Chương 7

Các mẫu thiết kế phục vụ khởi tạo đối tượng (Creational Patterns)

7.1 Tổng quát về nhóm mẫu "Creational"
7.2 Mẫu Abstract Factory
7.3 Mẫu Factory Method
7.4 Mẫu Prototype
7.5 Mẫu Builder
7.6 Mẫu Singleton
7.7 Kết chương

7.1 Tổng quát về nhóm mẫu "Creational"

- Đối tượng thường chứa nhiều thuộc tính dữ liệu và nhiều tham khảo đến các đối tượng khác. Đoạn code khởi tạo giá trị cho các thuộc tính và tham khảo này phụ thuộc hoàn toàn vào sự hiện thực cụ thể của đối tượng. Chúng ta cần tách đoạn code này khỏi code của client sử dụng đối tượng.
- Thường client muốn tạo đối tượng phức hợp mà không cần quan tâm đến các thông tin cụ thể chi tiết sau :
  - Đối tượng phức hợp thuộc class cụ thể nào.
  - Nó chứa các loại đối tượng con nào, số lượng ra sao.
  - Mối quan hệ giữa các đối tượng con như thế nào.

7.1 Tổng quát về nhóm mẫu “Creational”

- Các mẫu thiết kế “Creational” sẽ giúp code client linh động về mặt khởi tạo, quản lý và sử dụng đối tượng. Chúng có thể cho phép client chủ động trong việc xác định đối tượng nào được tạo ra, ai tạo ra đối tượng đó, cách thức và thời điểm khởi tạo đối tượng đó.
- Đặc điểm nổi bật của các mẫu thiết kế “creational” là đoạn code client cần sử dụng đối tượng không trực tiếp sinh ra đối tượng mà nhờ các phần tử trung gian để tăng độ linh động, tối thiểu hóa sự phụ thuộc vào các class đối tượng cụ thể.

7.2 Mẫu Abstract Factory

Mục tiêu :

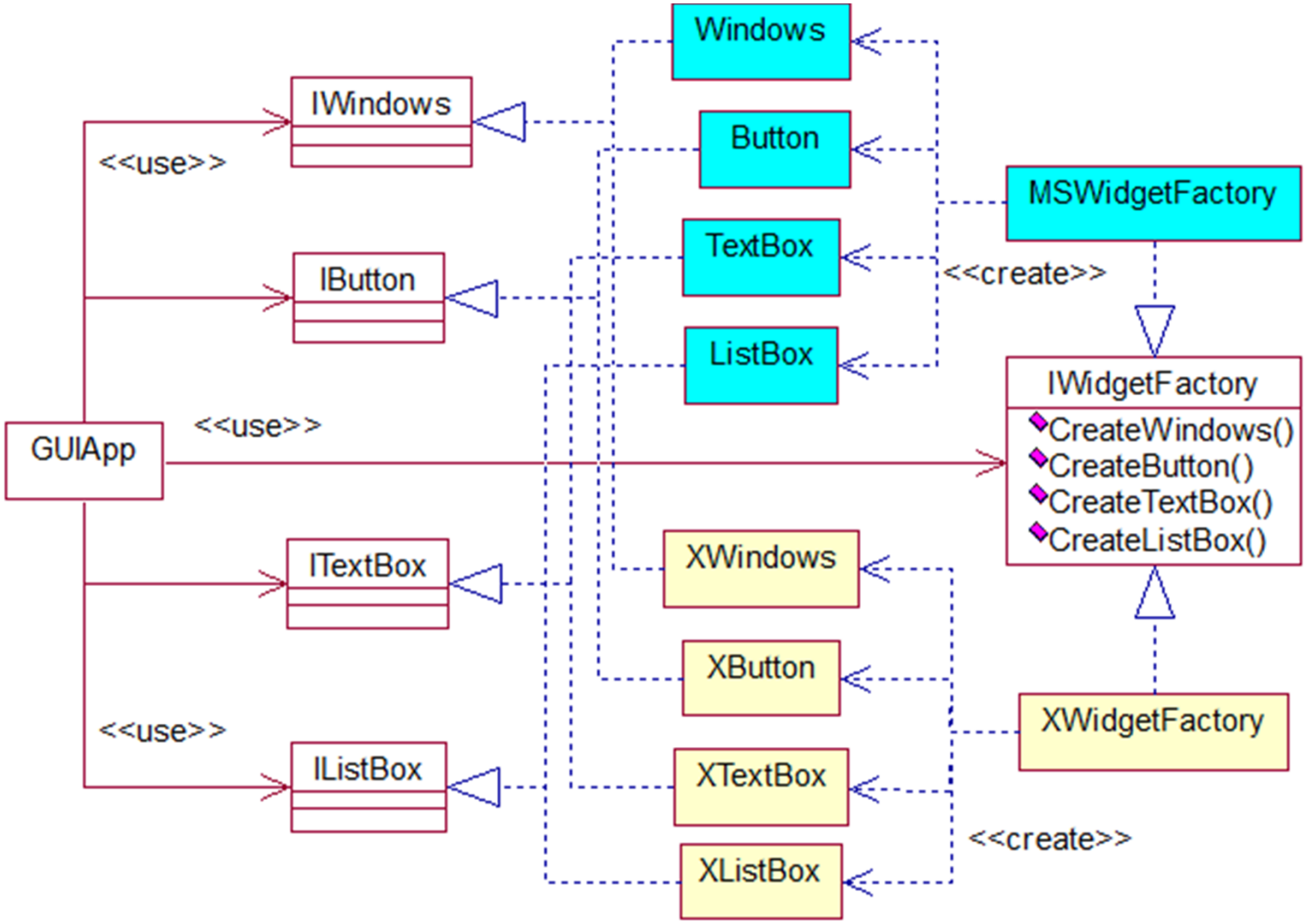
- Cung cấp cho Client 1 interface gồm tập tác vụ khởi tạo các đối tượng được dùng trong Client nhưng che dấu mọi chi tiết về đối tượng được tạo ra, thí dụ như thuộc class cụ thể nào…
- Một trong các mục tiêu của việc viết code ứng dụng là phải có tính tổng quát cao và độ độc lập cao với chi tiết hiện thực của các đối tượng mà nó dùng. Một trong các phương án để đạt được mục tiêu trên là không dùng lệnh new để tạo ra đối tượng cần dùng vì nếu dùng lệnh new thì phải xác định tên class cụ thể và như vậy sẽ phụ thuộc vào class này. Mẫu AbstractFactory sẽ giúp chúng ta tạo đối tượng mà không cần biết tên class cụ thể được dùng.

