine($" {page}");
            WriteLine();
        }

        // Wait for user
        ReadKey();
    }
}
```

Фабричный метод. Пример

```
interface ISnackFactory
{
    ISnack CreateSnack();
}
```

```
interface ISnack
{
    bool IsRefrigirationRequired { get; }
    void Eat();
}
```

Фабричный метод. Пример

```
class IcecreamFactory : ISnackFactory
{
    public ISnack CreateSnack()
    {
        return new Icecream();
    }
}
```

```
class ChocolateFactory : ISnackFactory
{
    public ISnack CreateSnack()
    {
        return new Chocolate();
    }
}
```

Фабричный метод. Пример

```
class Chocolate : ISnack
{
    public bool IsRefrigirationRequired
    {
        get { return false; }
    }

    public void Eat()
    {
        Console.WriteLine(string.Format("Refrigiration Required? {0}", IsRefrigirationRequ
        Console.WriteLine("Chocolate is sweet and yummy");
    }
}
```

Фабричный метод. Пример

```
class Icecream : ISnack
{
    public bool IsRefrigirationRequired
    {
        get { return true; }
    }

    public void Eat()
    {
        Console.WriteLine(string.Format("Refrigiration Required? {0}", IsRefrigirationRequ
        Console.WriteLine("Icecream is cool and soft");
    }
}
```

Фабричный метод. Пример

```
class Program
{
    static void Main(string[] args)
    {
        ISnackFactory snackFactory = LoadFactory("icecream");

        ISnack snack = snackFactory.CreateSnack();
        snack.Eat();
    }

    private static ISnackFactory LoadFactory(string snack)
    {
        switch (snack)
        {
            case "icecream":
                return new IcecreamFactory();

            default:
                return new ChocolateFactory();

                break;
        }
    }
}
```

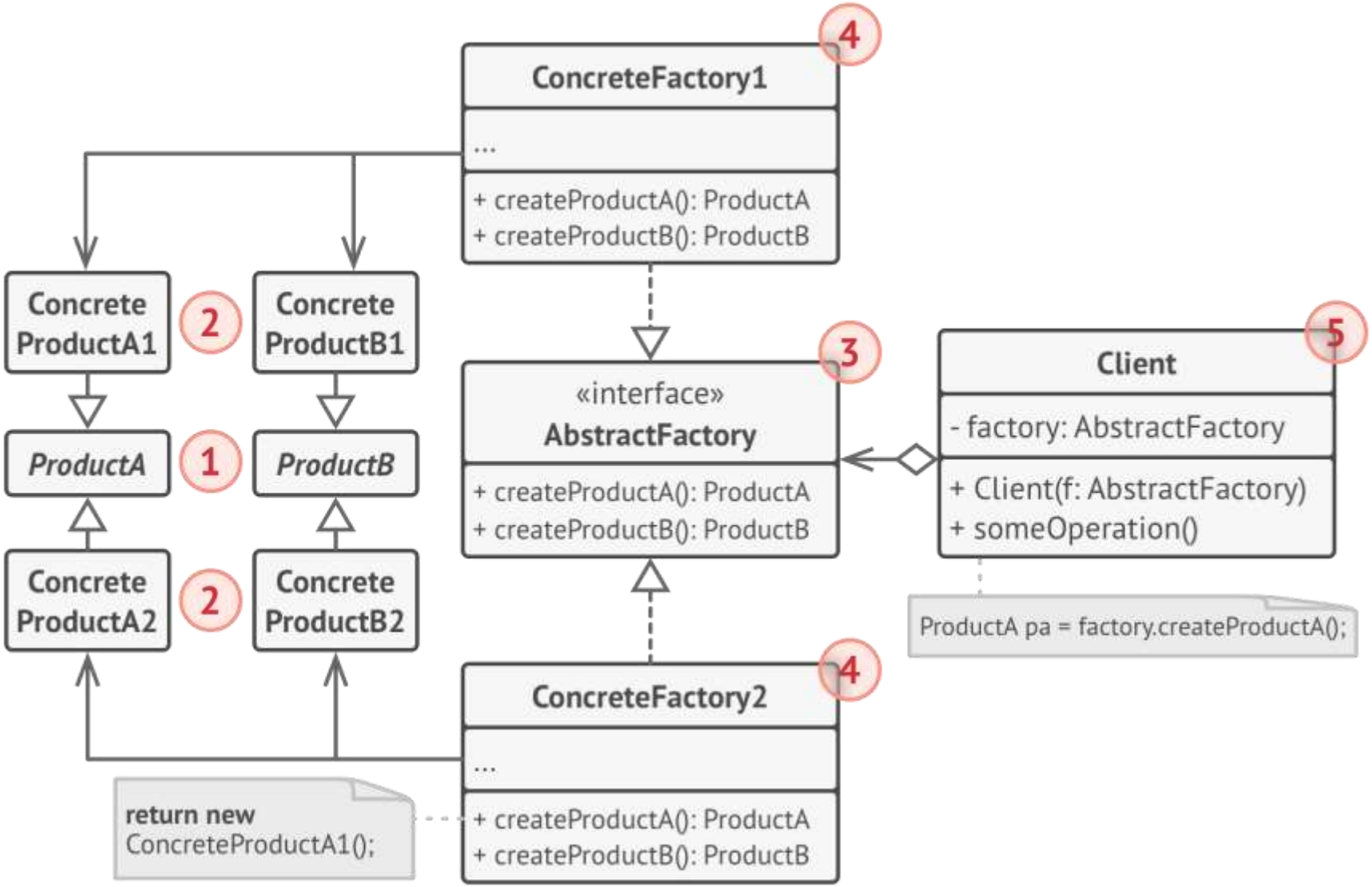
Фабричный метод

Потенциальный недостаток фабричного метода состоит в том, что клиентам, возможно, придется создавать подкласс класса Creator для создания лишь одного объекта ConcreteProduct. **Порождение подклассов оправдано, если клиенту так или иначе приходится создавать подклассы Creator,** в противном случае клиенту *придется иметь дело с дополнительным уровнем подклассов*

Абстрактная фабрика

Позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов.

Структура


4
ConcreteFactory1
...
+ createProductA(): ProductA
+ createProductB(): ProductB
Concrete ProductA1
2
Concrete ProductB1
ProductA
1
ProductB
Concrete ProductA2
2
Concrete ProductB2
3
«interface»
AbstractFactory
+ createProductA(): ProductA
+ createProductB(): ProductB
5
Client
- factory: AbstractFactory
+ Client(f: AbstractFactory)
+ someOperation()
ProductA pa = factory.createProductA();
4
ConcreteFactory2
...
+ createProductA(): ProductA
+ createProductB(): ProductB
return new ConcreteProductA1();

Когда применять?

- Когда система не должна зависеть от способа создания и компоновки новых объектов.
- Когда создаваемые объекты должны использоваться вместе и являются взаимосвязанными.

Абстрактная фабрика. Пример

Мы хотим написать программу, которая способна создавать обычный мобильный телефон и смартфон. Разные производители производят разные модели

Абстрактная фабрика. Пример

