Лабораторна робота №2

Тема: Породчуючі шаблони проектування.
Абстрактна фабрика та Будівельник

Теоретичні відомості

Усі шаблони проектування поділені на три великі групи, а саме: породжуючі, структурні та поведінкові. Основним завданням породжуючи шаблонів є спростити створення об’єктів.

Інколи ви працюєте із певним набором об’єктів через групу інтерфейсів. А тоді хочете створювати об’єкти тільки із іншого набору, щоб пристосувати ваш код до інших умов. Звичайно, група інтерфейсів, через які ви оперуєте, залишається та ж сама. Спростити створення відповідного набору допоможе Абстрактна Фабрика.

А інколи структура деякого об’єкта дуже складна і залежить від багатьох чинників. Щоб спростити створення такого об’єкту зазвичай використовують Будівельника.

А щоб зручно вибрати одну реалізацію та інстанціювати її, відштовхуючись від простої умови, можна використати Фабричний Метод.

Визначення

Породжучі шаблони (Creational patterns) - шаблони проектування, які абстрагують процес інстанціювання. Інстанціювання (англ. Instantiation) - створення екземпляра класу. На відміну від слова «створення», застосовується не до об'єкта, а до класу. Тобто, кажуть: створити екземпляр класу або інстанціювати клас.

Породжучі шаблони дозволяють зробити систему незалежною від способу створення, композиції і представлення об'єктів. Шаблон, який породжує класи, використовує успадкування, щоб змінювати інстанційований клас, а шаблон, який породжує об'єкти, делегує інстанціювання іншому об'єкту.

Використання

Ці шаблони важливі тоді, коли система більше залежить від композиції об'єктів, ніж від успадкування класів. Виходить так, що основний упор робиться не на жорсткому кодуванні фіксованого набору поводжень, а на визначенні невеликого набору фундаментальних поводжень, з допомогою композиції яких можна отримувати будь-яке число складніших. Таким чином, для створення об'єктів з конкретною поведінкою потрібно щось більше, ніж просте інстанціювання класу.

Породжують шаблони інкапсулюють(узагальнюють) знання про конкретні класах, які застосовуються в системі. Вони приховують деталі того, як ці класи створюються і стикуються. Єдина інформація про об'єкти, відома системі, - це їх інтерфейси, визначені за допомогою абстрактних класів. Отже, породжуючі шаблони забезпечують більшу гнучкість при вирішенні питання про те, що створюється, хто це створює, як і коли. Можна зібрати систему з «готових» об'єктів з різноманітною структурою та функціональністю статично (на етапі компіляції) або динамічно (під час виконання).

Іноді допустимо вибирати між тим чи іншим породжуючим шаблоном. Наприклад, є випадки, коли з однаковою користю можна використовувати як прототип, так і абстрактну фабрику. В інших ситуаціях породжуючі шаблони доповнюють один одного. Так, застосовуючи будівельник, можна використовувати інші шаблони для вирішення питання про те, які компоненти потрібно будувати, а прототип часто реалізується разом з одинаком. Породжують шаблони тісно пов'язані один з одним, їх розгляд краще проводити спільно, щоб краще було видно їхні подібності та відмінності.

Перелік породжуючих шаблонів:

- Абстрактна фабрика (abstract factory);
- Будівельник (builder);
- Фабричний метод (factory method);
- Лінива ініціалізація (lazy initialization);
- Об'єктний пул (object pool);
- Прототип (prototype);
- Одинак (singleton);
- Блокування з подвійною перевіркою (double checked locking).

Abstract Factory

Частота використання: *висока* (5 з 5)

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

Приклад використання:

Уявімо, що ви прийшли в іграшковий магазин і хочете накупити іграшок дітям. Мартуся любить плюшеві, а Дмитрик 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 т.д.

В даному 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.

Абстрактна фабрика визначає інтерфейс, що повертає об'єкти кота або ведмедя (базові класи). Конкретні реалізації фабрики повертають конкретні реалізації іграшок потрібного сімейства.