7.2 Mẫu Abstract Factory

Thí dụ về việc dùng mẫu Abstract factory :

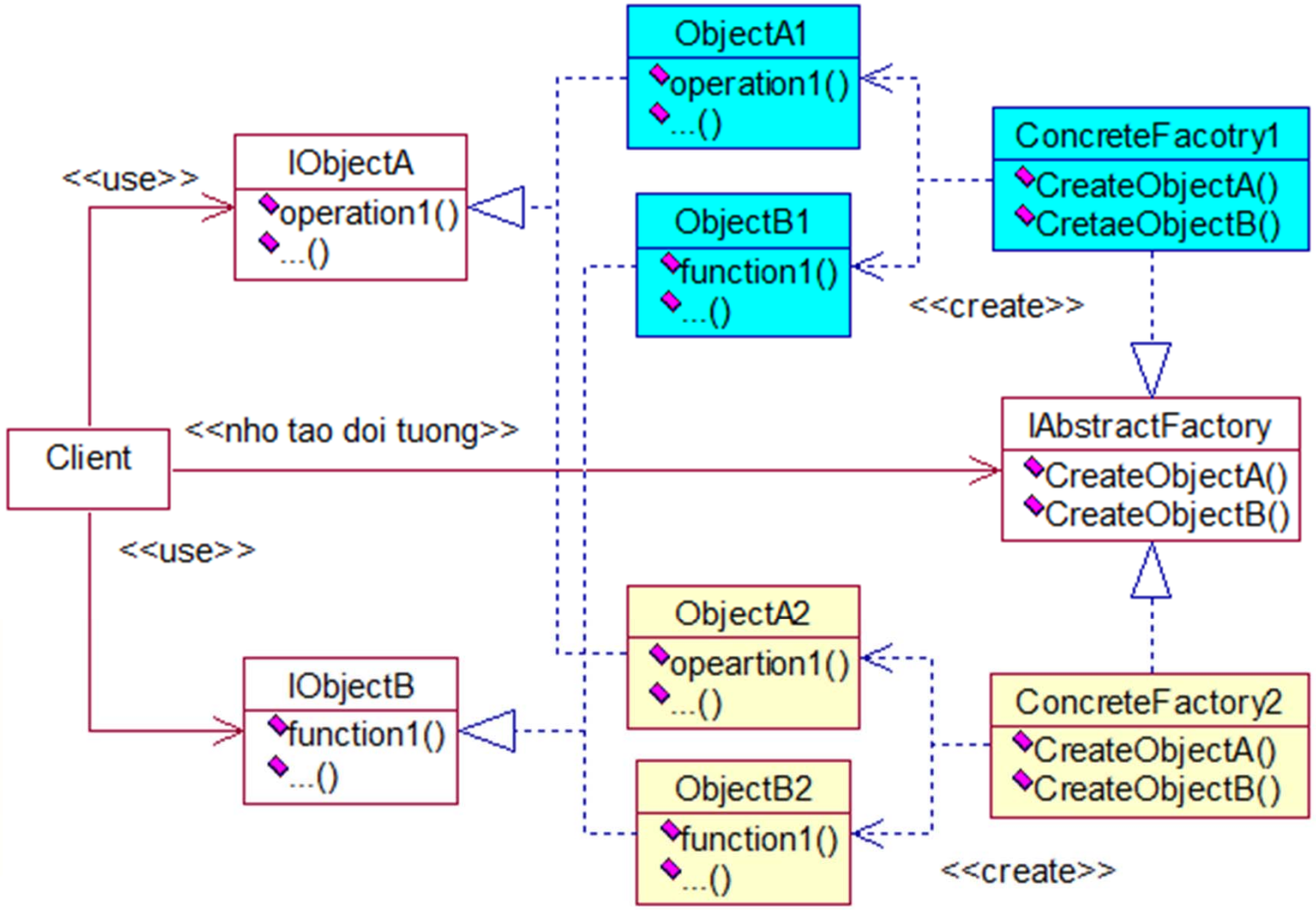
- Ta muốn viết chương trình GUI (dùng giao diện đồ họa trực quan) có khả năng chạy trên nhiều platform khác nhau (chương trình multi-platform). Mỗi platform có một họ các class miêu tả các đối tượng giao diện phổ dụng như TextBox, Button,… Giả sử các đối tượng giao diện cùng chức năng trên các platform khác nhau hỗ trợ cùng interface thống nhất (nếu không ta dùng mẫu Adapter để có được kết quả này). Việc sử dụng cụ thể họ các class giao diện nào chỉ biết khi chương trình bắt đầu chạy. Để giải quyết vấn đề trên, cách tốt nhất là dùng mẫu AbstractFactory với lược đồ class như sau :

7.2 Mẫu Abstract Factory



7.2 Mẫu Abstract Factory

Ta có thể xây dựng mẫu AbstractFactory theo loại class pattern với lược đồ class như sau :



7.2 Mẫu Abstract Factory

Các phần tử tham gia :

- IObjectA, IObjectB : interface thống nhất của các đối tượng cùng chức năng trong các họ khác nhau.
- IAbstractFactory : interface của đối tượng chuyên tạo các đối tượng dùm cho Client.
- ConcreteFactory1… : class hiện thực các tác vụ tạo đối tượng trong interface IAbstractFactory. Hệ thống có nhiều ConcreteFactory, mỗi ConcreteFactory sinh ra các đối tượng cùng họ, các họ đối tượng do các ConcreteFactory tạo ra tương đồng nhau về vai trò, về chức năng.

