Design Patterns GoF

Creational Patterns

- Prototype
- Abstract Factory
- Factory Method
- Builder
- Singleton

Означення

Патерн проектування:

1) опис *взаємодії об'єктів і класів*,
2) розроблених для *вирішення загальної задачі* проектування
3) в конкретному *контексті*

??? Creational Patterns ???

Патерн Prototype

- Назва та класифікація

Prototype – Прототип – твірний патерн для об’єктів

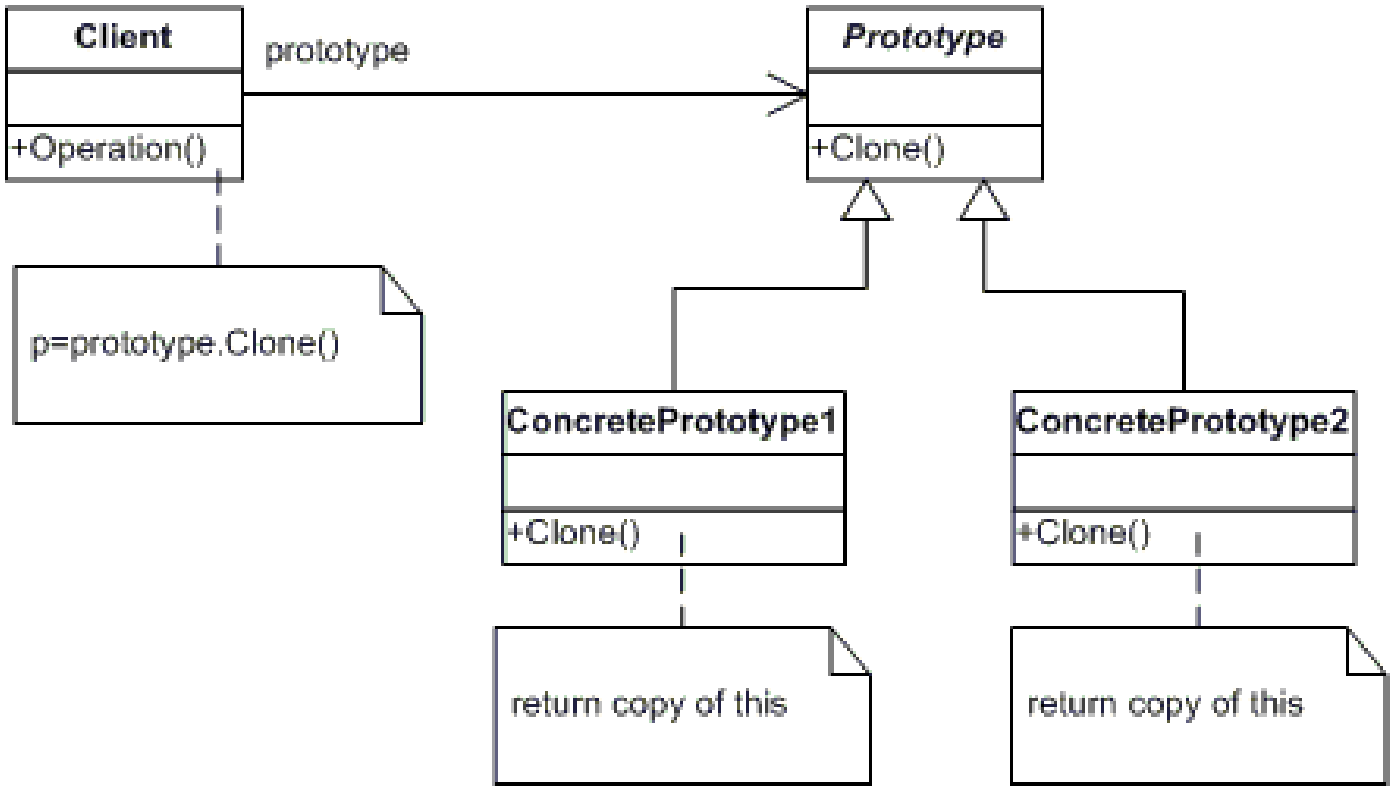
- Призначення

задає **види** об’єктів за допомогою екземпляра-прототипа та утворює нові об’єкти шляхом копіювання цього прототипа

Prototype Застосування

- **Застосування** у таких ситуаціях:
  - клас наперед не має інформації про конкретний тип об’єкта, який треба створити, а лише базовий абстрактний тип;
  - клас спроектовано так, що повноваження на створення об’єктів передається похідним типам

Prototype Структура



Prototype

❑ **Учасники:**

Prototype — **прототип**:

тип оголошує інтерфейс з методом клонування об’єкта цього типу

ConcretePrototype — **конкретний прототип** :

