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

1. **Đối tượng (Object)**

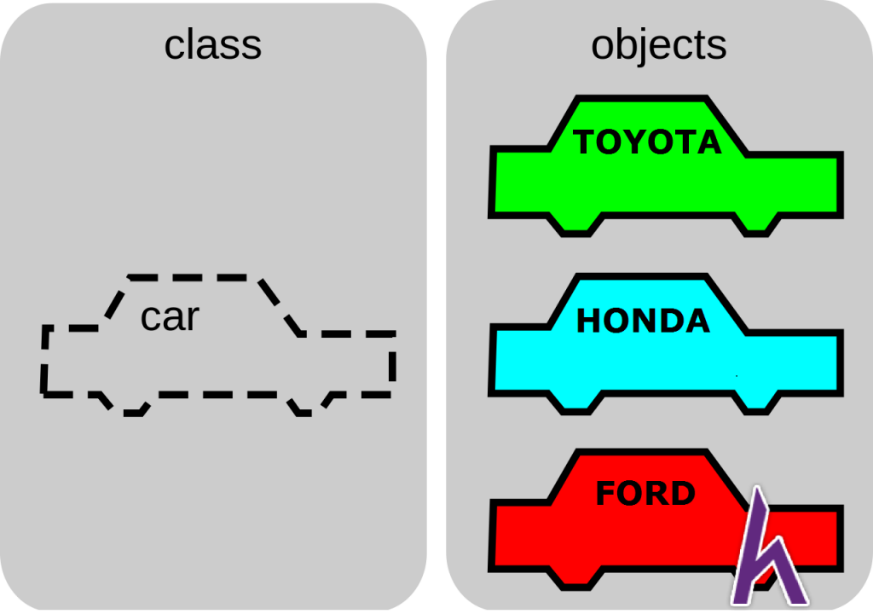
Đối tượng ở đây ta thể hiểu như khái niệm bên ngoài: Con người, Xe máy, Nhà cửa…

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.



* **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. **Phương thức khởi tạo (Constructor)**

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

Khai báo của Constructor giống với khai 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**).
* **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.

* **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)**

**Phạm vi truy cập** (access modifiers) là xác định độ truy cập phạm vi vào dữ liệu của các thuộc tính, phương thức hoặc class.

**Package** (gói) là nhóm các **class, interface** hoặc các **package con** liên quan lại với nhau. Mục đích của package ngăn cản xung đột đặt tên, điều kiện truy cập, thuận tiện tìm kiếm và lưu trữ.

* **Có 4 loại phạm vi truy cập:**
* private
* default
* protected
* public

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. **Từ khóa static**

Từ khóa **static** được sử dụng chỉnh để quản lý bộ nhớ. Chúng ta có thể áp dụng từ khóa static với các biến, phương thức

Khi gọi các biến, phương thức chứa từ khóa static chúng ta chỉ cần gọi bằng cách **<tên class>.<tên biến static>** thay vì new một đối tượng như biến instance

Từ khóa static thường được sử dụng cho thuộc tính chung của tất cả đối tượng

**Ví dụ:** Các học sinh cùng trường học có thuộc tính chung là tên trường học giống nhau. Nếu chúng ta tạo nó là static, thì trường này sẽ chỉ sử dụng bộ nhớ một lần để lưu biến này.

* **Một số câu hỏi thường gặp khi đi phỏng vấn liên quan đến từ khóa static**

**Câu hỏi 1: Ý nghĩa của từ khoá static trong Java là gì? Chúng ta có thể override (đè) một phương thức static trong Java không?**

Từ khoá static biểu thị cho biến hoặc phương thức có thể được truy cập 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) thì 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.

**Câu hỏi 2: 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.

**Câu hỏi 3: 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.

1. **Từ khóa this**

Từ khóa this là một biến tham chiếu được sử dụng để tham chiếu tới đối tượng của lớp hiện tại

**Ví dụ**: Bạn Tuấn là một người siêng năng cần cù. Ngoài ra bạn ấy (this) còn rất thông minh

* **Tham chiếu tới biến instance của lớp hiện tại.**

Từ khóa this trong java có thể được dùng để tham chiếu tới biến instance của lớp hiện tại.

Nếu có sự trùng tên nhau giữa biến toàn cục và tham số khiến bạn bị phân vân. Từ khóa this sẽ giúp bạn giải quyết sự phân vân của bạn.