7.2 Mẫu Abstract Factory

Các phần tử tham gia (tt) :

- ObjectA, ObjectB : class hiện thực đối tượng được tạo ra bởi class ConcreteFactory tương ứng, nó hỗ trợ interface sử dụng tương ứng.
- Client : đoạn code cần tạo và sử dụng các đối tượng. Client chỉ sử dụng các interface IAbstractFactory, IObjectA, IObjectB,… và như thế hoàn toàn độc lập với các class cụ thể mà nó sẽ dùng.

7.2 Mẫu Abstract Factory

Quá trình tương tác giữa các phần tử :

- Tại thời điểm dịch, Client có biến tham khảo đến đối tượng IAbstractFactory (giả sử tên là factory).
- Tại thời điểm run-time, dựa vào ngữ cảnh, Client sẽ biết cần sử dụng họ đối tượng nào và nó sẽ tạo đối tượng ConcreteFactory tương ứng rồi gán tham khảo vào biến factory. Sau đó trong giải thuật, mỗi lần tạo đối tượng cần dùng, thông qua biến factory, Client sẽ gởi thông điệp để yêu cầu ConcreteFactory tạo ra đối tượng mong muốn.
- Client dựa vào interface của đối tượng được tạo ra để sử dụng đối tượng này.

7.3 Mẫu Factory Method

Mục tiêu :

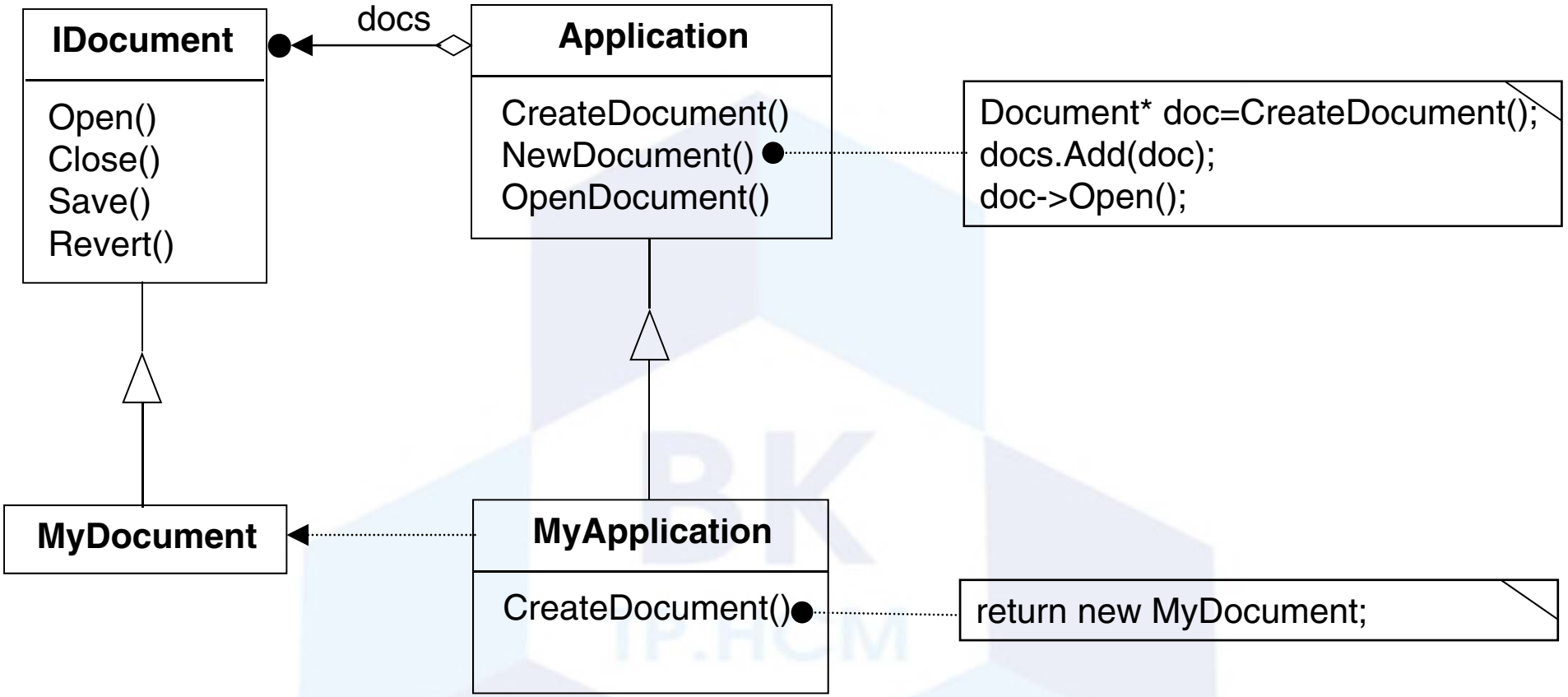
- Client định nghĩa interface các tác vụ khởi tạo đối tượng phức hợp cần dùng rồi sử dụng các tác vụ này trong đoạn code của mình, nhưng không hiện thực cụ thể các tác vụ khởi tạo đối tượng mà để class con của mình override và hiện thực theo yêu cầu riêng của từng class con.

