

NGÔN NGỮ LẬP TRÌNH JAVA

CHƯƠNG 1: CÁC KHÁI NIỆM CƠ BẢN VỀ JAVA

1.1 Java là gì? - Tại sao bạn nên học lập trình Java?

1.1.1 Java là gì?

Java là một ngôn ngữ lập trình hiện đại, bậc cao, hướng đối tượng, bảo mật và mạnh mẽ. và là một Platform.

Platform: Bất cứ môi trường phần cứng hoặc phần mềm nào mà trong đó có một chương trình chạy, thì được hiểu như là một Platform. Với môi trường runtime riêng cho mình (JRE) và API, Java được gọi là Platform.

Ngôn ngữ lập trình Java ban đầu được phát triển bởi **Sun Microsystems** do **James Gosling** khởi xướng và phát hành vào năm 1995. Phiên bản mới nhất của Java Standard Edition là Java SE 8. Với sự tiến bộ của Java và sự phổ biến rộng rãi của nó, nhiều cấu hình đã được xây dựng để phù hợp với nhiều loại nền tảng khác nhau. Ví dụ: J2EE cho các ứng dụng doanh nghiệp, J2ME cho các ứng dụng di động.

Các phiên bản J2 mới đã được đổi tên thành Java SE, Java EE và Java ME. Phương châm của java là "**Write Once, Run Anywhere**" - viết một lần chạy nhiều nơi, nghĩa là bạn chỉ cần viết một lần trên window chẳng hạn, sau đó vẫn chương trình đó bạn có thể chạy trên Linux, Android, các thiết bị J2ME...

1.1.2 Các tính năng của Java

Ngôn ngữ lập trình java có các tính năng sau:

- Hướng đối tượng** - Trong Java, mọi thứ đều là một Object. Java có thể dễ dàng mở rộng và bảo trì vì nó được xây dựng dựa trên mô hình Object.
- Nền tảng độc lập** - Không giống nhiều ngôn ngữ lập trình khác bao gồm cả C và C ++, khi Java được biên dịch, nó không được biên dịch thành ngôn ngữ máy nền tảng cụ thể, thay vào mã byte - nền tảng độc lập. Mã byte này được thông dịch bởi máy ảo (JVM) trên nền tảng nào đó mà nó đang chạy.
- Đơn giản** - Java được thiết kế để dễ học. Nếu bạn hiểu khái niệm cơ bản về OOP Java, sẽ rất dễ để trở thành master về java.

- **Bảo mật** - Với tính năng an toàn của Java, nó cho phép phát triển các hệ thống không có virut, giả mạo. Các kỹ thuật xác thực dựa trên mã hoá khóa công khai.
- **Kiến trúc - trung lập** - Trình biên dịch Java tạo ra định dạng tệp đối tượng kiến trúc trung lập, làm cho mã biên dịch được thực thi trên nhiều bộ vi xử lý, với sự hiện diện của hệ điều hành Java.
- **Portable** - Là kiến trúc tập trung và không có khía cạnh thực hiện phụ thuộc của đặc tả này làm cho Java khả chuyển. Trình biên dịch trong Java được viết bằng ANSI C, đó là một tập con POSIX.
- **Mạnh mẽ** - Java làm nỗ lực để loại trừ các tình huống dễ bị lỗi bằng cách kiểm tra lỗi tại thời gian biên dịch và kiểm tra lỗi tại runtime.
- **Đa luồng** - Với tính năng đa luồng của Java có thể viết các chương trình có thể thực hiện nhiều tác vụ đồng thời. Tính năng thiết kế này cho phép các nhà phát triển xây dựng các ứng dụng tương tác có thể chạy trơn tru hơn.
- **Thông dịch** - Mã byte Java được dịch trực tiếp tới các máy tính gốc và không được lưu trữ ở bất cứ đâu.
- **Hiệu năng cao** - Với việc sử dụng trình biên dịch Just-In-Time, Java cho phép thực hiện hiệu năng cao.
- **Phân tán** - Java được thiết kế cho môi trường phân tán của Internet.
- **Năng động** - Java là năng động hơn C hoặc C++ vì nó được thiết kế để thích nghi với môi trường đang phát triển. Các chương trình Java có thể mang một lượng lớn thông tin tại runtime mà có thể được sử dụng để xác minh và giải quyết các truy cập vào các đối tượng tại runtime.

1.1.3 Java được sử dụng để làm gì?

Trước khi tôi trả lời câu hỏi, Java được sử dụng để làm gì, hay lý do tại sao bạn nên chọn Java. Java rất phổ biến và đã thống trị lĩnh vực này từ đầu những năm 2000 đến nay 2020.

Theo tập đoàn SUN, hiện nay có khoảng 3 tỷ thiết bị đang chạy java.

Java đã được sử dụng trong các lĩnh vực khác nhau. Ví dụ:

1. Desktop App như acrobat reader, media player, antivirus, ...
2. Web App như irctc.co.in, javatpoint.com, ...
3. Enterprise App như các ứng dụng về xử lý nghiệp vụ ngân hàng, ...
4. Thiết bị Mobile như các ứng dụng Android.
5. Hệ thống nhúng
6. Smart Card

- 7. Robot
 - 8. Game App
-

1.1.4 Các kiểu của Java App

Có 4 kiểu ứng dụng chính của java app:

1. Standalone App

Standalone App cũng được biết đến như Desktop App hoặc Window-based App. Để tạo ra ứng dụng kiểu này người ta thường sử dụng AWT, Swing hoặc JavaFX framework.

2. Web App

Web App là ứng dụng chạy trên server và tạo được các trang động. Hiện nay, servlet, jsp, struts, jsf, spring... là những công nghệ được sử dụng để tạo Web App trong java.

3. 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.

4. Mobile App

Mobile App là ứng dụng được tạo ra cho các thiết bị di động. Hiện nay Android và Java ME được sử dụng để chạy các ứng dụng này.

1.1.5 Java Platforms

Có 4 nền tảng Java:

1. Java SE (Java Standard Edition)

Java SE là một nền tảng lập trình Java. Nó bao gồm các API lập trình Java như java.lang, java.io, java.net, java.util, java.sql, java.math, v.v. Nó bao gồm các chủ đề cốt lõi như OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, v.v.

2. Java EE (Java Enterprise Edition)

Đây là một nền tảng doanh nghiệp chủ yếu được sử dụng để phát triển các ứng dụng web và doanh nghiệp. Nó được xây dựng trên nền tảng Java SE. Nó bao gồm các chủ đề như Servlet, JSP, Web Services, EJB, JPA, v.v.

3. Java ME (Java Micro Edition)

Đây là một nền tảng vi mô chủ yếu được sử dụng để phát triển các ứng dụng di động.

4. JavaFX

JavaFX là một nền tảng phần mềm phát triển các ứng dụng Internet phong phú (Rich Internet Applications – RIAs) có thể chạy trên nhiều loại thiết bị, nhiều hệ điều hành khác nhau. JavaFX là một giải pháp công nghệ cho GUI trên nền tảng Java nhằm tạo giao diện đồ họa người dùng dựa trên Swing và Java2D.

1.1.6 Tại sao bạn nên học lập trình Java?

Ngoài việc độc lập nền tảng, phong cách "lập trình hướng đối tượng" của Java và sự hấp dẫn rất cao đối với các nhà tuyển dụng IT.

Như tên gọi của nó đã cho thấy, lập trình hướng đối tượng (OOP) sử dụng các đối tượng được định nghĩa đầy đủ - và các mối quan hệ giữa các đối tượng với nhau - để thực hiện các tác vụ khác nhau. Do nền tảng mô-đun tự nhiên của nó, OOP thường làm cho nó dễ dàng hơn, nhanh hơn và rẻ hơn trong phát triển và quản lý phần mềm. Cũng dễ hiểu khi những đặc điểm này buộc các tổ chức và doanh nghiệp ôm lấy Java trong vòng tay rộng mở, nâng ngôn ngữ lập trình này trở thành kỹ năng được mong muốn nhất của các nhà tuyển dụng.

Và trong khi Java là rất "hot" với các nhà tuyển dụng, nó cũng không kém phần nóng bỏng đối với những người thực sự viết ra các chương trình máy tính. Mức lương trung bình của một lập trình viên Java tại Mỹ là \$88K đô-la (hơn 1,8 tỷ VNĐ/năm) và Java tạo ra một nguồn cảm hứng bất tận để thu hút rất nhiều người theo đuổi nghề nghiệp cùng với nó.

➤ Điều kiện tiên quyết để học lập trình Java

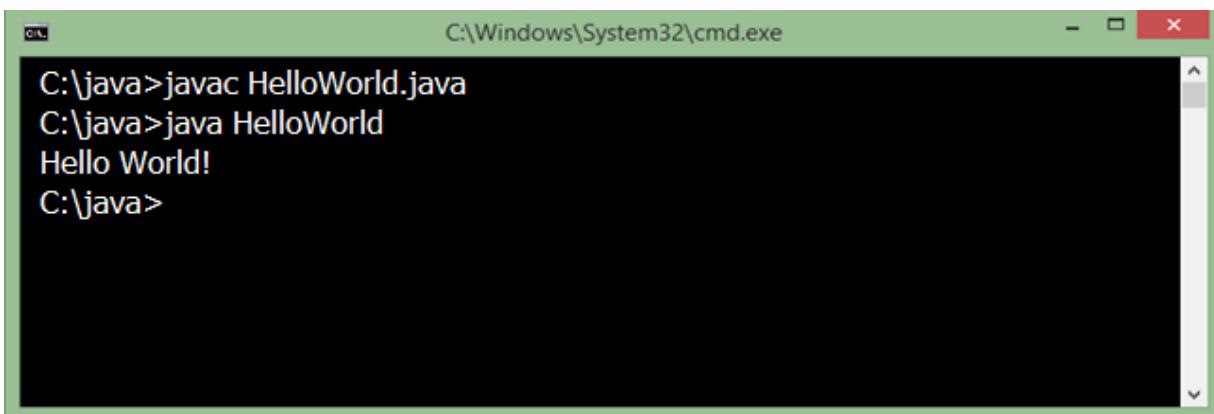
Bạn nên có kiến thức cơ bản về lập trình C/C++, và kiến thức tốt cấu trúc dữ liệu và giải thuật.

1.1.7 Ví dụ về Java

Dưới đây là ví dụ về chương trình đơn giản trong java để in "Hello World":

```
1  public class HelloWorld {  
2      public static void main(String args[]) {  
3          System.out.println("Hello Java");  
4      }  
5  }
```

Kết quả:



A screenshot of a Windows Command Prompt window titled 'C:\Windows\System32\cmd.exe'. The window contains the following text:
C:\java>javac HelloWorld.java
C:\java>java HelloWorld
Hello World!
C:\java>

1.2 Lịch sử Java

1.2.1 Lịch sử ngôn ngữ lập trình Java

Java được khởi xướng bởi **James Gosling** và các bạn đồng nghiệp ở Sun Microsystems năm 1991. Từ ý tưởng muốn lập trình để điều khiển mà không phụ thuộc vào loại CPU cho các thiết bị điện tử như tivi, máy giặt, lò nướng... Do đó, họ đã bắt tay vào xây dựng một ngôn ngữ chạy nhanh, gọn, hiệu quả, độc lập thiết bị và ngôn ngữ “**Oak**” ra đời, sau đó được đổi tên thành Java.

1.2.1.1 Tại sao ban đầu lại có tên là Oak?

1. Bởi vì **Oak** là tên một loại cây rất phổ biến ở các nước như Mỹ, Đức, Romania, ... và được xem như biểu tượng của các nước đó cũng như là biểu hiện cho sức mạnh. Xung quanh văn phòng của James Gosling cũng trồng rất nhiều cây sồi.
2. Cho mãi tới năm 1995, Oak đã được đổi tên thành Java bởi vì cái tên Oak đã được đăng ký trước đó bởi một công ty tên Oak Technologies.
3. Java là tên một hòn đảo của Indonesia nơi đầu tiên sản xuất ra cà phê. Đó là lý do tại sao biểu tượng của java là cốc coffee bốc khói ngút thơm lừng.

1.2.1.2 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 những lựa chọn hàng đầu cùng với Silk. Tuy nhiên, vì Java có tính duy nhất hơn, nên hầu như tất cả thành viên trong 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 James Gosling tại Sun Microsystems (bây giờ là công ty con của Oracle Corporation) 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.

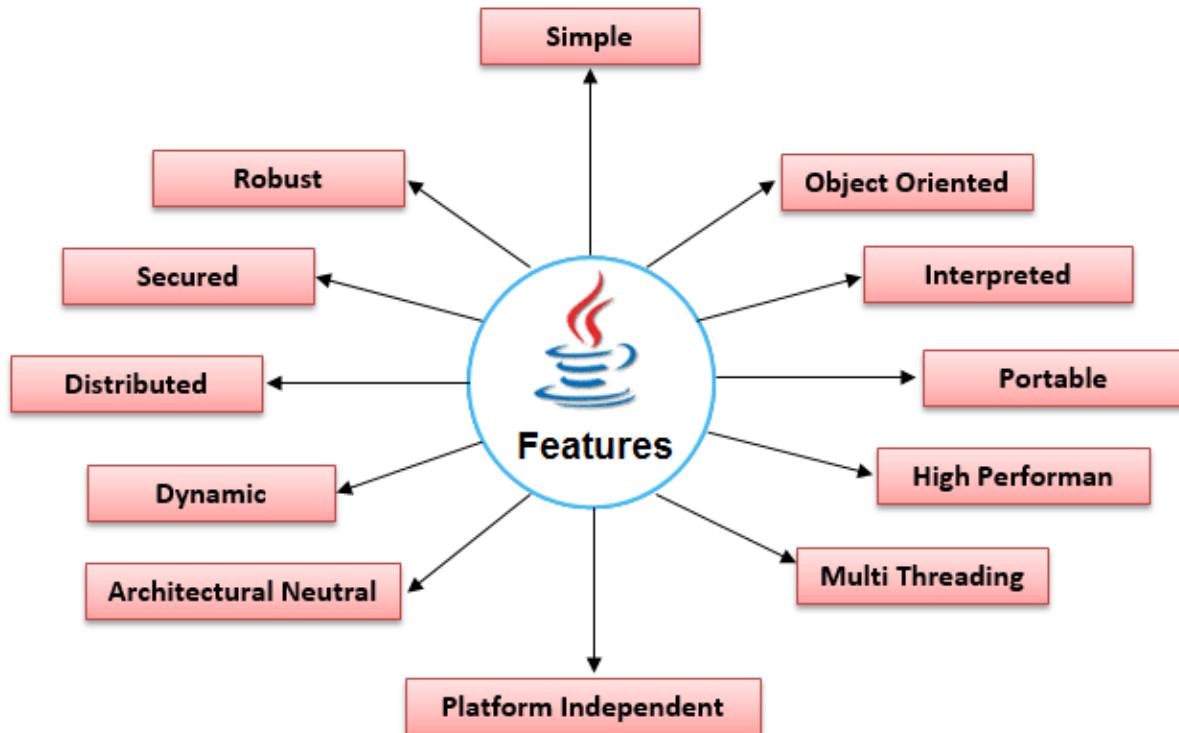
1.2.2 Lịch sử các phiên bản Java

Đã có nhiều phiên bản Java được tạo ra. Phiên bản Java hiện tại là Java SE 8.

1. JDK Alpha và Beta (1995)
2. JDK 1.0 (23/1/1996)
3. JDK 1.1 (19/2/1997)
4. J2SE 1.2 (8/12/1998)
5. J2SE 1.3 (8/5/2000)
6. J2SE 1.4 (6/5/2002)
7. J2SE 5.0 (30/9/2004)
8. Java SE 6 (11/12/2006)
9. Java SE 7 (28/7/2011)
10. Java SE 8 (18/3/2014)
11. JDK 9, 21 tháng 9 năm 2017
12. JDK 10, 20 tháng 3 năm 2018

1.3 Các tính năng của java

Java có rất nhiều tính năng nổi bật. Chúng được biết đến như những thuật ngữ quen thuộc. Các tính năng này đơn giản và dễ dàng để tìm hiểu. Dưới đây là danh sách **các tính năng của java**:



1. Đơn giản (Simple)
2. Hướng đối tượng (Object Oriented)
3. Độc lập nền tảng (Platform Independent)
4. Bảo mật (Secured)
5. Robust (Mạnh mẽ)
6. Kiến trúc - tập trung (Architecture-neutral)
7. Khả chuyển (Portable)
8. Năng động (Dynamic)
9. Thông dịch (Interpreted)
10. Hiệu suất cao (High Performance)
11. Đa luồng (Multi-thread)
12. Phân tán (Distributed)

1.3.1 Đơn giản

Ngôn ngữ Java có đặc điểm đơn giản là vì:

1. Cú pháp dựa trên C++ (vì vậy việc học Java sẽ rất dễ dàng sau khi lập trình viên học C++)
2. Gỡ 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, ...
3. Bạn không cần xóa các đối tượng mà không được tham chiếu, bởi vì Bộ dọn rác tự động (Garbage Collection) trong Java sẽ làm việc đó thay bạn.

1.3.2 Hướng đối tượng

Hướng đối tượng nghĩa là chúng ta tổ chức phần mềm dưới dạng kết hợp của nhiều loại đối tượng khác nhau, trong đó có sự kết hợp chặt chẽ cả về dữ liệu và hành vi của chúng.

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

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

1. Đối tượng (Object)
2. Lớp (Class)
3. Tính kế thừa
4. Tính đa hình
5. Tính trừu tượng
6. Tính đóng gói

1.3.3 Độc lập nền tảng

Một Platform là môi trường phần cứng hoặc phần mềm mà một hoặc nhiều chương trình chạy trong đó. 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 trên các nền tảng hardware-based khác nhau. Nó có hai thành phần:

1. JRE (Java Runtime Environment)
2. 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 Trình 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. Đó là lý do vì sao java có khẩu hiệu "Viết một lần, Chạy khắp nơi (Write Once and Run Anywhere)".

1.3.4 Bảo mật

Java bảo mật bởi vì:

- Không có con trỏ tự động minh.
- Chương trình chạy bên trong máy ảo.
- **Classloader:** Thêm sự bảo mật bằng việc phân chia package cho các class của hệ thống file trên local mà từ đó chúng được import với các file từ nguồn mạng.
- **Bytecode Verifier:** 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, một vài tính năng bảo mật khác được cung cấp thông qua nhà phát triển như SSL, JAAS, cryptography, ...

1.3.5 Robust (Mạnh mẽ)

Bạn có thể hiểu đơn giản Robust nghĩa là mạnh mẽ. Java sử dụng trình quản lý 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 Java còn có Trình dọn rác tự động (Garbage Collection). Có xử lý ngoại lệ (Exception Handling) và cơ chế kiểm tra kiểu ngoại lệ xảy ra. Đó là những điểm nổi bật khiêm cho Java mạnh mẽ!

1.3.6 Kiến trúc - tập trung

Một ứng dụng được biên dịch trên kiến trúc phần cứng này và ứng dụng đó chạy được trên tất cả các kiến trúc phần cứng khác. vd: Một ứng dụng được biên dịch với vi xử lý 32bit và nó sẽ chạy tốt trên vi xử lý 64bit.

1.3.7 Portable

Java là ngôn ngữ lập trình có tính Portable bởi vì java có thể thực thi ứng dụng của nó trên tất cả các hệ điều hành và phần cứng khác nhau.

1.3.8 Hiệu suất cao

Hiệu suất Java nhanh hơn kể từ khi được thông dịch thành ByteCode, mã nguồn gốc thì chậm hơn so với một số ngôn ngữ biên dịch (ví dụ như C++).

1.3.9 Đa luồng (Multi-thread)

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.

1.3.10 Phân tán (Distributed)

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, ...

1.4 Cài đặt môi trường Java

Bài học này plpsoft.vn hướng dẫn bạn:

1. Cách **cài đặt môi trường java** trên Windows.
2. Tạo chương trình java đầu tiên trên eclipse.
3. Cách cài đặt môi trường java trên Linux (Ubuntu).

1.4.1. Cài đặt môi trường java trên Windows

Để làm việc với java bạn phải cài đặt môi trường cho nó, môi trường java bao gồm những gì? Phần này plpsoft.vn hướng dẫn bạn cài đặt môi trường cho java trên môi trường Windows. Còn trên môi trường Linux(Ubuntu) ở phía cuối của bài học.

Dưới đây là những gì cơ bản và cần thiết bạn cần phải cài đặt:

- Cài đặt JDK (Java Development Kit)
- Cấu hình biến môi trường.

Về cơ bản thì chỉ cần 2 bước trên là đủ để tạo ra và chạy một chương trình java đơn giản. Dưới đây là chi tiết cho các bước trên.

1.4.1.1 Cài đặt JDK (Java Development Kit)

JDK là gì? JDK (là viết tắt của Java Development Kit) là một bộ phát triển phần mềm để phát triển các ứng dụng trong Java. JDK bao gồm JRE và các Development Tool. Ngoài JRE, JDK cũng chứa số công cụ phát triển (trình biên

dịch, JavaDoc, Java Debugger, ...) tham khảo tại [sự khác nhau giữa JDK, JRE và JVM](#). Nên bạn chỉ cần tải JDK về là đủ.

Hiện nay có 2 phiên bản java được sử dụng phổ biến đó là JDK7 và JDK8, dưới đây là link download JDK8:

- Link: <https://www.oracle.com/java/technologies/javase-jdk15-downloads.html>
- Click vào 'Accept License Agreement':

Java SE Development Kit 15.0.1		
This software is licensed under the Oracle Technology Network License Agreement for Oracle Java SE		
Product / File Description	File Size	Download
Linux ARM 64 RPM Package	141.81 MB	 jdk-15.0.1_linux-aarch64_bin.rpm
Linux ARM 64 Compressed Archive	157.01 MB	 jdk-15.0.1_linux-aarch64_bin.tar.gz
Linux x64 Debian Package	154.79 MB	 jdk-15.0.1_linux-x64_bin.deb
Linux x64 RPM Package	162.02 MB	 jdk-15.0.1_linux-x64_bin.rpm
Linux x64 Compressed Archive	179.33 MB	 jdk-15.0.1_linux-x64_bin.tar.gz
macOS Installer	175.94 MB	 jdk-15.0.1_osx-x64_bin.dmg
macOS Compressed Archive	176.53 MB	 jdk-15.0.1_osx-x64_bin.tar.gz
Windows x64 Installer	159.69 MB	 jdk-15.0.1_windows-x64_bin.exe
Windows x64 Compressed Archive	179.27 MB	 jdk-15.0.1_windows-x64_bin.zip

- Nếu hệ điều hành của bạn là 32bit thì bạn tải về bản x86, 64bit thì bạn có thể tải 1 trong 2 hoặc cả 2 x86 và x64 đều được.

Sau khi tải JDK về, bạn cài đặt giống như cài đặt các phần mềm khác.

1.4.1.2 Cấu hình biến môi trường

Việc cấu hình biến môi trường bao gồm 2 việc sau:

- Tạo biến JAVA_HOME
- Sửa biến Path

a. Tạo biến JAVA_HOME

Click chuột phải vào biểu tượng MyComputer --> Properties -> Advanced system settings -> **Environment Variables** -> Click 'New' trong System Variables

About

Your PC is monitored and protected.

[See details in Windows Security](#)

This page has a few new settings

Some settings from Control Panel have moved here, and you can copy your PC info so it's easier to share.

Device specifications

Device name	PLPSOFT-PC1
Processor	Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz
Installed RAM	8,00 GB (7.85 GB usable)
Device ID	B2D5F19E-1DB9-4E41-BBEE-3EAB2566C176
Product ID	00330-80000-00000-AA470
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

[Copy](#)

[Rename this PC](#)

[Related settings](#)

[BitLocker settings](#)

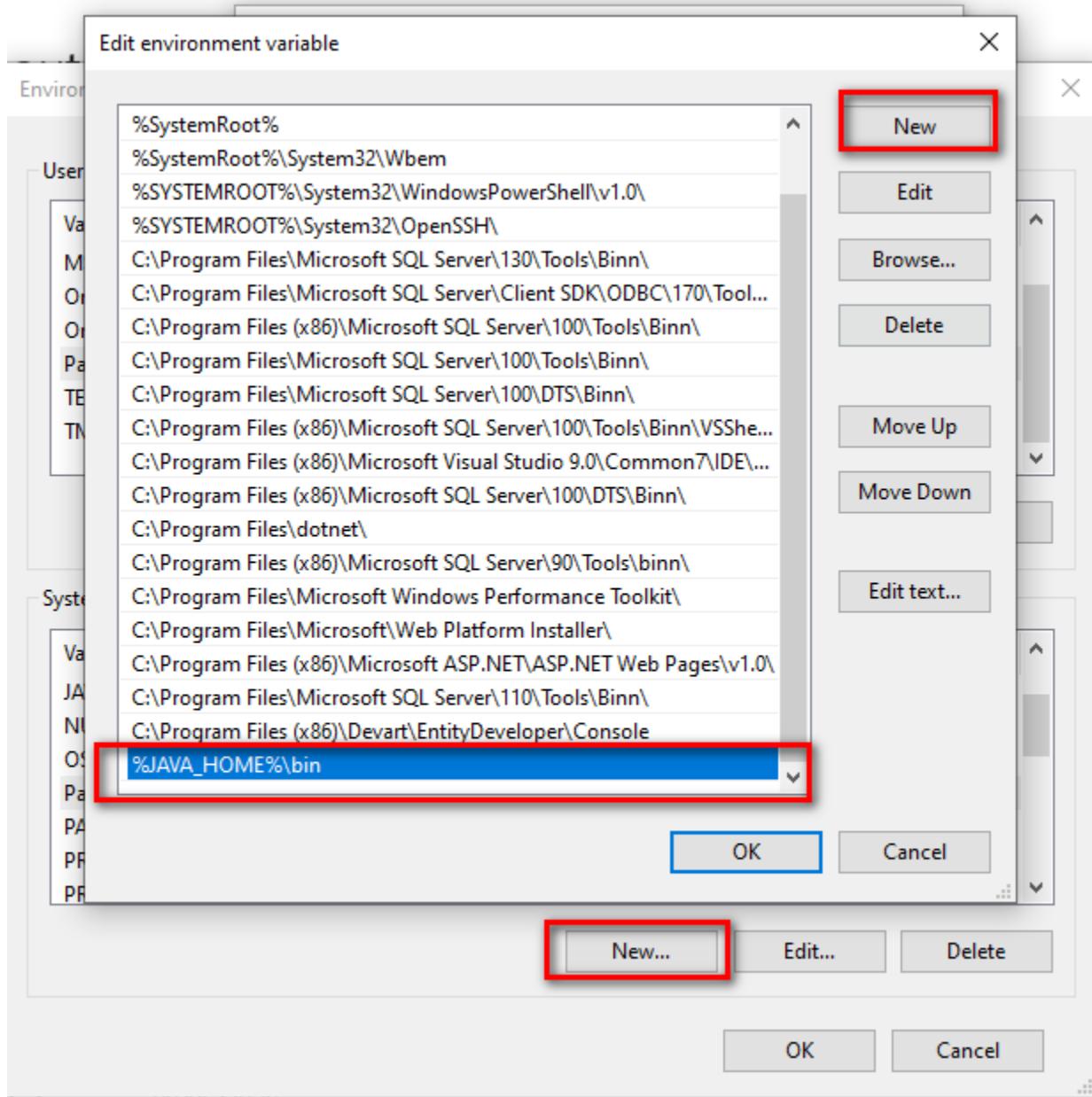
[Device Manager](#)

[Remote desktop](#)

[System protection](#)

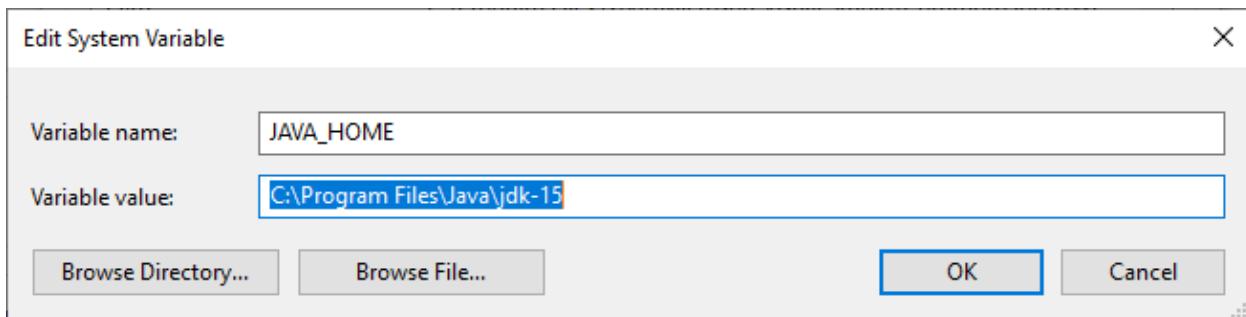
[Advanced system settings](#)

[Rename this PC \(advanced\)](#)



Nhập: Variable name = "JAVA_HOME" và Variable value = [java-path]

Ví dụ



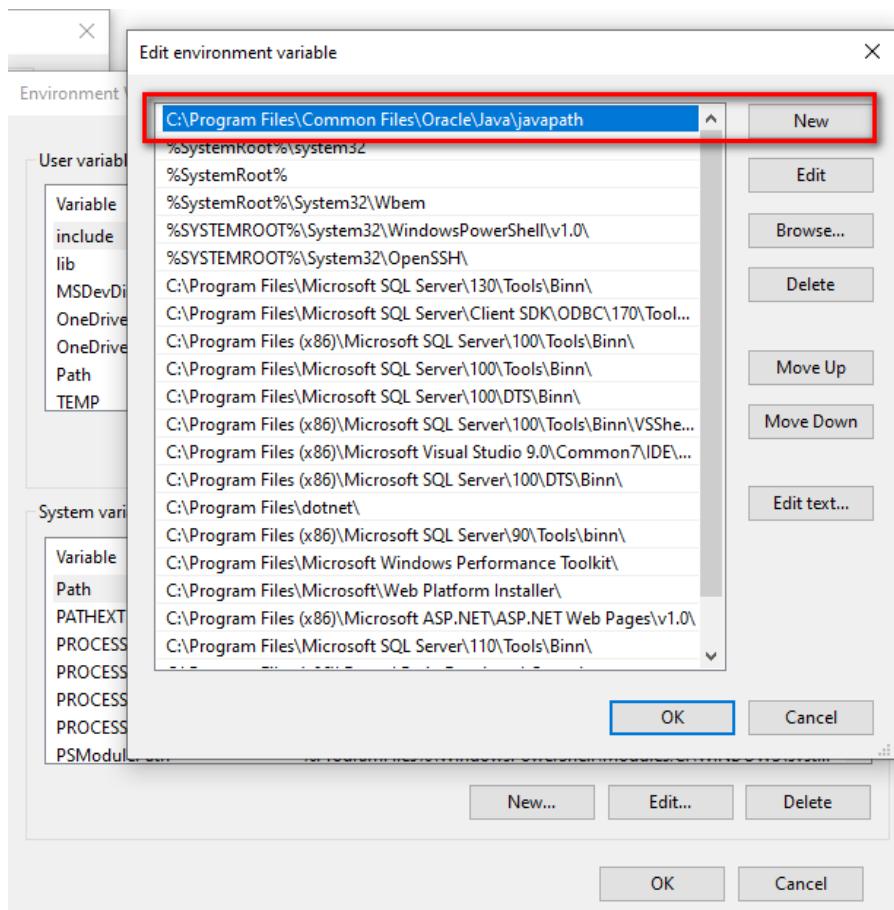
Click: OK -> OK -> OK

b. Sửa biến Path

Click chuột phải vào biểu tượng MyComputer --> Properties --> Advanced system settings --> Environment Variables --> edit biến Path trong System Variables --> thêm [java-path]\bin vào biến Path --> OK --> OK --> OK

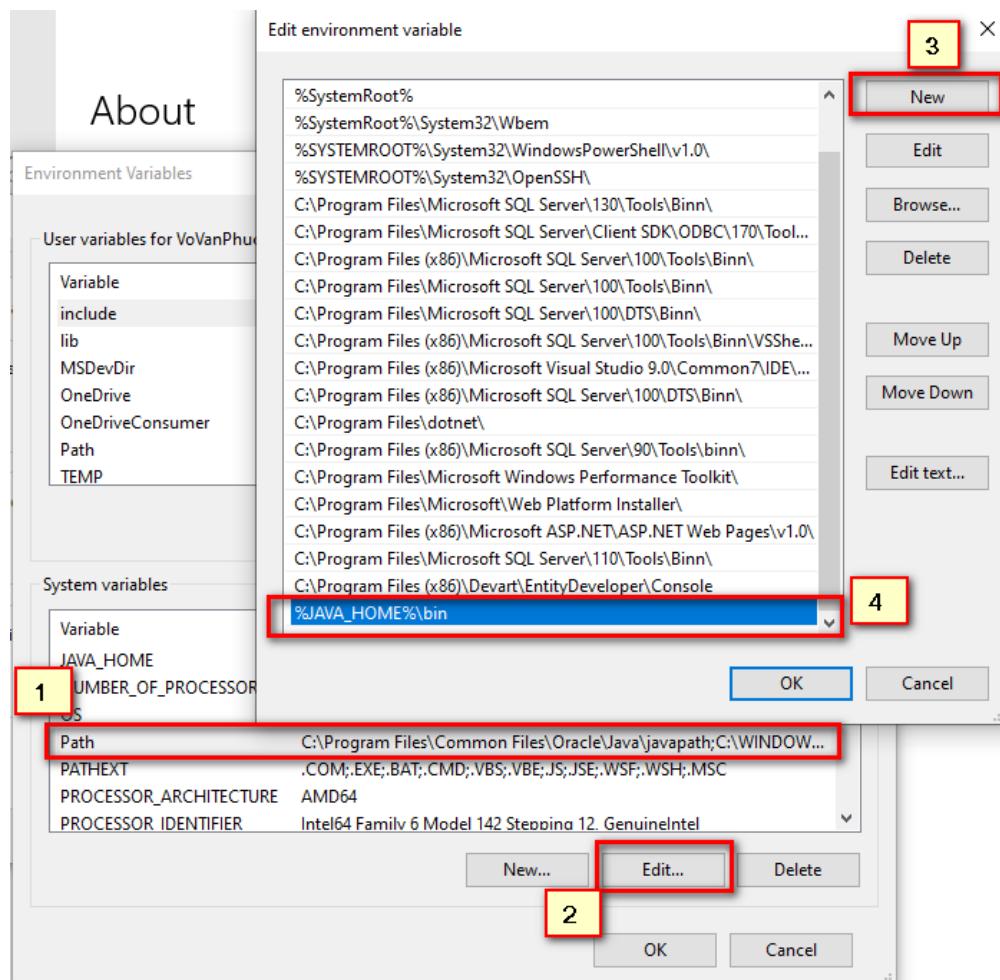
Ví dụ

Edit biến path trong System Variables



Nhập giá trị sau vào cuối biến path:

C:\Program Files\Java\jdk-15\bin



Click button OK -> OK -> OK.

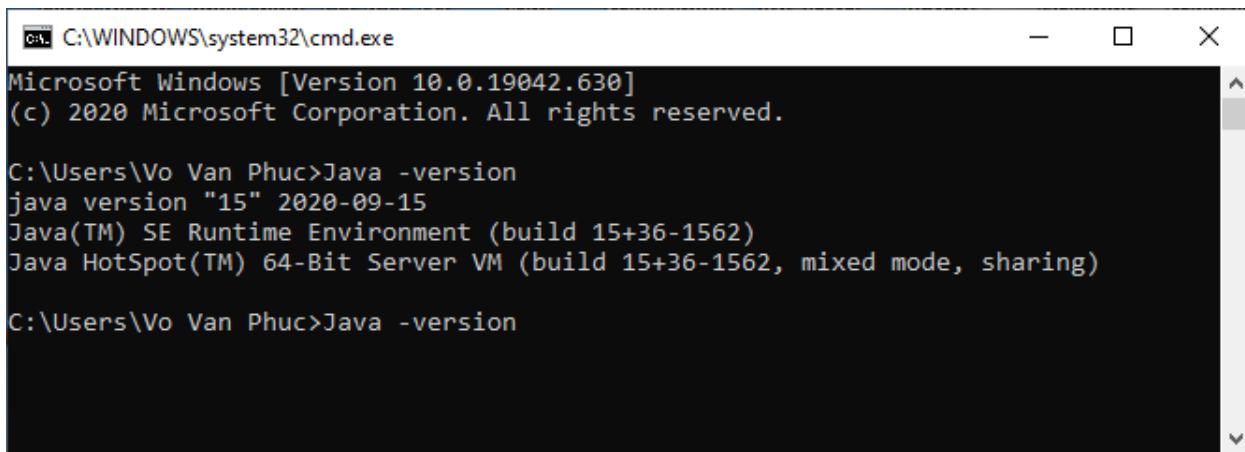
Bạn cần phải restart (hoặc sign out) máy tính sau khi cài đặt biến môi trường.

c. Kiểm tra lại xem cấu hình môi trường Java đúng chưa?

Nhấn tổ hợp phím Windows + R

Trong hộp thoại command, gõ lệnh cmd, nhấn Enter.