Фрагменти програмного коду:

Інтерфейс абстрактної фабрики та дві конкретні реалізації:

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

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]
```

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]
```

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") { }
    }
}
```

Нижче наведена UML-діаграма класів, на які продемонстровано взаємодію класів у описаному шаблоні проектування.

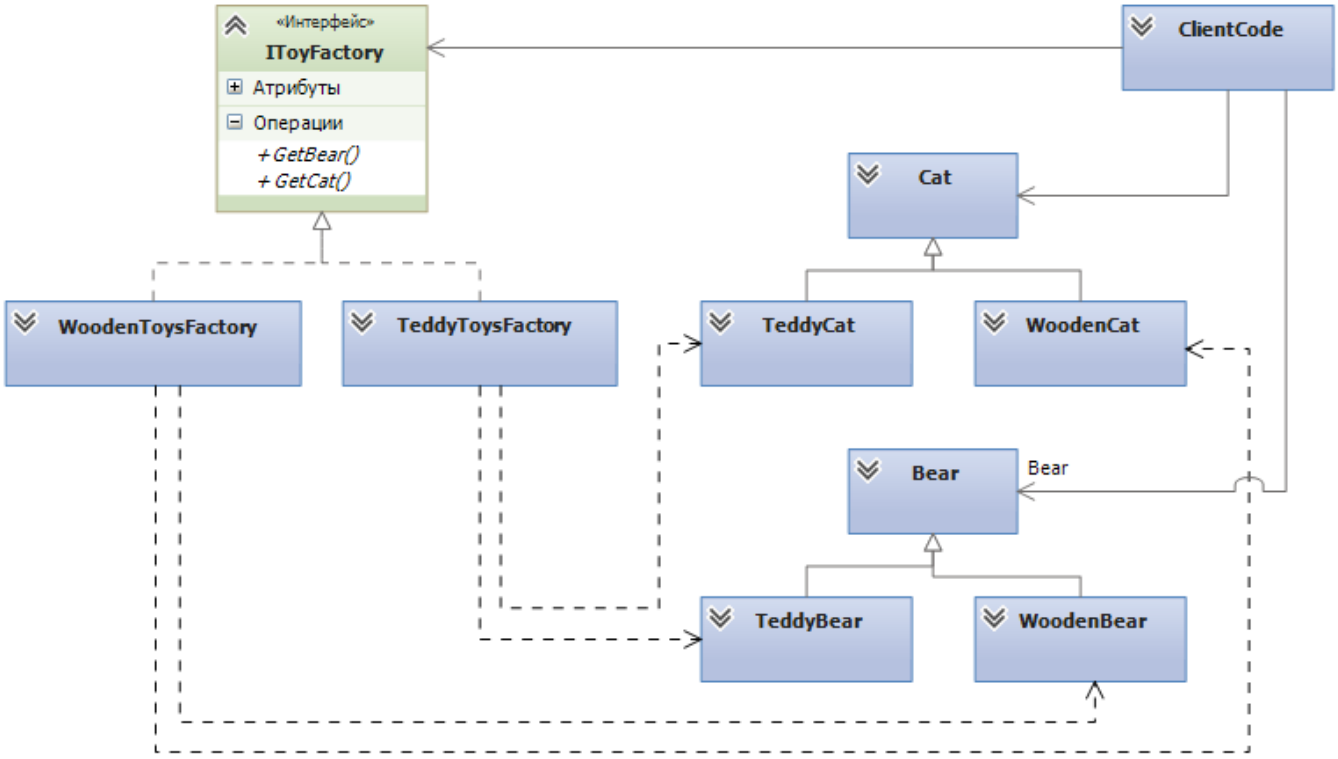

Рис.1. Діаграма класів патерну Abstract Factory

Будівельник (Builder)

Частота використання: *середня* (3 з 5)

Будівельник вимальовує стандартний процес створення складного об'єкта, розділяючи логіку будування об'єкта від його представлення

Приклад використання:

Уявіть, що ви володієте магазином з продажу персональних комп'ютерів, в якому можна вибирати конфігурацію прямо біля каси (як піцу в піцерії). Вам слід створити систему, що дозволить легко будувати будь-яку конфігурацію ноутбука для будь-якого покупця. Причому, стандартні конфігурації мають складатися «по накату». От як піца може бути «фірмова», «грибна», «справжня італійська», «пепероні», або ваша специфічна, так і ноутбук: для геймера або для подорожей.

Шляхом додавання певних частин, таких як процесор, пам'ять, жорсткий диск, батарея і т.д, можна скласти повноцінний комп'ютер. Часто конфігурації можуть набути певних стандартів. В кінці кінців, ваш продавець балакає із клієнтом і запитує, яку пам'ять він хоче, і так далі. Або ж, якщо клієнт не дуже розуміється в компютерах, продавець може запитати: «Ну вам той комп треба для ігор? Яких?». Звісно, що ви наперед знаєте певні кроки, щоб створити

ноутбук. Ці кроки визначаються в абстрактному будівельнику (abstract builder). Єдине що вам потрібно від клієнта, це дізнатися, які компоненти використовувати на кожному кроці.

Фрагменти програмного коду:

Абстракний будівельник