7.3 Mẫu Factory Method

Thí dụ về việc dùng mẫu Factory method :

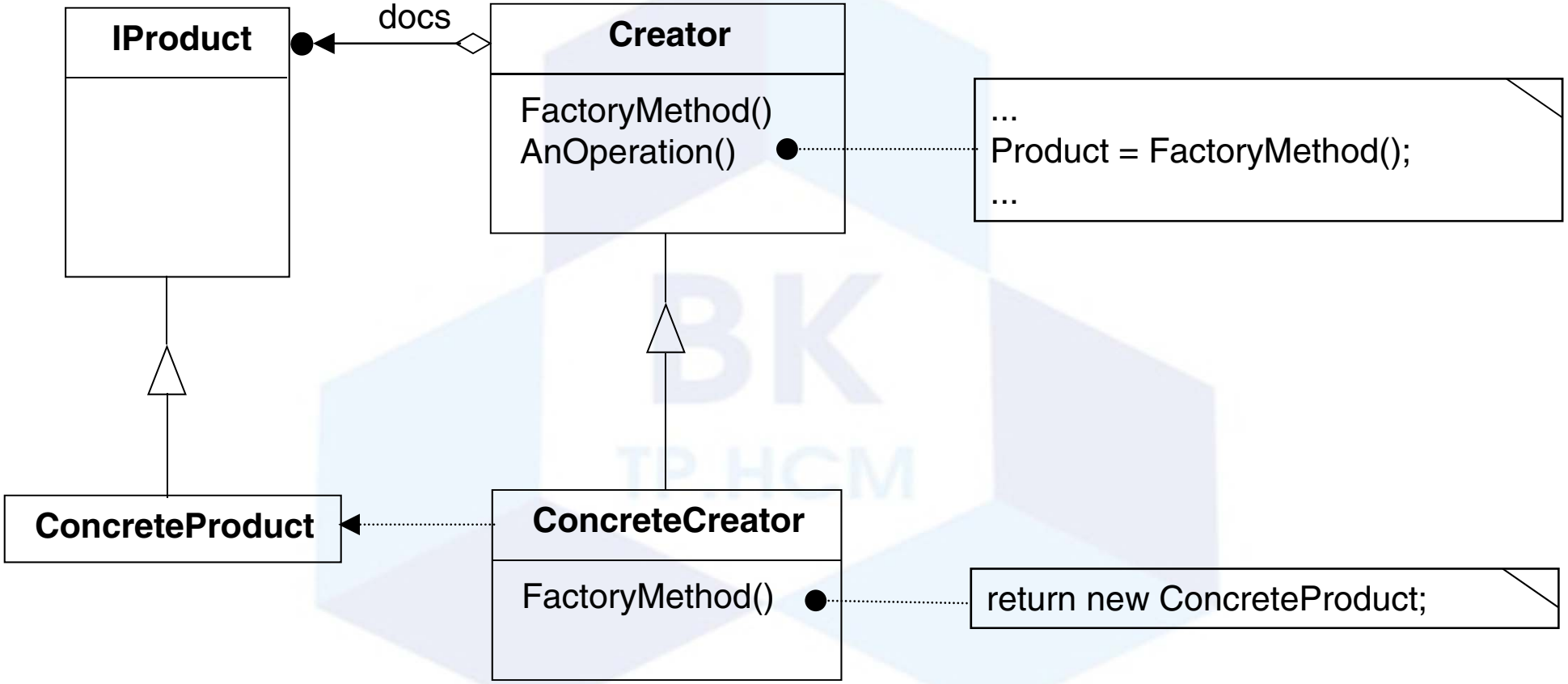
- Các ứng dụng trong bộ MS Office như Word, Excel, PowerPoint đều có cơ chế khởi tạo, xử lý tài liệu giống nhau cho dù cấu trúc vật lý của các tài liệu trong các ứng dụng là khác nhau. Để viết đoạn code xử lý tài liệu trong các ứng dụng MS Office nhất quán, độc lập với tài liệu cụ thể, Microsoft đã dùng mẫu Factory Method như sau để thiết kế các ứng dụng MS Office :

7.3 Mẫu Factory Method



7.3 Mẫu Factory Method

Ta có thể xây dựng mẫu Factory Method theo loại class pattern với lược đồ class như sau :



7.3 Mẫu Factory Method

Các phần tử tham gia :

- Creator (Application) : class chứa code client cần tạo và sử dụng đối tượng phức hợp, nó sẽ chứa đặc tả các tác vụ khởi tạo đối tượng phức hợp cần dùng.
- ConcreteCreator (MyApplication) : class hiện thực các tác vụ tạo đối tượng trong class cha Creator.
- IProduct (IDocument) : interface của đối tượng phức hợp cần tạo và sử dụng.
- ConcreteProduct (MyDocument) : class miêu tả đối tượng phức hợp cụ thể cần tạo và sử dụng.

7.4 Mẫu Prototype

Mục tiêu :

- Tạo đối tượng mới có nội dung y như đối tượng đã có sẵn.

