

# DESIGN PATTERNS

## Review Lý thuyết

### 1. Tổng quát về mẫu thiết kế HDT

- ✓ Trong việc phát triển 1 phần mềm, ta thường thực hiện các hoạt động chức năng sau đây :
  1. Thu thập yêu cầu phần mềm
  2. Phân tích từng chức năng
  3. Thiết kế
  4. Hiện thực
  5. Kiểm thử
- ✓ Các hoạt động trên có mối quan hệ phụ thuộc nhau, cụ thể kết quả của bước i là dữ liệu đầu vào của bước thứ i+1. Do đó nếu bước thứ i có lỗi, nghĩa là kết quả của nó không đúng thì sẽ kéo theo các bước sau đó sẽ bị lỗi cho dù ta cố gắng thực hiện chúng tốt cách gì đi nữa.

*Tóm lại, thiết kế phần mềm là một vấn đề rất khó khăn, nhất là khi phần mềm lớn, mối quan hệ giữa các phần tử sẽ nhiều và phức tạp, bản thiết kế thường không hiệu quả và chứa nhiều lỗi khó biết. Hơn nữa, ta thường phải trả giá cao cho các lỗi thiết kế vì chúng ảnh hưởng nặng nề đến các giai đoạn sau như viết code, kiểm thử....*

- ✓ Dùng phương pháp thiết kế hướng đối tượng sẽ giúp ta có thể thiết kế được phần mềm có cấu trúc rõ ràng, mạch lạc, nhờ đó ta dễ phát hiện lỗi nếu có, dễ hiệu chỉnh, dễ nâng cấp từng thành phần (thí dụ nhờ tính bao đóng, bao gộp, thừa kế, đa xạ, tổng quát hóa...)
- ✓ Hoạt động thiết kế phần mềm là phải đạt được 3 tiêu chí chính sau:
  - *Thiết kế được phần mềm giải quyết đúng các chức năng mà user yêu cầu.*
  - *Phải hạn chế được việc tái thiết kế lại trong tương lai, cho dù vì lý do gì.*
  - *Bản thiết kế hiện hành phải hỗ trợ tốt nhất việc tái thiết kế lại nếu vì lý do gì đó phải tái thiết kế lại phần mềm.*
- ✓ Một biện pháp được đề xuất để có những bản thiết kế tốt, hạn chế được việc tái thiết kế lại và khi cần thiết kế lại, nó phải hỗ trợ tốt nhất việc tái thiết kế là sử dụng lại những mẫu thiết kế hướng đối tượng (Object Oriented Design Patterns), hay gọi tắt là mẫu thiết kế (Design Patterns).
- ✓ Vậy mẫu thiết kế là gì ?
  - *Là bản thiết kế của những người chuyên nghiệp và nổi tiếng, đã được sử dụng trong các phần mềm đang được dùng phổ biến và được người dùng đánh giá tốt.*
  - *Giúp giải quyết 1 trong những vấn đề thiết kế thường gặp trong các phần mềm.*
  - *Giúp cho bản thiết kế phần mềm có tính uyển chuyển cao, dễ hiệu chỉnh và dễ nâng cấp*
- ✓ Vai trò của mẫu thiết kế :
  - *Cung cấp phương pháp giải quyết những vấn đề thực tế thường gặp trong phần mềm mà mọi người đã đánh giá, kiểm nghiệm là rất tốt.*
  - *Là biện pháp tái sử dụng tri thức các chuyên gia phần mềm.*

- Hình thành kho tri thức, ngữ vựng trong giao tiếp giữa những người làm phần mềm.
- Giúp ta tìm hiểu để nắm vững hơn đặc điểm ngôn ngữ lập trình hướng đối tượng.

Như vậy, nếu sử dụng triệt để các mẫu thiết kế trong việc thiết kế phần mềm mới, ta sẽ tiết kiệm được chi phí, thời gian và nguồn lực. Hơn nữa phần mềm tạo được sẽ có độ tin cậy, uyển chuyển cao, dễ dàng hiệu chỉnh và nâng cấp sau này khi cần thiết.

## 2. Phân loại

By Purpose				
		Creational	Structural	Behavioral
By Scope	Class	<ul style="list-style-type: none"> <li>Factory Method</li> </ul>	<ul style="list-style-type: none"> <li>Adapter (class)</li> </ul>	<ul style="list-style-type: none"> <li>Interpreter</li> <li>Template Method</li> </ul>
	Object	<ul style="list-style-type: none"> <li>Abstract Factory</li> <li>Builder</li> <li>Prototype</li> <li>Singleton</li> </ul>	<ul style="list-style-type: none"> <li>Adapter (object)</li> <li>Bridge</li> <li>Composite</li> <li>Decorator</li> <li>Façade</li> <li>Flyweight</li> <li>Proxy</li> </ul>	<ul style="list-style-type: none"> <li>Chain of Responsibility</li> <li>Command</li> <li>Iterator</li> <li>Mediator</li> <li>Memento</li> <li>Observer</li> <li>State</li> <li>Strategy</li> <li>Visitor</li> </ul>

Mỗi mẫu thiết kế cần nắm được các thông tin sau:

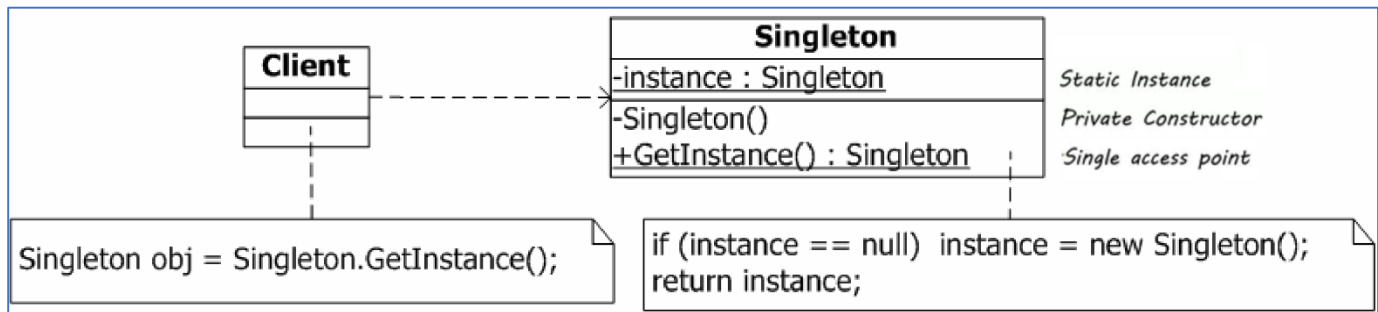
- ✓ Tên mẫu
- ✓ Mục tiêu của mẫu
- ✓ Lược đồ lớp miêu tả mẫu: trong mẫu có các phần tử nào (lớp, đối tượng) và mối quan hệ giữa chúng.
- ✓ Hiện thực

# Phần 1: CREATIONAL PATTERNS

## 1. Singleton Pattern:

**Mỗi Singleton** (thuộc nhóm *Creational Patterns*), đảm bảo một lớp chỉ duy nhất 1 đối tượng (instance) được tạo ra và cung cấp cho bên ngoài cách thức truy cập (thông qua một phương thức) đến đối tượng duy nhất đó.

**Mô hình lớp:**



**Đặc điểm Singleton Pattern:**

- ✓ private constructor để ngăn chặn truy cập từ lớp bên ngoài.
- ✓ Đặt biến private static đảm bảo đối tượng chỉ khởi tạo trong lớp.
- ✓ Có một method public static trả về đối tượng, cung cấp cho các client.

**Sử dụng Singleton Pattern:**

Singleton chỉ tồn tại 1 instance nên thường được dùng cho các trường hợp giải quyết các bài toán cần truy cập vào các ứng dụng như: Shared resource, Logger, Configuration, Caching, Thread pool.

