# GIỚI THIỆU KHÓA HỌC

## GIỚI THIỆU JAVA

### Ngôn ngữ lập trình Java

Java là một ngôn ngữ lập trình theo phương pháp hướng đối tượng được sử dụng phổ biến hiện nay. Hầu như các công ty phần mềm trong nước và trên thế giới đề sử dụng Java như một ngôn ngữ chủ đạo cho việc phát triển sản phẩm chiến lược. Ngôn ngữ Java cho phép xây dựng dễ dàng và rất hiệu quả nhiều loại ứng dụng từ xử lý đơn giản cho đến ứng dụng trên thiết bị di động, … Bởi vậy mà hiện nay gần như các trường đại học, cao đẳng, các trường trung cấp chuyên nghiệp hay các trung tâm đào tạo lập trình để đưa ngôn ngữ Java vào làm một trong những môn giảng dạy chính.

Nhằm giúp cho các bạn tiếp xúc gần hơn với ngôn ngữ lập trình Java dễ dàng và thuận tiện hơn, tài liệu “**Lập trình Java từ cơ bản đến nâng cao**” đến cho bạn đọc cùng nội dung như sau.

### Nội dung cụ thể

* **Phần 1:** Giới thiệu tổng quan khóa học
* **Phần 2:** Giới thiệu ngôn ngữ lập trình Java
* **Phần 3:** Nền tảng lập trình với ngôn ngữ Java
* **Phần 4:** Xử lý biệt lệ
* **Phần 5:** Các thư viện thường dùng
* **Phần 6:** Xử lý chuỗi
* **Phần 7:** Xử lý mảng
* **Phần 8:** Xử lý Collection
* **Phần 9:** Lập trình hướng đối tượng trong Java
* **Phần 10:** Xử lý đọc ghi file

## TRA CỨU TÀI LIỆU

### T**ài liệu**

### Bài Tập

### Tài liệu tham khảo

# GIỚI THIỆU NGÔN NGỮ LẬP TRÌNH JAVA

## GIỚI THIỆU SƠ LƯỢC VỀ JAVA

Java là ngôn ngữ lập trình máy tính có tính chất hướng đối tượng, dựa trên các lớp, thường được sử dụng cho các hệ thống có tính độc lập cao. Nó được sử dụng để hướng tới các lập trình viên viết ứng dụng "write one, run everywhere" (viết một lần, chạy mọi nơi, nghĩa là đoạn code Java sau khi được biên dịch có thể chạy được trên tất cả các nền tảng hỗ trợ Java mà không cần phải được biên dịch lại. Các ứng dụng Java sau khi đã được biên dịch thành bytecode có thể chạy trên bất kỳ máy ảo Java nào (Java virtual machine)

Cho đến năm 2018, Java là một trong những ngôn ngữ được dùng phổ biến nhất trên thế giới, đặc biệt cho các úng dựng web client- server. Theo thống kê trên thế giới có khoảng 9 triệu lập trình viên Java.

### Lịch sử Java

Cuối năm 1990, James Gosling và các cộng sự được công ty Sun Microsystems giao nhiệm vụ xây dựng phần mềm lập trình cho các mặt hàng điện tử dân dụng nhằm mục đích cài chương trình vào các bộ xử lý của các thiết bị như VCR, lò nướng, PDA (personal data assistant).  
Lúc đầu Gosling và các cộng sự định chọn ngôn ngữ C++ nhưng thấy rằng C++ có những hạn chế. Chương trình viết bằng C++ khi chuyển sang chạy trên một số hệ thống máy có bộ vi xử lý khác thì đòi hỏi phải biên dịch lại. Gosling, Mike Sheridan và Patrick Naughton khởi xướng dự án ngôn ngữ Java vào tháng 6 năm 1991, ngôn ngữ mới dựa trên nền tảng ngôn ngữ C và C++ và đặt tên là Oak (cây sồi, vì phòng làm việc của Gosling nhìn ra một cây sồi).

**Tại sao lại có tên là Oak?**

**1.** Oak là cây sồi, vì phòng làm việc của Gosling nhìn ra một cây sồi.

**2.** Năm 1995, Oak được đổi tên thành Java.

**Tại sao lại đổi tên thành Java?**

**1.** Team muốn thu thập để chọn lựa ra một tên mới. Các từ bao gồm dynamic, revolutionary, Silk, jolt, DNA, … Họ muốn cái gì đó mà phản ánh đúng bản chất của công nghệ, đó là: một cuộc cách mạng, có tính động cao, duy nhất, đánh vần dễ dàng, …

Theo James Gosling thì Java là một trong các lựa chọn hàng đầu cùng với Silk. Tuy nhiên, vì Java có tính duy nhất hơn, nên hầu như tất cả thành viên team đều lựa chọn Java.

**2.** Java là một hòn đảo ở Indonesia, ở nơi này sản phẩm coffee đầu tiên được sản xuất (gọi là java coffee).

**3.** Bạn nên nhớ rằng Java chỉ là một tên chứ không phải là tên lược danh.

**4.** Java được phát triển đầu tiên bởi Jamas Gosling tại Sun Microsystems (bây giờ là công ty con của Oracle Coporation) và được công bố năm 1995.

**5.** Năm 1995, tạp chí Time bình chọn Java là một trong 10 sản phẩm tốt nhất năm 1995.

**6.** JDK 1.0 được công bố vào 23/1/1996.

### Một số đặc tính của Java

Java có rất nhiều đặc điểm và tính năng nổi bật. Dưới đây là một số đặc điểm của Java mà bạn có thể dễ dàng biết và tìm hiểu:

* **Đơn giản**

Bạn có thể hiểu rằng, đặc điểm đơn giản của Java là do:

*Thứ nhất*, cú pháp của nó dựa trên C++ (vì thế nó khá dễ dàng cho các sinh viên sau khi đã học C++ tại năm đầu đại học).

*Thứ hai*, gõ bỏ nhiều đặc điểm gây bối rối và hiếm khi được sử dụng chẳng hạn như các con trỏ tường minh, nạp chồng toán tử, …

*Và cuối cùng, bạn không cần xóa các đối tượng mà không được tham chiếu, bởi vì hãy để những thứ đó cho Bộ dọn rác tự động (Garbage Collection) trong Java.*

* Hướng đối tượng

Hướng đối tượng nghĩa là chúng ta tổ chức phần mềm dưới dạng một sự kết hợp của nhiều loại đối tượng khác nhau mà kết hợp chặt chẽ cả về dữ liệu lẫn hành vi của chúng.

Lập trình hướng đối tượng (OOP) là một phương pháp làm đơn giản hóa việc phát triển và duy trì phần mềm bằng việc cung cấp một số qui tắc.

Một số khái niệm cơ bản của hướng đối tượng (OOP) là:

* Đối tượng (Object)
* Lớp (Class)
* Tính kế thừa
* Tính đa hình
* Tính trừu tượng
* Tính bao đóng

Độc lập nền tảng

Một Platform là môi trường phần cứng hoặc phần mềm trong đó một chương trình chạy. Có hai loại Platform: một loại dựa trên phần mềm (software-based) và một loại dựa trên phần cứng (hardware-based). Java cung cấp software-based platform. Java Platform khác với nhiều nền tảng khác ở chỗ nó chạy ở trên các nền tảng hardware-based khác. Nó có hai thành phần:

* Runtime Environment
* API (Application Programming Interface)

Java code có thể chạy trên nhiều nền tảng như Windows, Linux, Sun Solaris, Mac/OS, … Java code được biên dịch bởi Bộ biên dịch Compiler và được chuyển đổi thành Bytecode. Bytecode này là một code độc lập nền tảng bởi vì nó có thể chạy trên nhiều nền tảng khác nhau. Với lý do này, đến đây bạn có thể hiểu được tại sao khi nói về Java, người ta thường nói đến khẩu hiệu: **Viết một lần, Chạy khắp nơi (Write Once and Run Anywhere)**.

### Bảo mật

Java là an toàn bởi vì:

* Không có con trỏ tường minh.
* Chương trình chạy bên trong các hộp thiết bị ảo.
* **Classloader**: Thêm sự bảo vệ bằng việc phân biệt riêng rẽ các package cho các lớp của hệ thống local file mà từ đó chúng được import với các file từ nguồn mạng.
* **Bytecode Vertifier**: Kiểm tra các đoạn code để tìm ra các phần code không hợp lệ mà có thể truy cập trái phép tới các đối tượng.
* **Security Manager**: Quyết định xem nguồn resource nào mà một lớp có thể truy cập chẳng hạn như đọc và ghi tới local disk.

Những tính năng bảo mật này được cung cấp bởi Ngôn ngữ Java. Ngoài ra, lập trình viên còn có thể cung cấp một số tính năng bảo mật khác thông qua SSL, JAAS, …

### Robust

Bạn có thể hiểu đơn giản Robust nghĩa là mạnh mẽ. Java sử dụng hệ quản trị bộ nhớ mạnh mẽ. Đó là, Java sử dụng ít con trỏ hơn để tránh các vấn đề liên quan tới bảo mật. Bên cạnh đó còn có Trình dọn rác tự động (Garbage Collection) trong Java. Đó là Xử lý ngoại lệ (Exception Handling) và kiểm tra kiểu. Tất cả những thứ này là cho Java là Robust.

### Độc lập cấu trúc

Đó là không có đặc điểm nào mà phụ thuộc vào trình triển khai, ví dụ như kích cỡ của kiểu dữ liệu gốc đã được thiết lập.

### Portable

Chúng ta có thể mang Java Bytecode tới bất cứ nền tảng nào.

### Hiệu suất cao

Với việc sử dụng Just-In-Time compilers, Java giúp nâng cao hiệu năng, giúp việc debug được dễ dàng cũng như nhanh chóng phát hiện lỗi.

### Phân tán (Distributed)

Chúng ta có thể tạo các ứng dụng phân tán trong Java. RMI và EJB được sử dụng để tạo các ứng dụng này. Chúng ta có thể truy cập các file bằng việc gọi các phương thức từ bất cứ thiết bị nào trên internet.

### Đa luồng (Multi-thread)

Một Thread là giống như một chương trình riêng rẽ, thực thi một cách đồng thời. Chúng ta có thể viết các chương trình Java mà xử lý nhiều tác vụ cùng một lúc bằng việc định nghĩa nhiều Thread. Lợi thế chính của Multi-thread là nó chia sẻ cùng bộ nhớ. Các Thread là quan trọng cho Multi-media, Web App, …

### Nơi Java được sử dụng?

Có rất nhiều thiết bị hiện tại đang sử dụng Java. Bao gồm:

* Desktop App như media player, antivirus, reader, …
* Web App như irctc.co.in, javatpoint.com, …
* Enterprise App như các ứng dụng về xử lý nghiệp vụ ngân hàng, …
* Trên các thiết bị Mobile.

### Các loại Java App

Có 4 loại ứng dụng chính mà có thể được tạo bởi sử dụng ngôn ngữ lập trình Java:

#### Standalone App

Nó còn được biết đến với tên gọi khác là Destop App hoặc Windows-based App. Một ứng dụng mà chúng ta cần cài đặt trên mỗi thiết bị như media player, antivirus, … AWT và Swing được sử dụng trong Java để tạo các Standalone App.

#### Web App

Một ứng dụng mà chạy trên Server Side và tạo Dynamic Page, được gọi là Web App. Hiện tại, các công nghệ Servlet, JSP, Struts, JSF, … được sử dụng để tạo Web App trong Java.

#### Enterprise App

Một ứng dụng dạng như Banking App, có lợi thế là tính bảo mật cao, cân bằng tải (load balancing) và clustering. Trong java, EJB được sử dụng để tạo các Enterprise App.

#### Mobile App

Đây là loại ứng dụng được tạo cho thiết bị mobile. Hiện tại thì Android và Java ME được sử dụng để tạo loại ứng dụng này.

### Lịch sử Java

Java bắt đầu từ Green Team. Họ đề xướng một cuộc cách mạng để phát triển một ngôn ngữ cho các thiết bị digital như set-top box, television, ….

Hiện tại, Java được sử dụng trong lập trình mạng, các thiết bị mobile, game, các giải pháp về e-business, … Dưới đây là một số điểm chính miêu tả sơ lược về Lịch sử Java.

**1.** James Gosling, Mike Sheridan và Patrick Naughton khởi xướng dự án về ngôn ngữ Java trong tháng 6/1991. Họ được gọi là Green Team.

**2.** Thiết kế đầu tiên là cho các hệ thống nhỏ, có thể nhúng vào trong các thiết bị điện tử như set-top box.

**3.** Ban đầu, nó được gọi là Greentalk bởi James Gosling và với đuôi là .gt.

**4.** Sau đó, nó được gọi là Oak và được phát triển như là một phần của Green Project.

**Tại sao lại có tên là Oak?**

**5.** Oak là loại cây sồi khỏe mạnh, sống nhiều ở các quốc gia như Mỹ, Pháp, Đức, … Bên ngoài khu làm việc của Green Team là các cây sồi tươi tốt quanh năm.

**6.** Năm 1995, Oak được đổi tên thành Java.

**Tại sao lại đổi tên thành Java?**

**7.** Team muốn thu thập để chọn lựa ra một tên mới. Các từ bao gồm dynamic, revolutionary, Silk, jolt, DNA, … Họ muốn cái gì đó mà phản ánh đúng bản chất của công nghệ, đó là: một cuộc cách mạng, có tính động cao, duy nhất, đánh vần dễ dàng, …

Theo James Gosling thì Java là một trong các lựa chọn hàng đầu cùng với Silk. Tuy nhiên, vì Java có tính duy nhất hơn, nên hầu như tất cả thành viên team đều lựa chọn Java.

**8.** Java là một hòn đảo ở Indonesia, ở nơi này sản phẩm coffee đầu tiên được sản xuất (gọi là java coffee).

**9.** Bạn nên nhớ rằng Java chỉ là một tên chứ không phải là tên lược danh.

**10.** Java được phát triển đầu tiên bởi Jamas Gosling tại Sun Microsystems (bây giờ là công ty con của Oracle Coporation) và được công bố năm 1995.

**11.** Năm 1995, tạp chí Time bình chọn Java là một trong 10 sản phẩm tốt nhất năm 1995.

**12.** JDK 1.0 được công bố vào 23/1/1996.

### Lịch sử các phiên bản Java

Trải qua quá trình cải tiến và phát triển, đã có nhiều phiên bản Java được công bố. Phiên bản Java hiện tại là Java SE 8.

* JDK Alpha và Beta (1995)
* JDK 1.0 (23/1/1996)
* JDK 1.1 (19/2/1997)
* J2SE 1.2 (8/12/1998)
* J2SE 1.3 (8/5/2000)
* J2SE 1.4 (6/5/2002)
* J2SE 5.0 (30/9/2004)
* Java SE 6 (11/12/2006)
* Java SE 7 (28/7/2011)
* Java SE 8 (18/3/2014)

## CÀI ĐẶT JDK VÀ BIẾN MÔI TRƯỜNG CHO JAVA

Để lập trình Java thì cần phải có **2 thành phần**:

* **JDK:** viết tắt của Java Development Kit là một bộ phần mềm cung cấp môi trường phát triển ứng dụng viết bằng ngôn ngữ Java. JDK bao gồm cả Java Runtime Environment giúp lập trình viên có thể chạy thử để kiểm tra ứng dụng trong quá trình phát triển ứng dụng. Các bạn hoàn toàn có thể tải và cài đặt các phiên bản JDK, tuy nhiên trong phạm vi khóa học này mình sử dụng JDK phiên bản 8 có mã java là 1.8.0\_301
* **Phần mềm hỗ trợ soạn thảo mã nguồn** (source code). IDE mà mình sử dụng trong phạm vi khóa học là Visual Studio Code.

### Java JDK là gì? Nên tải / cài đặt phiên bản Java JDK nào?

**Java Development Kit** (Bộ công cụ phát triển Java - viết tắt là JDK) này cho phép bạn viết code và chạy các chương trình lập trình Java.

**Tải** và **Cài đặt Java** trong Windows: download miễn phí tại link sau [**Download Java**](http://java.sun.com/javase/downloads/index_jdk5.jsp). Bạn nên download phiên bản Java phù hợp với hệ điều hành máy mà bạn đang sử dụng.

**Cách cài đặt Java JDK** phiên bản JDK 7, JDK 8, JDK 8.1, JDK 9, JDK, 10, JDK 11, JDK 12... hay phiên bản **Java JDK mới nhất** cũng sẽ tương tự.

Bạn có thể cài đặt nhiều phiên bản Java JDK trên cùng một PC.

Nhưng Oracle (Nhà phát hành Java) khuyên bạn chỉ **nên cài đặt phiên bản mới nhất** hoặc **phiên bản được hỗ trợ dài hạn mới nhất** có dán nhãn **LTS** (Long-Term Support) vào sau tên phiên bản.

* **Ví dụ như Java SE 11.0.2 (LTS)**

**Bước 1**: Tải Phiên bản Java JDK chuẩn trên trang chủ của Oracle

Để **tải Java JDK** với phiên bản chuẩn nhất ta làm các bước sau:

* Vào trang tải Java JDK của Oracle [Tại đây](https://www.oracle.com/technetwork/java/javase/downloads/index.html)
* Chọn phiên bản mới nhất hoặc được hỗ trợ dài hạn (LTS)

Tải xuống Phiên bản Java JDK mới nhất Java SE 12

Tải xuống Phiên bản Java JDK được hỗ trợ dài hạn mới nhất Java SE 11.0.2 (LTS)

Ở đây, để hoạt động tốt nhất và ổn định nhất. Mình chọn Phiên bản Java SE 11.0.2 (LTS) vì nó được Oracle gắn nhãn hỗ trợ dài hạn.

**Bước 2**: Chọn Phiên bản Java JDK phù hợp với hệ điều hành sử dụng (Windows)

Bước 1 mới chỉ chọn phiên bản phù hợp.

Tuy nhiên hệ điều hành mỗi người sử dụng lại khác nhau. Java JDK cũng có phiên bản cho từng hệ điều hành riêng.

Ở đây mình sẽ hướng dẫn tải và cài đặt Java JDK cho Windows nên mình lựa chọn như sau:

* Tiếp tục ở bước 1, hoặc vào link chọn phiên bản JDK cho hệ điều hành phù hợp [tại đây](https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html)

Lựa chọn phiên bản Java JDK (LTS) phù hợp với hệ điều hành

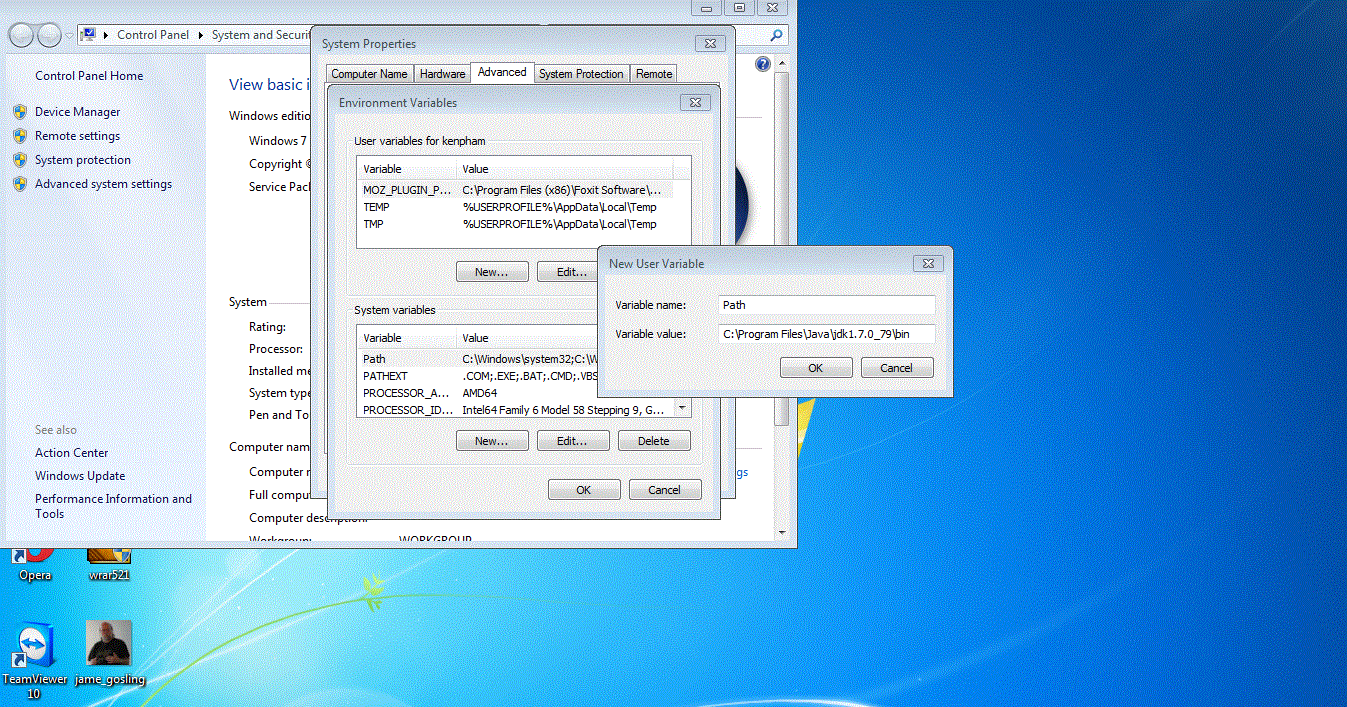
**Bước 3**: Tiến hành cài đặt Java JDK trên máy tính

Khi quá trình tải xuống hoàn tất, hãy chạy file .exe để **cài đặt Java JDK**. Nhấn Next  
Sau khi cài đặt hoàn tất, nhấn Close

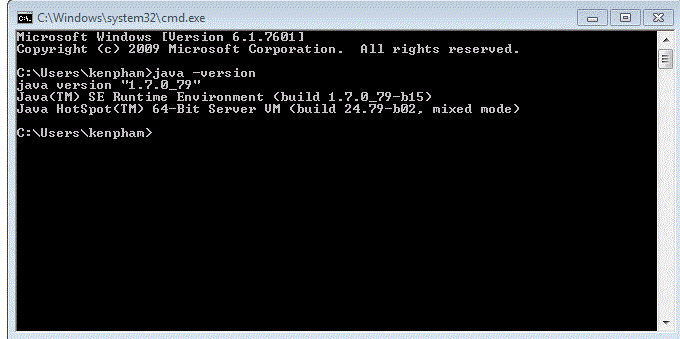
### Cách thiết lập path trong java

Thiết lập path trong Java là rất cần thiết nếu bạn cần sử dụng các tool liên quan đến javac hoặc java (dùng terminal chẳng hạn)

Vào MyComputer properties -> advanced tab -> environment variables -> tạo tab mới user variable -> viết giá trị path trong tên biến -> dẫn đến thư mục bin của cài đăt java -> ok -> ok -> ok



Dưới đây là các kiểm tra classpath của ngôn ngữ Java. Bạn nhớ phải khởi động lại máy sau khi thiết lập classpath nhé.



## CHƯƠNG TRÌNH HELLO WORLD

Chương này, chúng ta sẽ học cách viết một chương trình Java đơn giản. Trước khi viết chương trình Hello World, bạn nên cài đặt trước JDK.

Để tạo một chương trình Java đơn giản, bạn cần tạo một lớp chứa phương thức main. Để thực thi bất cứ chương trình Java nào, bạn cần:

**Platform:**Bất cứ môi trường phần cứng hoặc phần mền nào mà trong đó một chương trình chạy, thì được biết đến như là một Platform. Với môi trường runtime riêng cho mình là JRE và API, Java được gọi là Platform.

* Cài đặt JDK nếu bạn chưa cài đặt nó. Bạn [**tải JDK**](http://www.oracle.com/technetwork/java/javase/downloads/index.html), và cài đặt.
* Thiết lập path của thư mục jdk/bin. (Bạn có thể tham khảo chương [**Cách thiết lập Path trong Java**](https://vietjack.com/java/cach_thiet_lap_path_trong_java.jsp))
* Tạo chương trình Java.
* Biên dịch và chạy chương trình Java.

Tạo **ví dụ Hello World**

Bạn theo dõi ví dụ để viết chương trình Hello World:

class Simple{

public static void main(String args[]){

System.out.println("Hello World");

}

}

Sau đó, bạn lưu file dưới tên **Simple.java**.

* Để biên dịch, bạn gõ: **javac Simple.java**
* Để thực thi, bạn gõ: **java Simple**

### Phân tích chi tiết chương trình Java đầu tiên

Dưới đây, chúng tôi sẽ giúp bạn hiểu ý nghĩa của class, public, static, void, main, String[], System.out.println().