7.4 Mẫu Prototype

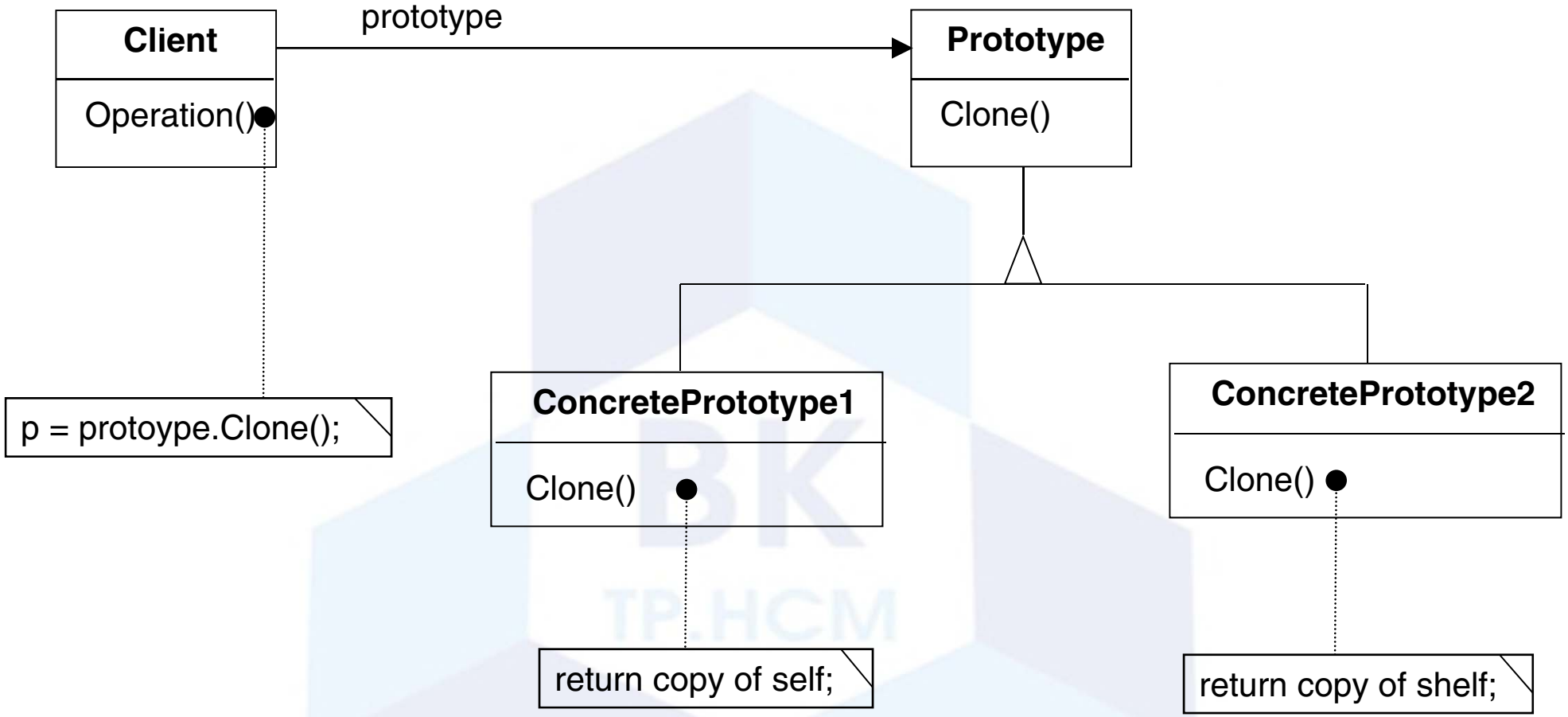
Thí dụ về việc dùng mẫu Prototype :

- Có nhiều khi ta muốn tạo đối tượng mới có cấu trúc và nội dung cụ thể giống y như một đội tượng đã có sẵn. Cách tốt nhất là dùng mẫu Prototype, đối tượng có sẵn được gọi là Prototype và phải chứa ít nhất tác vụ Clone() có nhiệm vụ trả về tham khảo đến đối tượng có cấu trúc và nội dung giống như như mình (đối tượng đó có thể chính là mình).
- Thường dùng trong các phát biểu gán đối tượng nhưng theo nghĩa giống như lệnh gán dữ liệu cổ điển :

```
int i1 = 5;
Int i2 = i1; //i2 = 5
I1 = 0; //i2 vẫn =5
```

```
C1 c1 = new C1(); //có c1.i = 0;
c1.i = 5;
C1 c2 = c1.Clone(); //c2.i sẽ = 5;
c1.i = 0; //c2.i vẫn =5
```

7.4 Mẫu Prototype



7.4 Mẫu Prototype

Các phần tử tham gia :

- Prototype : interface miêu tả các tác vụ của đối tượng phức hợp cần dùng bởi Client, trong đó cần có tác vụ Clone() với chức năng tạo ra đối tượng có cầu trúc và nội dung giống hệt mình.
- ConcretePrototype1,… : class hiện thực các tác vụ trong class cha Prototype theo cách riêng của mình.
- Client : class miêu tả đoạn code tạo và sử dụng các đối tượng ConcretePrototype1, ConcretePrototype2...

7.5 Mẫu Builder

Mục tiêu :

