What is object-oriented programming?

What is object-oriented programming? OOP explained in depth (educative.io)

Lập trình hướng đối tượng (Object-oriented programming – OOP) là một mô hình lập trình cơ bản được sử dụng bởi mọi lập trình viên. OOP là một mô hình lập trình phổ biến nhất hiện nay, được dạy như những thứ căn bản nhất một lập trình viên cần biết. Một mô hình lập trình phổ biến khác là lập trình hướng chức năng, nhưng chúng ta sẽ không đề cập trong bài viết này.

Hôm nay, chúng ta sẽ phân tích những thứ cơ bản tạo nên lập trình hướng đối tượng. Để bạn có thể sử dụng nó trong thuật toán, dự án và phỏng vấn.

Let’s go!!!!!

- Object-oriented programming

Object-oriented programming

Lập trình hướng đối tượng là gì?

Lập trình hướng đối tượng là một mô hình lập trình trong khoa học máy tính dựa trên khái niện về lớp (class) và đối tượng (object). Nó được sử dụng để giúp cấu trúc phần mềm đơn giản, có thể tái sử dụng, được sử dụng để tạo ra các phiên bản khác nhau của đối tượng. JavaScript, C++, C#, Java… những ngôn ngữ lập trình hướng đối tượng phổ biến.

Ngôn ngữ lập trình hướng đối tượng không nhất thiết phải bị giới hạn trong mô hình lập trình hướng đối tượng. Một số ngôn ngữ như JavaScript, Python, PHP… cho phép cả lập trình hướng đối tượng và lập trình hướng thủ tục.

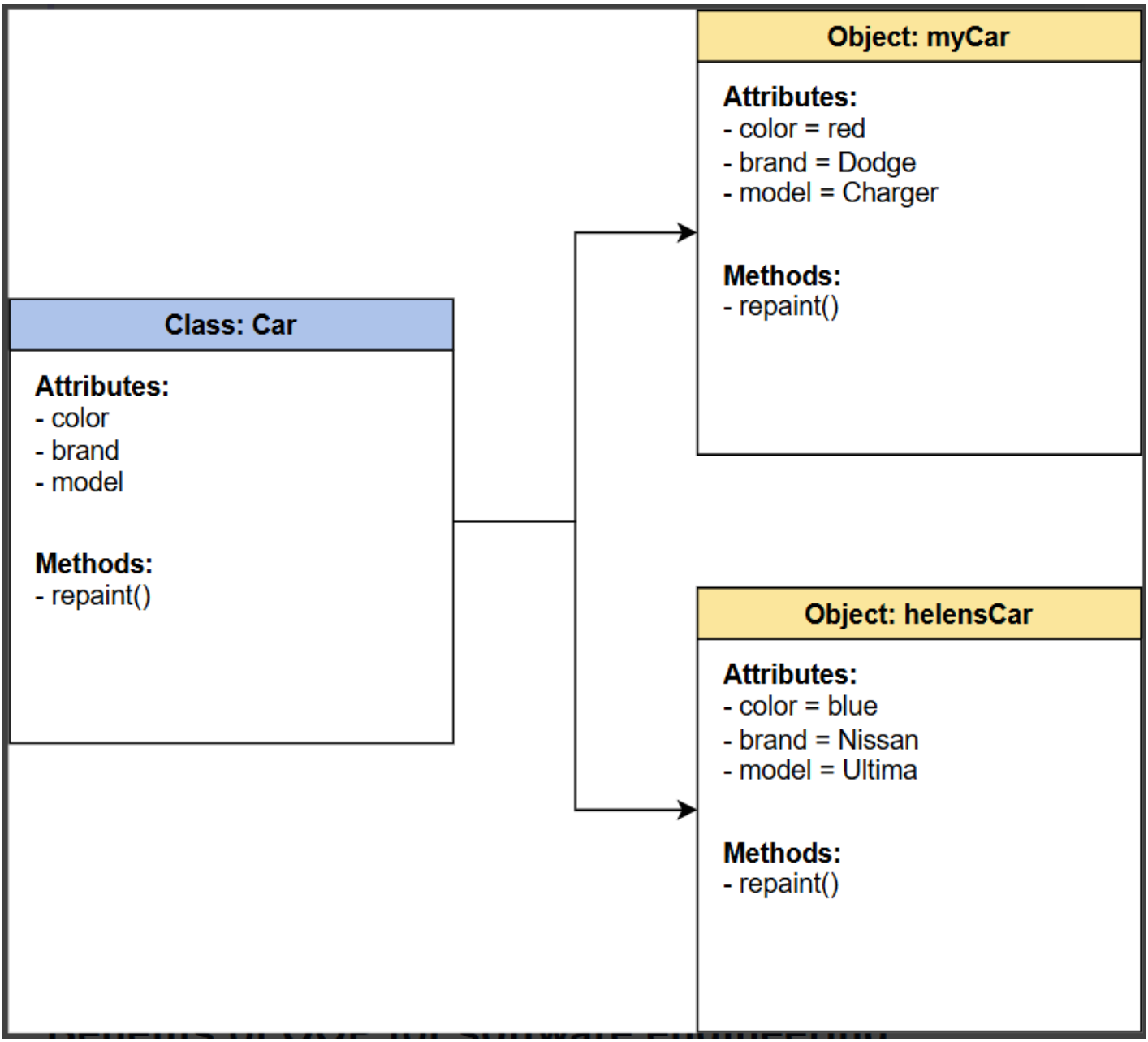
Lớp (class) có thể được coi là một bản thiết kế trừu tượng để tạo ra các đối tượng cụ thể hơn. Lớp thường đại diện cho các thể loại rộng như *chó, ô* tô có các thuộc tính riêng. Các lớp này các định thuộc tính mà các đối tượng thuộc loại này sẽ có ví dụ như *màu sắc* nhưng không bao gồm giá trị cụ thể cho đối tượng.