```
abstract class LaptopBuilder
    {
        protected Laptop Laptop { get; private set; }
        public void CreateNewLaptop()
        {
            Laptop = new Laptop();
        }
        // Метод, який повертає готовий ноутбук назовні
        public Laptop GetMyLaptop()
        {
            return Laptop;
        }
        // Кроки, необхідні щоб створити ноутбук
        public abstract void SetMonitorResolution();
        public abstract void SetProcessor();
        public abstract void SetMemory();
        public abstract void SetHDD();
        public abstract void SetBattery();
    }
```

Якщо покупець відповідає «Ух, так! Я хочу грати ігри... ууеех!» і у нього загорілися очі, то ви вже напоготові і маєте конкретну реалізацію для ігрового ноутбука:

Конкретна реалізація будівельника

```
    // Таким будівельником може бути працівник, що
    // спеціалізується у складанні «геймерських» ноутів
    class GamingLaptopBuilder : LaptopBuilder
    {
        public override void SetMonitorResolution()
        {
            Laptop.MonitorResolution = "1900X1200";
        }
        public override void SetProcessor()
        {
            Laptop.Processor = "Core 2 Duo, 3.2 GHz";
        }
        public override void SetMemory()
        {
            Laptop.Memory = "6144 Mb";
        }
        public override void SetHDD()
        {
            Laptop.HDD = "500 Gb";
        }
        public override void SetBattery()
        {
            Laptop.Battery = "6 lbs";
        }
    }
```

Але, якщо ваш покупець бізнесмен, то він обере комп'ютер для подорожей

Конкретний будівельник для ноутбуків

```
// А ось інший «збирач» ноутів
class TripLaptopBuilder : LaptopBuilder
{
    public override void SetMonitorResolution()
    {
        Laptop.MonitorResolution = "1200X800";
    }
    public override void SetProcessor()
    {
        //.. і так далі...
    }
}
```

Для того, щоб справитися із побудовою ноутбука, базуючись на відповіді покупця, знадобиться директор (director):

Директором може бути код приймаючий замовлення

```
class BuyLaptop
{
    private LaptopBuilder _laptopBuilder;
    public void SetLaptopBuilder(LaptopBuilder lBuilder)
    {
        _laptopBuilder = lBuilder;
    }
         // Змушує будівельника повернути цілий ноутбук
    public Laptop GetLaptop()
    {
        return _laptopBuilder.GetMyLaptop();
    }
    // Змушує будівельника додавати деталі
    public void ConstructLaptop()
    {
        _laptopBuilder.CreateNewLaptop();
        _laptopBuilder.SetMonitorResolution();
        _laptopBuilder.SetProcessor();
        _laptopBuilder.SetMemory();
        _laptopBuilder.SetHDD();
        _laptopBuilder.SetBattery();
    }
}
```

Використання патерну

