

# Lập trình hướng đối tượng chuyên sâu

*(củng cố kiến thức lập trình hướng đối tượng và tiếp cận với một số  
khái niệm mới trong lập trình hướng đối tượng)*

# Ôn tập Lập trình hướng đối tượng

1. Class & Object
2. Property & Method
3. Contructor & Destructor
4. Extends & Override
5. Final
6. Public – Private – Protected
7. Static
8. Parent & Self
9. constant
10. clone()
11. \_\_autoload()
12. spl\_autoload\_register()
13. \_\_sleep() & \_\_wakeup()
14. toString()
15. \_\_set() & \_\_get()

# Abstract Class and Interface in PHP

## *Abstract Class*









- Abstract Class (Lớp trừu tượng) là một Class dùng để định nghĩa các thuộc tính và phương thức chung cho các class khác
- Abstract Class không cho phép khởi tạo đối tượng từ class đó.
- Abstract method không cho phép định nghĩa rõ cách hoạt động của method, chỉ dừng lại ở khai báo method
- Từ khóa abstract được dùng để định nghĩa một Lớp trừu tượng.

# Abstract Class and Interface in PHP

## *Interface*

- Interface không phải là class, đây là một mẫu giao diện quy định thêm một số thuộc tính và phương thức cho một class nào đó
- Interface không cho phép định nghĩa rõ cách hoạt động của method, chỉ dừng lại ở khai báo method
- Việc định nghĩa các method này sẽ được thực hiện ở class con

# Abstract Class and Interface in PHP

- *Abstract class là một class*   Interface không là một class
- *Abstract class mỗi method không bắt buộc phải được định nghĩa ở lớp con*    
Interface mỗi method bắt buộc phải được định nghĩa ở lớp con
- *Abstract class mỗi lớp con chỉ kế thừa từ một class abstract, một class abstract có nhiều class con*   Interface mỗi lớp con kế thừa từ một hoặc nhiều Interface, một Interface có nhiều class con
- *Abstract class mỗi phương thức abstract có thể ở trạng thái protected hoặc public*  
  Interface mỗi phương thức chỉ ở trạng thái public



# \_\_call() & \_\_callstatic()

- Khi một đối tượng gọi một phương thức, mà phương thức này chưa được định nghĩa trong class, thì phương thức \_\_call() được gọi để thay thế
- Khi một đối tượng gọi một phương thức static, mà phương thức này chưa được định nghĩa trong class, thì phương thức \_\_callstatic() được gọi để thay thế
- \_\_call() và \_\_callstatic() phải được khai báo ở trạng thái public

# Lambda Functions & Closures

- Lambda là một hàm ẩn danh (hàm không có tên), chỉ đơn giản là các hàm sử dụng một lần, có thể được định nghĩa vào bất cứ lúc nào, và thường được gắn với một biến
- Closure tương tự như Lambda, ngoài ra nó có thể truy cập các biến bên ngoài phạm vi của nó

# Lambda Functions & Closures

## *Lambda*

- Anonymous functions hàm ẩn danh (hàm không có tên)
- Trong lập trình chức năng Lambda được đánh giá khá cao, bởi vì:
  - Truyền như một tham số vào hàm khác
  - Được gán cho biến
  - Được tạo và sử dụng trong thời gian thực thi



# Lambda Functions & Closures

## *Closure*

- Closure tương tự như Lambda, ngoài ra nó có thể truy cập các biến bên ngoài phạm vi của nó

# Design Pattern

## *Thông tin chung*

- Design pattern cung cấp các "mẫu thiết kế", các giải pháp để giải quyết các vấn đề chung, thường gặp trong lập trình.
- Có khá nhiều các mẫu design pattern, mỗi loại có thể được dùng để giải quyết một vấn đề nào đó, tuy nhiên đặc điểm chung là tăng tính dễ bảo trì của phần mềm

# Design Pattern

## *Phân nhóm*

- Creational Pattern: Abstract Factory, Factory Method, Singleton, Builder, Prototype → sử dụng một số thủ thuật để khởi tạo đối tượng mà không nhìn thấy từ khóa new
- Structural Pattern: Adapter, Bridge, Composite, Decorator, Facade, Proxy và Flyweight → thiết lập, định nghĩa quan hệ giữa các đối tượng.
- Behavioral Pattern: Interpreter, Template Method, Chain of Responsibility, Command, Iterator, Mediator, Memento, Observer, State, Strategy và Visitor. → thực hiện các hành vi của đối tượng.

