1. Абстрактна Фабрика (Abstract Factory)

Уявімо, що ви прийшли в іграшковий магазин (відіграючи роль Діда Мороза[11]) і хочете накупити іграшок дітям (і не обов'язково своїм). Мартуся любить плюшеві іграшки — вона часто із ними лягає у ліжко спати. А Дмитрик страшний розбишака, ламає все на світі, рве м'які іграшки і, зазвичай, віддає перевагу гратися із твердими, дерев'яними іграшками. Двоє дітей хочуть ведмедика і котика і ще купу інших тваринок. На щастя, магазин має широкий асортимент забавок і вам вдалося вдосталь закупитися. В один мішок ви накидали дерев'яних іграшок, а в інший плюшевих.

Таким чином, коли ви підійшли до Мартусі, яка любить м'які іграшки, ви витягали із свого мішка спочатку плюшевого ведмедя, а далі плюшевого котика і так далі. Аналогічно ви підійшли до Дмитрика і подарували йому дерев'яного ведмедика і котика, і собаку, і слона, і бегемота... і крокодила...

Абстрактна фабрика надає простий інтерфейс для створення об'єктів, які належать до того чи іншого сімейства.[12]

В нашому прикладі сімейством є набір іграшок-тварин, які по-сімейному реалізують базові класи ведмедика (*Bear*), кота (*Cat*), і інші. Тобто повний звіринець певної реалізації, дерев'яної або плюшевої, і буде сімейством. Конкретною фабрикою є мішок. Одна із конкретних фабрик повертає дерев'яні іграшки, а інша повертає плюшеві. Тому якщо одна дитина просить котика, то їй вернуть реалізацію котика у відповідності до інстанційованого мішка із іграшками.

Я сподіваюся, що приклад із аналогіями не видався заплутаним. А якщо все ж таки видався, пропоную подивитися трохи коду. *Абстрактна фабрика* визначає інтерфейс, що повертає об'єкти кота або ведмедя (базові класи). Конкретні реалізації фабрики повертають конкретні реалізації іграшок потрібного сімейства.

Уривок коду 1.1. Інтерфейс абстрактної фабрики та дві конкретні реалізації

```
// абстрактна фабрика (abstract factory)
public interface IToyFactory
{
    Bear GetBear();
    Cat GetCat();
}
```

[11] Життя складається із 4 фаз: а) Ви вірите в Діда Мороза (Миколая/Санту); б) Ви більше не вірите; в) Ви самі Дід Мороз; г) Ви схожі на Діда Мороза.

[12] **Abstract Factory**. Intent: Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
Абстрактна фабрика. Призначення: Надати інтерфейс для створення сімейств споріднених або залежних об'єктів без зазначення їхніх конкретних класів.

```
// конкретна фабрика (concrete factory)
public class TeddyToysFactory : IToyFactory
{
    public Bear GetBear()
    {
        return new TeddyBear();
    }
    public Cat GetCat()
    {
        return new TeddyCat();
    }
}
// і ще одна конкретна фабрика
public class WoodenToysFactory : IToyFactory
{
    public Bear GetBear()
    {
        return new WoodenBear();
    }
    public Cat GetCat()
    {
        return new WoodenCat();
    }
}
```

Уже зрозуміло, що, як тільки ми маємо якийсь екземпляр фабрики, ми можемо плодити сімейство потрібних іграшок. Тому глянемо на використання:

Уривок коду 2.2. Використання конкретної фабрики для дерев'яних іграшок

```
// Спочатку створимо «дерев'яну» фабрику
IToyFactory toyFactory = new WoodenToysFactory();
Bear bear = toyFactory.GetBear();
Cat cat = toyFactory.GetCat();
Console.WriteLine("I've got {0} and {1}", bear.Name, cat.Name);
// Вивід на консоль буде: [I've got Wooden Bear and Wooden Cat]
```

Уривок коду 3.3. Використання конкретної фабрики для плюшевих іграшок

```
// А тепер створимо «плюшеву» фабрику, наступна лінійка є єдиною різницею в коді
IToyFactory toyFactory = new TeddyToysFactory();
// Як бачимо код нижче не відрізняється від наведеного вище
Bear bear = toyFactory.GetBear();
Cat cat = toyFactory.GetCat();
Console.WriteLine("I've got {0} and {1}", bear.Name, cat.Name);
// А вивід на консоль буде інший: [I've got Teddy Bear and Teddy Cat]
```