```
    // Ваша система може мати багато конкретних будівельників
    var tripBuilder = new TripLaptopBuilder();
    var gamingBuilder = new GamingLaptopBuilder();
    var shopForYou = new BuyLaptop();//Директор
    // Покупець каже, що хоче грати ігри
    shopForYou.SetLaptopBuilder(gamingBuilder);
    shopForYou.ConstructLaptop();
    // Ну то нехай бере що хоче!
    Laptop laptop = shopForYou.GetLaptop();
    Console.WriteLine(laptop.ToString());
    // Вивід: [Laptop: 1900X1200, Core 2 Duo, 3.2 GHz, 6144 Mb, 500 Gb, 6 lbs]
```

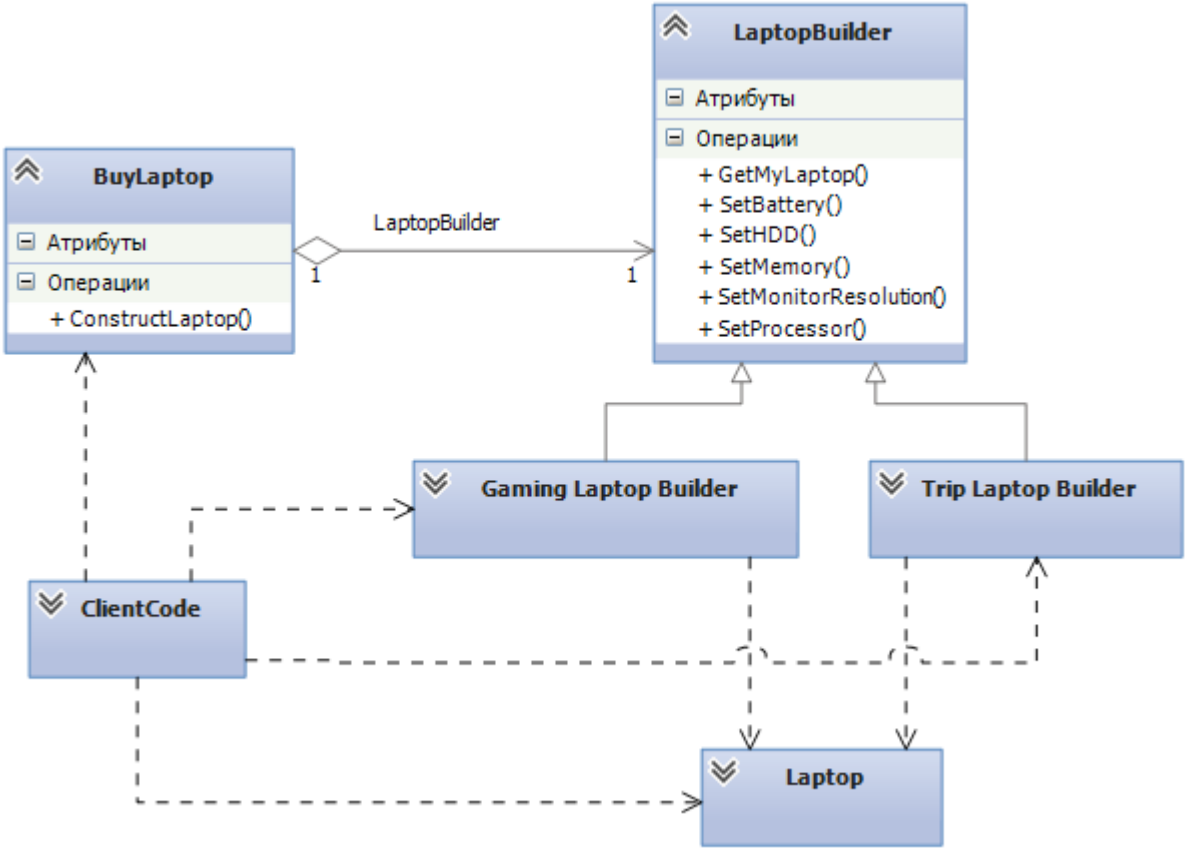

Рис.2. Діаграма класів шаблону проектування Builder

Індивідуальні завдання

Розробити обєктно-орієнтовану програму згідно варіанту з використанням запропонованого шаблону проектування. Побудувати діаграму класів та зробити висновки про доцільність використання відповідного шаблону.

Варіант №1

Ювелірний завод виготовляє золоті та срібні прикраси у вигляді сережок, каблучок, ланцюжків, підвісок та браслетів. Модель кожного виробу може бути виконана як у золоті та і у сріблі. Вартість моделі конкретного виробу залежить від виду дорогоцінного металу, ваги та складності роботи (золоті сережки однакової ваги можуть мати різну вартість). Вартість роботи конкретної моделі для золота і срібла однакова. Розробити програму для перегляду каталогу ювелірний виробів за категоріями «Золото» та «Срібло». Для створення об’єктів скористатись шаблоном Абстрактна фабрика(Abstract Factory).

Варіант №2

У фасувальному цеху кондитерської фабрики проводять пакування подарункових наборів цукерок трьох типів (економічний «Ласунка», стандартний «Наминайко» та екстра «Пан Коцький»). У кожен набір входять льодяники, шоколадні цукерки, вафлі та драже. Вартість за кілограм солодощів вказується при запуску системи пакування (програми). Кожен набір складається з різної кількість солодощів кожного типу, але однаковий за вагою (допускається невелике відхилення). Оператор сам визначає, який з подарункових наборів він запускає на пакування, а на дисплей виводиться назва подарункового набору, ціна, та вага кожного виду солодощів, які входять до набору. Поставлене завдання реалізувати за допомогою шаблону Будівельник (Builder).

Варіант №3

