**Những khái niệm cơ bản của lập trình hướng đối tượng**

1. **Đối tượng (Object)**

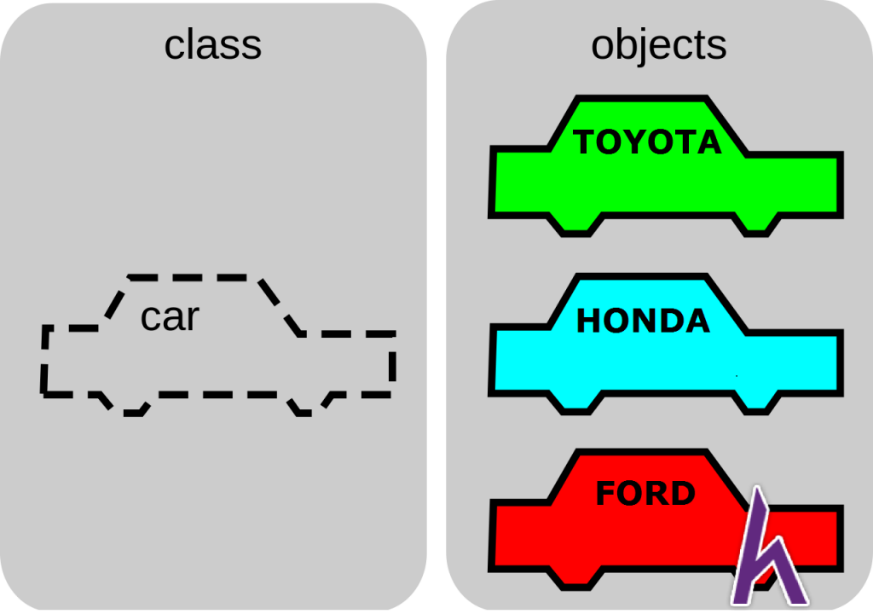
Đối tượng ở đây ta thể hiểu như khái niệm bên ngoài: Con người, Xe máy, Nhà cửa…

Trong một đối tượng sẽ bao gồm 2 thông tin: thuộc tính và phương thức.

* **Thuộc tính:** là những thông tin của đối tượng. Ví dụ: con người có họ tên, chiều cao, độ tuổi…
* **Phương thức:** là những thao tác, hành động mà đối tượng đó có thể thực hiện. Ví dụ: con người có những hành động ăn, ngủ, đi lại…

1. **Lớp (Class**)

Lớp chính là định nghĩa của đối tượng, ta sẽ xây dựng lớp để tạo ra những đối tượng khác nhau. Ví dụ như: Bạn **Nguyễn Văn A** và **Nguyễn Văn B** đều là **con người**, mà con người thì đều có tên, tuổi, chiều cao,.. tuy nhiên thông tin lại khác nhau. Như vậy **con người** chính là lớp, **Nguyễn Văn A** và **Nguyễn Văn B** là đối tượng.



* 1. **Sự khác nhau giữa lớp và đối tượng trong java**

| **Đối tượng** | **Lớp** |
| --- | --- |
| Đối tượng là thể hiện của 1 lớp. | Lớp là một khuân mẫu hay thiết kế để tạo ra các đối tượng. |
| Đối tượng là 1 thực thể trong thế giới thực như Con mèo (Cat), con chó (Dog), … | Lớp là một nhóm các đối tượng tương tự nhau. Ví dụ: Lớp động vật (Animal). |
| Đối tượng là 1 thực thể vật lý | Lớp là 1 thực thể logic |
| Đối tượng được tạo ra chủ yếu từ từ khóa **new**. Ví dụ: Student s1=new Student(); | Lớp được khai báo bằng việc sử dụng từ khóa **class**. Ví dụ: class Student{} |
| Đối tượng có thể được tạo nhiều lần. | Lớp được khai báo 1 lần duy nhất. |
| Đối tượng được cấp bộ nhớ khi nó được tạo ra. | Lớp không được cấp bộ nhớ khi nó được tạo ra. |

1. **Package**

Một package (gói) trong java là một nhóm các kiểu tương tự của các lớp, giao diện và các package con.

Package trong java có thể được phân loại theo hai hình thức, package được dựng sẵn và package do người dùng định nghĩa.

Có rất nhiều package được dựng sẵn như java, lang, net, io, util, sql, …

Lợi thế của việc sử dụng package trong java:

* Package được sử dụng để phân loại lớp và interface giúp dễ dàng bảo trì.
* Package cung cấp bảo vể truy cập
* Package khắc phục được việc đặt trùng tên

1. **Constructor**

Constructor trong java là một dạng đặc biệt của phương thức được sử dụng để khởi tạo các đối tượng.

