## MeU SOLUTIONS INCORPORATED

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



**REPORT PRACTICES**

**DEAR:**

**Mentor: Ha Cong Sang**

**Intern Name: Tran Duc Thien**

**Position: Developer PHP**

**Ho Chi Minh City, August 26, 2024**

**Acknowledgments**

First of all, I would like to send my sincere thank you my mentor Ha Cong Sang for directly guiding and supporting me in studying and working with my expertise. During the process of working and practicing, I always maintain a professional, serious attitude, and do my best with the assigned exercises to supplement myself with a lot of knowledge. However, because my knowledge is still limited and my ability to absorb it is still confusing. Although I have tried my best, it is certainly difficult to avoid shortcomings and inaccuracies in the exercises, i hope that our mentor can review and give suggestions to make our essay more complete.

I sincerely thank you.

**Contents**

[MeU SOLUTIONS INCORPORATED 1](#_Toc175597284)

[PART 1. Khái niệm OOP 4](#_Toc175597285)

[1.1. Đối tượng 4](#_Toc175597286)

[1.2. Lớp 4](#_Toc175597287)

[1.3. Một số tính chất của lớp 4](#_Toc175597288)

[1.4. Phạm vi truy cập 4](#_Toc175597289)

[1.5. Phương thức khởi tạo 5](#_Toc175597290)

[1.6. Thành phần tĩnh static 5](#_Toc175597291)

[1.7. Nạp chồng phương thức 5](#_Toc175597292)

[1.8. Generic 5](#_Toc175597293)

[1.9. Kế thừa 6](#_Toc175597294)

[1.10. Lớp trừu tượng 6](#_Toc175597295)

[1.11. Interface 6](#_Toc175597296)

[PART 2. Restful API 6](#_Toc175597297)

[2.1. Định nghĩa 6](#_Toc175597298)

[2.2. Đặc điểm 7](#_Toc175597299)

[PART 3. Sự khác nhau giữa PUT và PATCH 7](#_Toc175597300)

[PART 4. SOLID principle 8](#_Toc175597301)

[4.1. Khái niệm 8](#_Toc175597302)

[PART 5. HTTP Status 10](#_Toc175597303)

[5.1. Khái niệm 10](#_Toc175597304)

[5.2. Ý nghĩa của các HTTP Status 10](#_Toc175597305)

[PART 6. Design Pattern 10](#_Toc175597306)

[6.1. Khái niệm 10](#_Toc175597307)

[6.2. Đặc điểm 10](#_Toc175597308)

[6.3. Phân loại Design pattern 11](#_Toc175597309)

[PART 7. Middleware 12](#_Toc175597310)

[7.1. Khái niệm 12](#_Toc175597311)

# Khái niệm OOP

## Đối tượng

* Đối tượng là duy nhất. Không có hai đối tượng giống nhau dù cùng chia sẻ tính chất, trạng thái. Ví dụ người là một đối tượng, xe là một đối tượng.
* Mô hình hóa vào trong lập trình: Đóng gói thành trạng thái (state) và hành vi (behavior).
* Trạng thái được biểu diễn bởi các thuộc tính (attribute) và các mối quan hệ (relationship).
* Hành vi được biểu diễn bởi các thao tác (operations) hay phương thức (methods).

## Lớp

* Các đối tượng có đặc tính tương tự nhau được gom chung lại thành lớp đối tượng.
* Ví dụ: Người là lớp đối tượng.
* Lớp định nghĩa các thuộc tính và phương thức chung cho tất cả các đối tượng của cùng một loại nào đó.
* Một đối tượng là một thể hiện cụ thể của một lớp. Ví dụ: Mỗi đối tượng xe đạp là một thể hiện của lớp XeDap

## Một số tính chất của lớp

* Tính trừu tượng (abstraction): là cách nhìn đơn giản hóa về một đối tượng trong đó chỉ bao gồm những đặc điểm được quan tâm và bỏ qua những chi tiết không cần thiết.
* Tính đóng gói (encapsulation): Là việc che giấu việc thực thi một đối tượng.
* Tính kế thừa (inheritance): các lớp khác nhau nhưng có thể có chung một số đặc điểm. Có tể đặc tả lớp đối tượng mới từ những lớp đối tượng đã có. Khi đó lớp mới mang đặc tính của lớp đã có và đặc tính riêng của nó.
* Tính đa hình (polymorphism): Là khả năng thực hiện khác nhau cho cùng một hành vi. Là kỹ thuật cho phép thay đổi nội dung của cùng một hành vi trong 2 lớp cha con