реалізує метод клонування

Client — **клієнт**:

створює новий об’єкт, викликаючи метод клонування прототипу

❑ **Відношення:**

клієнт звертається до **прототипа** із запитом на створення його клону

Prototype
PrototypeRegister

Prototype Результати

- *Ізольованість конкретних типів* (подібно до **AbstractFactory** і **Builder**) : утворення клону виконується через абстрактний інтерфейс; зменшується кількість відомих клієнтові конкретних типів

- Зменшення кількості похідних типів

- Специфікація нових об’єктів за рахунок зміни значень чи структури їхніх змінних

- *Базовий клас продуктів може бути **не абстрактним*** (C++), а реалізовувати мінімальну функціональність, напр. у випадку документу створювати діалогове вікно для вибору файла існуючого документа

Prototype Реалізація

- *Реалізація методу* **Clone**:
  - особливості поверхневого і глибинного копіювання
  - при реалізації інтерфейсу клонування виконувати кастинг типу
- *Ініціалізація клонів*: оскільки метод **Clone** не отримує аргументів, то для зміни стану клону доцільно використовувати інший метод, напр., **Initialize** з необхідним набором аргументів
- *Диспетчер прототипів:*
  - ведеться реєстр (асоційований контейнер) доступних прототипів, з якого клієнт отримує прототип за деяким ключем, а потім робить запит на клонування
  - під час виконання програми клієнт може динамічно змінювати вміст реєстру.

Патерн Abstract Factory

- Назва та класифікація

AbstractFactory – Абстрактна фабрика – твірний патерн

- Призначення

надає інтерфейс для створення взаємопов’язаних або взаємозалежних об’єктів, не специфікуючи їхніх конкретних типів

Abstract Factory Мотивація

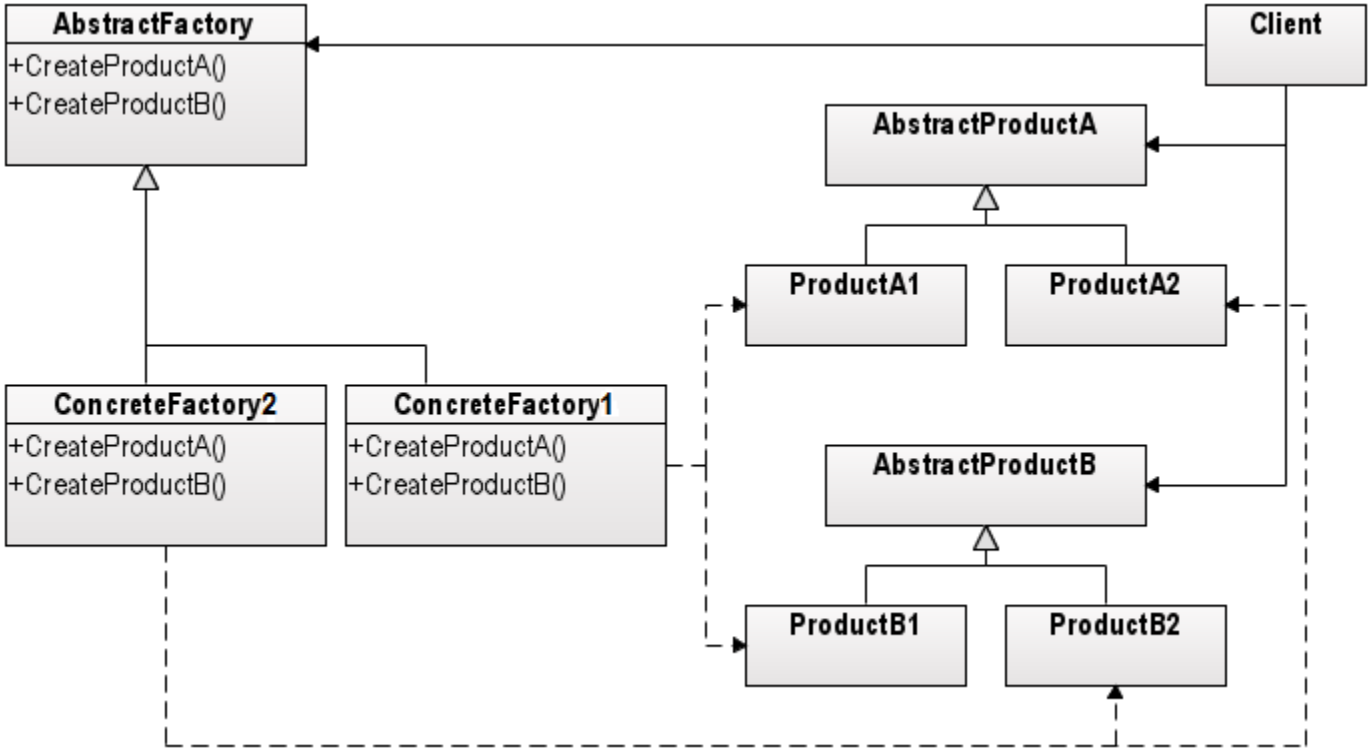
Приклад менеджера GUI, який підтримував би різні стандарти зовнішнього вигляду і поведінки:

- елементи GUI (вікна, кнопки, скролери і інш.) не мають бути жорстко закодовані по всьому коду систем
- у випадку ОО-мови це можна реалізувати за допомогою абстрактних класів як для утворення об’єктів, так і для кожного виду елементів GUI (взаємопов’язаних об’єктів)
- налаштування (конфігурування) системи на конкретний стандарт має виконуватися в конкретному (локалізовано) місці коду за допомогою типу конкретного стандарту
- клієнти не бачать реалізації жодного з конкретних класів, а працюють через інтерфейси абстрактних класів

Abstract Factory Застосування

- Застосування у таких ситуаціях:
  - система не залежить від того, як утворюються, компонуються і подаються її компоненти;
  - взаємопов’язані об’єкти, які входять в систему, мають використовуватися разом і це обмеження має бути дотриманим
  - система конфігурується централізовано (локалізовано в одному місці коду) одним із сімейств об’єктів, які її утворюють (конкретним типом сімейства)
  - для бібліотеки об’єктів розкривається лише їхній інтерфейс (абстрактний клас), але не реалізація

Abstract Factory Структура



AbstractFactory
AbstractFactoryTemplate

Abstract Factory Учасники

AbstractFactory — **абстрактна фабрика**: тип оголошує інтерфейс для операцій, якими створюють продукти;

ConcreteFactory — **конкретна фабрика**: тип реалізує операції, що створюють конкретні продукти, та надає метод доступу до конкретного продукту;

AbstractProduct — **абстрактний продукт**: тип оголошує інтерфейс для типу об'єкта-продукту;

ConcreteProduct — **конкретний продукт**: визначає тип об'єкта-продукта, який може утворювати відповідна конкретна фабрика, реалізує інтерфейс **AbstractProduct** ;

Client — **клієнт**: користується виключно інтерфейсами, які оголошенні у класах **AbstractFactory** та **AbstractProduct**.

Abstract Factory Відношення

- **клієнт** під час виконання створює єдиний екземпляр **конкретної фабрики** і ця конкретна фабрика створює об'єкти-продукти за допомогою методів, які є в інтерфейсі **AbstractFactory**
- **клієнт** використовує методи інтерфейсу **AbstractFactory** з об’єктом **конкретної фабрики** як аргументом; завдяки віртуальності, це приводить до виклику відповідних методів **ConcreteFactory**, які і утворюють об’єкт **конкретного продукту**, адреса якого повертається клієнтові
- Для створення інших видів об'єктів клієнт повинен користуватися іншою конкретною фабрикою.

Abstract Factory Результати

- *Ізольованість конкретних типів*: фабрика відповідає за процес утворення об’єкта і ізолює клієнта від деталей реалізації; клієнти маніпулюють об’єктами через абстрактні інтерфейси
- *Проста заміна сімейства продуктів*; для цього достатньо замінити тип конкретної фабрики, кількість конкретних фабрик необмежена
- *Гарантія сумісності об’єктів*: за всіх відповідає єдина конкретна фабрика
- *Розширення інтерфейсу* **AbstractFactory** вимагає значного перепрограмування

Abstract Factory Реалізація

- *Конкретні фабрики доцільно мати в одному екземплярі* на кожен клас продукту (патерн **Singleton**)
- *Створення продуктів*: відповідальність за типом **ConcreteProduct**, але для виклику відповідного конструктора конкретна фабрика може мати фабричний метод (патерн **Factory Method**)
- Якщо сімейств продуктів багато, то можна скористатися патерном **Prototype**
- Розширення інтерфейсу абстрактного класу **AbstractFactory** можна виконати за рахунок параметризації (компроміс між гнучкістю та надійністю: проблема наступного приведення до потрібного типу)
- У простих випадках **AbstractFactory** може бути не абстрактним, а повноцінним типом, тоді похідні типи мають перевизначати лише потрібні методи

Патерн Factory Method

- Назва та класифікація

Factory Method – фабричний метод – твірний патерн для класів

- Призначення

надає інтерфейс для створення об’єкта, конкретний тип якого визначається з ієрархії похідних типів