Фабрика меблевих фасадів виготовляє плівочні, фарбовані та пластикові фасади двох видів: суцільні та вітрини (з отвором для скла). Вартість фасаду залежить від типу матеріалу, з якого він виконаний та від розміру (у кв.метрах). Замовлення може містити лише фасади одного матеріалу. Якщо покупець хоче фасади з різних матеріалів, то йому необхідно оформити по одному замовленню на кожен тип матеріалів. Розробити програму, яка обраховує вартість замовлення та виводить інформацію про усі фасади, що входять до замовлення(суцільний фасад чи вітрина та його розміри). Для створення об’єктів скористатись шаблоном Абстрактна фабрика(Abstract Factory).

Варіант №4

Автовиробник пропонує нову модель легкового автомобіля. Дана модель доступна у 4 стандартних комплектаціях різної вартості. Кожна комплектація характеризується типом двигуна (бензиновий чи дизельний), об’ємом двигуна, антиблокувальною гальмівною системою (ABS), електронною системою стабілізації (ESP), кількістю подушок безпеки, наявністю бортового комп’ютера, кондиціонером чи клімат-контролем, обшивкою салону. За назвою комплектації покупець може переглянути її характеристики. Розробити програму для отримання інформації про конкретну комплектацію автомобіля та його вартість. Для створення об’єктів скористатись шаблоном Будівельник (Builder).

Варіант №5

Американська фірма-виробник мобільних телефонів відкрила філіал у одній з країн сходу. Моделі телефонів, що виготовляються у філіалі ті ж самі,

що і у США, але вартість суттєво відрізняється за рахунок економії на оплаті праці. Інтернет-магазин, який займається продажем телефонів даної марки, дає покупцю можливість обрати країну виробника. Після оформлення замовлення покупець отримує інформацію про термін доставки та вартість телефону. Дані характеристики залежать від країни виробника. Розробити програму для покупки телефону через інтернет-магазин. Для створення об’єктів скористатись шаблоном Абстрактна фабрика(Abstract Factory).

Контрольні запитання

1. Яке призначення породжуючи шаблонів проектування?
2. Поясніть поняття інстанціювання.
3. Назвіть п’ять породжуючих шаблонів.
4. Поясніть суть шаблону Abstract Factory.
5. В яких випадках доцільно використовувати шаблон Builder?