* **Sử dụng this() gọi Constructor của lớp hiện tại.**

Phương thức this() có thể được sử dụng để gọi Constructor của lớp hiện tại. Cách sử dụng này sẽ hữu dụng hơn nếu bạn có nhiều Constructor trong một lớp và bạn muốn sử dụng lại Constructor.

* **Gọi phương thức của lớp hiện tại.**

Bạn có thể sử dụng từ khóa this để gọi phương thức của lớp hiện tại. Nếu bạn không sử dụng từ khóa this, trình biên dịch sẽ tự động thêm từ khóa this cho việc gọi phương thức

## ****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.

* **Một số câu hỏi thường gặp khi đi phỏng vấn liên quan đến từ khóa final**

**Câu hỏi 1: Phương thức final có được kế thừa không?**

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

**Câu hỏi 2: 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.

**Câu hỏi 3: 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.

1. **Tính kế thừa**

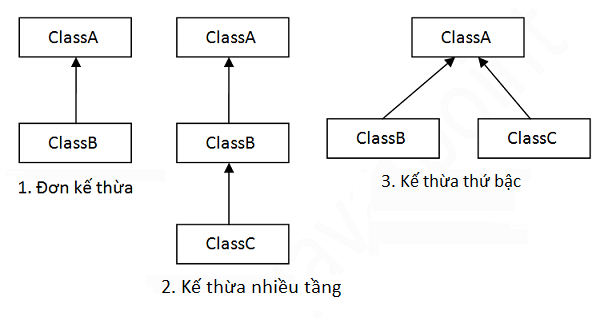
Kế thừa là một class thừa hưởng lại tất cả các phương thức và thuộc tính từ class khác

Slogan đặc trưng của kế thừa: "Cha có thì con có, con có chưa chắc cha đã có"

* **Các kiểu kế thừa trong java**

Có 3 kiểu kế thừa trong java đó là đơn kế thừa, kế thừa nhiều cấp, kế thừa thứ bậc.

Khi một class được kế thừa từ nhiều class đươc gọi là đa kế thừa. Trong java, đa kế thừa chỉ được support thông qua interface, như đã được nói đến trong bài [interface trong java](https://viettuts.vn/java/interface-trong-java)



## ****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.

* **Trong java, từ khóa super có 3 cách sử dụng như sau:**
* Từ khóa super được sử dụng để tham chiếu trực tiếp đến biến instance của lớp cha gần nhất.
* super() được sử dụng để gọi trực tiếp Constructor của lớp cha.
* Từ khóa super được sử dụng để gọi trực tiếp phương thức của lớp cha.

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

**Overloading** là nhiều phương thức trong một lớp có cùng tên nhưng khác nhau về kiểu dữ liệu hoặc số lượng các tham số.

* **Có 2 cách Overloading trong java:**
* Thay đổi số lượng tham số truyền vào
* Thay đổi kiểu dữ liệu của các tham số
* **Các câu hỏi về nạp chồng phương thức trong java**

**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.

**Câu hỏi 2: Có thể nạp chồng phương thức main() không?**

Có, bạn có thể nạp chồng n phương thức main. Nhưng JVM chỉ gọi phương thức main() có tham số truyền vào là một mảng String.

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

**Overriding** (ghi đè) có nghĩa là có 2 phương thức giống nhau về tên và tham số truyền vào. Một phương thức ở lớp cha, còn cái kia ở lớp con. Overriding cho phép lớp con có thể thực hiện riêng biệt cho phương thức mà lớp cha đã cung cấp.

Nếu không muốn lớp con có thể **Overriding** lại phương thức nào đó, ta sẽ sử dụng từ khóa **final**

* **Sử dụng ghi đè phương thức trong java**
* Ghi đè phương thức được sử dụng để cung cấp cài đặt đặc biệt của một phương thức mà đã được định nghĩa ở lớp cha.
* Ghi đè phương thức được sử dụng cho đa hình runtime.
* **Các nguyên tắc ghi đè phương thức trong java**
* Phương thức phải có tên giống với lớp cha.
* Phương thức phải có tham số giống với lớp cha.
* Lớp con và lớp cha có mối quan hệ kế thừa.
* **Câu hỏi về ghi đè phương thức trong java**

**Câu hỏi 1: Có ghi đè được phương thức static không?**

Không, phương thức static không thể ghi đè được, bằng chứng là đa hình runtime, vấn đề này sẽ được học trong bài sau.