- Відомий також під назвою

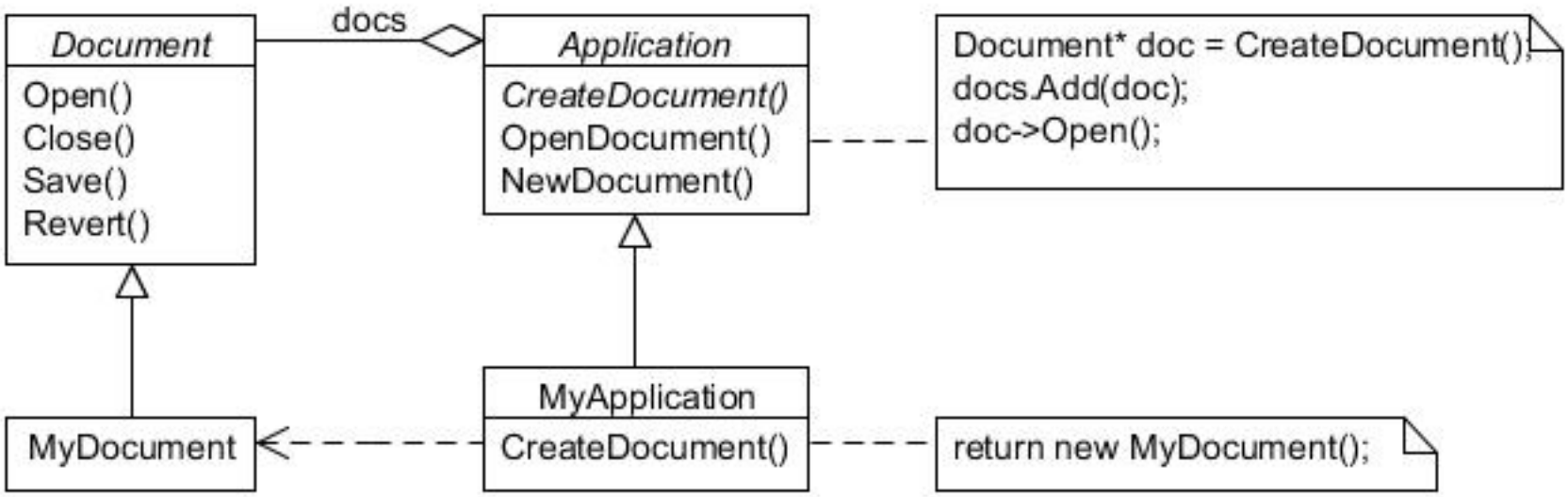
Virtual Constructor

Factory Method Мотивація

Приклад: код, який підтримував би створення документів різних типів, про які лише відомо, що вони походять від деякого абстрактного класу, напр., **Document** :

- у коді клієнта локалізовано створення документа шляхом виклику методу **CreateDocument** для створення об’єкта документа без уточнення його типу
- інформація про конкретний тип винесена за межі коду клієнта
- налаштування (конфігурування) системи на конкретний стандарт має виконуватися в конкретному (локалізовано) місці коду за допомогою типу конкретного стандарту
- клієнти не бачать реалізації жодного з конкретних класів, а працюють через інтерфейси абстрактних класів