* Từ khóa **class** được sử dụng để khai báo một lớp trong Java.
* Từ khóa **public** là một Access Modifier mà biểu diễn tính nhìn thấy, nghĩa rằng nó là nhìn nhất với tất cả.
* **static** là một từ khóa, mà nếu chúng ta khai báo bất cứ phương thức nào là static thì nó còn được gọi là phương thức tĩnh hoặc phương thức static. Lợi thế chủ yếu của phương thức static là không cần thiết tạo đối tượng để triệu hồi phương thức static. Phương thức main được thực thi bởi JVM, vì thế bạn không cần thiết tạo một đối tượng để gọi phương thức main. Việc này giúp tiết kiệm bộ nhớ.
* **void** là kiểu trả về của phương thức, nghĩa là phương thức không trả về bất cứ giá trị nào.
* **main** tượng trưng cho khởi động chương trình.
* **String[] args** được sử dụng cho tham số dòng lệnh. Bạn sẽ tìm hiểu về chúng sau.
* **System.out.println()** được sử dụng như là lệnh in. Chương sau, bạn sẽ thấy cách làm việc nội tại của lệnh System.out.println này.

Để viết một chương trình đơn giản, bạn mở notepad bởi **Start menu** -> **All Programs** -> **Accessories** -> **notepad** và viết chương trình đơn giản Hello World trên.

Viết và lưu chương trình dưới dạng Simple.java. Để biên dịch và chạy chương trình này, bạn cần mở dòng nhắc lệnh command prompt bởi **Start menu** -> **All Programs** -> **Accessories** -> **command prompt**.

Để biên dịch và chạy chương trình, đầu tiên bạn vào thư mục hiện tại, với mình đó là c:\new. Viết ở đây:

* Để biên dịch, gõ: javac Simple.java
* Để thực thi, gõ: java Simple

### Bạn có bao nhiêu cách để viết một chương trình Java

Có nhiều cách để viết một chương trình Java. Các sửa đổi có thể được thực hiện trong chương trình Java như sau:

Thứ nhất bằng việc thay đổi dãy sắp xếp của Modifier, thì nguyên mẫu phương thức không bị thay đổi. Bạn theo dõi phần code đơn giản của phương thức main:

static public void main(String args[])

Thứ hai, ký hiệu chỉ số dưới [] trong Mảng Java có thể được sử dụng sau kiểu, ở trước biến hoặc đằng sau biến. Bạn theo dõi các code khác nhau để viết phương thức main:

public static void main(String[] args)

public static void main(String []args)

public static void main(String args[])

Thứ ba, bạn có thể cung cấp sự hỗ trợ var-args tới phương thức main bằng cách truyền 3 dấu chấm. Bạn theo dõi code đơn giản sau sử dụng var-args trong phương thức main. Chúng ta sẽ học về var-args trong chương Các đặc điểm mới trong Java.

public static void main(String... args)

Thứ tư, việc có hay không dấu chấm phảy ở phần cuối của lớp trong Java là tùy ý. Bạn theo dõi code sau:

class A{

static public void main(String... args){

System.out.println("Hello World");

}

};

Một số phương thức main hợp lệ trong Java:

public static void main(String[] args)

public static void main(String []args)

public static void main(String args[])

public static void main(String... args)

static public void main(String[] args)

public static final void main(String[] args)

final public static void main(String[] args)

final strictfp public static void main(String[] args)

Một số phương thức main không hợp lệ trong Java:

public void main(String[] args)

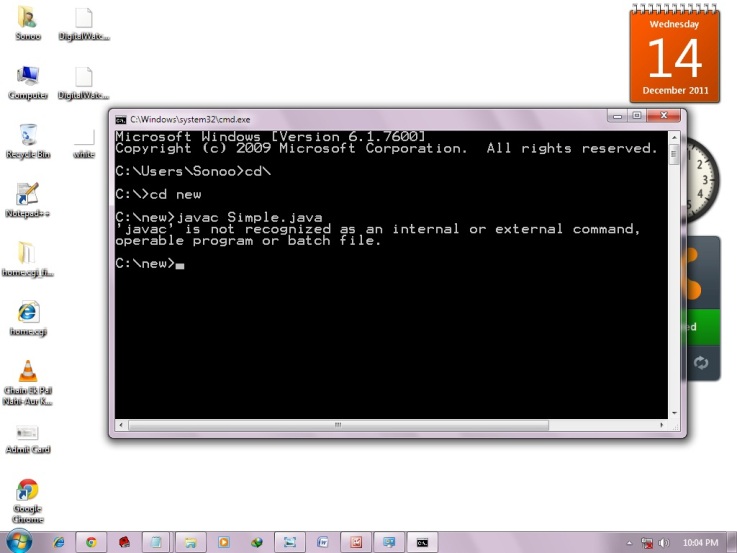
static void main(String[] args)

public void static main(String[] args)

abstract public static void main(String[] args)

### Giải quyết lỗi "javac is not recognized as an internal or external command"

Nếu xuất hiện một vấn đề như được hiển thị trong hình dưới, thì điều bạn cần làm là thiết lập path. Vì DOS không biết javac hoặc java, do đó chúng ta cần thiết phải thiết lập path. Path là không cần thiết trong những trường hợp mà nếu bạn lưu chương trình bên trong thư mục jdk/bin. Nhưng đôi khi, đây là điều bạn cần biết để có thể xử lý các trường hợp có thể xảy ra. Bạn theo dõi phần này tại **Cách thiết lập path trong Java**.

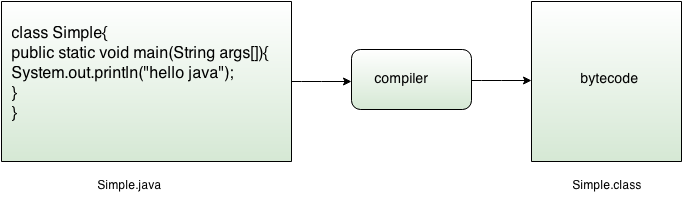


## PHÂN TÍCH NỘI TẠI CHƯƠNG TRÌNH HELLO WORLD TRONG JAVA

Trong chương trước, chúng ta đã học cách viết, biên dịch và chạy chương trình Hello World trong Java. Ở đây, chúng ta tiếp tục tìm hiểu những gì xảy ra trong khi biên dịch và chạy một chương trình Java. Ngoài ra, bạn sẽ thấy một số câu hỏi liên quan đến chương trình Java đầu tiên này.

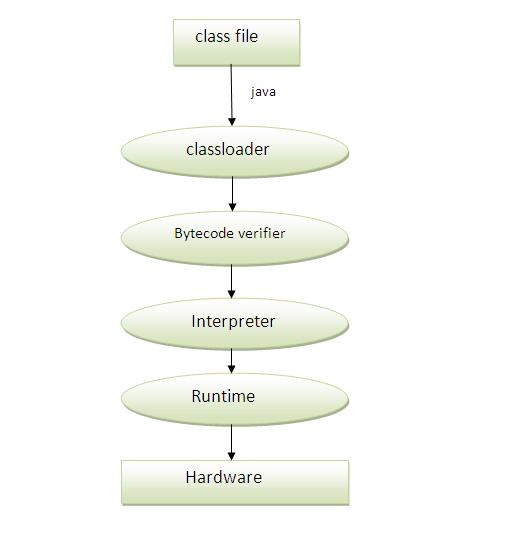
### Những gì xảy ra tại Compile time?

Tại compile time (thời gian biên dịch), java file được biên dịch bởi Java Compiler (Nó không tương tác với OS) và chuyển đổi Java code thành Bytecode.



### Điều gì xảy ra tại runtime?

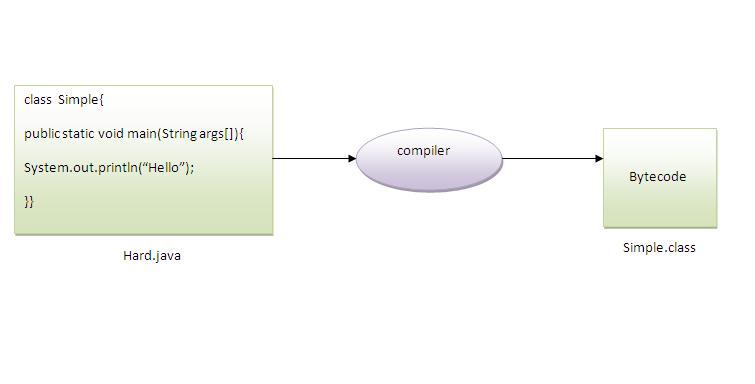
Tại Runtime (thời gian biên dịch), các bước sau được thực hiện:



* **Classloader**: Đây là hệ thống con của JVM được sử dụng để tải class file.
* **Bytecode Vertifier**: Kiểm tra các đoạn code để xem có hay không các phần code không hợp lệ có truy cập không hợp lệ tới các đối tượng.
* **Interpreter**: Đọc Bytecode Stream, sau đó thực thi các chỉ thị.

### Câu hỏi: Bạn có thể lưu một source file bởi một tên khác ngoài tên lớp không?

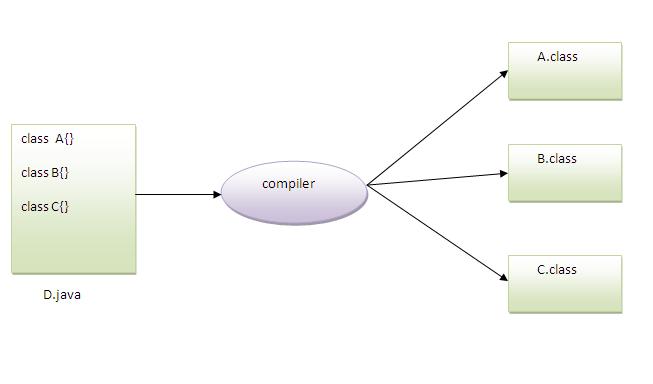
Có, nếu lớp không phải là public. Bạn có thể theo dõi ở sơ đồ dưới:



* Để biên dịch, gõ: javac Hard.java
* Để thực thi, gõ: java Simple

### Bạn có thể có nhiều lớp trong một source file không?

Có, sơ đồ sau minh họa điều này:



# CÚ PHÁP JAVA CƠ BẢN

## NHẬP VÀ XUẤT DỮ LIỆU TRONG JAVA

### Các phần tử cơ sở của Java

  Cũng giống như các ngôn ngữ lập trình khác, ngôn ngữ lập trình Java được định nghĩa bằng tập các quy tắc của văn phạm cho phép kiểm tra xem các cấu trúc được xây dựng từ các phần tử ngôn ngữ có hợp lệ hay không và thông qua các định nghĩa về mặt ngữ nghĩa để kiểm chứng ý nghĩa của các cấu trúc đó. Nếu các bạn độc giả nào đang theo học trong những môi trường học thuật như Đại học, cao đẳng mà đúng chuyên môn về công nghệ thông tin thì chắc hẳn không con lạ lẫm với hai môn “Otomat và ngôn ngữ hình thức” và môn “Chương trình dịch”, xin phép các bạn chút xíu thời gian để nói qua về môn “Chương trình dịch” nhé vì mình cũng có vài video về môn này. [Xem tại đây](https://youtu.be/arRJo7VB1fs)

**Mục đích môn học:**

* Hiểu nguyên lý của một chương trình dịch, tổng quan về các loại chương trình dịch.
* Hiểu được cấu trúc thành phần của một chương trình dịch.
* Áp dụng được các kiến thức về ngôn ngữ hình thức trong thực hành chương trình dịch.
* Nắm vững về lý thuyết về xây dựng chương trình dịch, tập trung vào phân tích từ vựng, phân tích cú pháp.
* Có khả năng tự xây dựng được các thành phần trong chương trình dịch, tập trung vào: phân tích từ vựng, phân tích cú pháp.

–> Ta phải tuân theo các quy tắc đã được xây dựng từ trước khi lập trình với Java.

#### Định danh

Tên gọi của các thành phần trong chương trình được gọi là định danh (Identifier). Định danh được sử dụng để xác định các phần tử như biến, kiểu, phương thức (method) hay còn gọi là hàm (function), đối tượng, …

Trong Java, định danh là một dãy các ký tự gồm các chữ cái, chữ số và một số các ký hiệu như: ký hiệu gạch dưới nối câu ‘\_’, các ký hiệu tiền tệ $, £, ¥, ¢ và không được bắt đầu bằng chữ số (0, …, 9)

Java phân biệt chữ thường và chữ hoa. Ví dụ Hoa và hoa là hai định danh khác nhau. Độ dài (số lượng ký tự) của định danh trong Java về lý thuyết là không bị giới hạn

Xu hướng chung hiện nay là nên sử dụng cách đặt tên theo một chuẩn nhất định để phân biệt được các loại khác nhau của các thành phần sử dụng. Đặc biệt là khi làm việc trong teamwork thì các quy tắc đã thống nhất là rất quan trọng, dễ dàng trao đổi, phát triển và bảo trì. **Quy ước thống nhất cách đặt tên như sau**:

* Định danh cho các lớp: chữ cái đầu của mỗi từ trong định danh đều viết hoa –> **Quy tắc Lạc đà (Camel Case)**. Ví dụ: HocSinh, KhachHang, Sach.
* Định danh cho các biến, phương thức, đối tượng: chữ cái đầu của mỗi từ trong định danh đều viết hoa từ từ đầu tiên. Ví dụ: hoTen, lop, diaChi, soLuongTrang.

#### Các từ khóa

  Các từ khóa của một ngôn ngữ lập trình là những định danh đã định sẵn được định nghĩa trước của một ngôn ngữ và không thể sử dụng cho những mục đích khác. Các từ khóa của Java có thể chia thành 09 nhóm như sau:

|  |  |
| --- | --- |
| **Chức năng** | **Từ khóa** |
| Tổ chức các lớp | package, import |
| Định nghĩa các lớp | interface, class, extends, implements |
| Các từ khóa cho các biến và các lớp | abstract, public, private, protected, static, synchronized, volatile, final, native |
| Các kiểu nguyên thủy | long, int, short, byte, char, float, double, boolean, void |
| Những từ khóa cho các giá trị và các biến | false, true, this, super, null |
| Xử lý ngoại lệ | throw, throws, try, catch, finally |
| Tạo lập và kiểm tra các đối tượng | new, instanceof |
| Dòng điều khiển | switch, case, default, break, if, else, continue, return, do, while, for |
| Những từ khóa chưa được sử dụng | byvalue, const, goto, cast |

#### Chú thích

  Trong các chương trình thường cần phải có những đoạn tư liệu giải thích công việc hoặc cách thực hiện để người đọc hiểu và tiện theo dõi. Những phần giải thích đó chỉ nhằm làm tư liệu và được các bộ trình biên dịch bỏ qua. Java cung cấp 3 loại chú thích chương trình:

* Chú thích trên một dòng
* Chú thích trên nhiều dòng
* Chú thích trong tư liệu (javadoc)

##### Chú thích trên một dòng

  Tất cả các ký tự sau // cho đến cuối dòng là chú thích.

##### Chú thích trên một dòng

  Giống như trong C/C++, phần nằm giữa /\* và \*/ là chú thích.

##### Chú thích trên một dòng

  Đây là loại chú thích đặc biệt vào những chỗ thích hợp trong chương trình để javadoc và sử dụng để tạo ra tư liệu dạng HTML cho chương trình. Chúng được đặt vào trước phần định nghĩa các lớp, interface, phương thức và biến. Phần chú thích trong tư liệu được bắt đầu bằng /\*\* và kết thúc bằng \*/

Ví dụ:

/\*\*

\* Lớp này nhắm mục đích lấy ví dụ.

\* Author: Hoàng Văn Tuân.

\* Ngày: 26/07/2021

\*/

**Lưu ý:**

**1. Đặt tên Package.** Tên Package phải tuân theo quy tắc chung ở trên và phải viết thường.

**2. Đặt tên Project.** Tên Project phải tuân theo quy tắc chung ở trên và chữ cái đầu tiên của mỗi từ phải viết hoa.

### Nhập dữ liệu từ bàn phím với Scanner

Đầu tiên các bạn phải import lớp Scanner vào nhé

package nhap;

import java.util.Scanner;

public class Test1 {

public static void main(String[] args) {

}

}

Sau đó các bạn khởi tạo một đối tượng sc thuộc lớp Scanner bằng câu lệnh:

Scanner sc=new Scanner(System.in);

Đối tượng sc sẽ giúp các bạn nhập dữ liệu từ bàn phím.

int a1 = sc.nextInt(); // Nhập dữ liệu kiểu int

float a2 = sc.nextFloat(); // Nhập dữ liệu kiêu float

byte a3 = sc.nextByte(); // Nhập dữ liệu kiểu byte

double a4 = sc.nextDouble(); // Nhập dữ liệu kiểu double

Và các từ khóa như int, float, … là gì và chúng có ý nghĩa gì thì mình sẽ trình bày với các bạn ở bài sau nhé. Hiện tại thì các bạn hiểu đơn giản là “Nồi nào úp vung nấy”.  
Tóm lại ta có các phương thức thường dùng trong lớp Scanner để nhập dữ liệu là:

|  |  |
| --- | --- |
| **Tên phương thức** | **Mục đích** |
| nextBoolean | Nhập vào kiểu Boolean ( true – false) từ bàn phím |
| nextByte | Nhập vào kiểu dữ liệu Byte |
| nextShort | Nhập vào kiểu Short ( số nguyên từ -32768 đến 32767) |
| nextInt | Nhập vào kiểu số nguyên từ bàn phím |
| nextFloat | Nhập vào kiểu số thực |
| nextDouble | Nhập vào kiểu Double ( số thực lớn hơn float) |
| nextLine | Nhập vào kiểu String ( String trong java giống char luôn nhé!) |
| nextLong | Nhập vào số nguyên lớn |

### Cách xuất dữ liệu ra màn hình

Trong Java bấy lâu nay, thường chúng ta tiếp xúc và làm việc nhiều với hai câu lệnh xuất cơ bản là **print**và **println**.  
Nhưng trên thực tế, ta còn có phương phức printf, trông có vẻ giống người hàng xóm C nhưng tính năng mà Java đem lại với printf là một phương thức rất hữu ích trong thực tế.  
Để làm rõ điều này, chúng ta cùng điểm qua xem 3 anh em nhà đó khác nhau như thế nào:

* Với Print: Xuất kết quả ra màn hình nhưng con trỏ chuột không xuống dòng.
* Với Println: Xuất kết quả ra màn hình đồng thời con trỏ chuột nhảy xuống dòng tiếp theo.
* Với Printf: Xuất ra màng hình kết quả đồng thời có thể định dạng được kết quả đó nhờ vào các đối số thích hợp.

**Cú pháp printf trong Java:**System.out.printf(local, format, arguments1, arguments2, … , argumentsn);

Trong đó:

* Local: Nếu khác null sẽ được tự động định dạng theo khu vực.
* format: Quy định chuẩn định dạng đầu ra cho các đối số
* Các argument: Đối số cần định dạng.

Các bộ định dạng có sẵn trong Printf:

* %c: Ký tự
* %d: Số thập phân (số nguyên) (cơ số 10)
* %e: Dấu phẩy động theo cấp số nhân
* %f: Dấu phẩy động
* %i: Số nguyên (cơ sở 10)
* %o: Số bát phân (cơ sở 8)
* %s: Chuỗi
* %u: Số thập phân (số nguyên) không dấu
* %x: Số trong hệ thập lục phân (cơ sở 16)
* %t: Định dạng ngày / giờ
* %%: Dấu phần trăm
* \%: Dấu phần trăm

**Ví dụ 1:**Hiển thị số thực theo định dạng mong muốn (Tương tự như C)

package xuat;

public class Printf

{

public static void main(String args[]){

System.out.printf("Lam tron 3.1456 la: %.2f", 3.1456);

}

}

**Ví dụ 2:**Hiển thị thông tin ngày tháng năm hiện tại

package xuat;

import java.util.Date;

public class Printf

{

public static void main(String args[]){

Date date = new Date();

System.out.printf("%tT%n", date);

System.out.printf("H : %tH, M: %tM, S: %tS%n",date,date,date);

System.out.printf("%s %tb %<te, %<ty", "Ngày hiện tại: ", date);

}

}

#### Điểm khác nhau giữa println và print

Java hỗ trợ hai lệnh là **println** và **print** để in thông tin trên màn hình chuẩn. Tuy nhiên giữa hai lệnh này có một điểm khác nhau nhỏ về con trỏ trong hai lệnh. Trong khi lệnh **print** giữ nguyên vị trí con trỏ trên cùng một dòng thì lệnh **println** di chuyển con trỏ xuống dòng tiếp theo (tương đương với ký tự newline (dòng mới)).

Bạn theo dõi ví dụ đơn giản sau để phân biệt hai lệnh này:

package edu.doannhg.basic;

// vi du phan biet hai lenh print va println

class Test {

public static void main(String args[]) {

// lenh print

System.out.print("VietJack ");

System.out.print("chuc cac ban ");

System.out.print("hoc tot!!!");

System.out.print("\n---------------\n");

// lenh println

System.out.println("VietJack ");

System.out.println("chuc cac ban ");

System.out.println("hoc tot!!!");

}

}

Chạy chương trình Java trên sẽ cho kết quả:

VietJack chuc cac ban hoc tot!!!

---------------

VietJack

chuc cac ban

hoc tot!!!

## GIỚI THIỆU JDK, JRE VÀ JVM TRONG JAVA

Hiểu rõ sự khác nhau giữa JDK, JRE và JVM là điều khá quan trọng trong Java. Ở đây, chúng tôi trình bày miêu tả ngắn gọn về JVM, để biết thêm chi tiết về nó, bạn truy cập vào chương tiếp theo. Đầu tiên chúng ta tìm hiểu sự khác nhau cơ bản giữa JDK, JRE và JVM.

### Sơ lược về JVM

JVM (viết tắt của Java Virtual Machine) là một thiết bị trừu tượng (ảo) có thể giúp máy tính chạy các chương trình Java. Nó cung cấp môi trường runtime mà trong đó Java Bytecode có thể được thực thi.

JVM là có sẵn cho nhiều nền tảng (Windows, Linux...). JVM, JRE và JDK là phụ thuộc nền tảng, bởi vì cấu hình của mỗi OS (hệ điều hành) là khác nhau. Nhưng, Java là độc lập nền tảng.

### JVM là gì?

JVM thực hiện các tác vụ chính sau:

* Tải code
* Kiểm tra code
* Thực thi code
* Cung cấp môi trường runtime

JVM cung cấp các định nghĩa cho: Khu vực bộ nhớ, định dạng class file, thiết lập Register, Heap cho Trình dọn rác và các báo cáo lỗi nghiêm trọng (Fatal Error), …

### Cấu trúc nội tại của JVM

Bạn theo dõi sơ đồ sau để hiểu cấu trúc nội tại của JVM. Nó gồm Classloader, Memory Area (khu vực bộ nhớ), Excetution Engine (phương tiện thực thi), …



* **Classloader**: Là một hệ thống con của JVM được sử dụng để tải class file.
* **Class (method) Area**: Lưu trữ cấu trúc mỗi lớp, chẳng hạn như hằng, trường, dữ liệu phương thức, code của phương thức, …
* **Heap**: Nó là khu vực dữ liệu runtime mà trong đó đối tượng được cấp phát.
* **Stack**: Stack trong Java lưu giữ các Frame. Nó giữ các biến cục bộ và các kết quả cục bộ, và thực hiện một phần nhiệm vụ trong phần triệu hồi và trả về phương thức. Mỗi Thread có một Stack riêng, được tạo tại cùng thời điểm với Thread.

Một Frame mới được tạo mỗi khi một phương thức được triệu hồi và bị hủy khi lời triệu hồi phương thức là kết thúc.

* **Program Counter Register**: Nó chứa địa chỉ của chỉ lệnh JVM hiện tại đang được thực thi.
* **Native Method Stack**: Bao gồm tất cả các phương thức tự nhiện được sử dụng trong ứng dụng.
* **Execution Engine**: Phần này bao gồm:

Một bộ xử lý ảo Virtual Processor

Một trình thông dịch Interpreter. Đọc Bytecode Stream sau đó thực thi các chỉ thị.

**Just-In-Time (JIT) Compiler**: được sử dụng để cải thiện hiệu suất. JIT biên dịch các phần của Bytecode mà có cùng tính năng tại cùng một thời điểm, và vì thế giảm lượng thời gian cần thiết để biên dịch. Ở đây khái niệm Compiler là một bộ biên dịch tập chỉ thị của JVM thành tập chỉ thị của một CPU cụ thể.

### Giới thiệu JRE

JRE (là viết tắt của Java Runtime Environment) được sử dụng để cung cấp môi trường runtime. Nó là trình triển khai của JVM. JRE bao gồm tập hợp các thư viện và các file khác mà JVM sử dụng tại runtime. Trình triển khai của JVM cũng được công bố bởi các công ty khác ngoài Sun Micro Systems.

### Sơ lược về JDK

JDK (là viết tắt của Java Development Kit) bao gồm JRE và các Development Tool.