**Hiện thực Singleton Pattern**

- Eager initialization
- Static block initialization
- Lazy Initialization
- Thread Safe Singleton
- Double Check Locking Singleton

Lazy Singleton

```
public class Singleton {

    private static volatile Singleton instance;

    private Singleton () {}

    public synchronized static Singleton getInstance() {
        if(instance == null)
            instance = new Singleton ();
        return instance;
    }
}
```

```
}  
}
```

### Eager Singleton

```
public class Singleton {  
  
    private static final Singleton instance = new Singleton ();  
  
    private Singleton () {}  
  
    public static Singleton getInstance() {  
        return instance;  
    }  
}
```

### Bài toán áp dụng:

```
import javax.persistence.EntityManager;  
import javax.persistence.Persistence;  
  
public class MyEntityManager {  
  
    public static volatile MyEntityManager instance;  
    private EntityManager entityManager;  
  
    private MyEntityManager() {  
        entityManager = Persistence.createEntityManagerFactory("JPA_Demo").createEntityManager();  
    }  
  
    public synchronized static MyEntityManager getInstance() {  
        if(instance == null)  
            instance = new MyEntityManager();  
        return instance;  
    }  
  
    public EntityManager getEntityManager() {  
        return entityManager;  
    }  
}
```

## 2. Factory Method Pattern

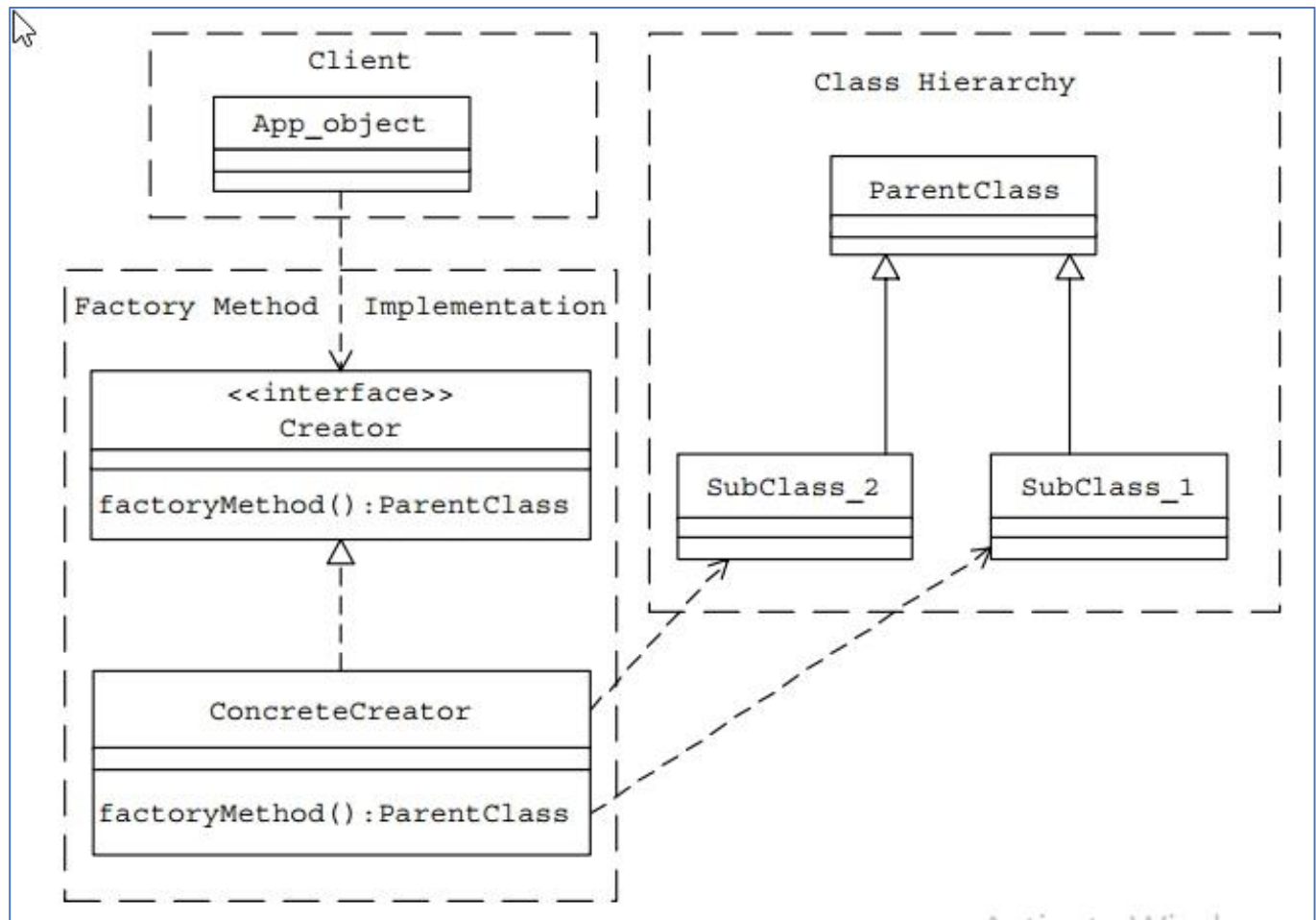
Define an interface for creating an object, but let subclasses decide which class to instantiate. The Factory method lets a class defer instantiation it uses to subclasses.

Factory Method Design Pattern hay gọi ngắn gọn là Factory Pattern (*thuộc nhóm Creational Patterns*), giải quyết vấn đề khởi tạo một loại đối tượng, mà không cần phải chỉ định rõ chúng thuộc những lớp cụ thể nào. Factory method giải quyết vấn đề này bằng cách định nghĩa một phương thức cho việc tạo đối tượng, và việc quyết định kiểu đối tượng nào được tạo ra thì phụ thuộc vào các lớp con.

### Các tình huống áp dụng

- ✓ Ta cần phải tạo ra đối tượng nhưng chưa biết đối tượng đó thuộc lớp cụ thể nào, ta chỉ biết interface để làm việc với đối tượng đó.
- ✓ Khi loại đối tượng cần tạo ra phụ thuộc vào những thông tin được cung cấp từ bên ngoài (tại thời điểm run-time), và ta phải dựa vào những thông tin này để tạo lập đối tượng xử lý cần thiết.