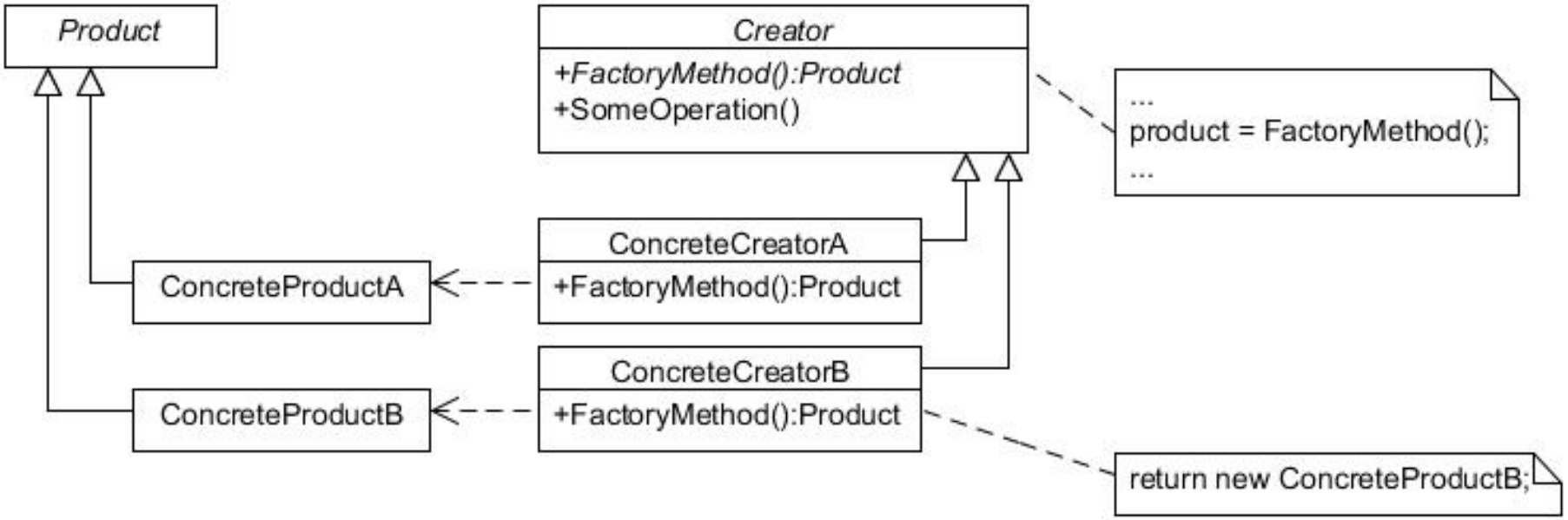
Factory Method Приклад



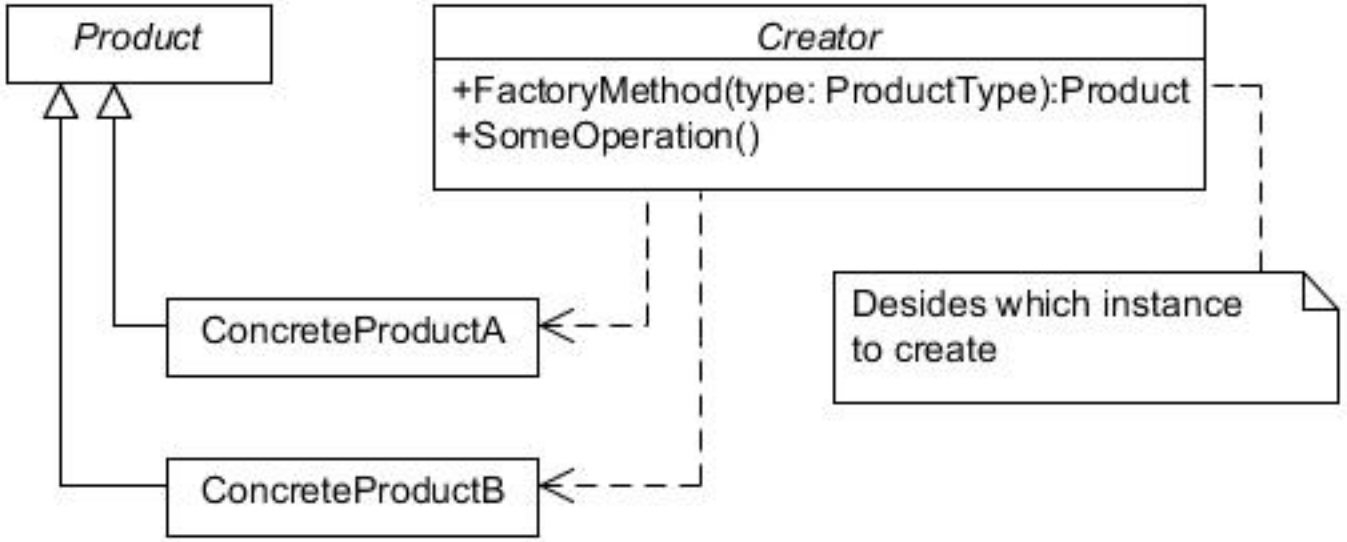
Factory Method Застосування

- Застосування у таких ситуаціях:
  - клас наперед не має інформації про конкретний тип об’єкта, який треба створити, а лише базовий абстрактний тип;
  - клас спроектовано так, що повноваження на створення об’єктів передається похідним типам

Factory Method Структура



Factory Method Структура



FactoryMethod,
FactoryMethod_Sandwich

Factory Method

❑ **Учасники:**

Creator — **творець**:

тип оголошує інтерфейс – фабричний метод, який призначений для створення об’єкта і перевизначений у похідному типі **ConcreteCreator**; може викликати фабричний метод для створення об’єкта **Product**

ConcreteCreator — **конкретний творець**:

перевизначає фабричний метод, який повертає **ConcreteProduct**

Product — **продукт**:

тип оголошує інтерфейс для типу об'єкта-продукту;

ConcreteProduct — **конкретний продукт**:

реалізує інтерфейс **Product** ;

❑ **Відношення:**

творець делегує похідному типові створення у перевизначеному фабричному методі об’єкта **конкретного продукту**