Dùng lệnh Java – version



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19042.630]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Vo Van Phuc>Java -version
java version "15" 2020-09-15
Java(TM) SE Runtime Environment (build 15+36-1562)
Java HotSpot(TM) 64-Bit Server VM (build 15+36-1562, mixed mode, sharing)

C:\Users\Vo Van Phuc>Java -version
```

1.4.2. Tạo chương trình java đầu tiên trên eclipse

Khi mới học java có khi nào bạn tự hỏi các đoạn code java được viết vào đâu (editor nào)? biên dịch và chạy nó như nào? Bạn có thể tham khảo bài [Hello World trong java](#) để biết cách tạo chương trình java bằng notepad và hiểu một chương trình java được biên dịch như thế nào.

Nhưng sau đó bạn nên biết và sử dụng **eclipse** hoặc **netbean**, vì chúng là 2 IDE miễn phí phổ biến mạnh mẽ nhất được sử dụng trong việc lập trình java. Phần này plpsoft.vn hướng dẫn bạn cách **tạo và chạy chương trình java đầu tiên trên eclipse**.

Tiếp theo là bạn cần phải tải về và cài đặt eclipse. Tham khảo bài [download và cài đặt Eclipse IDE cho lập trình Java](#).

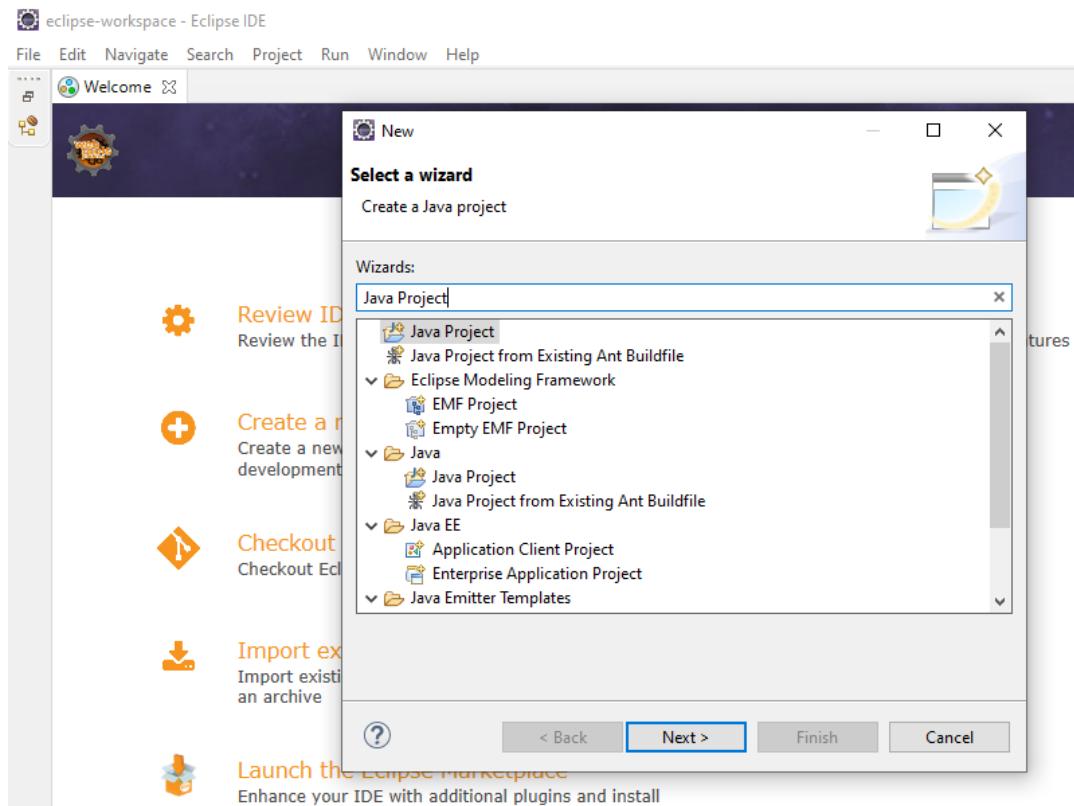
1.4.2.1 Tạo chương trình java đầu tiên trên eclipse

Dưới đây là các bước tạo chương trình java đầu tiên trên eclipse:

Chọn **File -> New -> Other...**

Nhập Wizards = "java project" để tìm kiếm nhanh:

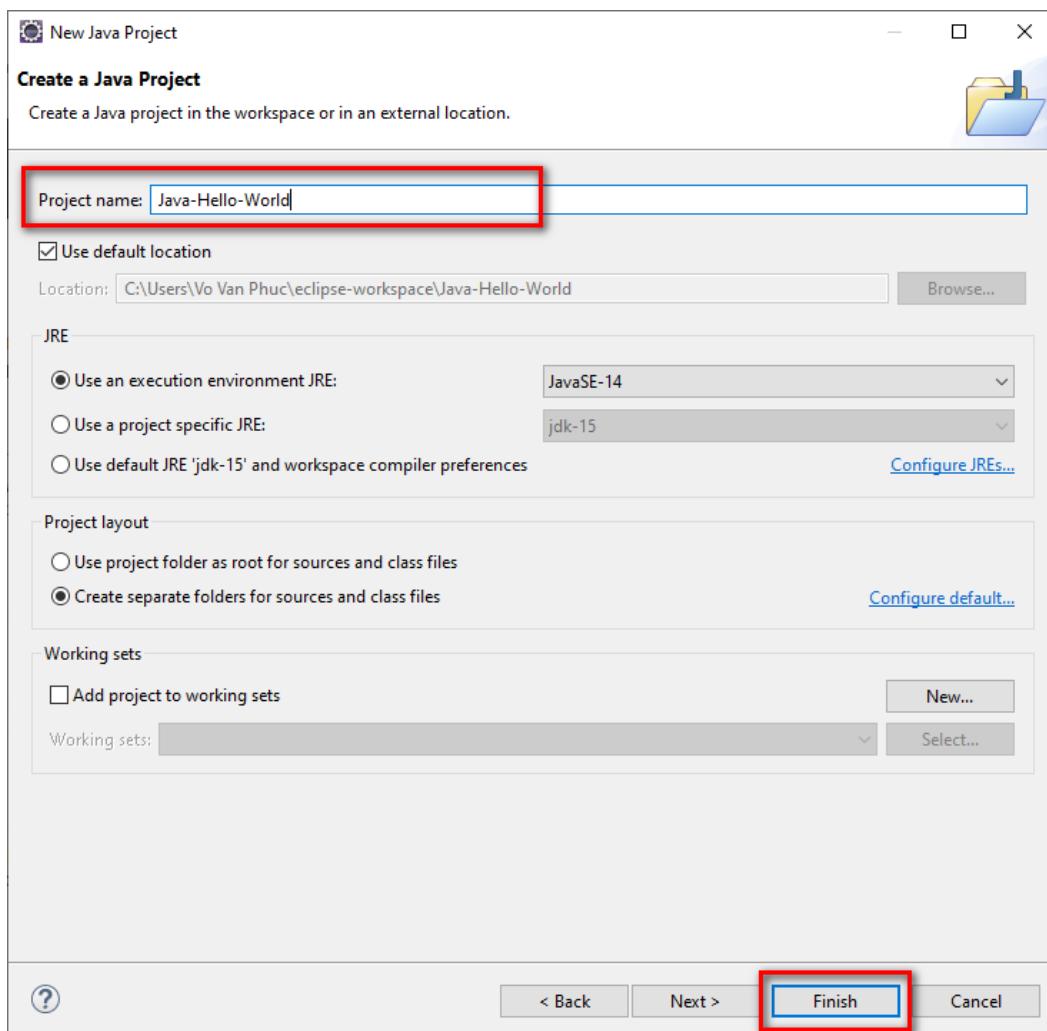
Sau đó chọn "**Java Project**" -> **Next:**



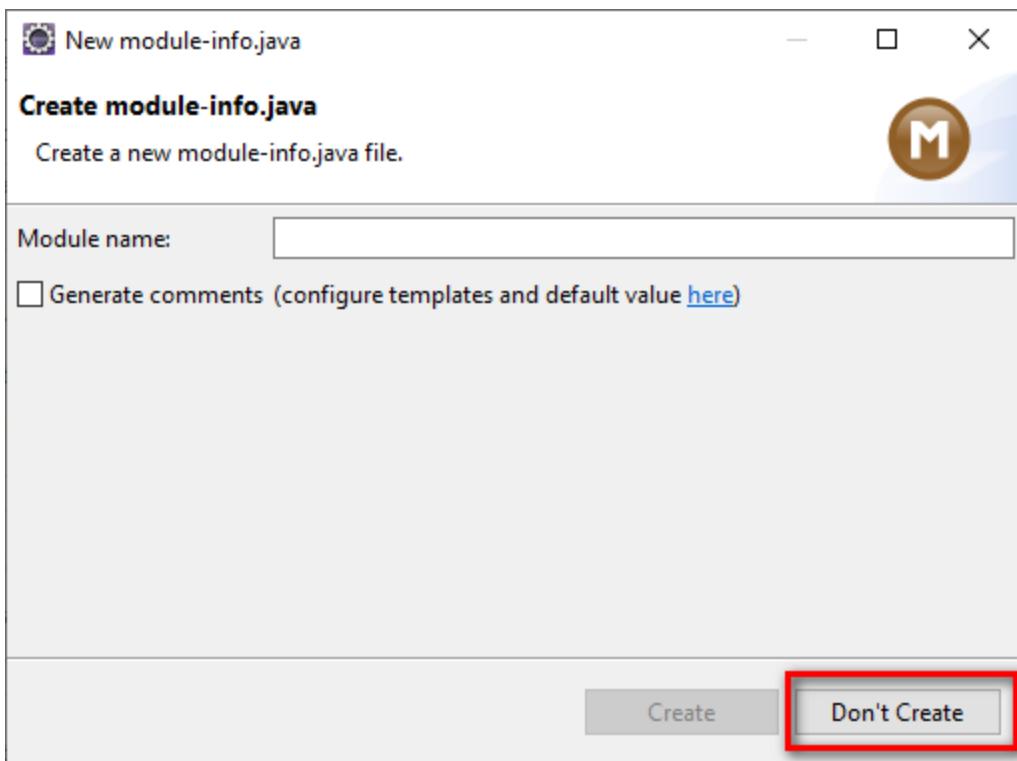
(Hoặc: File->New ->Project; Chọn Java Project)



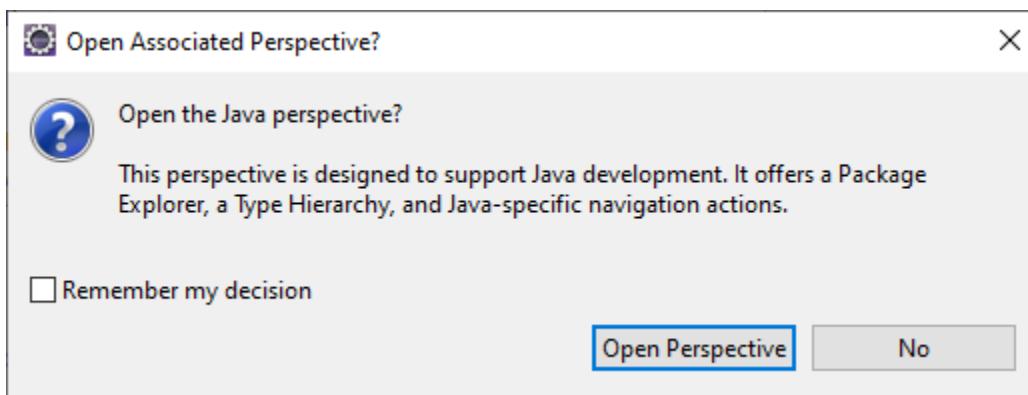
Nhập project name: "Java-Hello-World" -> Finish



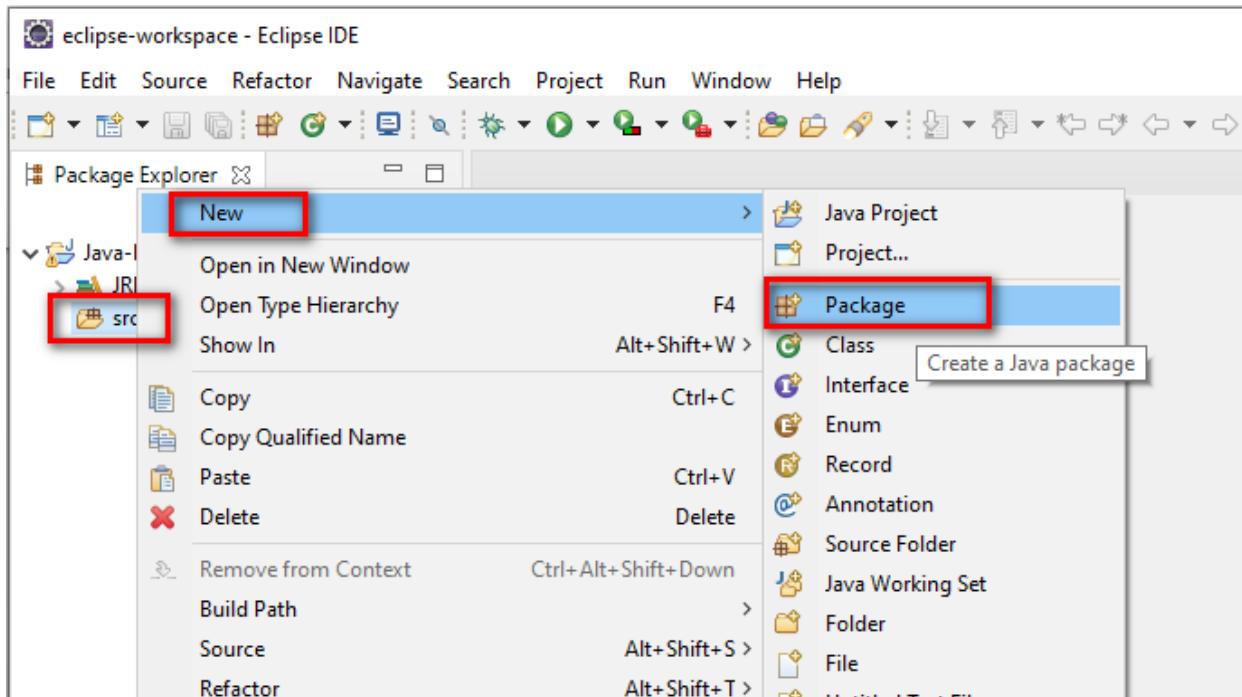
Chọn tiếp: **Don't Create**



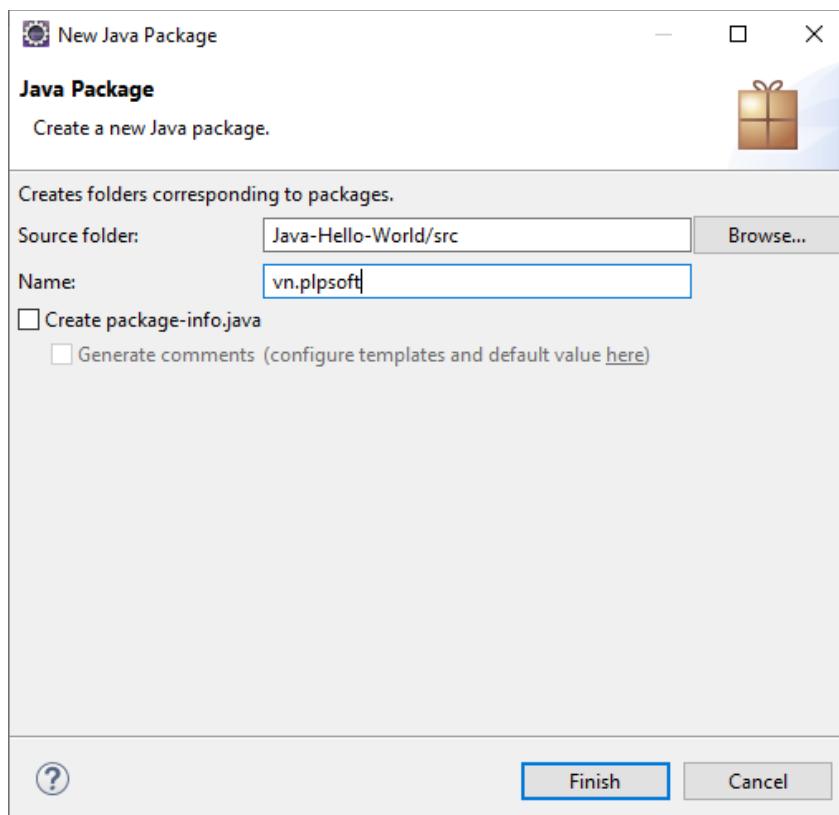
Click OK tại dialog "Open Associated Perspective"



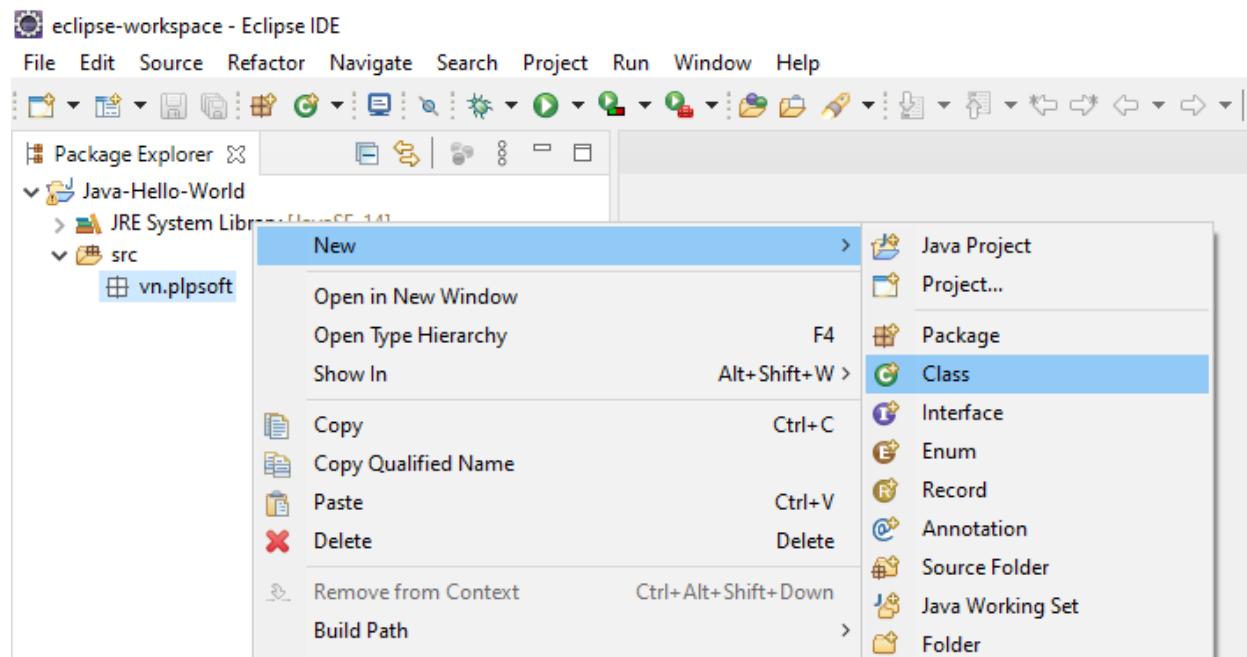
Tạo package cho project: **click chuột phải vào "src" -> New -> packpage**



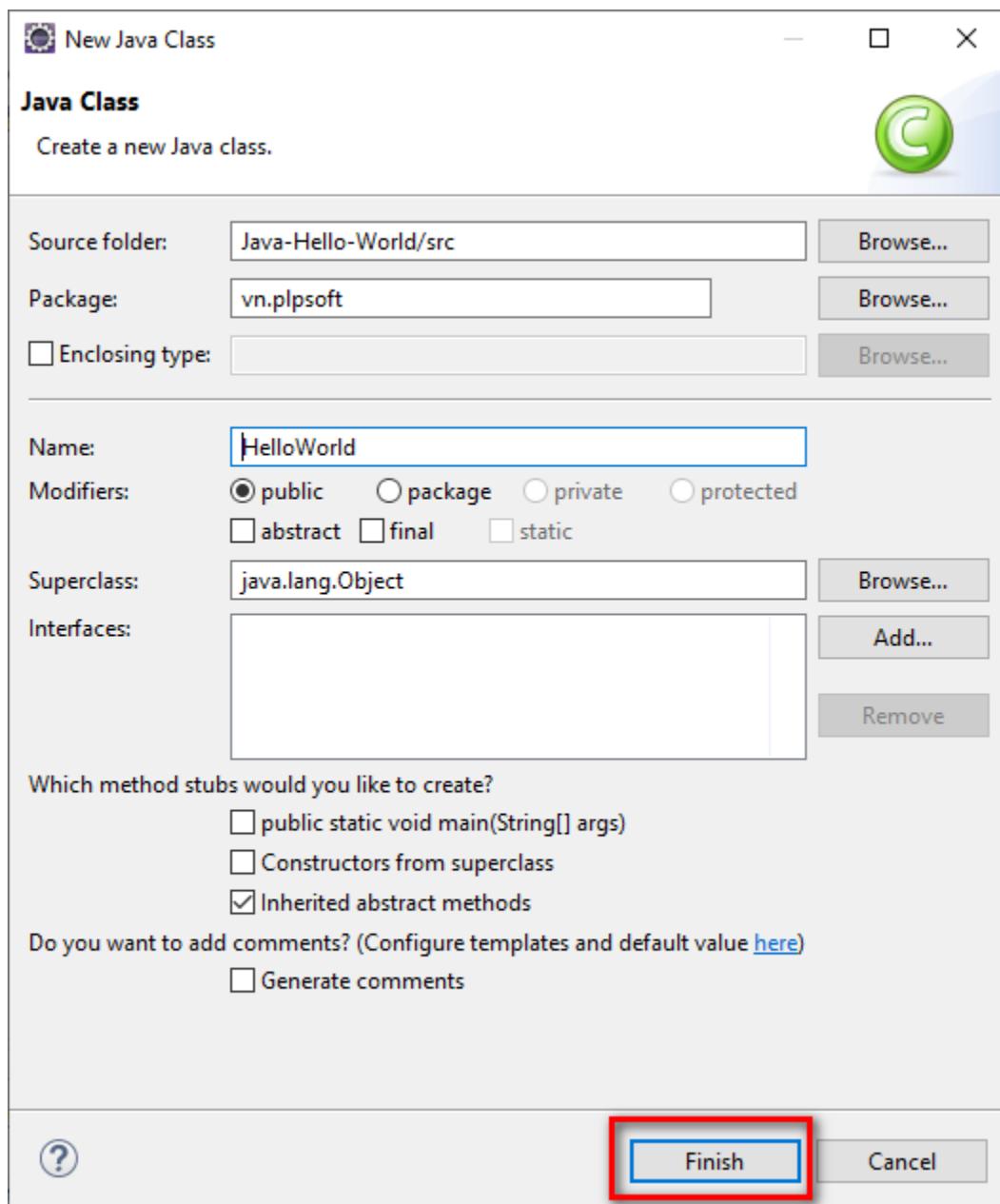
Nhập package name = "vn.plpsoft" -> Finish



Click chuột phải vào packpage "vn.plpsoft" -> New -> Class



Nhập Name = "HelloWorld" -> Finish



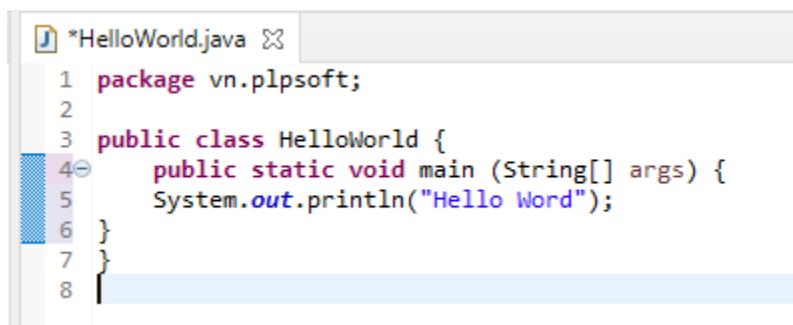
Eclipse tạo cho bạn lớp **HelloWorld.java** như sau:

```

1 package vn.plpsoft;
2
3 public class HelloWorld {
4
5 }
6

```

Viết hàm **main** cho lớp **HelloWorld.java** như sau:



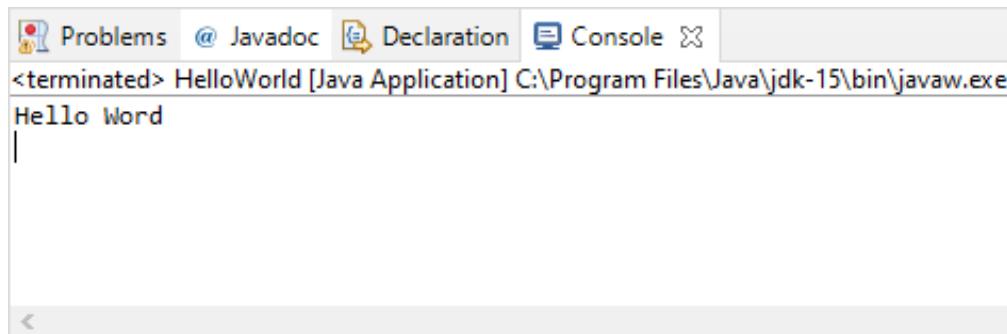
```

1 package vn.plpsoft;
2
3 public class HelloWorld {
4     public static void main (String[] args) {
5         System.out.println("Hello Word");
6     }
7 }
8

```

1.4.2.2 Chạy chương trình java đầu tiên trên eclipse

Sử dụng tổ hợp phím **CTRL + F11** để chạy (**run**) chương trình java trên eclipse, với chương trình đã tạo ở trên chúng ta có kết quả như sau:

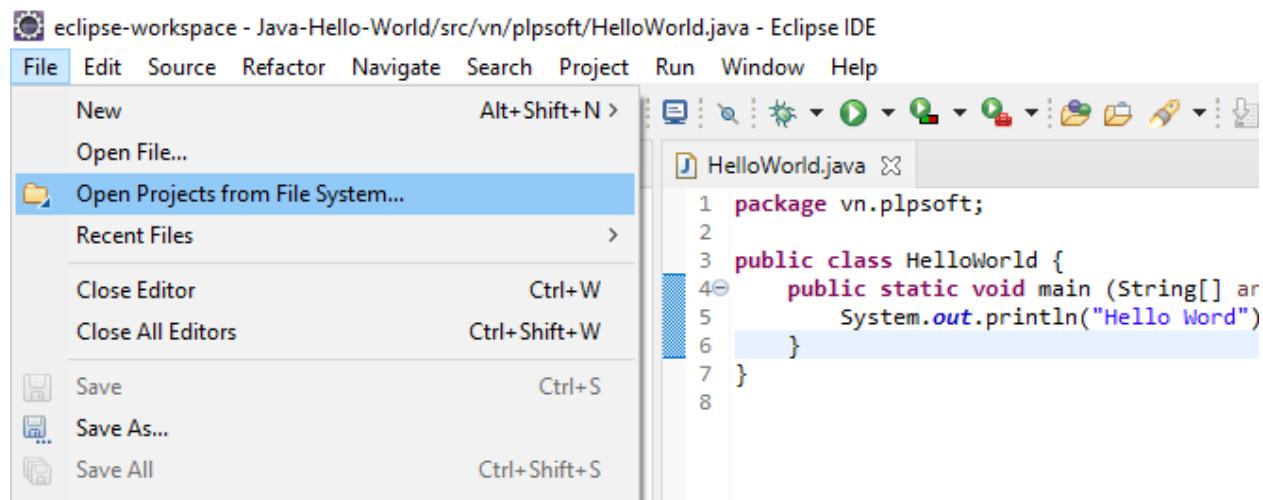


Console output:

```

<terminated> HelloWorld [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe
Hello Word
|
```

Xem thư mục chứa mã nguồn:



File menu options:

- New
- Open File...
- Open Projects from File System...**
- Recent Files
- Close Editor
- Close All Editors
- Save
- Save As...
- Save All

Keyboard shortcuts:

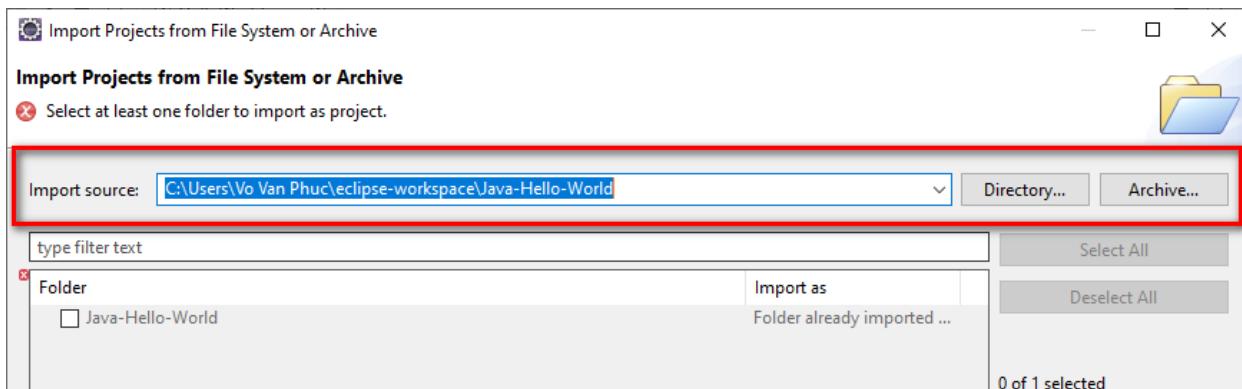
- Alt+Shift+N > for New
- Ctrl+W for Close Editor
- Ctrl+Shift+W for Close All Editors
- Ctrl+S for Save
- Ctrl+Shift+S for Save All

Code editor content:

```

1 package vn.plpsoft;
2
3 public class HelloWorld {
4     public static void main (String[] ar
5         System.out.println("Hello Word")
6     }
7 }
8

```



1.5 Biên dịch Hello World Trong Java bằng lệnh

Chương này, chúng ta sẽ học cách viết một chương trình đơn giản trong java. Để viết chương trình Hello World trong java, đầu tiên bạn nên cài 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 phải đáp ứng những yêu cầu sau:

- Cài đặt JDK nếu bạn chưa cài đặt nó. Bạn có thể tải JDK tại:
<https://www.oracle.com/java/technologies/downloads/>
- Thiết lập path cho java, tham khảo mục thiết lập path cho java.
- Tạo chương trình Java.
- Biên dịch và chạy chương trình Java.

1.5.1 Tạo ví dụ Hello World

Dưới đây là ví dụ Hello World trong java:

File: HelloWorld.java

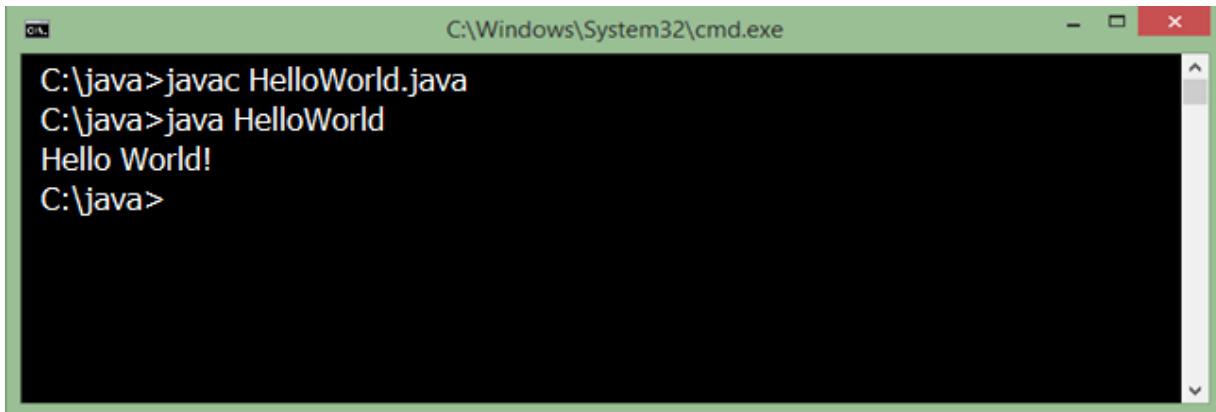
```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Save đoạn code trên trên thành file HelloWorld.java

Để biên dịch và chạy ví dụ trên bạn làm như sau:

1. **Run cmd.exe** bằng cách bấm tổ hợp phím Window + R --> gõ cmd --> enter
2. **Lệnh CD** đến thư mục chứa file HelloWorld.java
3. **Để biên dịch**, bạn gõ lệnh: javac HelloWorld.java
4. **Để run**, bạn gõ lệnh: java HelloWorld

Kết quả:



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\System32\cmd.exe'. The command line shows the user entering 'javac HelloWorld.java' followed by 'java HelloWorld'. The output displays the text 'Hello World!' which was printed by the Java code. The prompt then returns to 'C:\java>'.

Trên đây là một ví dụ về Hello World trong java sử dụng command line để biên dịch và run. Chúng tôi sử dụng command line để giới thiệu bạn cách biên dịch và run mã nguồn java. Ngoài ra, cũng như các ngôn ngữ lập trình khác, java có rất nhiều IDE hỗ trợ lập trình chẳng hạn như eclipse, netbean, intellij, ...

1.5.2 Phân tích chương trình java Hello World

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

- **class**: được sử dụng để khai báo một lớp trong Java.
- **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**: đại diện 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.