### Mô hình lớp tổng quát:



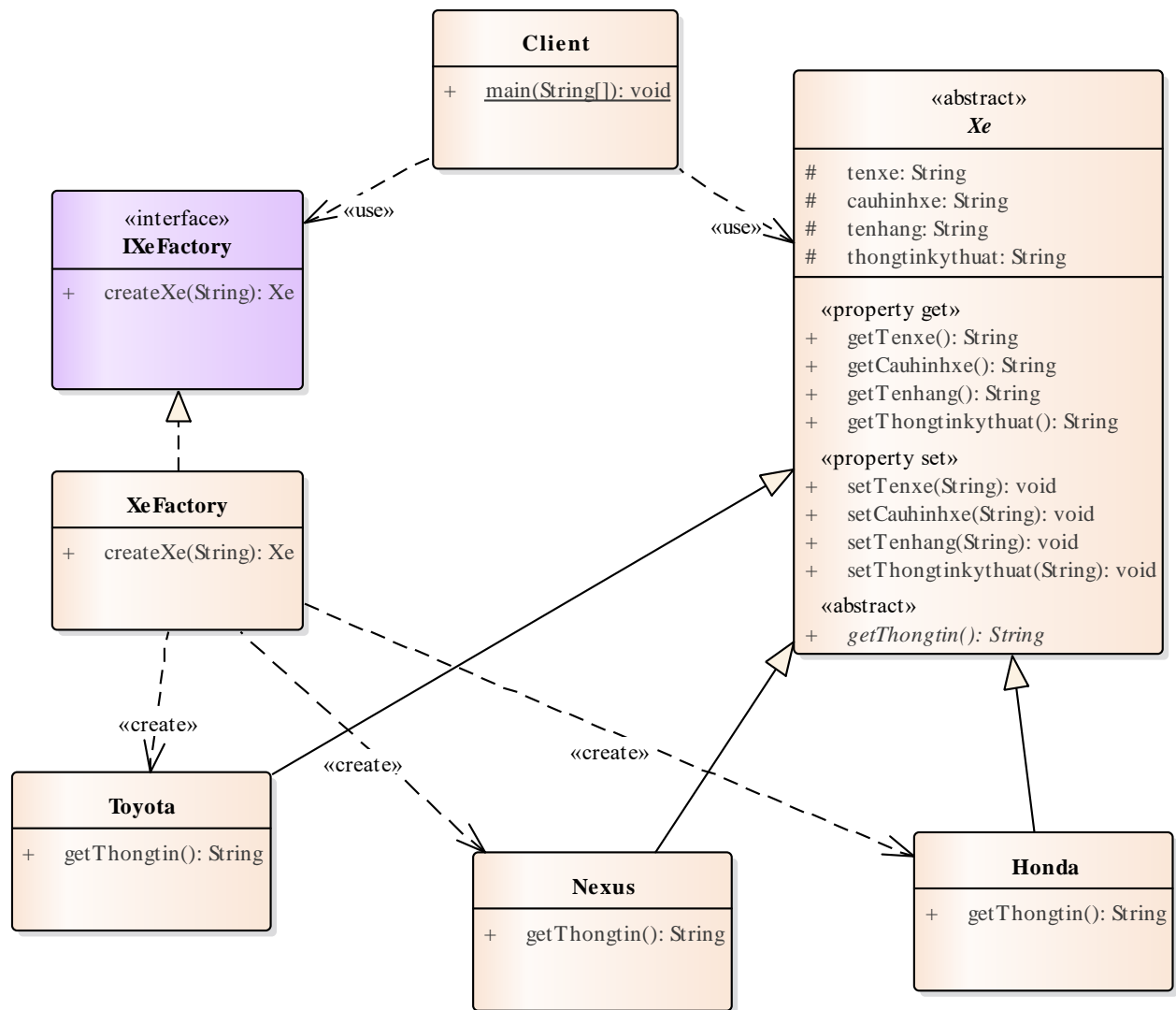
### Bài toán áp dụng:

**Bài 1: (Factory Method Pattern)** Công ty bán xe hơi hiện tại bán 3 loại xe: Honda, Nexus và Toyota. Để hỗ trợ khách hàng xem thông tin về các chiếc xe này, đội ngũ công nghệ thông tin của công ty cần xây dựng ứng dụng cho phép xem thông tin. Mỗi loại xe có các thông tin về xe, cấu hình xe, nhà sản xuất và tính năng kỹ thuật khác nhau.

Tuy nhiên công ty còn có nhu cầu bán các loại xe khác (Mercedes, Porsche, Mazda ...) trong tương lai, để giảm thiểu việc thay đổi code của ứng dụng, dùng Factory Method Pattern để thiết kế chương trình cho ứng dụng này.

- ✓ Vẽ mô hình lớp cho ứng dụng truy xuất thông tin xe hơi.
- ✓ Hiện thực ứng dụng này bằng ngôn ngữ lập trình cụ thể (Java/C#/C++).

## class Class Model



IXeFactory.java

```

package bai1_factory;

public interface IXeFactory {
    public Xe createXe(String loaixe);
}
  
```

XeFactory.java

```

package bai1_factory;

public class XeFactory implements IXeFactory{

    @Override
    public Xe createXe(String loaixe) {
        loaixe = loaixe.toLowerCase();
        switch (loaixe) {
  
```

```
        case "honda":
            return new Honda();
        case "nexus":
            return new Nexus();
        case "toyota":
            return new Toyota();
        default:
            break;
    }
    return null;
}
```

 Xe.java

```
package bai1_factory;
```

```
public abstract class Xe {
```

```
    private String tenxe;
```


```
    private String cauhinhxe;
```

```
    private String tenhang;
```

```
    private String thongtinkythuat;
```

```
    public abstract String getThongtin();
```

```
}
```

 Honda.java

```
package bai1_factory;
```

```
public class Honda extends Xe{
```


```
    @Override
```

```
    public String getThongtin() {
```

```
        return "Hon da";
```

```
    }
```

```
}
```

 Nexus.java

```
package bai1_factory;
```

```
public class Nexus extends Xe{
```

```
    @Override
```

```
    public String getThongtin() {
```

```
        return "Xe nexus";
```

```
    }
```

```
}
```

Toyota.java

```
package bai1_factory;

public class Toyota extends Xe {

    @Override
    public String getThongtin() {
        return "Toyota";
    }
}
```

Client.java

```
package bai1_factory;

public class Client {
    public static void main(String[] args) {

        IXeFactory factory = new XeFactory();
        Xe hd = factory.createXe("honda");
        System.out.println(hd.getThongtin());

        Xe t = factory.createXe("toyota");
        System.out.println(t.getThongtin());

        Xe n = factory.createXe("nexus");
        System.out.println(n.getThongtin());
    }
}
```

**Bài 2: (Factory Method Pattern)** Hệ thống có 3 kiểu Command: ListCommand, HelpCommand và RegionCommand. Dựa vào thông tin kiểu Command được nhập từ người dùng, hệ thống sẽ khởi tạo lệnh Command tương ứng và thi hành.

**Bài 3: (Factory Method Pattern)** Một khách sạn gồm có ba loại phòng là phòng hạng sang, phòng hạng vừa, phòng giá rẻ.

Khách hàng có thể đặt trước một trong 3 loại phòng bằng cách bấm số:

số 1: phòng hạng sang.

số 2 : phòng loại vừa.

số 3 : phòng giá rẻ

Khách sạn sẽ nhận thông tin, xác nhận lại và báo giá giá cả cho khách hàng.

**Bài 4: (Factory Method Pattern)** Trường học có 3 loại phòng học: phòng lý thuyết, phòng thực hành và phòng thí nghiệm. Dựa vào thông tin yêu cầu loại phòng nào, sẽ khởi tạo đối tượng phòng học tương ứng