Lớp cũng có thể chưa các function gọi là phương thức (method) và chỉ có sẵn cho các đối tượng thuộc loại đó. Các function này được khai báo trong lớp và thể hiện một số hành động hữu ích cho loại đối tượng cụ thể đó.

Ví dụ: lớp Car có phương thức Repaint sẽ thay đổi thuộc tính màu sắc của xe, Chức năng này chỉ có ích với đối tượng là Car vì chúng ta khai báo nói trong lớp Car, nó tạo ra một phương thức.

Các lớp mẫu được sử dụng như bản thiết kế để taọ ra các đối tượng độc lập. Chúng đại diện cho các ví dụ cụ thể về lớp trừu tượng kiểu như myCar hay goldenRetriever. Mỗi đối tượng có thể có những giá trị độc nhất cho các đặc tính được khai báo trong class.

Ví dụ chúng ta tạo ra lớp Car chứ tất cả các thuộc tính mà ô tô cần có như màu, thương hiệu, mẫu mã. Sau đó tạo ra một phiên bản của Car. myCar đại diện cho chiếc xe cụ thể.
Sau đó chúng ta có thể đặt giá trị cho các thuộc tính đã được khai báo mà không làm ảnh hưởng đến các đối tượng khác hoặc lớp mẫu.
Chúng ta có thể tái sử dụng lớp này để đại diện cho bất kỳ ô tô nào.



Lợi ích của OOP với phát triển phần mềm:

- OOP mô hình hoá những thứ phức tạp dưới dạng đơn giản và có thể tái tạo.
- Có thể tái sử dụng. Đối tượng có khả năng tái sử dụng giữa nhiều chương trình.
- Tính đa hình cho phép thực hiện hành vi dành riêng cho lớp.
- Dễ dàng sửa lỗi. Lớp thường chứa tất cả thông tin có thể ứng dụng cho chúng.
- Tính đóng gói giúp bảo vệ an toàn những thông tin nhạy cảm.

Cách cấu trúc chương trình OOP

Hãy lấy một ví dụ thực tế. Hãy tưởng tượng bạn đang điều hành một trang trại với hàng trăm con thú cưng và bạn phải lưu tên, tuổi, ngày nhận cho môi con.

Làm sao để thiết kế đơn giản, có thể tái sử dụng?

Với hàng trăm con chó, sẽ rất kém hiệu quả khi khai bao từng con bởi vì bạn bạn sẽ phải viết rất nhiều code thừa thãi lặp đi lặp lại. Dưới đây chúng ta có thể thấy đối tượng rufus và fluffy. Có rất nhiều code lặp đi lặp lại trong các object ví dụ như `name`, `age`. Với những trường hợp như thế này chúng ta có thể sử dụng class và object để thay thế.

```
//Object of one individual dog
var rufus = {
    name: "Rufus",
    birthday: "2/1/2017",
    age: function() {
        return Date.now() - this.birthday;
    },
    attendance: 0
}

//Object of second individual dog
var fluffy = {
    name: "Fluffy",
    birthday: "1/12/2019",
    age: function() {
        return Date.now() - this.birthday;
    },
    attendance: 0
}
```

Việc nhóm các thông tin liên quan lại với nhau để tạo thành cấu trúc lớp giúp code ngắn hơn và dễ bảo trì hơn.

Áp dụng OOP vào ví dụ trang trại chó trên

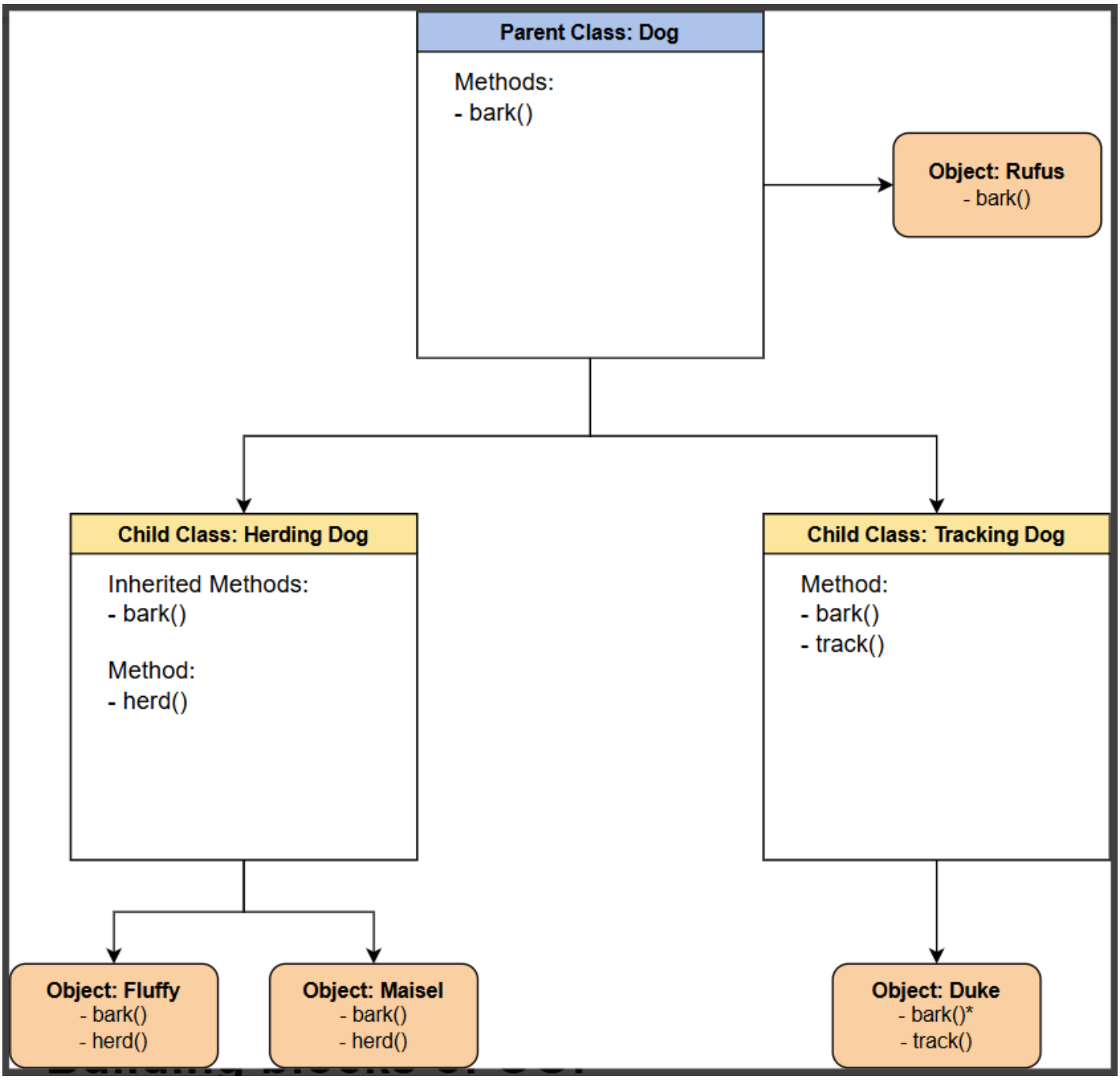
- Tạo một lớp cho tất cả các con chó. Có thể coi là một bản thiết kế bao gồm thông tin và hành vi mà tất cả các con chó đều có bất kể nó thuộc giống chó nào. Chúng ta gọi đó là **lớp cha (parent class)**.
- Tạo các **lớp con (child class)** bao gồm các đặc điểm và hành vi đặc trưng cho mỗi loài chó.
- Tạo đối tượng từ các lớp con đại diện cho giống chó đó.