## BIẾN VÀ KIỂU DỮ LIỆU TRONG JAVA

Chương này, chúng ta cùng tìm hiểu về các kiểu biến trong Java. Biến là tên một khu nhớ. Có ba kiểu biến: biến cục bộ (còn gọi là biến local), thuộc tính (biến của instance- đối tượng) và biến static.

Bạn phải khai báo tất cả các biến trước khi chúng có thể được sử dụng. Form cơ bản của một khai báo biến như sau:

kieu\_du\_lieu bien [ = giatri][, bien [= giatri] ...] ;

Ở đây, *kieu\_du\_lieu* là một kiểu dữ liệu của Java và *bien* là tên của biến. Để khai báo nhiều hơn một biến với kiểu cụ thể, bạn có thể sử dụng một danh sách biến phân biệt nhau bởi dấu phảy.

Dưới đây là ví dụ về khai báo và khởi tạo biến hợp lệ trong Java:

int a, b, c; // Khai bao ba bien kieu int la a, b, và c.

int a = 5, b = 7; // Vi du ve khoi tao bien

byte A = 11; // Khoi tao mot bien kieu byte ten la A.

double pi = 3.14159; // Khai bao va gan mot gia tri cua PI.

char nam = 'b'; // Bien nam duoc khoi tao voi gia tri 'a'.

### Biến local trong Java

* Các biến được khai báo trong các phương thức, constructor, hoặc khối.
* Các biến được tạo khi phương thức, constructor hoặc khối được nhập và biến bị hủy khi phương thức, constructor hoặc khối kết thúc.
* Chỉ định truy cập (access modifier) có thể không được sử dụng cho các biến local.
* Các biến local chỉ nhìn thấy trong phương thức, constructor, hoặc khối được khai báo.
* Các biến local được thực thi nội bộ.
* Không có giá trị mặc định nào cho các biến local, vì thế các biến local nên được khai báo và một giá trị khởi tạo nên được gán trước khi sử dụng.

**Ví** dụ**:**

Ở đây, *tuoi* là biến local. Nó được định nghĩa bên trong phương thức *tuoiCon()* và phạm vi của nó được giới hạn chỉ trong phương thức này.

public class Test{

public void tuoiCon(){

int tuoi = 0;

tuoi = tuoi + 10;

System.out.println("Tuoi con la : " + tuoi);

}

public static void main(String args[]){

Test test = new Test();

test.tuoiCon();

}

}

**Ví dụ:**

Ví dụ sau sử dụng *tuoi* mà không khởi tạo nó, vì thế nó sẽ tạo một lỗi tại thời gian biên dịch.

public class Test{

public void tuoiCon(){

int tuoi;

tuoi = tuoi + 10;

System.out.println("Tuoi con la : " + tuoi);

}

public static void main(String args[]){

Test test = new Test();

test.tuoiCon();

}

}

Nó sẽ cho một lỗi sau trong khi biên dịch:

Test.java:4:variable number might not have been initialized

tuoi = tuoi + 10;

^

1 error

### Thuộc tính (hay biến instance) trong Java

* Các thuộc tính được khai báo trong một lớp, nhưng ở bên ngoài một phương thức, constructor hoặc bất kỳ khối nào.
* Khi một không gian nhớ được cấp phát cho một đối tượng trong một heap (một dạng cấu trúc hàng đợi có thứ tự ưu tiên), một slot cho mỗi giá trị thuộc tính được tạo ra.
* Các thuộc tính được tạo khi một đối tượng được tạo bởi sử dụng từ khóa new và bị hủy khi đối tượng bị hủy.
* Các thuộc tính giữ các giá trị mà phải được tham chiếu bởi nhiều hơn một phương thức, constructor hoặc khối, hoặc các phần chủ yếu của trạng thái đối tượng mà phải có mặt xuyên suốt lớp đó.
* Các thuộc tính có thể được khai báo trong mức độ lớp trước hoặc sau khi sử dụng.
* Chỉ định truy cập (access modifier) có thể được cung cấp cho các thuộc tính.
* Các thuộc tính là nhìn thấy với tất cả các phương thức, constructor, và khối trong lớp. Tuy nhiên, tính nhìn thấy cho các lớp phụ có thể được cung cấp cho những biến này với sự sử dụng của chỉ định truy cập.
* Các thuộc tính có các giá trị mặc định. Với các số, giá trị mặc định là 0, với Boolean là false và với đối tượng là null. Các giá trị có thể được gán trong khi khai báo hoặc trong constructor.
* Các thuộc tính có thể được truy cập một cách trực tiếp bởi việc gọi tên biến bên trong lớp đó. Tuy nhiên, với các phương thức static và các lớp khác nhau (khi thuộc tính được cung cấp khả năng truy cập), nó nên được gọi bởi sử dụng tên đầy đủ hợp lệ như sau: *ObjectReference.VariableName*.

**Ví** dụ**:**

import java.io.\*;

public class Student{

// Bien instance nay la nhin thay cho bat ky lop con nao.

public String ten;

// Bien hocphi la chi nhin thay cho lop Student.

private double hocphi;

// Bien ten duoc gan trong constructor.

public Student (String tenSV){

ten = tenSV;

}

// Bien hocphi duoc gan mot gia tri.

public void setHocPhi(double hp){

hocphi = hp;

}

// Phuong thuc nay in chi tiet ve Student.

public void inThongTin(){

System.out.println("Ho va ten: " + ten );

System.out.println("Hoc phi: " + hocphi);

}

public static void main(String args[]){

Student sv1 = new Student("Nguyen Van Doan");

sv1.setHocPhi(4000);

sv1.inThongTin();

}

}

Nó sẽ cho kết quả sau:

Ho va ten: Nguyen Van Doan

Hoc phi: 4000.0

### Các biến Class/static trong Java:

* Các biến class cũng được biết như là các biến static được khai báo với từ khóa static trong một lớp, nhưng ở bên ngoài một phương thức, constructor hoặc một khối.
* Sẽ chỉ có một bản sao của mỗi biến class cho mỗi lớp, bất chấp việc bao nhiêu đối tượng được tạo từ nó.
* Các biến static hiếm khi được sử dụng, ngoài việc được khai báo như là các hằng số. Các hằng số là các biến, mà được khai báo như là các biến static, biến final, biến chung/riêng. Các biến hằng số không bao giờ thay đổi từ giá trị khởi tạo của chúng.
* Các biến static được lưu giữ trong bộ nhớ static.
* Các biến static được tạo khi chương trình bắt đầu và bị hủy khi chương trình kết thúc.
* Tính nhìn thấy là tương tự như các thuộc tính. Tuy nhiên, hầu hết biến static được khai báo chung khi chúng phải là có sẵn cho việc sử dụng của lớp.
* Giá trị mặc định là giống với các thuộc tính. Với các số, giá trị mặc định là 0; với Boolean là false, và với đối tượng là null. Các giá trị có thể được gán trong khi khai báo hoặc trong constructor. Ngoài ra, các giá trị có thể được gán trong các khối khởi tạo static đặc biệt.
* Các biến static có thể được truy cập bởi việc gọi tên lớp: *ClassName.VariableName*.
* Khi khai báo các biến class như public static final, thì các tên biến ở dạng chữ hoa. Nếu các biến static không là public và final, thì cú pháp đặt tên giống với các biến local và thuộc tính.

Ví dụ:

import java.io.\*;

public class Student{

// Bien hocphi la mot bien private static

private static double hocphi;

// FACULTY la mot constant

public static final String FACULTY = "KhoaIT ";

public static void main(String args[]){

hocphi = 4000;

System.out.println(FACULTY+"hocphi trung binh:"+hocphi);

}

}

Nó sẽ cho kết quả sau:

KhoaIT hocphi trung binh:4000

**Ghi chú:** Nếu các biến được truy cập từ lớp bên ngoài, hằng số nên được truy cập như ở dạng: Student.FACULTY

## ÉP KIỂU DỮ LIỆU TRONG JAVA

Các biến là không gì khác ngoài các vị trí bộ nhớ dành riêng để lưu các giá trị. Nghĩa là, khi bạn tạo một biến, bạn dự trữ một số không gian trong bộ nhớ.

Dựa trên kiểu dữ liệu của một biến, hệ điều hành cấp phát bộ nhớ và quyết định cái gì có thể được lưu giữ trong bộ nhớ dành riêng. Vì thế, bằng việc gán các kiểu dữ liệu khác nhau tới các biến, bạn có thể lưu giữ integer, thập phân, hoặc các ký tự trong những biến này.

Có hai kiểu dữ liệu có sẵn trong Java:

* Các kiểu dữ liệu gốc (Primitive)
* Các kiểu dữ liệu tham chiếu/đối tượng (không phải kiểu gốc Non-primitive)



Trước khi đi vào giới thiệu chi tiết về từng kiểu dữ liệu, bạn có thể theo dõi bảng tóm tắt sau:

|  |  |  |
| --- | --- | --- |
| **Kiểu dữ liệu** | **Giá trị mặc định** | **Kích cỡ mặc định** |
| boolean | false | 1 bit |
| char | '\u0000' | 2 byte |
| byte | 0 | 1 byte |
| short | 0 | 2 byte |
| int | 0 | 4 byte |
| long | 0L | 8 byte |
| float | 0.0f | 4 byte |
| double | 0.0d | 8 byte |

**Câu hỏi**: Tại sao char sử dụng 2 byte trong Java và \u0000 là gì?

Bởi vì Java sử dụng Unicode chứ không phải ASCII. \u0000 là dãy thấp nhất trong Unicode.

### Các kiểu dữ liệu gốc trong Java

Có 8 kiểu dữ liệu gốc được hỗ trợ bởi Java. Các kiểu dữ liệu gốc này được tiền định nghĩa bởi ngôn ngữ và được định danh bởi một từ khóa. Dưới đây là chi tiết về 8 kiểu dữ liệu gốc này:

### char:

* Dùng để lưu dữ liệu kiểu kí tự hoặc số nguyên không âm có kích thước 2 byte (16 bit)
* Giá trị nhỏ nhất là '\u0000' (hoặc 0) và giá trị lớn nhất là '\uffff' (hoặc 65,535)..
* Ví dụ: char nam ='T'

#### Tại sao Java sử dụng Unicode?

Trước Unicode, có nhiều chuẩn ngôn ngữ như ASCII, ISO 8859-1, KOI-8, GB18030, … Điều này gây là hai vấn đề sau:

* Một giá trị code cụ thể tương ứng với các chữ cái khác nhau trong các chuẩn ngôn ngữ đa dạng.
* Mã hóa cho các ngôn ngữ với các tập ký tự lớn có độ dài biến đổi. Một số ký tự chung được mã hóa thành các byte đơn, một số khác cần hai hoặc nhiều byte.

Để xử lý các vấn đề này, một chuẩn ngôn ngữ mới được phát triển, đó là Unicode. Trong Unicode, ký tự giữ 2 byte, vì thế Java cũng sử dụng 2 byte cho các ký tự.

### Kiểu dữ liệu byte:

* Dùng để lưu dữ liệu kiểu số nguyên có kích thước một byte (8 bit)
* Giá trị nhỏ nhất là -128 (-2^7) và giá trị lớn nhất là 127. (2^7 -1)
* Giá trị mặc định là 0
* Kiểu dữ liệu byte được sử dụng để lưu giữ khoảng trống trong các mảng lớn, chủ yếu là các số nguyên.
* Ví dụ: byte x = 20 , byte y = -10

### Kiểu dữ liệu short:

* Dùng để lưu dữ liệu có kiểu số nguyên, kích cỡ 2 byte (16 bit).
* Giá trị nhỏ nhất là -32,768 (-2^15) và giá trị lớn nhất là 32,767 (2^15 -1).
* Kiểu dữ liệu short cũng có thể được sử dụng để lưu bộ nhớ như kiểu dữ liệu byte.
* Giá trị mặc định là 0.
* Ví dụ: short t = 50, short z = -10

### int:

* Dùng để lưu dữ liệu có kiểu số nguyên, kích cỡ 4 byte (32 bit).
* Giá trị nhỏ nhất là - 2,147,483,648.(-2^31) và giá trị lớn nhất là 2,147,483,647 (2^31 -1)
* Nói chung, int được sử dụng như là kiểu dữ liệu mặc định cho các giá trị nguyên.
* Giá trị mặc định là 0.
* Ví dụ: int a = 5, int b = -50

### long:

* Dùng để lưu dữ liệu có kiểu số nguyên có kích thước lên đến 8 byte.
* Giá trị nhỏ nhất là -9,223,372,036,854,775,808.(-2^63) và lớn nhất là 9,223,372,036,854,775,807. (2^63 -1)
* Kiểu này được sử dụng khi cần một dải giá trị rộng hơn int.
* Giá trị mặc định là 0L.
* Ví dụ: long a = 100000L, int b = -200000L

### float:

* Dùng để lưu dữ liệu có kiểu số thực, kích cỡ 4 byte (32 bit)
* Kiểu Float được sử dụng chủ yếu để lưu bộ nhớ trong các mảng rộng hơn các số thực dấu chấm động.
* Giá trị mặc định là 0.0f.
* Kiểu Float không bao giờ được sử dụng cho các giá trị chính xác như currency.
* Ví dụ: float usd = 22.5f

### double:

* Kiểu dữ liệu double được sử dụng để lưu dữ liệu có kiểu số thực có kích thước lên đến 8 byte
* Nói chung, kiểu dữ liệu này được sử dụng như là kiểu mặc định cho các giá trị decimal.
* Kiểu double không bao giờ được sử dụng cho các giá trị chính xác như currency.
* Giá trị mặc định là 0.0d.
* Ví dụ: double ct = 676.7

### boolean:

* Độ lớn chỉ có 1 bit
* Dùng để lưu dữ liệu chỉ có hai trạng thái true hoặc false
* Giá trị mặc định là false.
* Ví dụ: boolean switch1 = true

### Kiểu dữ liệu Object trong Java

* Các biến đối tượng được tạo bởi sử dụng các constructor đã được định nghĩa của các lớp. Chúng được sử dụng để truy cập các đối tượng. Những biến này được khai báo ở kiểu cụ thể mà không thể thay đổi. Ví dụ: Employee, Puppy, …
* Giá trị mặc định của bất kỳ biến đối tượng nào là null
* Một biến đối tượng có thể được sử dụng để tham chiếu tới bất kỳ đối tượng nào trong kiểu được khai báo hoặc bất kỳ kiểu tương thích nào.
* Ví dụ: Animal animal = new Animal("giraffe");

### Literal trong Java

Một Literal hay một hằng là một code nguồn biểu diễn một giá trị cố định. Chúng được biểu diễn một cách trực tiếp trong code mà không cần kỹ thuật tính toán nào.

Các literal có thể được gán tới bất kỳ kiểu biến gốc nào. Ví dụ:

byte a = 12;

char a = 'CAT'

byte, int, long, và short có thể được biểu diễn trong hệ thập phân (cơ số 10), hệ thập lục phân (cơ số 16), hoặc hệ bát phân (cơ số 8).

Tiền tố 0 được sử dụng để chỉ hệ bát phân, và tiền tố 0x chỉ dẫn hệ cơ số 16 khi sử dụng những hệ cơ số này cho các literal. Ví dụ:

int decimal = 100;

int octal = 0144;

int hexa = 0x64;

String literal (hằng chuỗi) trong Java được xác định giống như trong hầu hết các ngôn ngữ khác bằng việc bao quanh một chuỗi ký tự liên tục trong trích dẫn kép. Ví dụ về string literal:

"Hello World"

"Hai\ndong"

"\"Day la trong trich dan\""

Các hằng chuỗi và hằng ký tự có thể chứa bất kỳ ký tự Unicode nào. Ví dụ:

char x = '\u0001';

String y = "\u0001";

Ngôn ngữ Java hỗ trợ một số dãy thoát đặc biệt cho hằng chuỗi và hằng ký tự như sau:

|  |  |
| --- | --- |
| **Ký hiệu** | **Biểu diễn ký tự** |
| \n | Newline (0x0a) |
| \r | Carriage return (0x0d) |
| \f | Formfeed (0x0c) |
| \b | Backspace (0x08) |
| \s | Space (0x20) |
| \t | tab |
| \" | Trích dẫn kép |
| \' | Trích dẫn đơn |
| \\ | backslash |
| \ddd | Octal character (ddd) |
| \uxxxx | Hexadecimal UNICODE character (xxxx) |

## TOÁN TỬ TRONG JAVA

Java cung cấp rất nhiều toán tử đa dạng để thao tác với các biến. Chúng ta có thể chia tất cả các toán tử trong Java thành các nhóm sau:

* Toán tử số học
* Toán tử quan hệ
* Toán tử thao tác bit
* Toán tử logic
* Toán tử gán
* Và các toán tử hỗn hợp

### Toán tử số học trong Java

Các toán tử số học được sử dụng trong các biểu thức toán học theo cách tương tự như chúng được sử dụng trong đại số học. Bảng sau liệt kê các toán tử số học trong Java.

Giả sử biến A giữ giá trị 10, biến B giữ giá trị 20, thì:

[**Ví dụ**](https://vietjack.com/java/vi_du_toan_tu_so_hoc_trong_java.jsp)

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Miêu tả** | **Ví dụ** |
| + | Phép cộng | A + B sẽ cho kết quả 30 |
| - | Phép trừ: trừ toán hạng trái cho toán hạng phải | A - B sẽ cho kết quả -10 |
| \* | Phép nhân | A \* B sẽ cho kết quả 200 |
| / | Phép chia: chia toán hạng trái cho toán hạng phải | B / A sẽ cho kết quả 2 |
| % | Phép chia lấy phần dư: Lấy phần dư của phép chia toán hạng trái cho toán hạng phải | B % A sẽ cho kết quả 0 |
| ++ | Phép lượng gia: lượng gia giá trị toán hạng thêm 1 | B++ sẽ cho kết quả 21 |
| -- | Phép lượng giảm: lượng giảm giá trị toán hạng đi 1 | B-- sẽ cho kết quả 19 |

### Toán tử quan hệ trong Java

Bảng dưới liệt kê các toán tử quan hệ được hỗ trợ bởi Java.

Giả sử biến A giữ giá trị 10, biến B giữ giá trị 20, thì:

[**Ví dụ**](https://vietjack.com/java/vi_du_toan_tu_quan_he_trong_java.jsp)

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Miêu tả** | **Ví dụ** |
| == | Kiểm tra nếu giá trị của hai toán hạng có cân bằng hay không, nếu có thì điều kiện là true. | (A == B) là không true. |
| != | Kiểm tra nếu giá trị hai toán hạng là cân bằng hay không, nếu không cân bằng, thì điều kiện là true | (A != B) là true. |
| > | Kiểm tra nếu toán hạng trái có lớn hơn toán hạng phải hay không, nếu có thì điều kiện là true | (A > B) là không true. |
| < | Kiểm tra nếu toán hạng phải có lớn hơn toán hạng trái hay không, nếu có thì điều kiện là true | (A < B) là true. |
| >= | Kiểm tra nếu toán hạng trái có lớn hơn hoặc bằng toán hạng phải hay không, nếu có thì điều kiện là true | (A >= B) là không true. |
| <= | Kiểm tra nếu toán hạng phải có lớn hơn hoặc bằng toán hạng trái hay không, nếu có thì điều kiện là true | (A <= B) là true. |

### Toán tử thao tác bit trong Java

Java định nghĩa một số toán tử thao tác bit có thể được áp dụng cho các kiểu giá trị integer, long, int, short, char, và byte.

Toán tử thao tác bit làm việc trên các bit. Giả sử nếu a = 60 và b = 13, thì trong định dạng nhị phân chúng sẽ như sau:

a = 0011 1100

b = 0000 1101

-----------------

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a  = 1100 0011

Bảng dưới đây liệt kê các toán tử bit được hỗ trợ trong Java:

Giả sử biến A giữ giá trị 60 và biến B giữ 13 thì khi đó:

[**Ví dụ**](https://vietjack.com/java/vi_du_toan_tu_thao_tac_bit_trong_java.jsp)

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Miêu tả** | **Ví dụ** |
| & | Toán tử Và nhị phân sao chép một bit tới kết quả nếu nó tồn tại trong cả hai toán hạng | (A & B) sẽ cho kết quả 12, hay là 0000 1100 |
| | | Toán tử Hoặc nhị phân sao chép một bit tới kết quả nếu nó tồn tại trong một hoặc hai toán hạng | (A | B) sẽ cho kết quả 61, hay là 0011 1101 |
| ^ | Toán tử Hoặc loại trừ nhị phân sao chép bit nếu nó được thiết lập trong một toán hạng nhưng không phải trong cả hai | (A ^ B) sẽ cho kết quả 49, hay là 0011 0001 |
| ~ | Toán tử đảo bit là toán tử một ngôi. Đảo bít 1 thành 0 và ngược lại | (~A ) sẽ cho kết quả -61, hay là 1100 0011 |
| << | Toán tử dịch trái. Giá trị toán hạng trái được dịch chuyển sang trái bởi số các bit được xác định bởi toán hạng bên phải. | A << 2 sẽ cho kết quả 240, hay là 1111 0000 |
| >> | Toán tử dịch phải. Giá trị toán hạng trái được dịch chuyển sang phải bởi số các bit được xác định bởi toán hạng bên phải | A >> 2 sẽ cho kết quả 15, hay là 1111 |
| >>> | Toán tử dịch phải và điền 0 vào chỗ trống | A >>>2 sẽ cho kết quả 15, hay là 0000 1111 |

### Toán tử logic trong Java

Bảng dưới liệt kê đầy đủ các toán tử logic trong Java:

Giả sử biến A giữ true và biến B giữ false thì khi đó:

[**Ví dụ**](https://vietjack.com/java/vi_du_toan_tu_logic_trong_java.jsp)

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Miêu tả** | **Ví dụ** |
| && | Toán tử Và logic. Nếu cả hai toán hạng là khác không, thì khi đó điều kiện là true | (A && B) là false. |
| || | Toán tử Hoặc logic. Nếu một trong hai toán tử khác 0, thì điều kiện là true | (A || B) là true. |
| ! | Toán tử Phủ định logic. Sử dụng để đảo ngược lại trạng thái logic của toán hạng đó. Nếu điều kiện toán hạng là true thì phủ định nó sẽ là false | !(A && B) là true. |

### Các toán tử gán trong Java

Dưới đây liệt kê các toán tử gán được hỗ trợ bởi Java:

[**Ví dụ**](https://vietjack.com/java/vi_du_toan_tu_gan_trong_java.jsp)

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Miêu tả** | **Ví dụ** |
| = | Toán tử gán đơn giản. Gán giá trị toán hạng bên phải cho toán hạng trái. | C = A + B sẽ gán giá trị của A + B vào cho C |
| += | Thêm giá trị toán hạng phải tới toán hạng trái và gán giá trị đó cho toán hạng trái. | C += A là tương đương với C = C + A |
| -= | Trừ đi giá trị toán hạng phải từ toán hạng trái và gán giá trị này cho toán hạng trái. | C -= A là tương đương với C = C - A |
| \*= | Nhân giá trị toán hạng phải với toán hạng trái và gán giá trị này cho toán hạng trái. | C \*= A là tương đương với C = C \* A |
| /= | Chia toán hạng trái cho toán hạng phải và gán giá trị này cho toán hạng trái. | C /= A là tương đương với C = C / A |
| %= | Lấy phần dư của phép chia toán hạng trái cho toán hạng phải và gán cho toán hạng trái. | C %= A là tương đương với C = C % A |
| <<= | Dịch trái toán hạng trái sang số vị trí là giá trị toán hạng phải. | C <<= 2 là giống như C = C << 2 |
| >>= | Dịch phải toán hạng trái sang số vị trí là giá trị toán hạng phải. | C >>= 2 là giống như C = C >> 2 |
| &= | Phép AND bit | C &= 2 là giống như C = C & 2 |
| ^= | Phép OR loại trừ bit | C ^= 2 là giống như C = C ^ 2 |
| |= | Phép OR bit. | C |= 2 là giống như C = C | 2 |

### Toán tử hỗn hợp trong Java

Ngôn ngữ Java cũng hỗ trợ một số toán tử hỗn hợp khác.

### Toán tử điều kiện (? :)

Toán tử này gồm ba toán hạng và được sử dụng để ước lượng các biểu thức quan hệ. Mục tiêu của toán tử là quyết định giá trị nào sẽ được gán cho biến. Toán tử này được viết như sau:

bien x = (bieu\_thuc) ? (giatri1 neu true) : (giatri1 neu true);

//hoac ban cung co the su dung trong cac lenh RETURN

return (bieu\_thuc) ? (giatri1 neu true) : (giatri2 neu false);

Tất nhiên, bạn không cần thiết phải sử dụng các dấu ngoặc đơn như trên. Mình viết như vậy để giúp các bạn dễ nhìn hơn. Sau đây là ví dụ minh họa toán tử điều kiện (? :) trong Java:

public class Test {

public static void main(String args[]){

int a , b;

a = 10;

b = (a == 1) ? 20: 30; //Day la vi du ve toan tu dieu kien.

