What is an object?

* An object is an real-world entity, such as a car, dog, cat … During its life cycle, an object has a state and behaviors. For example: a cat’s state can be color, name, while behaviors can be eating, sleeping… In java, an object is an instance of a class usually built via new keyword and it has state stored in fields and exposes its behavior through methods. Each instance takes some space in memory and can communicate with other object.

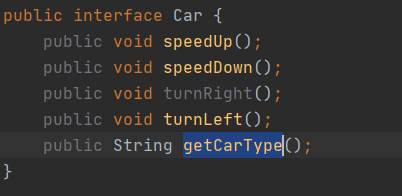
What is a class?

* A class is a set of instructions that are required to build a specific type of object. We can think of a class as a template, a blueprint, or a recipe that tells us how to create object of a class.
* A class doesn’t consume memory.
* A class should be designed and writtent to do one, and only one, thing.

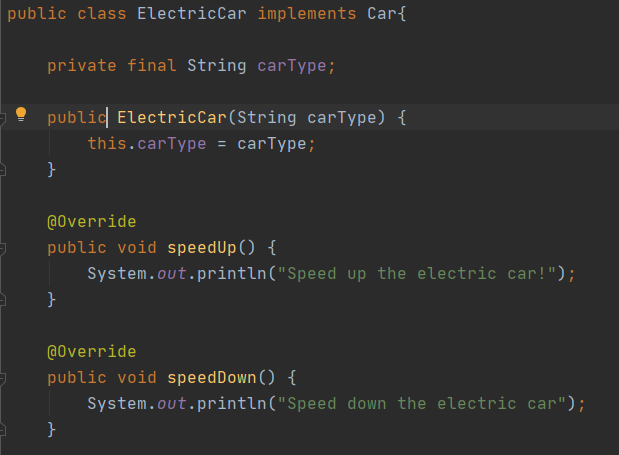
What is abstraction?

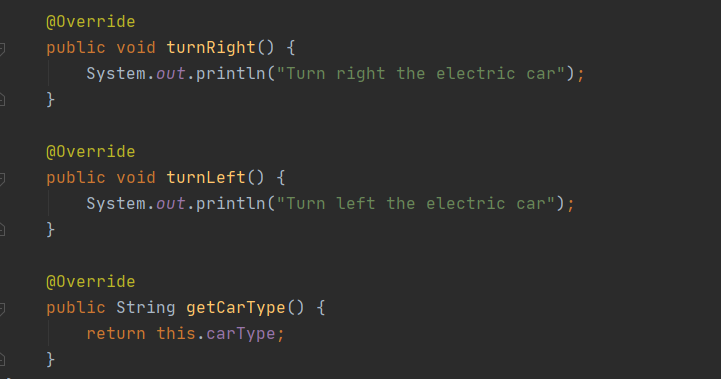
* Là một tiến trình ẩn đi các chi tiết, chỉ hiển thị tính năng tới người dùng.
* Tính trừu tượng cho phép bạn loại bỏ đi tính phức tạp của đối tượng bằng cách chỉ đưa ra các thuộc tính và phương thức cần thiết của đối tượng trong lập trình.
* VD: một người đàn ông lái xe ô tô. Người đàn ông biết mỗi bàn đạp làm gì và vô lăng làm gì, nhưng anh ta không biết những thứ này được thực hiện bên trong xe như thế nào. Anh ấy không biết về cơ chế bên trong trao quyền cho những thứ này.
* In Java, abstraction can be achieved via abstract classes and interfaces.

Interface Car.java

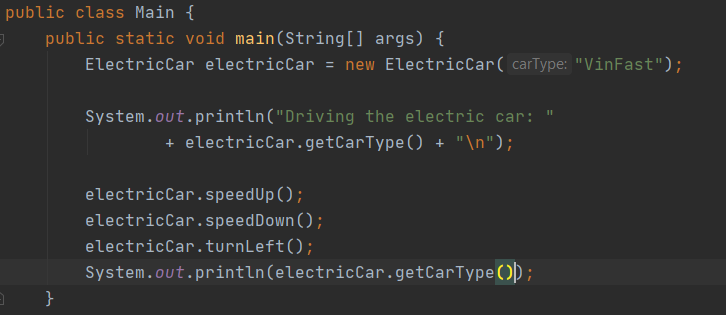


Class ElectricCar

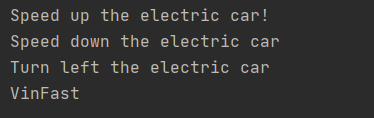




Main.java



Kết quả:



Encapsulation là gì ?

* Ẩn giấu đi thông tin không liên quan và hiển thị ra thông tin liên quan.

Contructor là gì ?

* Contructor là một dạng đặt biệt của phương thức, dùng để khởi tạo object.