Sơ đồ dưới đây thể hiện cách thiết kế một chương trình hướng đối tượng bằng cách nhóm các dữ liệu và hành vi liên quan lại với nhau tạo thành các mẫu đơn giản sau đó tạo các nhóm phụ cho dữ liệu và hành vi đặc biệt.

Lớp `Dog` là một mẫu chung chỉ chứa dữ liệu và hành vi chung mà tất cả các loài chó đều có.

Sau đó chúng ta tạo 2 lớp con của `Dog` là `HerdingDog` và `TrackingDog`. Chúng kế thừa hành vi của `Dog` nhưng có thêm những hành vi độc nhất của giống chó đó.

Cuối cùng, chúng ta tạo đối tượng thuộc loại `HerdingDog` để đại diện cho từng con chó. Ví dụ như `Fluffy` và `Maisel`.



Thành phần tạo nên OOP

Tiếp theo chúng ta sẽ tìm hiểu sâu hơn về những thành phần cơ bản tạo nên một chương trình hướng dữ liệu:

- Lớp (Class)
- Đối tượng (Object)
- Phương thức (Method)
- Thuộc tính (Attributes)

L?p (Class)

Đơn giản thì các lớp là các **kiểu dữ liệu mà người dùng tự định nghĩa**. Các lớp là nơi mà chúng ta tạo ra bản thiết kế cho cấu trúc của phương thức và thuộc tính. Các đối tượng đều được tạo ra từ bản thiết kế này.

Lớp chứa các trường cho thuộc tính và các phương thức cho hành vi. Trong ví dụ về chó ở trên, thuộc tính bao gồm: `name`, `birthday`, `age`... trong khi method bao gồm `bark()` và `updateAttendance()`.

Dưới đây là đoạn code tạo ra lớp chó trong JavaScript

```
class Dog {
    constructor(name, birthday) {
        this.name = name;
        this.birthday = birthday;
    }

    //Declare private variables
    _attendance = 0;

    getAge() {
        //Getter
        return this.calcAge();
    }

    calcAge() {
        //calculate age using today's date and birthday
        return Date.now() - this.birthday;
    }

    bark() {
        return console.log("Woof!");
    }

    updateAttendance() {
        //add a day to the dog's attendance days at the petsitters
        this._attendance++;
    }
}
```

Hãy nhớ rằng lớp là một mẫu để mô hình hoá con chó và một đối tượng được khởi tạo từ lớp thì đại diện cho một vật phẩm cụ thể trong thế giới thực.

??i t??ng (Object)

Không có gì ngạc nhiên khi nói đối tượng là một phần rất lớn của OOP. Đối tượng là **thể hiện của một lớp** được tạo ra với dữ liệu đặc biệt. Ví dụ với đoạn code dưới đây `Rufus` là thể hiện của lớp `Dog`.

```
class Dog {
    constructor(name, birthday) {
        this.name = name;
        this.birthday = birthday;
    }

    //Declare private variables
    _attendance = 0;

    getAge() {
        //Getter
        return this.calcAge();
```

```
    }

    calcAge() {
        //calculate age using today's date and birthday
        return Date.now() - this.birthday;
    }

    bark() {
        return console.log("Woof!");
    }

    updateAttendance() {
        //add a day to the dog's attendance days at the petsitters
        this._attendance++;
    }
}

//instantiate a new object of the Dog class, and individual dog named Rufus
const rufus = new Dog("Rufus", "2/1/2017");
```

Khi `new Dog` được gọi:

- Một đối tượng mới được tạo ra với tên `Rufus`
- Hàm tạo gán các giá trị `name` và `birthday`.

N/A	Là gì?	Chứa thông tin	Hành động	Ví dụ
Lớp (Class)	Bản thiết kế	Thuộc tính	Hành vi được khai báo thông qua phương thức	Mẫu Dog
Đối tượng (Object)	Thể hiện của lớp	Trạng thái, Dữ liệu	Phương thức	Con chó tên là Rufus, Fluffy

Thu?c tính (Attribute)

Thuộc tính là thông tin được lưu trữ. Thuộc tính được khai báo bên trong lớp. Khi đối tượng được khởi tạo, các đối tượng riêng lẻ lưu trữ dữ liệu trong các trường thuộc tính.