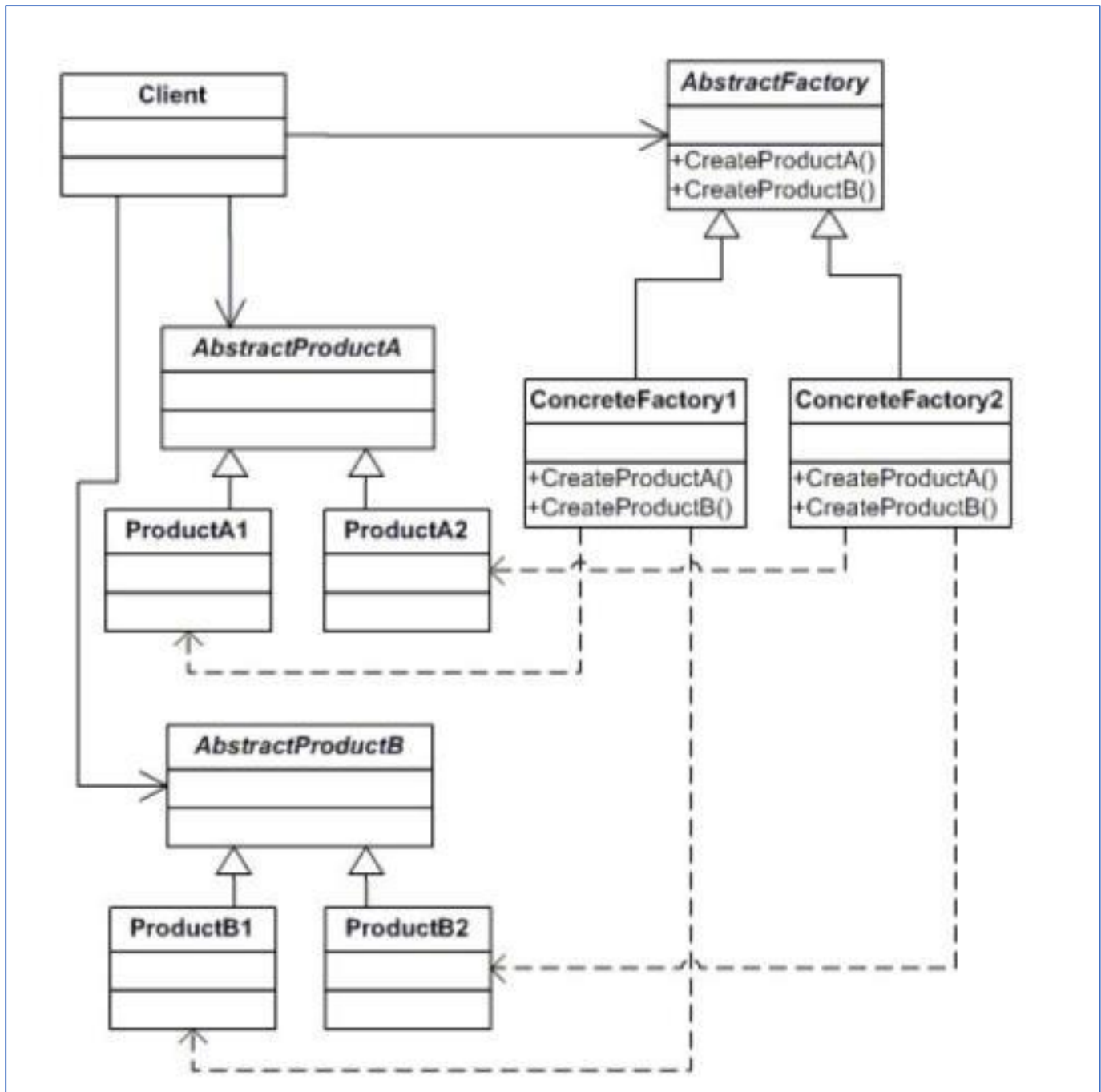
### 3. Abstract Factory Pattern

#### Mục tiêu:

Abstract Factory is a creational design pattern that provide an interface for creating families of related or dependent objects without specifying their concrete classes.

Abstract Factory pattern (*thuộc nhóm Creational pattern*). Nó là phương pháp dùng một Super-factory để tạo ra các Factory khác (*Factory của các Factory khác*). Abstract Factory Pattern là một Pattern cấp cao hơn so với Factory Method Pattern.

#### Mô hình lớp tổng quát:



## Bài toán áp dụng Abstract Factory Pattern:

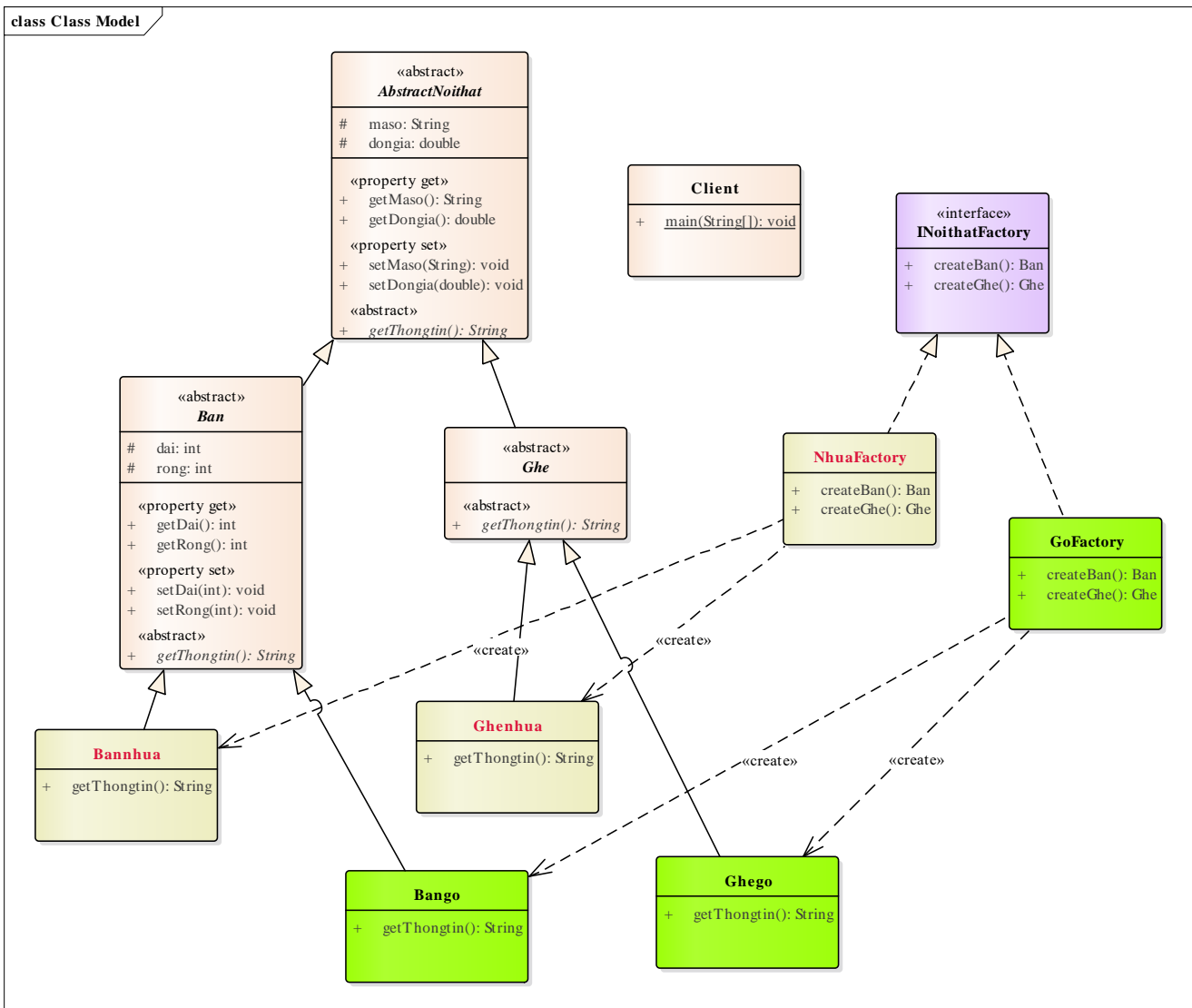
**Bài 1:** Tạo đối tượng Birds, Mammals ... trong thế giới động vật (Animal). Làm sao mở rộng ứng dụng khi bổ sung bất cứ 1 animal bất kỳ.

**Bài 2:** Một công ty đồ nội thất chuyên sản xuất ghế: ghế nhựa và ghế gỗ. Với tình hình kinh doanh ngày càng thuận lợi nên công ty quyết định mở rộng thêm sản xuất bàn. Với lợi thế là đã có kinh nghiệm từ sản xuất ghế nên công ty vẫn giữ chất liệu là nhựa và gỗ cho sản xuất bàn. Tuy nhiên, quy trình sản xuất ghế/ bàn theo từng chất liệu là khác nhau.