1.5.3 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:

1. Thay đổi thứ tự các modifier, không thay đổi thứ tự của tham số.

Ví dụ:

```
static public void main(String args[])
```

2. Thay kiểu khai báo tham số (mảng args)

Ví dụ:

```
public static void main(String[] args)  
public static void main(String []args)  
public static void main(String args[])
```

3. Sử dụng dấu ba chấm ... thay vì [] khi khai báo tham số (mảng args)

Ví dụ:

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

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)
```

1.6 Sự khác nhau giữa JDK, JRE và JVM

Hiểu rõ sự khác nhau giữa JDK, JRE và JVM là điều khá quan trọng trong Java.

1.6.1 Tìm hiểu JVM

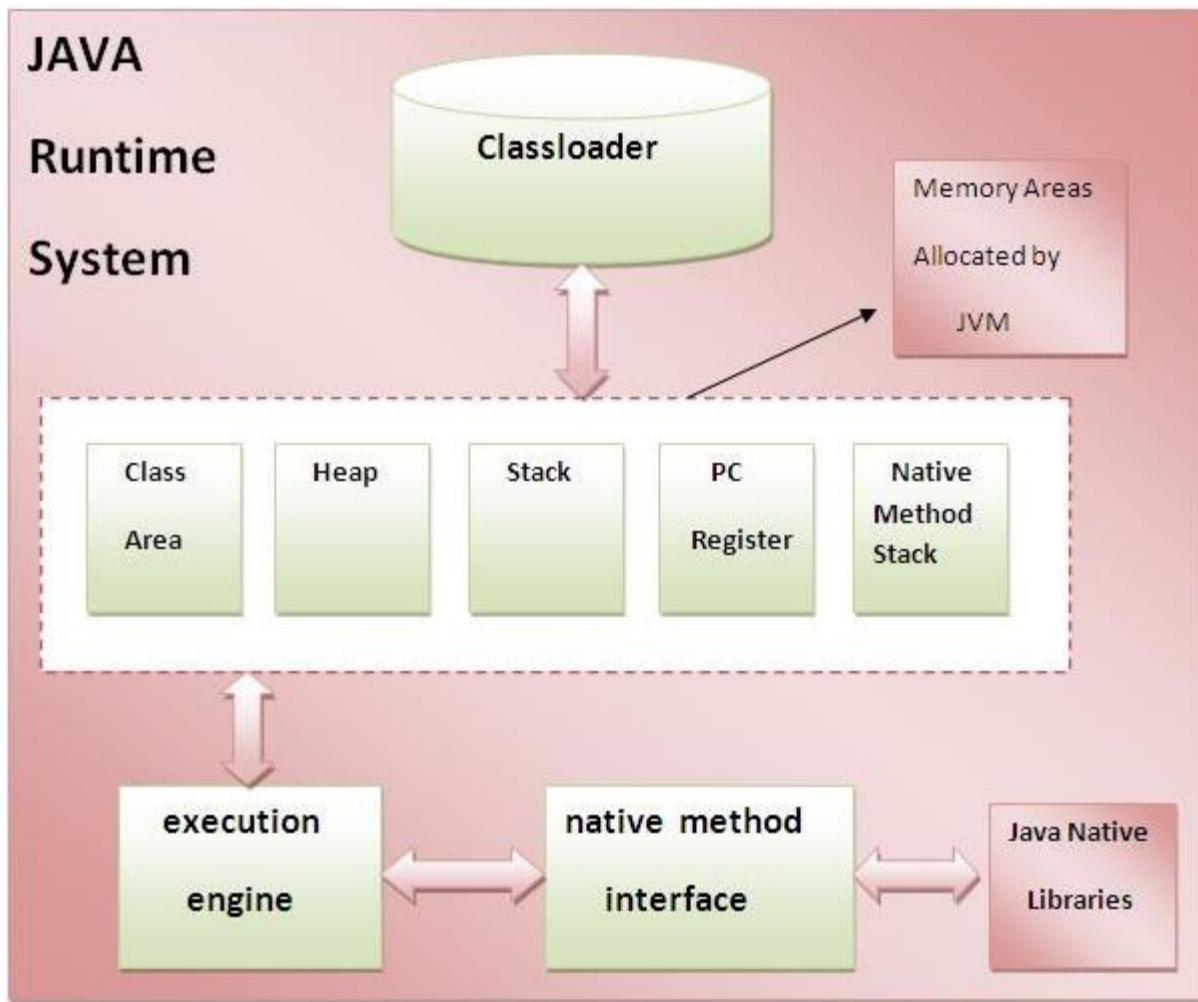
JVM (viết tắt của Java Virtual Machine) là một thiết bị trừu tượng (ảo) có thể giúp máy tính chạy các chương trình Java. Nó cung cấp môi trường runtime mà trong đó Java Bytecode có thể được thực thi.

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.

Các nhiệm vụ chính của JVM

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

Cấu trúc của JVM

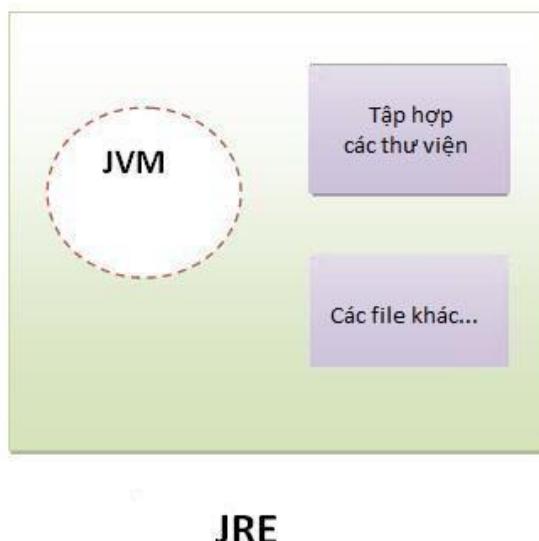


Trong đó:

- **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 nhiên được sử dụng trong ứng dụng.

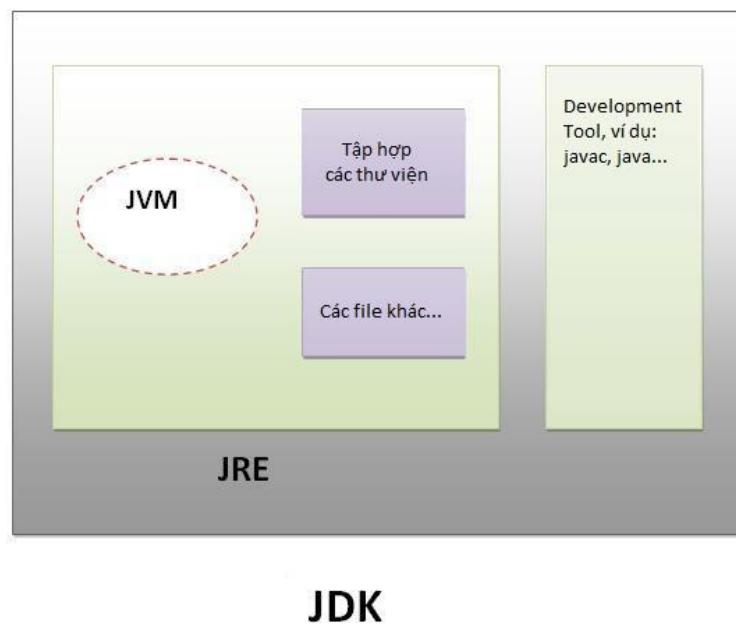
- **Execution Engine:** Phần này bao gồm: Một bộ xử lý ảo Virtual Processor Một trình thông dịch Interpreter. Đọc Bytecode Stream sau đó thực thi các chỉ thị.
- **Just-In-Time (JIT) Compiler:** được sử dụng để cải thiện hiệu suất. JIT biên dịch các phần của Bytecode mà có cùng tính năng tại cùng một 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.

1.6.2 Tìm hiểu 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.

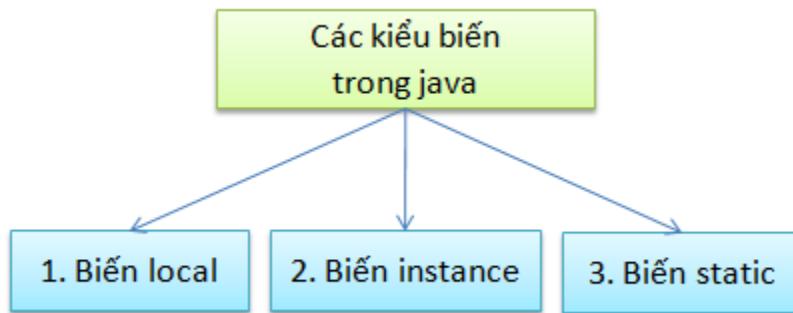
1.6.3 Tìm hiểu JDK



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

1.7 Biến trong java

Trong java, biến là tên của vùng nhớ. **Có 3 kiểu biến trong java**, bao gồm biến local (biến địa phương), biến instance (biến toàn cục) và biến static.



1.7.1 Khai báo biến trong java

Cú pháp khai báo biến:

```
DataType varName [ = value] [, varName2] [ = value2] ...;
```

Trong đó, DataType là kiểu dữ liệu của biến, varName là tên biến.

Quy tắc đặt tên biến trong java:

- Chỉ được bắt đầu bằng một ký tự(chữ), hoặc một dấu gạch dưới(_), hoặc một ký tự dollar(\$)
- Tên biến không được chứa khoảng trắng
- Bắt đầu từ ký tự thứ hai, có thể dùng ký tự(chữ), dấu gạch dưới(_), hoặc ký tự dollar(\$)
- Không được trùng với các từ khóa
- Có phân biệt chữ hoa và chữ thường

Ví dụ về khai báo biến trong java:

```
package vn.plpsoft.bienvadulieu;
public class Bien {
    public static float PI = 3.14f;      // Đây là biến static
    int n;                                // Đây là biến instance
    public Bien () {
        char c = 'c';                    // Đây là biến local
    }
}
```

1.7.2 Biến local trong java

- Biến local được khai báo trong các phương thức, hàm contructor hoặc trong các block.
- Biến local được tạo bên trong các phương thức, contructor, block và sẽ bị phá hủy khi kết thúc các phương thức, contructor và block.
- Không được sử dụng "access modifier" khi khai báo biến local.
- Các biến local được lưu trên vùng nhớ stack của bộ nhớ.
- Bạn cần khởi tạo giá trị mặc định cho biến local trước khi có thể sử dụng.

Ví dụ 1: Khởi tạo biến local:

```
package vn.plpsoft.bienvadulieu;
public class Bien {
    public void sayHello() {
        int n = 10;                  // Đây là biến local
        System.out.println("Gia tri cua n la: " + n);
    }
    public static void main(String[] args) {
        Bien bienLocal = new Bien();
        bienLocal.sayHello();
    }
}
```

Kết quả:

Gia trị của n là: 10

Ví dụ 2: Không khởi tạo biến local:

```
package vn.plpsoft.bienvadulieu;
public class Bien {
    public void sayHello() {
        int n; // Đây là biến local
        System.out.println("Gia tri cua n la: " + n);
    }
    public static void main(String[] args) {
        Bien bienLocal = new Bien();
        bienLocal.sayHello();
    }
}
```

Kết quả:

Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The local variable n may not have been initialized

Khi không khởi tạo biến local, chương trình java sẽ báo lỗi khi biên dịch.

1.7.3. Biến biến instance (biến toàn cục) trong java

- Biến instance được khai báo trong một lớp(class), bên ngoài các phương thức, constructor và các block.
- Biến instance được lưu trong bộ nhớ heap.
- Biến instance được tạo khi một đối tượng được tạo bằng việc sử dụng từ khóa “new” và sẽ bị phá hủy khi đối tượng bị phá hủy.
- Biến instance có thể được sử dụng bởi các phương thức, constructor, block, ... Nhưng nó phải được sử dụng thông qua một đối tượng cụ thể.
- Bạn được phép sử dụng "access modifier" khi khai báo biến instance, mặc định là "default".
- Biến instance có giá trị mặc định phụ thuộc vào kiểu dữ liệu của nó. Ví dụ nếu là kiểu int, short, byte thì giá trị mặc định là 0, kiểu double thì là 0.0d, ... Vì vậy, bạn sẽ không cần khởi tạo giá trị cho biến instance trước khi sử dụng.
- Bên trong class mà bạn khai báo biến instance, bạn có thể gọi nó trực tiếp bằng tên khi sử dụng ở khắp nơi bên trong class đó.

Ví dụ về biến instance trong java:

```
package vn.plpsoft.bienvadulieu;
public class Sinhvien {
    // biến instance "ten" kiểu String, có giá trị mặc định là null
    public String ten;

    // biến instance "tuoi" kiểu Integer, có giá trị mặc định là 0
    private int tuoi;

    // sử dụng biến ten trong một constructor
    public Sinhvien(String ten) {
        this.ten = ten;
    }

    // sử dụng biến tuoi trong phương thức setTuoi
    public void setTuoi(int tuoi) {
        this.tuoi = tuoi;
    }

    public void showStudent() {
        System.out.println("Ten : " + ten);
        System.out.println("Tuoi : " + tuoi);
    }
}

public static void main(String args[]) {
    Sinhvien sv = new Sinhvien("Nguyen Van A");
    sv.setTuoi(21);
    sv.showStudent();
}
```

Kết quả:

Ten : Nguyen Van A

Tuoi : 21

1.7.4. Biến static trong java

- Biến static được khai báo trong một class với từ khóa "static", phía bên ngoài các phương thức, constructor và block.
- Sẽ chỉ có duy nhất một bản sao của các biến static được tạo ra, dù bạn tạo bao nhiêu đối tượng từ lớp tương ứng.
- Biến static được lưu trữ trong bộ nhớ static riêng.
- Biến static được tạo khi chương trình bắt đầu chạy và chỉ bị phá hủy khi chương trình dừng.
- Giá trị mặc định của biến static phụ thuộc vào kiểu dữ liệu bạn khai báo tương tự biến instance.
- Biến static được truy cập thông qua tên của class chứa nó, với cú pháp: TenClass.tenBien.
- Trong class, các phương thức sử dụng biến static bằng cách gọi tên của nó khi phương thức đó cũng được khai báo với từ khóa "static".

Ví dụ về biến static trong java:

```
package vn.plpsoft.bienvadulieu;

public class Sinhvien {
    // biến static 'ten'
    public static String ten = "Nguyen Van A";

    // biến static 'tuoi'
    public static int tuoi = 21;

    public static void main(String args[]) {
        // Sử dụng biến static bằng cách gọi trực tiếp
        System.out.println("Ten : " + ten);

        // Sử dụng biến static bằng cách gọi thông qua tên class
        System.out.println("Ten : " + Sinhvien.tuoi);
    }
}
```

Kết quả:

Tên : Nguyen Van A

Ten : 21

1.8 Các kiểu dữ liệu trong java

Trong Java, các kiểu dữ liệu được chia thành hai loại:

- Các kiểu dữ liệu nguyên thủy
- Các kiểu dữ liệu đối tượng

1.8.1. Kiểu dữ liệu nguyên thủy

Java cung cấp các kiểu dữ liệu cơ bản như sau:

Kiểu dữ liệu	Mô tả
byte	Dùng để lưu dữ liệu kiểu số nguyên có kích thước một byte (8 bit). Phạm vi biểu diễn giá trị từ -128 đến 127. Giá trị mặc định là 0.
char	Dùng để lưu dữ liệu kiểu kí tự hoặc số nguyên không âm có kích thước 2 byte (16 bit). Phạm vi biểu diễn giá trị từ 0 đến u\ffff. Giá trị mặc định là 0.
boolean	Dùng để lưu dữ liệu chỉ có hai trạng thái đúng hoặc sai (độ lớn chỉ có 1 bit). Phạm vi biểu diễn giá trị là {"True", "False"}. Giá trị mặc định là False.
short	Dùng để lưu dữ liệu có kiểu số nguyên, kích cỡ 2 byte (16 bit). Phạm vi biểu diễn giá trị từ -32768 đến 32767. Giá trị mặc định là 0.
int	Dùng để lưu dữ liệu có kiểu số nguyên, kích cỡ 4 byte (32 bit). Phạm vi biểu diễn giá trị từ -2,147,483,648 đến 2,147,483,647. Giá trị mặc định là 0.
long	Dùng để lưu dữ liệu có kiểu số nguyên có kích thước lên đến 8 byte. Giá trị mặc định là 0L.
float	Dùng để lưu dữ liệu có kiểu số thực, kích cỡ 4 byte (32 bit). Giá trị mặc định là 0.0F.
double	Dùng để lưu dữ liệu có kiểu số thực có kích thước lên đến 8 byte. Giá trị mặc định là 0.00D

1.8.2. Kiểu dữ liệu đối tượng

Trong java có 3 kiểu dữ liệu đối tượng:

Kiểu dữ liệu	Mô tả
Array	Một mảng của các dữ liệu cùng kiểu.
class	Dữ liệu kiểu lớp đối tượng do người dùng định nghĩa. Chứa tập các thuộc tính và phương thức..
interface	Dữ liệu kiểu lớp giao tiếp do người dùng định nghĩa. Chứa các phương thức của giao tiếp.

1.9. Ép kiểu trong Java

Ép kiểu trong java là việc gán giá trị của một biến có kiểu dữ liệu này tới biến khác có kiểu dữ liệu khác.

Ví dụ:

```
float c = 35.8f;
int b = (int)c + 1;
```

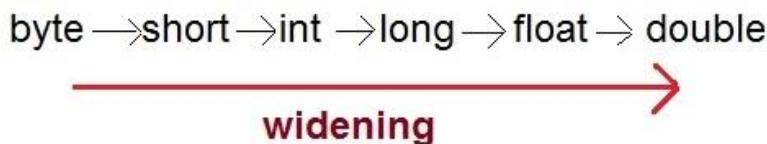
Trong ví dụ trên, đầu tiên giá trị dấu phẩy động c được đổi thành giá trị nguyên 35. Sau đó nó được cộng với 1 và kết quả là giá trị 36 được lưu vào b.

1.9.1 Phân loại ép kiểu trong java

Trong Java, có hai loại ép kiểu dữ liệu:

- Nới rộng (widening):** Là quá trình làm tròn số từ kiểu dữ liệu có kích thước nhỏ hơn sang kiểu có kích thước lớn hơn. Kiểu biến đổi này không làm mất thông tin.
- Thu hẹp (narrowing):** Là quá trình làm tròn số từ kiểu dữ liệu có kích thước lớn hơn sang kiểu có kích thước nhỏ hơn. Kiểu biến đổi này có thể làm mất thông tin

1.9.2. Nới rộng (widening)



Nới rộng (widening): Là quá trình làm tròn số từ kiểu dữ liệu có kích thước nhỏ hơn sang kiểu có kích thước lớn hơn. Kiểu biến đổi này không làm mất thông tin. Ví dụ

chuyển từ int sang float. Chuyển kiểu loại này có thể được thực hiện ngầm định bởi trình biên dịch.

Ví dụ:

```
public class TestWidening {
    public static void main(String[] args) {
        int i = 100;
        long l = i;      // không yêu cầu chỉ định ép kiểu
        float f = l;    // không yêu cầu chỉ định ép kiểu
        System.out.println("Giá trị Int: " + i);
        System.out.println("Giá trị Long: " + l);
        System.out.println("Giá trị Float: " + f);
    }
}
```

Kết quả:

```
Giá trị Int: 100
Giá trị Long: 100
Giá trị Float: 100.0
```

1.9.3. Thu hẹp (narrowing)



Thu hẹp (narrowing): Là quá trình làm tròn số từ kiểu dữ liệu có kích thước lớn hơn sang kiểu có kích thước nhỏ hơn. Kiểu biến đổi này có thể làm mất thông tin như ví dụ ở trên. Chuyển kiểu loại này không thể thực hiện ngầm định bởi trình biên dịch, người dùng phải thực hiện chuyển kiểu tường minh.

Ví dụ:

```
public class TestNarrowwing {
    public static void main(String[] args) {
        double d = 100.04;
        long l = (long) d; // yêu cầu chỉ định kiểu dữ liệu (long)
```

```

        int i = (int) l; // yêu cầu chỉ định kiểu dữ liệu (int)

        System.out.println("Giá trị Double: " + d);
        System.out.println("Giá trị Long: " + l);
        System.out.println("Giá trị Int: " + i);

    }
}

```

Kết quả:

```

Giá trị Double: 100.04
Giá trị Long: 100
Giá trị Int: 100

```

1.10. Toán tử trong java

Toán tử trong java là một ký hiệu được sử dụng để thực hiện một phép tính/chức năng nào đó. Java cung cấp các dạng toán tử sau:

- Toán tử số học
- Toán tử bit
- Toán tử quan hệ
- Toán tử logic
- Toán tử điều kiện
- Toán tử gán

1.10.1. Toán tử số học

Các toán hạng của các toán tử số học phải ở dạng số. Các toán hạng kiểu boolean không sử dụng được, các toán hạng ký tự cho phép sử dụng loại toán tử này. Một vài kiểu toán tử được liệt kê trong bảng dưới đây.

Giả sử chúng ta có biến số nguyên $a = 10$ và $b = 20$.

Toán tử	Mô tả	Ví dụ
+	Cộng Trả về giá trị là tổng của hai toán hạng	$a + b$ sẽ là 30

-	Trừ Trả về kết quả là hiệu của hai toán hạng.	a + b sẽ là -1
*	Nhân Trả về giá trị là tích của hai toán hạng.	a + b sẽ là 200
/	Chia Trả về giá trị là thương của phép chia.	b / a sẽ là 2
%	Phép lấy modul Giá trị trả về là phần dư của phép chia	b % a sẽ là 0
++	Tăng dần Tăng giá trị của biến lên 1. Ví dụ a++ tương đương với a = a + 1	a++ sẽ là 11
--	Giảm dần Giảm giá trị của biến 1 đơn vị. Ví dụ a-- tương đương với a = a - 1	a-- sẽ là 9
+ =	Cộng và gán giá trị Cộng các giá trị của toán hạng bên trái vào toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ c += a tương đương c = c + a	a += 2 sẽ là 12
- =	Trừ và gán giá trị Trừ các giá trị của toán hạng bên trái vào toán toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ c -= a tương đương c = c - a	a -= 2 sẽ là 8
* =	Nhân và gán Nhân các giá trị của toán hạng bên trái với toán toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ c *= a tương đương c = c*a	a *= 2 sẽ là 20
/ =	Chia và gán Chia giá trị của toán hạng bên trái cho toán toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ c /= a tương đương với c = c/a	a /= 2 sẽ là 5
% =	Lấy số dư và gán Chia giá trị của toán hạng bên trái cho toán toán hạng bên phải và gán giá trị số dư vào toán hạng bên trái. Ví dụ c %= a tương đương với c = c%a	a /= 8 sẽ là 2

1.10.2. Toán tử Bit

Các toán tử dạng bit cho phép chúng ta thao tác trên từng bit riêng biệt trong các kiểu dữ liệu nguyên thuỷ.

Toán tử	Mô tả
\sim	Phủ định NOT Trả về giá trị phủ định của một bít.
$\&$	Toán tử AND Trả về giá trị là 1 nếu các toán hạng là 1 và 0 trong các trường hợp khác
$ $	Toán tử OR Trả về giá trị là 1 nếu một trong các toán hạng là 1 và 0 trong các trường hợp khác.
\wedge	Toán tử Exclusive OR Trả về giá trị là 1 nếu chỉ một trong các toán hạng là 1 và trả về 0 trong các trường hợp khác.
$>>$	Dịch phải Chuyển toàn bộ các bít của một số sang phải một vị trí, giữ nguyên dấu của số âm. Toán hạng bên trái là số bị dịch còn số bên phải chỉ số vị trí mà các bít cần dịch.
$<<$	Dịch trái Chuyển toàn bộ các bít của một số sang trái một vị trí, giữ nguyên dấu của số âm. Toán hạng bên trái là số bị dịch còn số bên phải chỉ số vị trí mà các bít cần dịch.

1.10.3. Các toán tử quan hệ

Các toán tử quan hệ được sử dụng kiểm tra mối quan hệ giữa hai toán hạng. Kết quả của một biểu thức có dùng các toán tử quan hệ là những giá trị Boolean (logic “true” hoặc “false”). Các toán tử quan hệ được sử dụng trong các cấu trúc điều khiển.

Toán tử	Mô tả
$==$	So sánh bằng Toán tử này kiểm tra sự tương đương của hai toán hạng
$!=$	So sánh khác Toán tử này kiểm tra sự khác nhau của hai toán hạng

>	Lớn hơn Kiểm tra giá trị của toán hạng bên phải lớn hơn toán hạng bên trái hay không
<	Nhỏ hơn Kiểm tra giá trị của toán hạng bên phải có nhỏ hơn toán hạng bên trái hay không
>=	Lớn hơn hoặc bằng Kiểm tra giá trị của toán hạng bên phải có lớn hơn hoặc bằng toán hạng bên trái hay không
<=	Nhỏ hơn hoặc bằng Kiểm tra giá trị của toán hạng bên phải có nhỏ hơn hoặc bằng toán hạng bên trái hay không

1.10.4. Các toán tử logic

Các toán tử logic làm việc với các toán hạng Boolean. Các toán tử quan hệ được sử dụng trong các cấu trúc điều khiển.

Toán tử	Mô tả
&&	Toán tử và (AND) Trả về một giá trị “Đúng” (True) nếu chỉ khi cả hai toán tử có giá trị “True”
 	Toán tử hoặc (OR) Trả về giá trị “True” nếu ít nhất một giá trị là True
^	Toán tử XOR Trả về giá trị True nếu và chỉ nếu chỉ một trong các giá trị là True, các trường hợp còn lại cho giá trị False (sai)
!	Toán tử phủ định (NOT) Toán hạng đơn từ NOT. Chuyển giá trị từ True sang False và ngược lại.

1.10.5. Các toán tử điều kiện

Toán tử điều kiện là một loại toán tử đặc biệt vì nó bao gồm ba thành phần cấu thành biểu thức điều kiện. Cú pháp:

<biểu thức 1> ? <biểu thức 2> : <biểu thức 3>;

- **biểu thức 1:** Biểu thức logic. Trả trả về giá trị True hoặc False
- **biểu thức 2:** Là giá trị trả về nếu xác định là True
- **biểu thức 3:** Là giá trị trả về nếu xác định là False

Ví dụ:

```
public class Test {
    public static void main(String[] args) {
        int a = 20;
        int b = 3;
        String s = (a % b == 0) ? "a chia het cho b" : "a
                                  khong chia het cho b";
        System.out.println(s);
    }
}
```

Kết quả:

```
a khong chia het cho b
```

1.10.6. Toán tử gán

Toán tử gán (=) dùng để gán một giá trị vào một biến và có thể gán nhiều giá trị cho nhiều biến cùng một lúc.

Ví dụ:

```
int var = 20;
int p,q,r,s;
p=q=r=s=var;
```

Trong ví dụ trên, đoạn lệnh sau gán một giá trị cho biến var và giá trị này lại được gán cho nhiều biến trên một dòng lệnh đơn.

Dòng lệnh cuối cùng được thực hiện từ phải qua trái. Đầu tiên giá trị ở biến var được gán cho 's', sau đó giá trị của 's' được gán cho 'r' và cứ tiếp như vậy.

1.10.7. Thứ tự ưu tiên của các toán tử

Thứ tự ưu tiên quyết định trật tự thực hiện các toán tử trên các biểu thức. Bảng dưới đây liệt kê thứ tự thực hiện các toán tử trong Java

Toán tử	Mô tả
1	Các toán tử đơn như +,-,++,--
2	Các toán tử số học và các toán tử dịch như *,/,+,-,<<,>>
3	Các toán tử quan hệ như >,<,>=,<=,=,!=
4	Các toán tử logic và Bit như &&, ,&, ,^
5	Các toán tử gán như =,*=,/=,+=,-=

1.10.8. Thay đổi thứ tự ưu tiên của các toán tử

Để thay đổi thứ tự ưu tiên trên một biểu thức, bạn có thể sử dụng dấu ngoặc đơn ():

- Phần được giới hạn trong ngoặc đơn được thực hiện trước.
- Nếu dùng nhiều ngoặc đơn lồng nhau thì toán tử nằm trong ngoặc đơn phía trong sẽ thực thi trước, sau đó đến các vòng phia ngoài.
- Trong phạm vi một cặp ngoặc đơn thì quy tắc thứ tự ưu tiên vẫn giữ nguyên tác dụng.

Ví dụ:

```
public class Test {
    public static void main(String[] args) {
        int a = 20;
        int b = 5;
        int c = 10;
        System.out.println("a + b * c      = " + (a + b * c));
        System.out.println("(a + b) * c = " + ((a + b) * c));
        System.out.println("a / b - c      = " + (a / b - c));
        System.out.println("a / (b - c) = " + (a / (b - c)));
    }
}
```

Kết quả:

```
a + b * c      = 70
(a + b) * c = 250
```

a / b - c = -6

a / (b - c) = -4

1.11. Hệ thống Unicode trong java

1.11.1 Hệ thống Unicode

Unicode là một kiểu mã hóa ký tự chuẩn quốc tế. Unicode được sử dụng trong hầu hết các ngôn ngữ văn bản điện tử của thế giới.

1.11.2. Tại sao Java sử dụng Hệ thống Unicode

Trước khi có Unicode, đã có rất nhiều tiêu chuẩn khác:

- **ASCII** (American Standard Code for Information Interchange) được sử dụng ở Hoa Kỳ
- **ISO 8859-1** được sử dụng ở Tây Âu
- **KOI-8** được sử dụng ở Nga
- **GB18030 and BIG-5**

Và như vậy, có 2 vấn đề xảy ra!

- Thứ nhất, Một giá trị mã cụ thể tương ứng với ký tự khác nhau trong các tiêu chuẩn ngôn ngữ khác nhau.
- Thứ hai, Các kiểu mã hóa cho các ngôn ngữ với tập các ký tự lớn có biến length. Nhiều ký tự thông dụng được mã hóa thành 1 byte, những ký tự khác yêu cầu 2 hoặc nhiều byte.

Để giải quyết vấn đề này, có một vài chuẩn mã hóa ký tự được ra đời, trong đó có Unicode.

Trong Unicode, mỗi ký tự chiếm 2 byte, Vì thế java cũng sử dụng 2 byte cho mỗi ký tự.

Giá trị nhỏ nhất: \u0000

Giá trị lớn nhất: \uffff

1.12. Các lệnh điều khiển

1.12.1 Mệnh đề if-else trong java

Mệnh đề if trong java được sử dụng để kiểm tra giá trị dạng boolean của điều kiện. Mệnh đề này trả về giá trị **True** hoặc **False**. Có các kiểu của mệnh đề if-else trong java như sau:

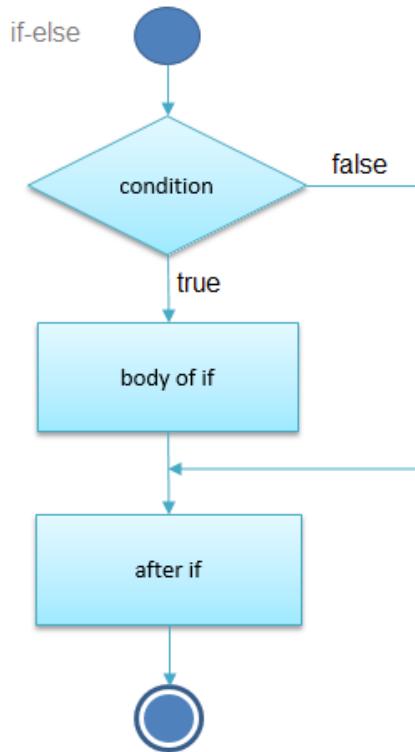
- Mệnh đề if
 - Mệnh đề if-else
 - Mệnh đề if-else-if
-

1.12.1.1 Mệnh đề if

Mệnh đề if được sử dụng để kiểm tra giá trị dạng boolean của điều kiện. Khối lệnh sau if được thực thi nếu giá trị của điều kiện là **True**

Cú pháp:

```
if (condition) {  
    // khối lệnh này thực thi  
    // nếu condition = true  
}
```



Ví dụ:

```

public class Test {
    public static void main(String[] args) {
        int age = 20;
        if (age > 18) {
            System.out.print("Tuổi lớn hơn 18");
        }
    }
}
  
```

Kết quả:

Tuổi lớn hơn 18

1.12.1.2 Mệnh đề if-else

Mệnh đề if-else cũng kiểm tra giá trị dạng boolean của điều kiện. Nếu giá trị điều kiện là **True** thì chỉ có khối lệnh sau if sẽ được thực hiện, nếu là **False** thì chỉ có khối lệnh sau else được thực hiện.

Cú pháp:

```

if (condition) {
    // khối lệnh này được thực thi
    // nếu condition = true
} else {
    // khối lệnh này được thực thi
    // nếu condition = false
}

```

Ví dụ:

```

public class Test {
    public static void main(String[] args) {
        int number = 13;
        if (number % 2 == 0) {
            System.out.println("Số " + number + " là số chẵn.");
        } else {
            System.out.println("Số " + number + " là số lẻ.");
        }
    }
}

```

Kết quả:

Số 13 là số lẻ.

1.12.1.3 Mệnh đề if-else-if

Mệnh đề if-else-if cũng kiểm tra giá trị dạng boolean của điều kiện. Nếu giá trị điều kiện if là **True** thì chỉ có khối lệnh sau if sẽ được thực hiện. Nếu giá trị điều kiện if else nào là **True** thì chỉ có khối lệnh sau else if đó sẽ được thực hiện... Nếu tất cả điều kiện của if và else if là **False** thì chỉ có khối lệnh sau else sẽ được thực hiện.

Cú pháp:

```

if (condition1) {
    // khối lệnh này được thực thi
    // nếu condition1 là true
} else if (condition2) {
    // khối lệnh này được thực thi
}

```

```
// nếu condition2 là true
} else if (condition3) {
    // khôi lệnh này được thực thi
    // nếu condition3 là true
}
...
else {
    // khôi lệnh này được thực thi
    // nếu tất cả những điều kiện trên là false
}
```

Ví dụ:

```
public class Test {
    public static void main(String[] args) {
        int marks = 65;

        if (marks < 50) {
            System.out.println("Tạch!");
        } else if (marks >= 50 && marks < 60) {
            System.out.println("Xếp loại D");
        } else if (marks >= 60 && marks < 70) {
            System.out.println("Xếp loại C");
        } else if (marks >= 70 && marks < 80) {
            System.out.println("Xếp loại B");
        } else if (marks >= 80 && marks < 90) {
            System.out.println("Xếp loại A");
        } else if (marks >= 90 && marks < 100) {
            System.out.println("Xếp loại A+");
        } else {
            System.out.println("Giá trị không hợp lệ!");
        }
    }
}
```

Kết quả:

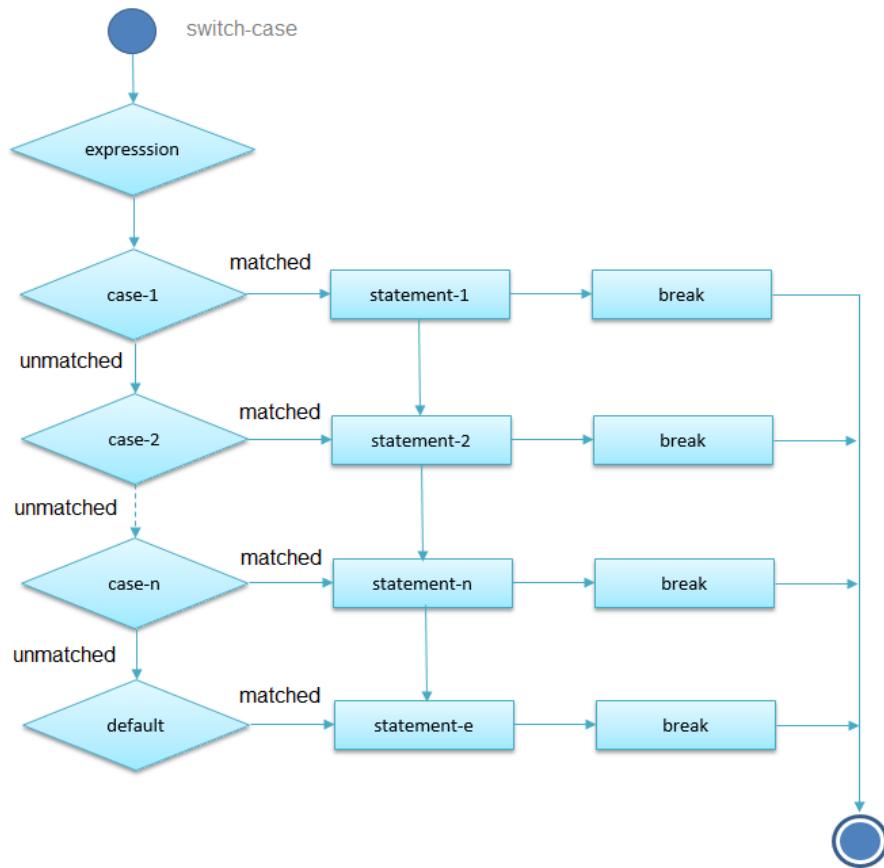
Xếp loại C

1.12.2 Mệnh đề Switch-case trong java

Mệnh đề switch-case trong java được sử dụng để thực thi 1 hoặc nhiều khối lệnh từ nhiều điều kiện.

Cú pháp:

```
switch (bieu_thuc) {  
    case gia_tri_1:  
        // Khối lệnh 1  
        break; //tùy chọn  
    case gia_tri_2:  
        // Khối lệnh 2  
        break; //tùy chọn  
    .....  
    case gia_tri_n:  
        // Khối lệnh n  
        break; //tùy chọn  
    default:  
        // Khối lệnh này được thực thi  
        // nếu tất cả các điều kiện trên không thỏa mãn  
}
```



Ví dụ về mệnh đề switch-case:

```

public class SwitchExample {
    public static void main(String[] args) {
        int number = 20;
        switch (number) {
            case 10:
                System.out.println("10");
                break;
            case 20:
                System.out.println("20");
                break;
            case 30:
                System.out.println("30");
                break;
            default:
                System.out.println("Not in 10, 20 or 30");
        }
    }
}
  
```

```

        }
    }
}

```

Kết quả: 20

➤ Mệnh đề Switch-case khi không sử dụng 'break'

Khi không sử dụng từ khóa 'break' trong mệnh đề switch-case. Điều này có nghĩa là các khối lệnh sau case có giá trị phù hợp sẽ được thực thi.

Ví dụ về mệnh đề switch-case:

```

public class SwitchExample2 {
    public static void main(String[] args) {
        int number = 20;
        switch (number) {
            case 10:
                System.out.println("10");
            case 20:
                System.out.println("20");
            case 30:
                System.out.println("30");
            default:
                System.out.println("Not in 10, 20 or 30");
        }
    }
}

```

Kết quả:

```

20
30
Not in 10, 20 or 30

```

1.12.3 Vòng lặp for trong java

Vòng lặp for trong java được sử dụng để lặp một phần của chương trình nhiều lần. Nếu số lần lặp là cố định thì vòng lặp for được khuyến khích sử dụng, còn nếu số lần lặp không cố định thì nên sử dụng vòng lặp while hoặc do while.

Có 3 kiểu của vòng lặp for trong java:

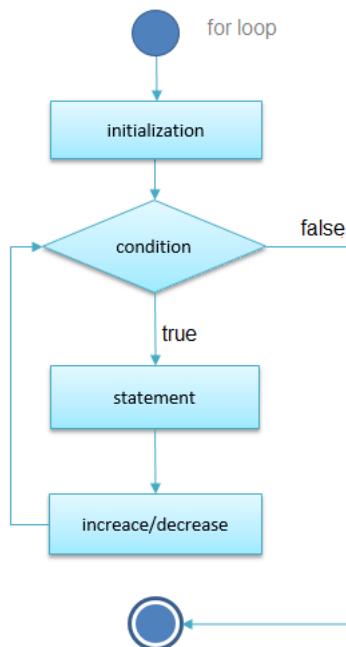
- Vòng lặp for đơn giản
- Vòng lặp for cải tiến
- Vòng lặp for gán nhãn

1.12.3.1 Vòng lặp for đơn giản

Vòng lặp for đơn giản giống như trong C/C++. Chúng ta có thể khởi tạo biến, kiểm tra điều kiện và tăng/giảm giá trị của biến.

Cú pháp:

```
for (khởi_tạo_bien ; check_dieu_kien ; tang/giam_bien) {
    // Khởi lệnh được thực thi
}
```



Ví dụ:

```
public class ForExample {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            System.out.println(i);
        }
    }
}
```

```
}
```

Kết quả:

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

1.12.3.2 Vòng lặp for cải tiến

Vòng lặp for cải tiến được sử dụng để lặp mảng(array) hoặc collection trong java. Bạn có thể sử dụng nó dễ dàng, dễ hơn cả vòng lặp for đơn giản. Bởi vì bạn không cần phải tăng hay giảm giá trị của biến rồi check điều kiện, bạn chỉ cần sử dụng ký hiệu hai chấm ":".

Cú pháp:

```
for (Type var : array) {  
    // Khởi lệnh được thực thi  
}
```

Ví dụ

```
public class ForEachExample {  
    public static void main(String[] args) {  
        int arr[] = { 12, 23, 44, 56, 78 };  
        for (int i : arr) {  
            System.out.println(i);  
        }  
    }  
}
```

Kết quả:

```
12
23
44
56
78
```

1.12.3.3 Vòng lặp for gán nhãn

Chúng ta có thể đặt tên cho mỗi vòng lặp for bằng cách gán nhãn trước vòng lặp for. Điều này rất hữu dụng khi chúng ta muốn thoát/tiếp tục(break/continues) chạy vòng lặp for.

Cú pháp:

```
ten_nhan:
for (khởi_tạo_bien ; check_dieu_kien ; tang/giam_bien) {
    // Khởi lệnh được thực thi
}
```

Ví dụ