System.out.println( "Gia tri cua b la : " + b );

b = (a == 10) ? 20: 30; //Day la vi du ve toan tu dieu kien.

System.out.println( "Gia tri cua b la : " + b );

}

}

Nó sẽ cho kết quả sau:

Gia tri cua b la : 30

Gia tri cua b la : 20

### Toán tử instanceof trong Java

Toán tử này chỉ được sử dụng cho các biến tham chiếu đối tượng. Toán tử kiểm tra có hay không đối tượng là một kiểu cụ thể (kiểu class hoặc kiểu interface). Toán tử instanceof được viết như sau:

( Object reference variable ) instanceof (class/interface type)

Toán tử trả về true nếu toán hạng trái là biến thể hiện của toán hạng phải. Ví dụ:

public class Test {

public static void main(String args[]){

String name = "Doan";

// Duoi day se tra ve true neu name la mot kieu String

boolean result = name instanceof String;

System.out.println( result );

}

}

Nó sẽ cho kết quả:

true

Toán tử này sẽ cũng trả về true nếu đối tượng đang được so sánh là tham số tương thích với kiểu toán hạng phải. Dưới đây là một ví dụ khác:

class Animal {}

public class Dog extends Animal {

public static void main(String args[]){

Animal a = new Dog();

boolean result = a instanceof Dog;

System.out.println( result );

}

}

Nó sẽ cho kết quả:

true

### Thứ tự ưu tiên của các toán tử trong Java:

Thứ tự ưu tiên của các toán tử xác định cách biểu thức được tính toán. Ví dụ: toán tử nhân có quyền ưu tiên hơn toán tử cộng.

Ví dụ, x = 7 + 3 \* 2; ở đây, x được gán giá trị 13, chứ không phải 20 bởi vì toán tử \* có quyền ưu tiên cao hơn toán tử +, vì thế đầu tiên nó thực hiện phép nhân 3 \* 2 và sau đó thêm với 7.

Bảng dưới đây liệt kê thứ tự ưu tiên của các toán tử. Các toán tử với quyền ưu tiên cao nhất xuất hiện trên cùng của bảng, và các toán tử có quyền ưu tiên thấp nhất thì ở bên dưới cùng của bảng. Trong một biểu thức, các toán tử có quyền ưu tiên cao nhất được tính toán đầu tiên.

|  |  |  |
| --- | --- | --- |
| **Loại** | **Toán tử** | **Thứ tự ưu tiên** |
| Postfix | () [] . (toán tử dot) | Trái sang phải |
| Unary | ++ - - ! ~ | Phải sang trái |
| Tính nhân | \* / % | Trái sang phải |
| Tính cộng | + - | Trái sang phải |
| Dịch chuyển | >> >>> << | Trái sang phải |
| Quan hệ | > >= < <= | Trái sang phải |
| Cân bằng | == != | Trái sang phải |
| Phép AND bit | & | Trái sang phải |
| Phép XOR bit | ^ | Trái sang phải |
| Phép OR bit | | | Trái sang phải |
| Phép AND logic | && | Trái sang phải |
| Phép OR logic | || | Trái sang phải |
| Điều kiện | ?: | Phải sang trái |
| Gán | = += -= \*= /= %= >>= <<= &= ^= |= | Phải sang trái |
| Dấu phảy | , | Trái sang phải |

## CÁC LỆNH ĐIỀU KHIỂN CHƯƠNG TRÌNH TRONG JAVA

Như các ngôn ngữ lập trình khác, java cung cấp các lệnh rẽ nhánh để có thể thực hiện một trong nhiều phương án khác nhau khi lập trình. Có 3 loại cấu trúc rẽ nhánh:

* Mệnh đề if
* Mệnh đề switch … case
* Toán tử điều kiện

### Cấu trúc rẽ nhánh if

* **Phân loại**

Mệnh đề if trong java được chai thành các loại sau đây:

|  |  |  |
| --- | --- | --- |
| **STT** | **Phân loại** | **Mô tả** |
| 1 | Cấu trúc rẽ nhánh if | Cấu trúc if |
| Cấu trúc if...else |
| Cấu trúc if-else nhiều nhánh |

#### Cấu trúc if

* **Khái niệm**

Lệnh if hay còn gọi là lệnh if thiếu, dùng để thực hiện một hành động mà chỉ có một trường hợp duy nhất.

* **Cú pháp**

|  |
| --- |
| if (<Biểu thức điều kiện>) {  <Các câu lệnh>  **}** |

Trong đó:

* Từ khóa if phải viết bằng chữ thường
* Kết quả của biểu thức điều kiện phải có tính **Đúng** hoặc **Sai.**
* Biểu thức luận lý phải đặt trong cặp dấu ( ).
* Các câu lệnh từ 2 lệnh trở lên thì phải đặt trong dấu { }.
* Không đặt dấu chấm phẩy sau câu lệnh if.
* **Lưu đồ**

A diagram of a diagram

Description automatically generated

**Hình 1.1**: Lưu đồ hoạt động if thiếu

* **Hoạt động**

Nếu biểu thức điều kiện có giá trị là True (Đúng) thì thực hiện các câu lệnh. Ngược lại biểu thức điều kiện có giá trị là False (Sai) thì các câu lệnh sẽ bị bỏ qua và thực hiện tiếp các lệnh còn lại của chương trình.

|  |
| --- |
| **VD 1:** Viết chương trình in ra thông báo “Ban da du tuoi vi thanh vien.” nếu Tuổi >= 18 |
| public class **IfThieu** {      public static void **main**(**String**[] args){          int Tuoi = 20;          if(Tuoi >= 18) {  **System**.out.**println**("Ban da du tuoi vi thanh vien.");          }      }  } |
| **Hiển thị:** |
| Ban da du tuoi vi thanh vien. |

#### Cấu trúc if else

* **Khái niệm**

Lệnh if else hay còn gọi là lệnh if đủ, dùng để thực hiện một hành động mà có hai trường hợp.

* **Cú pháp**

|  |
| --- |
| if (<Biểu thức điều kiện>)  <Các câu lệnh của if>  else  <Các câu lệnh của else> |

Trong đó:

* Từ khóa if else phải viết bằng chữ thường
* Kết quả của biểu thức điều kiện phải có tính **Đúng** hoặc **Sai.**
* Biểu thức luận lý phải đặt trong cặp dấu ( ).
* Các câu lệnh từ 2 lệnh trở lên thì phải đặt trong dấu { }.
* Không đặt dấu chấm phẩy sau câu lệnh if.
* **Lưu đồ**

A diagram of a diagram

Description automatically generated

**Hình 1.2**: Lưu đồ hoạt động if đủ

**Hoạt động:**

Nếu biểu thức điều kiện mang giá trị True (Đúng) thì thực hiện các câu lệnh của if, ngược lại nếu biểu thức điều kiện mang giá trị False (sai) thì thực hiện các câu lệnh của else.

|  |
| --- |
| **VD 2:** Viết chương trình in ra thông báo. Nếu Tuổi >= 18 “Ban da du tuoi vi thanh vien.” Ngược lại, in ra thông báo “Ban chua du tuoi vi thanh vien. ” |
| public class **IfDu** {      public static void **main**(**String**[] args){          int Tuoi = 10;          if(Tuoi >= 18) {  **System**.out.**println**("Ban da du tuoi vi thanh vien");          }          else{  **System**.out.**println**("Ban chua du tuoi vi thanh vien");          }      }  } |
| **Hiển thị:** |
| Ban chua du tuoi vi thanh vien. |

#### Cấu trúc if-else nhiều nhánh

Lệnh if-else nhiều nhánh hay còn gọi là lệnh else if, dùng để thực hiện một hành động có từ ba trường hợp trở lên.

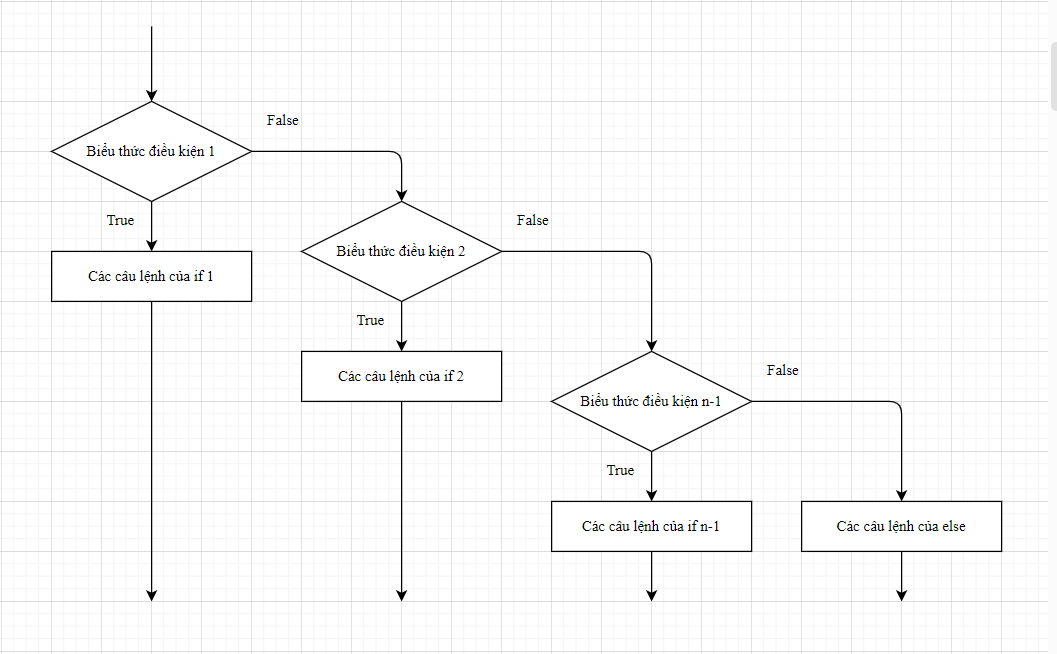
**Cú pháp:** Giả sử có n trường hợp (n>=3)

|  |
| --- |
| if (<Biểu thức điều kiện 1>)  <Các câu lệnh của if 1>  else if (<Biểu thức điều kiện 2>)  <Các câu lệnh của if 2>  ...  else if (<Biểu thức điều kiện n-1>)  <Các câu lệnh của if n-1>  else  <Các câu lệnh của n> |

Trong đó:

* Từ khóa if, else if, else phải viết bằng chữ thường
* Kết quả của biểu thức điều kiện phải có tính **Đúng** hoặc **Sai.**
* Biểu thức luận lý phải đặt trong cặp dấu ( ).
* Các câu lệnh từ 2 lệnh trở lên thì phải đặt trong dấu { }.
* Không đặt dấu chấm phẩy sau câu lệnh if.

**Lưu đồ:**



**Hình 1.3**: Lưu đồ hoạt động if-else nhiều nhánh

* **Hoạt động:**

Nếu biểu thức điều kiện mang giá trị True (Đúng) thì thực hiện các câu lệnh của if, ngược lại nếu biểu thức điều kiện mang giá trị False (sai) thì thực hiện các câu lệnh của else.

|  |
| --- |
| **VD 3:** Viết chương trình in ra xếp loại học tập của học sinh đó. |
| public class **IfElse** {      public static void **main**(**String**[] args){          int diem=9;            if((diem>10) || (diem<0)){  **System**.out.**println**("Diem nhap vao khong hop le!");          } else if(diem>=9){  **System**.out.**println**("Xuat sac!");          } else if(diem>=6.5){  **System**.out.**println**("Kha!");          } else if(diem>=5){  **System**.out.**println**("Trung binh!");          } else if(diem>=3.5){  **System**.out.**println**("Yeu!");          } else {  **System**.out.**println**("Kem!");          }      }  } |
| **Hiển thị:** |
| Xuat sac! |

### Cấu trúc rẽ nhánh switch

Cấu trúc rẽ nhánh switch là một cấu trúc điều khiển và rẽ nhánh hoàn toàn có thể được thay thế bằng cấu trúc if-else nhiều nhánh. Tuy nhiên, nó cũng có mặt hạn chế là giá trị của biểu thức điều kiện phải là giá trị hằng nguyên (có giá trị cụ thể).

Việc sử dụng switch sẽ giúp code của chúng ta dễ viết, dễ đọc và có hiệu năng tốt hơn so với sử dụng if … else trong trường hợp có nhiều điều kiện có thể xảy ra.

Câu lệnh switch gồm 2 loại

* Lệnh switch case hay còn gọi là switch thiếu.
* Lệnh switch case default hay còn gọi là switch đủ.

|  |  |
| --- | --- |
| **Tên** | **Cú pháp** |
| switch thiếu | switch (Biểu thức điều kiện) {  case <Giá trị 1> :  <Khối lệnh 1>  break;  case <Giá trị 2> :  <Khối lệnh 2>  break;  case <Giá trị n> :  <Khối lệnh n>  break;  } |
| switch đủ | switch (Biểu thức điều kiện) {  case <Giá trị 1> :  <Khối lệnh 1>  break;  case <Giá trị 2> :  <Khối lệnh 2>  break;  case <Giá trị n-1> :  <Khối lệnh n-1>  break;  default :  <Khối lệnh n>  break;  } |

Trong đó:

* Từ khóa switch, case, default, break phải viết bằng chữ thường.
* Biểu thức bắt buộc là giá trị hằng, có thể là biểu thức nhưng kết quả cần là hằng số nguyên (char, int, long,…)
* Khối lệnh 1, 2… n có thể gồm nhiều câu lệnh, nhưng không cần đặt trong cặp dấu { }
* **Lưu đồ:**

Diagram, schematic

Description automatically generated

**Hình 1.4**: Lưu đồ hoạt động swith case

Diagram, schematic

Description automatically generated

**Hình 1.5**: Lưu đồ hoạt động swith case default

* **Hoạt động:**

Biểu thức điều kiện sẽ được so sánh lần lượt với giá trị của các case:

* Nếu có 1 case nào đó khớp giá trị thì khối lệnh tương ứng của case đó sẽ được thực hiện cho tới khi gặp lệnh break. Do đó, nếu chúng ta không sử dụng break thì tất cả các case kể từ case khớp giá trị đều được thực hiện.
* Đối với cấu trúc switch thiếu, nếu không có 1 case nào khớp giá trị biểu thức điều kiện thì sẽ không có một khối lệnh nào được thực hiện.
* Đối với cấu trúc switch đủ, nếu không có 1 case nào khớp giá trị biểu thức điều kiện thì khối lệnh của default sẽ được thực hiện.

|  |
| --- |
| **VD 4:** Viết chương trình nhập vào tháng và in ra quý. **(Sử dụng switch thiếu).** |
| **Giải thuật:** Nhập vào giá trị tháng, kiểm tra xem tháng có hợp lệ (trong khoảng 1 đến 12). Nếu hợp lệ in ra quý tương ứng (13: quý 1, 46: quý 2, 79: quý 3, 1012: quý 4). |
| import **java**.**util**.**Scanner**;  public class **Main** {      public static void **main**(**String**[] args) {  **Scanner** scanner = new **Scanner**(**System**.in);  **System**.out.**print**("Nhap vao thang: ");          int thang = scanner.**nextInt**();          if (thang > 0 && thang <= 12) {              switch (thang) {                  case 1:                  case 2:                  case 3:  **System**.out.**println**("Quy 1.");                      break;                  case 4:                  case 5:                  case 6:  **System**.out.**println**("Quy 2.");                      break;                  case 7:                  case 8:                  case 9:  **System**.out.**println**("Quy 3.");                      break;                  case 10:                  case 11:                  case 12:  **System**.out.**println**("Quy 4.");                          break;              }          }          else {  **System**.out.**println**("Thang khong hop le.");          }      }  } |
| **Hiển thị:** |
| Nhap vao thang: 5  Quy 2. |

|  |
| --- |
| **VD 5:** Viết chương trình nhập vào tháng và in ra quý. (**Sử dụng switch đủ).** |
| **Giải thuật:** Nhập vào giá trị tháng, in ra quý tương ứng (13: quý 1, 46: quý 2, 79: quý 3, 1012: quý 4). Nếu không có case nào khớp thì hiển thị ra ("Nhap Thang 1 -> 12."). |
| import **java**.**util**.**Scanner**;  public class **Main** {      public static void **main**(**String**[] args) {  **Scanner** scanner = new **Scanner**(**System**.in);  **System**.out.**print**("Nhap vao thang: ");          int thang = scanner.**nextInt**();          if (thang > 0 && thang <= 12) {              switch (thang) {                  case 1:                  case 2:                  case 3:  **System**.out.**println**("Quy 1.");                      break;                  case 4:                  case 5:                  case 6:  **System**.out.**println**("Quy 2.");                      break;                  case 7:                  case 8:                  case 9:  **System**.out.**println**("Quy 3.");                      break;                  default:  **System**.out.**println**("Quy 4.");                      break;              }          }          else {  **System**.out.**println**("Thang khong hop le.");          }      }  } |
| **Hiển thị:** |
| Nhap vao thang: 11  Quy 4. |

### Toán tử điều kiện (? :) trong Java

Ngoài các lệnh kể trên, ngôn ngữ Java còn có một loại toán tử điều kiện giúp bạn kiểm tra nhanh các điều kiện và thực hiện phép gán giá trị cho một biến một cách rất nhanh chóng.

Toán tử này gồm ba toán hạng và được sử dụng để ước lượng các biểu thức quan hệ. Mục tiêu của toán tử là quyết định giá trị nào sẽ được gán cho biến. Toán tử này được viết như sau:

**Cú pháp:**

|  |
| --- |
| <Biểu thức 1> ? <Biểu thức 2> : <Biểu thức 3> |

**Hoạt động:**

Biểu thức 1 nhận giá trị đúng thì toán tử trả về giá trị là Biểu thức 2, ngược lại toán tử trả về giá trị là Biểu thức 3

|  |
| --- |
| **VD 6:** Toán tử điều kiện. |
| public class **Test** {     public static void **main**(**String** args[]){        int a , b;        a = 10;        b = (a == 1) ? 20: 30; *//Day la vi du ve toan tu dieu kien.*  **System**.out.**println**( "Gia tri cua b la : " +  b );        b = (a == 10) ? 20: 30; *//Day la vi du ve toan tu dieu kien.*  **System**.out.**println**( "Gia tri cua b la : " + b );     }  } |
| **Hiển thị:** |
| Gia tri cua b la : 30  Gia tri cua b la : 20 |

## VÒNG LẶP TRONG JAVA

### Vòng lặp trong Java

Có thể có một tình huống khi chúng ta cần thực hiện một khối code một vài lần, điều này có thể được xem như một vòng lặp.

Java có 3 kỹ thuật lặp linh động. Bạn có thể sử dụng một trong ba vòng lặp sau:

* Vòng lặp while
* Vòng lặp do…while
* Vòng lặp for

Trong Java 5, vòng lặp **foreach (hay enhanced for)** đã được giới thiệu. Vòng lặp này được sử dụng chủ yếu cho các mảng.

|  |  |
| --- | --- |
| **Loại vòng lặp** | **Miêu tả** |
| **Vòng lặp for** | Một vòng lặp for cho phép bạn viết một vòng lặp một cách hiệu quả để cần thực thi một số lần lặp cụ thể nào đó. |
| **Vòng lặp while** | Một vòng lặp while là một cấu trúc điều khiển cho phép bạn lặp đi lặp lại một tác vụ một số lần nào đó. Với kiểu vòng lặp này, chương trình sẽ kiểm tra điều kiện trước khi thực thi phần thân vòng lặp. |
| **Vòng lặp do...while** | Một vòng lặp do … while là tương tự như vòng lặp while, ngoại trừ rằng phần thân của vòng lặp do…while được bảo đảm thực thi ít nhất một lần. Nói cách khác, vòng lặp do ... while thực hiện phần thân vòng lặp trước khi kiểm tra điều kiện. |
| **Vòng lặp foreach** | Trong Java 5, vòng lặp foreach (một tên gọi khác là enhanced for) đã được giới thiệu. Nó được sử dụng chủ yếu với các mảng, các ArrayList, ... |

#### Vòng lặp for

Vòng lặp for thường được sử dụng để giải quyết các hành động (khối code) có sự lặp đi lặp lại nhiều lần với số lần đã biết trước.

**Cú pháp:**

|  |
| --- |
| for (Khởi tạo giá trị biến lặp; Điều kiện lặp; Cập nhật biến lặp)  {  *// Khối lệnh lặp*  } |

Trong đó:

Vòng lặp for được định nghĩa bởi từ khóa for và được chia làm 3 phần chính, mỗi phần được ngăn cách bởi dấu chấm phẩy(;)

* **Phần khởi tạo biến lặp**: Kởi tạo biến vòng lặp có thể được khai báo và khởi tạo giá trị ngay bên trong phần khởi tạo của vòng lặp for. Phần khởi tạo biến được thực thi đầu tiên và chỉ thực thi 1 lần duy nhất trong vòng lặp for.
* **Điều kiện lặp:** Khối lệnh của vòng lặp for sẽ được thực hiện nếu **Biểu thức điều kiện lặp** cho giá trị đúng. Vòng lặp for kiểm tra biểu thức điều kiện trước khi thực hiện khối lệnh.
* **Cập nhật biến lặp:** Phần này sẽ được thực thi cuối mỗi lần lặp, sau khi khối lệnh của vòng lặp for được thực thi. Phần này thường chịu trách nhiệm thay đổi giá trị biến vòng lặp được sử dụng trong biểu thức điều kiện (nhằm tránh tình trạng lặp vô hạn). Sau khi thực thi xong phần cập nhật biến vòng lặp, chương trình quay trở lại đánh giá biểu thức điều kiện của vòng lặp **for** và cứ như thế đến khi biểu thức điều kiện lặp là sai.

**Chú ý:**

* Có dấu chấm phẩy sau lệnh for (i = 0; i <3; i++); → vòng lặp for sẽ không chạy đúng chương trình.
* Trong vòng lặp for có sử dụng từ 2 lệnh trở lên, nhớ sử dụng cặp ngoặc { } để bọc các lệnh đó lại.

**Lưu đồ:**

A picture containing text, diagram, line, plan

Description automatically generated

**Bước 1**: Khởi tạo giá trị biến lặp (1 lần duy nhất)

**Bước 2**: Kiểm tra điều kiện lặp, nếu sai chuyển sang Bước 5, nếu đúng chuyển sang Bước 3.

**Bước 3**: Thực hiện nội dung trong thân vòng lặp

**Bước 4**: Cập nhật giá trị lặp và quay lại Bước 2

**Bước 5**: Kết thúc vòng lặp.

|  |
| --- |
| **VD 6:** Toán tử điều kiện. |
|  |
| **Hiển thị:** |
|  |

### Các lệnh điều khiển vòng lặp trong Java

Câu hỏi đặt ra là nếu giả sử trong một vòng lặp tại một điều kiện cụ thể nào đó, bạn muốn dừng thực thi hoặc tiếp tục thực thi vòng lặp đó, bạn sử dụng cách nào. Cũng như các ngôn ngữ lập trình khác, Java hỗ trợ các lệnh giúp bạn điều khiển và kiểm soát vòng lặp.

Bảng dưới liệt kê các lệnh được sử dụng để điều khiển vòng lặp mà Ngôn ngữ lập trình Java hỗ trợ:

|  |  |
| --- | --- |
| **Lệnh** | **Miêu tả** |
| **Lệnh** [**break**](https://vietjack.com/java/lenh_break_trong_java.jsp) | Từ khóa break được sử dụng để dừng toàn bộ vòng lặp. Từ khóa break phải được sử dụng bên trong bất kỳ vòng lặp nào hoặc một lệnh switch. |
| **Lệnh** [**continue**](https://vietjack.com/java/lenh_continue_trong_java.jsp) | Từ khóa continue có thể được sử dụng trong bất kỳ cấu trúc điều khiển vòng lặp nào. Nó làm cho vòng lặp ngay lập tức tiếp tục tiến trình lặp tiếp theo của vòng lặp. |

## NUMBER TRONG JAVA

Thường thì, khi làm việc với Number, chúng ta sử dụng các kiểu dữ liệu gốc như byte, int, long, double, …

**Ví dụ:**

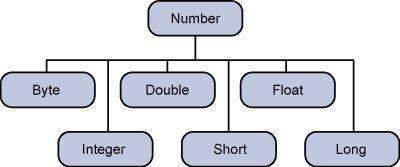
int i = 5000;

float gpa = 13.65;

byte mask = 0xaf;

Tuy nhiên, khi mở rộng, chúng ta cũng gặp các tình huống mà chúng ta cần sử dụng các đối tượng thay cho các kiểu dữ liệu gốc. Để thực hiện điều này, Java cung cấp các lớp wrapper (lớp bọc) cho mỗi kiểu dữ liệu gốc.

Tất cả các lớp wrapper (Integer, Long, Byte, Double, Float, Short) là lớp phụ của lớp trừu tượng Number.