Nên công ty tách ra làm 2 nhà máy: 1 cho sản xuất vật liệu bằng nhựa, 1 cho sản xuất vật liệu bằng gỗ, nhưng cả 2 đều có thể sản xuất ghế và bàn. Khi khách hàng cần mua một món đồ nào, khách hàng chỉ cần đến cửa hàng để đặt mua. Khi có đơn đặt hàng của khách hàng, công ty xem xét thông tin ứng với từng hàng hóa và vật liệu sẽ được chuyển về phân xưởng tương ứng để sản xuất ra bàn và ghế.

Bàn lưu trữ thông tin gồm: Mã số, mô tả, chiều dài, chiều rộng và đơn giá

Ghế lưu trữ thông tin gồm: Mã số, mô tả và đơn giá.



**Bài 3:** Có 2 loại mặt hàng là quần áo và giày dép. Và cả 2 loại đều có 2 nhà máy sản xuất 1 là ở Anh và 1 là ở Mỹ. Khi được đem ra bán thì 2 loại mặt hàng được chia ra cho 2 đại lý phân phối chính. 1 đại lý chuyên cung cấp quần áo, 1 đại lý chuyên cung cấp giày dép.

Khi khách hàng mua hàng thì sẽ được hỏi mua sản phẩm loại nào (quần áo hoặc giày dép) sau đó sẽ được hỏi mua hàng xuất xứ của quốc gia nào (Anh hoặc Mỹ). Sau khi mua hàng khách hàng sẽ được báo lại thông tin về sản phẩm mình vừa mua.

# Phần 2: STRUCTURAL PATTERNS

## 1. Bridge Pattern

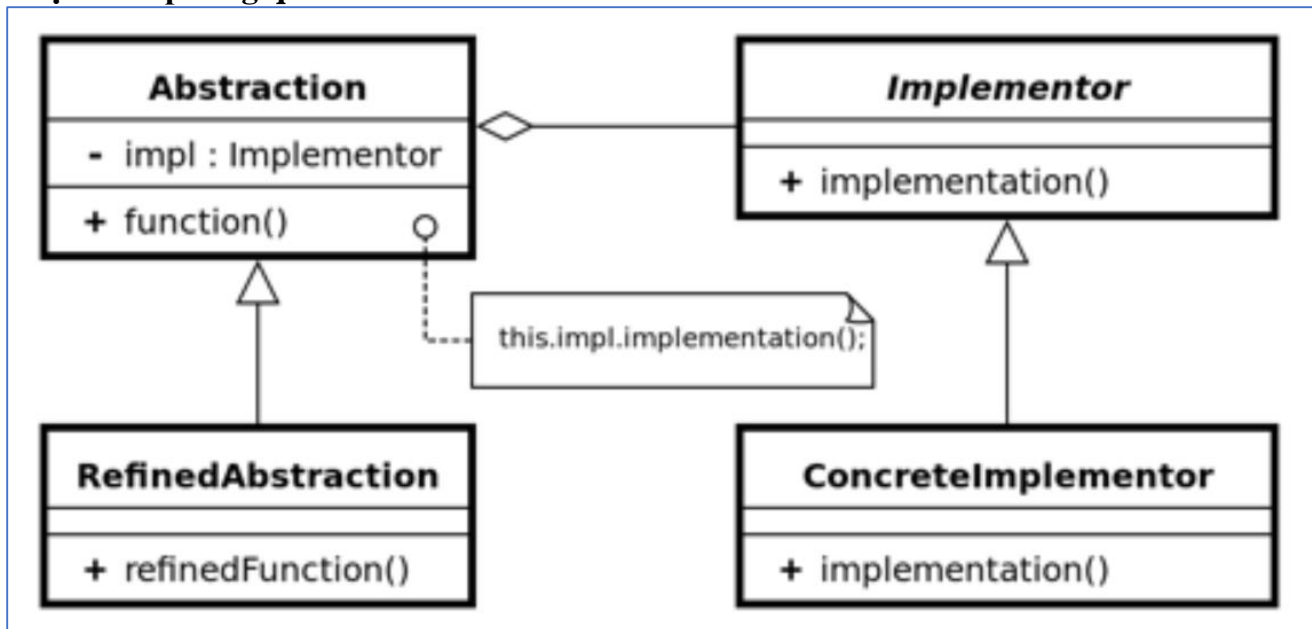
### Mục tiêu:

Decouples an abstraction from its implementation so that the two can vary independently.

Tách riêng phần trừu tượng ra khỏi phần hiện thực của nó để cả 2 có thể thay đổi độc lập.

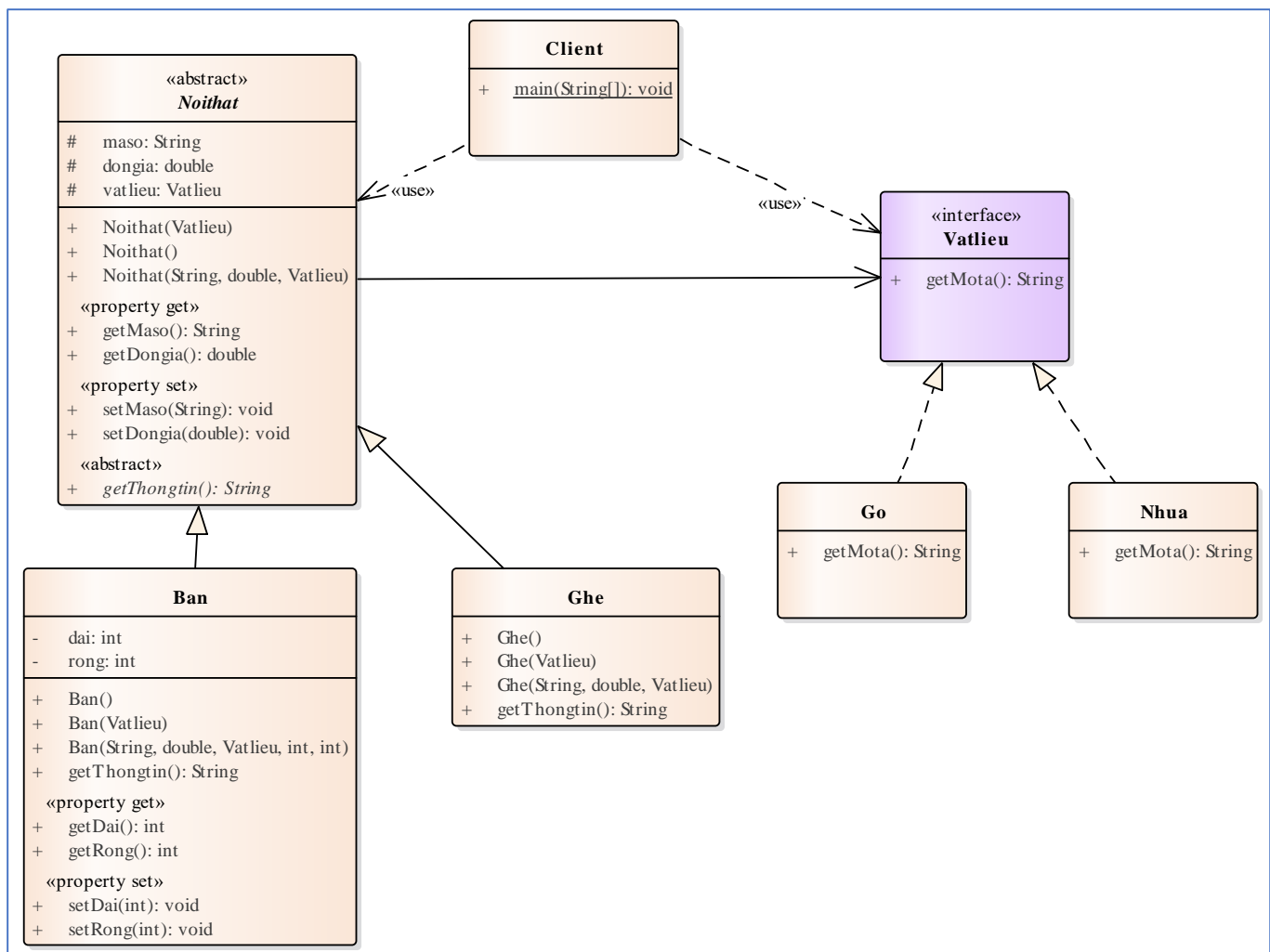
- ✓ Nếu phần trừu tượng và phần hiện thực thông qua quan hệ kế thừa (*is-a*), thì khi thay đổi trong phần trừu tượng sẽ phải thay đổi trong phần hiện thực.
- ✓ Thay vì vậy, phần trừu tượng và phần thực thi liên hệ thông qua mối quan hệ bao gộp (*has-a*). Do đó có thể thay đổi một cách độc lập mà không ảnh hưởng đến các thành phần khác.