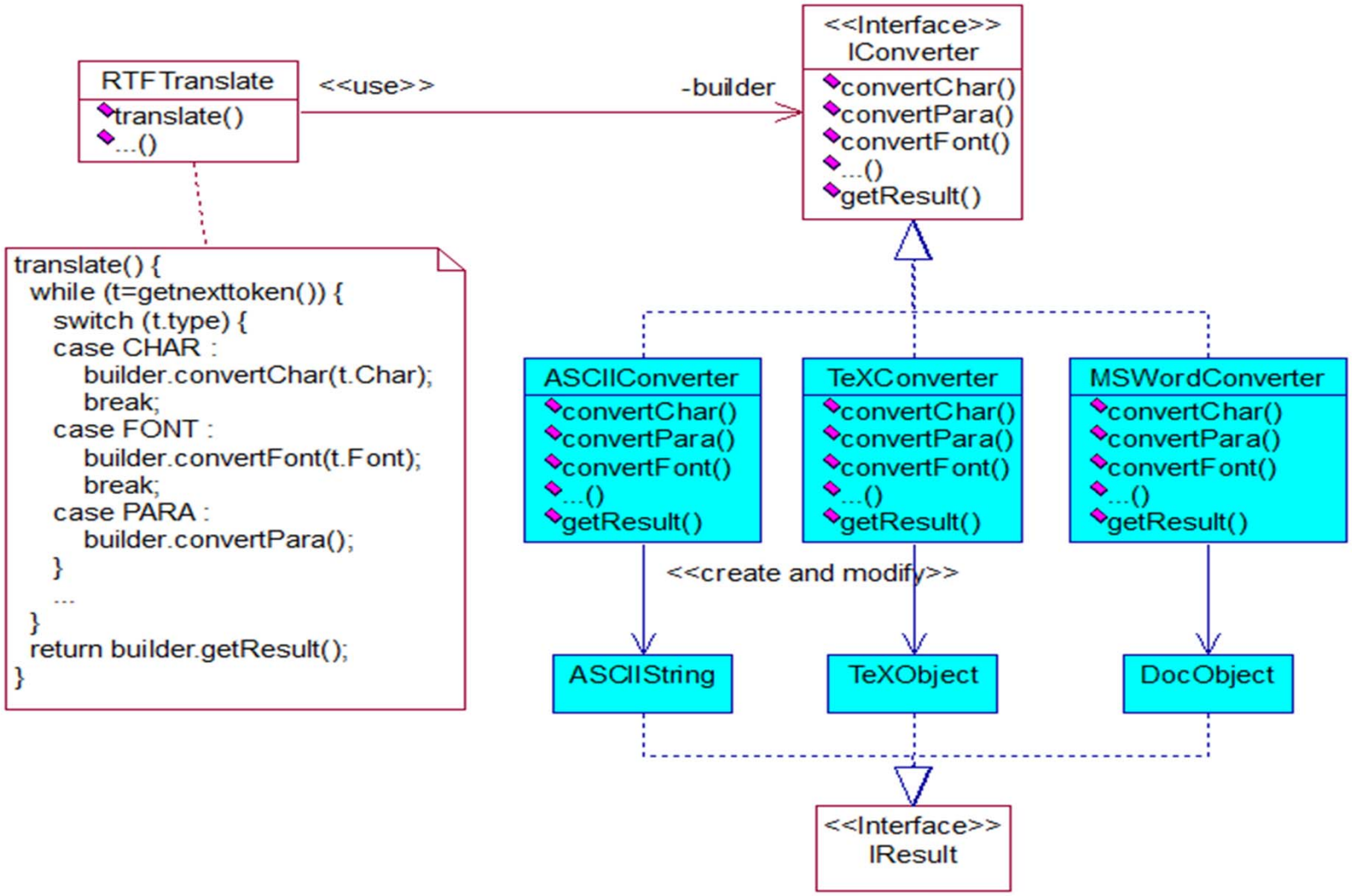
- Giúp Client khởi tạo đối tượng phức hợp (bao gộp) theo cơ chế tích lũy tăng dần thông qua nhiều lần gọi dịch vụ của đối tượng Builder.
- Client không cần biết các thông tin hiện thực của đối tượng đó. Cụ thể Client không cần biết đối tượng được tạo ra thuộc class nào, có cấu trúc cụ thể nào, các đối tượng thành phần thuộc class cụ thể nào…

7.5 Mẫu Builder

Thí dụ về việc dùng mẫu Builder :

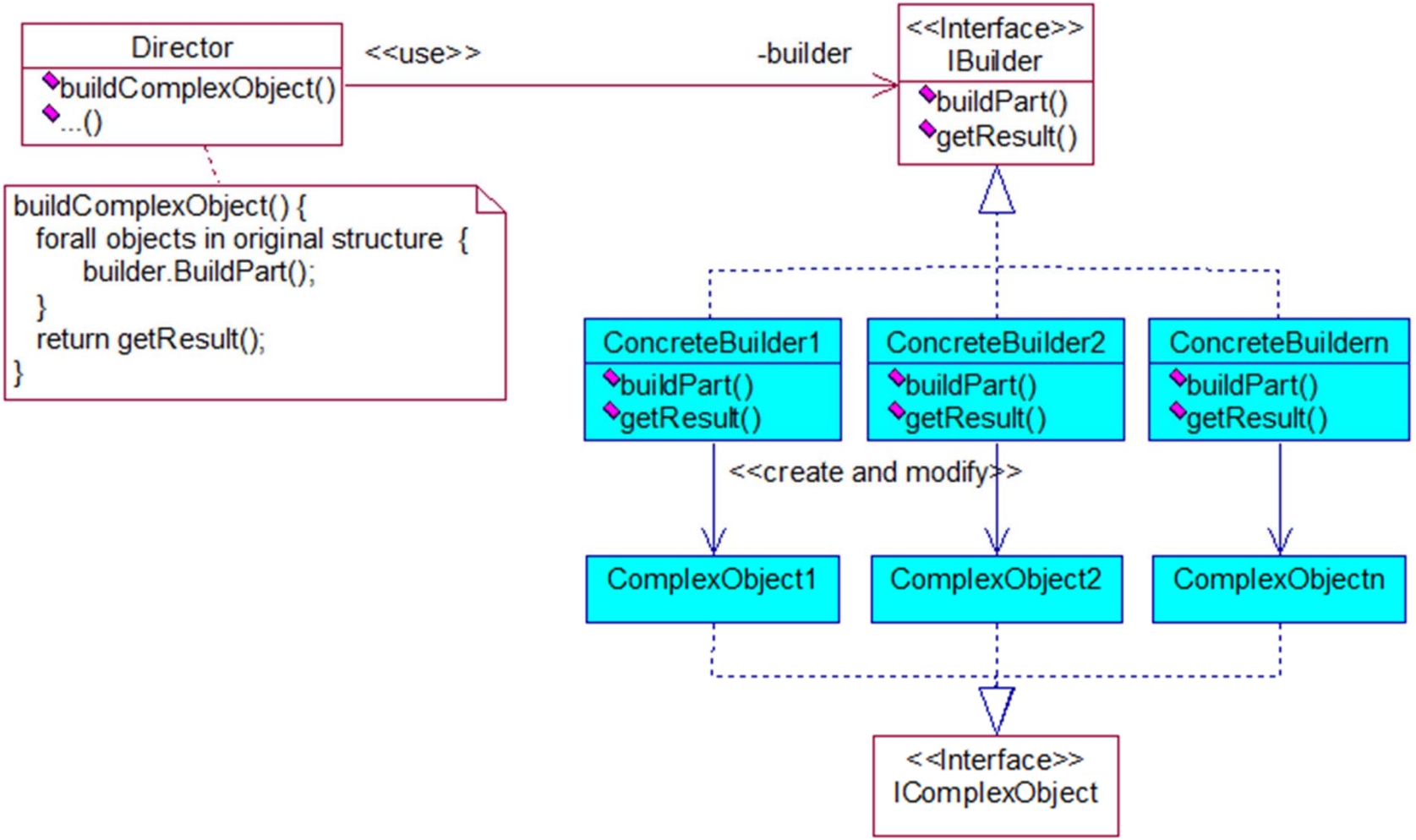
- Ta muốn viết chương trình RTFTranslate có chức năng đọc file tài liệu theo định dạng RTF rồi chuyển nội dung gốc thành 1 định dạng xác định nào đó, thí dụ như chuỗi văn bản thô mã ASCII, hay định dạng tài liệu khoa học TeX, hay định dạng MSWord,… Cho dù cần chuyển về định dạng nào (biết trước hay chưa biết trước), ta muốn đoạn code thực hiện chuyển định dạng phải có tính tổng quát và độc lập với định dạng đích.
- Cách tốt nhất để giải quyết vấn đề trên là dùng mẫu Builder với lược đồ class như sau :