## Phạm vi truy cập

* Public (+): công cộng, có thể truy xuất bất kỳ đâu bên trong và bên ngoài lớp
* Protected (#): chỉ được truy cập ở bên trong lớp và lớp con.
* Private (-): chỉ được truy cập bên trong lớp.

## Phương thức khởi tạo

* Dùng để khởi tạo giá trị ban đầu cho các thành phần dữ liệu của đối tượng.
* Có 3 loại phương thức khởi tạo
* Phương thức khởi tạo mặc định: là phương thức không có tham số đầu vào. Các thông tin ban đầu cho đối tượng của lớp bằng những giá trị mặc định hoặc do lập trình viên quy định
* Phương thức khởi tạo sao chép: là phương thức nhận tham số đầu vào là 1 đối tượng thuộc cùng 1 lớp
* Phương thức khởi tạo có tham số: là phương thức khởi tạo không phuộc vào 2 loại trên. Các thông tin ban đầu của đối tượng sẽ phụ thuộc vào giá trị các tham số của phương thức khởi tạo

## Thành phần tĩnh static

* Được khởi tạo một lần duy nhất ngay khi biên dịch chương trình
* Dùng chung cho mọi đối tượng.
* Được gọi thông qua tên lớp
* Được hủy khi kết thúc chương trình
* **Khuyết điểm: khai báo nhiều static khi biên dịch chương trình sẽ gây tiêu hao bộ nhớ, chương trình sẽ bị chậm.**

## Nạp chồng phương thức

* Khái niệm: là hiện tượng hai hay nhiều phương thức trùng tên cùng tồn tại trong một lớp.
* Đặc điểm:
  + - * Trùng tên phương thức
      * Thuộc cùng một lớp
      * Khác nhau: số lượng tham số, thứ tự các tham số, kiểu dữ liệu của tham số.

## Generic

* Định nghĩa một phương thức, một lớp mà không cần chỉ ra cụ thể kiểu dữ liệu là gì.
* Tùy vào kiểu dữ liệu mà người dung đó truyền vào, lớp sẽ hoạt động theo kiểu đó.

## Kế thừa

* Khái niệm: kế thừa là xây dựng lớp mới trên cơ sở lớp cũ đã tồn tại. Giải quyết các vấn đề dư thừa thông tin.
* Tái sử dụng code: những phần code chung chỉ cần định nghĩa một lần tại lớp cha, các lớp con đều có thể sử dụng mà không cần viết lại.
* Thuận tiện trong việc bảo trì nâng cấp và phát triển. Khi sửa lỗi hay nâng cấp chỉ cần định nghĩa lại ở lớp cha.

## Lớp trừu tượng

* Lớp trừu tượng thực chất là một lớp cơ sở mà các lớp khác có cùng bản chất (kiểu, loại, nhiệm vụ) có thể dẫn xuất từ nó.
* Lớp trừu tượng là một lớp chưa hoàn chỉnh
* Từ khóa abstract được dùng để khai báo lớp trừu tượng

## Interface

* Một lớp chỉ có thể kế thừa từ một superclass trực tiếp (đơn kế thừa). Để hỗ trợ đa kế thừa trong lập trình hướng đối tượng ta sử dụng interface
* Interface như là một mặt nạ, một bản thiết kế của một lớp, có cùng cách thức hoạt động nhưng khác bản chất bên trong
* Interface không có constructor, cũng không có destructor
* Không thể khai báo phạm vi truy cập cho các thành phần bên trong interface. Các thành phần này tự mặc định là public
* Interface cũng biểu diễn mối quan hệ thừa kế và không thể khởi tạo đối tượng.
* Các lớp có thể thực thi nhiều interface một lúc.