Sự bao bọc này được chăm sóc bởi bộ biên dịch, tiến trình này được gọi là boxing. Vì thế khi một kiểu dữ liệu gốc được sử dụng khi một đối tượng được yêu cầu, bộ biên dịch "đóng hộp" kiểu dữ liệu gốc trong lớp wrapper của nó. Tương tự, bộ biên dịch cũng "mở hộp" đối tượng thành một kiểu dữ liệu gốc. *Number* là một phần của java.lang package.

Dưới đây là ví dụ của box và unbox:

public class Test{

public static void main(String args[]){

Integer x = 5; // dong hop int thanh doi tuong Integer

x = x + 10; // mo hop Integer thanh int

System.out.println(x);

}

}

Nó sẽ cho kết quả sau:

15

Khi x được gán các giá trị integer, bộ biên dịch "đóng hộp" giá trị integer này bởi vì x là đối tượng integer. Sau đó, x được "mở hộp" để mà chúng có thể được cộng thêm các integer.

### Các phương thức của lớp Number trong Java

Bảng dưới liệt kê các phương thức mà tất cả các lớp phụ của lớp Number thực thi:

|  |  |
| --- | --- |
| **STT** | **Phương thức và miêu tả** |
| 1 | [**xxxValue() trong Java**](https://vietjack.com/java/number_xxxvalue_trong_java.jsp)  Biến đổi giá trị của đối tượng Number này thành kiểu dữ liệu xxx mà được trả về từ phương thức |
| 2 | [**compareTo() trong Java**](https://vietjack.com/java/number_compareto_trong_java.jsp)  So sánh đối tượng Number này với tham số |
| 3 | [**equals() trong Java**](https://vietjack.com/java/number_equals_trong_java.jsp)  Kiểm tra có hay không đối tượng Number này là bằng với tham số |
| 4 | [**valueOf() trong Java**](https://vietjack.com/java/number_valueof_trong_java.jsp)  Trả về một đối tượng integer giữ giá trị gốc đã xác định |
| 5 | [**toString() trong Java**](https://vietjack.com/java/number_tostring_trong_java.jsp)  Trả về đối tượng String biểu diễn giá trị Integer đã xác định |
| 6 | [**parseInt() trong Java**](https://vietjack.com/java/number_parseint_trong_java.jsp)  Phương thức này được sử dụng để nhận kiểu dữ liệu gốc của một String cụ thể |
| 7 | [**abs() trong Java**](https://vietjack.com/java/number_abs_trong_java.jsp)  Trả về giá trị tuyệt đối của tham số |
| 8 | [**ceil() trong Java**](https://vietjack.com/java/number_ceil_trong_java.jsp)  Trả về số integer nhỏ nhất mà lớn hơn hoặc bằng tham số. Giá trị trả về ở dạng double. |
| 9 | [**floor() trong Java**](https://vietjack.com/java/number_floor_trong_java.jsp)  Trả về số integer lớn nhất mà nhỏ hơn hoặc bằng tham số. Giá trị trả về ở dạng double. |
| 10 | [**rint() trong Java**](https://vietjack.com/java/number_rint_trong_java.jsp)  Làm tròn số, trả về giá trị integer gần nhất với giá trị tham số. Giá trị trả về ở dạng double. |
| 11 | [**round() trong Java**](https://vietjack.com/java/number_round_trong_java.jsp)  Trả về long hoặc int gần nhất, như kiểu dữ liệu được chỉ dẫn bởi phương thức, với tham số. |
| 12 | [**min() trong Java**](https://vietjack.com/java/number_min_trong_java.jsp)  Trả về số nhỏ nhất trong hai tham số |
| 13 | [**max() trong Java**](https://vietjack.com/java/number_max_trong_java.jsp)  Trả về số lớn nhất trong hai tham số |
| 14 | [**exp() trong Java**](https://vietjack.com/java/number_exp_trong_java.jsp)  Hàm mũ cơ số e. |
| 15 | [**log() trong Java**](https://vietjack.com/java/number_log_trong_java.jsp)  Hàm logarit cơ số e |
| 16 | [**pow() trong Java**](https://vietjack.com/java/number_pow_trong_java.jsp)  Trả về giá trị hàm mũ với cơ số là tham số đầu tiên và mũ là tham số thứ hai |
| 17 | [**sqrt() trong Java**](https://vietjack.com/java/number_sqrt_trong_java.jsp)  Trả về căn bậc hai của tham số |
| 18 | [**sin() trong Java**](https://vietjack.com/java/number_sin_trong_java.jsp)  Trả về sin của giá trị double đã cho. |
| 19 | [**cos() trong Java**](https://vietjack.com/java/number_cos_trong_java.jsp)  Trả về cos của giá trị double đã cho. |
| 20 | [**tan() trong Java**](https://vietjack.com/java/number_tan_trong_java.jsp)  Trả về tan của giá trị double đã cho. |
| 21 | [**asin() trong Java**](https://vietjack.com/java/number_asin_trong_java.jsp)  Trả về arcsin của giá trị double đã cho. |
| 22 | [**acos() trong Java**](https://vietjack.com/java/number_acos_trong_java.jsp)  Trả về arccos của giá trị double đã cho |
| 23 | [**atan() trong Java**](https://vietjack.com/java/number_atan_trong_java.jsp)  Trả về arctan của giá trị double đã cho |
| 24 | [**atan2() trong Java**](https://vietjack.com/java/number_atan2_trong_java.jsp)  Biến đổi tọa độ (x, y) thành tọa độ cực (r, theta) và trả về theta. |
| 25 | [**toDegrees() trong Java**](https://vietjack.com/java/number_todegrees_trong_java.jsp)  Biến đổi tham số thành giá trị độ đo góc |
| 26 | [**toRadians() trong Java**](https://vietjack.com/java/number_toradians_trong_java.jsp)  Biến đổi tham số thành radian. |
| 27 | [**random() trong Java**](https://vietjack.com/java/number_random_trong_java.jsp)  Trả về một số ngẫu nhiên |

## CHARACTER TRONG JAVA

Thường thì, khi làm việc với các ký tự, chúng ta sử dụng kiểu dữ liệu char gốc.

**Ví dụ:**

char ch = 'a';

// Ma Unicode cho ky tu Greek omega dang chu hoa

char uniChar = '\u039A';

// Mot mang ky tu

char[] charArray ={ 'a', 'b', 'c', 'd', 'e' };

Tuy nhiên, khi mở rộng làm việc, chúng ta thường gặp các tính huống cần sử dụng các đối tượng thay vì các kiểu dữ liệu gốc. Để thực hiện điều này, Java cung cấp lớp wrapper là **Charater** với kiểu dữ liệu char gốc.

Lớp Character có một số phương thức hữu ích (ví dụ: static) để thao tác với các ký tự. Bạn có thể tạo một đối tượng Character với Character constructor.

Character ch = new Character('a');

Bộ biên dịch Java sẽ cũng tạo một đối tượng Character cho bạn. Ví dụ, nếu bạn truyền một dữ liệu char gốc vào trong một phương thức mà cho một đối tượng, bộ biên dịch tự động biến đổi kiểu char tới một Character cho bạn. Tính năng này được gọi là autobox hoặc unbox, nếu sự biến đổi này theo hướng ngược lại.

**Ví** dụ**:**

// O day la kieu char goc 'a'

// duoc dong hop thanh doi tuong Character la ch

Character ch = 'a';

// Tai day kieu goc 'x' duoc dong hop cho phuong thuc test,

// gia tri tra ve duoc mo hop cho ky tu 'c'

char c = test('x');

### Các ký tự ngắt văn bản trong Java

Một ký tự được đặt trước bởi một dấu gạch chéo ngược (\) là một ký tự ngắt và có ý nghĩa đặc biệt với bộ biên dịch.

Ký tự dòng mới (\n) thường được sử dụng trong bài học có trong lệnh System.out.println() để có được dòng tiếp theo sau khi chuỗi được in.

Bảng dưới liệt kê các ký tự ngắt trong Java:

|  |  |
| --- | --- |
| **Ký tự ngắt** | **Miêu tả** |
| \t | Chèn một tab vào văn bản tại điểm này |
| \b | Chèn một backspace vào văn bản tại điểm này |
| \n | Chèn một dòng mới vào văn bản tại điểm này |
| \r | Chèn một carriage return vào văn bản tại điểm này |
| \f | Chèn một form feed vào văn bản tại điểm này |
| \' | Chèn một dấu trích dẫn đơn vào văn bản tại điểm này |
| \" | Chèn một dấu trích dẫn kép vào văn bản tại điểm này |
| \\ | Chèn một ký tự dấu chéo ngược vào văn bản tại điểm này |

Khi gặp một ký tự ngắt trong một lệnh print, bộ biên dịch thông dịch nó cho phù hợp.

**Ví dụ:**

Nếu bạn muốn đặt các trích dẫn bên trong các trích dẫn, bạn phải sử dụng ký tự ngắt, \", trong trích dẫn bên trong.

public class Test {

public static void main(String args[]) {

System.out.println("Co ta noi \"Hello!\" voi toi.");

}

}

Nó sẽ cho kết quả:

Co ta noi "Hello!" voi toi.

### Các phương thức của lớp Character trong Java

Dưới đây liệt kê các phương thức quan trọng mà tất cả lớp phụ của lớp Character trong Java thực thi:

|  |  |
| --- | --- |
| **STT** | **Phương thức và Miêu tả** |
| 1 | [**isLetter() trong Java**](https://vietjack.com/java/character_isletter_trong_java.jsp)  Kiểm tra có hay không giá trị char đã cho là một chữ cái |
| 2 | [**isDigit() trong Java**](https://vietjack.com/java/character_isdigit_trong_java.jsp)  Xác định có hay không giá trị char đã cho là một digit |
| 3 | [**isWhitespace() trong Java**](https://vietjack.com/java/character_iswhitespace_trong_java.jsp)  Xác định có hay không giá trị char đã cho là một khoảng trắng |
| 4 | [**isUpperCase() trong Java**](https://vietjack.com/java/character_isuppercase_trong_java.jsp)  Xác định có hay không giá trị char đã cho là chữ hoa |
| 5 | [**isLowerCase() trong Java**](https://vietjack.com/java/character_islowercase_trong_java.jsp)  Xác định có hay không giá trị char đã cho là chữ thường |
| 6 | [**toUpperCase() trong Java**](https://vietjack.com/java/character_touppercase_trong_java.jsp)  Trả về form dạng chữ hoa của giá trị char đã cho |
| 7 | [**toLowerCase() trong Java**](https://vietjack.com/java/character_tolowercase_trong_java.jsp)  Trả về form dạng chữ thường của giá trị char đã cho |
| 8 | [**toString() trong Java**](https://vietjack.com/java/character_tostring_trong_java.jsp)  Trả về một đối tượng String biểu diễn giá trị ký tự đã cho, mà là, một chuỗi gồm một ký tự |

Để có danh sách đầy đủ các phương thức, bạn tham khảo java.lang.Character API.

# HƯỚNG ĐỐI TƯỢNG (OOP) TRONG JAVA

**[Khái niệm hướng đối tượng OOP trong Java]**Chương này, chúng ta sẽ tìm hiểu các khái niệm cơ bản về Hướng đối tượng (OOP). Lập trình hướng đối tượng bao gồm nhiều khái niệm như tính kế thừa, gắn kết dữ liệu (Data Binding), tính đa hình, …

**Simula** được xem như là ngôn ngữ lập trình hướng đối tượng đầu tiên. Là một hệ lập trình mà ở đó mọi thứ được biểu diễn dưới dạng một đối tượng, và được biết đến như là ngôn ngữ hướng đối tượng thực sự. Tuy nhiên, **Smalltalk** mới được xem như là ngôn ngữ lập trình hướng đối tượng thực sự đầu tiên.

## LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

**Object** (đối tượng) nghĩa là một thực thể trong thế giới thực, chẳng hạn như bàn, quả bóng, con bò, … Lập trình hướng đối tượng là một phương pháp để thiết kế một chương trình bởi sử dụng các lớp và các đối tượng. Nó làm đơn giản hóa việc duy trì và phát triển phần mềm bằng việc cung cấp một số khái niệm:

* **Đối tượng**: Một thực thể có trạng thái và hành vi. Ví dụ như xe đạp, bàn, ghế, … Nó có thể mang tính vật lý hoặc logic.
* **Lớp**: Một tập hợp các đối tượng. Nó là một thực thể logic.
* **Tính kế thừa**: Khi một đối tượng đạt được các thuộc tính và các hành vi của đối tượng cha, thì đó là tính kế thừa. Điều này làm tăng tính tái sử dụng cho code. Nó được sử dụng để đạt được tính đa hình tại runtime.
* **Tính đa hình**: Khi một tác vụ được thực hiện theo nhiều cách khác nhau được gọi là tính đa hình. Ví dụ: như vẽ hình chữ nhật hoặc hình tam giác, … Trong Java, chúng ta sử dụng nạp chồng phương thức (method overloading) và ghi đè phương thức (method overriding) để có tính đa hình. Một ví dụ khác: con mèo kêu meooo, còn chú chó thì sủa goooo.
* **Tính trừu tượng**: Đó là ẩn các chi tiết nội tại và hiển thị tính năng. Ví dụ, với cuộc gọi điện thoại, chúng ta không biết tiến trình xử lý nội tại là như thế nào. Trong Java, chúng là sử dụng lớp abstract và abstract interface để có tính trừu tượng.
* **Tính bao đóng**: Đó là gắn kết code và dữ liệu cùng với nhau vào trong một đơn vị unit đơn. Ví dụ: có thể bạn đã biết đến viên thuốc con nhộng (hay đơn giản hơn là gói bột giặt), các viên thuốc (hạt bột giặt) khác nhau được đóng gói.

Một lớp Java là một ví dụ về tính bao đóng. Java Bean là lớp bao đóng thực sự, vì tất cả thành viên dữ liệu là private.

### Lợi thế của OOP khi so với ngôn ngữ lập trình hướng thủ tục (procedure-oriented)

OOP giúp việc thiết kế, phát triển và bảo trì dễ dàng hơn trong khi với lập trình hướng thủ tục thì việc quản lý code là khá khó khăn nếu lượng code tăng lên. Điều này làm tăng hiệu quả có quá trình phát triển phần mềm.

OOP cung cấp Data Hiding (ẩn dữ liệu) trong khi đó trong hướng thủ tục một dữ liệu toàn cục có thể được truy cập từ bất cứ đâu.

OOP cung cấp cho bạn khả năng để mô phỏng các sự kiện trong thế giới thực một cách hiệu quả hơn. Chúng ta có thể cung cấp giải pháp cho các vấn đề trong thế giới thực nếu chúng ta sử dụng Lập trình hướng đối tượng.

### Điểm khác nhau giữa OOP và OBP (Object-base programming)

Lập trình dựa trên đối tượng có tất cả đặc điểm của OOP ngoại trừ tính kế thừa. JavaScript và VBScript là các ví dụ về các ngôn ngữ lập trình dựa trên đối tượng.

## ĐỐI TƯỢNG VÀ LỚP (CLASS) TRONG JAVA

Trong chương này, chúng ta sẽ tìm hiểu về lớp (class) và đối tượng (object) trong Java. Trong kỹ thuật lập trình hướng đối tượng, chúng ta thiết kế một chương trình bởi sử dụng các lớp và các đối tượng.

* **Object -** Đối tượng là thực thể mang tính vật lý cũng như mang tính logic, trong khi lớp chỉ là thực thể logic. Đối tượng có các trạng thái và các hành vi. Ví dụ: Một dog có trạng thái là color, name, breed (dòng dõi) và cũng có các hành vi: Wag (vẫy đuôi), bark (sủa), eat (ăn). Một đối tượng là một instance (ví dụ,trường hợp) của một lớp.
* **Class -** Một lớp là một nhóm các đối tượng mà có các thuộc tính chung. Lớp là một Template hoặc bản thiết kế từ đó đối tượng được tạo.

### Đối tượng trong Java

Đó là một thực thể có trạng thái và hành vi, ví dụ như bàn, ghế, xe con, mèo, … Nó có thể mang tính vật lý hoặc logic. Ví dụ về logic đó là Banking system.

Một đối tượng có ba đặc trưng sau:

* **Trạng thái**: biểu diễn dữ liệu (giá trị) của một đối tượng.
* **Hành vi**: biểu diễn hành vi (tính năng) của một đối tượng như gửi tiền vào, rút tiền ra, …
* **Nhận diện**: việc nhận diện đối tượng được triển khai thông qua một ID duy nhất. Giá trị của ID là không thể nhìn thấy với người dùng bên ngoài. Nhưng nó được sử dụng nội tại bởi JVM để nhận diện mỗi đối tượng một cách duy nhất.

Ví dụ: Bút là một đối tượng. Nó có tên là Thiên Long, có màu trắng, … được xem như là trạng thái của nó. Nó được sử dụng để viết, do đó viết là hành vi của nó.

Đối tượng là sự thể hiện (Instance) của một lớp. Lớp là một Template hoặc bản thiết kế từ đó đối tượng được tạo. Vì thế đối tượng là Instance (kết quả) của một lớp.

### Lớp trong Java

Một lớp là một nhóm các đối tượng mà có các thuộc tính chung. Lớp là một Template hoặc bản thiết kế từ đó đối tượng được tạo. Một lớp trong Java có thể bao gồm:

* Thành viên dữ liệu
* Phương thức
* Constructor
* Block
* Lớp và Interface

**Cú pháp để khai báo một lớp**

class ten\_lop{

thanh\_vien\_du\_lieu;

phuong\_thuc;

}

**Ví dụ đơn giản về Lớp và Đối tượng trong Java**

Trong ví dụ này, chúng ta tạo một lớp Student có hai thành viên dữ liệu là id và name. Chúng ta đang tạo đối tượng của lớp Student bởi từ khóa new và in giá trị đối tượng.

class Student1{

int id; //thanh vien du lieu (cung la bien instance)

String name; //thanh vien du lieu (cung la bien instance)

public static void main(String args[]){

Student1 s1=new Student1(); //tao mot doi tuong Student

System.out.println(s1.id);

System.out.println(s1.name);

}

}

Một lớp có thể chứa bất kỳ loại biến sau:

* **Biến Local:**Các biến được định nghĩa bên trong các phương thức, constructor hoặc block code được gọi là biến Local. Biến này sẽ được khai báo và khởi tạo bên trong phương thức và biến này sẽ bị hủy khi phương thức đã hoàn thành.
* **Biến Instance:**Các biến instance là các biến trong một lớp nhưng ở bên ngoài bất kỳ phương thức nào. Những biến này được khởi tạo khi lớp này được tải. Các biến instance có thể được truy cập từ bên trong bất kỳ phương thức, constructor hoặc khối nào của lớp cụ thể đó.
* **Biến Class:**Các biến class là các biến được khai báo với một lớp, bên ngoài bất kỳ phương thức nào, với từ khóa static.

### Phương thức trong Java

Trong Java, một phương thức là khá giống hàm, được sử dụng để trưng bày hành vi của một đối tượng. Phương thức giúp code tăng tính tái sử dụng và tối ưu hóa code.

Từ khóa new được sử dụng để cấp phát bộ nhớ tại runtime.

### Constructor trong Java

Khi bàn luận về các lớp, một trong những chủ đề quan trọng là các constructor. Mỗi lớp có một constructor. Nếu chúng ta không viết một constructor một cách rõ ràng cho một lớp thì bộ biên dịch Java xây dựng một constructor mặc định cho lớp đó.

Mỗi khi một đối tượng mới được tạo ra, ít nhất một constructor sẽ được gọi. Quy tắc chính của các constructor là chúng có cùng tên như lớp đó. Một lớp có thể có nhiều hơn một constructor.

Sau đây là ví dụ về một constructor:

public class Xecon{

public Xecon(){

}

public Xecon(String ten){

// Contructor nay co mot tham so la *ten*.

}

}

Java cũng hỗ trợ [**Lớp Singleton trong Java**](https://vietjack.com/java/lop_singleton_trong_java.jsp), ở đây bạn sẽ có thể tạo chỉ một instance của một lớp.

### Tạo một đối tượng trong Java:

Như đã đề cập trước đó, một lớp cung cấp bản thiết kế cho các đối tượng. Vì thế, về cơ bản, một đối tượng được tạo từ một lớp. Trong Java, từ khóa new được sử dụng để tạo một đối tượng mới.

Có ba bước khi tạo một đối tượng từ một lớp:

* **Khai báo:**Một khai báo biến với một tên biến với một loại đối tượng.
* **Cài đặt:**Từ khóa new được sử dụng để tạo đối tượng
* **Khởi tạo:**Từ khóa new được theo sau bởi một lời gọi một constructor. Gọi hàm này khởi tạo đối tượng mới.

Dưới đây là ví dụ về tạo một đối tượng:

public class Xecon{

public Xecon(String ten){

// Contructor nay co mot tham so la *ten*.

System.out.println("Ten xe la :" + ten );

}

public static void main(String []args){

// Lenh sau se tao mot doi tuong la Xecuatoi

Xecon Xecuatoi = new Xecon( "Toyota" );

}

}

Nếu chúng ta biên dịch và chạy chương trình, nó sẽ cho kết quả sau:

Ten xe la :Toyota

### Truy cập các biến instance và các phương thức trong Java

Các biến instance và các phương thức được truy cập thông qua các đối tượng được tạo. Để truy cập một biến instance, path sẽ là như sau:

/\* Dau tien, ban tao mot doi tuong \*/

Doituongthamchieu = new Constructor();

/\* Sau do ban goi mot bien nhu sau \*/

Doituongthamchieu.TenBien;

/\* Bay gio ban co the goi mot phuong thuc lop nhu sau \*/

Doituongthamchieu.TenPhuongThuc();

**Ví dụ:**

Ví dụ này giải thích cách để truy cập các biến instance và các phương thức của một lớp:

public class Xecon{

int Giaxe;

public Xecon(String ten){

// Contructor nay co mot tham so la *ten*.

System.out.println("Ten xe la :" + ten );

}

public void setGia( int gia ){

Giaxe = gia;

}

public int getGia( ){

System.out.println("Gia mua xe la :" + Giaxe );

return Giaxe;

}

public static void main(String []args){

/\* Tao doi tuong \*/

Xecon Xecuatoi = new Xecon( "Toyota" );

/\* Goi mot phuong thuc lop de thiet lap gia xe \*/

Xecuatoi.setGia( 1000000000 );

/\* Goi mot phuong thuc lop khac de lay gia xe \*/

Xecuatoi.getGia( );

/\* Ban cung co the truy cap bien instance nhu sau \*/

System.out.println("Gia tri bien :" + Xecuatoi.Giaxe );

}

}

Biên dịch và thực thi chương trình sẽ cho kết quả sau:

Ten xe la :Toyota

Gia mua xe la :1000000000

Gia tri bien :1000000000

**Ví dụ đối tượng và lớp mà duy trì bản ghi các sinh viên**

Trong ví dụ này, chúng ta tạo hai đối tượng của lớp Student và khởi tạo giá trị của các đối tượng này bằng việc triệu hồi phương thức insertRecord trên nó. Ở đây, chúng ta đang hiển thị trạng thái (dữ liệu) của các đối tượng bằng việc triệu hồi phương thức displayInformation.

class Student2{

int rollno;

String name;

void insertRecord(int r, String n){ //phuong thuc

rollno=r;

name=n;

}

void displayInformation(){System.out.println(rollno+" "+name);} //phuong thuc

public static void main(String args[]){

Student2 s1=new Student2();

Student2 s2=new Student2();

s1.insertRecord(111,"HoangThanh");

s2.insertRecord(222,"ThanhHuong");

s1.displayInformation();

s2.displayInformation();

}

}

**Ví dụ khác về lớp và đối tượng trong Java**

Ví dụ khác duy trì các bản ghi của lớp Rectangle. Phần giải thích tương tự như trên:

class Rectangle{

int length;

int width;

void insert(int l,int w){

length=l;

width=w;

}

void calculateArea(){System.out.println(length\*width);}

public static void main(String args[]){

Rectangle r1=new Rectangle();

Rectangle r2=new Rectangle();

r1.insert(11,5);

r2.insert(3,15);

r1.calculateArea();

r2.calculateArea();

}

}

### Các cách khác nhau để tạo đối tượng trong Java?

Có nhiều cách để tạo một đối tượng trong Java. Đó là:

* Bằng từ khóa new
* Bằng phương thức newInstance()
* Bằng phương thức clone(), ….
* Bằng phương thức factory, ...