Trạng thái của một đối tượng được xác định dựa trên những trường thuộc tính của đối tượng đó. Ví dụ, một con chó con và một con chó trưởng thành có thể được chăm sóc khác nhau ở trang trại. Ngày sinh có thể xác định trạng thái của đối tượng và cho phép phần mềm xử lý những con chó có độ tuổi khác nhau theo cách khác nhau.

Ph??ng th?c (Method)

Phương thức đại diện cho hành vi. Phương thức thể hiện hành động. Phương thức trả về thông tin của đối tượng hoặc cập nhật dữ liệu của đối tượng. Phương thức nằm bên trong một class.

Khi một đối tượng được khởi tạo, đối tượng này có thể gọi phương thức được khai báo trong lớp. Trong đoạn code dưới đây phương thức `bark` được khai báo trong lớp `Dog` và `bark()` có thể được gọi bởi `Rufus`

```
class Dog {
    //Declare protected (private) fields
    _attendance = 0;

    constructor(name, birthday) {
        this.namee = name;
        this.birthday = birthday;
    }

    getAge() {
        //Getter
        return this.calcAge();
    }

    calcAge() {
        //calculate age using today's date and birthday
        return this.calcAge();
    }

    bark() {
        return console.log("Woof!");
    }

    updateAttendance() {
        //add a day to the dog's attendance days at the petsitters
        this._attendance++;
    }
}
```

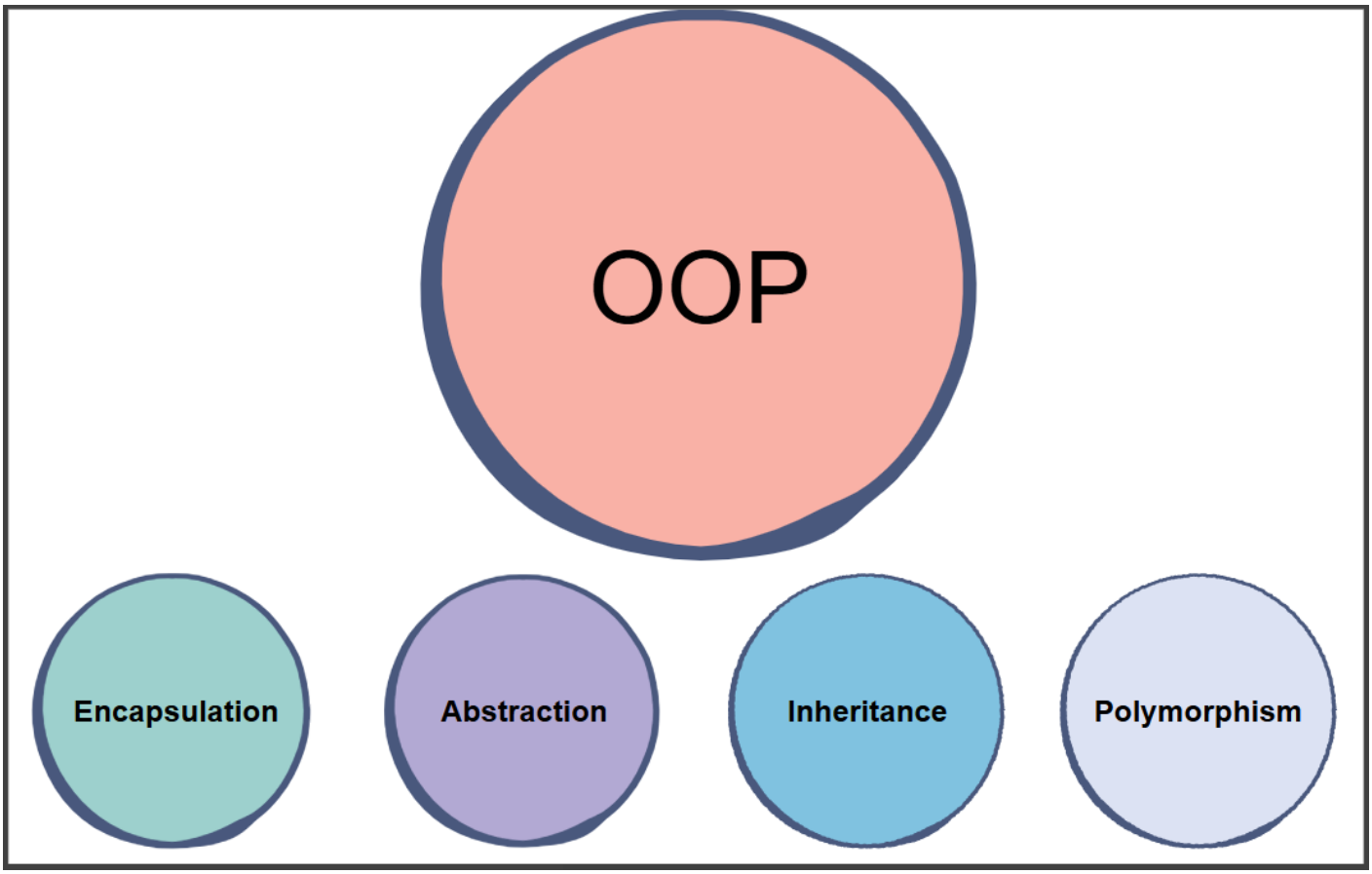
Phương thức thường chỉnh sửa, cập nhật hoặc xoá dữ liệu. Tuy nhiên không phải tất cả các phương thức đều thay đổi dữ liệu. Ví dụ phương thức `bark()` không cập nhật bất kỳ dữ liệu nào bởi vì sủa không ảnh hưởng đến bất kỳ trường dữ liệu nào của lớp `Dog`.

Phương thức `updateAttendance()` thêm một ngày con chó được chăm sóc trong trại. Thuộc tính `_attendance` rất quan trọng để thanh toán tiền cho chủ chó vào cuối tháng.