7.5 Mẫu Builder



7.5 Mẫu Builder

Ta có thể xây dựng mẫu Builder theo loại class pattern với lược đồ class như sau :



7.5 Mẫu Builder

Các phần tử tham gia :

- IBuilder (IConverter) : interface thống nhất của các builder khác nhau.
- Dicrector (RTFTranslate) : đoạn code Client thực hiện giải thuật tạo và phát triển đối tượng phức hợp, đoạn code này phải độc lập với chi tiết cụ thể về đối tượng phức hợp được tạo ra. Director chỉ sử dụng đối tượng phức hợp thông qua interface thống nhất IComplexObject.
- ConcreteBuilder1… (ASCIIConverter…) : class đặc tả 1 builder cụ thể, builder này sẽ tạo và phát triển đối tượng phức hợp theo cấu trúc riêng do nó quản lý. Nó cũng cung cấp tác vụ getResult() để trả về tham khảo đến đối tượng phức hợp sau khi xây dựng xong, đối tượng phức hợp này phải hiện thực interface IComplexObject.

7.5 Mẫu Builder

Các phần tử tham gia :

- IComplexObject (IResult) : interface thống nhất của các đối tượng phức hợp mà các builder khác nhau tạo ra.
- ComplexObject1… (ASCIIString…) : class đặc tả 1 đối tượng phức hợp cụ thể do 1 builder cụ thể tạo ra.

7.6 Mẫu Singleton

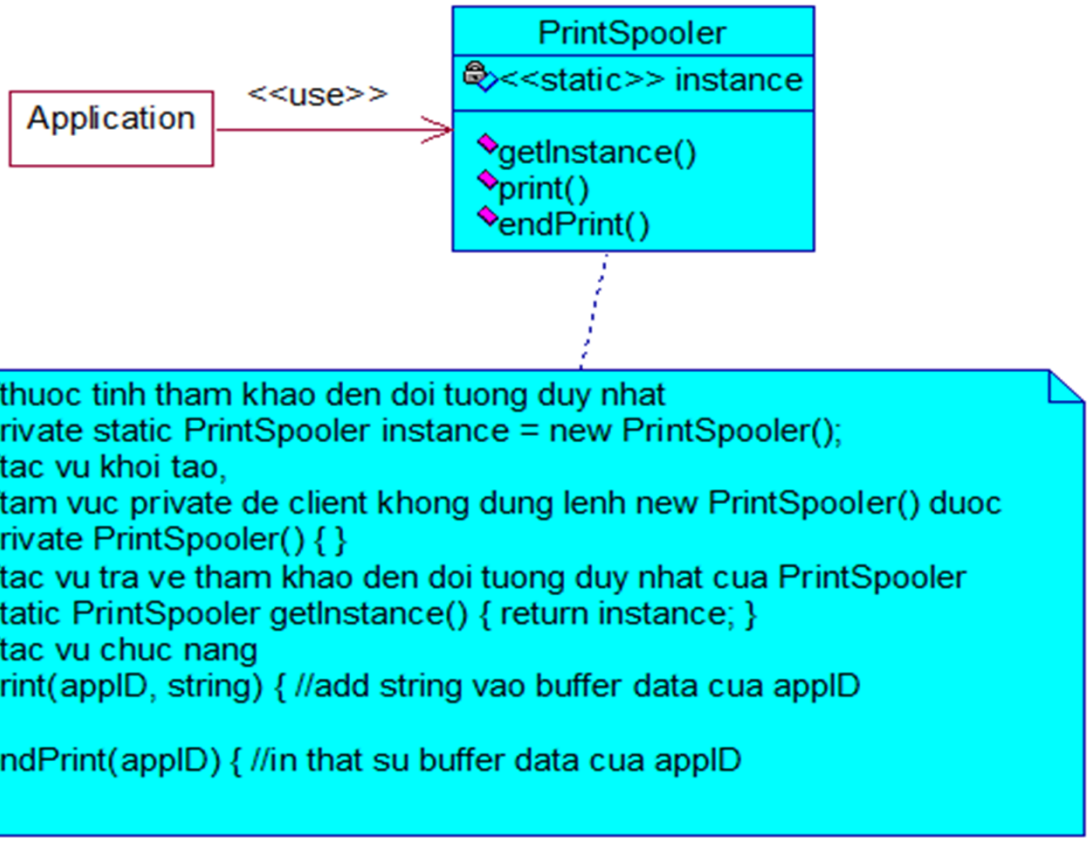
Mục tiêu :