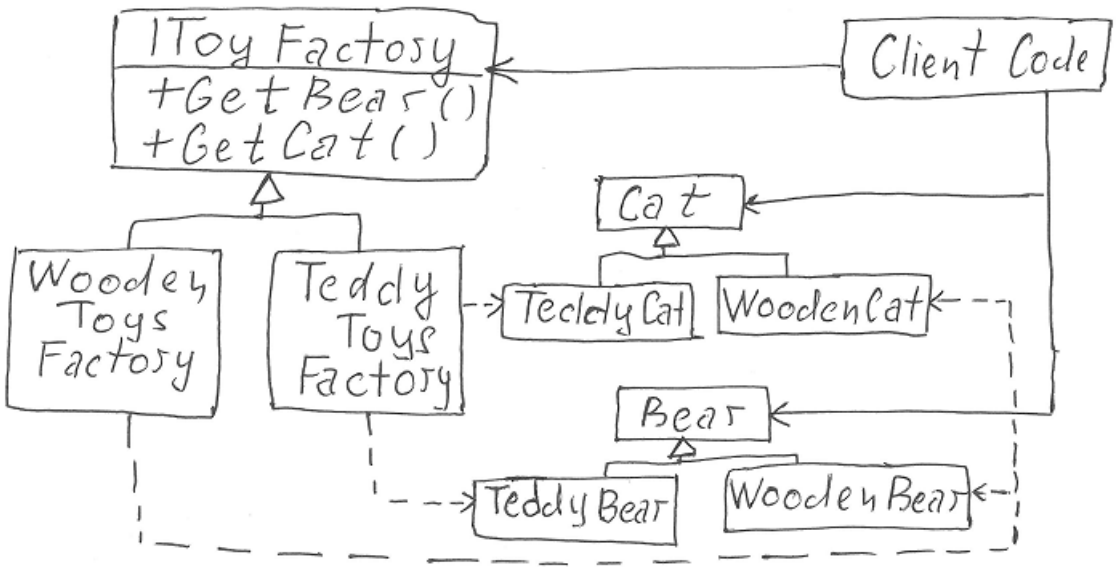
Два шматки коду майже абсолютно ідентичні – різниця тільки в конкретному «мішку». Якщо вас ще цікавлять *реалізації іграшок-тваринок*, то вони доволі тривіальні.

Уривок коду 1.3. Реалізації іграшок-тваринок

```
// Базовий клас для усіх котиків, базовий клас AnimalToy містить Name
public abstract class Cat : AnimalToy
{
    protected Cat(string name) : base(name) { }
}
```

```
// Базовий клас для усіх ведмедиків
public abstract class Bear : AnimalToy
{
    protected Bear(string name) : base(name) { }
}
// Конкретні реалізації
class WoodenCat : Cat
{
    public WoodenCat() : base("Wooden Cat") { }
}
class TeddyCat : Cat
{
    public TeddyCat() : base("Teddy Cat") { }
}
class WoodenBear : Bear
{
    public WoodenBear() : base("Wooden Bear") { }
}
class TeddyBear : Bear
{
    public TeddyBear() : base("Teddy Bear") { }
}
```

Абстрактна Фабрика є дуже широковикористовуваним дизайн-патерном. Дуже яскравим прикладом буде ADO.NET *DbProviderFactory* [13], яка є абстракною фабрикою, що визначає інтерфейси для отримання *DbCommand*, *DbConnection*, *DbParameter* і так далі. Конкретна фабрика *SqlClientFactory* поверне відповідно *SqlCommand*, *SqlConnection* і так далі. Це дозволяє працювати із різними джерелами даних. Дякую за те, що дізналися про цей дизайн-патерн саме із моєї книги та із моїм прикладом.



UML-діаграма 1. Абстрактна фабрика

[13] DbProviderFactory Class: http://msdn.microsoft.com/en-us/library/system.data.common.dbproviderfactory(v=vs.80).aspx