Java Constructor được gọi tại thời điểm tạo đối tượng. Nó khởi tạo các giá trị để cung cấp dữ liệu cho các đối tượng, đó là lý do tại sao nó được gọi là constructor.

Khai báo của Constructor giống với khải báo của method (phương thức). Nó phải có **cùng tên với class** (lớp) và **không có giá trị trả về**.

Có 2 kiểu của constructor:

* Constructor mặc định (không có tham số truyền vào – **default constructors**).
* Constructor tham số (**parameterized constructors**).
  1. **Constructor mặc định trong java**

Một constructor mà không có tham số được gọi là constructor mặc định. Constructor mặc định được sử dụng để cung cấp các giá trị mặc định cho các thuộc tính như 0, null, false … , tùy thuộc vào kiểu dữ liệu được sử dụng.

Nếu một lớp không khai báo contructor thì trình biên dịch sẽ tự động tạo một constructor mặc định trong lớp đó.

* Từ khóa **this** trong java là một biến tham chiếu được sử dụng để tham chiếu tới đối tượng của **lớp hiện tại**.
* Từ khóa **super** trong java là một biến tham chiếu được sử dụng để tham chiếu trực tiếp đến đối tượng của **lớp cha gần nhất**.
  1. **Sự khác nhau giữa constructor và phương thức trong java**

|  |  |
| --- | --- |
| **Constructor** | **Phương thức** |
| Constructor được sử dụng để khởi tạo trạng thái của một đối tượng. | Phương thức được sử dụng để thể hiện hành động của một đối tượng. |
| Constructor không có kiểu trả về. | Phương thức có kiểu trả về. |
| Trình biên dịch Java tạo ra constructor mặc định nếu bạn không có constructor nào. | Phương thức không được tạo ra bởi trình biên dịch Java. |
| Tên của constructor phải giống tên lớp. | Tên phương thức có thể giống hoặc khác tên lớp. |

1. **Phạm vi truy cập (Access modifier)**

Có hai loại modifier trong java: **access modifiers** và **non-access modifiers**.

Các access modifiers trong java xác định độ truy cập (Phạm vi) vào dữ liệu của của các trường (field), phương thức (method), cấu tử (constructor) hoặc lớp (class).

Có 4 kiểu của java access modifiers:

* private
* default
* protected
* public

Và có một vài non-access modifiers chẳng hạn static, abstract, synchronized, native, volatile, transient, v.v.. Trong tài liệu này chúng ta sẽ học về access modifier.

Bảng mô tả tổng quan về cách sử dụng các access modifier:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **Truy cập bên trong class?** | **Truy cập bên trong package?** | **Truy cập bên ngoài package bởi class con?** | **Truy cập bên ngoài class và không thuộc class con?** |
| private | Y |  |  |  |
| Default | Y | Y |  |  |
| protected | Y | Y | Y |  |
| public | Y | Y | Y | Y |

1. **Overloading (Nạp chồng phương thức)**

Method overloading là một tính năng cho phép một class có nhiều method cùng tên nhưng khác danh sách các tham số đầu vào.

Chúng ta có 3 cách để implement method overloading

* Thay đổi số lượng tham số lượng tham số đầu vào
* Thay đổi kiểu dữ liệu của các tham số đầu vào
* Thay đổi trình tự kiểu dữ liệu của các tham số đầu vào

1. **Overriding (Ghi đề phương thức)**

Khi khai báo một method ở class con mà có sẵn ở class cha được gọi là method overriding. Overriding cho phép các class con tự đưa ra cách triển khai riêng cho một method đã được class cha cung cấp sẵn.

## Sự khác nhau giữa overloading và overriding

|  |  |  |
| --- | --- | --- |
| **No.** | **Nạp chồng phương thức (overloading)** | **Ghi đè phương thức (overriding)** |
| 1) | Nạp chồng phương thức được sử dụng để giúp code của chương trình *dễ đọc hơn*. | Ghi đè phương thức được sử dụng để cung cấp *cài đặt cụ thể* cho phương thức được khai báo ở lớp cha. |
| 2) | Nạp chồng được thực hiện bên *trong một class*. | Ghi đè phương thức xảy ra *trong 2 class* có quan hệ kế thừa. |
| 3) | Nạp chồng phương thức thì *tham số phải khác nhau*. | Ghi đè phương thức thì *tham số phải giống nhau*. |
| 4) | Nạp chồng phương thức là ví dụ về *đa hình lúc biên dịch*. | Ghi đè phương thức là ví dụ về *đa hình lúc runtime*. |
| 5) | Trong java, nạp chồng phương thức không thể được thực hiện khi chỉ thay đổi kiểu giá trị trả về của phương thức. Kiểu giá trị trả về có thể giống hoặc khác. *Giá trị trả về có thể giống hoặc khác*, nhưng tham số phải khác nhau. | Giá trị trả về phải giống nhau. |

1. **Kế thừa (Inheritance)**

**Kế thừa trong java** là sự liên quan giữa hai class với nhau, trong đó có class cha (superclass) và class con (subclass). Khi kế thừa class con được hưởng tất cả các phương thức và thuộc tính của class cha. Tuy nhiên, nó chỉ được truy cập các thành viên public và protected của class cha. Nó không được phép truy cập đến thành viên private của class cha.