**Câu hỏi 2: 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.

**Câu hỏi 3: Có ghi đè phương thức main được không?**

Không, vì main là phương thức static.

## 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. **Tính đa hình**

**Tính đa hình** là hiện tượng các đối tượng thuộc các lớp khác nhau có thể hiểu cùng 1 thông điệp theo các cách khác nhau.

Một ví dụ về đa hình trong thực tế. Ta có 3 con vật: chó, mèo, lợn. Cả 3 con vật này đều là động vật. Nhưng khi ta bảo cả 3 động vật kêu thì con chó sẽ kêu gâu gâu, con mèo sẽ kêu meo meo và con heo sẽ kêu ẹt ẹt.

Trong ví dụ trên 3 con vật: chó, mèo, lợn xem như là các đối tượng. Việc ta bảo 3 động vật kêu chính là thông điệp. Rõ ràng cả 3 con vật có thể hiểu cùng 1 thông điệp là kêu theo các cách khác nhau.

* **Để thể hiện được tính đa hình:**
* Các lớp phải có quan hệ kế thừa với cùng 1 lớp cha nào đó.
* Phương thức đa hình phải được ghi đè (override) ở các lớp con.

1. **Tính trừu tượng**

Nếu ai còn nhớ, đây chính là que tính được sử dụng khi học toán lớp 1. Việc học toán lúc đầu cho các em nhỏ cần phải hiểu về các con số và toán tử, có thể giải thích quy trình như sau:

* **Bước 1:**Cho các em nhỏ tập đếm dựa theo que tính để làm quen với con số, để cho tụi nhỏ hiểu ý nghĩa các con số như số 1 là một cái, số 2 là hai cái,…
* **Bước 2:** Khi làm quen được các con số, các thầy cô sẽ dạy cách tính toán: thường các cô để 1 cái bên trái, 2 cái bên phải và hỏi tổng là bao nhiều. Các bạn đừng có nghĩ đa số tụi nhó sẽ biết 1 + 2 = 3 (tùy theo khả năng mỗi đứa). Chúng sẽ đếm bên trái xong rồi đếm bên phải, dần dần trong đầu sẽ trừu tượng ra phép cộng. Rồi thầy cô thử lấy ra một vài que tính, tụi nhóc cũng phải tập đếm lại và cũng trừu tượng ra phép trừ…

Như vậy, việc giáo dục toán que tính đang tạo trừu tượng toán học cho học sinh, dần dần các khái niệm trừu trượng của toán học cao hơn sẽ được dạy dần như nhân chia, lũy thừa, căn bậc, đạo hàm…

## Tính trừu tượng trong lập trình hướng đối tượng là gì?

**Tính trừu tượng trong lập trình hướng đối tượng** là chỉ nêu ra vấn đề mà không hiển thị cụ thể, chỉ hiện thị tính năng thiết yếu đối với đối tượng người dùng mà không nói quy trình hoạt động. Ví dụ: như tạo ra tính năng gửi tin nhắn, ta chỉ cần hiểu là người dùng viết tin rồi nhấn gửi đi. Còn quy trình xử lý tin nhắn gửi như thế nào thì ta chưa đề cập đến.

Như vậy, tính trừu tượng là che giấu thông tin thực hiện từ người dùng, họ chỉ biết tính năng được cung cấp: Chỉ biết thông tin đối tượng thay vì cách nó sử dụng như thế nào. Nó có những ưu điểm sau:

* Cho phép lập trình viên bỏ qua những phức tạp trong đối tượng mà chỉ đưa ra những khái niệm phương thức và thuộc tính cần thiết. Ta sẽ dựa những khái niệm đó để viết ra, nâng cấp và bảo trì.
* Nó giúp ta tập trung cái cốt lõi đối tượng. Giúp người dùng không quên bản chất đối tượng đó làm gì.
* **Tính trừu tượng trong Java**

**Lớp trừu tượng (Abstract class)**

**Lớp trừu tượng** là lớp được khai báo mà không thể tạo ra đối tượng từ lớp đó. Ta sẽ tạo những lớp con kế thừa lớp trừu tượng.