```
public class LabeledForExample {
    public static void main(String[] args) {
        aa: for (int i = 1; i <= 3; i++) {
            bb: for (int j = 1; j <= 3; j++) {
                if (i == 2 && j == 2) {
                    break aa;
                }
                System.out.println(i + " " + j);
            }
        }
    }
}
```

Kết quả:

```
1 1
1 2
1 3
```

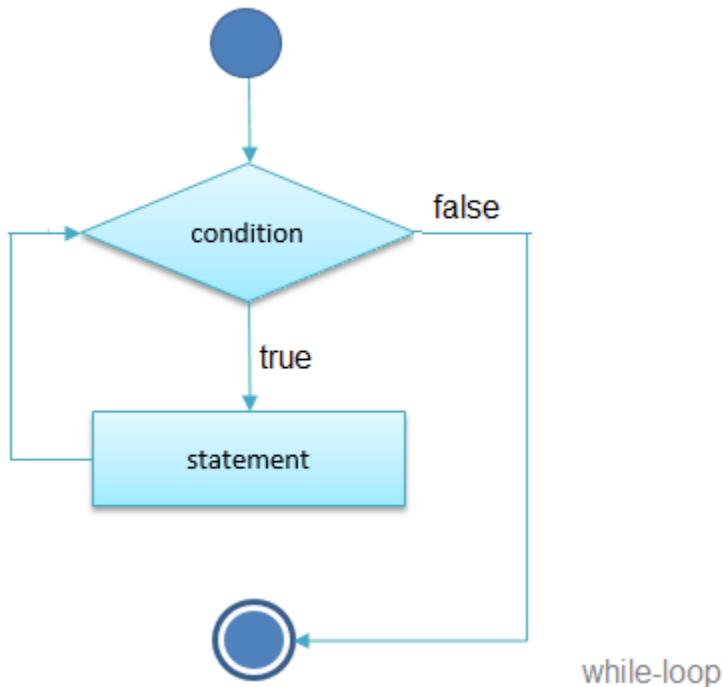
1.12.4 Vòng lặp while trong java

1.12.4.1 Vòng lặp while trong java

Vòng lặp while trong java được sử dụng để lặp một phần của chương trình một vài lần. Nếu số lần lặp không được xác định trước thì vòng lặp while được khuyến khích sử dụng trong trường hợp này.

Cú pháp:

```
while(condition) {
    // Khối lệnh được lặp lại cho đến khi condition = False
}
```



Ví dụ về vòng lặp while trong java:

```
public class WhileExample1 {
    public static void main(String[] args) {
        int i = 1;
        while (i <= 10) {
            System.out.println(i);
            i++;
        }
    }
}
```

```
    }  
}  
}
```

Kết quả:

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

1.12.4.2 Vòng lặp while vô tận

Nếu bạn để điều kiện lặp là True thì vòng lặp while sẽ chạy đến vô tận... Đến khi bạn stop chương trình đối với mỗi IDE(Eclipse, Netbean...) hoặc bấm Ctrl + C khi chạy bằng command.

Ví dụ về vòng lặp while vô tận trong java:

```
public class WhileExample2 {  
    public static void main(String[] args) {  
        while (true) {  
            System.out.println("Vòng lặp while vô tận...");  
        }  
    }  
}
```

Kết quả:

```
Vòng lặp while vô tận...  
Vòng lặp while vô tận...  
Vòng lặp while vô tận...  
Vòng lặp while vô tận...
```

Vòng lặp while vô tận...

Ctrl + C

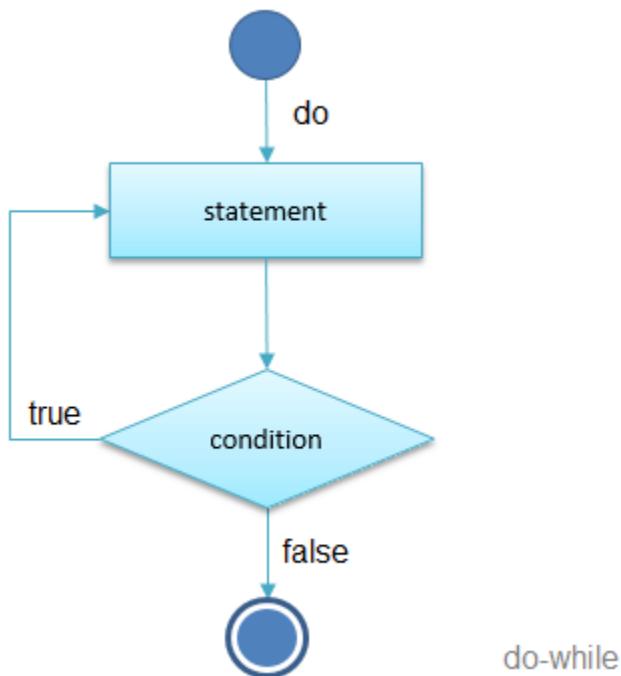
1.12.5 Vòng lặp do-while trong java

1.12.5.1 Vòng lặp do-while trong java

Vòng lặp do-while trong java được sử dụng để lặp một phần của chương trình một vài lần. Tương tự như vòng lặp while, ngoại trừ do-while thực hiện lệnh ít nhất một lần ngay cả khi điều kiện là False.

Cú pháp:

```
do {
    // Khởi lệnh được thực thi
} while(condition);
```



Ví dụ sau tính tổng của 5 số tự nhiên đầu tiên dùng cấu trúc do-while:

```
public class DoWhileExample1 {
    public static void main(String[] args) {
        int a = 1, sum = 0;
        do {
            sum += a;
        }
    }
}
```

```

        a++;
    } while (a <= 5);
    System.out.println("Sum of 1 to 5 is " + sum);
}
}

```

Biến a được khởi tạo với giá trị 1, sau đó nó vừa được dùng làm biến chạy (tăng lên 1 sau mỗi lần lặp) vừa được dùng để cộng dồn vào biến sum. Tại thời điểm kết thúc, chương trình sẽ in ra Sum of 1 to 5 is 15.

Kết quả:

```
Sum of 1 to 5 is 15
```

1.12.5.2 Vòng lặp do-while vô tận

Nếu bạn để điều kiện lặp là True thì vòng lặp do-while sẽ chạy đến vô tận... Đến khi bạn stop chương trình đối với mỗi IDE(Eclipse, Netbean...) hoặc bấm Ctrl + C khi chạy bằng command.

Ví dụ về vòng lặp do-while vô tận:

```

public class DoWhileExample2 {
    public static void main(String[] args) {
        do {
            System.out.println("Vòng lặp do-while vô tận...");
        } while (true);
    }
}

```

Kết quả:

```
Vòng lặp do-while vô tận...
Ctrl + C
```

1.12.6 Sử dụng Break trong java

1.12.6.1 Sử dụng Break trong java

Từ khóa **break** trong java được sử dụng để stop thực thi lệnh trong vòng lặp hoặc trong mệnh đề switch tại điều kiện đã được chỉ định. Đối với vòng lặp bên trong vòng lặp khác, thì nó chỉ stop vòng lặp bên trong đó.

1.12.6.2 Sử dụng Break với vòng lặp for

Ví dụ sử dụng break với vòng lặp for:

```
public class BreakExample {  
    public static void main(String[] args) {  
        for (int i = 1; i <= 10; i++) {  
            if (i == 5) {  
                break;  
            }  
            System.out.println(i);  
        }  
    }  
}
```

Kết quả:

```
1  
2  
3  
4
```

Ví dụ sử dụng break với vòng lặp bên trong vòng lặp for khác:

```
public class BreakExample2 {  
    public static void main(String[] args) {  
        for (int i = 1; i <= 3; i++) {  
            for (int j = 1; j <= 3; j++) {  
                //...  
            }  
        }  
    }  
}
```

```

        if (i == 2 && j == 2) {
            break;
        }
        System.out.println(i + " " + j);
    }
}
}
}

```

Kết quả:

```

1 1
1 2
1 3
2 1
3 1
3 2
3 3

```

1.12.6.3 Sử dụng Break với mệnh đề switch-case

Để hiểu cách sử dụng break với mệnh đề switch-case, bạn có thể tham khảo bài học này [Mệnh đề switch-case trong java](#)

1.12.7 Sử dụng Continue trong java

Tùy khóa continue trong java được sử dụng để tiếp tục vòng lặp tại điều kiện đã được xác định, với điều kiện đó khối lệnh phía sau từ khóa continue sẽ không được thực thi. Đối với vòng lặp bên trong một vòng lặp khác, continue chỉ có tác dụng với vòng lặp bên trong đó.

Ví dụ sử dụng Continue trong java với vòng lặp for:

```

public class ContinueExample {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {

```

```

        if (i == 5) {
            continue;
        }
        // Khi i == 5 thì không in i = 5 ra màn hình
        System.out.println(i);
    }
}

```

Kết quả:

```

1
2
3
4
5
6
7
8
9
10

```

Ví dụ sử dụng Continue với vòng lặp bên trong vòng lặp for khác:

```

public class ContinueExample2 {
    public static void main(String[] args) {
        for (int i = 1; i <= 3; i++) {
            for (int j = 1; j <= 3; j++) {
                if (i == 2 && j == 2) {
                    continue;
                }
                // Không in trường hợp i=2 và j=2 ra màn hình
                System.out.println(i + " " + j);
            }
        }
    }
}

```

}

Kết quả:

```
1 1  
1 2  
1 3  
2 1  
2 3  
3 1  
3 2  
3 3
```

1.12.8 CÁC BÀI TẬP CƠ BẢN TRONG JAVA

Plpsoft.Vn giới thiệu các bạn một vài chương trình kinh điển trong java thường được hỏi khi đi phỏng vấn. Các chương trình này có liên quan đến cấu trúc điều khiển, mảng, Collection, String, lập trình hướng đối tượng (OOP), ... Dưới đây là danh sách chương trình java kinh điển.

Bài 1. Dãy số Fibonacci

Viết một chương trình java in ra dãy số Fibonacci không sử dụng đệ quy và có sử dụng đệ quy.

Input: 10

Output: 0 1 1 2 3 5 8 13 21 34

Bài 2. Số nguyên tố

Viết một chương trình java kiểm tra số nguyên tố.

Input: 44

Output: không phải là số nguyên tố.

Input: 7

Output: là số nguyên tố.

Bài 3. Giai thừa

Viết một chương trình java tính giai thừa của một số không sử dụng đệ quy và có sử dụng đệ quy.

Input: 0

Output: 1

Input: 5

Output: 120

Bài 4. Thuật toán nổi bọt

Viết một chương trình java để sắp xếp dãy số bằng cách sử dụng thuật toán nổi bọt.

Input: 18 9 33 4 84 32

Output: 4 9 18 32 33 84

Bài 5. Thuật toán chon

Viết một chương trình java để sắp xếp dãy số bằng cách sử dụng thuật toán chọn(Selection Sort).

Input: 18 9 33 4 84 32

Output: 4 9 18 32 33 84

Bài 6. Thuật toán chèn

Viết một chương trình java để sắp xếp dãy số bằng cách sử dụng thuật toán chèn(Insertion Sort).

Input: 18 9 33 4 84 32

Output: 4 9 18 32 33 84

Bài 7. Chuyển đổi hệ cơ số

Viết một chương trình java để chuyển đổi số nguyên N sang hệ cơ số B.

Input: Chuyển số 15 sang hệ cơ số 2

Output: 1111

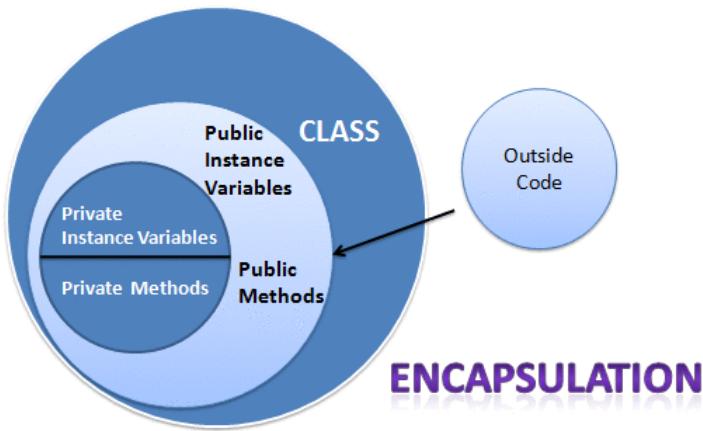
CHƯƠNG 2:

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG JAVA (JAVA OOPs)

2.1 Các tính chất của lập trình hướng đối tượng trong Java

2.1.1 Tính đóng gói trong java

Tính đóng gói trong java là kỹ thuật ẩn giấu thông tin không liên quan và hiện thị ra thông liên quan. Mục đích chính của đóng gói trong java là giảm thiểu mức độ phức tạp phát triển phần mềm.



Đóng gói cũng được sử dụng để bảo vệ trạng thái bên trong của một đối tượng. Bởi việc ẩn giấu các biến biểu diễn trạng thái của đối tượng. Việc chỉnh sửa đối tượng được thực hiện, xác nhận thông qua các phương thức. Hơn nữa, việc ẩn giấu các biến thì các lớp sẽ không chia sẻ thông tin với nhau được. Điều này làm giảm số lượng khớp nối có thể có trong một ứng dụng.

2.1.1.1 Lợi ích của đóng gói trong java

Bạn có thể tạo lớp **read-only** hoặc **write-only** bằng việc cài đặt phương thức setter hoặc getter.

Bạn có thể kiểm soát đối với dữ liệu. Giả sử bạn muốn đặt giá trị của id chỉ lớn hơn 100 bạn có thể viết logic bên trong lớp setter.

2.1.1.2 Ví dụ về đóng gói trong java

Hãy xem ví dụ sau về đóng gói trong java với một lớp chỉ có một trường và các phương thức setter và getter của nó.

File: Student.java

```
public class Student {  
    private String name;  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

File: Test.java

```
class Test {  
    public static void main(String[] args) {  
        Student s = new Student();  
        s.setName("Hai");  
        System.out.println(s.getName());  
    }  
}
```

Kết quả:

Hai

2.1.2. Tính kế thừa trong java

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

Tư tưởng của kế thừa trong java là có thể tạo ra một class mới được xây dựng trên các lớp đang tồn tại. Khi kế thừa từ một lớp đang tồn tại bạn có sử dụng lại các phương thức và thuộc tính của lớp cha, đồng thời có thể khai báo thêm các phương thức và thuộc tính khác.

2.1.2.1. Cú pháp của kế thừa trong java

Sử dụng từ khóa `extends` để kế thừa.

```
class Subclass-name extends Superclass-name {
    //methods and fields
}
```

2.1.2.2. Ví dụ về kế thừa trong java

```
class Employee {
    float salary = 1000;
}

class Programmer extends Employee {
    int bonus = 150;
}

public class InheritanceSample1 {
    public static void main(String args[]) {
        Programmer p = new Programmer();
        System.out.println("Programmer salary is: " + p.salary);
        System.out.println("Bonus of Programmer is: " + p.bonus);
    }
}
```

Kết quả:

```
Programmer salary is: 1000.0
```

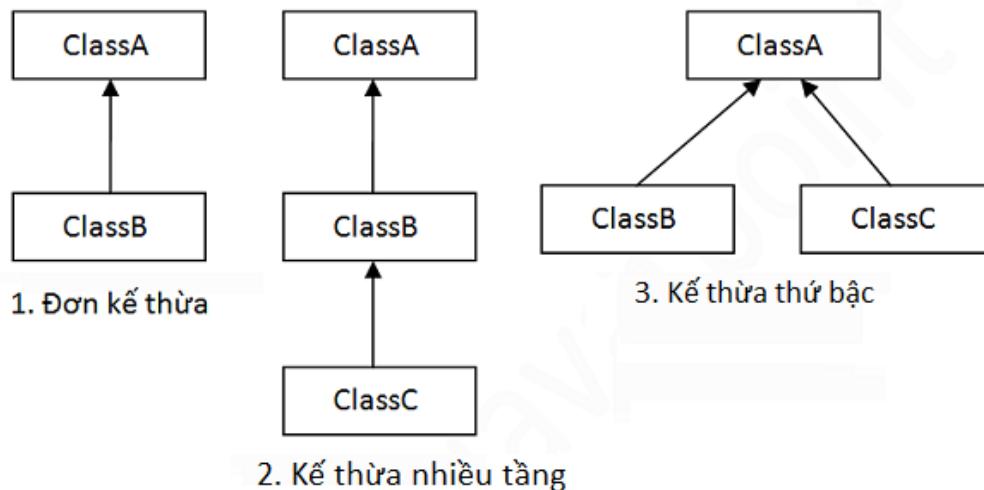
```
Bonus of Programmer is: 150
```

Trong ví dụ trên class `Programmer` là con của class `Employee`, nên nó được phép truy cập đến trường `salary` của class cha.

2.1.2.3. Các kiểu kế thừa trong java

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

Khi một class được kế thừa từ nhiều class được gọi là đa kế thừa. Trong java, đa kế thừa chỉ được support thông qua interface, như đã được nói đến trong bài *interface trong java*



Chú ý: Đa kế thừa trong java không được support thông qua class.

2.1.2.4. Ví dụ về đơn kế thừa

File: TestInheritance1.java

```

class Animal {
    void eat() {
        System.out.println("eating...");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("barking...");
    }
}

public class TestInheritance1 {
    public static void main(String args[]) {
        Dog d = new Dog();
    }
}

```

```
        d.bark();  
        d.eat();  
    }  
}
```

Output:

```
barking...  
eating...
```

2.1.2.5. Ví dụ về kế thừa nhiều cấp

File: TestInheritance2.java

```
class Animal {  
    void eat() {  
        System.out.println("eating...");  
    }  
}  
  
class Dog extends Animal {  
    void bark() {  
        System.out.println("barking...");  
    }  
}  
  
class BabyDog extends Dog {  
    void weep() {  
        System.out.println("weeping...");  
    }  
}  
  
public class TestInheritance2 {  
    public static void main(String args[]) {  
        BabyDog d = new BabyDog();  
        d.weep();  
        d.bark();  
    }  
}
```

```
    d.eat();  
}  
}
```

Kết quả:

```
weeping...  
barking...  
eating...
```

2.1.2.6. Ví dụ về kế thừa thứ bậc

File: TestInheritance3.java

```
class Animal {  
    void eat() {  
        System.out.println("eating...");  
    }  
}  
  
class Dog extends Animal {  
    void bark() {  
        System.out.println("barking...");  
    }  
}  
  
class Cat extends Animal {  
    void meow() {  
        System.out.println("meowing...");  
    }  
}  
  
public class TestInheritance3 {
```

```

public static void main(String args[]) {
    Cat c = new Cat();
    c.meow();
    c.eat();
    // c.bark(); // compile error
}
}

```

Kết quả:

```

meowing...
eating...

```

2.1.2.7. Tại sao đa kế thừa không được support trong java?

Để giảm thiểu sự phức tạp và đơn giản hóa ngôn ngữ, đa kế thừa không được support trong java.

Hãy suy xét kịch bản sau: Có 3 lớp A, B, C. Trong đó lớp C kế thừa từ các lớp A và B. Nếu các lớp A và B có phương thức giống nhau và bạn gọi nó từ đối tượng của lớp con, như vậy khó có thể xác định được việc gọi phương thức của lớp A hay B.

Vì vậy lỗi khi biên dịch sẽ tốt hơn lỗi khi runtime, java sẽ print ra lỗi "compile time error" nếu bạn có tình kế thừa 2 class.

```

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

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

```

```

        }

public class C extends A, B {
    public static void main(String args[]) {
        C obj = new C();
        obj.msg();
    }
}

```

Kết quả:

Compile Time Error

2.1.3. Tính đa hình trong java

Đa hình trong java (Polymorphism) là một khái niệm mà chúng ta có thể thực hiện một hành động bằng nhiều cách khác nhau. Polymorphism được cấu tạo từ 2 từ Hy Lạp: poly và morphs. Trong đó "poly" có nghĩa là nhiều và "morphs" có nghĩa là hình thể. Vậy polymorphism có nghĩa là nhiều hình thể.

Có hai kiểu của đa hình trong java, đó là đa hình lúc biên dịch (compile) và đa hình lúc thực thi (runtime). Chúng ta có thể thực hiện đa hình trong java bằng cách nạp chồng phương thức và ghi đè phương thức.

Nếu bạn nạp chồng phương thức static trong java, đó là một ví dụ về đa hình lúc biên dịch. Trong bài này, chúng ta tập trung vào đa hình lúc runtime trong java.

2.1.3.1. Đa hình lúc runtime trong java

Đa hình lúc runtime là quá trình gọi phương thức đã được ghi đè trong thời gian thực thi chương trình. Trong quá trình này, một phương thức được ghi đè được gọi thông qua biến tham chiếu của một lớp cha.

Trước khi tìm hiểu về đa hình tại runtime, chúng ta cùng tìm hiểu về Upcasting.

2.1.3.2. Upcasting là gì?

Khi biến tham chiếu của lớp cha tham chiếu tới đối tượng của lớp con, thì đó là Upcasting. **Ví dụ:**

```
class A{}
```

```
class B extends A{}  
  
A a=new B(); //upcasting
```

2.1.3.3. Ví dụ về đa hình lúc runtime trong java

Ví dụ 1:

Chúng ta tạo hai lớp Bike và Splendar. Lớp Splendar kế thừa lớp Bike và ghi đè phương thức run() của nó. Chúng ta gọi phương thức run bởi biến tham chiếu của lớp cha. Khi nó tham chiếu tới đối tượng của lớp con và phương thức lớp con ghi đè phương thức của lớp cha, phương thức lớp con được gọi lúc runtime.

Khi việc gọi phương thức được quyết định bởi JVM chứ không phải Compiler, vì thế đó là đa hình lúc runtime.

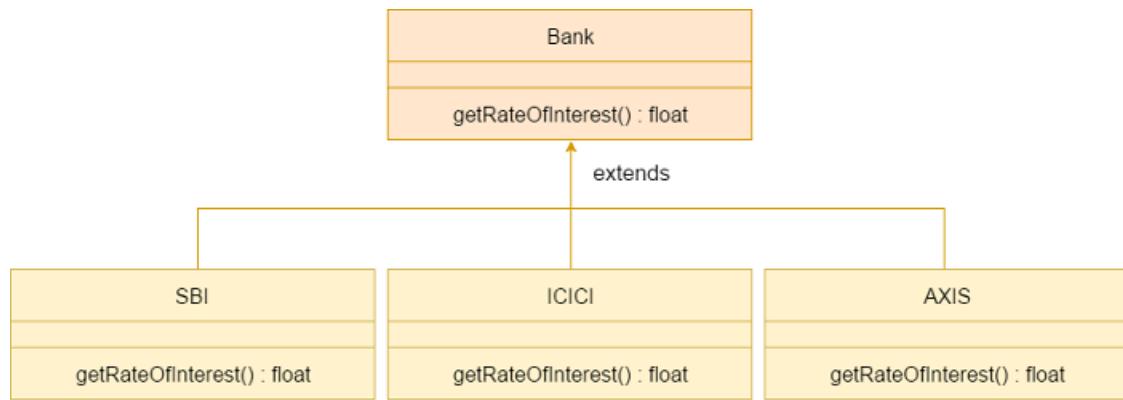
```
class Bike {  
    void run() {  
        System.out.println("running");  
    }  
}  
  
public class Splender extends Bike {  
    void run() {  
        System.out.println("running safely with 60km");  
    }  
  
    public static void main(String args[]) {  
        Bike b = new Splender(); // upcasting  
        b.run();  
    }  
}
```

Kết quả:

running safely with 60km

Ví dụ 2: Bank:

Giả sử Bank là một lớp cung cấp chức năng xem thông tin tỷ lệ lãi suất. Nhưng mỗi ngân hàng có một lãi xuất khác nhau, ví dụ các ngân hàng SBI, ICICI và AXIS có tỷ lệ lãi suất lần lượt là 8%, 7% và 9%.



```

class Bank {
    int getRateOfInterest() {
        return 0;
    }
}
  
```

```

class SBI extends Bank {
    int getRateOfInterest() {
        return 8;
    }
}
  
```

```

class ICICI extends Bank {
    int getRateOfInterest() {
        return 7;
    }
}
  
```

```

class AXIS extends Bank {
    int getRateOfInterest() {
        return 9;
    }
}
  
```

```
    }
}

public class Test2 {
    public static void main(String args[]) {
        Bank b;
        b = new SBI();
        System.out.println("SBI Rate of Interest: " +
b.getRateOfInterest());
        b = new ICICI();
        System.out.println("ICICI Rate of Interest: " +
b.getRateOfInterest());
        b = new AXIS();
        System.out.println("AXIS Rate of Interest: " +
b.getRateOfInterest());
    }
}
```

Kết quả:

```
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

Ví dụ 3: Shape:

```
class Shape {
    void draw() {
        System.out.println("drawing...");
    }
}

class Rectangle extends Shape {
    void draw() {
        System.out.println("drawing rectangle...");
    }
}
```

```
class Circle extends Shape {  
    void draw() {  
        System.out.println("drawing circle...");  
    }  
}  
  
class Triangle extends Shape {  
    void draw() {  
        System.out.println("drawing triangle...");  
    }  
}  
  
class TestPolymorphism2 {  
    public static void main(String args[]) {  
        Shape s;  
        s = new Rectangle();  
        s.draw();  
        s = new Circle();  
        s.draw();  
        s = new Triangle();  
        s.draw();  
    }  
}
```

Kết quả:

```
drawing rectangle...  
drawing circle...  
drawing triangle...
```

2.1.3.4. Đa hình lúc runtime trong Java với thành viên dữ liệu

Phương thức bị ghi đè không là thành viên dữ liệu, vì thế đa hình tại runtime không thể có được bởi thành viên dữ liệu. Trong ví dụ sau đây, cả hai lớp có một thành viên dữ liệu là speedlimit, chúng ta truy cập thành viên dữ liệu bởi biến tham chiếu của lớp cha mà tham chiếu tới đối tượng lớp con. Khi chúng ta

truy cập thành viên dữ liệu mà không bị ghi đè, thì nó sẽ luôn luôn truy cập thành viên dữ liệu của lớp cha.

Quy tắc: Đa hình lúc runtime không thể xảy ra với thành viên dữ liệu.

```
class Bike{  
    int speedlimit=90;  
}  
  
class Honda3 extends Bike{  
    int speedlimit=150;  
  
    public static void main(String args[]){  
        Bike obj=new Honda3();  
        System.out.println(obj.speedlimit);//90  
    }  
}
```

Kết quả:

90

2.1.3.5. Đa hình lúc runtime trong Java với kế thừa nhiều tầng

Ví dụ 1:

```
class Animal {  
    void eat() {  
        System.out.println("eating");  
    }  
}  
  
class Dog extends Animal {  
    void eat() {  
        System.out.println("eating fruits");  
    }  
}  
  
class BabyDog extends Dog {
```

```
void eat() {  
    System.out.println("drinking milk");  
}  
  
public static void main(String args[]) {  
    Animal a1, a2, a3;  
    a1 = new Animal();  
    a2 = new Dog();  
    a3 = new BabyDog();  
    a1.eat();  
    a2.eat();  
    a3.eat();  
}  
}
```

Kết quả:

```
eating  
eating fruits  
drinking Milk
```

Ví dụ 2:

```
class Animal {  
    void eat() {  
        System.out.println("animal is eating...");  
    }  
}  
  
class Dog extends Animal {  
    void eat() {  
        System.out.println("dog is eating...");  
    }  
}  
  
class BabyDog1 extends Dog {  
    public static void main(String args[]) {
```

```

        Animal a = new BabyDog1();
        a.eat();
    }
}

```

Kết quả:

Dog is eating

Vì BabyDog1 không ghi đè phương thức eat(), nên phương thức eat() của lớp Dog được gọi.

2.1.4. Nạp chồng phương thức trong java

2.1.4.1. Nạp chồng phương thức (method overloading)

Nạp chồng phương thức trong java xảy ra nếu một lớp có nhiều phương thức có tên giống nhau nhưng khác nhau về kiểu dữ liệu hoặc số lượng các tham số.

Giả sử bạn phải thực hiện tính tổng của các số đã cho với bất kỳ số lượng các đối số, nếu bạn viết phương thức a(int, int) cho 2 tham số, b(int, int, int) cho 3 tham số điều này có thể gây khó hiểu cho các lập trình viên khác về ý nghĩa của các phương thức đó vì chúng có tên khác nhau.

2.1.4.2. Lợi ích của nạp chồng phương thức

Sử dụng nạp chồng phương thức giúp tăng khả năng đọc hiểu chương trình.

2.1.4.3. Các cách nạp chồng phương thức

Có 2 cách nạp chồng phương thức trong java

- Thay đổi số lượng các tham số
- Thay đổi kiểu dữ liệu của các tham số

Trong java, không thể nạp chồng phương thức bằng cách chỉ thay đổi kiểu trả về của phương thức.

2.1.4.3.1. Nạp chồng phương thức: thay đổi số lượng các tham số

Trong ví dụ này, chúng ta cần tạo 2 phương thức, phương thức add() đầu tiên thực hiện việc tính tổng của 2 số, phương thức thứ hai thực hiện việc tính tổng của 3 số. Sử dụng phương thức static để gọi hàm thông qua tên class thay vì phải tạo thể hiện của lớp.

```

class Adder{
    static int add(int a,int b){return a+b; }
    static int add(int a,int b,int c){return a+b+c; }
}

class TestOverloading1{
    public static void main(String[] args){
        System.out.println(Adder.add(11,11));
        System.out.println(Adder.add(11,11,11));
    }
}

```

Kết quả:

```

22
33

```

2.1.4.3.2. Nạp chồng phương thức: thay đổi kiểu dữ liệu của các tham số

Trong ví dụ này, chúng ta sẽ tạo ra 2 phương thức có kiểu dữ liệu khác nhau. Phương thức add() đầu tiên nhận 2 đối số có kiểu giá trị là integer, phương thức thứ hai nhận 2 đối số có kiểu giá trị là double.

```

class Adder{
    static int add(int a, int b){return a+b; }
    static double add(double a, double b){return a+b; }
}

class TestOverloading2{
    public static void main(String[] args){
        System.out.println(Adder.add(11,11));
        System.out.println(Adder.add(12.3,12.6));
    }
}

```

Kết quả:

```

22
24.9

```

2.1.4.4. Các câu hỏi về nạp chồng phương thức trong java

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

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

Ví dụ:

```
class Adder{
    static int add(int a,int b){return a+b; }
    static double add(int a,int b){return a+b; }
}

class TestOverloading3{
    public static void main(String[] args){
        System.out.println(Adder.add(11,11));//ambiguity
    }
}
```

Kết quả:

```
Compile Time Error: method add(int,int) is already defined
in class Adder
```

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

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

```
public class TestOverloading4 {
    public static void main(String[] args) {
        System.out.println("main with String[]");
    }

    public static void main(String args) {
        System.out.println("main with String");
    }

    public static void main() {
        System.out.println("main without args");
    }
}
```

```

    }
}

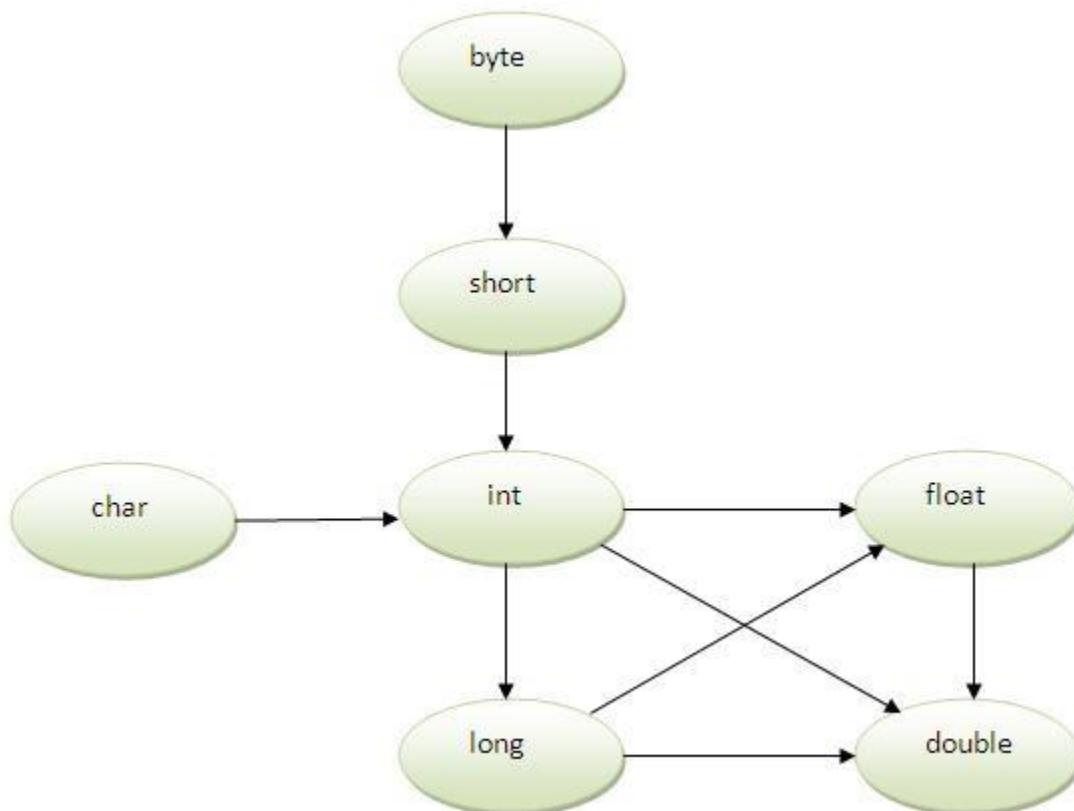
```

Kết quả:

```
main with String[]
```

2.1.4.5. Nạp chồng phương thức và sự thay đổi kiểu giá trị

Kiểu dữ liệu của đối số truyền vào được thay đổi sang kiểu dữ liệu khác (tự động ép kiểu) nếu giá trị của đối số đó không phù hợp với kiểu dữ liệu của tham số đã được định nghĩa. Để hiểu khái niệm này hãy xem ảnh sau:



Kiểu byte có thể được ép sang các kiểu short, int, long, float hoặc double. Kiểu dữ liệu short có thể được ép sang các kiểu int, long, float hoặc double. Kiểu dữ liệu char có thể được ép sang các kiểu int, long, float or double...

Ví dụ 1:

```
public 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();  
        // kiểu integer tham số 2 sẽ được thay đổi thành kiểu long  
        obj.sum(20, 20);  
        obj.sum(20, 20, 20);  
    }  
}
```

Kết quả:

```
40  
60
```

Ví dụ 2: nếu không có kiểu đối số nào phù hợp, chuyển đổi kiểu sẽ không được thực hiện.

```
public class OverloadingCalculation2 {  
    void sum(int a, int b) {  
        System.out.println("int arg method invoked");  
    }  
  
    void sum(long a, long b) {  
        System.out.println("long arg method invoked");  
    }  
  
    public static void main(String args[]) {  
        OverloadingCalculation2 obj = new OverloadingCalculation2();  
    }  
}
```

```
// phương thứ sum() có đối số int sẽ được gọi
obj.sum(20, 20);

}

}
```

Kết quả:

```
int arg method invoked
```

Ví dụ 3: không có kiểu đối số nào phù hợp trong phương thức và mỗi phương thức thay đổi số đối số tương tự nhau. Thì này sẽ không xác định được phương thức nào được gọi.

```
public class OverloadingCalculation3 {
    void sum(int a, long b) {
        System.out.println("a method invoked");
    }

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

    public static void main(String args[]) {
        OverloadingCalculation3 obj = new OverloadingCalculation3();
        // không xác định được phương thức nào được gọi
        obj.sum(20, 20);
    }
}
```

Kết quả:

```
Compile Time Error
```

2.1.5. Ghi đè phương thức trong java

2.1.5.1. Ghi đè phương thức (method overriding)

Ghi đè phương thức trong java xảy ra nếu lớp con có phương thức giống lớp cha.

Nói cách khác, nếu lớp con cung cấp sự cài đặt cụ thể cho phương thức đã được cung cấp bởi một lớp cha của nó được gọi là ghi đè phương thức (method overriding) trong java.

2.1.5.2. Sử dụng ghi đè phương thức trong java

- Ghi đè phương thức được sử dụng để cung cấp cài đặt đặc biệt của một phương thức mà đã được định nghĩa ở lớp cha.
- Ghi đè phương thức được sử dụng cho đa hình runtime.

2.1.5.3. Các nguyên tắc ghi đè phương thức trong java

1. Phương thức phải có tên giống với lớp cha.
2. Phương thức phải có tham số giống với lớp cha.
3. Lớp con và lớp cha có mối quan hệ kế thừa.

2.1.5.4. Ví dụ ghi đè phương thức trong java

Trong ví dụ này, chúng ta định nghĩa phương thức run() trong lớp con giống như đã được định nghĩa trong lớp cha, nhưng được cài đặt rõ ràng trong lớp con. Tên và tham số của phương thức là giống nhau, 2 lớp cha và con có quan hệ kế thừa.