# Design Pattern

## *Singleton*

- **What ?** Đảm bảo chỉ duy nhất một thực thể của một class được tạo ra
- **When?** Khi cần tạo ra một class mà class đó chỉ có một đối tượng duy nhất và có thể truy cập đến đối tượng đó ở bất kỳ nơi đâu

# Design Pattern

## *Decorator*

- Decorator thuộc nhóm Structural pattern
- Decorator pattern cho phép thêm method hoặc property của 1 đối tượng trong lúc chương trình thực thi



# Design Pattern

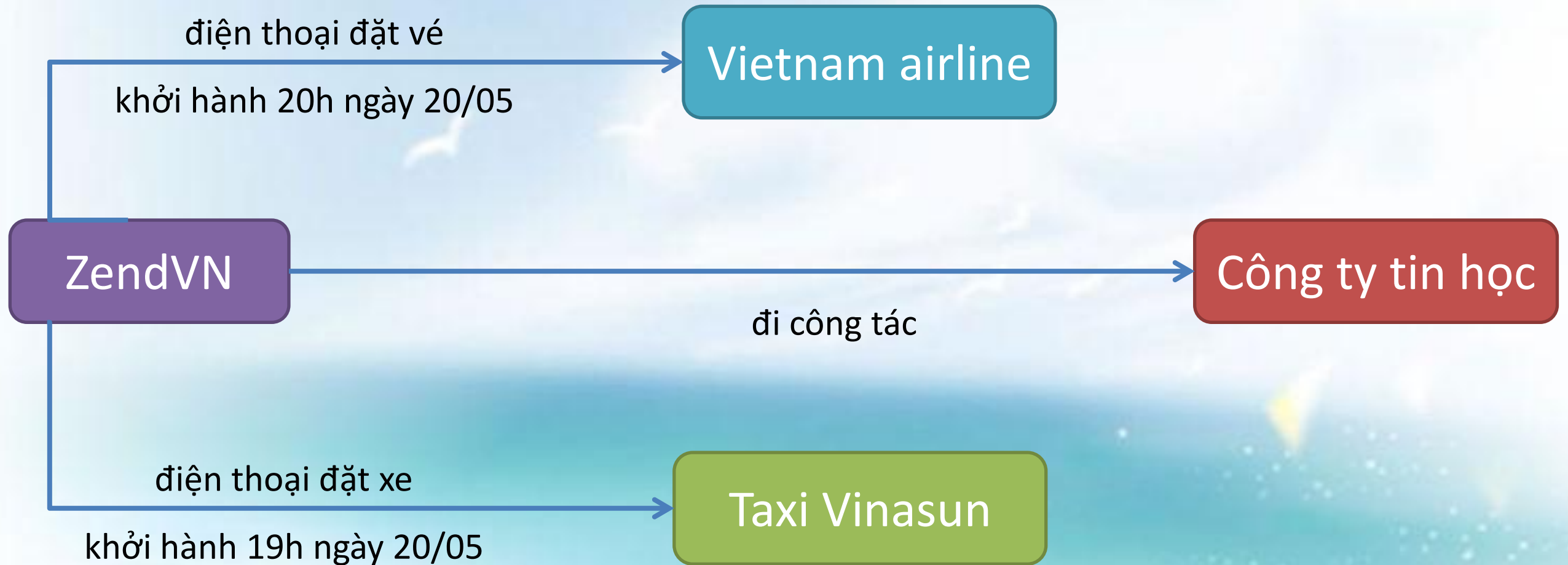
## *Factory*

- Tạo một đối tượng mà không cần thiết chỉ ra một cách chính xác lớp nào sẽ được sử dụng để tạo đối tượng đó.
- Giảm bớt sự phụ thuộc của việc tạo đối tượng với logic của ứng dụng
- Tăng tính dễ đọc và bảo trì của ứng dụng

# Dependency Injection

- Dependency Injection (DI) giúp giảm sự phụ thuộc giữa các lớp, tạo ra sự linh hoạt cao và tính dễ bảo trì cho ứng dụng
- Lợi ích khi sử dụng DI: giảm sự phụ thuộc giữa các class, tăng khả năng dùng lại và khả năng dễ đọc của mã nguồn, tạo cho ứng dụng tính mềm dẻo và khả năng nâng cấp bảo trì trở nên đơn giản và nhanh chóng hơn

# Tình huống thực tế “Đi công tác”



# Tình huống thực tế “Đi công tác”

- Thay đổi hãng Taxi
- Thay đổi hãng máy bay
- Thay đổi hình thức đặt vé

# Tình huống thực tế “Đi công tác”





# Sử dụng namespace trong PHP

- Tránh xung đột đối với tên lớp, tên hàm và tên hằng số giữa tập tin khác nhau
- Làm cho mã nguồn ngắn gọn và dễ đọc hơn

- Namespaces đơn
- Namespaces phân cấp
- Multi namespace

- Sử dụng hằng số `__NAMESPACE__`
- Truy xuất class, function, constant khi sử dụng namespace
- Sử dụng từ khoá `use` đặt alias cho namespace

# Lưu ý sử dụng namespace trong PHP

\MyProject\Database\Connection

- Unqualified Name
- Qualified Name
- Fully Qualified Name