- Đảm bảo chỉ có thể tạo ra 1 instance duy nhất của 1 class chức năng xác định và cấp phát tham khảo đến đối tượng duy nhất này cho các client để chúng dùng chung.
- Thực tế trong nhiều hoạt động của hệ thống, để đảm bảo tính nhất quán và đúng đắn trong việc sử dụng tài nguyên của hệ thống, ta phải đảm bảo chỉ có thể tạo ra 1 đối tượng quản lý tài nguyên đó. Thí dụ để quản lý việc in ấn của các phần mềm ra 1 máy in được đúng đắn và nhất quán, ta chỉ được tạo 1 đối tượng "printer spooler" để quản lý máy in tương ứng. Để quản lý việc truy xuất các file trên thiết bị chứa tin của máy, ta chỉ được tạo ra 1 đối tượng quản lý hệ thống file duy nhất. Để quản lý các cửa sổ ứng dụng, ta chỉ được tạo ra 1 đối tượng quản lý windows duy nhất…
- Để giải quyết các vấn đề trên, cách tốt nhất là dùng mẫu Singleton

7.6 Mẫu Singleton

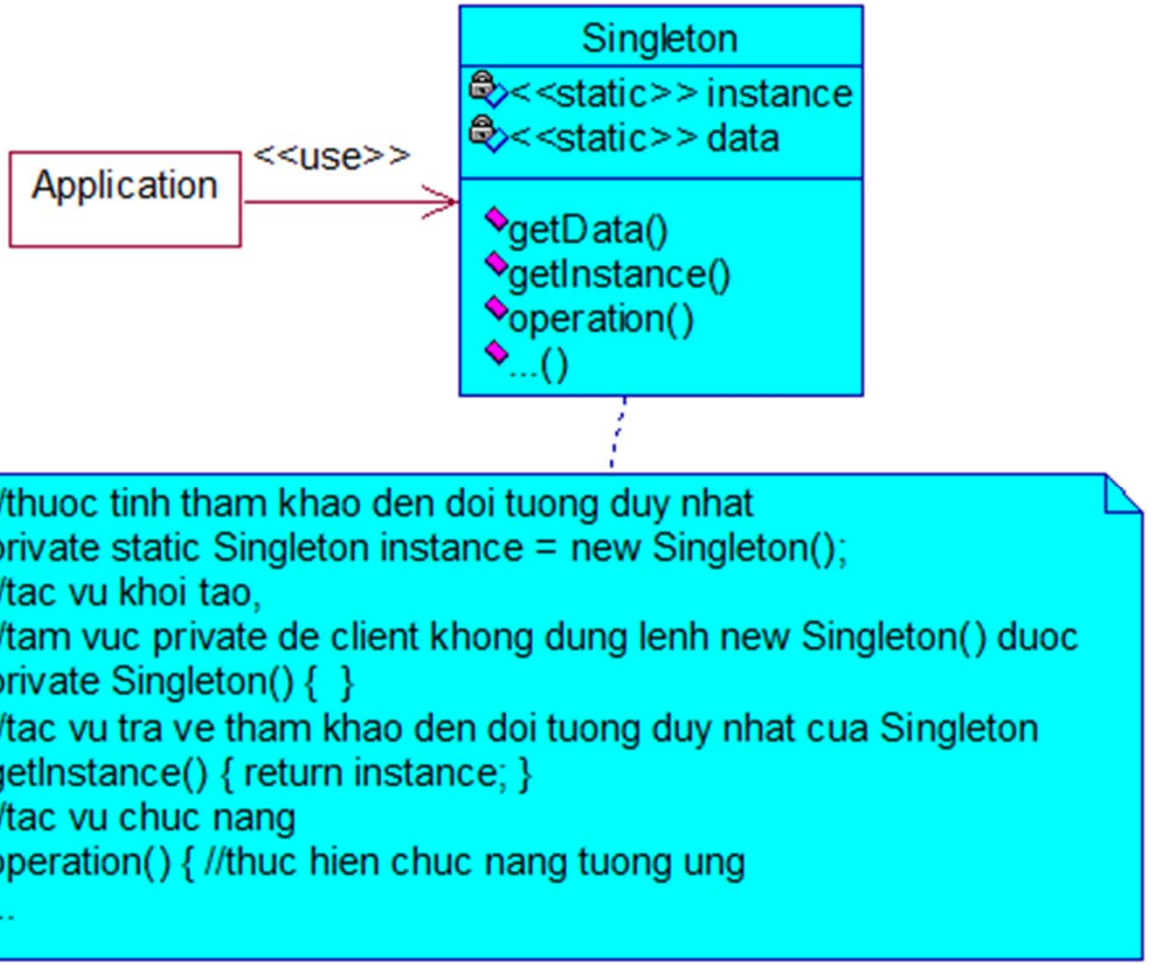
Thí dụ về việc dùng mẫu Builder :

- Sau đây là lược đồ class của mẫu Singleton giải quyết việc in ấn của các ứng dụng trên 1 máy in xác định.



7.6 Mẫu Singleton

Ta có thể xây dựng mẫu Singleton theo loại class pattern với lược đồ class như sau :



7.6 Mẫu Singleton

Các phần tử tham gia :

- Singleton (PrintSpooler) : chịu trách nhiệm về việc tạo instance duy nhất cho class Singleton, trả về tham khảo đến đối tượng duy nhất này khi có yêu cầu và không cho phép client tạo thêm đối tượng Singleton khác.
- Application : khi cần dùng đối tượng Singleton, nó gọi tác vụ getInstance() của class Singleton rồi dùng tham khảo này để truy xuất dịch vụ của đối tượng. Nó không thể dùng lệnh new để tạo đối tượng Singleton khác

7.7 Kết chương

- Chương này đã giới thiệu các thông tin cơ bản về nhóm mẫu phục vụ khởi tạo đối tượng phức hợp và thông tin chi tiết cụ thể về các mẫu Asbtract Factory, Factory Method, Prototype, Builder, Singleton.