Factory Method Результати

- *І зольованість конкретних типів*:
  - творець відповідає за процес утворення об’єкта і ізолює клієнта від деталей реалізації
  - клієнти маніпулюють об’єктами через абстрактні інтерфейси
- *Базовий клас продуктів може бути не абстрактним,* а реалізовувати мінімальну функціональність, напр. у випадку документу створювати діалогове вікно для вибору файла існуючого документа

Factory Method Реалізація

- *Основні різновидності патерну:*
  1. **Creator** абстрактний, тому необхідна реалізація з перевизначеним віртуальним методом
  2. **Creator** має реалізацію методу за замовчуванням
- *Параметризовані фабричні методи*: у фабричний метод передається додатковий аргумент
- *Параметризований творець*: тип **Creator** параметризується типом продукту

Патерн Singleton

- Назва та класифікація

Singleton (Одинак) – патерн, який застосовують для утворення об'єктів

- Призначення

гарантує, що може існувати не більше одного об'єкта даного класу, і надає доступ до нього

- Мотивація

Сам клас повинен гарантувати єдиність об'єкта, легкий доступ до нього, може заборонити утворення нових об'єктів, або керувати утворенням пулу об'єктів

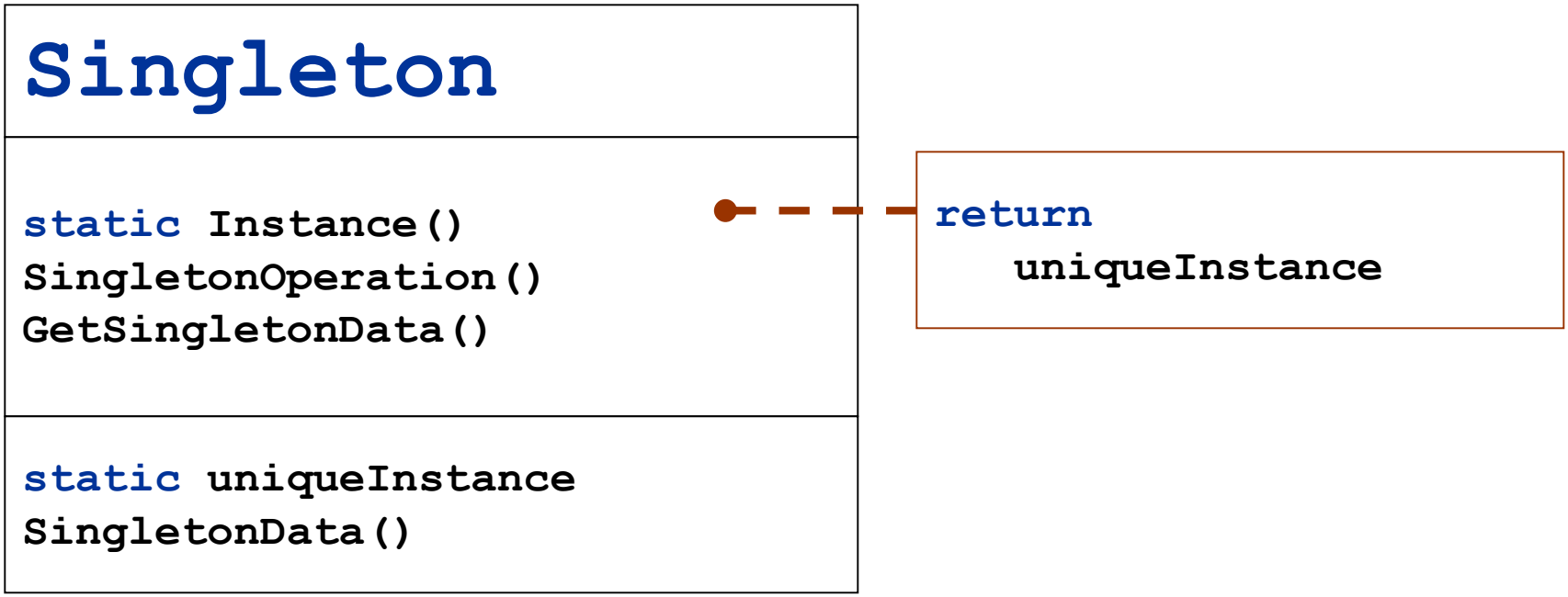
- Застосування

Шаблон застосовують у таких ситуаціях:

– повинен існувати лише один об'єкт даного класу, легко доступний усім клієнтам

– єдиний екземпляр може бути розширений за рахунок похідних класів

Singleton Структура



Singleton

- **Відношення**

  Клієнти отримують доступ до об’єкта класу **Singleton** лише операцією (методом) **Instance**