### Lược đồ lớp tổng quát mô tả mẫu:



### Bài toán áp dụng:

Công ty chuyên sản xuất nội thất đang sản xuất 2 mặt hàng là bàn và ghế, mỗi loại dùng 2 loại vật liệu nhựa và gỗ. Tương lai có thể mở rộng mặt thêm như tủ hay kệ, hoặc mở rộng loại vật liệu là sắt hay inox...



## 2. Decorator Pattern

### Mục tiêu:

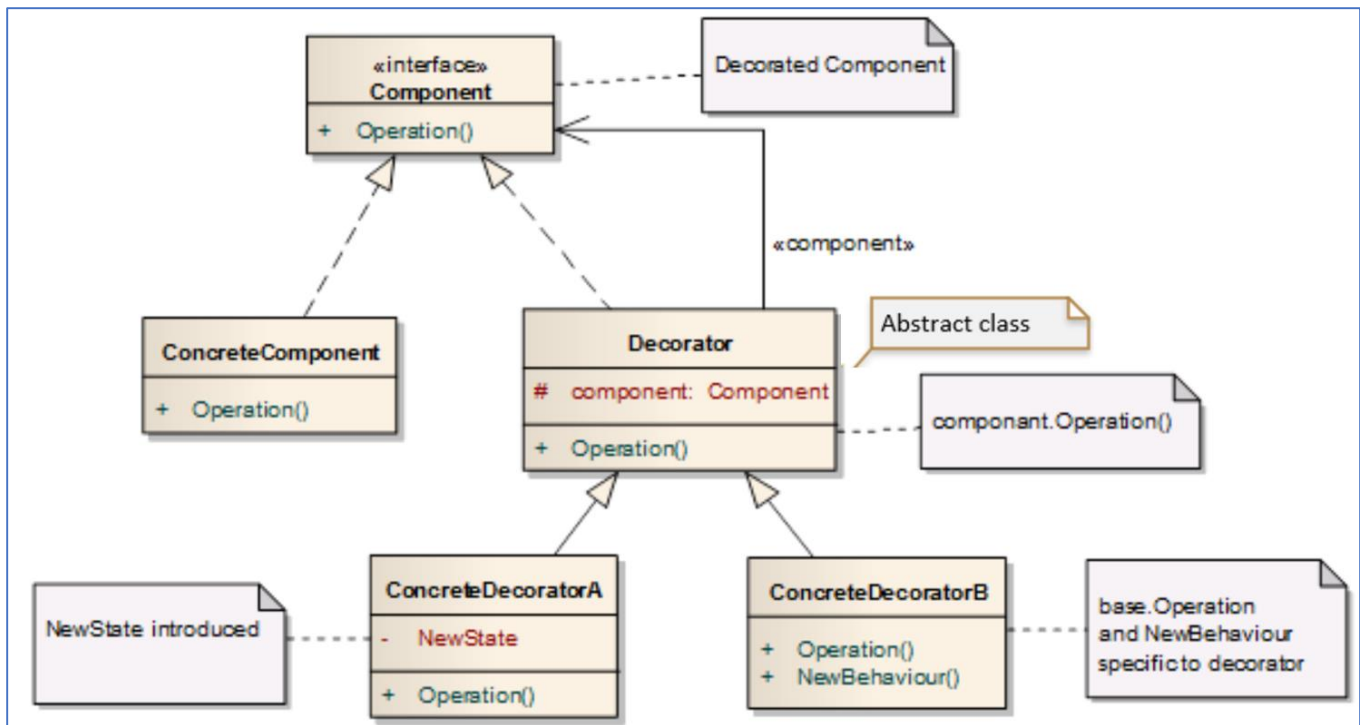
The Decorator Pattern attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality <sup>1</sup>

Thêm động một vài chức năng bổ sung cho các đối tượng. Các chức năng bổ sung được gắn thêm cho đối tượng tại thời điểm run-time so với đối tượng gốc được tạo ra từ class ban đầu, đối tượng lúc này có khả năng thực hiện được những chức năng bổ sung mà ban đầu không có.

Các tình huống áp dụng: Muốn linh động bổ sung thêm chức năng cho đối tượng và việc bổ sung này không làm ảnh hưởng đến những đối tượng cùng loại khác.

### Lược đồ lớp tổng quát mô tả mẫu:

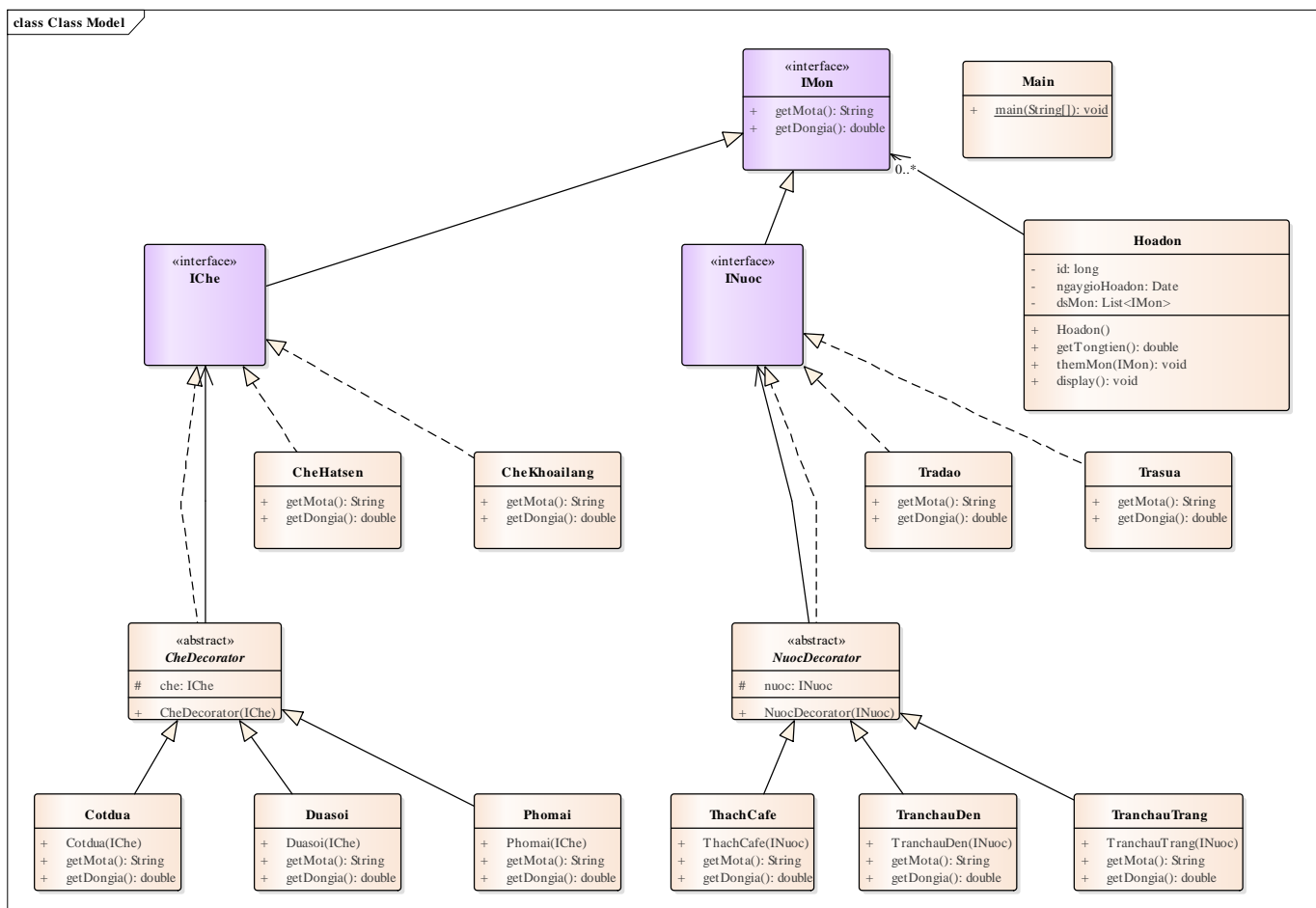
<sup>1</sup> Chapter 3: the Decorator Pattern –Head First Design Pattern