This là gì ?

* Là một biến tham chiếu được sử dụng để tham chiếu tới đối tượng của lớp hiện tại.

Super là gì ?

* Là một biến tham chiếu được sử dụng để tham chiếu trực tiếp tới đối tượng của lớp cha gần nhất.

Final là gì ?

* Được dùng để hạn chế người dùng. Từ khóa final có thể được sử dụng trong nhiều ngữ cảnh.
  + Biến final: bạn không thể thay đổi giá trị của biến final (hằng số)
  + Phương thức final: không thể ghi đè, nhưng có thể kế thừa.
  + Lớp final: không thể kế thừa
  + Biến static final trống: một biến final mà không được khởi tạo tại thời điểm khai báo thì được gọi là biến final trống.

Static là gì ?

* Được sử dụng để quản lý bộ nhớ. Chúng ta có thể áp dụng static cho biến, phương thức, khối, nested class. Từ khối static thuộc về class chứ không phải thuộc về object.
* Trong java static có thể là:
  + Biến static: khi bạn khai báo biến là static, thì biến đó được gọi là biến tĩnh.
  + Phương thức static
  + Khối static

Biến local là biến khai báo bên trong method.

Biến instance là biến khai báo bên ngoài method.

Biến static là biến được khai báo với từ khóa static và bên ngoài method.

Vùng nhớ heap:

* Là vùng nhớ lưu trữ các đối tượng khi từ khóa new được gọi, các biến static và biến toàn cục (biến instance).

Vùng nhớ stack:

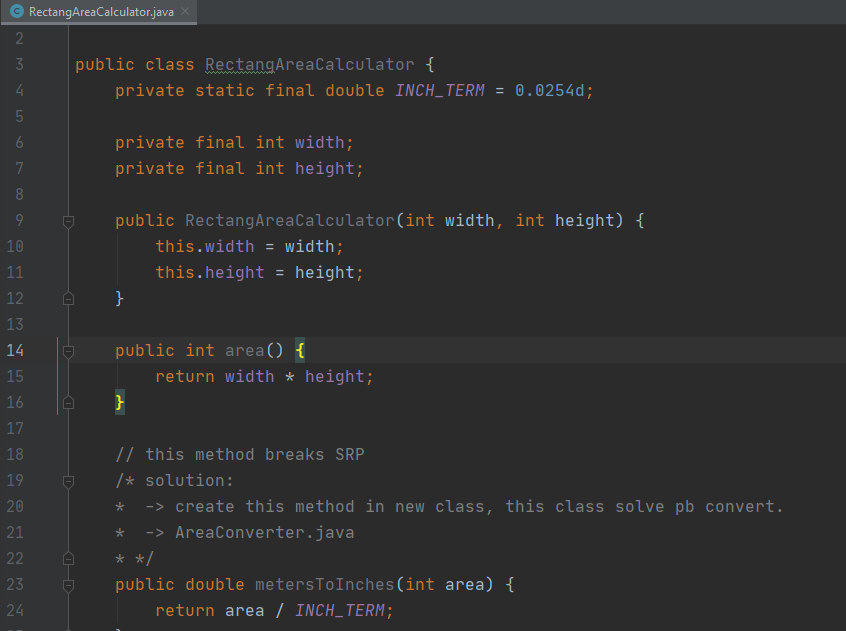
* Là vùng nhớ lưu trữ các tham số, các biến local mỗi khi method được gọi.

Composition là khai báo tham chiếu của class trong một class khác (HAS-A).

SOLID PRINCIPLE

1. Single Responsibility Principle.

* S stands for One class should have one, and only one, responsibility.
* S tells us to write a class for only one goal.
* This principle suitains encapsulation.



1. Open Closed Principle

* O stands for Software components should be open for extention, but closed for modification.
* Muốn thay đổi hành vi của 1 lớp thì chúng ta cần tạo ra 1 lớp mới extend lại lớp hiện tại

1. Liskov substitution Principle

* Trong một chương trình, các object class con có thể thay thế class cha mà không làm thay đổi tính đúng đắn của chương trình.
* Những quy phạm về nguyên lý LSP
  + Các lớp dẫn xuất có method override lại method lớp cha nhưng chức năng hoàn toàn khác
  + Các lớp dẫn xuất có method overrid lại method lớp cha nhưng lại rỗng.
  + Các phương thức bắt buộc kế thừa từ lớp cha ở lớp dẫn xuất nhưng không được sử dụng.
  + Phát sinh ngoại lệ trong lớp dẫn xuất.
* Đây là nguyên lý dễ vi phạm nhất trong SOLID.

VD: A là B (hình vuông là hình chữ nhật) không có nghĩa là class A nên kế thừa class B. Chỉ cho class A kế thừa class B khi class A thay thế được cho class B.