Bốn tính chất của OOP

Bốn trụ cột của lập trình hướng đối tượng là:

- **Tính kế thừa (inheritance)**: Lớp con được kế thừa dữ liệu và hành vi từ lớp cha.
- **Tính đóng gói (encapsulation)**: Dữ liệu được chưa trong một đối tượng chỉ show ra những thông tin cần thiết.
- **Tính trừu tượng (abstraction)**: Chỉ truy cập đối tượng thông qua các phương thức có mức độ công khai cao.
- **Tính đa hình (polymorphism)**: Một phương thức có thể thực hiện những nhiệm vụ khác nhau.



Tính k? th?a (Inheritance)

Tính kế thừa cho phép các lớp kế thừa tính năng của một lớp khác. Nói cách khác lớp lớp cha cung cấp thuộc tính, hành vi cho lớp con. Tính kế thừa còn hỗ trợ cho khả năng tái sử dụng.

Nếu thuộc tính cơ bản và hành vi được khai báo ở lớp cha, lớp con có thể tạo, mở rộng tính năng của lớp cha, thêm các thuộc tính và hành vi khác.

Ví dụ chó chăn cừu có khả năng độc đáo là chăn gia súc. Nói cách khác thì tất cả chó chăn cừu đều là chó. Nhưng không phải con chó nào cũng là chó chăn cừu. Chúng ta thể hiện sự khác biệt đó bằng cách tạo ra lớp con `HerdingDog` từ lớp cha `Dog` rồi sau đó thêm hành động `herd()`.

Lợi ích của tính kế thừa là chương trình có thể tạo một lớp cha chung sau đó tạo ra nhiều lớp con đặc biệt hơn nếu cần. Điều này giúp đơn giản hoá việc lập trình, thay vì phải tạo ra lớp `Dog` nhiều lần, các lớp con tự động được truy cập vào các tính năng của lớp cha.

Trong đoạn code dưới, lớp con `HerdingDog` kế thừa phương thức `bark` từ lớp cha `Dog` và lớp con có thể thêm phương thức `herd()`.

```
//Parent class Dog
class Dog{
    //Declare protected (private) fields
    _attendance = 0;

    constructor(namee, birthday) {
        this.name = name;
        this.birthday = birthday;
```

```
    }

    getAge() {
        //Getter
        return this.calcAge();
    }

    calcAge() {
        //calculate age using today's date and birthday
        return this.calcAge();
    }

    bark() {
        return console.log("Woof!");
    }

    updateAttendance() {
        //add a day to the dog's attendance days at the petsitters
        this._attendance++;
    }
}

//Child class HerdingDog, inherits from parent Dog
class HerdingDog extends Dog {
    constructor(name, birthday) {
        super(name);
        super(birthday);
    }

    herd() {
        //additional method for HerdingDog child class
        return console.log("Stay together!")
    }
}
```

Chú ý rằng lớp `HerdingDog` không hề khai báo phương thức `bark()`. Nó kế thừa từ lớp cha `Dog`.

Khi gọi phương thức `fluffy.bark()`. Phương thức `bark()` sẽ đi ngược lên lớp cha, tìm nơi mà phương thức `bark` được khai báo.

```
//Parent class Dog
class Dog{
    //Declare protected (private) fields
    _attendance = 0;

    constructor(namee, birthday) {
        this.name = name;
        this.birthday = birthday;
    }

    getAge() {
        //Getter
        return this.calcAge();
    }

    calcAge() {
        //calculate age using today's date and birthday
        return this.calcAge();
    }

    bark() {
        return console.log("Woof!");
    }
```

```
    updateAttendance() {
        //add a day to the dog's attendance days at the petsitters
        this._attendance++;
    }
}

//Child class HerdingDog, inherits from parent Dog
class HerdingDog extends Dog {
    constructor(name, birthday) {
        super(name);
        super(birthday);
    }

    herd() {
        //additional method for HerdingDog child class
        return console.log("Stay together!")
    }
}

//instantiate a new HerdingDog object
const fluffy = new HerdingDog("Fluffy", "1/12/2019");
fluffy.bark();
```

*Chú ý***:** *Lớp cha còn có thể được gọi là superclasses hoặc base classes. Lớp con cũng có thể được gọi là subclass, derived class, or extended class.*

Trong ví dụ dưới ta thấy cả 3 đối tượng đều có thể sủa. Hành vi chăn gia súc được khai báo ở lớp chó chăn cừu nên chỉ có Maisel và Fluffy có thể truy cập phương thức này. R được khởi tạo từ lớp chó nên chỉ có thể truy cập hành vi sủa.

Object	Instantiated from Class	Parent Class	Methods
Rufus	Dog	N/A	bark()
Maisel	Herding Dog	Dog	bark(), herd()
Fluffy	Herding Dog	Dog	bark(), herd()

Tính ?óng gói (Encapsulation)

Tính đóng gói là giữ tất cả các thông tin quan trọng trong một đối tượng và chỉ show những thông tin cần thiết ra bên ngoài. Thuộc tính và hành vi được định nghĩa bởi code bên trong của lớp mẫu.