# Restful API

## Định nghĩa

* RESTful API là một cách thiết kế và phát triển các dịch vụ web dự trên kiến trúc REST (Representational State Transfer). RESTful API cho phép các ứng dụng giao tiếp với nhau qua HTTP bằng cách sử dụng các phương thức HTTP tiểu chuẩn như GET, POST, PUT, DELETE,…
* API là một tập hợp các quy tắc và cơ chế mà theo đó, một ứng dụng hay một thành phần sẽ tương tác với một ứng dụng hay thành phần khác. API có thể trả về dữ liệu mà bạn cần cho ứng dụng của mình ở những kiểu dữ liệu phổ biến như JSON hay XML.
* **REST** (**Re**presentational **S**tate **T**ransfer) là một dạng chuyển đổi cấu trúc dữ liệu, một kiểu kiến trúc để viết API. Nó sử dụng phương thức HTTP đơn giản để tạo cho giao tiếp giữa các máy. Vì vậy, thay vì sử dụng một URL cho việc xử lý một số thông tin người dùng, REST gửi một yêu cầu HTTP như GET, POST, DELETE, vv đến một URL để xử lý dữ liệu

## Đặc điểm

• Client-Server: Kiến trúc REST tuân thủ mô hình client-server, trong đó client gửi yêu cầu tới server, và server xử lý yêu cầu và trả về kết quả. Điều này giúp phân tách giao diện người dùng khỏi việc quản lý dữ liệu.

• Stateless: Mỗi yêu cầu từ client đến server phải chứa đầy đủ thông tin cần thiết để server có thể hiểu và xử lý yêu cầu. Server không lưu trữ trạng thái của client giữa các yêu cầu.

• Cacheable: Dữ liệu trong RESTful API có thể được đánh dấu là cacheable hoặc không. Nếu có thể cache, điều này giúp giảm tải cho server khi cùng một yêu cầu được gửi nhiều lần.

• Giao diện đồng nhất (Uniform Interface): RESTful API sử dụng các phương thức HTTP tiêu chuẩn và URL để định danh tài nguyên. Mỗi URL tương ứng với một tài nguyên cụ thể, và các phương thức HTTP sẽ xác định hành động được thực hiện trên tài nguyên đó (ví dụ: GET để lấy dữ liệu, POST để tạo dữ liệu mới, PUT để cập nhật dữ liệu, DELETE để xóa dữ liệu).

# Sự khác nhau giữa PUT và PATCH

|  |  |
| --- | --- |
| PUT | PATCH |
| * PUT được sử dụng để update thay thế hoàn một tài nguyên hiện có với dữ liệu mới hoặc tạo mới tài nguyên nếu nó tồn tại. | * PATCH được sử dụng để cập nhật một phần tài nguyên hiện có, các thuộc tính không được đề cập sẽ giữ nguyên giá trị hiện tại |
| Ví dụ:   * Ta có dữ liệu sau   {  “Name”: “Thiện”,  “Age”: “20”,  “Sport”: “Football”  }  Ta dung lệnh PUT  {  “Sport”: “Football, Running”  }  Ta sẽ có response là:  {  “Sport”: “Football, Running”  }  Như vậy dữ liệu cũ đã được thay thế hoàn toàn. | Ví dụ:   * Ta có dữ liệu sau   {  “Name”: “Thiện”,  “Age”: “20”,  “Sport”: “Football”  }  Ta dung lệnh PATCH  {  “Sport”: “Football, Running”  }  Ta sẽ có response là:  {  “Name”: “Thiện”,  “Age”: “20”,  “Sport”: “Football, Running”  }  Như vậy dữ liệu chỉ cập nhật những field được yêu cầu thay vì cập nhật toàn bộ. |

# SOLID principle

## Khái niệm

* SOLID là một tập hợp các nguyên tắc thiết kế phần mềm hướng đối tượng. Những nguyên tắc này giúp các nhà phát triển tạo ra phần mềm có cấu trúc tốt, dễ bảo trì, dễ mở rộng, và tránh được nhiều lỗi phổ biến trong lập trình hướng đối tượng. SOLID là viết tắt của 5 nguyên tắc sau:

**1**. S - Single Responsibility Principle (Nguyên tắc trách nhiệm đơn lẻ):

Định nghĩa: Một lớp chỉ nên có một lý do duy nhất để thay đổi, nghĩa là nó chỉ nên có một trách nhiệm duy nhất.