```
class Vehicle {  
    void run() {  
        System.out.println("Vehicle is running");  
    }  
}  
  
public class Bike2 extends Vehicle {  
    void run() {  
        System.out.println("Bike is running safely");  
    }  
  
    public static void main(String args[]) {  
        Bike2 obj = new Bike2();  
    }  
}
```

```

        obj.run();
    }
}

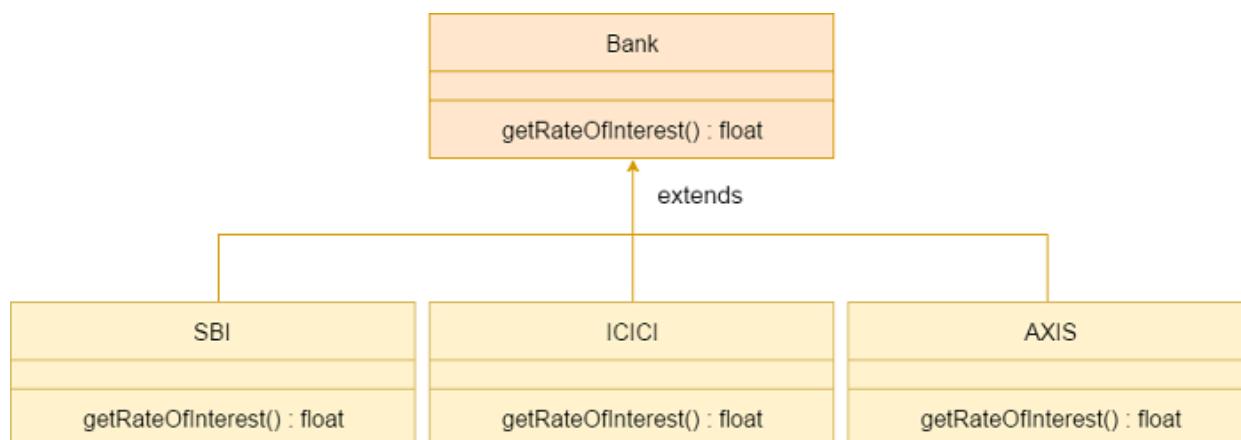
```

Kết quả:

Bike is running safely

2.1.5.5. Ví dụ thực tế về ghi đè phương thức trong java

Giả sử Bank là một lớp cung cấp chức năng xem thông tin tỷ lệ lãi suất. Nhưng mỗi ngân hàng có một lãi suất khác nhau, ví dụ các ngân hàng SBI, ICICI và AXIS có tỷ lệ lãi suất lần lượt là 8%, 7% và 9%.



Dưới đây là cài đặt cho ví dụ trên:

```

class Bank {
    int getRateOfInterest() {
        return 0;
    }
}

```

```

class SBI extends Bank {
    int getRateOfInterest() {
        return 8;
    }
}

```

```
class ICICI extends Bank {  
    int getRateOfInterest() {  
        return 7;  
    }  
}  
  
class AXIS extends Bank {  
    int getRateOfInterest() {  
        return 9;  
    }  
}  
  
public class Test2 {  
    public static void main(String args[]) {  
        SBI s = new SBI();  
        ICICI i = new ICICI();  
        AXIS a = new AXIS();  
        System.out.println("SBI Rate of Interest: " +  
s.getRateOfInterest());  
        System.out.println("ICICI Rate of Interest: " +  
i.getRateOfInterest());  
        System.out.println("AXIS Rate of Interest: " +  
a.getRateOfInterest());  
    }  
}
```

Kết quả:

```
SBI Rate of Interest: 8  
ICICI Rate of Interest: 7  
AXIS Rate of Interest: 9
```

2.1.5.6. Câu hỏi về ghi đè phương thức trong java

Có ghi đè được phương thức static không?

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

Tại sao không ghi đè được phương thức static?

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

Có ghi đè phương thức main được không?

Không, vì main là phương thức static.

2.1.7 Sự khác nhau giữa overloading và overriding trong java

2.1.7.1. Sự khác nhau giữa overloading và overriding trong java

Sự khác nhau giữa overloading và overriding phương thức trong java được thể hiện trong bảng sau:

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

2.1.7.2. Ví dụ nạp chồng phương thức

```
public class OverloadingExample {  
    static int add(int a, int b) {  
        return a + b;  
    }  
  
    static int add(int a, int b, int c) {  
        return a + b + c;  
    }  
}
```

2.1.7.3. Ví dụ ghi đè phương thức

```
class Animal {  
    void eat() {  
        System.out.println("eating...");  
    }  
}  
  
class Dog extends Animal {  
    void eat() {  
        System.out.println("eating bread...");  
    }  
}
```

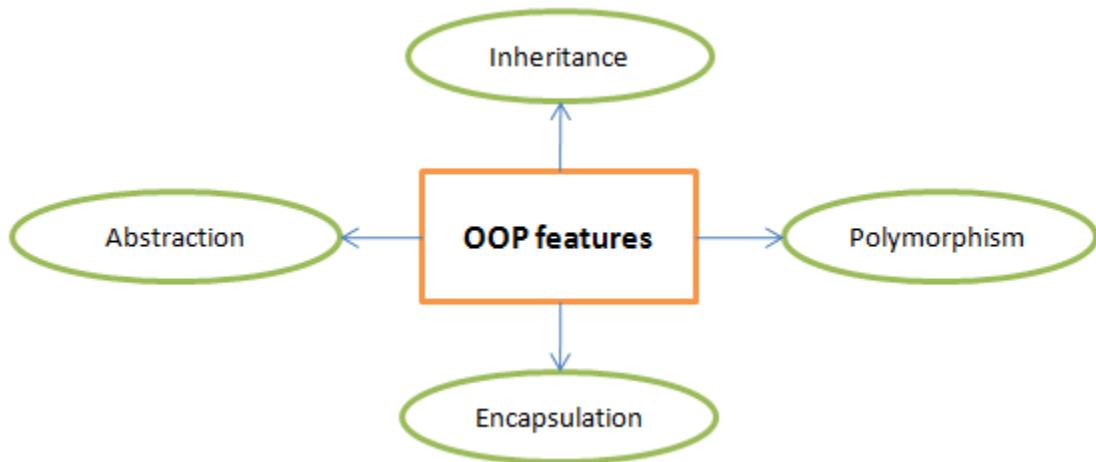
2.2 Các khái niệm lập trình hướng đối tượng java (java oops)

2.2.1. Khái niệm về lập trình hướng đối tượng trong java

Lập trình hướng đối tượng (OOP-Object-Oriented Programming) là một phương pháp hay mô hình giúp tăng năng suất, đơn giản hóa việc bảo trì, dễ dàng mở rộng trong thiết kế phần mềm bởi việc cung cấp một vài khái niệm như:

- Đối tượng (Object)
- Lớp (Class)
- Kế thừa (Inheritance)
- Đa hình (Polymorphism)
- Trừu tượng (Abstraction)
- Đóng gói (Encapsulation)

Lập trình hướng đối tượng có 4 tính chất:



2.2.1.2. Đối tượng (Object)

Tất cả những thực thể có trạng thái và hành vi được biết đến như là một đối tượng. Ví dụ: bàn, ghế, bút chì, xe đạp, ô tô...

2.2.1.3. Lớp (Class)

Tập hợp các đối tượng được gọi là lớp.

2.2.1.4. Kế thừa (Inheritance)

Khi một đối tượng được truyền lại tất cả các thuộc tính và phương thức của đối tượng cha được gọi là kế thừa. Kế thừa giúp tái sử dụng lại mã nguồn. Nó được sử dụng cho đa hình lúc runtime.

2.2.1.5. Đa hình (Polymorphism)

Khi một nhiệm vụ được thực hiện bởi nhiều cách khác nhau, tính chất này được gọi là đa hình. Ví dụ có nhiều cách để thuyết phục các khách hàng khác nhau, để vẽ một cái gì đó như hình tròn, hình chữ nhật, ...

Trong java, để áp dụng tính đa hình chúng ta sử dụng phương thức overloading hoặc overriding.

2.2.1.6. Trừu tượng (Abstraction)

Trùu tượng là sự ẩn đi những chi tiết bên trong và hiển thị ra các chức năng, tính chất này gọi là trừu tượng. Ví dụ: khi gọi điện thoại chúng ta không biết xử lý nội bộ thế nào, khi đi xe máy cũng vậy, mà chúng ta chỉ biết đến các chức năng thông qua giao tiếp bên ngoài.

Trong java, chúng ta áp dụng tính chất trừu tượng bằng cách sử dụng abstract class và interface.

2.2.1.7. Đóng gói (Encapsulation)

Việc ràng buộc giữa code và data với nhau tạo thành một khối duy nhất được biết đến là đóng gói. Ví dụ: viên thuốc con nhộng được đóng gói với nhiều loại thuốc bên trong.

Một class trong java là một ví dụ về đóng gói. Java bean là một lớp được đóng gói hoàn toàn vì tất cả các dữ liệu thành viên là private.

2.2.1.8. Thế mạnh của OOPs so với ngôn ngữ lập trình hướng thủ tục

- 1) Lập trình hướng đối tượng giúp việc phát triển và bảo trì dễ dàng hơn. Trong khi phương pháp lập trình hướng thủ tục là không dễ dàng quản lý khi code lớn.
- 2) Lập trình hướng đối tượng có tính năng ẩn dấu thông tin, trong khi hướng thủ tục có thể truy cập dữ liệu toàn cục ở bất kỳ nơi nào
- 3) Lập trình hướng đối tượng cung cấp khả năng mô phỏng sự kiện thực tế hiệu quả hơn.

2.2.2. Lớp và đối tượng trong java

Trong bài này chúng ta sẽ học về lớp và đối tượng trong java. Trong phương pháp lập trình hướng đối tượng, chúng ta thiết kế chương trình bằng việc sử dụng các lớp và đối tượng.

2.2.2.1. Đối tượng

Một thực thể có trạng thái và hành vi được gọi là đối tượng. Ví dụ như máy pha cà phê, xe đạp, cái quạt...

Một đối tượng có ba đặc điểm:

- Trạng thái: Đại diện cho dữ liệu (giá trị) của một đối tượng.
- Hành vi: Đại diện cho hành vi (chức năng) của một đối tượng như gửi tiền, rút tiền, ...
- Danh tính: Danh tính của một đối tượng thường được cài đặt thông qua một ID duy nhất. ID này được ẩn dưới với user bên ngoài. Tuy nhiên nó được sử dụng trong nội bộ máy ảo JVM để định danh từng đối tượng.

Ví dụ: Bút chì là một đối tượng. Tên của nó là A, màu trắng, ... được gọi là trạng thái. Nó được sử dụng để viết, viết được gọi là hành vi.

Đối tượng(Object) là một thể hiện của một lớp(Class). Lớp là một mẫu hoặc thiết kế từ đó các đối tượng được tạo ra. Vì vậy, đối tượng là các thể hiện (kết quả) của một lớp.

2.2.2.2. Lớp

Một lớp là một nhóm đối tượng có các thuộc tính chung. Nó là một mẫu hoặc thiết kế từ đó các đối tượng được tạo ra.

Một lớp trong java có thể chứa:

- Thành viên dữ liệu
- Constructor
- Phương thức
- Khối lệnh
- Lớp và Interface

2.2.2.3. Các ví dụ đơn giản về lớp và đối tượng trong java

Ví dụ 1:

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

```
public class Student {  
    int id; // thành viên dữ liệu  
    String name; // thành viên dữ liệu  
  
    public static void main(String args[]) {  
        Student student1 = new Student(); // tạo một đối tượng student1  
        System.out.println(student1.id);  
        System.out.println(student1.name);  
    }  
}
```

Kết quả:

```
0 null
```

Ví dụ 2:

```
public class Student2 {  
    int id;  
    String name;  
  
    // phương thức insertRecord  
    void insertRecord(int id, String name) {  
        this.id = id;  
        this.name = name;  
    }  
  
    // phương thức displayInformation  
    void displayInformation() {  
        System.out.println(id + " " + name);  
    }  
  
    public static void main(String args[]) {  
        Student2 s1 = new Student2();  
        Student2 s2 = new Student2();
```

```
s1.insertRecord(111, "Viet");
s2.insertRecord(222, "Tuts");

s1.displayInformation();
s2.displayInformation();

}
```

Kết quả:

Viet

Tuts

Ví dụ 3:

```
public class Student3 {
    int id;
    String name;

    // constructor
    public Student3(int id, String name) {
        this.id = id;
        this.name = name;
    }

    // phương thức displayInformation
    void displayInformation() {
        System.out.println(id + " " + name);
    }

    public static void main(String args[]) {
        Student3 s1 = new Student3(111, "Viet");
        Student3 s2 = new Student3(222, "Tuts");

        s1.displayInformation();
        s2.displayInformation();
    }
}
```

```
    }  
}
```

Kết quả:

Viet

Tuts

2.2.3. Package trong java

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

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

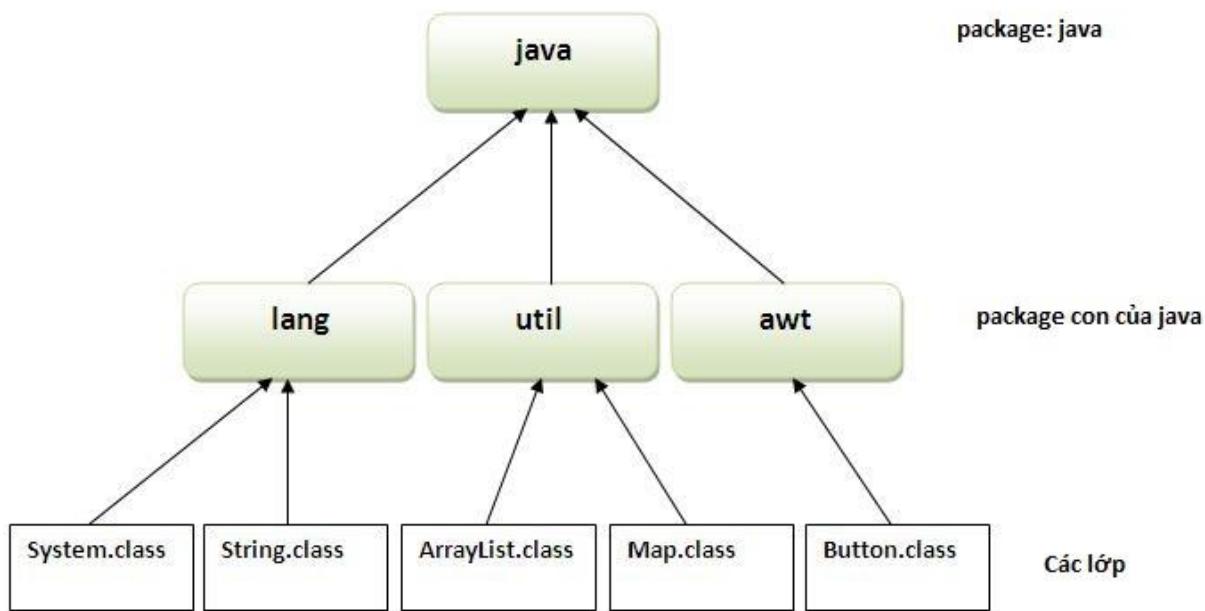
Có rất nhiều package được dựng sẵn như java, lang, AWT, javax, swing, net, io, util, sql, ...

Trong bài này, **chúng ta sẽ học:**

- Chi tiết về việc tạo và sử dụng các package người dùng định nghĩa.
- Ví dụ cách tổ chức package chuyên nghiệp trong java.

2.2.3.1. Lợi thế của việc sử dụng package trong java

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



2.2.3.2. Ví dụ về java package

Từ khóa **package** được sử dụng để tạo một package trong java

```
//Save file Simple.java
package mypack;
public class Simple {
    public static void main(String args[]) {
        System.out.println("Learn java package");
    }
}
```

2.2.3.3. Biên dịch java package

Nếu bạn không sử dụng IDE, bạn cần thực hiện theo cú pháp dưới đây:

```
javac -d directory javafilename
```

Ví dụ:

```
javac -d . Simple.java
```

Lệnh **-d** được sử dụng để xác định nơi lưu trữ file .class sau khi biên dịch. Bạn có thể sử dụng bất kỳ tên thư mục nào như /home (Trong Linux OS), D:/temp (Trong Window OS). Nếu bạn muốn giữ các package này trong thư mục hiện tại bạn sử dụng dấu chấm (.).

2.2.3.4. Run java package

Để run java package, bạn cần phải sử dụng tên đầy đủ. Ví dụ mypack.Simple.

Compile: javac -d . Simple.java

Run: java mypack.Simple

Kết quả:

```
Learn java package
```

Dấu chấm(.) biểu diễn thư mục hiện tại.

2.2.3.5. Truy cập package từ package khác

Có 3 cách để truy cập package từ package bên ngoài:

- Khai báo import package.*;
- Khai báo import package.classname;
- Sử dụng tên đầy đủ.

a. Sử dụng packagename.*

Nếu bạn sử dụng packagename.*. Thì tất cả các lớp và các interface của các gói này sẽ có thể truy cập, nhưng gói con của gói này thì không được truy cập.

Từ khóa import được sử dụng để truy cập các lớp và interface của gói khác từ gói hiện tại.

Ví dụ:

```
package pack;
public class A {
    public void msg() {
        System.out.println("Hello");
    }
}
```

```
package mypack;
```

```
import pack.*;  
  
class B {  
    public static void main(String args[]) {  
        A obj = new A();  
        obj.msg();  
    }  
}
```

Kết quả:

```
Hello
```

b. Sử dụng packagename.classname

Nếu bạn khai báo import package.classname thì chỉ được truy cập tới lớp đã được khai báo của package này.

Ví dụ:

```
package pack;  
  
public class A {  
    public void msg() {  
        System.out.println("Hello");  
    }  
}
```

```
package mypack;  
import pack.A;  
  
class B {  
    public static void main(String args[]) {  
        A obj = new A();  
        obj.msg();  
    }  
}
```

Kết quả:

```
Hello
```

c. Sử dụng tên đầy đủ

Nếu bạn sử dụng tên đầy đủ thì chỉ được truy cập tới lớp đã được khai báo của package này. Bạn không cần phải sử dụng đến từ khóa import. Nhưng bạn cần phải sử dụng tên đầy đủ mỗi khi bạn truy cập vào các lớp hoặc interface.

Cách này thường được sử dụng khi 2 package có tên lớp giống nhau. Ví dụ, 2 package java.util và java.sql chứa lớp có tên giống nhau là lớp Date

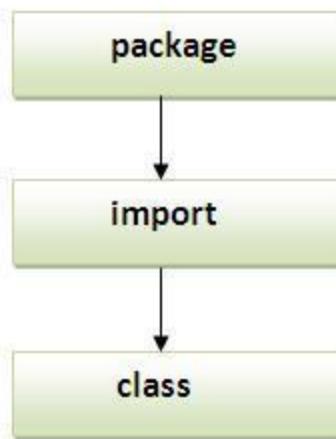
Ví dụ:

```
package pack;  
public class A {  
    public void msg() {  
        System.out.println("Hello");  
    }  
}  
  
package mypack;  
class B{  
    public static void main(String args[]){  
        pack.A obj = new pack.A(); //Sử dụng tên đầy đủ  
        obj.msg();  
    }  
}
```

Kết quả:

```
Hello
```

Note: Nếu bạn import một package thì package con của package đó không được import. Thứ tự của chương trình phải là package->import->class.



2.2.3.6. Package con trong java

Package bên trong một package khác được gọi là **subpackage** hay package con trong java.

Ví dụ, Sun Microsystem đã định nghĩa một gói có tên **java** chứa nhiều lớp như System, String, Reader, Writer, Socket, ... Các lớp này đại diện cho một nhóm cụ thể ví dụ như các lớp Reader và Writer cho các hoạt động Input/Output, Socket và ServerSocket các lớp xử lý mạng, ... Vì vậy, Sun đã phân loại lại gói java thành các gói phụ như lang, net, io, ... Và đặt các lớp liên quan đến Input/Output trong gói io, các lớp Server và ServerSocket trong các gói net.

Tiêu chuẩn để định nghĩa tên package trong java là **domain.company.package** ví dụ như: vn.plpsoft.action hoặc org.sssit.dao.

Ví dụ về Subpackage

```

package vn.plpsoft.core;

public class Simple {
    public static void main(String args[]) {
        System.out.println("Hello subpackage");
    }
}
  
```

Compile: javac -d . Simple.java

Run: java vn.plpsoft.core.Simple

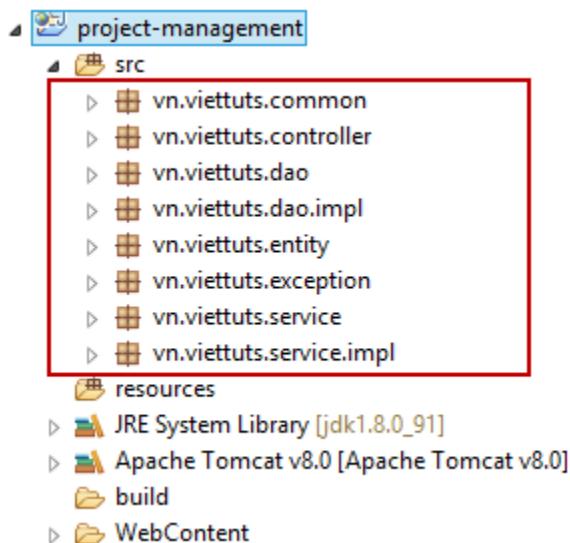
Kết quả:

```
Hello subpackage
```

2.2.3.7. Ví dụ cách tổ chức package chuyên nghiệp trong java

Các ví dụ ở trên cung cấp cho bạn cách để compile và run một lớp bên trong package do người dùng định nghĩa một cách thủ công. Khi đi làm về java trong môi trường thực tế, bạn sẽ phải học và sử dụng các IDE như eclipse, netbean, ... Những IDE này cung cấp cho bạn cách tạo và biên dịch cả một dự án đơn giản hơn rất nhiều. Khi đó bạn không cần phải nhớ các lệnh compile và run như ở trên.

Dưới đây là ví dụ cách tổ chức package chuyên nghiệp cho một dự án java web trên eclipse:



2.2.4. Constructor trong java

2.2.4.1. Constructor trong java

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

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

2.2.4.1.1 Các quy tắc tạo constructor trong java

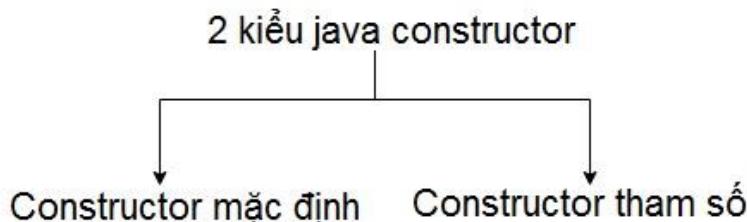
Có 2 quy tắc cơ bản cho việc tạo constructor:

1. Tên constructor phải giống tên lớp chứa nó.
2. Constructor không có kiểu trả về tường minh.

2.2.4.1.2. Các kiểu của java constructor

Có 2 kiểu của constructor

1. Constructor mặc định (không có tham số truyền vào)
2. Constructor tham số



2.2.4.2. Constructor mặc định trong java

Một constructor mà không có tham số được gọi là constructor mặc định.

Cú pháp của constructor mặc định:

```
<class_name>() {
    // code
}
```

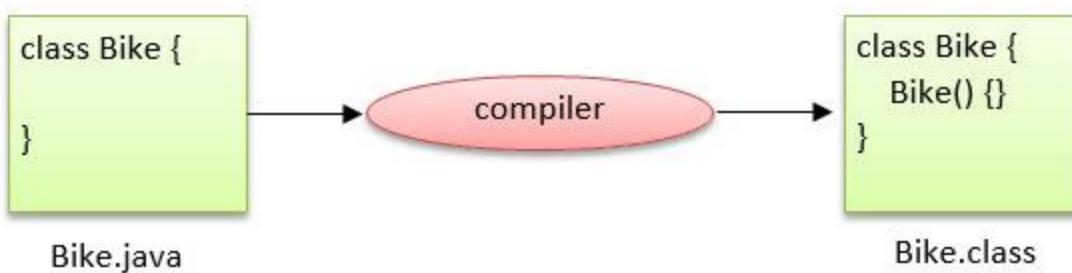
Ví dụ về constructor mặc định:

```
class Bike1 {
    Bike1() {
        System.out.println("Bike is created");
    }
    public static void main(String args[]) {
        Bike1 b=new Bike1();
    }
}
```

Output:

```
Bike is created
```

Quy tắc: Nếu không có constructor trong một lớp, trình biên dịch sẽ tự động tạo một constructor mặc định trong lớp đó.



Mục đích của constructor mặc định là gì?

Constructor mặc định cung cấp các giá trị mặc định như 0, null, (tùy thuộc vào kiểu dữ liệu) ... tới đối tượng được khởi tạo.

Ví dụ:

```

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();
    }
}
  
```

Output:

```
0 null
0 null
```

Giải thích: Trong ví dụ trên, bạn không tạo bất kỳ constructor nào, vì vậy trình biên dịch tự động tạo một constructor mặc định cho bạn. Giá trị 0 và null được cung cấp bởi constructor mặc định đó.

2.2.4.3. Constructor tham số trong java

Một constructor có tham số truyền vào được gọi là constructor tham số.

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

Ví dụ:

```
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, "Viet");  
        Student4 s2 = new Student4(222, "Tuts");  
        s1.display();  
        s2.display();  
    }  
}
```

Output:

```
111 Viet  
222 Tuts
```

2.2.4.4. Constructor Overloading trong java

Constructor Overloading là một kỹ thuật trong Java. Bạn có thể tạo nhiều constructor trong cùng một lớp với danh sách tham số truyền vào khác nhau. Trình biên dịch phân biệt các constructor này thông qua số lượng và kiểu của các tham số truyền vào.

Ví dụ:

```
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, "Viet");  
        Student5 s2 = new Student5(222, "Tuts", 25);  
        s1.display();  
        s2.display();  
    }  
}
```

Output:

```
111 Viet 0  
222 Tuts 25
```

2.2.4.5. Sự khác nhau giữa constructor và phương thức trong java

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

2.2.5. Từ khóa this trong java

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

Trong java, Từ khóa **this** có 6 cách sử dụng như sau:

1. Từ khóa this có thể được dùng để tham chiếu tới biến instance của lớp hiện tại.
2. this() có thể được dùng để gọi Constructor của lớp hiện tại.
3. Từ khóa this có thể được dùng để gọi phương thức của lớp hiện tại.
4. Từ khóa this có thể được truyền như một tham số trong phương thức.
5. Từ khóa this có thể được truyền như một tham số trong phương Constructor.
6. Từ khóa this có thể được dùng để trả về instance của lớp hiện tại.

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

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

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

Bạn sẽ hiểu ra vấn đề nếu không dùng từ khóa this trong ví dụ sau:

```
public 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, "Viet");
    Student10 s2 = new Student10(222, "Nam");
    s1.display();
    s2.display();
}
}

```

Kết quả:

```

0 null
0 null

```

Trong ví dụ trên, tên của tham số của Constructor Student10() trùng với tên của biến toàn cục đó là lý do tại sao cần phải sử dụng từ khóa this để phân biệt biến cục bộ và biến toàn cục.

Ví dụ dưới đây giải quyết vấn đề trên bằng cách sử dụng từ khóa this:

```

public 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, "Viet");
    Student11 s2 = new Student11(222, "Nam");
    s1.display();
    s2.display();
}
}
```

Kết quả:

```
111 Viet
222 Nam
```

Nếu biến cục bộ và biến toàn cục có tên khác nhau thì không cần sử dụng từ khóa this.

```
public 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, "Viet");
        Student12 e2 = new Student12(222, "Nam");
        e1.display();
        e2.display();
    }
}

```

Kết quả:

111 Viet

222 Name

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

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

Ví dụ:

```

public class Student13 {
    int id;
    String name;

    Student13() {
        System.out.println("call Constructor mặc định");
    }

    Student13(int id, String name) {
        this(); // nó được sử dụng để gọi Constructor của lớp hiện
        tại
        this.id = id;
        this.name = name;
    }

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

```

```

public static void main(String args[]) {
    Student13 e1 = new Student13(111, "Viet");
    Student13 e2 = new Student13(222, "Nam");
    e1.display();
    e2.display();
}
}

```

Kết quả:

```

call Constructor mặc định
call Constructor mặc định
111 Viet
222 Nam

```

4.5.3. Ví trí sử dụng this() để gọi Constructor

this() được dùng để sử dụng lại Constructor trong Constructor khác. Nó duy trì 1 chuỗi các Constructor.

Ví dụ:

```

public 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); // now no need to initialize id and name
        this.city = city;
    }
}

```

```
void display() {  
    System.out.println(id + " " + name + " " + city);  
}  
  
public static void main(String args[]) {  
    Student14 e1 = new Student14(111, "Viet");  
    Student14 e2 = new Student14(222, "Nam", "Ha Noi");  
    e1.display();  
    e2.display();  
}  
}
```

Kết quả:

```
111 Viet null  
222 Nam Ha Noi
```

Quy tắc: this() phải được khai báo dòng lệnh đầu tiên trong Constructor.

Ví dụ:

```
public class Student15 {  
    int id;  
    String name;  
  
    Student15() {  
        System.out.println("default Constructor is invoked");  
    }  
  
    Student15(int id, String name) {  
        id = id;  
        name = name;  
        this(); // must be the first statement  
    }  
  
    void display() {  
        System.out.println(id + " " + name);  
    }  
}
```

```

        }
    }

    public static void main(String args[]) {
        Student15 e1 = new Student15(111, "karan");
        Student15 e2 = new Student15(222, "Aryan");
        e1.display();
        e2.display();
    }
}

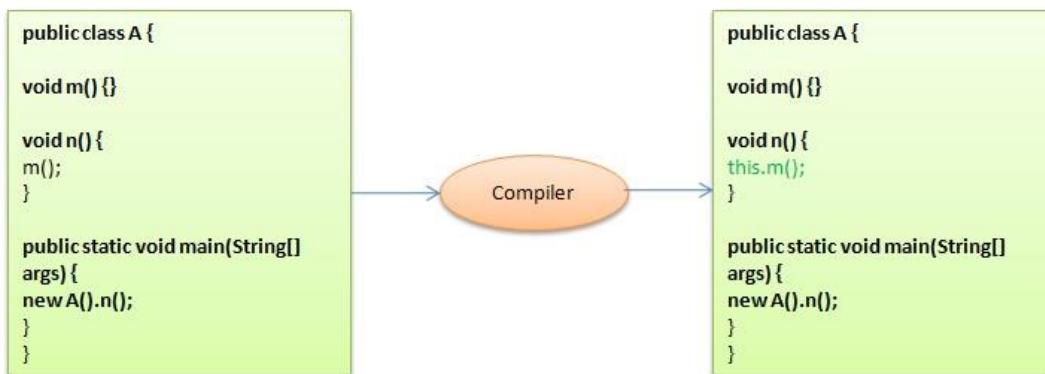
```

Kết quả:

Compile Time Error

2.2.5.4. Gọi phương thức của lớp hiện tại.

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



Hãy xem ví dụ sau để hiểu rõ hơn.

```

public class Example3 {
    void m() {
        System.out.println("Gọi phương thức bằng từ khóa
this");
    }

    void n() {

```

```

        this.m();
    }

    void p() {
        n(); // trình biên dịch sẽ thêm this để gọi phương thức
    n() như this.n()
    }

    public static void main(String args[]) {
        Example3 o1 = new Example3();
        o1.p();
    }
}

```

Kết quả:

Gọi phương thức bằng từ khóa this

2.2.5.5. Sử dụng từ khóa this như một tham số của phương thức.

Từ khóa this có thể được dùng như một tham số trong phương thức. Cách dùng này chủ yếu được sử dụng trong xử lý sự kiện.

Hãy xem ví dụ sau để hiểu rõ hơn.

```

public class Example4 {
    void m(Example4 obj) {
        System.out.println("Hello Java");
    }

    void p() {
        m(this);
    }

    public static void main(String args[]) {
        Example4 o1 = new Example4();
        o1.p();
    }
}

```

```
    }  
}
```

Kết quả:

```
Hello Java
```

Từ khóa this được sử dụng như một tham số trong việc xử lý sự kiện hoặc trong trường hợp mà chúng ta phải cung cấp tham chiếu của một lớp cho một lớp khác.

2.2.5.6. Sử dụng từ khóa this như một tham số của Constructor.

Bạn cũng có thể truyền từ khóa this trong Constructor. Tính năng này rất hữu ích nếu chúng ta phải sử dụng một đối tượng trong nhiều lớp.

Ví dụ:

```
class B {  
    A4 obj;  
    B(A4 obj) {  
        this.obj=obj;  
    }  
    void display() {  
        System.out.println(obj.data); // sử dụng biến thành viên của  
lớp A4  
    }  
}  
  
class A4 {  
    int data=10;  
    A4 () {  
        B b = new B(this);  
        b.display();  
    }  
    public static void main(String args[]) {  
        A4 a = new A4();  
    }  
}
```

Kết quả:

10

2.2.5.7. Sử dụng từ khóa this để trả về instance của lớp hiện tại.

Chúng ta có thể trả về instance của lớp hiện tại bằng cách sử dụng từ khóa this. Trong trường hợp này, kiểu trả về của phương thức phải là kiểu class (không là kiểu nguyên thủy).

Ví dụ:

```
class A {
    A getA() {
        return this;
    }
    void msg() {
        System.out.println("Hello Java");
    }
}

class Test1 {
    public static void main(String args[]) {
        new A().getA().msg();
    }
}
```

Kết quả:

Hello java

Ví dụ chúng minh rằng từ khóa this tham chiếu tới biến instance của lớp hiện tại. 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 chiếu ID

    obj.m();
}
}

```

Kết quả:

```
A5@22b3ea59
```

```
A5@22b3ea59
```

2.2.6. Từ khóa super trong java

Từ khóa super trong java là một biến tham chiếu được sử dụng để tham chiếu trực tiếp đến đối tượng của lớp cha gần nhất.

Bất cứ khi nào bạn tạo ra instance(thể hiện) của lớp con, một instance của lớp cha được tạo ra ngầm định, nghĩa là được tham chiếu bởi biến **super**.

Trong java, từ khóa **super** có 3 cách sử dụng như sau:

1. Từ khóa super được sử dụng để tham chiếu trực tiếp đến biến instance của lớp cha gần nhất.
2. super() được sử dụng để gọi trực tiếp Constructor của lớp cha.
3. Từ khóa super được sử dụng để gọi trực tiếp phương thức của lớp cha.

2.2.6.1. Tham chiếu trực tiếp đến biến instance của lớp cha.

Từ khóa super được sử dụng để tham chiếu trực tiếp đến biến instance của lớp cha.

Ví dụ: khi không sử dụng từ khóa super

```

class Vehicle {
    int speed = 50;
}

public class Bike extends Vehicle {
    int speed = 100;
}

```

```
void display() {  
    System.out.println(speed); //in speed của lớp Bike  
}  
  
public static void main(String args[]) {  
    Bike b = new Bike();  
    b.display();  
}  
}
```

Kết quả:

```
100
```

Ví dụ: khi sử dụng từ khóa super

```
class Vehicle {  
    int speed = 50;  
}  
  
public class Bike2 extends Vehicle {  
    int speed = 100;  
  
    void display() {  
        System.out.println(super.speed); //in speed của lớp Vehicle  
    }  
  
    public static void main(String args[]) {  
        Bike2 b = new Bike2();  
        b.display();  
    }  
}
```

Kết quả:

```
50
```

2.2.6.2. Sử dụng super()

Trong java, super() được sử dụng để gọi trực tiếp Constructor của lớp cha.

```
class Vehicle {
    Vehicle() {
        System.out.println("Vehicle is created");
    }
}

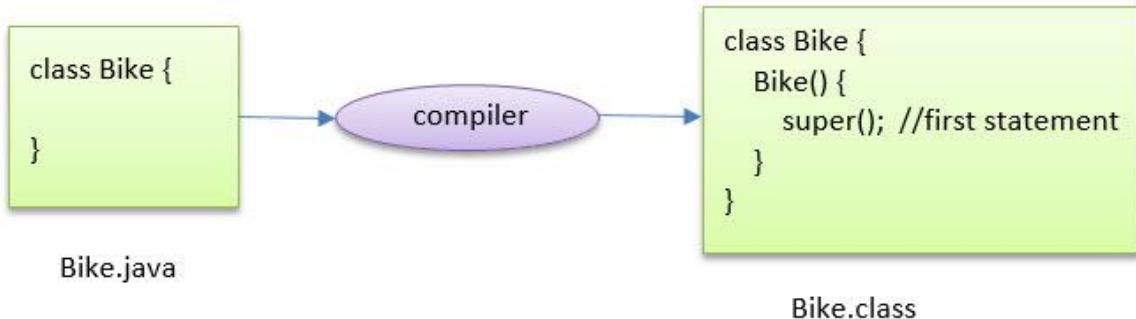
class Bike2 extends Vehicle {
    Bike2() {
        super(); //gọi Constructor của lớp cha
        System.out.println("Bike is created");
    }
}

public static void main(String args[]) {
    Bike2 b = new Bike2();
}
```

Kết quả:

```
Vehicle is created
Bike is created
```

Note: super() được tự động thêm vào mỗi Constructor của class bởi trình biên dịch.



Như chúng ta đã biết Constructor được tạo ra tự động bởi trình biên dịch nhưng nó cũng thêm super() vào câu lệnh đầu tiên. Nếu bạn tạo Constructor và bạn không có this() hoặc super() ở dòng lệnh đầu tiên, trình biên dịch sẽ cung cấp super() của Constructor.