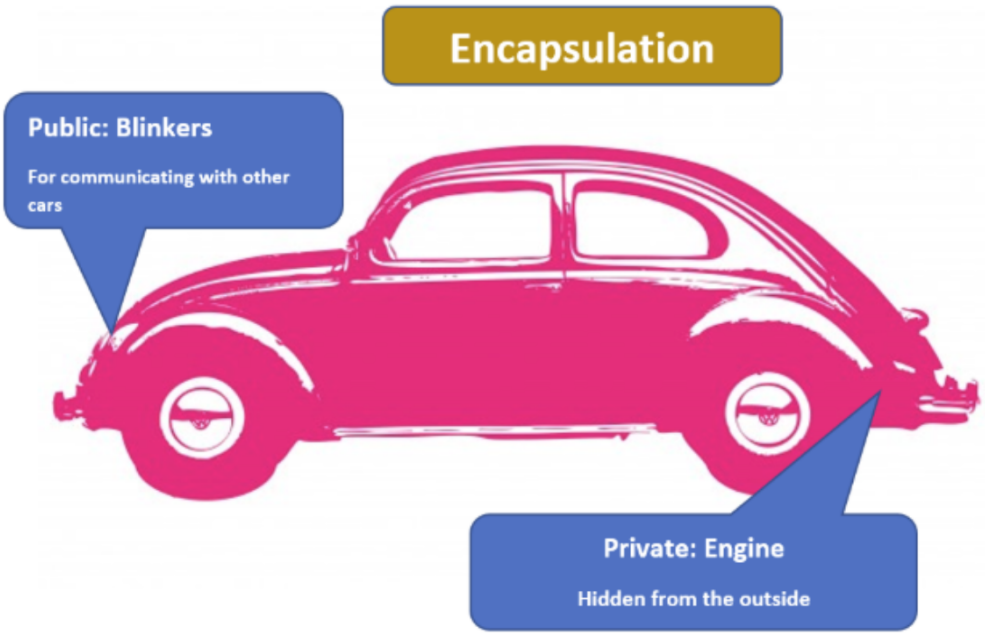
Sau đó, khi một đối tượng được khởi tạo từ một lớp, dữ liệu và phương thức sẽ được đóng gói bên trong đối tượng đó. Đóng gói ẩn cách triển khai phần mềm bên trong một class và ẩn dữ liệu bên trong một đối tượng.

Tính đóng gói yêu cầu khai báo private cho một số trường và public cho một số trường khác.

- **Private/Internal interface:** phương thức và đặc tính chỉ có thể được truy cập từ các phương thức khác nhau trong cùng class.
- **Public/External interface:** phương thức và đặc tính có thể được truy cập từ bên ngoài class.

Lấy một chiếc xe ô tô làm ẩn dụ cho tính bao đóng gói. Thông tin của chiếc xe, sử dụng xi nhan để báo rẽ là những thông tin được public. Ngược lại, động cơ được giấu dưới mui xe.

Khi bạn lái xe trên đường, những chiếc xe khác sẽ cần thông tịn từ bạn để đưa ra quyết định ví dụ như kích thước chiếc xe của bạn, bạn muốn rẽ trái hay phải. Mặt khác, việc chia sẻ những thông tin như nhiệt độ xe... là thừa thãi và có thể gây rối cho những tài xế khác.



Tính đóng gói giúp tăng cường bảo mật. Thuộc tính và phương thức có thể đặt private nên chúng không thể bị truy cập từ bên ngoài class. Để lấy thông tin về dữ liệu của một đối tượng, phương thức và thuộc tính public được sử dụng để truy cập và cập nhật dữ liệu.

Cách này thêm một lớp bảo mật, lập trình viên có thể chọn dữ liệu nào có thể được xem thông thông qua các phương thức public.

Trong một class hầu hết các ngôn ngữ lập trình đều có các thành phần public, protected và private.

- Thành phần **public** cho phép thuộc tính và phương thức có thể được truy cập từ bên ngoài hoặc từ class khác của chương trình.
- Thành phần **protected** chỉ có thể truy cập từ class đó và các class con của nó.
- Thành phần **private** chỉ có thể truy cập bên trong class (kể cả các class con cũng không thể truy cập).

Quay lại ví dụ trại chó, tính đóng gói là giúp bảo mật thông tin của chó và chủ của nó. Chủ chó khác không thể xem thông tin của bạn cũng như chó của bạn.

Đóng gói và cập nhật dữ liệu: Bởi vì phương thức cũng có thể cập nhật dữ liệu của đối tượng nên lập trình viên có thể kiểm soát dữ liệu nào được phép thay đổi thông qua phương thức.

Nó cho phép chúng ta ẩn giấu thông tin quan trọng khỏi khác đối tượng lừa đảo hoặc tránh việc thay đổi nhầm dữ liệu.

Tính bao đóng tăng tính bảo mật cho code của bạn và đơn giản hoá quá trình làm việc với các lập trình viên khác. Khi bạn lập trình bạn sẽ khộng muốn chia sẻ class gốc hoặc dữ liệu private với các công ty khác vì công ty của bạn có quyền sở hữu trí tuệ với tài sản đó.

Thay vào đó, lập trình viên tạp ra các phương thức public cho phép lập trình viên khác gọi phương thức đó thông qua object. Lý tưởng nhất là để các phương thức đó vào tài liệu mà bạn sẽ gửi cho các lập trình viên khác.

Tóm lại, lợi ích của tính đóng gói bao gồm:

- **Tăng cường bảo mật:** Giới hạn dữ liệu có thể truy cập của đối tượng.
- **Tránh các lỗi thường gặp:** Tránh các lỗi do thay đổi nhầm dữ liệu.
- **Bảo vệ quyền sở hữu trí tuệ:** Code được giấu trong các class. Và các lập trình viên khác chỉ có thể truy cập thông qua các public class.
- **Có thể hỗ trợ:** Hầu hết code đều có thể được cập nhật và cải tiến.
- **Đơn giản hoá:** Không phải ai cũng cần biết tất cả thuộc tính và phương thức của đối tượng. Biết quá nhiều đôi khi gây ra sự bối rối.

