# Thực hành tuần 06

## Nội dung:

* Thiết kế đối tượng theo tư tưởng Design pattern.
* Áp dụng Pactory pattern vào demo cấu trúc đơn gian quản lý bán hàng.

## Yêu cầu thực hành:

***Xây dựng một ứng dụng vận tải(logistics).***

Hiện tại ứng dụng chỉ phục vụ việc vận chuyển bằng đường bộ(Truck class) và đường sắt (Train class). Nhưng trong tương lại công ty sẽ phát triển thêm hình thức vận chuyển đường thuỷ(Sea class) và nhiều hình thức khác.

Các bạn hay xây ứng dụng cho phép nhân viên điều phối có thể nhập thông tin hàng hóa và điểm giao hàng (trong nước/ngoài nước) vào là in ra thông tin phương tiện phù hợn đề vận chuyển (trạng thái xe là *rảnh* và *hàng Box/trong nước là* **xe tải** vận chuyển; *hàng là container/trong nước* thì do **tàu hỏa** chở; hàng là *container/ngoài nước* là **tàu thủy** chở)

***Biết các phương tiện gồm các thuộc tính:***

* MaPT:
* Loai:
* ID:
* Staus: (phương tiện đang rảnh hay đã chở hàng)

Hướng dẫn:

## Tổng quan về Design pattern:

* Design Pattern là một kỹ thuật trong lập trình hướng đối tượng.
* Nó được áp dụng như là một giải pháp cơ bản để tổ chức; cập nhật; mở rộng các ứng dụng từ mức quan niệm mô hình cho đến triển khai cài đặt code.
* Design Patterns độc lập với ngôn ngữ lập trình.
* Có nhiều pattern và các biến thể của nó. *Mỗi pattern mô tả một vấn đề xảy ra lặp đi lặp lại, và trình bày trọng tâm của giải pháp cho vấn đề đó.*
* Ưu điểm: Hệ thống phát triển một cách tường minh theo hướng module.
* Nhược điểm: Cấu trúc code phức tạp hơn:
* Phân loại các pattern:

|  |  |  |
| --- | --- | --- |
| **Các mẫu tạo lập**  **Creational patterns** | **Các mẫu cấu trúc**  **Structural patterns** | **Các mẫu ứng xử**  **Behavioral patterns** |
| 1. Abstract factory pattern 2. Builder 3. Factory method pattern 4. Prototype 5. Singleton | 1. Adapter 2. Bridge 3. Composite 4. Decorator 5. Facade 6. Flyweight 7. Proxy | 1. Chain of responsibility 2. Command 3. Interpreter 4. Iterator 5. Mediator 6. Memento 7. Observer 8. State 9. Strategy 10. Template method 11. Visitor |

## Đôi nét về Pactory pattern:

### Mục đích:

* Giúp chúng ta tạo đối tượng mà không để lộ logic tạo đối tượng ở phía người dùng và tham chiếu đến đối tượng mới được tạo ra bằng cách sử dụng một interface chung.
* Tăng tính module hóa cho hệ thống.
* Mở rộng code dễ dàng hơn (khi cần mở rộng, chỉ việc tạo ra sub class và implement thêm vào factory method)
* Quản lý vòng đời của các đối tượng thông qua Class Factory Pattern
* Thống nhất về quy tắc đặt tên(**naming convention**) trong làm việc team.

### Trường hợp dụng pattern:

* Có quá nhiều class đối tượng thuộc cùng một nhóm nhiệm vụ. Mà tại một thời điểm cụ thể ta cần trả về một class đối tượng cụ thể dựa vào input.
* Khi mở rộng hệ thống, mà không thể xác định được sẽ cần cài đặt thêm bao nhiêu lớp con.

**Ex:**

Nếu cần mua xe máy. Thì rõ ràng việc chúng ta đến từng hãng (Sym; Honda; Yamha …) để xem “xe” thì bất tiện hơn việc đến một cửa hàng kinh doanh ủy quyền bán buôn của tất cả các hãng.

**Cài đặt đối tượng:**

public interface XeMay

{

void view();

}

public class Honda: publicXeMay

{

public void view()

{

cout<<"Honda view";

}

}

public class Yamaha: publicXeMay

{

public void view()

{

cout<<"Yamaha view";

}

}

|  |  |
| --- | --- |
| **Case đến từng hãng** | **Case đến cửa hàng ủy quyền** |
|  | public class Factory  {  public void viewCar(String Type)  {  XeMay xe;  if (Type=="Honda")  {  xe = new Honda();  xe.view();  }  else if (Type=="Yamaha")  {  xe = new Yamaha();  xe.view();  }  }  } |
| public class Customer  {  public void viewHonda()  {  Honda\* honda = new Honda();  honda.view();  }  public void viewYamaha()  {  Yamaha\* yamaha = new Yamaha();  yamaha.view();  }  } | public class Customer  {  public void viewCar(string Type)  {  Factory\* factory = new Factory();  factory.viewCar(Type);  }  } |

Hàm main()???