Một ví dụ khác của từ khóa super nơi super() được cung cấp ngầm định bởi trình biên dịch.

```

class Vehicle {
    Vehicle() {
        System.out.println("Vehicle is created");
    }
}

public class Bike6 extends Vehicle {
    int speed;

    Bike6(int speed) {
        this.speed = speed;
        System.out.println(speed);
    }

    public static void main(String args[]) {
        Bike6 b = new Bike6(10);
    }
}

```

Kết quả:

```

Vehicle is created
10

```

2.2.6.3. super được sử dụng để gọi trực tiếp phương thức của lớp cha.

Tùy khóa super cũng có thể được sử dụng để gọi phương thức của lớp cha. Nó nên được sử dụng trong trường hợp lớp chứa các phương thức tương tự như lớp cha như trong ví dụ dưới đây:

```

class Person {
    void message() {
        System.out.println("welcomē");
    }
}

```

```
public class Student16 extends Person {  
    void message() {  
        System.out.println("welcome to java");  
    }  
  
    void display() {  
        message(); // gọi phương thức message() của lớp hiện tại  
        super.message(); // gọi phương thức message() của lớp cha  
    }  
  
    public static void main(String args[]) {  
        Student16 s = new Student16();  
        s.display();  
    }  
}
```

Kết quả:

```
welcome to java  
welcome
```

Trong ví dụ trên cả hai lớp Student và Person đều có phương thức message(). Nếu bạn gọi phương thức message() từ lớp Student thì phương thức message của Student sẽ được thực thi vì phương thức ở local sẽ được ưu tiên.

Trong TH không có phương thức ở class con giống class cha thì không cần phải sử dụng super. Ví dụ:

```
class Person {  
    void message() {  
        System.out.println("welcome");  
    }  
}  
  
public class Student17 extends Person {  
  
    void display() {
```

```

        message(); // will invoke parent class message() method
    }

    public static void main(String args[]) {
        Student17 s = new Student17();
        s.display();
    }
}

```

Kết quả:

```
welcome
```

2.2.7. Từ khóa final trong java

Từ khóa final trong Java được sử dụng để hạn chế người dùng. Từ khóa final có thể được sử dụng trong nhiều ngữ cảnh:

1. **Biến final:** bạn không thể thay đổi giá trị của biến final (nó sẽ là hằng số).
2. **Phương thức final:** bạn không thể ghi đè phương thức final.
3. **Lớp final:** bạn không thể kế thừa lớp final.
4. **Biến static final trống:** Một biến final mà không được khởi tạo tại thời điểm khai báo được gọi là biến final trống.

Từ khóa final có thể được áp dụng với các biến, một biến final mà không có giá trị nào được gọi là biến final trống hoặc biến final không được khởi tạo. Nó chỉ có thể được khởi tạo trong Constructor. Biến final trống cũng có thể là static mà sẽ chỉ được khởi tạo trong khối static. Chúng ta sẽ tìm hiểu chi tiết về những điều này.

2.2.7.1. Biến final trong Java

Nếu bạn tạo bất cứ biến nào là final, bạn không thể thay đổi giá trị của biến final (nó sẽ là hằng số).

Ví dụ của biến final trong Java:

Giả sử có một biến final là MAX_SPEED. Bạn cố ý thay đổi giá trị của biến này nhưng nó không bị thay đổi, bởi vì biến final một khi được gán giá trị thì không bao giờ thay đổi được.

```

public class FinalExam1 {
    final int MAX_SPEED = 90; // biến final
}

```

```

void run() {
    MAX_SPEED = 400;
}

public static void main(String args[]) {
    FinalExam1 obj = new FinalExam1();
    obj.run();
}
}

```

Kết quả:

```

Compile Time Error
Exception in thread "main" java.lang.Error: Unresolved
compilation problem:
    The final field FinalExam1.MAX_SPEED cannot be assigned

```

2.2.7.2. Phương thức final trong Java

Trong java, bạn không thể ghi đè phương thức final.

Ví dụ về phương thức final trong java

```

class Bike {
    final void run() {
        System.out.println("running");
    }
}

public class SH extends Bike {
    void run() {
        System.out.println("Chay an toan voi 150km/h");
    }
}

public static void main(String args[]) {
}

```

```

    SH sh = new SH();
    sh.run();
}
}

```

Kết quả:

```
Exception in thread "main" java.lang.VerifyError: class vn.plpsoft.keywords.SH overrides final method run.()V
```

2.2.7.3. Lớp final trong Java

Trong java, bạn không thể kế thừa lớp final.

Ví dụ về lớp final trong java:

```

final class Bike {
}

public class SH1 extends Bike {
    void run() {
        System.out.println("Chạy an toàn với 150km/h");
    }
}

public static void main(String args[]) {
    SH1 sh = new SH1();
    sh.run();
}
}

```

Kết quả:

```
Compile Time Error
```

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

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

```

class Bike {
    final void run() {

```

```

        System.out.println("running...");  

    }  

}  
  

class Honda2 extends Bike {  

    public static void main(String args[]) {  

        new Honda2().run();  

    }  

}

```

Kết quả:

running...

Câu hỏi: Biến final trống hoặc không được khởi tạo là gì

Một biến final mà không được khởi tạo tại thời điểm khai báo được gọi là biến final trống.

Nếu bạn muốn tạo một biến mà được khởi tạo tại thời điểm tạo đối tượng và một khi nó đã được khởi tạo thì không thể bị thay đổi, thì biến final trống là hữu ích trong trường hợp này. Ví dụ như số thẻ PAN CARD của một nhân viên.

Nó chỉ có thể được khởi tạo trong Constructor.

Ví dụ:

```

public class Student {  

    int id;  

    String name;  

    final String PAN_CARD_NUMBER; // Biến final trống  
  

    public Student() {  

        PAN_CARD_NUMBER = "7777";  

    }  

}

```

2.2.7.4. Biến static final trống trong Java

Một biến static final mà không được khởi tạo tại thời điểm khai báo thì đó là biến static final trống. Nó chỉ có thể được khởi tạo trong khối static.

Ví dụ:

```
public class A {
    static final int data; // Biến static final trống
    static {
        data = 50;
    }

    public static void main(String args[]) {
        System.out.println(A.data);
    }
}
```

Câu hỏi: Tham số final là gì?

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

Ví dụ:

```
class B {
    int cube(final int n) {
        n = n + 2; // không thể thay đổi được khi n là final
        return n;
    }

    public static void main(String args[]) {
        B b = new B();
        b.cube(5);
    }
}
```

Kết quả:

Compile Time Error

Câu hỏi: Chúng ta có thể khai báo một constructor final không?

Không, bạn không thể khai báo một constructor với từ khóa final trong java.

2.2.8. Từ khóa static trong java

Từ khóa static trong Java được sử dụng chính để quản lý bộ nhớ. Chúng ta có thể áp dụng từ khóa static với các biến, các phương thức, các khôi, các lớp lồng nhau(nested class). Từ khóa static thuộc về lớp chứ không thuộc về instance(thể hiện) của lớp.

Trong java, Static có thể là:

1. **Biến static:** Khi bạn khai báo một biến là static, thì biến đó được gọi là biến tĩnh, hay biến static.
2. **Phương thức static:** Khi bạn khai báo một phương thức là static, thì phương thức đó gọi là phương thức static.
3. **Khôi static:** Được sử dụng để khởi tạo thành viên dữ liệu static.

2.2.8.1. Biến static trong Java

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

1. 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, ...
2. Biến static lấy bộ nhớ chỉ một lần trong Class Area tại thời gian tải lớp đó.

2.2.8.1.1. Lợi thế của biến static

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

Vấn đề khi không sử dụng biến static

```
class Student{
    int rollno;
    String name;
    String college="Bưu Chính Viễn Thông";
}
```

Giả sử có 1000 sinh viên trong trường đại học, bây giờ instance của các dữ liệu thành viên sẽ sử dụng bộ nhớ mỗi khi đối tượng được tạo. Tất cả sinh viên có rollno và name là thuộc tính riêng. Tuy nhiên, college là thuộc tính chung của tất cả đối tượng. Nếu chúng ta tạo nó là static, thì trường này sẽ chỉ sử dụng bộ nhớ một lần để lưu biến này.

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

2.2.8.1.2. Ví dụ về biến static trong java

```

public class Student8 {
    int rollno;
    String name;
    static String college = "Bưu Chính Viễn Thông";

    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, "Thông");
    Student8 s2 = new Student8(222, "Minh");

    s1.display();
    s2.display();
}
}

```

Kết quả:

```

111 - Thông - Bưu Chính Viễn Thông
222 - Minh - Bưu Chính Viễn Thông

```

2.2.8.1.3. Chương trình đếm số không sử dụng biến static trong java

Trong ví dụ dưới đây, 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 ảnh hưởng đến các đối tượng khác. Vì thế mỗi đối tượng sẽ có giá trị 1 trong biến count.

```
public class Counter1 {  
    int count = 0; // sẽ lấy bộ nhớ khi instance được tạo ra  
  
    Counter1() {  
        count++;  
        System.out.println(count);  
    }  
  
    public static void main(String args[]) {  
  
        Counter1 c1 = new Counter1();  
        Counter1 c2 = new Counter1();  
        Counter1 c3 = new Counter1();  
  
    }  
}
```

Kết quả:

```
1  
1  
1
```

4.8.1.4. Chương trình đếm số có sử dụng biến static trong java

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

```
public class Counter2 {  
    static int count = 0; // sẽ lấy bộ nhớ chỉ một lần  
  
    Counter2() {  
        count++;  
        System.out.println(count);  
    }  
}
```

```
public static void main(String args[]) {  
  
    Counter2 c1 = new Counter2();  
    Counter2 c2 = new Counter2();  
    Counter2 c3 = new Counter2();  
  
}  
}
```

Kết quả:

```
1  
2  
3
```

2.2.8.2. 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.

1. Một phương thức static thuộc lớp chứ không phải đối tượng của lớp.
2. Một phương thức static gọi mà không cần tạo một instance của một lớp.
3. Phương thức static có thể truy cập biến static và có thể thay đổi giá trị của nó.

2.2.8.2.1. Ví dụ về phương thức static trong Java

```
public class Student9 {  
    int rollno;  
    String name;  
    static String college = "Bưu Chính Viễn Thông";  
  
    static void change() {  
        // Thay đổi thuộc tính static (thuộc tính chung của tất  
        // cả các đối tượng)  
        college = "Đại Học Công Nghệ";  
    }  
}
```

```

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, "Thông");
    Student9 s2 = new Student9(222, "Minh");
    Student9 s3 = new Student9(333, "Anh");

    s1.display();
    s2.display();
    s3.display();
}
}

```

Kết quả:

111 - Thông - Đại Học Công Nghệ

222 - Minh - Đại Học Công Nghệ

333 - Anh - Đại Học Công Nghệ

2.2.8.2.2. Sự hạn chế của phương thức static

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

1. Phương thức static không thể sử dụng biến non-static hoặc gọi trực tiếp phương thức non-static.
2. Từ khóa this và super không thể được sử dụng trong ngữ cảnh static.

Ví dụ:

```
class A {
    int a = 40; // non static

    public static void main(String args[]) {
        System.out.println(a);
    }
}
```

Kết quả:

Compile Time Error

2.2.8.3. Khối static trong Java

1. Được sử dụng để khởi tạo thành viên dữ liệu static.
2. Nó được thực thi trước phương thức main tại lúc tải lớp.

Ví dụ về khối static trong Java

```
public class A2 {
    static {
        System.out.println("Khối static: hello !");
    }

    public static void main(String args[]) {
        System.out.println("Main: hello !");
    }
}
```

Kết quả:

Khối static: hello !
Main: hello !

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

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

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

Có, một trong các cách đó là khôi static trong phiên bản trước của JDK. Không phải là JDK 1.7

Ví dụ:

```
public class A3 {  
    static {  
        System.out.println("static block is invoked");  
        System.exit(0);  
    }  
}
```

Kết quả: (TH < jdk7)

```
static block is invoked
```

Kết quả: (TH >= jdk7)

```
Error: Main method not found in class A3, please define the main  
method as:
```

```
public static void main(String[] args)
```

2.2.9. Access Modifier trong Java

Có hai loại Access Modifier trong Java, đó là: **Access Modifier** và **Non-access Modifier**.

Access Modifer trong Java xác định phạm vi có thể truy cập của biến, phương thức, constructor hoặc lớp.

Trong java, có 4 phạm vi truy cập của **Access Modifier** như sau:

1. **private**
2. **default**
3. **protected**
4. **public**

Ngoài ra, còn có nhiều **Non-access Modifier** như static, abstract, synchronized, native, volatile, transient,...

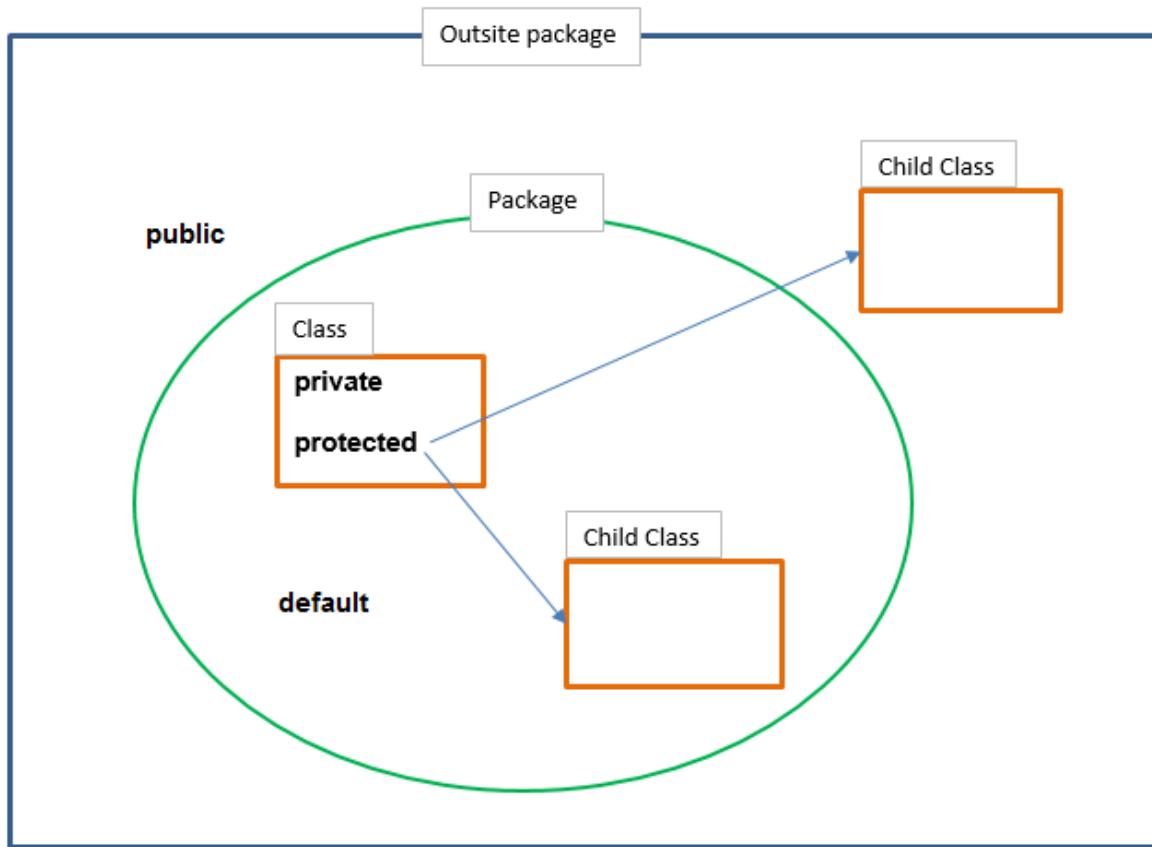
Bài này chúng ta sẽ tìm hiểu về các Access Modifier.

2.2.9.1. Tổng quan về Access Modifier trong java

Bảng dưới đây mô tả khả năng truy cập của các Access Modifier trong java:

Access Modifier	Trong lớp	Trong package	Ngoài package bởi lớp con	Ngoài package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Hình ảnh minh họa:



2.2.9.1.1. Phạm vi truy cập private

Private Access Modifier chỉ được truy cập trong phạm vi lớp.

Ví dụ về private access modifier trong java

Trong ví dụ, chúng ta tạo 2 lớp A và Simple. Lớp A chứa biến và phương thức được khai báo là private. Chúng ta cố gắng truy cập chúng từ bên ngoài lớp A. Điều này dẫn đến Compile time error:

```
class A {  
    private int data = 40;  
  
    private void msg() {  
        System.out.println("Hello java");  
    }  
}  
  
public class Simple {  
    public static void main(String args[]) {  
        A obj = new A();  
        System.out.println(obj.data); // Compile Time Error  
        obj.msg(); // Compile Time Error  
    }  
}
```

Vai trò của Private Constructor

Nếu bạn tạo bất kỳ constructor là private trong lớp, bạn sẽ không thể tạo instance của class bên ngoài nó. Ví dụ:

```
class A {  
    private A() {  
        //private constructor  
  
        public void msg() {  
            System.out.println("Hello java");  
        }  
    }  
}
```

```
public class Simple {  
    public static void main(String args[]) {  
        A obj = new A(); //Compile Time Error  
    }  
}
```

Chú ý: Một lớp không thể là private hoặc protected, ngoại trừ lớp lồng nhau.

2.2.9.1.2. Phạm vi truy cập default

Nếu bạn không khai báo modifier nào, thì nó chính là trường hợp mặc định. Default Access Modifier là chỉ được phép truy cập trong cùng package.

Ví dụ về Default Access Modifier trong Java

```
// Lưu file với tên A.java  
package vn.plpsoft.demo;  
  
class A {  
    void msg() {  
        System.out.println("Hello");  
    }  
}  
  
// Lưu file với tên B.java  
package vn.plpsoft.mypack;  
  
import vn.plpsoft.demo.*;  
  
public class B {  
    public static void main(String args[]) {  
        A obj = new A(); // Compile Time Error  
        obj.msg(); // Compile Time Error  
    }  
}
```

Trong ví dụ trên, phạm vi truy cập của lớp A và phương thức của msg() của nó là mặc định nên chúng không thể được truy cập từ bên ngoài package.

2.2.9.1.3. Phạm vi truy cập protected

Protected access modifier được truy cập bên trong package và bên ngoài package nhưng phải kế thừa.

Protected access modifier có thể được áp dụng cho biến, phương thức, constructor. Nó không thể áp dụng cho lớp.

Ví dụ về protected access modifier trong Java:

```
// Lưu file với tên A.java
package vn.plpsoft.demo;

public class A {
    protected void msg() {
        System.out.println("Hello");
    }
}

// Lưu file với tên B.java
package vn.plpsoft.mypack;

import vn.plpsoft.demo.*;

public class B extends A {
    public static void main(String args[]) {
        B obj = new B();
        obj.msg();
    }
}
```

Kết quả:

```
Hello
```

2.2.9.1.4. Phạm vi truy cập public

Public access modifier được truy cập ở mọi nơi.

Ví dụ về public access modifier trong java:

```
// Lưu file với tên A.java
package vn.plpsoft.demo;

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

// Lưu file với tên B.java
package vn.plpsoft.mypack;

import vn.plpsoft.demo.*;

public class B {
    public static void main(String args[]) {
        A obj = new A();
        obj.msg();
    }
}
```

Kết quả:

```
Hello
```

2.2.9.2. Lớp abstract trong java

Một lớp được khai báo với từ khóa abstract là lớp abstract trong Java. Lớp abstract có nghĩa là lớp trừu tượng, nó có thể có các phương thức abstract hoặc non-abstract.

Trước khi tìm hiểu về lớp trừu tượng trong Java, bạn cần hiểu tính trừu tượng trong Java là gì.

2.2.9.2.1. Tính trừu tượng trong java

Tính trừu tượng là một tiến trình ẩn các cài đặt chi tiết và chỉ hiển thị tính năng tới người dùng.

Nói cách khác, nó chỉ hiển thị các thứ quan trọng tới người dùng và ẩn các chi tiết nội tại, ví dụ: để gửi tin nhắn, người dùng chỉ cần soạn text và gửi tin. Bạn không biết tiến trình xử lý nội tại về phân phối tin nhắn.

Tính trừu tượng giúp bạn trọng tâm hơn vào đối tượng thay vì quan tâm đến cách nó thực hiện.

Các cách để đạt được sự trừu tượng hóa

Có 2 cách để đạt được sự trừu tượng hóa trong java

1. Sử dụng lớp abstract
2. Sử dụng interface

2.2.9.2.2. Phương thức trừu tượng trong Java

Một phương thức được khai báo là abstract và không có trình triển khai thì đó là phương thức trừu tượng.

Nếu bạn muốn một lớp chứa một phương thức cụ thể nhưng bạn muốn triển khai thực sự phương thức đó để được quyết định bởi các lớp con, thì bạn có thể khai báo phương thức đó trong lớp cha ở dạng abstract.

Từ khóa abstract được sử dụng để khai báo một phương thức dạng abstract. Một phương thức abstract không có thân phương thức.

Phương thức abstract sẽ không có định nghĩa, được theo sau bởi dấu chấm phẩy, không có dấu ngoặc nhọn theo sau:

```
// Khai bao phuong thuc voi tu khoa abstract va khong co than phuong thuc
abstract void printStatus();
```

Ví dụ về lớp trừu tượng và phương thức trừu tượng trong Java

Trong ví dụ này, Bike là lớp trừu tượng chỉ chứa một phương thức trừu tượng là run. Trình triển khai của nó được cung cấp bởi lớp Honda.

```
abstract class Bike{
    abstract void run();
}

class Honda4 extends Bike{
    void run() {
```

```

        System.out.println("running safely..");
    }

    public static void main(String args[]) {
        Bike obj = new Honda4();
        obj.run();
    }
}

```

Kết quả:

```
running safely..
```

2.2.9.2.3. Kế thừa lớp Abstract trong Java

Trong ví dụ này, Shape là lớp trừu tượng, trình triển khai của nó được cung cấp bởi lớp Rectangle và lớp Circle. Hai lớp này kế thừa lớp trừu tượng Shape.

TestAbstraction1.java

```

// lop truu tuong Shape
abstract class Shape{
    abstract void draw();
}

// Trong tinh huong nay, trinh trien khai duoc cung cap boi ai
do,
// vi du: nguoi su dung cuoi cung nao do
class Rectangle extends Shape{
    void draw() {
        System.out.println("Ve hinh chu nhat");
    }
}

class Circle1 extends Shape{
    void draw() {
        System.out.println("Ve hinh tron");
    }
}

```

```
//Trong tinh huong nay, phuong thuc duoc goi boi lap trinh vien  
hoac nguoi dung  
  
class TestAbstraction1{  
public static void main(String args[]) {  
    // Trong tinh huong nay, doi tuong duoc cung cap thong qua  
    phuong thuc,  
    // chang han nhu getShape()  
    Shape s=new Circle1();  
    s.draw();  
}  
}
```

Kết quả:

Vẽ hình tròn

2.2.10. Interface trong java

Một **Interface trong Java** là một bản thiết kế của một lớp. Nó chỉ có các phương thức trừu tượng. Interface là một kỹ thuật để thu được tính trừu tượng hoàn toàn và đa kế thừa trong Java. Interface trong Java cũng biểu diễn mối quan hệ IS-A. Nó không thể được khởi tạo giống như lớp trừu tượng.

Java Compiler thêm từ khóa **public** và **abstract** trước phương thức của interface và các từ khóa **public**, **static** và **final** trước các thành viên dữ liệu.

Nói cách khác, các trường của Interface là **public**, **static** và **final** theo mặc định và các phương thức là **public** và **abstract**.

Một Interface trong Java là một tập hợp các phương thức trừu tượng (**abstract**). Một class triển khai một interface, do đó kế thừa các phương thức abstract của interface.

Một interface không phải là một lớp. Viết một interface giống như viết một lớp, nhưng chúng có 2 định nghĩa khác nhau. Một lớp mô tả các thuộc tính và hành vi của một đối tượng. Một interface chứa các hành vi mà một class triển khai.

Trừ khi một lớp triển khai interface là lớp trừu tượng **abstract**, còn lại tất cả các phương thức của interface cần được định nghĩa trong class.

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

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

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

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

2.2.10.1. Ví dụ đơn giản về Interface trong Java

Trong ví dụ này, Printable Interface chỉ có một phương thức, trình triển khai của nó được cung cấp bởi lớp A.

```
interface printable {
    void print();
}

class A6 implements printable {
    public void print() {
        System.out.println("Hello");
    }

    public static void main(String args[]) {
        A6 obj = new A6();
        obj.print();
    }
}
```

Khi ghi đè các phương thức được định nghĩa trong interface, có một số qui tắc sau:

- Các checked exception không nên được khai báo trong phương thức implements, thay vào đó nó nên được khai báo trong phương thức interface hoặc các lớp phụ được khai báo bởi phương thức interface.
- Signature (ký số) của phương thức interface và kiểu trả về nên được duy trì khi ghi đè phương thức (overriding method).
- Một lớp triển khai chính nó có thể là abstract và vì thế các phương thức interface không cần được triển khai.

Khi triển khai interface, có vài quy tắc sau:

- Một lớp có thể triển khai một hoặc nhiều interface tại một thời điểm.
- Một lớp chỉ có thể kế thừa một lớp khác, nhưng được triển khai nhiều interface.
- Một interface có thể kế thừa từ một interface khác, tương tự cách một lớp có thể kế thừa lớp khác.

2.2.10.2. Đa kế thừa trong Java bởi Interface

Nếu một lớp triển khai đa kế thừa, hoặc một Interface kế thừa từ nhiều Interface thì đó là đa kế thừa.

```
interface Printable {  
    void print();  
}  
  
interface Showable{  
    void show();  
}  
  
class A7 implements Printable,Showable {  
  
    public void print() {  
        System.out.println("Hello");  
    }  
    public void show() {  
        System.out.println("Welcome");  
    }  
  
    public static void main(String args[]) {  
        A7 obj = new A7();  
        obj.print();  
        obj.show();  
    }  
}
```

Câu hỏi: Đa kế thừa không được hỗ trợ thông qua lớp trong Java nhưng là có thể bởi Interface, tại sao?

Như đã thảo luận trong chương về tính kế thừa, đa kế thừa không được hỗ trợ thông qua lớp. Nhưng nó được hỗ trợ bởi Interface bởi vì không có tính lưỡng nghĩa khi trình triển khai được cung cấp bởi lớp Implementation. Ví dụ:

```
interface Printable{
    void print();
}

interface Showable{
    void print();
}

class TestInterface1 implements Printable, Showable{
    public void print() {
        System.out.println("Hello");
    }

    public static void main(String args[]) {
        TestInterface1 obj = new TestInterface1();
        obj.print();
    }
}
```

Trong ví dụ trên, Printable và Showable interface có cùng các phương thức nhưng trình triển khai của nó được cung cấp bởi lớp TestInterface1, vì thế không có tính lưỡng nghĩa ở đây.

2.2.10.3. Kế thừa Interface trong Java

Một lớp triển khai Interface nhưng một Interface kế thừa từ Interface khác.

```
interface Printable{
    void print();
}

interface Showable extends Printable{
    void show();
}
```

```

}

class Testinterface2 implements Showable{

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

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

    public static void main(String args[]){
        Testinterface2 obj = new Testinterface2();
        obj.print();
        obj.show();
    }
}

```

2.2.10.4. Marker (hay Tagging) Interface trong Java là gì?

Đó là một Interface mà không có thành viên nào. Ví dụ: Serializable, Cloneable, Remote, ... Chúng được sử dụng để cung cấp một số thông tin thiết yếu tới JVM để mà JVM có thể thực hiện một số hoạt động hữu ích.

```

public interface Serializable {
}

```

Có hai mục đích thiết kế chủ yếu của tagging interface là:

- Tạo một cha chung: Như với EventListener interface, mà được kế thừa bởi hàng tá các interface khác trong Java API, bạn có thể sử dụng một tagging interface để tạo một cha chung cho một nhóm interface. Ví dụ, khi một interface kế thừa EventListener, thì JVM biết rằng interface cụ thể này đang được sử dụng trong một event.
- Thêm một kiểu dữ liệu tới một class: Đó là khái niệm tagging. Một class mà triển khai một tagging interface không cần định nghĩa bất kỳ phương thức nào, nhưng class trở thành một kiểu interface thông qua tính đa hình (polymorphism).

2.2.10.5. Interface lồng nhau trong Java

Một Interface có thể có Interface khác, đó là lồng Interface. Chúng ta sẽ tìm hiểu chi tiết trong chương về Lồng các lớp trong Java. Ví dụ:

```
interface printable{
    void print();
    interface MessagePrintable{
        void msg();
    }
}
```

2.2.11. Sự khác nhau giữa Abstract class và Interface

2.2.11.1. Sự khác nhau giữa Abstract class và Interface

Abstract class và interface đều được sử dụng để có được sự trừu tượng mà ở đó chúng ta có thể khai báo các phương thức trừu tượng. Nhưng có một vài sự khác nhau giữa abstract class và interface được liệt kê trong bảng sau:

Abstract class	Interface
1) Abstract class có phương thức abstract (không có thân hàm) và phương thức non-abstract (có thân hàm).	Interface chỉ có phương thức abstract . Từ java 8, nó có thêm các phương thức default và static .
2) Abstract class không hỗ trợ đa kế thừa .	Interface có hỗ trợ đa kế thừa
3) Abstract class có các biến final, non-final, static and non-static .	Interface chỉ có các biến static và final .
4) Abstract class có thể cung cấp nội dung cài đặt cho phương thức của interface .	Interface không thể cung cấp nội dung cài đặt cho phương thức của abstract class .
5) Từ khóa abstract được sử dụng để khai báo abstract class.	Từ khóa interface được sử dụng để khai báo interface.
6) Ví dụ: public abstract class Shape { public abstract void draw(); }	Ví dụ: public interface Drawable { void draw(); }

Đơn giản để hiểu, đó là abstract class có được trừu tượng 1 phần (0 tới 100%), còn interface có được trừu tượng toàn phần (100%).

2.2.11.2. Ví dụ về abstract class và interface trong java

Ví dụ sau là sự kết hợp giữa abstract class và interface.

```
//tạo interface có 4 phương thức
interface A {
    void a();
    abstract void b();
    public void c();
    public abstract void d();
}

// tạo abstract class cung cấp cài đặt cho một phương thức của
// interface A
abstract class B implements A {
    public void c() {
        System.out.println("I am C");
    }
}

// tạo subclass của abstract class B, cung cấp cài đặt cho các phương thức
// còn lại
class M extends B {
    public void a() {
        System.out.println("I am a");
    }

    public void b() {
        System.out.println("I am b");
    }

    public void d() {
        System.out.println("I am d");
    }
}
```

```
    }
}

// tạo lớp Test5 để gọi các phương thức của interface A
public class Test5 {
    public static void main(String args[]) {
        A a = new M();
        a.a();
        a.b();
        a.c();
        a.d();
    }
}
```

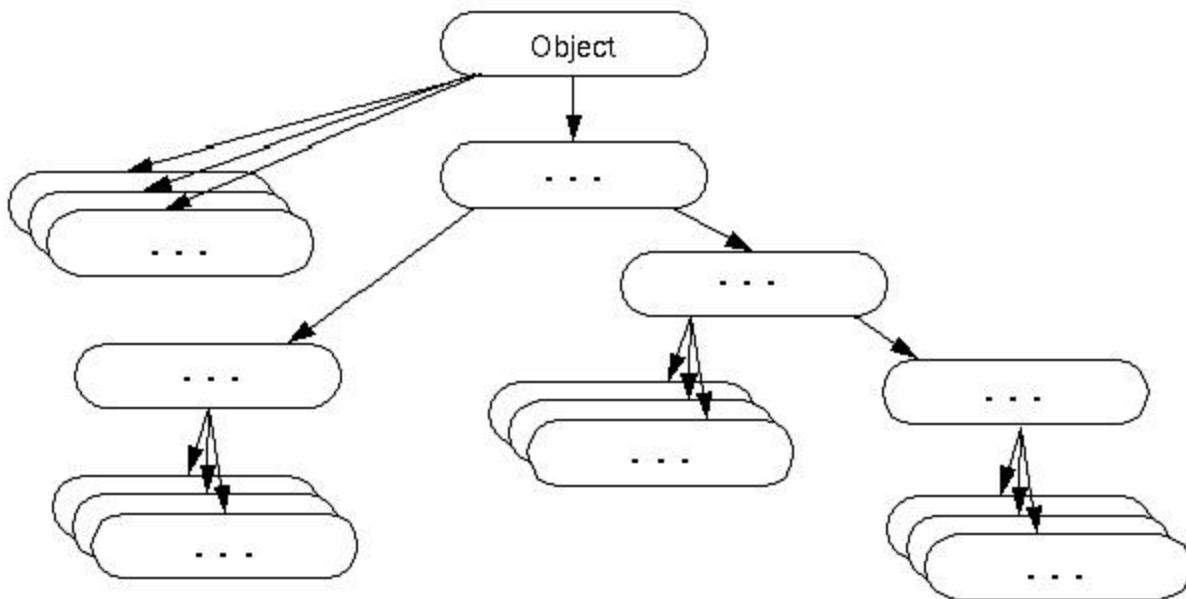
Output:

```
I am a
I am b
I am C
I am d
```

2.2.12. Lớp Object trong java

2.2.12.1. Lớp Object trong java

Mặc định lớp Object là lớp cha của tất cả các lớp trong java. Nói cách khác nó là một lớp cáo nhất trong java.



Sử dụng lớp Object là hữu ích nếu bạn muốn tham chiếu bất kỳ đối tượng nào mà bạn chưa biết kiểu dữ liệu của đối tượng đó. Chú ý rằng biến tham chiếu của lớp cha có thể tham chiếu đến đối tượng của lớp con được gọi là *upcasting*.

Ví dụ: giả sử phương thức getObject() trả về một đối tượng nhưng nó có thể là bất kỳ kiểu nào như Employee, Student, ... chúng ta có thể sử dụng biến tham chiếu của lớp Object để tham chiếu tới đối tượng đó.

```
Object obj=getObject(); // chúng ta không biết đối tượng gì được trả về từ phương thức này.
```

Lớp Object cung cấp một vài cách xử lý chung cho tất cả các đối tượng như đối tượng có thể được so sánh, đối tượng có thể được cloned, đối tượng có thể được notified...

2.2.12.2. Các phương thức của lớp Object

Lớp Object cung cấp các phương thức như trong bảng sau:

Phương thức	Mô tả
public final Class getClass()	trả về đối tượng lớp Class của đối tượng hiện tại. Từ lớp Class đó có thể lấy được các thông tin metadata của class hiện tại.
public int hashCode()	trả về số hashcode cho đối tượng hiện tại.

public boolean equals(Object obj)	so sánh đối tượng đã cho với đối tượng hiện tại.
protected Object clone() throws CloneNotSupportedException	tạo và trả về bản sao chép (clone) của đối tượng hiện tại.
public String toString()	trả về chuỗi ký tự đại diện của đối tượng hiện tại.
public final void notify()	đánh thức một luồng, đợi trình giám sát của đối tượng hiện tại.
public final void notifyAll()	đánh thức tất cả các luồng, đợi trình giám sát của đối tượng hiện tại.
public final void wait(long timeout)throws InterruptedException	làm cho Thread hiện tại đợi trong khoảng thời gian là số mili giây cụ thể, tới khi Thread khác thông báo (gọi phương thức notify() hoặc notifyAll()).
public final void wait(long timeout,int nanos)throws InterruptedException	làm cho Thread hiện tại đợi trong khoảng thời gian là số mili giây và nano giây cụ thể, tới khi Thread khác thông báo (gọi phương thức notify() hoặc notifyAll()).
public final void wait()throws InterruptedException	làm Thread hiện tại đợi, tới khi Thread khác thông báo (gọi phương thức notify() hoặc notifyAll()).
protected void finalize()throws Throwable	Được gọi bởi Garbage Collector trước khi đối tượng bị dọn rác.

2.2.13. Object cloning trong java

2.2.13.1. Khái niệm về Object cloning trong java

Object cloning là cách để tạo ra một bản sao chính xác của một đối tượng bị clone. Phương thức clone() được sử dụng để tạo ra một object mới.

Class của đối tượng mà chúng ta muốn clone phải được implements từ interface **java.lang.Cloneable**. Nếu chúng ta không implements interface Cloneable, phương thức clone() sinh ra lỗi ngoại lệ **CloneNotSupportedException**.

Phương thức phương thức clone() được định nghĩa trong lớp Object có cú pháp như sau:

```
protected native Object clone() throws CloneNotSupportedException;
```

2.2.13.2. Tại sao sử dụng phương thức clone()

Phương thức clone () tiết kiệm tác vụ xử lý bổ sung để tạo ra một bản sao chính xác của một đối tượng. Nếu chúng ta thực hiện nó bằng cách sử dụng từ khoá *new* sẽ mất rất nhiều tiền trình xử lý được thực hiện, đó là lý do tại sao chúng ta sử dụng object cloning.

2.2.13.3. Lợi ích của object cloning

Giảm thiểu tác vụ xử lý

2.2.13.4. Ví dụ về phương thức clone() (object cloning)

```
public class Student implements Cloneable {
    int rollno;
    String name;

    Student(int rollno, String name) {
        this.rollno = rollno;
        this.name = name;
    }

    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }

    public static void main(String args[]) {
        try {
            Student s1 = new Student(101, "An");
            Student s2 = (Student) s1.clone();
            System.out.println(s1.rollno + " " + s1.name);
            System.out.println(s2.rollno + " " + s2.name);
        } catch (CloneNotSupportedException c) {
        }
    }
}
```

Output:

```
101 An
101 An
```

Như bạn đã thấy trong ví dụ trên, cả hai biến tham chiếu có giá trị giống nhau. Vì phương thức `clone()` đã tạo ra một ô nhớ mới và sao chép giá trị của đối tượng này sang đối tượng khác.

