# Object Oriented Programing

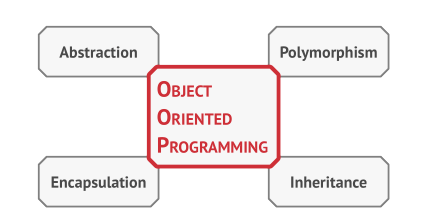
## CLASS

|  |  |
| --- | --- |
|  | **UML Summary**  + Class ( class of instances )  + Fields ( State )  + pulic  + private  + Methods ( Behavior )  Mỗi một lớp sẽ có |
|  | Mỗi một instance của class sẽ chứa các thuộc tính **standard attributes**: name, sex, age, weight, color, favorite food, etc. được định nghĩa ở trong **CLASS CAT ( CAT FIELDS )**  Bên cạnh đó mỗi instance đều có cùng hành vi được định nghĩa ở **CLASS CAT (Methods)** |

## Class hierarchies ( phân lớp )

|  |  |
| --- | --- |
|  | **UML Summary**  + Super Class  + Fields ( State )  + pulic  + private  + Methods ( Behavior )  SuperClass là lớp cha có những thành phần chung nhất để cho SubClass (lớp con) kế thừa. Trong UML SubClass có mũi tên hình tam giác trống trỏ đến SuperClass ngầm hiểu là kế thừa từ SuperClass. Cũng như vậy mà nét đứt thì sẽ chỉ ra rằng subclass có thể overelap |
|  | Subclasses can override the behavior of methods that they inherit from parent classes. A subclass can either completely replace the default behavior or just enhance it with some extra stuff.  Các lớp con có thể ghi đè hành vi của các phương thức mà chúng kế thừa từ các lớp cha. Một lớp con có thể thay thế hoàn toàn hành vi mặc định hoặc chỉ nâng cao nó bằng một số công cụ bổ sung. |

## Pillars of OOP - Main Concept



|  |  |
| --- | --- |
| **Abstraction ( ap ‘trak sn ) – Tính trừ tượng**  Lớp máy bay có thể tồn tại ở 2 trường hợp sau đây:   * Mô phỏng chuyển bay   Tốc độ  Độ cao  Góc cuộn  Góc sân  Góc nghiên   * Ứng dụng đặt vé máy bay   Chỗ ngồi  Chỗ ngồi chống  Trừu tượng là một mô hình của một đối tượng hoặc hiện tượng trong thế giới thực, giới hạn trong một bối cảnh cụ thể, thể hiện tất cả các chi tiết liên quan đến bối cảnh này với độ chính xác cao và bỏ qua tất cả các phần còn lại. | **Abstraction**    Abstraction is a model of a real-world object or phenomenon, limited to a specific context, which represents all details relevant to this context with high accuracy and omits all the rest.. |
| **Encapsulation (in kap su ‘lei sn) Đóng gói**  Tính đóng gói là khả năng của một đối tượng che giấu các phần trạng thái và hành vi của nó khỏi các đối tượng khác, chỉ để lộ một giao diện giới hạn cho phần còn lại của chương trình.  Thực tế ***Interface*** là từ viết tắt của một phần công khai của đối tượng. | **Encapsulation**  Encapsulation is the ability of an object to hide parts of its state and behaviors from other objects, exposing only a limit­ed interface to the rest of the program. |

# Design Pattern (GOF)

## Facade pattern [fa sa:d]

<https://github.com/doctor-blue/design-patterns/tree/master/Structural/Facade>

* Mỗi một hàm chỉ làm một việc duy nhất. Nguyên tắc S trong solid ( Single Responsibility Principles)
* Gói một subsystem vào một class và cung cấp interface để client sử dụng đơn giản hơn
* Trong trường hợp bạn có nhiều hệ thống con và mỗi lần gọi tới những hệ thống này lại quá sức phức tạp và tốn nhiều thời gian. Vậy **Facade** sẽ giúp đỡ bạn trong việc này. Cũng giống như [**Adapter**](https://github.com/doctor-blue/design-patterns/tree/master/Structural/Adapter)**Facade** cung cấp cho chúng ta một lớp để làm chung gian giữa hai thành phần, nhưng khác nhau ở chỗ Adapter thì giúp chúng ta làm việc giữa hai mã nguồn, còn đối với Facade thì thì sẽ là thành phần bao bọc che giấu đi các hệ thống con để giúp giao tiếp với thành phần khác dễ dàng hơn.
* Như vậy ta có thể hiệu Facade sẽ cung cấp một giao diện cho việc giao tiếp giữa các hệ thống con, giúp người dùng có thể giao tiếp với chúng một cách dễ dàng. Facade giúp che dấu đi xử lý logic phức tạp bên trong hệ thống con.

## Strategy pattern

* Để giải quyết một việc (tùy thuộc arguments) thì sẽ cho một kết quả trong một lần thực thi ( Switch case )

1. a