### Đối tượng vô danh (annonymous) trong Java

Vô danh hiểu đơn giản là không có tên. Một đối tượng mà không có tham chiếu thì được xem như là đối tượng vô danh. Nếu bạn phải sử dụng một đối tượng chỉ một lần, thì đối tượng vô danh là một hướng tiếp cận tốt.

class Calculation{

void fact(int n){

int fact=1;

for(int i=1;i<=n;i++){

fact=fact\*i;

}

System.out.println("factorial is "+fact);

}

public static void main(String args[]){

new Calculation().fact(5); //Goi phuong thuc voi doi tuong vo danh (annonymous)

}

}

### Tạo nhiều đối tượng bởi chỉ một kiểu

Chúng ta có thể tạo nhiều đối tượng bởi chỉ một kiểu như khi chúng ta thực hiện trong các kiểu gốc. Ví dụ:

Rectangle r1=new Rectangle(),r2=new Rectangle(); //Tao hai doi tuong

**Ví dụ:**

class Rectangle{

int length;

int width;

void insert(int l,int w){

length=l;

width=w;

}

void calculateArea(){System.out.println(length\*width);}

public static void main(String args[]){

Rectangle r1=new Rectangle(),r2=new Rectangle(); //Tao hai doi tuong

r1.insert(11,5);

r2.insert(3,15);

r1.calculateArea();

r2.calculateArea();

}

}

Kết quả là:

Output:55

45

## PHƯƠNG THỨC TRONG JAVA

Một phương thức trong Java là một tập hợp các lệnh mà được nhóm cùng với nhau để thực hiện một hành động. Ví dụ khi bạn gọi phương thức System.out.print, hệ thống thực sự thực thi một vài lệnh để hiển thị một thông báo trên bàn điều khiển console.

Bây giờ, bạn sẽ học cách tạo các phương thức cho riêng bạn với hoặc không với các giá trị trả về, gọi một phương thức với hoặc không với các tham số, tải các phương thức sử dụng cùng tên, và áp dụng phương thức trừu tượng trong thiết kế chương trình.

Để hiểu sâu hơn các khái niệm được trình bày trong chương này, mời bạn tham khảo loạt bài: [**Ví dụ về Phương thức trong Java**](https://vietjack.com/bai_tap_java/phuong_thuc_trong_java.jsp).

### Tạo phương thức trong Java

Xem xét ví dụ sau để giải thích cú pháp của một phương thức:

public static int tenPhuongThuc(int a, int b) {

// phan than phuong thuc

}

Ở đây:

* **public static** : Là modifier.
* **int**: Kiểu trả về
* **tenPhuongThuc**: Tên phương thức
* **a, b**: Các tham số chính thức
* **int a, int b**: Danh sách các tham số

Các phương thức cũng còn được biết như các Procedure (thủ tục) hoặc Function (hàm):

* **Procedure:** Chúng không trả về bất kỳ giá trị nào.
* **Function:** Chúng trả về giá trị.

Sự định nghĩa phương thức bao gồm một header và phần thân phương thức. Tương tự như sau:

modifier kieuTraVe tenPhuongThuc (Danh sach tham so) {

// Than phuong thuc

}

Cú pháp trên bao gồm:

* **modifier:** Nó định nghĩa kiểu truy cập của phương thức và nó là tùy ý để sử dụng.
* **kieuTraVe:** Phương thức có thể trả về một giá trị.
* **tenPhuongThuc:** Đây là tên phương thức.
* **Danh sach tham so:** Danh sách các tham số, nó là kiểu, thứ tự, và số tham số của một phương thức. Đây là tùy ý, phương thức có thể không chứa tham số nào.
* **Than phuong thuc:** Phần thân phương thức định nghĩa những gì phương thức đó thực hiện với các lệnh.

**Ví dụ:**

Đây là code nguồn của phương thức max() được định nghĩa ở trên. Phương thức này nhận hai tham số là num1 và num2 và trả về giá trị lớn nhất của hai số:

/\*\* Vi du code tra ve so nho nhat cua hai so \*/

public static int minFunction(int n1, int n2) {

int min;

if (n1 > n2)

min = n2;

else

min = n1;

return min;

}

### Gọi phương thức trong Java

Để sử dụng một phương thức, nó nên được gọi. Có hai cách để gọi một phương thức, ví dụ: phương thức trả về một giá trị hoặc phương thức không trả về giá trị nào.

Tiến trình gọi phương thức là đơn giản. Khi một chương trình gọi phương thức, điều khiển chương trình truyền tín hiệu tới phương thức được gọi. Phương thức được gọi này sau đó trả về điều khiển tới caller trong hai điều kiện, khi:

* Lệnh return được thực thi.
* Tiến tới dấu ngoặc đóng ở cuối phương thức.

Phương thức trả về void được xem như là gọi tới một lệnh. Bạn xét ví dụ sau:

System.out.println("Day la Vietjack.com!");

Ví dụ sau minh họa về phương thức trả về giá trị:

int result = sum(6, 9);

**Ví dụ:**

Ví dụ sau minh họa cách định nghĩa một phương thức và cách để gọi nó:

public class ExampleMinNumber{

public static void main(String[] args) {

int a = 11;

int b = 6;

int c = minFunction(a, b);

System.out.println("Gia tri nho nhat = " + c);

}

/\*\* Tra ve gia tri nho nhat cua hai so \*/

public static int minFunction(int n1, int n2) {

int min;

if (n1 > n2)

min = n2;

else

min = n1;

return min;

}

}

Nó sẽ cho kết quả sau:

Gia tri nho nhat = 6

### Từ khóa void trong Java

Từ khóa void cho phép chúng ta tạo các phương thức mà không trả về giá trị nào. Ở đây, trong ví dụ sau, chúng ta xem xét một phương thức void là *Phương thứcRankPoints*. Phương thức này là một phương thức void mà không trả về bất kỳ giá trị nào. Gọi tới một phương thức void phải là một lệnh, ví dụ như *Phương thứcRankPoints(255.7);*. Nó là một lệnh Java mà kết thúc với dấu chấm phảy như dưới đây.

**Ví dụ:**

public class ExampleVoid {

public static void main(String[] args) {

methodRankPoints(255.7);

}

public static void methodRankPoints(double points) {

if (points >= 202.5) {

System.out.println("Rank:A1");

}

else if (points >= 122.4) {

System.out.println("Rank:A2");

}

else {

System.out.println("Rank:A3");

}

}

}

Nó sẽ cho kết quả sau:

Rank:A1

### Truyền các tham số bởi giá trị trong Java

Trong khi làm việc dưới tiến trình gọi, các tham số được truyền. Điều này nên trong cùng thứ tự như các tham số tương ứng của chúng trong phương thức. Các tham số có thể được truyền bởi giá trị hoặc bởi tham chiếu.

Truyền các tham số bởi giá trị nghĩa là gọi một phương thức với một tham số. Thông qua điều này, giá trị tham số được truyền tới tham số.

**Ví dụ:**

Chương trình sau minh họa việc truyền tham số bởi giá trị trong Java. Các giá trị của các tham số vẫn tồn tại giống như vậy cho dù sau lời gọi phương thức đó.

package edu.doannhg.basic;

// vi du de trao doi gia tri cua hai so

public class Test {

public static void main(String[] args) {

int a = 30;

int b = 45;

System.out.println("Truoc khi trao doi, gia tri cua a = " +

a + " va b = " + b);

// Trieu hoi phuong thuc hamTraoDoi

hamTraoDoi(a, b);

System.out.println("\n\*\*Bay gio, Truoc va Sau khi trao doi, cac gia tri se giong nhu nhau o day\*\*:");

System.out.println("Sau khi trao doi, a = " +

a + " va b = " + b);

}

public static void hamTraoDoi(int a, int b) {

System.out.println("Truoc khi trao doi (ben trong phuong thuc) , a = " + a

+ " va b = " + b);

// trao doi gia tri cua hai so

int c = a;

a = b;

b = c;

System.out.println("Sau khi trao doi (ben trong phuong thuc), a = " + a

+ " va b = " + b);

}

}

Nó sẽ cho kết quả:

Truoc khi trao doi, gia tri cua a = 30 va b = 45

Truoc khi trao doi (ben trong phuong thuc) , a = 30 va b = 45

Sau khi trao doi (ben trong phuong thuc), a = 45 va b = 30

\*\*Bay gio, Truoc va Sau khi trao doi, cac gia tri se giong nhu nhau o day\*\*:

Sau khi trao doi, a = 30 va b = 45

### Nạp chồng phương thức (Method overloading) trong Java

Khi một lớp có hai hoặc nhiều phương thức cùng tên nhưng khác nhau về tham số, nó được biết đến như là *Phương thức overloading*. Nó khác với **Overriding**. Trong overriding, một phương thức có một phương thức khác cùng tên, kiểu, số tham số, …

Bạn xem xét ví dụ sau trước khi tìm các số nhỏ nhất trong kiểu integer. Nếu chúng ta muốn tìm số nhỏ nhất ở kiểu double. Thì khi đó khái niệm về Overloading sẽ được giới thiệu để tạo hai hoặc nhiều phương thức cùng tên nhưng khác nhau về tham số.

Ví dụ sau giải thích điều này:

public class ExampleOverloading{

public static void main(String[] args) {

int a = 11;

int b = 6;

double c = 7.3;

double d = 9.4;

int result1 = minFunction(a, b);

// cung ten ham voi tham so khac nhau

double result2 = minFunction(c, d);

System.out.println("Gia tri nho nhat = " + result1);

System.out.println("Gia tri nho nhat = " + result2);

}

// cho integer

public static int minFunction(int n1, int n2) {

int min;

if (n1 > n2)

min = n2;

else

min = n1;

return min;

}

// cho double

public static double minFunction(double n1, double n2) {

double min;

if (n1 > n2)

min = n2;

else

min = n1;

return min;

}

}

Nó sẽ cho kết quả sau:

Gia tri nho nhat = 6

Gia tri nho nhat = 7.3

Nạp chồng phương thức làm chương trình có thể đọc được. Ở đây, hai phương thức được cung cấp cùng một tên nhưng khác tham số. Số nhỏ nhất từ kiểu integer và kiểu double là kết quả.

### Sử dụng các tham số dòng lệnh (command-line) trong Java

Đôi khi bạn muốn truyền thông tin vào trong một chương trình khi bạn chạy nó. Điều này được thực hiện bởi việc truyền các tham số dòng lệnh tới phương thức *main()*.

Một tham số dòng lệnh là thông tin mà trực tiếp theo sau tên của chương trình trên dòng lệnh khi nó được thực thi. Truy cập các tham số dòng lệnh bên trong một chương trình Java là khá dễ dàng. Chúng được lưu giữ như là các chuỗi trong mảng String đã truyền tới main().

**Ví dụ:**

Chương trình sau hiển thị tất cả tham số dòng lệnh mà được gọi với:

public class CommandLine {

public static void main(String args[]){

for(int i=0; i<args.length; i++){

System.out.println("args[" + i + "]: " +

args[i]);

}

}

}

Thử thực thi chương trình này như đã chỉ ở đây:

java CommandLine this is a command line 200 -100

Nó sẽ cho kết quả sau:

args[0]: this

args[1]: is

args[2]: a

args[3]: command

args[4]: line

args[5]: 200

args[6]: -100

### Constructor trong Java

Một constructor khởi tạo một đối tượng khi nó được tạo. Nó có cùng tên với lớp của nó và cú pháp tương tự như một phương thức. Tuy nhiên, các constructor không có kiểu trả về rõ ràng.

Một nét đặc trưng là, bạn sẽ sử dụng một constructor để cung cấp các giá trị khởi tạo tới các biến instance được định nghĩa bởi lớp, hoặc để thực thi bất kỳ thủ tục khởi đầu nào khác được yêu cầu để tạo một đối tượng theo mẫu.

Tất cả các lớp đều có các constructor, dù bạn có hay không định nghĩa nó, bởi vì Java tự động cung cấp một constructor mặc định mà khởi tạo tất cả biến thành viên về zero. Tuy nhiên, một khi bạn định nghĩa constructor cho riêng bạn, thì constructor mặc định sẽ không còn được sử dụng nữa.

**Ví dụ:**

Đây là một ví dụ đơn giản mà sử dụng một constructor trong Java:

// Mot constructor.

class MyClass {

int x;

// Sau day la constructor

MyClass() {

x = 10;

}

}

Bạn sẽ gọi constructor để khởi tạo các đối tượng như sau:

public class ConsDemo {

public static void main(String args[]) {

MyClass t1 = new MyClass();

MyClass t2 = new MyClass();

System.out.println(t1.x + " " + t2.x);

}

}

Thường thì, bạn sẽ cần một constructor mà chấp nhận một hoặc nhiều tham số. Các tham số được thêm tới một constructor theo cách tương tự như chúng được thêm tới một phương thức, vừa khai báo chúng bên trong dấu ngoặc đơn ở sau tên của constructor.

**Ví dụ:**

Đây là một ví dụ đơn giản mà sử dụng một constructor trong Java:

// Mot constructor.

class MyClass {

int x;

// Sau day la constructor

MyClass(int i ) {

x = i;

}

}

Bạn sẽ gọi constructor để khởi tạo các đối tượng như sau:

public class ConsDemo {

public static void main(String args[]) {

MyClass t1 = new MyClass( 10 );

MyClass t2 = new MyClass( 20 );

System.out.println(t1.x + " " + t2.x);

}

}

Nó sẽ cho kết quả sau:

10 20

### Các tham số biến (var-args) trong Java

JDK 1.5 cho bạn khả năng truyền một số các tham số biến cùng kiểu tới một phương thức. tham số trong phương thức được khai báo như sau:

tenKieu... tenThamSo

Trong khai báo phương thức này, bạn xác định kiểu được theo sau bởi một ellipsis (…). Chỉ một tham số độ dài biến có thể được xác định trong một phương thức, và tham số này phải là tham số cuối cùng. Bất kỳ tham số thông thường nào phải đặt trước nó.

**Ví dụ:**

public class VarargsDemo {

public static void main(String args[]) {

// Goi phuong thuc voi bien args

printMax(34, 3, 3, 2, 56.5);

printMax(new double[]{1, 2, 3});

}

public static void printMax( double... numbers) {

if (numbers.length == 0) {

System.out.println("Khong co tham so nao duoc truyen");

return;

}

double result = numbers[0];

for (int i = 1; i < numbers.length; i++)

if (numbers[i] > result)

result = numbers[i];

System.out.println("Gia tri max la " + result);

}

}

Nó sẽ cho kết quả sau:

Gia tri max la 56.5

Gia tri max la 3.0

### Phương thức finalize() trong Java

Nó là có thể để định nghĩa một phương thức mà sẽ được gọi ngay trước khi hủy một đối tượng bởi Garbage Collector. Phương thức này được gọi là *finalize()*, và nó có thể được sử dụng để chắc chắn rằng một đối tượng hoàn toàn kết thúc.

Ví dụ, bạn có thể sử dụng finalize() để đảm bảo rằng một open file mà sở hữu bởi một đối tượng nào đó đã được đóng.

Để thêm một finalizer tới một lớp, đơn giản bạn định nghĩa phương thức finalize(). Java runtime gọi phương thức đó bất cứ khi nào nó chuẩn bị để tái chế một đối tượng của lớp đó.

Bên trong phương thức finalize(), bạn sẽ xác định những hành động nào phải được thực hiện trước khi một đối tượng bị phá hủy.

Phương thức finalize() có form chung là:

protected void finalize( )

{

// tai day la phan code ket thuc

}

Ở đây, từ khóa protected là một specifier mà ngăn cản việc truy cập tới finalize() bởi code được định nghĩa bên ngoài lớp của nó.

Nghĩa là, bạn không thể biết khi nào hoặc lúc nào finalize() sẽ được thực thi. Ví dụ, nếu chương trình của bạn kết thúc trước khi Garbage Collection xuất hiện, finalize() sẽ không thực thi.

Để hiểu sâu hơn các khái niệm được trình bày trong chương này, mời bạn tham khảo loạt bài ví dụ về Phương thức trong Java: [**Ví dụ về Phương thức trong Java**](https://vietjack.com/bai_tap_java/phuong_thuc_trong_java.jsp).

## NẠP CHỒNG PHƯƠNG THỨC TRONG JAVA

Nếu một lớp có nhiều phương thức cùng tên nhưng có số tham số khác nhau, thì đó là nạp chồng phương thức (Method Overloading). Nếu bạn phải thực hiện chỉ một hoạt động, có cùng tên phương thức, thì kỹ thuật này làm tăng tính có thể đọc cho chương trình. Giả sử bạn viết phương thức như a(int, int) cho hai tham số, và b(int, int, int) cho ba tham số, thì khi đó điều này có thể gây khó khăn cho bạn cũng như cho các lập trình viên khác để hiểu hành vi của phương thức, bởi vì tên của nó là khác nhau. Vì thế, chúng ta thực hiện nạp chồng phương thức để giúp việc phân tích chương trình nhanh hơn.

Có hai cách để nạp chồng phương thức trong Java, đó là:

* Bằng việc thay đổi số tham số
* Bằng việc thay đổi kiểu dữ liệu

**Ghi chú**: Trong Java, nạp chồng phương thức là không thể bằng việc thay đổi kiểu trả về của phương thức.

### Ví dụ về nạp chồng phương thức bằng cách thay đổi số tham số

Trong ví dụ này, chúng ta sử dụng nạp chồng phương thức: phương thức sum đầu tiên thực hiện phép cộng hai số và phương thức sum thứ hai thực hiện phép cộng ba số.

class Calculation{

void sum(int a,int b){System.out.println(a+b);}

void sum(int a,int b,int c){System.out.println(a+b+c);}

public static void main(String args[]){

Calculation obj=new Calculation();

obj.sum(10,10,10);

obj.sum(20,20);

}

}

Kết quả là:

Output:30

40

### Ví dụ nạp chồng phương thức bằng cách thay đổi kiểu dữ liệu của tham số

Trong ví dụ này, chúng ta sử dụng nạp chồng phương thức mà khác nhau về kiểu dữ liệu. Phương thức sum đầu tiên nhận hai số nguyên và phương thức sum thứ hai nhận hai số thực double.

class Calculation2{

void sum(int a,int b){System.out.println(a+b);}

void sum(double a,double b){System.out.println(a+b);}

public static void main(String args[]){

Calculation2 obj=new Calculation2();

obj.sum(10.5,10.5);

obj.sum(20,20);

}

}

Kết quả là:

Output:21.0

40

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

Bởi vì việc này có thể gây ra tính lưỡng nghĩa, mơ hồ (ambiguity). Bạn theo dõi ví dụ sau để thấy rõ điều này:

Kết quả là:

class Calculation3{

int sum(int a,int b){System.out.println(a+b);}

double sum(int a,int b){System.out.println(a+b);}

public static void main(String args[]){

Calculation3 obj=new Calculation3();

int result=obj.sum(20,20); //Gay ra Compile Time Error

}

}

Trong dòng code này: *int result=obj.sum(20,20);* thì Java có thể quyết định phương thức sum() nào nên được gọi?

**Câu hỏi**: Chúng ta có thể nạp chồng phương thức main()?

Có. Bạn có thể có bất kỳ số lượng phương thức main nào trong môt lớp bởi nạp chồng phương thức. Bạn theo dõi ví dụ đơn giản sau:

class Overloading1{

public static void main(int a){

System.out.println(a);

}

public static void main(String args[]){

System.out.println("Phuong thuc main() duoc trieu hoi");

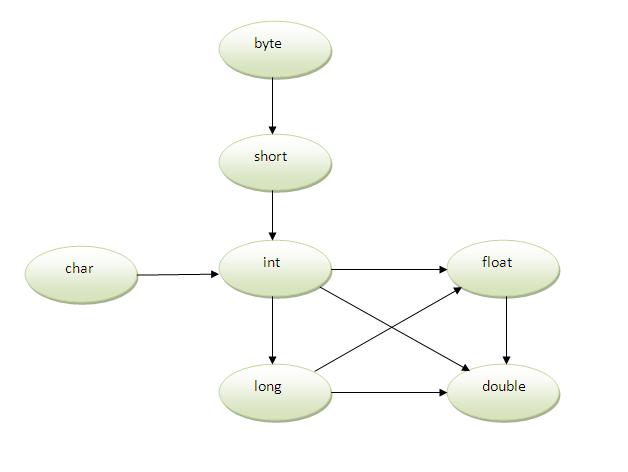
main(10);

}

}

### Nạp chồng phương thức và TypePromotion trong Java

Một kiểu được promote thành một kiểu khác hoàn toàn nếu không tìm thấy kiểu dữ liệu nào được kết nối. Bạn theo dõi sơ đồ sau để hiểu khái niệm này.



Như sơ đồ trên, byte có thể được promote thành short, int, long, float, hoặc double. Kiểu dữ liệu short có thể được promote thành int, long, float, hoặc double. Kiểu dữ liệu char có thể được promote thành kiểu int, long, float hoặc double và …

Ví dụ về nạp chồng phương thức với TypePromotion:

class OverloadingCalculation1{

void sum(int a,long b){System.out.println(a+b);}

void sum(int a,int b,int c){System.out.println(a+b+c);}

public static void main(String args[]){

OverloadingCalculation1 obj=new OverloadingCalculation1();

obj.sum(20,20); //Bay gio int literal thu hai se duoc promote thanh long

obj.sum(20,20,20);

}

}

### Ví dụ về nạp chồng phương thức với TypePromotion nếu kết nối là không được tìm thấy

Nếu có kết nối về các tham số kiểu trong phương thức, thì TypePromotion không được thực hiện.

class OverloadingCalculation2{

void sum(int a,int b){System.out.println("phuong thuc int arg duoc trieu hoi");}

void sum(long a,long b){System.out.println("phuong thuc long arg duoc trieu hoi");}

public static void main(String args[]){

OverloadingCalculation2 obj=new OverloadingCalculation2();

obj.sum(20,20); //Bay gio phuong thuc int arg sum() duoc trieu hoi

}

}

### Ví dụ về nạp chồng phương thức với TypePromotion trong trường hợp lưỡng nghĩa

Nếu không có kết nối các tham số kiểu trong phương thức, và mỗi phương thức promote số tham số giống nhau, thì đó là trường hợp lưỡng nghĩa.

class OverloadingCalculation3{

void sum(int a,long b){System.out.println("Phuong thuc a duoc trieu hoi");}

void sum(long a,int b){System.out.println("Phuong thuc b duoc trieu hoi");}

public static void main(String args[]){

OverloadingCalculation3 obj=new OverloadingCalculation3();

obj.sum(20,20); //Khong co tinh luong nghia o day

}

}

Kết quả là:

Output:Compile Time Error

Ps:Overload các bạn có thể hiểu 2 hay nhiều hàm cùng tên mà khác số lượng tham số hoặc khác kiểu tham số.