Nếu bạn tạo một đối tượng khác bằng cách sử dụng từ khóa `new` và gán các giá trị của đối tượng khác cho đối tượng vừa tạo ra, nó sẽ mất nhiều tiến trình để xử lý trên đối tượng này. Vì vậy để tiết kiệm tác vụ xử lý ta sử dụng phương thức `clone()`.

2.2.13.5. Phương thức `equals()` và `hashCode()` trong java

Bài viết này giúp bạn hiểu khái niệm 2 phương thức quan trọng: **Phương thức `equals()` và `hashCode()` trong Java**.

Khi sử dụng các [collection](#), Để nhận được các hành vi mong muốn, chúng ta nên ghi đè các phương thức `equals()` và `hashCode()` trong các lớp của các phần tử được thêm vào collection.

Lớp Object (lớp cha của tất cả các lớp trong Java) định nghĩa hai phương thức `equal()` và `hashCode()`. Điều đó có nghĩa là tất cả các lớp trong Java (bao gồm cả các lớp bạn đã tạo) thừa kế các phương thức này. Về cơ bản, lớp Object thực hiện các phương thức này cho mục đích chung.

Tuy nhiên, bạn sẽ phải ghi đè chúng một cách cụ thể cho các lớp có đối tượng được thêm vào các collection, đặc biệt là các collection dựa trên bảng băm như [HashSet](#) và [HashMap](#).

2.2.13.5.1. Phương thức `equals()` trong Java

Khi so sánh hai đối tượng với nhau, Java gọi phương thức `equals()` của chúng trả về true nếu hai đối tượng bằng nhau hoặc false nếu hai đối tượng là khác nhau. Lưu ý rằng phép so sánh sử dụng phương thức `equals()` so với sử dụng toán tử `==` là khác nhau.

Đây là sự khác biệt:

Phương thức `equals()` được thiết kế để so sánh hai đối tượng về mặt ngữ nghĩa (bằng cách so sánh các thành viên dữ liệu của lớp), trong khi toán tử `==` so sánh hai đối tượng về mặt kỹ thuật (bằng cách so sánh các tham chiếu của chúng, nghĩa là địa chỉ bộ nhớ).

LƯU Ý: Việc cài đặt phương thức `equals()` trong lớp Object so sánh các tham chiếu của hai đối tượng. Điều đó có nghĩa là bạn nên ghi đè nó trong các lớp của bạn để so sánh ngữ nghĩa. Hầu hết các lớp trong JDK ghi đè phương thức `equals()` của riêng chúng, chẳng hạn như `String`, `Date`, `Integer`, `Double`, v.v.

a. Ví dụ phương thức `equals()` của đối tượng String

Ví dụ điển hình so sánh chuỗi trong Java, để thấy sự khác nhau giữa phương thức `equal()` và toán tử `==`.

```

package vn.plpsoft;

public class EqualExample1 {
    public static void main(String[] args) {
        String s1 = new String("This is a string");
        String s2 = new String("This is a string");

        System.out.println("s1 == s2: " + (s1 == s2));
        System.out.println("s1.equals(s2): " + (s1.equals(s2)));
    }
}

```

Kết quả:

```

s1 == s2: false
s1.equals(s2): true

```

So sánh tham chiếu (toán tử ==) trả về false vì s1 và s2 là hai đối tượng khác nhau được lưu trữ ở các vị trí khác nhau trong bộ nhớ. Trong khi so sánh ngữ nghĩa trả về true bởi vì s1 và s2 có cùng giá trị ("This is a string") có thể được coi là bằng nhau về mặt ngữ nghĩa.

b. Ví dụ ghi đè phương thức equals()

Tương tự như vậy, giả sử chúng ta có lớp Student và cài đặt phương thức equal() như sau:

Trong thực tế, chúng ta có thể xem xét hai đối tượng Student có ngữ nghĩa tương đương nhau nếu chúng có cùng thuộc tính (id, name, email và age). Jetzt, hãy xem cách ghi đè phương thức equals() trong lớp này để xác nhận rằng hai đối tượng Student có các thuộc tính giống nhau được coi là bằng nhau:

```

package vn.plpsoft;

public class Student {
    private String id;
    private String name;
    private String email;
    private int age;

    public Student(String id, String name, String email, int age) {

```

```
        this.id = id;
        this.name = name;
        this.email = email;
        this.age = age;
    }

    public String toString() {
        String studentInfo = "Student " + id;
        studentInfo += ": " + name;
        studentInfo += " - " + email;
        studentInfo += " - " + age;
        return studentInfo;
    }

    public boolean equals(Object obj) {
        if (obj instanceof Student) {
            Student another = (Student) obj;
            if (this.id.equals(another.id) &&
                this.name.equals(another.name) &&
                this.email.equals(another.email) &&
                this.age == another.age) {
                return true;
            }
        }
        return false;
    }
}
```

Tạo ra lớp EqualStudent.java để kiểm tra phương thức equal() trong lớp Student.

```
package vn.plpsoft;

public class EqualStudent {
    public static void main(String[] args) {
        Student student1 = new Student("123", "Cong",
```

```
"cong@gmail.com", 22);  
        Student student2 = new Student("123", "Cong",  
"cong@gmail.com", 22);  
        Student student3 = new Student("456", "Dung",  
"dung@gmail.com", 18);  
  
        System.out.println("student1 == student2: " + (student1 ==  
student2));  
        System.out.println("student1.equals(student2): "  
+ (student1.equals(student2)));  
        System.out.println("student2.equals(student3): "  
+ (student2.equals(student3)));  
    }  
}
```

Kết quả:

```
student1 == student2: false  
student1.equals(student2): true  
student2.equals(student3): false
```

c. Ví dụ ghi đè phương thức equals() của phần tử của collection

Ta có lớp Student.java có nội dung như sau:

```
package vn.plpsoft;  
  
public class Student {  
    private String id;  
    private String name;  
    private String email;  
    private int age;  
  
    public Student(String id) {  
        this.id = id;  
    }
```

```
public Student(String id, String name, String email, int age) {  
    this.id = id;  
    this.name = name;  
    this.email = email;  
    this.age = age;  
}  
  
public String toString() {  
    String studentInfo = "Student " + id;  
    studentInfo += ": " + name;  
    studentInfo += " - " + email;  
    studentInfo += " - " + age;  
    return studentInfo;  
}  
  
public boolean equals(Object obj) {  
    if (obj instanceof Student) {  
        Student another = (Student) obj;  
        if (this.id.equals(another.id)) {  
            return true;  
        }  
    }  
    return false;  
}  
}
```

Ở đây, phương thức equals() này chỉ so sánh thuộc tính ID của hai đối tượng Student. Chúng ta sẽ coi mỗi đối tượng Student có một ID duy nhất, và xem hai đối tượng sinh viên là bằng nhau nếu chúng có ID giống nhau.

Phương thức contains(Object) của interface List trong java có thể được sử dụng để kiểm tra nếu đối tượng được chỉ định tồn tại trong danh sách. Về bản chất, phương thức equal() được gọi bên trong phương thức contains(Object).

```
package vn.plpsoft;
```

```
import java.util.ArrayList;
import java.util.List;

public class EqualStudent2 {
    public static void main(String[] args) {
        Student student1 = new Student("123", "Cong", "cong@gmail.com", 22);
        Student student2 = new Student("123", "Cong", "cong@gmail.com", 22);
        Student student3 = new Student("456", "Dung", "dung@gmail.com", 18);

        // tạo danh sách student
        List<Student> listStudents = new ArrayList<>();

        // thêm các đối tượng student vào listStudents
        listStudents.add(student1);
        listStudents.add(student2);
        listStudents.add(student3);

        // tạo các đối tượng student chỉ có thuộc tính ID
        Student searchStudent1 = new Student("123");
        Student searchStudent4 = new Student("789");

        // tìm kiếm student trong danh sách
        System.out.println("Search student1: "
            + listStudents.contains(searchStudent1));
        System.out.println("Search student4: "
            + listStudents.contains(searchStudent4));
    }
}
```

Kết quả:

```
Search student1: true
Search student4: false
```

2.2.13.5.2. Phương thức hashCode() trong Java

Định nghĩa phương thức hashCode() trong lớp Object:

```
public native int hashCode();
```

Bạn có thể thấy phương thức này trả về một số nguyên. Vậy nó được sử dụng ở đâu?

Đây là bí mật:

Số băm này được sử dụng bởi các collection dựa trên bảng băm như Hashtable , HashSet và HashMap để lưu trữ các đối tượng trong các container nhỏ được gọi là "nhóm". Mỗi nhóm được liên kết với mã băm và mỗi nhóm chỉ chứa các đối tượng có mã băm giống hệt nhau.

Nói cách khác, một bảng băm nhóm các phần tử của nó bằng các giá trị mã băm của chúng. Sự sắp xếp này giúp cho bảng băm định vị một phần tử một cách nhanh chóng và hiệu quả bằng cách tìm kiếm trên các phần tử của collection thay vì toàn bộ collection.

Dưới đây là các bước để định vị một phần tử trong một bảng băm:

- Nhận giá trị mã băm của phần tử được chỉ định bằng cách gọi phương thức hashCode().
- Tìm nhóm thích hợp được liên kết với mã băm đó.
- Bên trong nhóm, tìm phần tử chính xác bằng cách so sánh phần tử được chỉ định với tất cả các phần tử trong nhóm. Bằng phương thức equals() của phần tử đã chỉ định được gọi.

Có nói rằng, khi chúng ta thêm các đối tượng của một lớp vào một collection dựa trên bảng băm (HashSet, HashMap), phương thức hashCode() của lớp được gọi để tạo ra một số nguyên (có thể là một giá trị tùy ý). Con số này được sử dụng bởi bộ sưu tập để lưu trữ và định vị các đối tượng một cách nhanh chóng và hiệu quả, vì collection dựa trên bảng băm không duy trì thứ tự các phần tử của nó.

LƯU Ý: Việc thực thi phương thức mặc định hashCode() trong lớp Object trả về một số nguyên là địa chỉ bộ nhớ của đối tượng. Bạn nên ghi đè phương thức trong các lớp của bạn. Hầu hết các lớp trong JDK ghi đè phương thức hashCode() của riêng chúng, chẳng hạn như String , Date , Integer , Double , v.v.

2.2.13.5.3. Các quy tắc cho phương thức equals() và hashCode() trong Java

Như đã giải thích ở trên, collection dựa trên bảng băm xác định một phần tử bằng cách gọi phương thức hashCode() và equals() của nó, vì vậy khi ghi đè các phương thức này chúng ta phải tuân theo các quy tắc sau:

- Khi phương thức equals() được ghi đè, phương thức hashCode() cũng phải được ghi đè.
- Nếu hai đối tượng bằng nhau, mã băm của chúng phải bằng nhau.

- Nếu hai đối tượng không bằng nhau, không có ràng buộc về mã băm của chúng (mã băm của chúng có thể bằng nhau hay không).
- Nếu hai đối tượng có mã băm giống nhau, thì không có ràng buộc nào về sự bình nhau của chúng (chúng có thể bằng nhau hay không).
- Nếu hai đối tượng có mã băm khác nhau, chúng không được bằng nhau.

Nếu chúng ta vi phạm các quy tắc này, các collection sẽ hoạt động có thể không đúng như các đối tượng không thể tìm thấy, hoặc các đối tượng sai được trả về thay vì các đối tượng chính xác.

➤ **Ví dụ ghi đè phương thức equals(), không ghi đè hashCode()**

Hãy xem phương thức hashCode() và equals() ảnh hưởng như thế nào đến hành vi của một đối tượng Set.

Lớp Student.java

```
package vn.plpsoft;

public class Student {
    private String id;
    private String name;
    private String email;
    private int age;

    public Student(String id) {
        this.id = id;
    }

    public Student(String id, String name, String email, int age) {
        this.id = id;
        this.name = name;
        this.email = email;
        this.age = age;
    }

    public String toString() {
        String studentInfo = "Student " + id;
```

```
        studentInfo += ":" + name;
        studentInfo += " - " + email;
        studentInfo += " - " + age;
        return studentInfo;
    }

    public boolean equals(Object obj) {
        if (obj instanceof Student) {
            Student another = (Student) obj;
            if (this.id.equals(another.id)) {
                return true;
            }
        }
        return false;
    }
}
```

Lớp EqualStudent3.java

```
package vn.plpsoft;

import java.util.HashSet;
import java.util.Set;

public class EqualStudent3 {
    public static void main(String[] args) {
        Student student1 = new Student("123", "Cong",
"cong@gmail.com", 22);
        Student student2 = new Student("123", "Cong",
"cong@gmail.com", 22);
        Student student3 = new Student("456", "Dung",
"dung@gmail.com", 18);

        Set<Student> setStudents = new HashSet<Student>();
        setStudents.add(student1);
```

```

        setStudents.add(student2);

        setStudents.add(student3);

        // in các phần tử của set ra màn hình
        for (Student student : setStudents) {
            System.out.println(student);
        }
    }
}

```

Kết quả:

Student 456: Dung - dung@gmail.com - 18

Student 123: Cong - cong@gmail.com - 22

Student 123: Cong - cong@gmail.com - 22

Hãy nhìn xem, bạn có nhận thấy rằng có 2 sinh viên trùng lặp (ID: 123), phải không?

Đây là lý do:

Tập Set gọi các phương thức equals() và hashCode() trên mỗi đối tượng được thêm vào để đảm bảo không có sự trùng lặp. Trong trường hợp của chúng ta, lớp Student chỉ ghi đè phương thức equals(). Và phương thức hashCode() thừa kế từ lớp Object trả về các địa chỉ bộ nhớ của mỗi đối tượng không nhất quán với phương thức equals(). Do đó, đối tượng Set xử lý đối tượng student1 và student2 thành hai phần tử khác nhau.

Bây giờ, chúng ta hãy ghi đè phương thức hashCode() trong lớp Student như sau:

```

public int hashCode() {
    return 31 + id.hashCode();
}

```

Kết quả:

Student 123: Cong - cong@gmail.com - 22

Student 456: Dung - dung@gmail.com - 18

Good! Phần tử trùng lặp hiện đã bị xóa. Đó chính là điều chúng tôi muốn.

Với các phương thức equals() và hashCode() được ghi đè đúng cách, chúng ta cũng có thể thực hiện tìm kiếm trên tập hợp như sau:

```

Student searchStudent = new Student("456");
boolean found = setStudents.contains(searchStudent);
System.out.println("Search student: " + found);

```

Kết quả:

```
Search student: true
```

Để tự mình thử nghiệm nhiều hơn, hãy thử loại bỏ phương thức equals() hoặc hashCode() và quan sát kết quả.

Hy vọng bài này giúp bạn hiểu cách sử dụng các phương thức equals() và hashCode(). Và áp dụng chúng cho các collection.

Tham khảo

[https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#equals-
java.lang.Object-](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#equals-java.lang.Object-)

<https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#hashCode-->

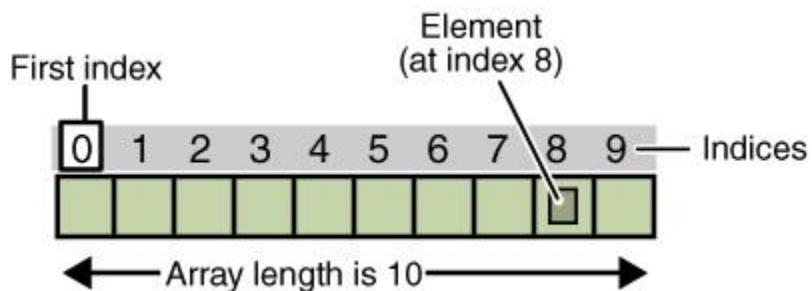
2.2.14. Mảng (Array) trong java

2.2.14.1. Mảng trong Java

Thông thường, mảng (array) là một tập hợp các phần tử có cùng kiểu được lưu trữ gần nhau trong bộ nhớ.

Mảng trong java là một đối tượng chứa các phần tử có kiểu dữ liệu giống nhau. Mảng là một cấu trúc dữ liệu nơi lưu trữ các phần tử giống nhau. Với mảng trong java chúng ta chỉ có thể lưu trữ một tập các phần tử có số lượng phần tử cố định.

Mảng trong java lưu các phần tử theo chỉ số, chỉ số của phần tử đầu tiên là 0.



2.2.14.2. Các kiểu của mảng trong java

Có hai kiểu mảng trong java

- Mảng một chiều
- Mảng đa chiều

2.2.14.3. Tạo một mảng trong Java

Để khai báo một mảng, khai báo loại biến với dấu ngoặc vuông []:

```
dataType[] arr;
dataType []arr;
dataType arr[];
```

Ví dụ chúng ta đã khai báo một biến chứa một chuỗi các chuỗi.

Để chèn giá trị vào nó, chúng ta có thể sử dụng một mảng bằng chữ - đặt các giá trị trong danh sách được phân tách bằng dấu phẩy, bên trong dấu ngoặc nhọn:

```
String[] cars = { "Honda", "BMW", "Ford", "Mazda" };
```

Để tạo một mảng các số nguyên, bạn có thể viết:

```
int[] myNum = { 10, 20, 30, 40 };
```

2.2.14.4. Truy cập các phần tử của một mảng trong Java

Bạn truy cập một phần tử mảng bằng cách sử dụng số chỉ mục (index).

Ví dụ:

```
package vn.plpsoft.array;

public class TruyCapArray1 {
    public static void main(String[] args) {
        String[] cars = { "Honda", "BMW", "Ford", "Mazda" };
        System.out.println(cars[0]);
    }
}
```

Kết quả:

Honda

Lưu ý: Chỉ mục của mảng bắt đầu bằng 0: [0] là phần tử đầu tiên. [1] là phần tử thứ hai, v.v.

2.2.14.5. Thay đổi một phần tử mảng trong Java

Để thay đổi giá trị của một phần tử cụ thể, hãy sử dụng số chỉ mục:

Ví dụ:

```

package vn.plpsoft.array;

public class TruyCapArray2 {
    public static void main(String[] args) {
        String[] cars = { "Honda", "BMW", "Ford", "Mazda" };
        // thay đổi phần tử đầu tiên của mảng cars
        cars[0] = "Morning";
        // hiển thị phần tử đầu tiên của mảng cars
        System.out.println("Phần tử đầu tiên: " + cars[0]);
    }
}

```

Kết quả:

Phần tử đầu tiên: Morning

2.2.14.6. Độ dài mảng trong Java

Để biết có bao nhiêu phần tử một mảng, sử dụng thuộc tính **length**:

Ví dụ:

```

package vn.plpsoft.array;

public class DodaiArray1 {
    public static void main(String[] args) {
        String[] cars = { "Honda", "BMW", "Ford", "Mazda" };
        System.out.println("Độ dài của mảng cars là: " + cars.length);
    }
}

```

Kết quả:

4

2.2.14.7. Duyệt các phần tử của mảng trong Java

Có 2 cách để duyệt các phần tử của mảng:

- Sử dụng vòng lặp for
- Sử dụng foreach

2.2.14.7.1. Sử dụng vòng lặp for

Bạn có thể lặp qua các phần tử mảng bằng vòng lặp for và sử dụng thuộc tính length để chỉ định số lần vòng lặp sẽ chạy. Ví dụ:

```
package vn.plpsoft.array;

public class DuyetArray1 {
    public static void main(String[] args) {
        String[] cars = { "Honda", "BMW", "Ford", "Mazda" };
        for (int i = 0; i < cars.length; i++) {
            System.out.println(cars[i]);
        }
    }
}
```

Kết quả:

```
Honda
BMW
Ford
Mazda
```

2.2.14.7.2. Sử dụng foreach

```
package vn.plpsoft.array;

public class DuyetArray2 {
    public static void main(String[] args) {
        String[] cars = { "Honda", "BMW", "Ford", "Mazda" };
        for (String car : cars) {
            System.out.println(car);
        }
    }
}
```

Kết quả:

```
Honda
BMW
Ford
Mazda
```

2.2.14.7.3. So sánh for với foreach

Nếu bạn so sánh vòng lặp for và vòng lặp foreach, bạn sẽ thấy rằng phương thức foreach dễ viết hơn, nó không yêu cầu bộ đếm (sử dụng thuộc tính length) và nó dễ đọc hơn.

2.2.14.8. Sắp xếp mảng

Có nhiều phương thức mảng có sẵn, ví dụ Sort(), sắp xếp một mảng theo thứ tự bảng chữ cái hoặc theo thứ tự tăng dần, ví dụ:

```
package vn.plpsoft.array;

import java.util.Arrays;

public class DuyetArray2 {

    public static void main(String[] args) {
        String[] cars = { "Honda", "BMW", "Ford", "Mazda" };
        // sap xep mangr cars theo thu tu tang dan
        Arrays.sort(cars);
        System.out.println("Mảng cars sau khi được sắp xếp:");
        for (String car : cars) {
            System.out.println(car);
        }
    }
}
```

Kết quả:

```
Mảng cars sau khi được sắp xếp:
BMW
Ford
Honda
Mazda
```

2.2.14.9. Các cách khác để tạo một mảng trong Java

Trong Java, có nhiều cách khác nhau để tạo một mảng:

```
// tạo một mảng có độ dài 4, thêm các phần tử sau tạo
String[] cars1 = new String[4];
```

```
// tạo một mảng không cần chỉ định số phần tử cụ thể
String[] cars2 = new String[] { "Honda", "BMW", "Ford", "Mazda" };
```

```
// tạo một mảng không cần chỉ định số phần tử cụ thể
// và không cần dùng từ khóa new
String[] cars3 = { "Honda", "BMW", "Ford", "Mazda" };
```

Bạn có thể chọn bất kỳ cách khởi tạo nào. Trong hướng dẫn của chúng tôi, chúng tôi sẽ thường sử dụng tùy chọn cuối cùng, vì nó nhanh hơn và dễ đọc hơn.

Một trường hợp khác, nếu bạn khai báo một mảng và khởi tạo nó sau, bạn phải sử dụng từ khóa new:

```
// khai báo một mảng
String[] cars;

// thêm giá trị, sử dụng từ khóa new
cars = new String[] { "Honda", "BMW", "Ford" };

// nếu không dùng new sẽ bị lỗi
cars = {"Honda", "BMW", "Ford"};
```

2.2.14.10. Truyền mảng vào phương thức trong java

Chúng ta có thể truyền mảng vào phương thức, điều này giúp tái sử dụng code logic để xử lý mảng bất kỳ. Ví dụ:

```
public class TestArray2 {
    static void min(int arr[]) {
        int min = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if (min > arr[i]) {
                min = arr[i];
            }
        }
        System.out.println(min);
    }

    public static void main(String args[]) {
```

```

        int a[] = { 33, 3, 4, 5 };
        min(a); // truyền mảng tới phương thức
    }
}

```

Kết quả:

3

2.2.14.11. Sao chép một mảng trong java

Chúng ta có thể sao chép một mảng tới mảng khác bởi phương thức arraycopy của lớp System. Cú pháp của phương thức arraycopy:

```

public static void arraycopy(
Object src, int srcPos, Object dest, int destPos, int length
)

```

Ví dụ về sao chép mảng trong java

```

public class TestArrayCopyDemo {
    public static void main(String[] args) {
        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f',
                           'f', 'e', 'i', 'n', 'a', 't', 'e', 'd' };
        char[] copyTo = new char[7];

        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo));
    }
}

```

Kết quả:

caffein

2.2.14.12. Mảng hai chiều và đa chiều trong Java

Trong TH này, dữ liệu được lưu trữ theo hàng và cột theo chỉ mục (hay còn gọi là dạng ma trận).

2.2.14.12.1. Khai báo mảng đa chiều trong java

```
dataType[][] arr;
dataType [] [] arr;
dataType arr[][];
dataType [] arr[];
```

2.2.14.12.2. Ví dụ về khởi tạo và gán giá trị cho mảng đa chiều trong java

```
int[][] arr=new int[3][3]; // 3 hàng 3 cột
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

2.2.14.12.3. Ví dụ khai báo và khởi tạo mảng đa chiều trong java

```
public class TestArray3 {
    public static void main(String args[]) {

        // khai báo và khởi tạo mảng 2 chiều
        int arr[][] = { { 1, 2, 3 }, { 2, 4, 5 }, { 4, 4, 5 } };

        // in mảng 2 chiều ra màn hình
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(arr[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

```
    }  
}
```

Kết quả:

```
1 2 3  
2 4 5  
4 4 5
```

2.2.14.12.4. Ví dụ cộng 2 ma trận (mảng đa chiều) trong Java

```
package vn.plpsoft.array;  
  
public class CongMaTran {  
    public static void main(String[] args) {  
        // tao hai ma tran  
        int a[][] = { { 1, 3, 4 }, { 3, 4, 5 } };  
        int b[][] = { { 1, 3, 4 }, { 3, 4, 5 } };  
  
        // tao ma tran khac de luu giu ket qua phep cong hai ma tran  
        int c[][] = new int[2][3];  
  
        // cong va in tong hai ma tran  
        for (int i = 0; i < 2; i++) {  
            for (int j = 0; j < 3; j++) {  
                c[i][j] = a[i][j] + b[i][j];  
                System.out.print(c[i][j] + " ");  
            }  
            System.out.println();  
        }  
    }  
}
```

Kết quả:

```
2 6 8  
6 8 10
```

2.2.14.13. Lợi thế và hạn chế của mảng trong java

➤ Lợi thế

- Tối ưu code:** Mảng giúp lấy hoặc sắp xếp dữ liệu dễ dàng.
- Truy cập ngẫu nhiên:** Chúng ta có thể lấy dữ liệu được lưu trữ ở bất kỳ vị trí nào thông qua chỉ số.

➤ Hạn chế

- Giới hạn kích thước:** Chúng ta phải khởi tạo kích thước của mảng trong java trước khi sử dụng. Không thể thay đổi được kích thước của nó lúc runtime. Collection framework được sử dụng trong java để giải quyết vấn đề này.

2.2.14.14 Giới thiệu Lớp Array trong Java

Lớp `java.util.Arrays` chứa nhiều phương thức static đa dạng để xếp thứ tự và tìm kiếm các mảng, so sánh các mảng và điền các phần tử vào mảng.

STT	Phương thức và Miêu tả
1	public static int binarySearch(Object[] a, Object key) Tìm kiếm mảng của Object (byte, int, double, ...) đã cho với giá trị đã xác định bởi sử dụng thuật toán tìm kiếm nhị phân. Mảng này phải được xếp thứ tự trước khi gọi phương thức này. Nó trả về chỉ mục của từ khóa tìm kiếm, nếu nó nằm trong danh sách, nếu không thì, bằng (-điểm chèn + 1)).
2	public static boolean equals(long[] a, long[] a2) Trả về true nếu hai mảng long đã cho là cân bằng nhau. Hai mảng này được cho là cân bằng nếu cả hai mảng chứa cùng số lượng phần tử, và tất cả các cặp phần tử tương ứng của hai mảng là cân bằng. Phương thức tương tự có thể được sử dụng bởi tất cả kiểu dữ liệu gốc khác (byte, short, int, ...).
3	public static void fill(int[] a, int val) Gán giá trị int đã cho tới mỗi phần tử của mảng int đã cho. Phương thức tương tự có thể được sử dụng bởi tất cả kiểu dữ liệu gốc khác (byte, short, int, ...).
4	public static void sort(Object[] a) Xếp thứ tự mảng các đối tượng đã cho theo thứ tự tăng dần, theo thứ tự tự nhiên của các phần tử. Phương thức tương tự có thể được sử dụng bởi tất cả kiểu dữ liệu gốc khác (byte, short, int, ...).

2.2.14.14.1. Khai báo mảng trên lớp Array trong java

Như bạn biết **mảng trong java** là một đối tượng chứa các phần tử có kiểu dữ liệu giống nhau. Mảng là một cấu trúc dữ liệu nơi lưu trữ các phần tử giống nhau. Với mảng trong java chúng ta chỉ có thể lưu trữ một tập các phần tử có số lượng phần tử cố định.

Trong bài này, **chúng ta sẽ học:**

- Ví dụ khai báo mảng trong java.
 - Ví dụ nhập mảng từ bàn phím trong java.
-

2.2.14.14.2. Ví dụ khai báo mảng trong java

Trong java, bạn có thể khai báo mảng một chiều, hai chiều, ... n chiều.

a. Khai báo mảng một chiều

Dưới đây là ví dụ khai báo mảng một chiều các số nguyên trong java:

```
1 int[] a = null;  
2 int b[] = null;
```

Ví dụ về khởi tạo mảng một chiều các số nguyên trong java:

```
// khởi tạo mảng mặc định  
int[] c = {1, 2, 3, 4, 5};  
// khởi tạo mảng bằng vòng lặp for  
int[] b = new int[10];  
for (int i = 0; i < 10; i++) {  
    b[i] = i;  
}
```

b. Khai báo mảng hai chiều

Dưới đây là ví dụ khai báo mảng hai chiều các số nguyên trong java:

```
int[][] a = null;
```

```
int b[][] = null;
int c[][] = new int[4][];
```

Ví dụ về khởi tạo mảng hai chiều các số nguyên trong java:

```
// khai báo và khởi tạo mảng 2 chiều nặc danh
int[][] c = {{1, 2}, {3, 4}, {3, 4}, {3, 4}};
// khởi tạo mảng 2 chiều bằng vòng lặp for
int[][] b = new int[4][2];
for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 2; j++) {
        b[i][j] = i;
    }
}
```

2.2.14.14.3. Ví dụ nhập mảng từ bàn phím trong java

Ví dụ 1: khai báo và nhập mảng một chiều từ bàn phím trong java:

```
package vn.plpsoft.array;

import java.util.Scanner;

public class ArrayExample1 {
    public static Scanner scanner = new Scanner(System.in);

    /**
     * main
     *
     * @param args
     */
    public static void main(String[] args) {
        System.out.print("Nhập số phần tử của mảng: ");
        int n = scanner.nextInt();
        // khởi tạo mảng arr
```

```
int[] arr = new int[n];
System.out.print("Nhập các phần tử của mảng: \n");
for (int i = 0; i < n; i++) {
    System.out.printf("a[%d] = ", i);
    arr[i] = scanner.nextInt();
}
System.out.print("Các phần tử của mảng: ");
show(arr);
}

/**
 * in các phần tử của mảng ra màn hình
 *
 * @param arr: mảng các số nguyên
 * @param n: số phần tử của mảng
 */
public static void show(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        System.out.print(arr[i] + " ");
    }
}
```

Kết quả:

Nhập số phần tử của mảng: 10

Nhập các phần tử của mảng:

a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 5
a[5] = 6
a[6] = 7
a[7] = 8
a[8] = 9

```
a[9] = 10
```

```
Các phần tử của mảng: 1 2 3 4 5 6 7 8 9 10
```

Ví dụ 2: khai báo và nhập mảng hai chiều từ bàn phím trong java:

```
package vn.plpsoft.array;

import java.util.Scanner;

public class ArrayExample2 {

    public static Scanner scanner = new Scanner(System.in);

    /**
     * main
     *
     * @param args
     */
    public static void main(String[] args) {
        System.out.print("Nhập số hàng của ma trận: ");
        int n = scanner.nextInt();
        System.out.print("Nhập số cột của ma trận: ");
        int m = scanner.nextInt();
        // khởi tạo (ma trận) mảng hai chiều arr
        int[][] arr = new int[n][m];
        System.out.print("Nhập các phần tử của mảng: \n");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                System.out.printf("a[%d][%d] = ", i, j);
                arr[i][j] = scanner.nextInt();
            }
        }
        System.out.println("Các phần tử của (ma trận) mảng hai chiều: ");
        show(arr);
    }

    /**
     * in các phần tử của mảng ra màn hình
    
```

```
*  
 * @param arr: mảng các số nguyên  
 * @param n: số phần tử của mảng  
 */  
  
public static void show(int[][] arr) {  
    for (int i = 0; i < arr.length; i++) {  
        for (int j = 0; j < arr[i].length; j++) {  
            System.out.print(arr[i][j] + " ");  
        }  
        System.out.println();  
    }  
}
```

Kết quả:

Nhập số hàng của ma trận: 4

Nhập số cột của ma trận: 2

Nhập các phần tử của mảng:

a[0][0] = 1

a[0][1] = 1

a[1][0] = 2

a[1][1] = 4

a[2][0] = 6

a[2][1] = 4

a[3][0] = 9

a[3][1] = 10

Các phần tử của (ma trận) mảng hai chiều:

1 1

2 4

6 4

9 10

Các kiểu dữ liệu khác khai báo tương tự.

2.2.14.15. Duyệt mảng trong java

Duyệt mảng trong java là quá trình truy cập từng phần tử của một mảng. Duyệt mảng thường được thực hiện bắt đầu với phần tử đầu tiên và cho đến khi kết thúc mảng. Tuy nhiên, nó cũng có thể di chuyển ngược hoặc bỏ qua các phần tử. Ở đây, chúng ta sẽ tập trung vào việc duyệt một mảng từ đầu đến cuối bằng cách sử dụng hai kỹ thuật khác nhau:

- Sử dụng vòng lặp for đơn giản.
- Sử dụng câu lệnh for-each.

Chúng ta sẽ sử dụng mảng ages, được khai báo như sau, để minh họa cách duyệt qua một mảng:

```
static final int SIZE = 5;
int[] ages = new int[SIZE];
```

Chúng ta sẽ sử dụng mảng trên trong các ví dụ sau:

2.2.14.15.1. Sử dụng vòng lặp đơn giản

Bất kỳ vòng lặp đơn giản nào cũng có thể được sử dụng để duyệt các phần tử của một mảng. Ở đây, chúng ta sẽ sử dụng vòng lặp for và vòng lặp while. Đầu tiên, chúng ta sẽ kiểm tra vòng lặp for. Trong trình duyệt mảng, sử dụng một biến số nguyên bắt đầu từ 0 và tiến tới độ dài của mảng trừ đi một:

```
for(int i = 0; i < ages.length; i++) {
    ages[i] = 5;
}
```

Sử dụng vòng lặp while, lưu ý biến i được khai báo bên ngoài vòng lặp:

```
int i = 0;
while(i < ages.length) {
    ages[i++] = 5;
}
```

Vòng lặp for thường thích hợp hơn vì chúng ta biết độ dài của mảng và nó đơn giản hơn cho vấn đề này. Đối với cả hai ví dụ, chúng ta đã sử dụng thuộc tính độ dài của mảng để điều khiển vòng lặp. Nó thích hợp hơn khi sử dụng một biến cố định có thể

đã được sử dụng để khai báo mảng. Hãy xem xét tình huống định nghĩa lại mảng sau đây:

```
int[] ages = new int[SIZE];
...
for(int i = 0; i < SIZE; i++) {
    ages[i] = 5;
}
// định nghĩa lại mang
int[] ages = new int[DIFFERENT_SIZE];
...
for(int i = 0; i < SIZE; i++) {
    ages[i] = 5;
}
```

Vòng lặp thứ hai sẽ không thực thi đúng bởi vì chúng ta quên thay đổi hằng số SIZE và thậm chí có thể ném ra một ngoại lệ nếu mảng mới có kích thước nhỏ hơn SIZE. Nếu chúng ta sử dụng thuộc tính length thay vào đó, sẽ không có vấn đề gì. Lưu ý rằng vòng lặp for, như được viết, khai báo biến i trong vòng lặp for. Điều này hạn chế quyền truy cập vào biến chỉ cho các câu lệnh đó trong vòng lặp for. Trong ví dụ vòng lặp while, chúng ta đã khai báo i bên ngoài vòng lặp làm cho nó có thể truy cập bên trong và bên ngoài vòng lặp while. Chúng ta có thể viết lại vòng lặp for để sử dụng biến i bên ngoài. Tuy nhiên, cách khai báo như trên được coi là hình thức tốt hơn để hạn chế quyền truy cập vào một biến chỉ cho những câu lệnh cần truy cập.

Ví dụ sử dụng vòng lặp for và while để duyệt mảng trong java

```
package vn.plpsoft;

public class Duyetmang1 {
    private static final int SIZE = 5;

    public static void main(String[] args) {
        int i;
        int[] ages = new int[SIZE];
        // khởi tạo mảng
        for (i = 0; i < SIZE; i++) {
            ages[i] = 5 + i;
        }
    }
}
```

```

    // duyet cac phan tu cua mang bang vong lap for
    System.out.println("Su dung vong lap for: ");
    for (i = 0; i < SIZE; i++) {
        System.out.print(ages[i] + " ");
    }
    // duyet cac phan tu cua mang bang vong lap while
    System.out.println("\nSu dung vong lap while: ");
    i = 0;
    while (i < SIZE) {
        System.out.print(ages[i] + " ");
        i++;
    }
}
}

```

Kết quả:

Su dung vong lap for:

5 6 7 8 9

Su dung vong lap while:

5 6 7 8 9

2.2.14.15.2. Sử dụng câu lệnh for-each

Cú pháp:

```

for (ClassName element : array) {
    // element là phần tử của mảng
}

```

Ví dụ sử dụng lệnh for-each để duyệt mảng trong java

```

package vn.plpsoft;

public class Duyetmang2 {
    private static final int SIZE = 5;
}

```

```

public static void main(String[] args) {
    int i;
    int[] ages = new int[SIZE];
    // khai tao mang
    for (i = 0; i < SIZE; i++) {
        ages[i] = 5 + i;
    }
    // duyet cac phan tu cua mang bang vong lap for
    System.out.println("Su dung vong lap for-each: ");
    for (int age : ages) {
        System.out.print(age + " ");
    }
}
}

```

Kết quả:

Sử dụng vòng lặp for-each:
5 6 7 8 9

2.2.15. Lớp Wrapper trong java

2.2.15.1. Lớp Wrapper trong java

Lớp Wrapper trong java cung cấp cơ chế để chuyển đổi kiểu dữ liệu nguyên thủy thành kiểu đối tượng và từ đối tượng thành kiểu dữ liệu nguyên thủy.