- **Результати**

  - Контрольований доступ до єдиного інкапсульованого екземпляра
  - Зменшення кількості ідентифікаторів (глобальних змінних)
  - Уточнення операцій та стану за рахунок похідних типів
  - Проста модифікація з метою утворення пулу об'єктів і керування ними

Singleton Реалізація

- **Гарантування єдиності об'єкта (C++)**
  - єдиний інкапсульований **статичний** об'єкт

    (синглетна природа статичних полів)

    DP_Singleton.Singleton
  - *відкладена ініціалізація* - об'єкт доступний через єдиний відкритий метод (не **inline**), який може контролювати утворення об'єкта та його ініціалізацію(**Scott Meyers**)

    DP_Singleton.SingletonInTime
  - використання вбудованого **Singleton-**об’єкта

    DP_Singleton.SingletonRelation
- **Утворення похідних типів**
  - перенесення реалізації методу **Instance()** в похідний клас

    DP_Singleton.DerivedSingleton
  - параметризований **Singleton** для перетворення довільного класу в синглет

    DP_Singleton.DerivedSingleton

Singleton

- **Відомі застосування**

  ???

- **Споріднені патерни**

  За допомогою **Singleton** можна реалізувати багато інших патернів:

  - **Abstract Factory**
  - **Builder**
  - **Prototype**

SingleCarFactory

Патерн `Builder`

- Назва та класифікація

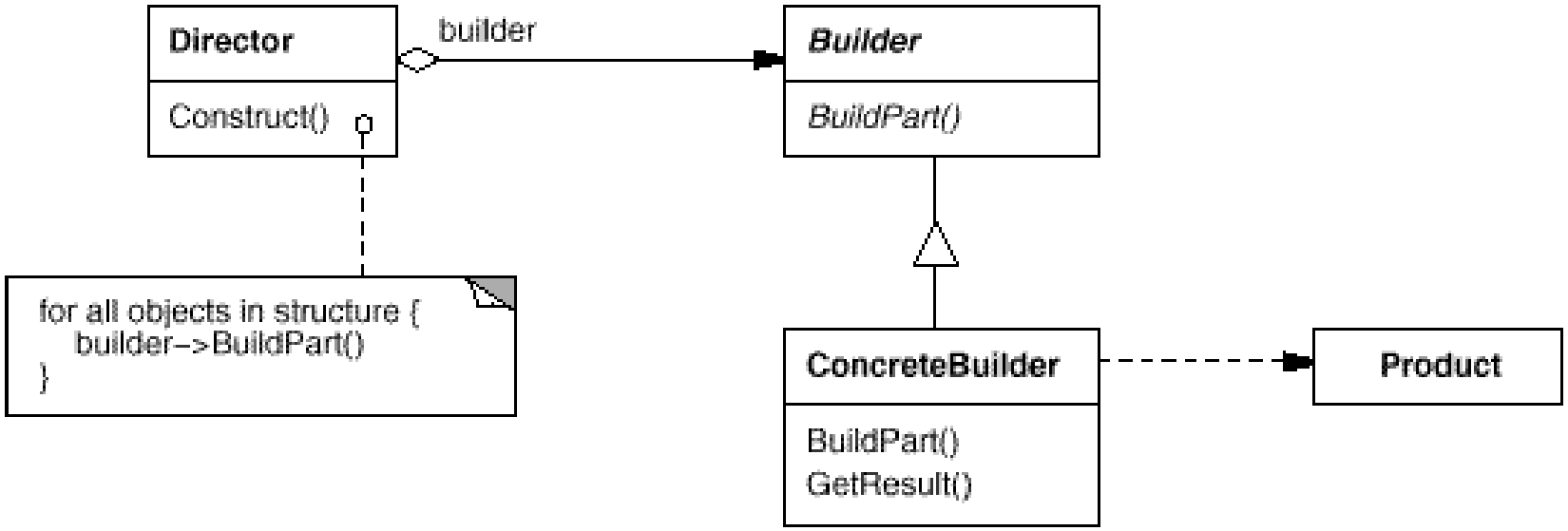
  `Builder` – Будівник – твірний патерн

- Призначення
  - відокремлює алгоритм конструювання складного об’єкта від конкретного типу, методами якого утворюватимуться компоненти цього складного об’єкта
  - у результаті виконання **незмінного алгоритму** конструювання за рахунок зміни типів компонентів утворюватимуться **об’єкти різних типів**

Патерн Builder

- Мотивація:
  - алгоритм утворення складного об’єкта не залежить від типу конкретного будівника
  - методи будівника є доступні через абстрактний клас чи інтерфейс
  - кількість конкретних типів будівельників наперед не відома, кожен з них спеціалізований на побудову складного об’єкта конкретного типу
- Застосування у таких ситуаціях:
  - алгоритм створення складного об'єкта не залежить від його конкретного типу і стикування його компонентів поміж собою
  - один і той же алгоритм конструювання виконується для різних типів складного об'єкта