Mục đích lớp trừu tượng là tạo ra lớp chung cho những lớp có liên quan với nhau kế thừa. Ví dụ khi xây dựng phần mềm quản lý nhà trường:  Những lớp sinh viên, giảng viên, cán bộ,… có những thuộc tính và phương thức chung như tên, năm sinh, quê quán,… thì ta sẽ tạo một lớp con người là lớp trừu tượng và những đặc điểm chung được để trong lớp con người. Khi phát triển chương trình, ta chỉ có thể tạo các đối tượng từ lớp con kế thừa lớp con người; không thể cho tạo đối tượng từ lớp con người được.

**Phương thức trừu tượng**

**Các phương thức trừu tượng** là là chỉ định nghĩa mà không có chương trình bên trong, lớp con kế thừa phải bắt buộc override nó lại để sử dụng. Phương thức trừu tượng có ý nghĩa định nghĩa phương thức bắt buộc phải có trong lớp con kế thừa.

1. **Đóng gói**

**Tính đóng gói** nhằm bảo vệ đối tượng không bị truy cập từ code bên ngoài vào để thay đổi giá trị các thuộc tính hay có thể truy cập trực tiếp. Việc cho phép truy cập các giá trị của đối tượng tùy theo sự đồng ý của người viết ra lớp của đối tượng đó. Tính chất này đảm bảo sự bảo mật, toàn vẹn của đối tượng trong Java.

Tính chất này được thể hiện qua:

* [Phạm vi truy cập](https://howkteam.vn/redirect?Id=CDPvobsj8EgqSWOjQulkj7KLp%2fXKC0Hu8VwCLICnWtTCf0%2b1w1e8nqgbLi99hbzBA3u1VPG44GP7i4yN8IorEGLa53iYj7PYleBTZpFOR6yQ%2fGzSC09l2ajm%2bnB92AapP0cbjUaASWnj8VLfiy%2fZuyjt6%2brtH5BEGvWc%2fv7yNOqnsIbqR23%2fmm0eZCie1J25)
* [Setter và Getter](https://howkteam.vn/redirect?Id=CDPvobsj8EgqSWOjQulkj7KLp%2fXKC0Hu8VwCLICnWtTCf0%2b1w1e8nqgbLi99hbzBA3u1VPG44GP7i4yN8IorEJd%2f%2fu23jrr%2b9gS3XvxW3eGzB84eVeBEVXin7Ov1tgwFhw0IIB7jvulHmRzxJyzW8bFTpVOupv7sRZoaW7SDYXHKTk8fTj8pM5Fl%2bLKwC0r%2b)

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.

Các class con thừa kế **abstract class** phải dùng từ khóa **extends**.

Abstract class không thể khởi tạo bằng từ khóa **new**.

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

1. **Interface**

**Interface** được sử dụng để biểu diễn 100% tính trừu tượng.

**Interface** có những đặc điểm sau:

* Interface luôn luôn có modifier là: public interface, cho dù bạn có khai báo rõ hay không.
* Nếu có các trường (field) thì chúng đều là: public static final, cho dù bạn có khai báo rõ hay không.
* Các method của nó đều là method trừu tượng, nghĩa là không có thân hàm, và đều có modifier là: public abstract, cho dù bạn có khai báo hay không.
* Interface không có Constructor
* Các class trong java không được extends nhiều class, nhưng với có thể implement nhiều interface khác nhau.
* Các class implement interface phải implement tất cả các method được khai báo trong interface.

**Một interface tương tự với một class bởi những điểm sau đây:**

* Một interface được viết trong một file với định dạng .java, với tên của interface giống tên của file.
* Bytecode của interface được lưu trong file có định dạng .class.
* Khai báo interface trong một package, những file bytecode tương ứng cũng có cấu trúc thư mục có cùng tên package.

**Một interface khác với một class ở một số điểm sau đây:**

* Bạn không thể khởi tạo một interface.
* Một interface không chứa bất cứ hàm Contructor nào.
* Tất cả các phương thức của interface đều là abstract.
* Một interface không thể chứa một trường nào trừ các trường vừa static và final.
* Một interface không thể kế thừa từ lớp, nó được triển khai bởi một lớp.
* Một interface có thể kế thừa từ nhiều interface khác.

# **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. |