1. **Đa hình**

Một đối tượng có thể biểu hiện bằng nhiều hình thức và các hành động khác nhau gọi là [tính đa hình](https://shareprogramming.net/da-hinhpolymorphism-trong-java/).

Như ở ví dụ trên, **Hình** là một đối tượng nó có thể biến hoá thành các hình tam giác, hình tròn, hình vuông…

1. **Trừu tượng**

[Trừu tượng](https://shareprogramming.net/truu-tuong-abstraction-trong-java/) cho phép ẩn đi quá trình thực thi bên trong và chỉ hiển thị những chức năng.

Ví dụ: Khi bạn dùng một chiếc điện thoại di dộng có thể gọi điện và nhắn tin và bạn dùng nó để gọi điện và nhắn tin cho bạn bè, thế nhưng bạn không biết được bên trong điện thoại nó làm những gì để thực hiện các yêu cầu gọi và nhắn tin cho chúng ta phải không nào.

1. **Đóng gói**

Trong lập trình hướng đối tượng việc liên kết dữ liệu và method lại với nhau và giữ chúng an toàn khỏi những tác động bên ngoài gọi là [tính đóng gói](https://shareprogramming.net/tinh-dong-goi-trong-java/).

Trong java một class có thể cấp quyền cho một thuộc tính hay một phương thức ra bên ngoài dựa vào các từ khoá:

* **public**: Cho phép truy cập ở mọi phạm vi
* **private**: Cho phép truy cập trong nội bộ class
* **protected**:  Chỉ cho phép các class kế thừa truy cập
* **default**: Chỉ cho phép truy cập nội bộ class và cùng package

## ****Từ khóa static****

Trong Java, từ khóa **static** được sử dụng để quản lý bộ nhớ tốt hơn và nó có thể được truy cập trực tiếp thông qua lớp mà không cần khởi tạo.

Từ khóa static thuộc về lớp chứ không thuộc về instance (thể hiện) của lớp.

Chúng ta có thể áp dụng từ khóa static với các biến, các phương thức, các khối, các lớp lồng nhau(nested class).

Các trường hợp sử dụng static:

1. **Biến static (biến class):** khi bạn khai báo một biến là static, thì biến đó được gọi là biến tĩnh, hay biến static.
2. **Phương thức static:** khi bạn khai báo một phương thức là static, thì phương thức đó gọi là phương thức static.
3. **Lớp static (static class)**: một class được có thể được đặt là static chỉ khi nó là một nested class. Một nested static class có thể được truy cập mà không cần một object của outer class (lớp bên ngoài).
   1. **Một số câu hỏi thường gặp khi đi phỏng vấn liên quan đến từ khóa static**
4. **Ý nghĩa của từ khoá static trong Java là gì? Chúng ta có thể override (đè) một hàm private hoặc static trong Java không?**

Từ khoá static biểu thị cho biến hoặc phương thức có thể được truy cập (sử dụng) mà không cần tạo ra thực thể của lớp chứa nó. Người dùng không thể override phương thức static trong Java, bởi vì kỹ thuật đè (overriding) phương thức được dựa trên quá trình gán (binding) động khi runtime (khi chương trình đang chạy) và những phương thức static  được gán tĩnh trong thời gian biên dịch. Phương thức tĩnh không ràng buộc với thực thể của lớp nên phương thức tĩnh sẽ không thể override.

1. **Chúng ta có thể truy cập một biến không tĩnh(non-static) trong một ngữ cảnh static được không?**

Một biến static phụ thuộc vào lớp của nó và giá trị của nó sẽ tồn tại (giữ) cho tất cả các thực thể của lớp đó. Biến static được tạo ra khi lớp chứa đó được tải (load) bởi JVM. Nếu cố gắng truy cập vào một biến non-static (trong hàm static) mà không có trong thực thể (instance) nào thì trình biên dịch sẽ báo lỗi, bởi vì những biến đó (non-static) chưa được khởi tạo và chúng không có ràng buộc với bất kỳ thực thể nào.