```
/// <summary>
/// The 'AbstractFactory' interface.
/// </summary>
interface IMobilePhone
{
    ISmartPhone GetSmartPhone();
    INormalPhone GetNormalPhone();
}
```

```
/// <summary>
/// The 'AbstractProductA' interface
/// </summary>
interface ISmartPhone
{
    string GetModelDetails();
}
```

```
/// <summary>
/// The 'AbstractProductB' interface
/// </summary>
interface INormalPhone
{
    string GetModelDetails();
}
```

Абстрактная фабрика. Пример

```
/// <summary>
/// The 'ProductA1' class
/// </summary>
class NokiaPixel : ISmartPhone
{
    public string GetModelDetails()
    {
        return "Model: Nokia Pixel\nRAM: 3GB\nCamera: 8MP\n";
    }
}
```

```
/// <summary>
/// The 'ProductA2' class
/// </summary>
class SamsungGalaxy : ISmartPhone
{
    public string GetModelDetails()
    {
        return "Model: Samsung Galaxy\nRAM: 2GB\nCamera: 13MP\n";
    }
}
```

Абстрактная фабрика. Пример

```
/// <summary>
/// The 'ProductB1' class
/// </summary>
class Nokia1600 : INormalPhone
{
    public string GetModelDetails()
    {
        return "Model: Nokia 1600\nRAM: NA\nCamera: NA\n";
    }
}
```

```
/// <summary>
/// The 'ProductB2' class
/// </summary>
class SamsungGuru : INormalPhone
{
    public string GetModelDetails()
    {
        return "Model: Samsung Guru\nRAM: NA\nCamera: NA\n";
    }
}
```

Абстрактная фабрика. Пример

```
/// <summary>
/// The 'ConcreteFactory1' class.
/// </summary>
class Nokia : IMobilePhone
{
    public ISmartPhone GetSmartPhone()
    {
        return new NokiaPixel();
    }

    public INormalPhone GetNormalPhone()
    {
        return new Nokia1600();
    }
}
```

```
/// <summary>
/// The 'ConcreteFactory2' class.
/// </summary>
class Samsung : IMobilePhone
{
    public ISmartPhone GetSmartPhone()
    {
        return new SamsungGalaxy();
    }