Tính tr?u t??ng (Abstraction)

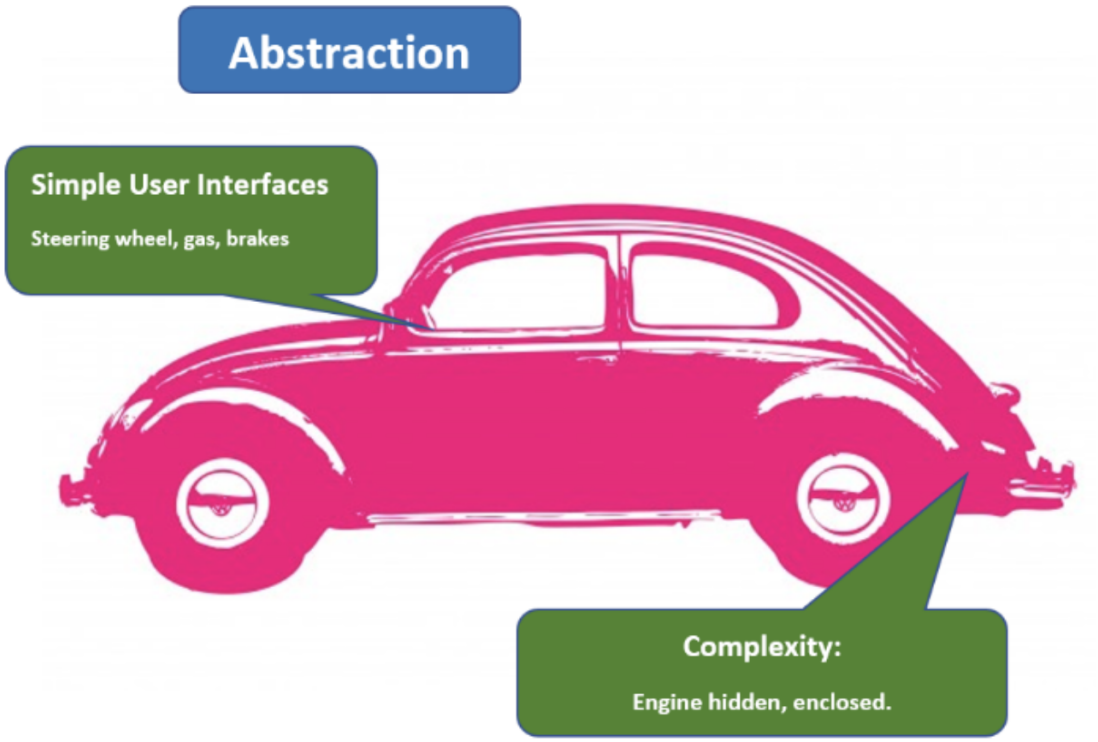
Tính trừu tượng có thể coi là một phần mở rộng của tính đóng gói sử dụng các lớp và đối tượng chứa code và dữ liệu để ẩn giấu chi tiết chương trình với người dùng. Điều này được thực hiện bằng cách tạo ra **lớp trừu tượng** giữa người dùng và mã nguồn phức tạp hơn. Nó giúp bảo vệ thông tin nhạy cảm bên trong mã nguồn.

Tính trừu tượng

- Giảm sự phức tạp và cải thiện khả năng đọc của code
- Tạo điều kiện để tái sử dụng code
- Tăng cường bảo mật dữ liệu bằng cách ẩn giấu thông tin nhạy cảm
- Năng cao năng suất bằng cách loại bỏ các chi tiết ở tầng thấp.

Tính trừu tượng cũng có thể được giải thích thông qua cái ô tô. Hãy nghĩ về cách tài xế lái xe.

Một tài xế sử dụng vô lăng, chân ga và phanh để điều khiển chiếc xe mà không cần lo lắng là động cơ hoạt động như nào hay bộ phận nào được sử dụng cho việc chuyển động. Đó là tính trừu tượng – tài xế chỉ cần biết đến những yếu tố quan trọng liên quan trực tiếp đến việc lái xe.



Tương tự, tính trừu tượng cho phép lập trình viên làm việc với những thông tin quan trọng mà không cần lo lắng về cách hoạt động bên trong của chúng. Bằng cách này nó giúp cải thiện chất lượng code và khả năng đọc của code.

Tính trừu tượng cũng phục vụ cho tính bảo mật. Bằng cách chỉ hiển thị một phần được chọn của dữ liệu và chỉ cho phép dữ liệu **truy cập thông qua class** và **chỉnh sửa thông qua phương thức** chúng ta bảo vệ dữ liệu khỏi bị lộ.

Lợi ích của tính trừu tượng:

- Tạo giao diện người dùng đơn giản và cao cấp.
- Ẩn giấu code phức tạp.
- Bảo mật.
- Bảo trì dễ dàng hơn.
- Câp nhật code mà không ảnh hưởng đến người dùng.

Tính ?a hình (Polymorphism)