**QUẢNG CÁO**

[Tìm Kiếm Quảng Cáo](https://clck.mgid.com/ghits/17555279/i/57734082/0/pp/1/4?h=zgtsw4Ww8-CKptgrfUC0cMwGosGIJwQS8iV1E4G3YGXyRMU6kiu48_yBQr5UU0FSbRA5E5crjKyeg-QA10JsvQ**&rid=865853d2-77d2-11ee-93e2-c84bd6826d76&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[**Tự đánh giá trầm cảm: Làm bài kiểm tra và nhận trợ giúp**](https://clck.mgid.com/ghits/17555279/i/57734082/0/pp/1/4?h=zgtsw4Ww8-CKptgrfUC0cMwGosGIJwQS8iV1E4G3YGXyRMU6kiu48_yBQr5UU0FSbRA5E5crjKyeg-QA10JsvQ**&rid=865853d2-77d2-11ee-93e2-c84bd6826d76&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[Sức Khỏe Nam Giới](https://clck.mgid.com/ghits/14436309/i/57734082/0/pp/2/4?h=zgtsw4Ww8-CKptgrfUC0cA_HAeSSHGPyI9r4YllIVL181mLSHE7UtzLo9KoC3WIv-6btLRWWNMUB2ubNYIqACg**&rid=865853d2-77d2-11ee-93e2-c84bd6826d76&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[**Phong độ đàn ông xuống dốc? Phục hồi ngay với mẹo này**](https://clck.mgid.com/ghits/14436309/i/57734082/0/pp/2/4?h=zgtsw4Ww8-CKptgrfUC0cA_HAeSSHGPyI9r4YllIVL181mLSHE7UtzLo9KoC3WIv-6btLRWWNMUB2ubNYIqACg**&rid=865853d2-77d2-11ee-93e2-c84bd6826d76&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[Brainberries](https://clck.mgid.com/ghits/15490950/i/57734082/0/pp/3/4?h=zgtsw4Ww8-CKptgrfUC0cNE2nClJNM35kpXF-tFrCO0guPane2ZK1oIqpRIdW-FW3ktHq8mC8nBTojWT5LMKFA**&rid=865853d2-77d2-11ee-93e2-c84bd6826d76&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[**Đã tìm ra cây thuốc độc gấp x6000 xyanua. Phổ biến ở Việt Nam?**](https://clck.mgid.com/ghits/15490950/i/57734082/0/pp/3/4?h=zgtsw4Ww8-CKptgrfUC0cNE2nClJNM35kpXF-tFrCO0guPane2ZK1oIqpRIdW-FW3ktHq8mC8nBTojWT5LMKFA**&rid=865853d2-77d2-11ee-93e2-c84bd6826d76&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[Brainberries](https://clck.mgid.com/ghits/6472132/i/57734082/0/pp/4/4?h=zgtsw4Ww8-CKptgrfUC0cMQnly0QGyWmu7had7yLdeH932fiEXGrsIRn1Y5kb7_I8EFx_0l9g9F2mBWRPupKsQ**&rid=865853d2-77d2-11ee-93e2-c84bd6826d76&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[**Cặp đôi song sinh với tài diễn cosplay đang khuấy đảo cư dân mạng**](https://clck.mgid.com/ghits/6472132/i/57734082/0/pp/4/4?h=zgtsw4Ww8-CKptgrfUC0cMQnly0QGyWmu7had7yLdeH932fiEXGrsIRn1Y5kb7_I8EFx_0l9g9F2mBWRPupKsQ**&rid=865853d2-77d2-11ee-93e2-c84bd6826d76&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[Brainberries](https://clck.mgid.com/ghits/16888031/i/57734082/0/pp/5/4?h=zgtsw4Ww8-CKptgrfUC0cF0g4_MqvUsQ7yV4TeunKP0FYdYhZmkN-g_ZvYyfyDkVyhUdTne1jOLVf_DRR8jWyA**&rid=865853d2-77d2-11ee-93e2-c84bd6826d76&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[**Thanh Sơn-Khả Ngân phủ nhận yêu nhau nhưng liên tục rải "đường"**](https://clck.mgid.com/ghits/16888031/i/57734082/0/pp/5/4?h=zgtsw4Ww8-CKptgrfUC0cF0g4_MqvUsQ7yV4TeunKP0FYdYhZmkN-g_ZvYyfyDkVyhUdTne1jOLVf_DRR8jWyA**&rid=865853d2-77d2-11ee-93e2-c84bd6826d76&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[Brainberries](https://clck.mgid.com/ghits/12336635/i/57734082/0/pp/6/4?h=zgtsw4Ww8-CKptgrfUC0cGNfLYpGO2164NqXMiYESaHoMVhLE4k_KT4II3Oej3ddv5suuGLLXbUo-iz6vS20bw**&rid=865853d2-77d2-11ee-93e2-c84bd6826d76&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[**Những dấu hiệu cho thấy nhất định bạn phải giữ cô gái này**](https://clck.mgid.com/ghits/12336635/i/57734082/0/pp/6/4?h=zgtsw4Ww8-CKptgrfUC0cGNfLYpGO2164NqXMiYESaHoMVhLE4k_KT4II3Oej3ddv5suuGLLXbUo-iz6vS20bw**&rid=865853d2-77d2-11ee-93e2-c84bd6826d76&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

Đã có app VietJack trên điện thoại, giải bài tập SGK, SBT Soạn văn, Văn mẫu, Thi online, Bài giảng....miễn phí. Tải ngay ứng dụng trên Android và iOS.

[](https://apps.apple.com/vn/app/vietjack/id1490262941?l=vi&mt=8)

[](https://play.google.com/store/apps/details?id=com.jsmile.android.vietjack)

Theo dõi chúng tôi miễn phí trên mạng xã hội facebook và youtube:

Các bạn có thể mua thêm khóa học JAVA CORE ONLINE VÀ ỨNG DỤNG cực hay, giúp các bạn vượt qua các dự án trên trường và đi thực tập Java. Khóa học có giá chỉ 300K, nhằm ưu đãi, tạo điều kiện cho sinh viên cho thể mua khóa học.

Nội dung khóa học gồm 16 chuơng và 100 video cực hay, học trực tiếp tại [**https://www.udemy.com/tu-tin-di-lam-voi-kien-thuc-ve-java-core-toan-tap/**](https://www.udemy.com/tu-tin-di-lam-voi-kien-thuc-ve-java-core-toan-tap/) Bạn nào có nhu cầu mua, inbox trực tiếp a Tuyền, cựu sinh viên Bách Khoa K53, fb: [**https://www.facebook.com/tuyen.vietjack**](https://www.facebook.com/tuyen.vietjack)

Loạt bài hướng dẫn của chúng tôi dựa một phần trên nguồn tài liệu của: Tutorialspoint.com

Follow facebook cá nhân Nguyễn Thanh Tuyền [**https://www.facebook.com/tuyen.vietjack**](https://www.facebook.com/tuyen.vietjack)để tiếp tục theo dõi các loạt bài mới nhất về Java,C,C++,Javascript,HTML,Python,Database,Mobile.... mới nhất của chúng tôi.

**Bài học Java phổ biến tại vietjack.com:**

* [Nạp chồng phương thức (Overloading)](https://vietjack.com/java/overloading_trong_java.jsp)
* [Constructor](https://vietjack.com/java/constructor_trong_java.jsp)
* [ArrayList trong Java](https://vietjack.com/java/lop_arraylist_trong_java.jsp)
* [Interface trong Java](https://vietjack.com/java/interface_trong_java.jsp)
* [Thread trong Java](https://vietjack.com/java/multithread_trong_java.jsp)
* [247 bài tập Java](https://vietjack.com/bai_tap_java/bai_tap_mau_java_va_vi_du_java.jsp)
* [200 câu phỏng vấn Java](https://vietjack.com/cau_hoi_phong_van_java/index.jsp)

## CONSTRUCTOR TRONG JAVA

Constructor trong Java là một kiểu phương thức đặc biệt mà được sử dụng để khởi tạo đối tượng. Constructor được triệu hồi tại thời gian tạo đối tượng. Nó xây dựng giá trị, cung cấp dữ liệu cho đối tượng, đó là lý do nó được gọi là Constructor.

Khi bàn luận về các lớp, một trong những chủ đề quan trọng là các constructor. Mỗi lớp có ít nhất một constructor. Nếu chúng ta không viết một constructor một cách rõ ràng cho một lớp thì bộ biên dịch Java xây dựng một constructor mặc định cho lớp đó.

Mỗi khi một đối tượng mới được tạo ra, ít nhất một constructor sẽ được gọi. Quy tắc chính của các constructor là chúng có cùng tên như lớp đó. Một lớp có thể có nhiều hơn một constructor.

Sau đây là ví dụ về một constructor:

public class Xecon{

public Xecon(){

}

public Xecon(String ten){

// Contructor nay co mot tham so la *ten*.

}

}

### Qui tắc để tạo Constructor trong Java

Có hai loại Constructor: Loại thứ nhất là Constructor mặc định, loại này không có tham số. Và loại thứ hai là Constructor được tham số hóa.

### Constructor mặc định trong Java

Đây là loại Constructor mà không có tham số. Chúng có cú pháp như sau:

ten\_lop(){}

Trong ví dụ này, chúng ta tạo Constructor không có tham số trong lớp Bike. Nó sẽ được triệu hồi tại thời gian tạo đối tượng.

class Bike1{

Bike1(){System.out.println("Bike duoc tao");}

public static void main(String args[]){

Bike1 b=new Bike1();

}

}

**Qui tắc**: Nếu không có Constructor nào được xác định trong một lớp, thì Compiler tự động tạo một Constructor mặc định.

### Câu hỏi: Mục đích của Constructor mặc định?

Constructor mặc định cung cấp các giá trị mặc định cho đối tượng như 0, null, … tùy thuộc vào kiểu dữ liệu.

Ví dụ Constructor mặc định mà hiển thị các giá trị mặc định

class Student3{

int id;

String name;

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){

Student3 s1=new Student3();

Student3 s2=new Student3();

s1.display();

s2.display();

}

}

Giải thích: Trong lớp trên, bạn không tạo bất cứ Constructor nào, vì thế Compiler cung cấp cho bạn một Constructor mặc định. Ở đây các giá trị 0 và null được cung cấp bởi Constructor mặc định.

### Constructor được tham số hóa trong Java

Một Constructor mà có các tham số được gọi là constructor được tham số hóa.

**Câu hỏi**: Tại sao sử dụng Constructor được tham số hóa?

Constructor được tham số hóa được sử dụng để cung cấp các giá trị khác nhau cho các đối tượng riêng biệt.

**Ví dụ** về Constructor được tham số hóa:

Trong ví dụ này, chúng ta tạo Constructor của lớp Student mà có hai tham số. Chúng ta có thể có bất cứ số tham số nào trong Constructor.

class Student4{

int id;

String name;

Student4(int i,String n){

id = i;

name = n;

}

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){

Student4 s1 = new Student4(111,"Karan");

Student4 s2 = new Student4(222,"Aryan");

s1.display();

s2.display();

}

}

## NẠP CHỒNG CONSTRUCTOR TRONG JAVA

Nạp chồng Constructor là một kỹ thuật trong Java mà trong đó một lớp có thể có bất cứ số lượng Constructor nào mà khác nhau về danh sách tham số. Trình biên dịch Compiler phân biệt sự khác nhau giữa các Constructor này bằng việc phân tích số tham số trong danh sách và kiểu của chúng.

Ví dụ về nạp chồng Constructor

class Student5{

int id;

String name;

int age;

Student5(int i,String n){

id = i;

name = n;

}

Student5(int i,String n,int a){

id = i;

name = n;

age=a;

}

void display(){System.out.println(id+" "+name+" "+age);}

public static void main(String args[]){

Student5 s1 = new Student5(111,"Hoang");

Student5 s2 = new Student5(222,"Thanh",25);

s1.display();

s2.display();

}

}

### Điểm khác nhau giữa Constructor và Phương thức trong Java

Bảng dưới đây liệt kê một số điểm 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 trưng bày hành vi của một đối tượng |
| Constructor phải không có kiểu trả về | Phương thức phải có kiểu trả về |
| Constructor được triệu hồi một cách ngầm định | Phương thức phải được triệu hồi một cách tường minh |
| Compiler cung cấp một Constructor mặc định nếu bạn không có bất cứ Constructor nào | Phương thức không được cung cấp bởi Compiler trong bất cứ trường hợp nào |
| Tên Constructor phải giống tên lớp | Tên phương thức có thể hoặc không giống như tên lớp |

### Copy Constructor trong Java

Trong Java không có Copy Constructor như trong C++. Tuy nhiên, bạn có thể sao chép các giá trị của một đối tượng tới đối tượng khác. Có nhiều cách để thực hiện việc sao chép các giá trị, đó là:

* Bởi Constructor
* Bởi gán các giá trị của một đối tượng vào trong đối tượng khác.
* Bởi phương thức clone() của lớp Object.

Trong ví dụ này, chúng ta sao chép các giá trị của một đối tượng vào trong đối tượng khác bởi sử dụng Constructor trong Java.

class Student6{

int id;

String name;

Student6(int i,String n){

id = i;

name = n;

}

Student6(Student6 s){

id = s.id;

name =s.name;

}

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){

Student6 s1 = new Student6(111,"Hoang");

Student6 s2 = new Student6(s1);

s1.display();

s2.display();

}

}

### Sao chép các giá trị mà không sử dụng Constructor

Chúng ta có thể sao chép các giá trị của một đối tượng vào trong đối tượng khác bởi gán các giá trị của đối tượng đó vào trong đối tượng khác. Trong trường hợp này, chúng ta không cần tạo Constructor.

class Student7{

int id;

String name;

Student7(int i,String n){

id = i;

name = n;

}

Student7(){}

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){

Student7 s1 = new Student7(111,"Hoang");

Student7 s2 = new Student7();

s2.id=s1.id;

s2.name=s1.name;

s1.display();

s2.display();

}

}

**Câu hỏi**: Có phải Constructor trả về giá trị?

Có, đó là instance (sự thể hiện) của lớp hiện tại. (bạn không thể sử dụng kiểu trả về, tuy vậy nó trả về một giá trị).

**Câu hỏi**: Constructor có thể thực hiện các tác vụ khác ngoài khởi tạo không?

Có, giống như quá trình tạo đối tượng, bắt đầu một Thread, gọi phương thức, … Bạn có thể thực hiện bất cứ hoạt động nào trong Constructor như khi bạn thực hiện trong phương thức.

Ps: Các bạn xem video dưới đây để hiểu thêm về constructor trong Java nhé https://goo.gl/ZX4JDf

**QUẢNG CÁO**

## TỪ KHÓA STATIC TRONG JAVA

Từ khóa static trong Java được sử dụng chính để quản trị bộ nhớ. Chúng ta có thể áp dụng từ khóa static với biến (cũng được gọi là biến lớp, biến class), phương thức (cũng được gọi là phương thức lớp), khối, các lớp được lặp. Từ khóa static thuộc về lớp chứ không thuộc về instance (sự thể hiện) của lớp.

### Biến static trong Java

Khi bạn khai báo một biến là static, thì biến đó được gọi là biến tĩnh, hay biến static.

Biến static có thể được sử dụng để tham chiếu thuộc tính chung của tất cả đối tượng (mà không là duy nhất cho mỗi đối tượng), ví dụ như tên công ty của nhân viên, tên trường học của các sinh viên, …

Biến static lấy bộ nhớ chỉ một lần trong Class Area tại thời gian tải lớp đó.

### Lợi thế của biến static

Biến static giúp bộ nhớ chương trình của bạn được sử dụng hiệu quả hơn (tiết kiệm bộ nhớ).

### Tìm hiểu vấn đề xảy ra khi không có biến static

class Student{

int rollno;

String name;

String college="BachKhoa";

}

Giả sử có 500 sinh viên trong trường đại học, bây giờ instance của các thành viên dữ liệu sẽ lấy bộ nhớ mỗi khi đối tượng được tạo. Tất cả sinh viên có rollno và name duy nhất vì thế instance của thành viên dữ liệu là tốt. Ở đây, college là thuộc tính chung của tất cả đối tượng. Nếu chúng ta tạo nó là static, thì trường này sẽ chỉ lấy bộ nhớ một lần.

*Ghi chú*: Thuộc tính static trong Java được chia sẻ tới tất cả đối tượng.

Ví dụ về biến static trong Java

//Chuong trinh vi du ve bien static trong Java

class Student8{

int rollno;

String name;

static String college ="BachKhoa";

Student8(int r,String n){

rollno = r;

name = n;

}

void display (){System.out.println(rollno+" "+name+" "+college);}

public static void main(String args[]){

Student8 s1 = new Student8(111,"Hoang");

Student8 s2 = new Student8(222,"Thanh");

s1.display();

s2.display();

}

}

### Chương trình Counter mà không sử dụng biến static

Trong ví dụ, chúng ta tạo một biến instance có tên count mà được tăng lên trong constructor. Khi biến instance này lấy bộ nhớ tại thời điểm tạo đối tượng, mỗi đối tượng sẽ có bản sao của biến instance đó, nếu nó được tăng lên, nó sẽ không phản ánh các đối tượng khác. Vì thế mỗi đối tượng sẽ có giá trị 1 trong biến count.

class Counter{

int count=0; //se lay bo nho (memory) khi bien instance duoc tao

//Ket qua thuc hien chuong trinh hien ra 3 so 1 o 3 dong

Counter(){

count++;

System.out.println(count);

}

public static void main(String args[]){

Counter c1=new Counter();

Counter c2=new Counter();

Counter c3=new Counter();

}

}

### Chương trình counter với biến static trong Java

Như bạn đã thấy ở trên, biến static sẽ lấy bộ nhớ chỉ một lần, nếu bất cứ đối tượng nào thay đổi giá trị của biến static, nó sẽ vẫn ghi nhớ giá trị của nó.

class Counter2{

static int count=0; //se lay bo nho chi mot lan và giu lai gia tri cua no

//ket qua thuc hien in ra 3 dong cac gia tri : 1,2,3

Counter2(){

count++;

System.out.println(count);

}

public static void main(String args[]){

Counter2 c1=new Counter2();

Counter2 c2=new Counter2();

Counter2 c3=new Counter2();

}

}

### Phương thức static trong Java

Nếu bạn áp dụng từ khóa static với bất cứ phương thức nào, thì phương thức đó được gọi là phương thức static.

* Một phương thức static thuộc lớp chứ không phải đối tượng của lớp.
* Một phương thức static có thể được triệu hồi mà không cần tạo một instance của một lớp.
* Phương thức static có thể truy cập thành viên dữ liệu static và có thể thay đổi giá trị của nó.

Ví dụ về phương thức static trong Java

//Chuong trinh thay doi thuoc tinh chung cua tat ca doi tuong (truong static).

class Student9{

int rollno;

String name;

static String college = "BachKhoa";

static void change(){

college = "QuocGia";

}

Student9(int r, String n){

rollno = r;

name = n;

}

void display (){System.out.println(rollno+" "+name+" "+college);}

public static void main(String args[]){

Student9.change();

Student9 s1 = new Student9 (111,"Hoang");

Student9 s2 = new Student9 (222,"Thanh");

Student9 s3 = new Student9 (333,"Nam");

s1.display();

s2.display();

s3.display();

}

}

Ví dụ khác về phương thức static mà thực hiện phép tính toán thông thường

//Chuong trinh lay cube (gia tri lap phuong) cua so da cho boi phuong thuc static

class Calculate{

static int cube(int x){

return x\*x\*x;

}

public static void main(String args[]){

int result=Calculate.cube(5);

System.out.println(result);

}

}

### Một số hạn chế cho phương thức static

Có hai hạn chế chính cho phương thức static. Đó là:

* Phương thức static không thể sử dụng thành viên dữ liệu non-static hoặc gọi trực tiếp phương thức non-static.
* Từ khóa this và super không thể được sử dụng trong ngữ cảnh static.

class A{

int a=40;//non static

public static void main(String args[]){

System.out.println(a);

}

}

Chạy chương trình trên sẽ cho kết quả là Compile Time Error.

**Câu hỏi**: Tại sao phương thức main trong Java là static?

Bởi vì đối tượng là không cần thiết 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.

### Khối static trong Java

Được sử dụng để khởi tạo thành viên dữ liệu static. Nó được thực thi trước phương thức main tại thời gian tải lớp. Dưới đây là ví dụ về khối static trong Java:

class A2{

static{System.out.println("Khoi static duoc trieu hoi");}

public static void main(String args[]){

System.out.println("Hello main");

}

}

**Câu hỏi**: Chúng ta có thể thực thi một chương trình mà không có phương thức main()?

Có, một trong các cách đó là khối static trong phiên bản trước của JDK, không trong JDK 1.7.

class A3{

static{

System.out.println("Khoi static duoc trieu hoi");

System.exit(0);

}

}

**QUẢNG CÁO**

## TỪ KHÓA THIS TRONG JAVA

Có nhiều cách sử dụng từ khóa this trong Java. Trong Java, this là một biến tham chiếu mà tham chiếu tới đối tượng hiện tại.

### Sự sử dụng của từ khóa this trong Java

Trong Java, từ khóa this có 6 cách sử dụng, như sau:

* Từ khóa this có thể được sử dụng để tham chiếu biến instance của lớp.
* this() có thể được sử dụng để triệu hồi Constructor của lớp hiện tại.
* Từ khóa this có thể được sử dụng để triệu hồi ngầm định phương thức lớp hiện tại.
* Từ khóa this có thể được truyền như là một tham số trong lời gọi phương thức.
* Từ khóa this có thể được truyền như là một tham số trong lời gọi Constructor.
* Từ khóa this cũng có thể được sử dụng để trả về instance của lớp hiện tại.

**Đề nghị**: Nếu bạn mới học về Java thì bạn nên chỉ theo dõi hai cách sử dụng của từ khóa this.

### Từ khóa this có thể được sử dụng để tham chiếu biến instance của lớp hiện tại

Nếu có tính lưỡng nghĩa giữa biến instance và tham số, thì từ khóa this xử lý vấn đề này. Đầu tiên, chúng ta theo dõi ví dụ để hiểu vấn đề xảy ra khi không sử dụng từ khóa this.

Ví dụ

class Student10{

int id;

String name;

Student10(int id,String name){

id = id;

name = name;

}

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){

Student10 s1 = new Student10(111,"Hoang");

Student10 s2 = new Student10(321,"Thanh");

s1.display();

s2.display();

}

}

Trong ví dụ này, tham số và biến instance là giống nhau, và đó là lý do tại sao chúng ta sử dụng từ khóa this để phân biệt giữa các biến cục bộ và các biến instance.

### Xử lý vấn đề trên bởi từ khóa this trong Java

//Vi du ve tu khoa this trong Java

class Student11{

int id;

String name;

Student11(int id,String name){

this.id = id;

this.name = name;

}

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){

Student11 s1 = new Student11(111,"Hoang");

Student11 s2 = new Student11(222,"Thanh");

s1.display();

s2.display();

}

}

Nếu biến cục bộ và biến instance là khác nhau, thì không cần thiết sử dụng từ khóa this như trong chương trình sau đây:

class Student12{

int id;

String name;

Student12(int i,String n){

id = i;

name = n;

}

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){

Student12 e1 = new Student12(111,"Hoang");

Student12 e2 = new Student12(222,"Thanh");

e1.display();

e2.display();

}

}

### this() có thể được sử dụng để triệu hồi constructor của lớp hiện tại

Lời gọi this() constructor có thể được sử dụng để triệu hồi Constructor của lớp hiện tại. Hướng tiếp cận này là tốt hơn nếu bạn có nhiều Constructor trong lớp và muốn tái sử dụng Constructor đó.

//Chuong trinh vi du loi goi this() constructor (constructor chaining)

class Student13{

int id;

String name;

Student13(){System.out.println("Constructor mac dinh duoc goi");}

Student13(int id,String name){

this (); //no duoc su dung de goi constructor cua lop hien tai.

this.id = id;

this.name = name;

}

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){

Student13 e1 = new Student13(111,"Hoang");

Student13 e2 = new Student13(222,"Thanh");

e1.display();

e2.display();

}

}

**Câu hỏi**: Nơi để sử dụng lời gọi this() constructor?

Lời gọi this() constructor nên được sử dụng để tái sử dụng constructor. Nó duy trì dây chuyền giữa các Constructor, và được sử dụng cho Constructor Chaining. Bạn theo dõi ví dụ sau để hiểu hơn về điều này.

class Student14{

int id;

String name;

String city;

Student14(int id,String name){

this.id = id;

this.name = name;

}

Student14(int id,String name,String city){

this(id,name); //Bay gio khong can khoi tao id va name

this.city=city;

}

void display(){System.out.println(id+" "+name+" "+city);}

public static void main(String args[]){

Student14 e1 = new Student14(111,"Hoang");

Student14 e2 = new Student14(222,"Thanh","NamDinh");

e1.display();

e2.display();

}

}

**Qui tắc**: Lời gọi tới this() phải là lệnh đầu tiên trong Constructor.

Trong chương trình ví dụ sau sẽ cho một thông báo lỗi tại compile time.

class Student15{

int id;

String name;

Student15(){System.out.println("Constructor mac dinh duoc trieu hoi");}

Student15(int id,String name){

id = id;

name = name;

this (); //phai la lenh dau tien

}

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){

Student15 e1 = new Student15(111,"Hoang");

Student15 e2 = new Student15(222,"Thanh");

e1.display();

e2.display();

}

}

### Từ khóa this() có thể được sử dụng để triệu hồi ngầm định phương thức lớp hiện tại

Bạn có thể triệu hồi phương thức của lớp hiện tại bởi sử dụng từ khóa this. Nếu bạn không sử dụng từ khóa this, trình biên dịch Compiler tự động thêm từ khóa this trong khi gọi phương thức. Bạn theo dõi ví dụ sau:

class S{

void m(){

System.out.println("Phuong thuc duoc trieu hoi");

}

void n(){

this.m(); //khong can boi vi compiler thuc hien no cho ban.