1. **Tại sao phương thức main trong Java là static?**

Trả lời: Bởi vì không cần thiết phải tạo đối tượng để gọi phương thức static. Nếu nó là phương thức non-static, JVM đầu tiên tạo đối tượng và sau đó gọi phương thức main() mà có thể gây ra vấn đề về cấp phát bộ nhớ bộ nhớ phụ.

## ****Từ khóa final****

Từ khóa final trong Java được sử dụng để hạn chế thao tác của người dùng.

Các trường hợp sử dụng:

* **Biến final**: khi một biến được khai báo với từ khoá final, nó chỉ chứa một giá trị duy nhất trong toàn bộ chương trình (hay dễ hiểu hơn gọi là **biến hằng**).
* **Phương thức final**: khi một phương thức được khai báo với từ khoá final, các class con kế thừa sẽ không thể ghi đè (override) phương thức này.
* **Lớp final**: khi từ khoá final sử dụng cho một lớp, lớp này sẽ không thể được kế thừa.
* **Biến static final trống**: Một biến final mà không được khởi tạo tại thời điểm khai báo được gọi là biến final trống.
  1. **Một số câu hỏi thường gặp khi đi phỏng vấn liên quan đến từ khóa final**
* Phương thức final có được kế thừa không?

Trả lời: **Có**, phương thức final được kế thừa nhưng bạn không thể ghi đè nó.

* Biến final trống hoặc không được khởi tạo là gì?

Trả lời: Một biến final mà không được khởi tạo tại thời điểm khai báo được gọi là biến final trống. Biến được khởi tạo tại thời điểm tạo đối tượng và một khi nó đã được khởi tạo thì không thể bị thay đổi.

* Tham số final là gì?

Nếu bạn khai báo bất cứ tham số nào là final, thì bạn không thể thay đổi giá trị của nó.

## ****Từ khóa super**** Từ khóa super trong Java là một biến tham chiếu, được sử dụng để tham chiếu trực tiếp đến đối tượng của lớp cha gần nhất. Bất cứ khi nào bạn tạo ra một thể hiện (hay còn gọi là một instance) của lớp con, một instance của lớp cha được tạo ra ngầm định, nghĩa là được tham chiếu bởi biến super.

* 1. **Khi nào nên sử dụng từ khóa super trong Java?**

Qua phần kiến thức và ví dụ mình đã trình bày ở trên, chúng ta có thể rút ra được một số trường hợp nên sử dụng super như sau:

* Sử dụng từ khóa super khi cần truy xuất đến thuộc tính của lớp cha gần nhất. Nó được sử dụng nếu lớp cha và lớp con có cùng tên thuộc tính.
* Sử dụng từ khóa super khi cần truy xuất đến phương thức của lớp cha gần nhất. Nó nên được sử dụng nếu lớp con chứa cùng phương thức với lớp cha. Nói cách khác, nó được sử dụng nếu phương thức bị override (ghi đè).
* Sử dụng từ khóa super khi cần gọi đến constructor của lớp cha gần nhất.

1. **Abstract class**

[**Abstract class**](https://shareprogramming.net/abstract-class-trong-java-voi-vi-du-cu-the/) được sử dụng để triển khai tính trừu tượng trong java.

Một abstract class được khai báo bằng từ khoá abstract.

Một abstract method là một method được khai báo không có thân hàm.

Các class con thừa kế abstract class phải implement abstract method.

Abstract class không thể khởi tạo .

Đơn giản hoá đối tượng bằng cách đưa ra các thuộc tính và phương thức mà nó có và thực hiện.

1. **Interface**

Interface được sử dụng để biểu diễn 100% tính trừu tượng. Trong một interface các method sẽ mặc định là abstract method. Nói đến interface chúng ta sẽ nhìn nhận nó như là một khuôn mẫu các hành động được cung cấp mà bạn không thể biết bên trong nó thực hiện như thế nào.

Các method trong interface mặc định đều là asbtract method.

Trước mỗi method trong interface đều mặc định được thêm vào trước cặp từ khoá **public static.**

Các class implement interface phải implement tất cả các method được khai báo trong interface.

Các class trong java không được extends nhiều class, nhưng với có thể implement nhiều interface khác nhau.

# **Sự khác nhau giữa abstract class và interface**

|  |  |  |
| --- | --- | --- |
| **STT** | **Abstract class** | **Interface** |
| 1 | Abstract class chỉ có thể thừa kế một class hoặc một abstract class khác. | Interface có thể thừa kế một hoặc nhiều interface khác. |
| 2 | Abstract class có thể chứa abstract method và method thông thường(method có thân hàm). | Interface chỉ có abstract method. |
| 3 | Abstract class có thể chứa protected hoặc public abstract method. | Interface chỉ có public abstract method. |
| 4 | Abstract class có thể chứa static, final hoặc static final biến. | Interface chỉ có public static final biến. |