Ý nghĩa: Khi một lớp có nhiều trách nhiệm, những thay đổi trong một trách nhiệm có thể ảnh hưởng đến các trách nhiệm khác, dẫn đến việc mã nguồn khó bảo trì và dễ phát sinh lỗi.

**2**. O - Open/Closed Principle (Nguyên tắc mở/đóng):

Định nghĩa: Các thực thể phần mềm (lớp, module, hàm, v.v.) nên được mở để mở rộng nhưng đóng cho việc sửa đổi.

Ý nghĩa: Bạn nên có khả năng mở rộng chức năng của một lớp mà không cần thay đổi mã nguồn hiện có. Điều này thường được thực hiện thông qua việc sử dụng kế thừa và đa hình.

**3**. L - Liskov Substitution Principle (Nguyên tắc thay thế của Liskov):

Định nghĩa: Các đối tượng của một lớp con nên có thể thay thế các đối tượng của lớp cha mà không làm thay đổi tính đúng đắn của chương trình.

Ý nghĩa: Khi kế thừa từ một lớp cha, lớp con nên duy trì hành vi của lớp cha, tức là không nên làm thay đổi cách thức hoạt động của các phương thức được kế thừa.

**4**. I - Interface Segregation Principle (Nguyên tắc phân tách Interface):

Định nghĩa: Khách hàng không nên bị buộc phải phụ thuộc vào các interface mà họ không sử dụng.

Ý nghĩa: Thay vì tạo ra một interface lớn chứa nhiều phương thức không liên quan, hãy tách chúng ra thành nhiều interface nhỏ hơn để các lớp triển khai chỉ cần thực hiện những gì cần thiết cho chúng.

**5**. D - Dependency Inversion Principle (Nguyên tắc đảo ngược phụ thuộc):

Định nghĩa: Các module cấp cao không nên phụ thuộc vào các module cấp thấp. Cả hai nên phụ thuộc vào các abstraction. Thêm vào đó, các abstraction không nên phụ thuộc vào chi tiết cụ thể, mà ngược lại, chi tiết cụ thể nên phụ thuộc vào abstraction.

Ý nghĩa: Nguyên tắc này khuyến khích việc sử dụng các interface hoặc abstract class để giảm sự phụ thuộc giữa các module, từ đó làm cho hệ thống linh hoạt và dễ dàng mở rộng hơn.

# HTTP Status

## Khái niệm

* HTTP Status code là một mã gồm 3 chữ số được gửi trả về cho client từ server dung để mô tả trạng thái của quá trình server xử lý một yêu cầu (request) cho trước gửi từ client tới server dưới giao thức HTTP.

## Ý nghĩa của các HTTP Status

* 1xx: Các status code loại này dung để đơn giản thông báo với client rằng server đã nhận được request. Các status 1xx ít được sử dụng và thậm chí không được định nghĩa trong HTTP phiên bản 1.0
* 2xx: Các status code loại này có ý nghĩa rằng request được server nhận, hiểu và xử lý thành công
* 3xx: Các status code loại này ý nghĩa rằng server sẽ chuyển tiếp request hiện tại sang một request khác và client cần thực hiện việc gửi request tiếp theo đó để có thể hoàn tất. Thông thường khi trình duyệt nhận được status code loại này sẽ tự động thực hiện việc gửi request tiếp theo để lấy về kết quả.
* 4xx: Các status code loại này có ý nghĩa rằng đã có lỗi từ phía client trong khi gửi request. Ví dụ sai URL, sai HTTP method, không có quyền truy cập vào trang,…
* 5xx: Các status code loại này có ý nghĩa rằng đã có lỗi từ phía server trong xử lý request. Ví dụ database lỗi hoặc server bị hết bộ nhớ.

# Design Pattern

## Khái niệm

* Design patterns là các giải pháp tổng quát, lặp lại được áp dụng cho các vấn đề thường gặp trong quá trình thiết kế phần mềm. Các mẫu thiết kế này không phải là mã cụ thể mà là các mô tả hoặc mẫu về cách giải quyết vấn đề trong các ngữ cảnh khác nhau. Chúng giúp lập trình viên thiết kế phần mềm hiệu quả, dễ bảo trì, mở rộng, và có tính tái sử dụng cao.