Builder Структура



Builder Учасники

Builder — **будівник**:

- визначає абстрактний інтерфейс для створення частин об'єкта **Product**

ConcreteBuilder — **конкретний будівник**:

- реалізує інтерфейс **Builder**
- надає метод доступу до конкретного продукту

Director — **управитель**:

- конструює об'єкт продукту, користуючись інтерфейсом **Builder**

Product — **продукт**:

- подає складний конструйований об’єкт; кожен **ConcreteBuilder** будує по-своєму внутрішнє подання продукту та визначає процес його збірки
- може задавати класи компонентів

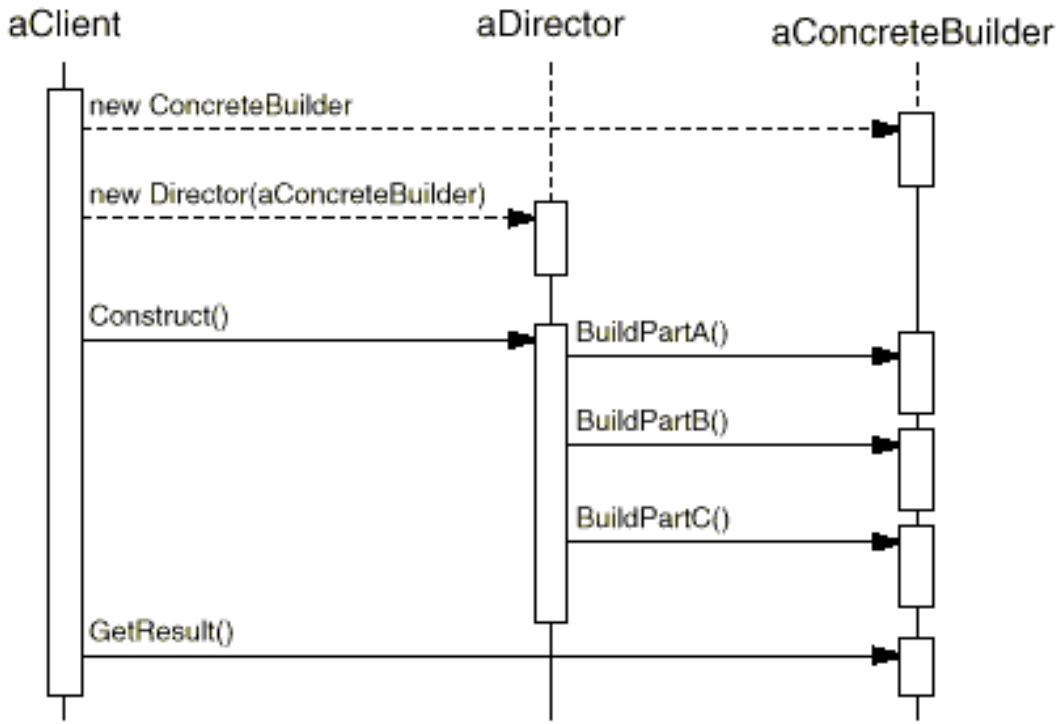
Builder Відношення

1. **клієнт** утворює об'єкт-**управитель** **Director** та конфігурує його потрібним об'єктом-**будівником** **ConcreteBuilder**;

2. **управитель** в згідно фіксованого алгоритму повідомляє **будівника** про те, що потрібно побудувати частини **продукту**;

3. **будівник** оброблює запити **управителя** та послідовно додає частини до **продукту**;

4. **клієнт** отримує **продукт** у **будівника**.



Builder Результати

- *Модульність коду*: інкапсуляція алгоритму побудови складного об’єкта в одному типі (управителя), а конкретних методів утворення частин – в інших типах (будівельниках), які мають весь необхідний код для створення конкретного об’єкта
- *Проста зміна внутрішньої будови складного об’єкта* без зміни алгоритму його утворення; для цього достатньо замінити тип будівельника
- *Контроль над процесом конструювання*: на відміну від інших твірних патернів будівельник конструює об’єкт по кроках, що дає змогу контролювати як сам процес, так і внутрішню структуру готового продукту

Builder Реалізація

- **Абстактний** клас будівельника надає інтерфейс у вигляді набору порожніх (реалізованих) методів
- Якщо конкретний будівельник не потребує якогось компонента, то відповідний метод не перевизначають
- Абстрактного класу для продукту не передбачено, оскільки продукти можуть не мати нічого спільного. Натомість всю інформацію про продукт може мати відповідний конкретний будівельник