    public INormalPhone GetNormalPhone()
    {
        return new SamsungGuru();
    }
}
```

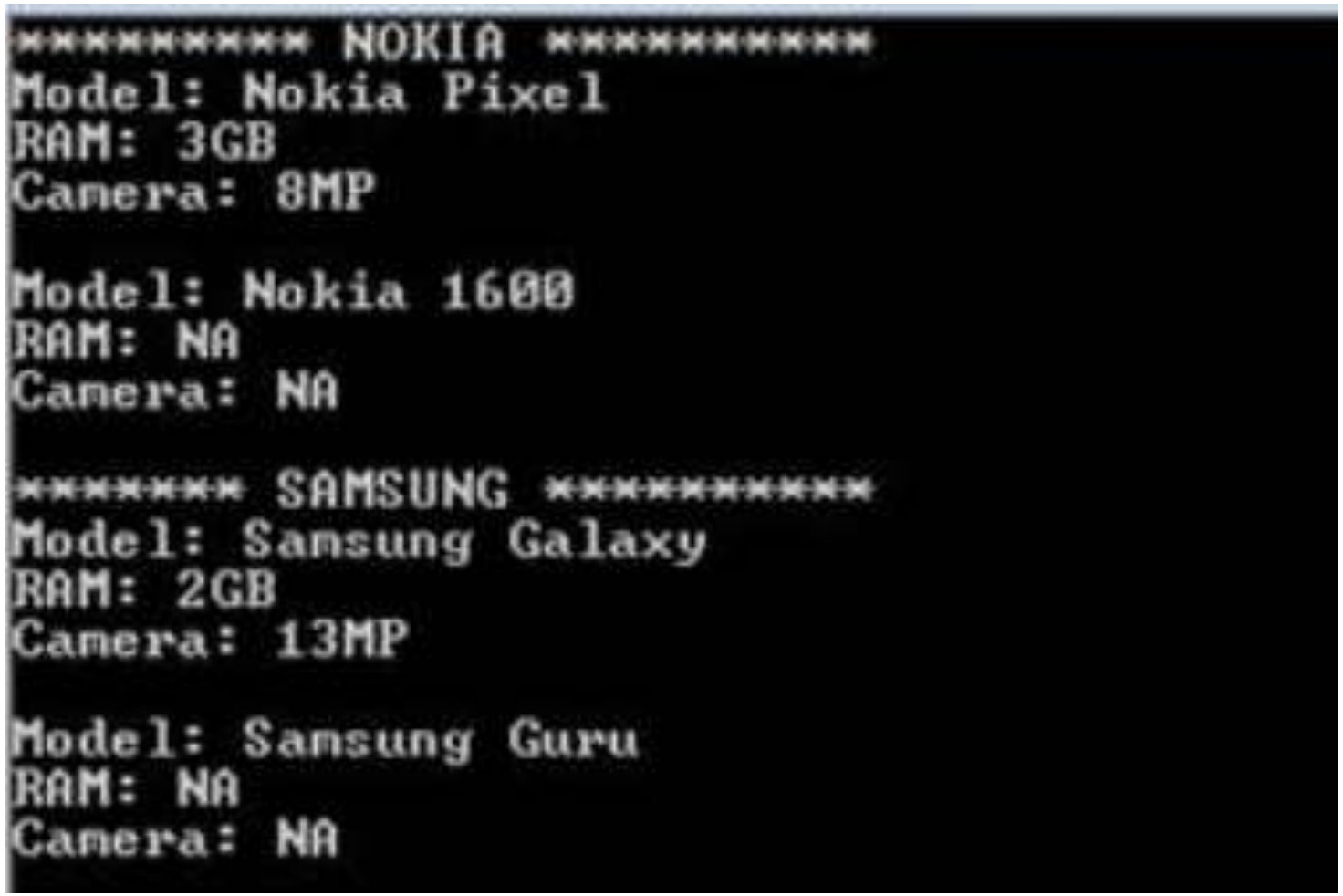
Абстрактная фабрика. Пример

```
/// <summary>
/// The 'Client' class
/// </summary>
class MobileClient
{
    ISmartPhone smartPhone;
    INormalPhone normalPhone;

    public Client(IMobilePhone factory)
    {
        smartPhone = factory.GetSmartPhone();
        normalPhone = factory.GetNormalPhone();
    }

    public string GetSmartPhoneModelDetails()
    {
        return smartPhone.GetModelDetails();
    }

    public string GetNormalPhoneModelDetails()
    {
        return normalPhone.GetModelDetails();
    }
}
```

Абстрактная фабрика. Пример

```
/// <summary>
/// Abstract Factory Pattern Demo
/// </summary>
class Program
{
    static void Main()
    {
        IMobilePhone nokiaMobilePhone = new Nokia();
        MobileClient nokiaClient = new MobileClient(nokiaMobilePhone);

        Console.WriteLine("********** NOKIA **********");
        Console.WriteLine(nokiaClient.GetSmartPhoneModelDetails());
        Console.WriteLine(nokiaClient.GetNormalPhoneModelDetails());

        IMobilePhone samsungMobilePhone = new Samsung();
        MobileClient samsungClient = new MobileClient(samsungMobilePhone);

        Console.WriteLine("******* SAMSUNG **********");
        Console.WriteLine(samsungClient.GetSmartPhoneModelDetails());
        Console.WriteLine(samsungClient.GetNormalPhoneModelDetails());

        Console.ReadKey();
    }
}
```

Абстрактная фабрика. Пример



Спасибо за внимание!