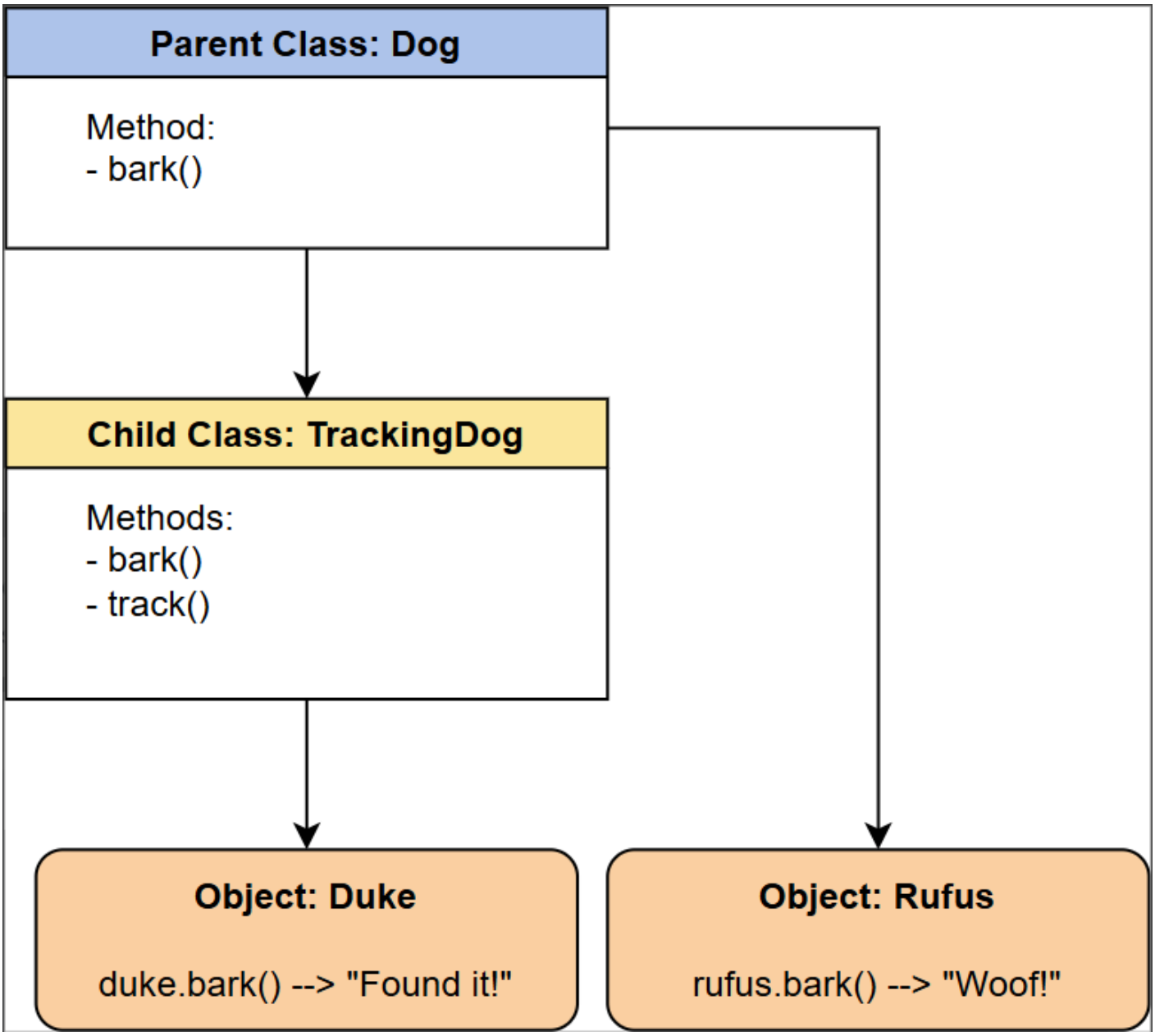
Tính đa hình là thiết kế đối tượng để để chia sẻ hành vi. Sử dụng tính kế thừa, đối tượng có thể ghi đè lên các hành vi mà lớp cha chia sẻ với lớp con bằng các hành vi đặc biệt khác. Tính đa hình cho phép những phương thức giống nhau thực hiện các hành vi khác nhau bằng 2 cách là overriding và overloading.

Ph??ng th?c ghi ?è (Override)

Trong runtime, đa hình sử dụng phương thức ghi đè. Trong phương thức ghi đè, một lớp con có thể triển khai khác với lớp cha của nó.

Trong ví dụ về chó, chúng ta muốn giống chó săn sửa khác với mặc định. Ta có thể tạo phương thức `bark()` trong lớp con và ghi đè phương thức `bark()` của lớp cha.

```
//Parent class Dog
class Dog{
    //Declare protected (private) fields
    _attendance = 0;

    constructor(namee, birthday) {
        this.name = name;
        this.birthday = birthday;
    }

    getAge() {
        //Getter
        return this.calcAge();
    }

    calcAge() {
        //calculate age using today's date and birthday
        return this.calcAge();
    }

    bark() {
        return console.log("Woof!");
    }

    updateAttendance() {
        //add a day to the dog's attendance days at the petsitters
        this._attendance++;
    }
}

//Child class TrackingDog, inherits from parent
class TrackingDog extends Dog {
    constructor(name, birthday)
        super(name);
        super(birthday);
    }

    track() {
        //additional method for TrackingDog child class
        return console.log("Searching...")
    }

    bark() {
        return console.log("Found it!");
    }


//instantiate a new TrackingDog object
const duke = new TrackingDog("Duke", "1/12/2019");
duke.bark(); //returns "Found it!"
```



Ph??ng th?c n?p ch?ng (Overload)

Trong compile time, đa hình sử dụng phương thức nạp chồng. Phương thức hoặc chức năng có cùng tên nhưng có thể khác số lượng tham số và khác kết quả trả về.

```
//Parent class Dog
class Dog{
    //Declare protected (private) fields
    _attendance = 0;

    constructor(namee, birthday) {
        this.name = name;
        this.birthday = birthday;
    }

    getAge() {
        //Getter
        return this.calcAge();
    }

    calcAge() {
```

```
        //calculate age using today's date and birthday
        return this.calcAge();
    }

    bark() {
        return console.log("Woof!");
    }

    updateAttendance() {
        //add a day to the dog's attendance days at the petsitters
        this._attendance++;
    }

    updateAttendance(x) {
        //adds multiple to the dog's attendance days at the petsitters
        this._attendance = this._attendance + x;
    }
}

//instantiate a new instance of Dog class, an individual dog named Rufus
const rufus = new Dog("Rufus", "2/1/2017");
rufus.updateAttendance(); //attendance = 1
rufus.updateAttendance(4); // attendance = 5
```

Trong ví dụ trên, nếu không có tham số truyền vào thì ngày chăm sóc mặc định cộng thêm 1, nếu có tham số thì số ngày chăm sóc tăng lên tương ứng với tham số được truyền vào.

Lợi ích của tính đa hình:

- Các đối tượng khác loại có thể chuyển qua một giao diện chung.
- Phương thức ghi đè
- Phương thức nạp chồng

Tổng kết

Lập trình hướng đối tượng yêu cầu lập trình viên nghĩ về cấu trúc của chương trình và lên kế hoạch thiết kế hướng đối tượng trước khi code. OOP trong lập trình tập trung vào cách phân tích yêu cầu thành các lớp đơn giản và có thể tái sử dụng như một bản thiết kế của đối tượng. Nhìn chung, OOP cho phép cấu trúc dữ liệu và tái sử dụng tốt hơn, tiết kiệm thời gian với những dự án dài.