## Bài toán áp dụng:

Viết ứng dụng quản lý món trong quán nước. Món là nước uống chính và thành phần khác thêm vào nước uống.

Thiết kế ứng dụng và hiện thực ứng dụng.



**Bài toán áp dụng:**

**Bài 1:** Giảng viên trong trường đại học có nhiệm vụ chính là giảng dạy. Ngoài công nhiệm vụ chính, thì một giảng viên có thể có các nhiệm vụ khác. Ví dụ, giảng viên ngoài giảng dạy có thể là trưởng bộ môn hay trưởng khoa...

Thiết kết và hiện thực bài toán.

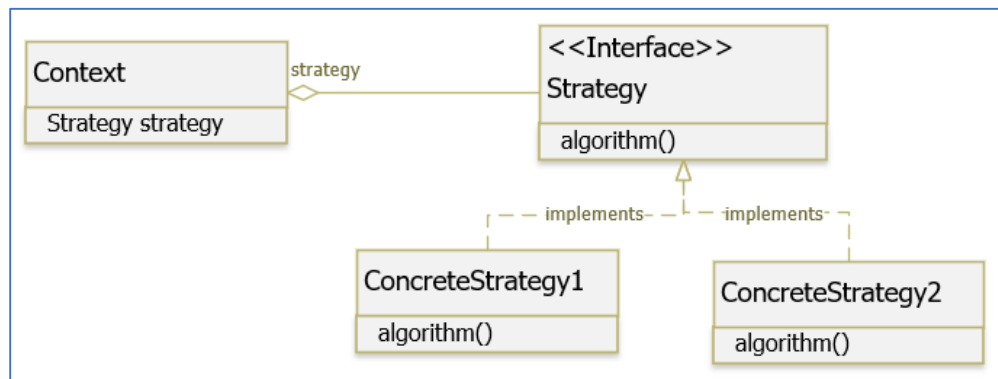
## Phần 3: BEHAVIORAL PATTERNS

### 1. Strategy Pattern

#### Mục tiêu:

- ✓ Cung cấp một họ giải thuật khác nhau để giải quyết cùng 1 vấn đề nào đó và cho phép client chọn lựa linh động để dùng một giải thuật cụ thể theo từng tình huống sử dụng.
- ✓ Strategy pattern rất hữu ích khi chúng ta có nhiều thuật toán cho tác vụ cụ thể và chúng ta muốn ứng dụng linh hoạt lựa chọn thuật toán trong thời gian chạy cho tác vụ cụ thể.

#### Class diagram



#### Bài tập áp dụng:

**Bài 1:** Tìm kiếm sản phẩm trong một tập sản phẩm, ta có nhiều giải thuật tìm kiếm khác nhau như: Binary Search, Linear Search ... Tìm giải pháp giúp các giải thuật khác nhau độc lập với client sử dụng.

**Bài 2:** Hệ thống có các thuật toán sắp xếp: QuickSort, BubbleSort, ShellSort. Client sẽ lựa chọn thuật toán sắp xếp để sắp xếp dãy các số nguyên.

#### Bài 3:

Cho một tình huống trong bài toán quản lý bán hàng như sau:

Một giỏ hàng gồm một hoặc nhiều mặt hàng. Có thể thêm mặt hàng, xóa mặt hàng trong giỏ hàng và tính tổng tiền của đơn hàng.

Thông tin mặt hàng gồm: Mã hàng, tên hàng và đơn giá.

Khi thực hiện thanh toán tiền của giỏ hàng, có thể lựa chọn một trong hai hình thức thanh toán sau: thanh toán bằng thẻ tín dụng (CreditCard) hoặc thanh toán trực tuyến (Paypal).

Thông tin CreditCard gồm: Số thẻ, tên in trên thẻ, mã bảo mật (*Card Verification Value*) và ngày hết hạn.

Thông tin Paypal gồm: Email và mật khẩu (*password*).



## 2. State Pattern

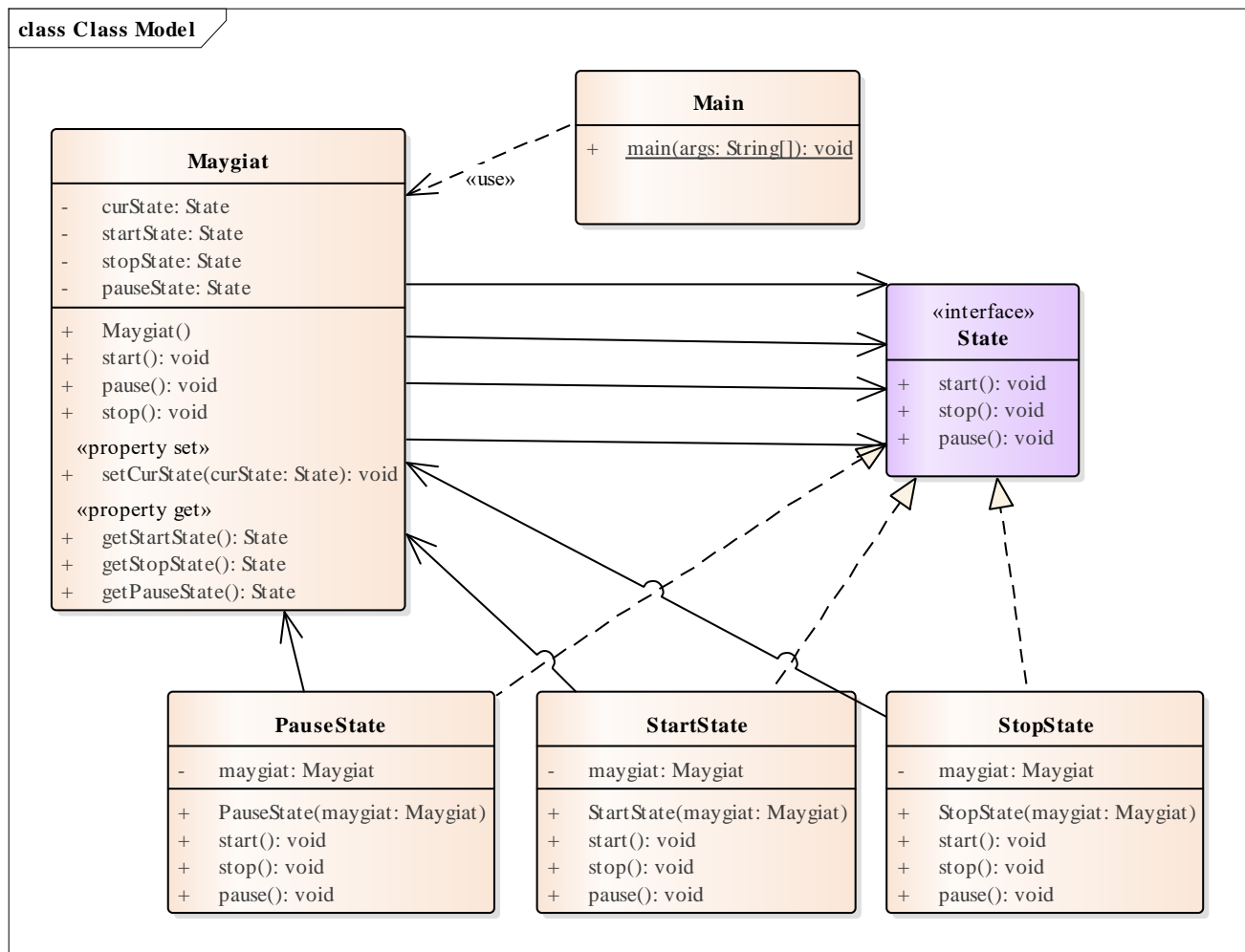
### Mục tiêu:

**State Pattern** : Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.

- ✓ Cho phép 1 đối tượng thay đổi hành vi khi trạng thái bên trong của nó thay đổi.
- ✓ Về nguyên lý chung, hành vi của đối tượng có thể phụ thuộc vào trạng thái hiện hành của đối tượng đó. Cách tốt nhất để giúp đối tượng thay đổi linh động và dễ dàng một hành vi phù hợp theo từng trạng thái là dùng mẫu State

### Bài toán áp dụng:

**Bài 1: (State Pattern)** Thiết kế cài đặt các trạng thái hoạt động của chiếc máy giặt (giả sử có 3 trạng thái *start*, *pause* và *stop*)



## Phần 4: Bài tập tổng hợp

### ĐỀ BÀI 1

Công ty ABC chuyên về lĩnh vực phần mềm máy tính.

Ngoài việc trả tiền lương theo hợp đồng cho nhân viên, hàng tháng công ty còn trả tiền thưởng cho nhân viên dựa trên kết quả làm việc hoặc các đóng góp của nhân viên cho công ty trong tháng đó.

Mỗi nhân viên cần lưu trữ các thông tin gồm: mã số, họ tên và tiền lương căn bản. Nhân viên trong công ty có thể là: lập trình viên, nhân viên kiểm thử, chuyên viên phân tích dữ liệu, nhân viên kế toán... Yêu cầu chương trình quản lý có tính độc lập cao với các đối tượng nhân viên cụ thể trong chương trình và có thể dễ dàng mở rộng thêm cho nhiều loại nhân viên nữa.

Ngoài ra, để linh hoạt trong việc áp dụng chế độ tiền thưởng, công ty đưa ra nhiều cách tính tiền thưởng khác nhau để áp dụng cho từng nhân viên trong từng tình huống cụ thể.

Hiện tại, công ty áp dụng các cách để tính tiền thưởng cho nhân viên như sau: Tiền thưởng thông thường cho nhân viên là 2% lương theo hợp đồng ban đầu. Nhân viên có làm việc ngoài giờ, tiền thưởng là 10% lương theo hợp đồng ban đầu; Nhân viên có tham gia các dự án ngoài tỉnh, tiền thưởng là 15% lương theo hợp đồng ban đầu. Tương lai, công ty sẽ bổ sung thêm các cách tính tiền thưởng khác nữa.

#### **Yêu cầu:**

1. Hãy chọn mẫu thiết kế mà bạn cho là phù hợp nhất với tình huống này. Hãy giải thích rõ ràng các mẫu đã chọn là phù hợp.
2. Thiết kế mô hình lớp (*class diagram*) tương ứng với các mẫu đã chọn.
3. Viết chương trình hiện thực mô hình lớp đã thiết kế.

### ĐỀ BÀI 2

Tình huống quản lý đơn hàng cho một công ty bán hàng qua điện thoại được mô tả như sau:

Một đơn hàng có thể đặt mua một hoặc nhiều mặt hàng với số lượng tương ứng. Một mặt hàng có thể được đặt bởi nhiều đơn hàng. Mỗi đơn hàng chỉ thuộc về một khách hàng, một khách hàng có thể đặt nhiều đơn hàng khác nhau.

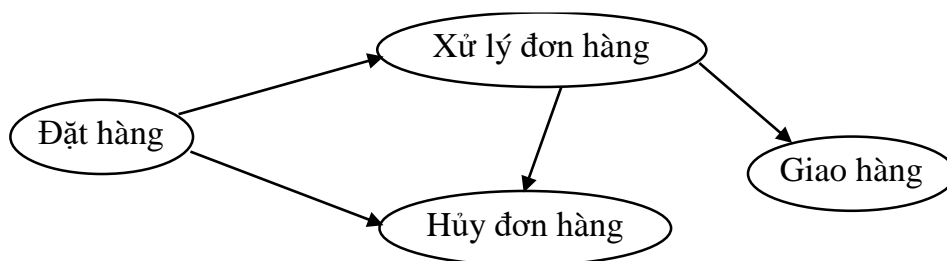
Một khách hàng cần lưu trữ thông tin tối thiểu gồm: mã khách hàng, họ tên, địa chỉ và số điện thoại liên lạc. Mỗi một mặt hàng trong hệ thống lưu trữ thông tin gồm: mã mặt hàng, tên mặt hàng và đơn giá. Đơn hàng lưu thông tin gồm: mã số đơn hàng và ngày đặt hàng.

Khi khách hàng mua hàng, nhân viên ghi nhận các mặt hàng vào đơn hàng của hệ thống. Khách hàng có thể thay đổi số lượng, mỗi lần thay đổi thông tin, đơn hàng của khách hàng đó sẽ

cập nhật lại tổng tiền. Khi khách hàng xác nhận đặt hàng, hệ thống sẽ ghi nhận thông tin đơn hàng với 1 trạng thái đơn hàng.

Chương trình cho phép khách hàng theo dõi trạng thái đơn hàng. Các trạng thái đơn hàng gồm: đặt hàng (*đơn hàng khách hàng đã đặt hàng chưa qua xử lý*), xử lý đơn hàng (*đơn hàng được nhân viên chuẩn bị các mặt hàng trong kho*), giao hàng (*đơn hàng đang được chuyển đi cho khách hàng*) và hủy đơn hàng (*nếu đơn hàng chưa giao, khách hàng có thể hủy đơn hàng bất kỳ lúc nào; hoặc hết hàng, hệ thống sẽ hủy đơn hàng*).

Sơ đồ chuyển trạng thái đơn hàng được mô phỏng như sau:



### **Yêu cầu:**

1. Hãy chọn mẫu thiết kế mà bạn cho là phù hợp nhất với tình huống này. Hãy giải thích rõ ràng mẫu đã chọn phù hợp.
2. Thiết kế mô hình lớp (*class diagram*) tương ứng với mẫu đã chọn.
3. Viết chương trình hiện thực mô hình lớp đã thiết kế.

Mỗi khi khách hàng kiểm tra trạng thái đơn hàng, cần in thông tin chi tiết của đơn hàng.