}

void p(){

n(); //complier se them this de trieu hoi phuong thuc n() duoi dang this.n()

}

public static void main(String args[]){

S s1 = new S();

s1.p();

}

}

### Từ khóa this có thể được truyền như là một tham số trong phương thức

Cách này chủ yếu được sử dụng trong xử lý sự kiện. Bạn theo dõi ví dụ:

class S2{

void m(S2 obj){

System.out.println("Phuong thuc duoc trieu hoi");

}

void p(){

m(this);

}

public static void main(String args[]){

S2 s1 = new S2();

s1.p();

}

}

### Từ khóa this có thể được truyền như là tham số trong lời gọi constructor

Bạn cũng có thể truyền từ khóa this trong constructor. Nó là hữu ích nếu chúng ta phải sử dụng một đối tượng trong nhiều lớp. Bạn theo dõi ví dụ:

class B{

A4 obj;

B(A4 obj){

this.obj=obj;

}

void display(){

System.out.println(obj.data); //su dung thanh vien du lieu cua lop A4

}

}

class A4{

int data=10;

A4(){

B b=new B(this);

b.display();

}

public static void main(String args[]){

A4 a=new A4();

}

}

## Từ khóa this có thể được sử dụng để trả về instance của lớp hiện tại

Chúng ta có thể trả về từ khóa this như một lệnh từ phương thức. Trong tình huống này, kiểu trả về của phương thức phải là kiểu class (không là kiểu gốc). Bạn theo dõi ví dụ:

Cú pháp của this mà có thể được trả về như là một lệnh.

kieu\_tra\_ve ten\_phuong\_thuc(){

return this;

}

Ví dụ

class A{

A getA(){

return this;

}

void msg(){System.out.println("Hello World");}

}

class Test1{

public static void main(String args[]){

new A().getA().msg();

}

}

Chúng ta cùng chứng minh rằng từ khóa this tham chiếu tới biến instance của lớp hiện tại. Trong chương trình sau, chúng ta in biến tham chiếu và this, kết quả của chúng là giống nhau.

class A5{

void m(){

System.out.println(this);//in ra cung tham chieu ID

}

public static void main(String args[]){

A5 obj=new A5();

System.out.println(obj);//in tham chieu ID

obj.m();

}

}

**QUẢNG CÁO**

[Sức Khỏe Nam Giới](https://clck.mgid.com/ghits/14436309/i/57734082/0/pp/1/7?h=X6SvTAJTesAsUc3kFdURnw_HAeSSHGPyI9r4YllIVL14Gezr8S0PNvdthDhIjRJpQDArLFk_a22HnHx0iP8jGA**&rid=db75d539-77d2-11ee-ad18-c84bd683e9ba&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[**Phong độ đàn ông xuống dốc? Phục hồi ngay với mẹo này**](https://clck.mgid.com/ghits/14436309/i/57734082/0/pp/1/7?h=X6SvTAJTesAsUc3kFdURnw_HAeSSHGPyI9r4YllIVL14Gezr8S0PNvdthDhIjRJpQDArLFk_a22HnHx0iP8jGA**&rid=db75d539-77d2-11ee-ad18-c84bd683e9ba&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[Brainberries](https://clck.mgid.com/ghits/15490950/i/57734082/0/pp/2/7?h=X6SvTAJTesAsUc3kFdURn9E2nClJNM35kpXF-tFrCO3BuRQIvW55wlvzwwPAM1j-ChdHukyr68bbvGbIUjuy9A**&rid=db75d539-77d2-11ee-ad18-c84bd683e9ba&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[**Đã tìm ra cây thuốc độc gấp x6000 xyanua. Phổ biến ở Việt Nam?**](https://clck.mgid.com/ghits/15490950/i/57734082/0/pp/2/7?h=X6SvTAJTesAsUc3kFdURn9E2nClJNM35kpXF-tFrCO3BuRQIvW55wlvzwwPAM1j-ChdHukyr68bbvGbIUjuy9A**&rid=db75d539-77d2-11ee-ad18-c84bd683e9ba&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[Tìm Kiếm Quảng Cáo](https://clck.mgid.com/ghits/17555279/i/57734082/0/pp/3/7?h=X6SvTAJTesAsUc3kFdURn8wGosGIJwQS8iV1E4G3YGUlA4PdFEVhVaZp_-kh6YI0Os43bqiocVZMrNYueU9-cA**&rid=db75d539-77d2-11ee-ad18-c84bd683e9ba&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[**Tự đánh giá trầm cảm: Làm bài kiểm tra và nhận trợ giúp**](https://clck.mgid.com/ghits/17555279/i/57734082/0/pp/3/7?h=X6SvTAJTesAsUc3kFdURn8wGosGIJwQS8iV1E4G3YGUlA4PdFEVhVaZp_-kh6YI0Os43bqiocVZMrNYueU9-cA**&rid=db75d539-77d2-11ee-ad18-c84bd683e9ba&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[Brainberries](https://clck.mgid.com/ghits/15787462/i/57734082/0/pp/4/7?h=X6SvTAJTesAsUc3kFdURnzc6RaEaYoPzrRMRFIueRd2_jeezHPNKIiOyNUuXT_nwI7xdfGB8P0wxLk2UrR0F8Q**&rid=db75d539-77d2-11ee-ad18-c84bd683e9ba&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[**Sau nửa năm suy sụp vì liệt cơ mặt, Hari Won quyết tâm điều này**](https://clck.mgid.com/ghits/15787462/i/57734082/0/pp/4/7?h=X6SvTAJTesAsUc3kFdURnzc6RaEaYoPzrRMRFIueRd2_jeezHPNKIiOyNUuXT_nwI7xdfGB8P0wxLk2UrR0F8Q**&rid=db75d539-77d2-11ee-ad18-c84bd683e9ba&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[Brainberries](https://clck.mgid.com/ghits/6174320/i/57734082/0/pp/5/7?h=X6SvTAJTesAsUc3kFdURn7p2p8_B570utkM5h2AUOQhNDnKyaea_ctUKGZycZSUoRNoglVp8fxPRhcVHvUrpMQ**&rid=db75d539-77d2-11ee-ad18-c84bd683e9ba&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[**Đẹp không tì vết, Son Ye Jin là cô gái đẹp nhất thế giới năm 2020**](https://clck.mgid.com/ghits/6174320/i/57734082/0/pp/5/7?h=X6SvTAJTesAsUc3kFdURn7p2p8_B570utkM5h2AUOQhNDnKyaea_ctUKGZycZSUoRNoglVp8fxPRhcVHvUrpMQ**&rid=db75d539-77d2-11ee-ad18-c84bd683e9ba&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

[Brainberries](https://clck.mgid.com/ghits/12336635/i/57734082/0/pp/6/7?h=X6SvTAJTesAsUc3kFdURn2NfLYpGO2164NqXMiYESaEO_L9WCO34UEUOsFgkzAx12JwPmDXNnu3bb2D-i6cgTA**&rid=db75d539-77d2-11ee-ad18-c84bd683e9ba&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&k=1401949fc*ffYuFIzUmfYuFJAZoffOTcw*DU0OQ%3D%3Df!fNjfI%3DfIYffMjM0Ljg2MjUwMzA1MTc1Nzh8MTY%3DffffQf%3AfaHR0cHM6Ly92aWV0amFjay5jb20vamF2Y%2490dV9raG9hX3RoaXN*dHJvbmd*amF2Y%245qc3A%3DfaHR0cHM6Ly92aWV0amFjay5jb20vamF2Y%2490dV9raG9hX3N0YXRpY190cm9uZ19qYXZhLmpzcA%3D%3DfaHR0cHM6Ly92aWV0amFjay5jb20vamF2Y%249pbmRleC5qc3A%3DfqfOTcw*DYzNTl8ODY4*DYzNDY%3DfMQ%3D%3Df%3Df!fcfMTZ8MjM0*Dg2N3w1MzY%3DfYAfNgfYuFIzcjf!fTW96aWxsY%2481LjAgKFdpbmRvd3MgTlQgMTAuMDsgV2luNjQ7IHg2NCkgQXBwbGVXZWJLaXQvNTM3LjM2IChL%24FRNTCwgbGlrZ%24BHZWNrbykgQ2hyb21lLzExOC4wLjAuMCBTYWZhcmkvNTM3LjM2fUERGVmlld2Vy*ENocm9tZVBERlZpZXdlcnxDaHJvbWl1bVBERlZpZXdlcnxNaWNyb3NvZnRFZGdlUERGVmlld2Vy*FdlYktpdGJ1aWx0LWluUERGfCfV2luMzI%3DfNDIwfMXwxMDA%3DfMTUzNnw4MjQ%3DfdW5rbm93bnw0Z3wwf!f!fQfeUEOA~f*f*(YdD2M%3B&crst=1698745812&wrst=1698745811&muid=n78hK_eerlQ6)

[**Những dấu hiệu cho thấy nhất định bạn phải giữ cô gái này**](https://clck.mgid.com/ghits/12336635/i/57734082/0/pp/6/7?h=X6SvTAJTesAsUc3kFdURn2NfLYpGO2164NqXMiYESaEO_L9WCO34UEUOsFgkzAx12JwPmDXNnu3bb2D-i6cgTA**&rid=db75d539-77d2-11ee-ad18-c84bd683e9ba&ts=vietjack.com&tt=Referral&att=4&cpm=1&abd=1&iv=11&ct=1&gdprApplies=0&st=420&mp4=1&h2=Vt3_nBafiKvyGys58teryeSOV3uIVTtzTgUzqbCxGDs*&muid=n78hK_eerlQ6)

Đã có app VietJack trên điện thoại, giải bài tập SGK, SBT Soạn văn, Văn mẫu, Thi online, Bài giảng....miễn phí. Tải ngay ứng dụng trên Android và iOS.

[](https://apps.apple.com/vn/app/vietjack/id1490262941?l=vi&mt=8)

[](https://play.google.com/store/apps/details?id=com.jsmile.android.vietjack)

Theo dõi chúng tôi miễn phí trên mạng xã hội facebook và youtube:

Các bạn có thể mua thêm khóa học JAVA CORE ONLINE VÀ ỨNG DỤNG cực hay, giúp các bạn vượt qua các dự án trên trường và đi thực tập Java. Khóa học có giá chỉ 300K, nhằm ưu đãi, tạo điều kiện cho sinh viên cho thể mua khóa học.

Nội dung khóa học gồm 16 chuơng và 100 video cực hay, học trực tiếp tại [**https://www.udemy.com/tu-tin-di-lam-voi-kien-thuc-ve-java-core-toan-tap/**](https://www.udemy.com/tu-tin-di-lam-voi-kien-thuc-ve-java-core-toan-tap/) Bạn nào có nhu cầu mua, inbox trực tiếp a Tuyền, cựu sinh viên Bách Khoa K53, fb: [**https://www.facebook.com/tuyen.vietjack**](https://www.facebook.com/tuyen.vietjack)

Loạt bài hướng dẫn của chúng tôi dựa một phần trên nguồn tài liệu của: Tutorialspoint.com

Follow facebook cá nhân Nguyễn Thanh Tuyền [**https://www.facebook.com/tuyen.vietjack**](https://www.facebook.com/tuyen.vietjack)để tiếp tục theo dõi các loạt bài mới nhất về Java,C,C++,Javascript,HTML,Python,Database,Mobile.... mới nhất của chúng tôi.

**Bài học Java phổ biến tại vietjack.com:**

* [Nạp chồng phương thức (Overloading)](https://vietjack.com/java/overloading_trong_java.jsp)
* [Constructor](https://vietjack.com/java/constructor_trong_java.jsp)
* [ArrayList trong Java](https://vietjack.com/java/lop_arraylist_trong_java.jsp)
* [Interface trong Java](https://vietjack.com/java/interface_trong_java.jsp)
* [Thread trong Java](https://vietjack.com/java/multithread_trong_java.jsp)
* [247 bài tập Java](https://vietjack.com/bai_tap_java/bai_tap_mau_java_va_vi_du_java.jsp)
* [200 câu phỏng vấn Java](https://vietjack.com/cau_hoi_phong_van_java/index.jsp)

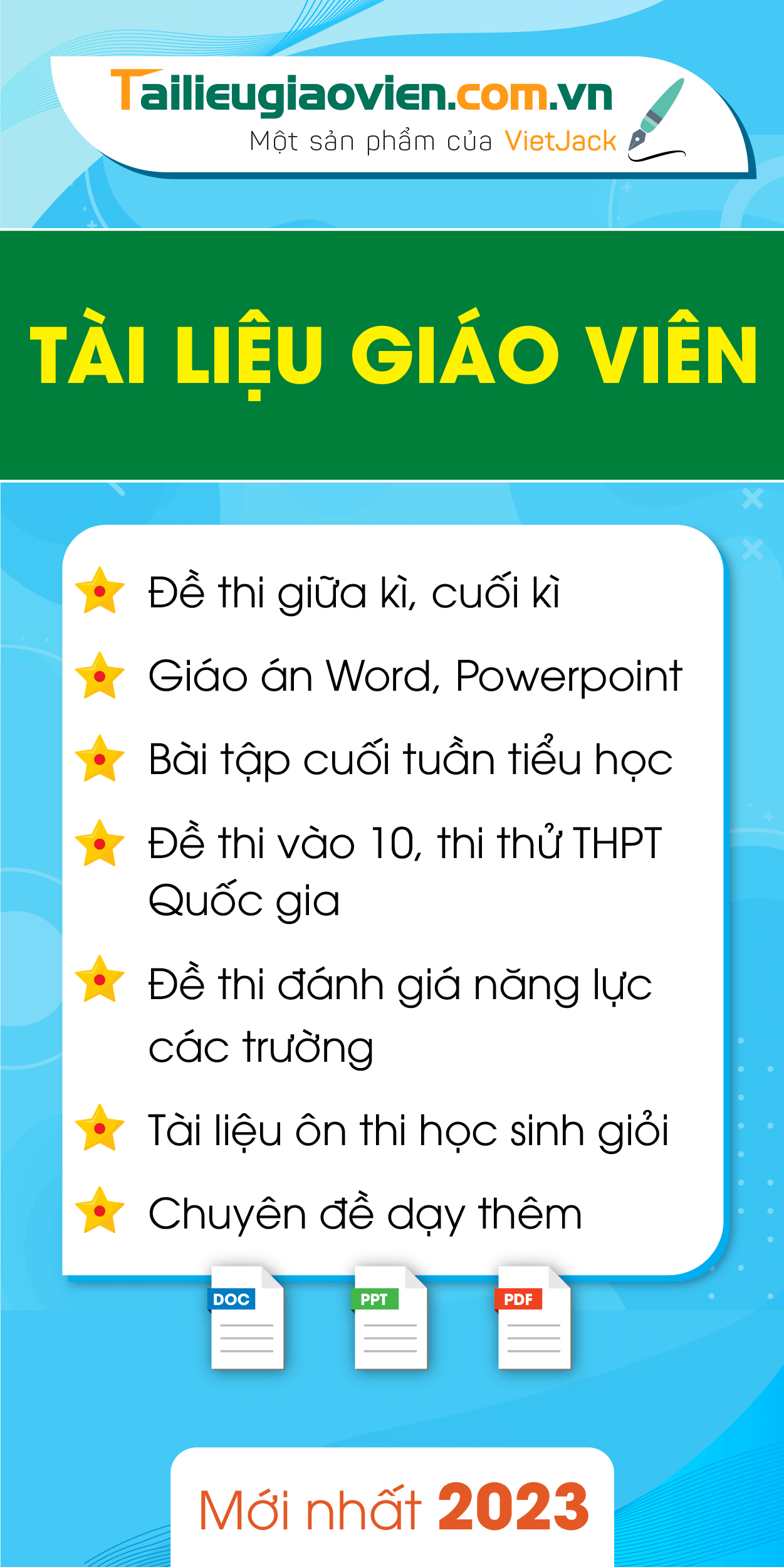
[Trang trước](https://vietjack.com/java/tu_khoa_static_trong_java.jsp)

[Trang sau](https://vietjack.com/java/tinh_ke_thua_trong_java.jsp)

Quảng cáo

Bài viết liên quan

* [160 bài học ngữ pháp tiếng Anh hay nhất](https://vietjack.com/ngu-phap-tieng-anh/index.jsp)
* [155 bài học Java tiếng Việt hay nhất](https://vietjack.com/java/index.jsp)
* [100 bài học Android tiếng Việt hay nhất](https://vietjack.com/android/index.jsp)
* [247 bài học CSS tiếng Việt hay nhất](https://vietjack.com/css/index.jsp)
* [197 thẻ HTML cơ bản](https://vietjack.com/html/index.jsp)
* [297 bài học PHP](https://vietjack.com/php/index.jsp)
* [101 bài học C++ hay nhất](https://vietjack.com/cplusplus/index.jsp)
* [97 bài tập C++ có giải hay nhất](https://vietjack.com/bai_tap_cplusplus_co_giai/index.jsp)
* [208 bài học Javascript có giải hay nhất](https://vietjack.com/javascript/index.jsp)

[](https://tailieugiaovien.com.vn/)

[](https://vietjack.com/)

Trang web chia sẻ nội dung miễn phí dành cho người Việt.

[**Lớp 1-2-3**](https://vietjack.com/series/lop-3.jsp)[**Lớp 4**](https://vietjack.com/series/lop-4.jsp)[**Lớp 5**](https://vietjack.com/series/lop-5.jsp)[**Lớp 6**](https://vietjack.com/series/lop-6.jsp)[**Lớp 7**](https://vietjack.com/series/lop-7.jsp)[**Lớp 8**](https://vietjack.com/series/lop-8.jsp)[**Lớp 9**](https://vietjack.com/series/lop-9.jsp)[**Lớp 10**](https://vietjack.com/series/lop-10.jsp)[**Lớp 11**](https://vietjack.com/series/lop-11.jsp)[**Lớp 12**](https://vietjack.com/series/lop-12.jsp)[**Lập trình**](https://vietjack.com/series/it-lap-trinh.jsp)[**Tiếng Anh**](https://vietjack.com/series/mon-tieng-anh.jsp)

## Chính sách

[Chính sách bảo mật](https://vietjack.com/lien-he/chinh-sach-bao-mat-thong-tin.jsp)

[Hình thức thanh toán](https://vietjack.com/lien-he/quy-dinh-va-hinh-thuc-thanh-toan.jsp)

[Chính sách đổi trả khóa học](https://vietjack.com/lien-he/chinh-sach-doi-tra-khoa-hoc.jsp)

[Chính sách hủy khóa học](https://vietjack.com/lien-he/chinh-sach-huy-khoa-hoc.jsp)

[Tuyển dụng](https://vietjack.com/lien-he/tuyen-dung.jsp)

## Liên hệ với chúng tôi

Tầng 2, số nhà 541 Vũ Tông Phan, Phường Khương Đình, Quận Thanh Xuân, Thành phố Hà Nội, Việt Nam

Phone: 084 283 45 85

Email: [vietjackteam@gmail.com](mailto:vietjackteam@gmail.com)

[Tải nội dung trên Google Play](https://play.google.com/store/apps/details?id=com.jsmile.android.vietjack) [Tải nội dung trên IOS Store](https://apps.apple.com/vn/app/vietjack/id1490262941?l=vi&mt=8)

CÔNG TY TNHH ĐẦU TƯ VÀ DỊCH VỤ GIÁO DỤC VIETJACK

Người đại diện: Nguyễn Thanh Tuyền

Số giấy chứng nhận đăng ký kinh doanh: 0108307822, ngày cấp: 04/06/2018, nơi cấp: Sở Kế hoạch và Đầu tư thành phố Hà Nội.

2015 © All Rights Reserved. [DMCA.com Protection Status](https://www.dmca.com/Protection/Status.aspx?ID=944e9e2f-8254-45fd-b671-33124d5b3df5&refurl=https://vietjack.com/java/tu_khoa_this_trong_java.jsp)

## TÍNH KẾ THỪA TRONG JAVA - TỪ KHÓA EXTENDS VÀ IMPLEMENTS TRONG JAVA

**[Tính kế thừa trong Java - Từ khóa extends và implements trong Java]**Tính kế thừa trong Java là môt kỹ thuật mà trong đó một đối tượng thu được tất cả thuộc tính và hành vi của đối tượng cha. Ý tưởng đằng sau tính kế thừa trong Java là bạn có thể tạo các lớp mới mà được xây dựng dựa trên các lớp đang tồn tại. Khi bạn kế thừa từ một lớp đang tồn tại, bạn có thể tái sử dụng các phương thức và các trường của lớp cha, và bạn cũng có thể bổ sung thêm các phương thức và các trường khác. Tính kế thừa biểu diễn mối quan hệ IS-A, còn được gọi là mối quan hệ cha-con.

Khi chúng ta nói về tính kế thừa, từ khóa thường xuyên nhất được sử dụng là **extends trong java** và **implements trong java**. Những từ khóa này có thể định nghĩa một kiểu là loại IS-A của loại khác. Sử dụng những từ khóa, chúng ta có thể tạo một đối tượng sử dụng thuộc tính của đối tượng khác. Chúng ta sử dụng từ khóa extends của lớp con để có thể kế thừa các thuộc tính của lớp cha trừ các thuộc tính private của lớp cha.

### Tại sao sử dụng tính kế thừa trong Java?

* Để ghi đè phương thức (Method Overriding), do đó có thể thu được tính đa hình tại runtime.
* Để làm tăng tính tái sử dụng của code.

### Cú pháp của Tính kế thừa trong Java

class ten\_lop\_con extends ten\_lop\_cha

{

//cac phuong thuc va cac truong

}

Từ khóa extends chỉ rằng bạn đang tạo một lớp mới mà kế thừa từ một lớp đang tồn tại. Trong Java, một lớp mà được kế thừa được gọi là một lớp cha. Lớp mới được gọi là lớp con.

Trong ví dụ sau, Programmer là lớp con và Employee là lớp cha. Mối quan hệ giữa hai lớp là Programmer IS-A Employee. Nghĩa là Programmer là một kiểu của Employee.

class Employee{

float salary=40000;

}

class Programmer extends Employee{

int bonus=10000;

public static void main(String args[]){

Programmer p=new Programmer();

System.out.println("Luong Lap trinh vien la:"+p.salary);

System.out.println("Bonus cua Lap trinh vien la:"+p.bonus);

}

}

Ở trên, đối tượng Programmer có thể truy cập trường của riêng lớp nó cũng như của lớp Employee, đó là ví dụ cho tính tái sử dụng.

### Các loại kế thừa trong Java

Trên cơ sở các lớp thì có 3 loại kế thừa trong Java, đó là *single (đơn), multilevel (nhiều tầng) và hierarchical (có cấu trúc)*. Trong lập trình Java, *đa kế thừa (multiple) và kế thừa lai (hybrid)* chỉ được hỗ trợ thông qua Interface. Chúng ta sẽ tìm hiểu về Interface trong chương sau đó.

**Ghi chú**: Đa kế thừa không được hỗ trợ trong Java thông qua lớp. Khi một lớp kế thừa từ nhiều lớp, thì đây là đa kế thừa.

**Câu hỏi**: Tại sao đa kế thừa không được hỗ trợ trong Java thông qua lớp?

**Trả lời**: Để giảm tính phức tạp và làm đơn giản hóa ngôn ngữ, đa kế thừa không được hỗ trợ trong Java. Giả sử có tình huống có ba lớp là A, B và C. Lớp C kế thừa lớp A và B. Nếu các lớp A và B có cùng phương thức và bạn gọi nó từ đối tượng lớp con, thì điều này gây là tính lưỡng nghĩa là để gọi phương thức của lớp A hoặc lớp B.

Bởi vì, compile time error thì tốt hơn là runtime error, Java sẽ thông báo một compile time error nếu bạn kế thừa 2 lớp. Do đó, dù bạn có hay không có cùng phương thức hay khác phương thức, thì đó cũng là một lỗi tại compile time.

class A{

void msg(){System.out.println("Hello");}

}

class B{

void msg(){System.out.println("Welcome");}

}

class C extends A,B

{ //gia su neu no da co

Public Static void main(String args[]){

C obj=new C();

obj.msg();//Bay gio phuong thuc msg() nao se duoc goi?

}

}

Chương trình trên sẽ cho một Compile Time Error.

Khi bạn đã hiểu rõ về từ khóa **extends**, chúng ta cùng tìm hiểu về từ khóa **implements** trong quan hệ IS-A.

Từ khóa **implements** được sử dụng bởi các lớp mà kế thừa từ Interface. Interface có thể không bao giờ được kết thừa bởi các lớp.

Ví dụ:

public interface A {}

public class B implements A{

}

public class C extends B{

}

### Từ khóa instanceof trong Java

Bây giờ chúng ta sẽ dùng toán tử **instanceof** để kiểm tra xem B có phải là một A và dog có phải là một A.

interface A{}

class B implements A{}

public class C extends B{

public static void main(String args[]){

B m = new B();

C d = new C();

System.out.println(m instanceof A);

System.out.println(d instanceof B);

System.out.println(d instanceof A);

}

}

Kết quả in ra sẽ như sau:

true

true

true

### Quan hệ HAS-A trong Java

Có những quan hệ chủ yếu dựa vào cách sử dụng. Nó xác định có hay không một lớp cụ thể HAS-A. Quan hệ này giúp chúng ta giảm được dư thừa trong code cũng như tránh các bug.

Cùng xem ví dụ dưới đây:

public class Vehicle{}

public class Speed{}

public class Van extends Vehicle{

private Speed sp;

}

Điều này chỉ ra rằng lớp Van có quan hệ HAS-A với lớp Speed. Việc sử dụng lớp riêng rẽ cho lớp Speed, chúng ta không cần thiết phải đặt toàn bộ code của lớp Speed bên trong lớp Van, điều này tăng tính tái sử dụng của lớp Speed cho nhiều ứng dụng.

Một đặc điểm quan trọng nữa phải ghi nhớ là Java chỉ hỗ trợ kế thừa đơn. Điều này nghĩa là một lớp không thể kế thừa từ nhiều hơn một lớp. Do đó, đoạn code dưới đây là không hợp lệ:

public class C extends A, B{}

Mặc dù vậy một lớp vẫn có thể implement một hoặc nhiều interface. Điều này loại bỏ khả năng không thể đa kế thừa trong Java.