1. Interface segregation principle (ISP)

* Thay vì dùng một interface lớn, ta nên tách interface thành nhiều interface nhỏ, với nhiều mục đích cụ thể.

1. Dependency inversion principle (DIP)

* Các module cấp cao không nên phụ thuộc vào các modules cấp thấp. Cả 2 nên phụ thuộc vào abstraction
* Interface (abstraction) không nên phụ thuộc vào chi tiết, mà ngược lại. (Các class giao tiếp với nhau thông qua interface, không phải qua implementation).

DESIGN PATTERN

Trong design pattern có 3 nhóm gồm:

* Creational Pattern: sẽ sử dụng một số thủ thuật để khởi tạo đối tượng mà bạn sẽ không nhìn thấy từ khóa new.
* Structural Pattern: thiết lập, định nghĩa quan hệ giữa các đối tượng.
* Behavioral Pattern: thực hiện hành vi của đối tượng.

Runtime: là thời điểm chương trình đang chạy

Compile time là thời điểm chương trình đang được compile

* Thông thường runtime và compile time được nhắc tới khi nói tới lỗi.

Runtime là lỗi khi chạy mới thấy, còn compile time là lỗi compile là thấy liền.

1. Overriding là gì ?

* 2 phương thức có cùng tên nhưng khác hành vi.
* Ghi đè phương thức được sử dụng khi kế thừa hoặc runtime Polymorphism.
* Trong kế thừa, ghi đè ở phương thức con.
* Trong runtime Polymorphism, ghi đè lại phương thức implement interface.

Rules:

* Giống tên giống kiểu trả về như method cha.
* Không ghi đè được phương thức trong cùng class.
* Không ghi đè được private, static và final method.

##### Tại sao không ghi đè được phương thức static?

* Vì phương thức static được ràng buộc với class còn phương thức instance được ràng buộc với đối tượng. Static thuộc về vùng nhớ class còn instance thuộc về vùng nhớ heap.

1. Overloading là gì ?

* 2 phương thức cùng tên, khác nhau về kiểu dữ liệu hoặc số lượng tham số.

Có 2 cách nạp chồng:

C1: Nạp chồng thay đổi số lượng tham số

C2: Nạp chồng thay đổi kiểu dữ liệu của tham số

Rules:

* Có thể nạp chồng private, static và final method.
* Có thể nạp chồng phương thức trong cùng class.

Câu hỏi 1: Tại sao không thể nạp chồng phương thức bằng cách chỉ thay đổi kiểu trả về của phương thức?

* Trong java, không thể nạp chồng phương thức bằng cách chỉ thay đổi kiểu trả về của phương thức bởi vì không biết phương thức nào sẽ được gọi.
* Compile time error.

Tạo ví dụ.

Can we have a non-abstract method inside a Java interface?

* Trước java 8 thì không, nhưng khi có java 8 thì được. Sử dụng từ khóa default để khai báo 1 non-abstract method bên trong interface.

|  |  |  |
| --- | --- | --- |
|  | Abstract class | Interface |
| Support method | Phương thức abstract có thân hàm và không có thân hàm | Là phương thức abstract không có thân hàm, đến java 8 thì có thêm phương thức default và static |
| Đa kế thừa | Không | Có |
| Support variable | Final, non-final, static, non-static | Static và final |
| implementation | Can implement an interface | Only extend interface |

COLLECTION

List: danh sách chứa các phần tử có thể trùng lập nhau

Set: danh sách chứa các phần tử không trùng lập nhau

Xử lý ngoại lệ

Exception Handling trong java hay xử lý ngoại lệ trong java là một cơ chế mạnh mẽ để xử lý các lỗi runtime để duy trì luồng bình thường của ứng dụng.

1. Catch

* Khối lệnh try trong java được sử dụng để chứa một đoạn code có xảy ra một ngoại lệ. Nó phải được khai báo trong method.
* Sau một khối lệnh try bạn phải khai báo một khối lệnh catch hoặc finally hoặc cả 2.
* Có thể sử dụng nhiều khối catch với một khối try duy nhất.
* Nếu bạn phải thực hiện các tác vụ khác nhau mà ở đó có thể xảy ra nhiều ngoại lệ khác nhau, hãy sử dụng đa khối lệnh catch trong java.
* Quy tắc: Vào một thời điểm chỉ xảy ra một ngoại lệ và tại một thời điểm chỉ có một khối catch được thực thi.
* Quy tắc: Tất cả các khối lệnh catch phải được sắp xếp cụ thể nhất đến chung nhất, tức là phải khai báo khối lệnh catch để xử lý lỗi ArithmeticException trước khi khai báo catch để xử lý lỗi Exception.