## Đặc điểm

* Tính tái sử dụng: Design pattern cung cấp các giải pháp có thể được áp dụng lại trong nhiều tình huống khác nhau, giúp giảm tời gian phát triển và tối ưu hóa mã nguồn.
* Tính phổ biến: design pattern được sử dụng rộng rãi bởi các lập trình viên làm cho mã nguồn trở nên dễ hiểu hơn khi làm việc nhóm.
* Tính linh hoạt: Design pattern không ép buộc người dung vào một cách duy nhất để giải quyết vấn đề mà cung cấp các nguyên tắc có thể được tùy chỉnh cho từng trường hợp cụ thể.

## Phân loại Design pattern

1. Creational Design Pattern (nhóm khởi tạo): nhóm này sẽ quản lý và kiểm soát việc tạo đối tượng.

* Abstract Factory: cung cấp một interface cho việc tạo lập các đối tượng có quan hệ với nhau mà không cần quy định lớp hay xác định lớp cụ thể.
* Factory Method: định nghĩa interface để sinh ra đối tượng nhưng để cho lớp con quyết định dùng lớp nào được sinh ra đối tượng Factory method cho phép một lớp chuyển quá trình khởi tạo đối tượng cho lớp con
* Sigleton: đảm bảo 1 class chỉ có 1 instance và cung cấp 1 điểm truy xuất toàn cục đến nó.

1. Structural Pattern (Mẫu cấu trúc): các mẫu liên quan đến cấu trúc, thiết kế và tổ chức các đối tượng và lớp để tạo nên cấu trúc lớn hơn.

* Adapter/Wrapper: do vấn đề tương thích, thay đổi interface của một lớp thành một interface khác phù hợp với yêu cầu người sử dụng lớp.
* Composite: Tổ chức các đối tượng theo cấu trúc phân cấp dạng cây; Tất cả các đối tượng trong cấu trúc được thao tác theo một cách thuần nhất như nhau.Tạo quan hệ thứ bậc bao gộp giữa các đối tượng. Client có thể xem đối tượng bao gộp và bịbao gộp như nhau -> khả năng tổng quát hoá trong code của client -> dễ phát triển, nâng cấp, bảo trì
* Decorator: Gán thêm trách nhiệm cho đối tượng, mở rộng chức năng khi chạy
* Proxy: Cung cấp đối tượng đại diện cho một đối tượng khác để hỗ trợ, kiểm soát quá trình truy xuất đối tượng đó. Đối tượng thay thế gọi là proxy.

1. Behavior Pattern (Mẫu hành vi): quản lý các vấn đề, hành vi của đối tượng

* Observer: Định nghĩa mối quan hệ một-nhiều giữa các đối tượng để khi một đối tượng thay đổi trạng thái, tất cả các phụ thuộc của nó đều được thông báo và cập nhật tự động.
* Strategy: Định nghĩa một họ thuật toán, đóng gói từng thuật toán và làm cho chúng có thể thay thế nhau. Strategy cho phép thay đổi thuật toán độc lập với các client sử dụng nó

1. Singleton Design Pattern: singleton design pattern thuộc về Creational design pattern. Đây là một design pattern có tần suất sử dụng nhiều trong việc lập trình. Singleton design pattern đảm bảo rằng một lớp chỉ có một sự xuất hiện (instance) và cung cấp một cổng giao tiếp chung nhất để truy cập vào lớp đó.

# Middleware

## Khái niệm

* Middleware là những đoạn mã trung gian giữa các request và response. Nó nhận các yêu cầu (request) và thực thi các mệnh lệnh trên request đó. Sau khi hoàn thành nó response hoặc chuyển kết quả ủy thác cho một middleware khác trong hàng đợi.
* Một số chức năng phổ biến của middleware:
* Xác thực và phân quyền
* Yêu cầu đăng nhập
* Chuyển hướng người dung
* Thay đổi/chuẩn hóa các tham số