Từ J2SE 5.0, tính năng **autoboxing** và **unboxing** chuyển đổi kiểu dữ liệu nguyên thủy thành kiểu đối tượng và từ đối tượng thành kiểu dữ liệu nguyên thủy một cách tự động. Sự chuyển đổi tự động kiểu dữ liệu nguyên thủy thành kiểu đối tượng được gọi là autoboxing và ngược lại được gọi là unboxing.

Trong java, có 8 lớp Wrapper chúng được thiết kế trong gói java.lang.

Kiểu nguyên thủy	Kiểu Wrapper
boolean	Boolean
char	Character
byte	Byte

short	Short
int	Integer
long	Long
float	Float
double	Double

2.2.15.2. Ví dụ chuyển kiểu dữ liệu nguyên thủy thành kiểu Wrapper

```
public class WrapperExample1 {
    public static void main(String args[]) {
        // Đổi int thành Integer
        int a = 20;
        Integer i = Integer.valueOf(a); // đổi int thành Integer
        Integer j = a; // autoboxing, tự động đổi int thành
        Integer trong nội bộ trình biên dịch

        System.out.println(a + " " + i + " " + j);
    }
}
```

Output:

20 20 20

4.15.3. Ví dụ chuyển kiểu Wrapper thành kiểu dữ liệu nguyên thủy

```
public class WrapperExample2 {
    public static void main(String args[]) {
        // đổi Integer thành int
        Integer a = new Integer(3);
```

```

        int i = a.intValue(); // đổi Integer thành int
        int j = a; // unboxing, tự động đổi Integer thành int trong
nội bộ trình biên dịch

        System.out.println(a + " " + i + " " + j);
    }
}

```

Output:

```
3 3 3
```

2.2.16. Đệ quy trong java

Đệ quy trong java là quá trình trong đó một phương thức gọi lại chính nó một cách liên tiếp. Một phương thức trong java gọi lại chính nó được gọi là phương thức đệ quy.

Sử dụng đệ quy giúp code chặt chẽ hơn nhưng sẽ khó để hiểu hơn.

Cú pháp:

```

returntype methodname() {
    // code
    methodname();
}

```

➤ Ví dụ về đệ quy trong java

Dưới đây là các ví dụ về cách sử dụng đệ quy trong java.

Ví dụ 1: vòng lặp vô tận

```

public class RecursionExample1 {
    static void p() {
        System.out.println("hello");
        p();
    }
}

```

```
public static void main(String[] args) {  
    p();  
}  
}
```

Kết quả:

```
hello  
hello  
...  
Exception in thread "main" java.lang.StackOverflowError
```

Ví dụ 2: vòng lặp có hạn

```
public class RecursionExample2 {  
    static int count = 0;  
  
    static void p() {  
        count++;  
        if (count <= 5) {  
            System.out.println("hello " + count);  
            p();  
        }  
    }  
  
    public static void main(String[] args) {  
        p();  
    }  
}
```

Kết quả:

```
hello 1  
hello 2  
hello 3  
hello 4  
hello 5
```

Ví dụ 3: tính gai thừa

```
public class RecursionExample3 {  
    static int factorial(int n) {  
        if (n == 1)  
            return 1;  
        else  
            return (n * factorial(n - 1));  
    }  
  
    public static void main(String[] args) {  
        System.out.println("Gai thừa của 5 là: " + factorial(5));  
    }  
}
```

Kết quả:

```
Gai thừa của 5 là: 120
```

Chương trình trên hoạt động như sau:

```
factorial(5)  
  factorial(4)  
    factorial(3)  
      factorial(2)  
        factorial(1)  
          return 1  
          return 2*1 = 2  
          return 3*2 = 6  
          return 4*6 = 24  
        return 5*24 = 120
```

Ví dụ 4: dãy số Fibonacci

```

public class RecursionExample4 {
    static int n1 = 0, n2 = 1, n3 = 0;

    static void printFibo(int count) {
        if (count > 0) {
            n3 = n1 + n2;
            n1 = n2;
            n2 = n3;
            System.out.print(" " + n3);
            printFibo(count - 1);
        }
    }

    public static void main(String[] args) {
        int count = 15;
        System.out.print(n1 + " " + n2); // in 0 và 1
        printFibo(count - 2); // n-2 vì 2 số 0 và 1 đã được in ra
    }
}

```

Kết quả:

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

2.2.17. Truyền giá trị và tham chiếu (pass-by-value và pass-by-reference) trong java

Nếu chúng ta gọi một phương thức và truyền một giá trị cho phương thức đó được gọi là **truyền giá trị**. Việc thay đổi giá trị chỉ có hiệu lực trong phương thức được gọi, không có hiệu lực bên ngoài phương thức.

Khi chúng ta gọi một phương thức và truyền một tham chiếu cho phương thức đó được gọi là **truyền tham chiếu**. Việc thay đổi giá trị của biến tham chiếu bên trong phương thức làm thay đổi giá trị gốc của nó.

Hãy xem ví dụ để hiểu rõ hơn.

Ví dụ về việc truyền giá trị (pass by value) trong java

Trong ví dụ này, giá trị data không bị thay đổi sau khi gọi phương thức change()

```
public class Operation1 {  
    int data = 50;  
  
    void change(int data) {  
        data = data + 100;  
    }  
  
    public static void main(String args[]) {  
        Operation1 op = new Operation1();  
  
        System.out.println("Trước khi thay đổi: " + op.data);  
        op.change(500);  
        System.out.println("Sau khi thay đổi: " + op.data);  
    }  
}
```

Output:

```
Trước khi thay đổi: 50
```

```
Sau khi thay đổi: 50
```

Ví dụ về việc truyền tham chiếu (pass by reference) trong java

Trong ví dụ này, giá trị của biến data của đối tượng op bị thay đổi sau khi gọi phương thức change()

```
public class Operation2 {  
    int data = 50;  
  
    void change(Operation2 op) {  
        op.data = op.data + 100;  
    }  
  
    public static void main(String args[]) {  
        Operation2 op = new Operation2();  
    }  
}
```

```

        System.out.println("Trước khi thay đổi: " + op.data);
        op.change(op); // truyền đối tượng
        System.out.println("Sau khi thay đổi: " + op.data);
    }
}

```

Output:

```

Trước khi thay đổi: 50
Sau khi thay đổi: 150

```

2.2.18. Toán tử instanceof trong java

2.2.18.1. Toán tử instanceof trong java

Toán tử instanceof trong java được sử dụng để kiểm tra một đối tượng có phải là thể hiện của một kiểu dữ liệu cụ thể không (lớp, lớp con, interface).

instanceof trong java được gọi là toán tử so sánh kiểu vì nó so sánh thể hiện với kiểu dữ liệu. Nó trả về giá trị boolean là true hoặc false. Nếu bạn dùng toán tử instanceof với bất kỳ biến nào mà có giá trị null, giá trị trả về sẽ là false.

Ví dụ về toán tử instanceof trong java

```

public class Simple1 {
    public static void main(String args[]) {
        Simple1 s = new Simple1();
        System.out.println(s instanceof Simple1); // true
    }
}

```

Output:

```

true

```

Một đối tượng có kiểu của lớp con thì cũng có kiểu của lớp cha. Ví dụ, nếu Dog kế thừa Animal thì đối tượng của Dog có thể tham chiếu đến cả hai lớp Dog và Animal.

Ví dụ khác về toán tử instanceof trong java

```
class Animal { }
```

```
public class Dog extends Animal { // Dog inherits Animal
```

```
public static void main(String args[]) {
    Dog dog = new Dog();
    System.out.println(dog instanceof Animal); // true
}
```

Output:

```
true
```

2.2.18.2. instanceof trong java với biến có giá trị null

Nếu bạn sử dụng toán tử instanceof với biến có kiểu bất kỳ có giá trị null thì giá trị trả về luôn là null. Ví dụ.

```
public class Dog2 {
    public static void main(String args[]) {
        Dog2 d = null;
        System.out.println(d instanceof Dog2); // false
    }
}
```

Output:

```
false
```

2.2.18.3. Downcasting với toán tử instanceof trong java

Khi kiểu của lớp con tham chiếu tới đối tượng của lớp cha được gọi là downcasting. Nếu bạn thực hiện tham chiếu trực tiếp thì trình biên dịch sẽ báo lỗi biên dịch. Nếu bạn thực hiện bằng việc ép kiểu thì lỗi ngoại lệ ClassCastException được ném ra lúc runtime. Nhưng nếu bạn sử dụng toán tử instanceof thì downcasting được.

Tham chiếu trực tiếp:

```
Dog d=new Animal(); //Compilation error
```

Sử dụng ép kiểu dữ liệu:

```
Dog d=(Dog) new Animal();
```

// Compile ok, nhưng lỗi ngoại lệ ClassCastException lúc runtime

Ví dụ downcasting với instanceof

```
class Animal {}  
  
public class Dog3 extends Animal {  
    static void method(Animal a) {  
        if (a instanceof Dog3) {  
            Dog3 d = (Dog3) a; // downcasting  
            System.out.println("downcasting is ok");  
        }  
    }  
  
    public static void main(String[] args) {  
        Animal a = new Dog3();  
        Dog3.method(a);  
    }  
}
```

Output:

```
downcasting is ok
```

Ví dụ downcasting không sử dụng instanceof

```
class Animal {}  
  
public class Dog4 extends Animal {  
    static void method(Animal a) {  
        Dog4 d = (Dog4) a; // downcasting  
        System.out.println("downcasting is ok");  
    }  
  
    public static void main(String[] args) {  
        Animal a = new Dog4();  
        Dog4.method(a);  
    }  
}
```

```

    }
}
```

Output:

```
downcasting is ok
```

Hãy đi sâu vào vấn đề gần hơn chút, thực tế đổi tượng được tham chiếu bởi biến a là một đổi tượng của lớp Dog4. Vì vậy chúng ta thực hiện downcasting nó thì ok. Nhưng nếu code như sau thì chuyện gì sẽ xảy ra?

```

Animal a=new Animal();
Dog.method(a);
// lỗi ngoại lệ ClassCastException
```

Ví dụ sử dụng toán tử instanceof để xác định kiểu phần tử của mảng

```

public class Test6 {
    public static void main(String[] args) {
        Object[] objArray = new Object[4];
        objArray[0] = "hello";
        objArray[1] = 1000L;
        objArray[2] = 20;
        objArray[3] = 'c';

        for (int i = 0; i < objArray.length; i++) {
            if (objArray[i] instanceof Integer) {
                System.out.println("kieu integer: " + objArray[i]);
            } else if (objArray[i] instanceof Long) {
                System.out.println("kieu long: " + objArray[i]);
            } else if (objArray[i] instanceof Character) {
                System.out.println("kieu char: " + objArray[i]);
            } else {
                System.out.println("kieu khac: " + objArray[i]);
            }
        }
    }
}
```

```

    }
}

```

Output:

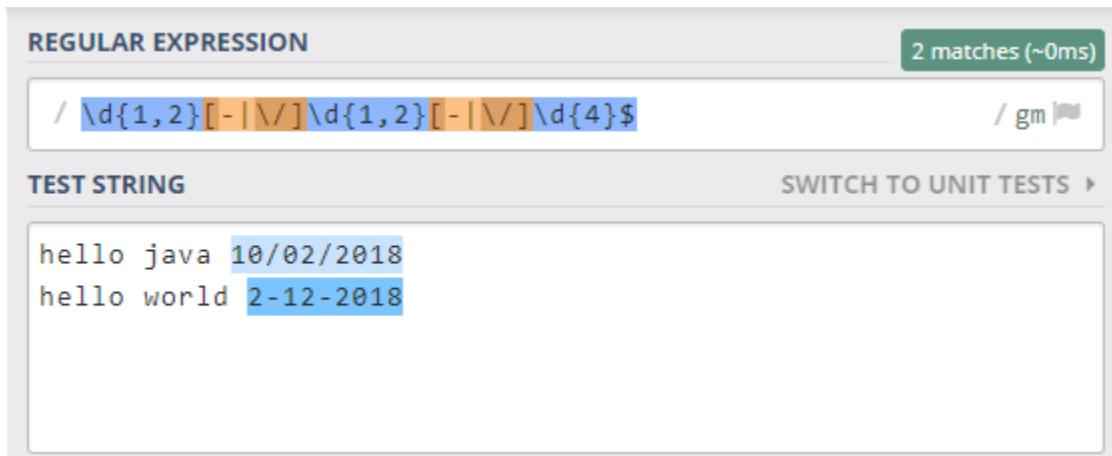
```

kieu khac: hello
kieu long: 1000
kieu integer: 20
kieu char: c

```

2.2.19. Sử dụng Regex trong Java - Java Regex

Java Regex hoặc Regular Expression (biểu thức chính quy) là một API để định nghĩa một mẫu để tìm kiếm hoặc thao tác với chuỗi. Nó được sử dụng rộng rãi để xác định ràng buộc trên các chuỗi như xác thực mật khẩu, email, kiểu dữ liệu datetime, ...



2.2.19.1. Gói java.util.regex

Java Regex API cung cấp 1 interface và 3 lớp trong gói `java.util.regex`.

Lớp `Matcher` và `Pattern` trong `java` cung cấp cơ sở của biểu thức chính quy. Gói `java.util.regex` cung cấp các lớp và giao diện sau cho các biểu thức chính quy.

1. Interface `MatchResult`
2. Lớp `Matcher`
3. Lớp `Pattern`
4. Lớp `PatternSyntaxException`

2.2.19.1. 1. Lớp Matcher

Nó implements interface MatchResult, cung cấp bộ máy xử lý biểu thức chính quy để thao tác với chuỗi ký tự.

No.	Phương thức	Mô tả
1	boolean matches()	kiểm tra xem biểu thức chính quy có khớp với mẫu hay không.
2	boolean find()	tìm biểu thức tiếp theo khớp với mẫu.
3	boolean find(int start)	tìm biểu thức tiếp theo khớp với mẫu từ số bắt đầu đã cho.
4	String group()	trả về chuỗi con phù hợp.
5	int start()	trả về vị trí bắt đầu của chuỗi con phù hợp.
6	int end()	trả về vị trí kết thúc của chuỗi con phù hợp.
7	int groupCount()	trả về tổng số các chuỗi con phù hợp.

2.2.19.1.2. Lớp Pattern

Đây là phiên bản được biên dịch của một biểu thức chính quy. Nó được sử dụng để xác định một mẫu cho bộ máy regex.

No.	Phương thức	Mô tả
1	static Pattern compile(String regex)	biên dịch regex đã cho và trả về thể hiện của Pattern.
2	Matcher matcher(CharSequence input)	tạo một matcher khớp với đầu vào đã cho với mẫu.
3	static boolean matches(String regex, CharSequence input)	Nó biên dịch biểu thức chính quy và tìm kiếm các chuỗi con từ chuỗi input phù hợp với mẫu regex.
4	String[] split(CharSequence input)	chia chuỗi input đã cho thành mảng các kết quả trùng khớp với mẫu đã cho.

5	String pattern()	trả về mẫu regex.
---	------------------	-------------------

2.2.19.2. Ví dụ sử dụng Regex trong Java

1. Ví dụ sử dụng Regex trong Java - tìm kiếm chuỗi con

Ví dụ sau tìm tất cả các chuỗi ngày tháng có định dạng dd-mm-yyyy hoặc dd/mm/yyyy trong chuỗi văn bản text1 và xác minh xem chuỗi text2 và text3 có định dạng ngày tháng hay không.

Định nghĩa regex:

\d{1,2} [-/] \d{1,2} [-/] \d{4}

\d{1,2}: nghĩa là một số có 1 hoặc 2 chữ số (ngày và tháng).

[-/]: nghĩa là ký tự - hoặc /.

\d{4}: nghĩa là một số có 4 chữ số (năm).

File: RegexExample1.java

```
package vn.plpsoft;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexExample1 {
    public static void main(String[] args) {
        String text1 = "Hello java regex 2-12-2018, hello world
12/12/2018";
        Pattern pattern = Pattern.compile("\d{1,2}[-/]\d{1,2}[-/]\d{4}");
        Matcher matcher = pattern.matcher(text1);

        System.out.println("Ngày tháng trong chuỗi text1: " + text1);
        while (matcher.find()) {
            System.out.println(text1.substring(matcher.start(),
                    matcher.end()));
        }
    }
}
```

```

        String text2 = "2/12/2018";
        String text3 = "12/12/aaaa";
        pattern = Pattern.compile("^\\d{1,2}[-/]\\d{1,2}[-/]\\d{4}$");
        System.out.println("\nChuỗi " + text2 + " có định dạng ngày tháng: "
    tháng: "
        + pattern.matcher(text2).matches());
    }

}

System.out.println("Chuỗi " + text3 + " có định dạng ngày tháng: "
"
        + pattern.matcher(text3).matches());
}

```

Kết quả:

Ngày tháng trong chuỗi text1: Hello java regex 2-12-2018, hello world 12/12/2018

2-12-2018

12/12/2018

Chuỗi 2/12/2018 có định dạng ngày tháng: true

Chuỗi 12/12/aaaa có định dạng ngày tháng: false

2. Ví dụ sử dụng Regex trong Java - xác thực email

Định nghĩa email:

- Bắt đầu bằng chữ cái.
- Chỉ chứa chữ cái, chữ số và dấu gạch ngang (-).
- Chứa một ký tự @, sau @ là tên miền.
- Tên miền có thể là domain.xxx.yyy hoặc domain.xxx. Trong đó xxx và yyy là các chữ cái và có độ dài từ 2 trở lên.

Định nghĩa regex:

`^ [a-zA-Z] [\w-]+@ ([\w]+\.\[\w\]+\|\[\w\]+\|\.\[\w\]{2,}\.\[\w\]{2,})$`

File: RegexExample2.java

```

package vn.plpsoft;
import java.util.regex.Pattern;

```

```
public class RegexExample2 {  
    public static void main(String[] args) {  
        String EMAIL_PATTERN =  
            "^ [a-zA-Z] [\w-]  
] + @ ([\w]+ \. [\w]+ | [\w]+ \. [\w]{2,} \. [\w]{2,} ) $";  
  
        String email1 = "test1@gmail.com.vn";  
        String email2 = "123test@gmail.com.vn";  
        String email3 = "test2@gmail.com";  
        String email4 = "test3-1@gmail.com";  
        String email5 = "test4@@gmail.com";  
        String email6 = "test5@domain.com";  
        String email7 = "test6@gmail";  
  
        System.out.println(email1 + ": " +  
Pattern.matches(EMAIL_PATTERN, email1));  
        System.out.println(email2 + ": " +  
Pattern.matches(EMAIL_PATTERN, email2));  
        System.out.println(email3 + ": " +  
Pattern.matches(EMAIL_PATTERN, email3));  
        System.out.println(email4 + ": " +  
Pattern.matches(EMAIL_PATTERN, email4));  
        System.out.println(email5 + ": " +  
Pattern.matches(EMAIL_PATTERN, email5));  
        System.out.println(email6 + ": " +  
Pattern.matches(EMAIL_PATTERN, email6));  
        System.out.println(email7 + ": " +  
Pattern.matches(EMAIL_PATTERN, email7));  
    }  
}
```

Kết quả:

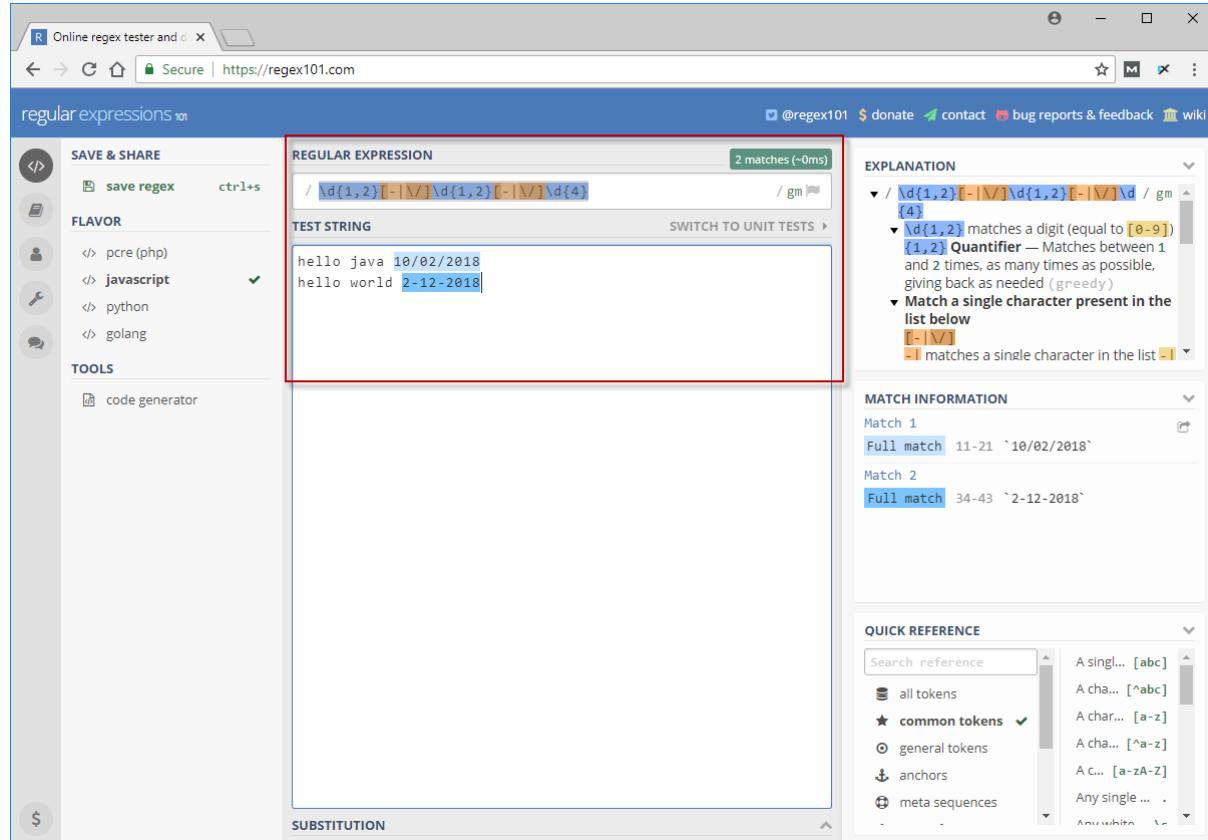
```
test1@gmail.com.vn: true  
123test@gmail.com.vn: false  
test2@gmail.com: true  
test3-1@gmail.com: true  
test4@@gmail.com: false
```

```
test5@domain.com: true
test6@gmail.com: false
```

2.2.19.3. Test Regex Online

Bạn có thể test regex online bằng cách sử dụng trang web <https://regex101.com/>

Ví dụ: tìm tất cả các chuỗi ngày tháng có định dạng dd-mm-yyyy hoặc dd/mm/yyyy trong chuỗi văn bản. Ta sử dụng regex = "\d{1,2}[-/]\d{1,2}[-/]\d{4}"



Cú pháp của biểu thức chính quy sẽ được giải thích bên dưới.

2.2.19.4. Cú pháp Regex trong Java

Với các cú pháp và ví dụ sau bạn có thể kiểm tra kết quả bằng cách sử dụng trang web <https://regex101.com/>

1. Các lớp ký tự Regex

Regex	Mô tả
[...]	trả về một ký tự phù hợp

[abc]	a, b, hoặc c
[^abc]	Bất kỳ ký tự nào ngoại trừ a, b, hoặc c
[a-zA-Z]	a tới z hoặc A tới Z
[a-d[m-p]]	a tới d, hoặc m tới p: [a-dm-p]
[a-z&&[def]]	d, e, hoặc f
[a-z&&[^bc]]	a tới z, ngoại trừ b và c: [ad-z]
[a-z&&[^m-p]]	a tới z, ngoại trừ m tới p: [a-lq-z]
[0-9]	0 tới 9

Ví dụ:

```
package vn.plpsoft;

import java.util.regex.Pattern;

public class RegexExample2 {
    public static void main(String args[]) {
        System.out.println(Pattern.matches("[a-z&&[^bc]]", "a"));
        System.out.println(Pattern.matches("[a-z&&[^bc]]", "b"));
        System.out.println(Pattern.matches("[[a-z&&[^m-p]]]", "a"));
        System.out.println(Pattern.matches("[abc]", "c"));
        System.out.println(Pattern.matches("[abc]", "abc"));
        System.out.println(Pattern.matches("[0-9]", "8"));
    }
}
```

Kết quả:

true

false

true

true

false

true

2. Số lượng ký tự trong Regex

Số lượng trong Regex chỉ định số lượng xảy ra của một ký tự.

Regex	Mô tả	Pattern	Ví dụ
X?	X xảy ra một hoặc không lần	hellos?	hello, hellos, helloss
X+	X xảy ra một hoặc nhiều lần	Version \w-\w+	Version A-b1_1
X*	X xảy ra không hoặc nhiều lần	A*B*C*	AAACC
X{n}	X chỉ xảy ra n lần	\d{4}	2018, 20189, 201
X{n,}	X xảy ra n hoặc nhiều lần	\d{4,}	2018, 20189, 201
X{y,z}	X xảy ra ít nhất y lần nhưng nhỏ hơn z lần	\d{2,3}	2018, 201

3. Ký tự đặc biệt trong Regex

Bảng sau dây liệt kê một số ký tự đặc biệt trong regex.

Regex	Mô tả
.	Bất kỳ ký tự nào
^	Có 2 cách sử dụng. 1. Đánh dấu bắt đầu của một dòng, một chuỗi. 2. Nếu sử dụng trong dấu [...] thì nó có nghĩa là phủ định.
\$	Đánh dấu Kết thúc của một dòng
\d	Bất kỳ chữ số nào, viết tắt của [0-9]

\D	Bất kỳ ký tự nào không phải chữ số, viết tắt của [^0-9]
\s	Bất kỳ ký tự trống nào (như dấu cách, tab, xuống dòng, ...), viết tắt của [\t\n\x0B\f\r]
\S	Bất kỳ ký tự trống nào không phải ký tự trống, viết tắt của [^\s]
\w	Bất kỳ ký tự chữ nào (chữ cái và chữ số), viết tắt của [a-zA-Z_0-9]
\W	Bất kỳ ký tự nào không phải chữ cái và chữ số, viết tắt của [^\w]
\b	Ranh giới của một từ
\B	Không phải ranh giới của một từ

4. Ký tự logic trong Regex

Bảng sau liệt kê một số ký tự logic trong Regex:

Regex	Mô tả	Pattern	Ví dụ
	Hoặc	22 33	33
(...)	Group các ký tự và chụp lại	A(nt pple)	Ant, Apple
\1	Nội dung của Group 1	r(\w)g\1x	regex
\2	Nội dung của Group 2	(\d\d)\+(\d\d)=\2+\1	12+65=65+12
(?: ...)	Group không được chụp lại, bạn không thể sử dụng \1	A(?:nt pple)	Ant, Apple

CHƯƠNG 3: Xử lý ngoại lệ trong java

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

Trong bài này, chúng ta sẽ tìm hiểu về ngoại lệ (Exception) trong java, các kiểu ngoại lệ và sự khác biệt giữa các ngoại lệ **checked** và **unchecked**.

Nội dung chính

- Exception là gì?
- Exception Handling là gì?
- Lợi thế của Exception Handling trong java
- Hệ thống cấp bậc của các lớp ngoại lệ trong Java
- Các kiểu của ngoại lệ
- Sự khác nhau giữa các ngoại lệ checked và unchecked
 - 1. Checked Exception
 - 2. Unchecked Exception
 - 3. Error
- Các kịch bản phổ biến nơi ngoại lệ có thể xảy ra
 - 1. Kịch bản ArithmeticException xảy ra
 - 2. Kịch bản NullPointerException xảy ra
 - 3. Kịch bản NumberFormatException xảy ra
 - Kịch bản ArrayIndexOutOfBoundsException xảy ra
- Các từ khóa xử lý ngoại lệ trong java

3.1. Exception là gì?

Theo từ điển: Exception (ngoại lệ) là một tình trạng bất thường.

Trong java, ngoại lệ là một sự kiện làm gián đoạn luồng bình thường của chương trình. Nó là một đối tượng được ném ra tại runtime.

3.2. Exception Handling là gì?

Exception Handling (xử lý ngoại lệ) là một cơ chế xử lý các lỗi runtime như ClassNotFoundException, IOException, SQLException, RemoteException, vv

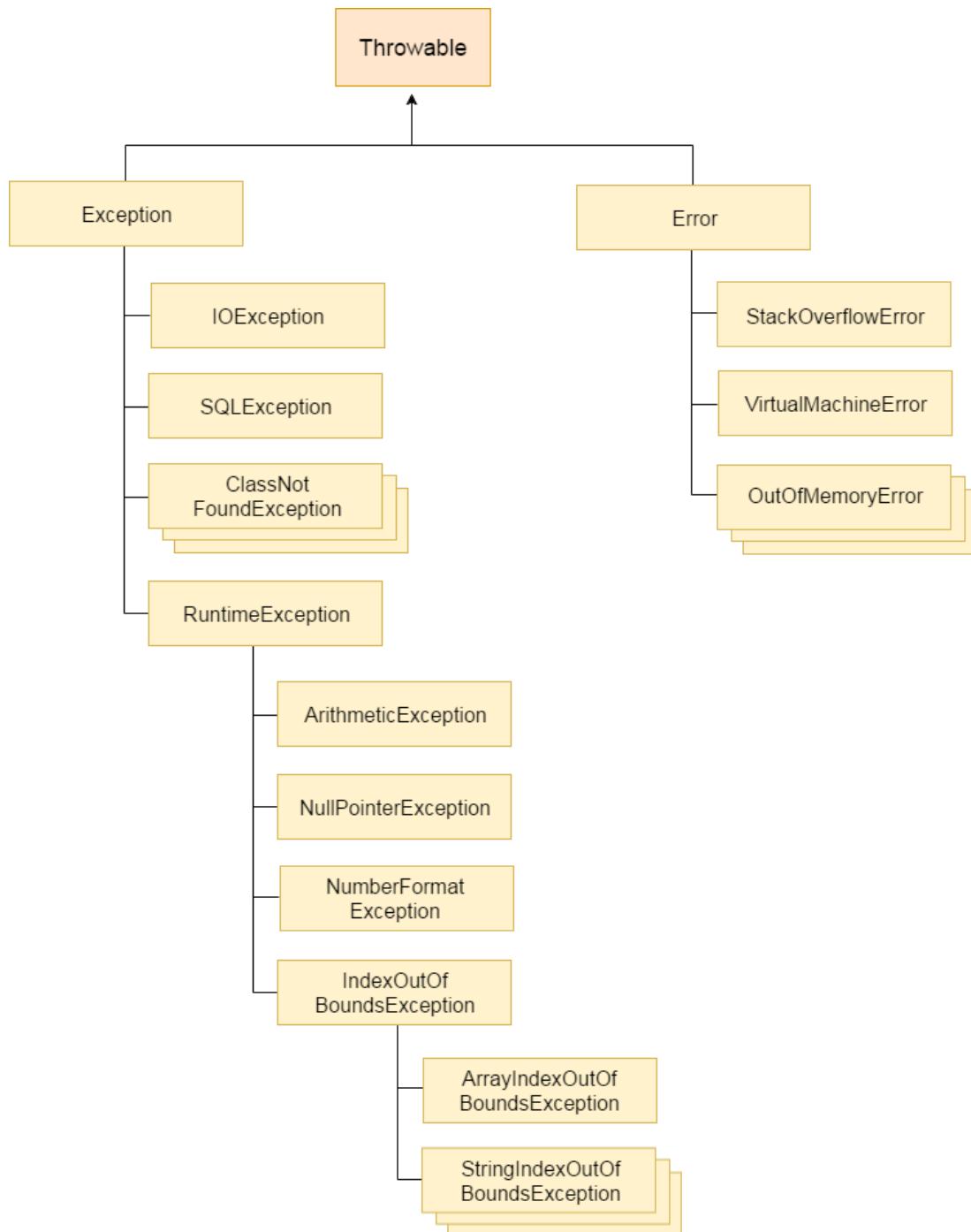
3.3. Lợi thế của Exception Handling trong java

Lợi thế cốt lõi của việc xử lý ngoại lệ là duy trì luồng bình thường của ứng dụng. Ngoại lệ thường làm gián đoạn luồng bình thường của ứng dụng đó là lý do tại sao chúng ta sử dụng xử lý ngoại lệ. Hãy xem xét kịch bản sau:

```
1      statement 1;
2      statement 2;
3      statement 3;
4      statement 4;
5      statement 5; //ngoại lệ xảy ra
6      statement 6;
7      statement 7;
8      statement 8;
9      statement 9;
10     statement 10;
```

Giả sử có 10 câu lệnh trong chương trình của bạn và xảy ra trường hợp ngoại lệ ở câu lệnh 5, phần còn lại của chương trình sẽ không được thực thi, nghĩa là câu lệnh 6 đến 10 sẽ không chạy. Nếu chúng ta thực hiện xử lý ngoại lệ, phần còn lại của câu lệnh sẽ được thực hiện. Đó là lý do tại sao chúng ta sử dụng xử lý ngoại lệ trong java.

3.4. Hệ thống cấp bậc của các lớp ngoại lệ trong Java



3.5. Các kiểu của ngoại lệ

Có hai loại ngoại lệ chính là: **checked** và **unchecked**. Còn Sun Microsystem nói rằng có ba loại ngoại lệ:

1. Checked Exception
2. Unchecked Exception
3. Error

3.6. Sự khác nhau giữa các ngoại lệ checked và unchecked

3.6. 1. Checked Exception

Các lớp extends từ lớp Throwable ngoại trừ RuntimeException và Error được gọi là checked exception, ví dụ như Exception, SQLException vv. Các checked exception được kiểm tra tại compile-time.

Ví dụ: ta có hàm testCheckedException() ném ra một ngoại lệ được extends từ lớp Exception, nên khi nó được gọi ra trong hàm main, trình biên dịch sẽ check(báo lỗi) tức là trình biên dịch sẽ nói ràng hàm này có ném lỗi ra đấy phải xử lý lỗi đi.

```

1 package learn.exception;
2
3 class MyCheckedException extends Exception { }
4 ...
5 public MyCheckedException(String msg) {
6     super(msg);
7 }
8
9
10 public class CheckedExceptionDemo {
11
12     public static void main(String[] args) {
13         CheckedExceptionDemo.testCheckedException();
14     }
15
16     public static void testCheckedException() throws MyCheckedException {
17         System.out.println("Checked exception demo");
18     }
19 }

```

3.6. 2. Unchecked Exception

Các lớp extends từ RuntimeException được gọi là unchecked exception, ví dụ: ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException,... Các ngoại lệ unchecked không được kiểm tra tại compile-time mà chúng được kiểm tra tại runtime.

Ví dụ: ta có hàm testUncheckedException() ném ra một ngoại lệ được extends từ lớp RuntimeException, nên khi nó được gọi ra trong hàm main, trình biên dịch sẽ không check (không báo lỗi). Mà khi chạy nếu có lỗi sẽ bắn ra tại runtime.

```

1 package learn.exception;
2
3 class MyUncheckedException extends RuntimeException { }
4
5 public MyUncheckedException(String msg) {
6     super(msg);
7 }
8
9
10 public class UncheckedExceptionDemo {
11
12     public static void main(String[] args) {
13         UncheckedExceptionDemo.testUncheckException();
14     }
15
16     public static void testUncheckException() throws MyUncheckedException {
17         System.out.println("Unchecked exception demo");
18     }
19 }

```

Không được check tại compile-time
mà được check tại runtime

3.6. 3. Error

Error là lỗi không thể cứu chữa được, ví dụ: OutOfMemoryError, VirtualMachineError, AssertionError, vv

3.7. Các kịch bản phổ biến nơi ngoại lệ có thể xảy ra

Có một số kịch bản mà ngoại lệ unchecked có thể xảy ra. Như các trường hợp sau:

1. Kịch bản ArithmeticException xảy ra

Nếu chúng ta chia bất kỳ số nào cho số 0, xảy ra ngoại lệ ArithmeticException.

```
int a=50/0;//ArithmaticException
```

2. Kịch bản NullPointerException xảy ra

Nếu chúng ta có bất kỳ biến nào có giá trị null , thực hiện bất kỳ hoạt động nào bởi biến đó sẽ xảy ra ngoại lệ NullPointerException.

```
String s=null;
System.out.println(s.length());//NullPointerException
```

3. Kịch bản NumberFormatException xảy ra

Sự định dạng sai của bất kỳ giá trị nào, có thể xảy ra NumberFormatException. Giả sử ta có một biến String có giá trị là các ký tự, chuyển đổi biến này thành số sẽ xảy ra NumberFormatException

```
String s="abc";
int i=Integer.parseInt(s); //NumberFormatException
```

Kịch bản ArrayIndexOutOfBoundsException xảy ra

Nếu bạn chèn bất kỳ giá trị nào vào index sai, sẽ xảy ra ngoại lệ ArrayIndexOutOfBoundsException như thể hiện dưới đây:

```
int a []=new int [5];
a[10]=50; //ArrayIndexOutOfBoundsException
```

3.8. Các từ khóa xử lý ngoại lệ trong java

Có 5 từ khóa được sử dụng để xử lý ngoại lệ trong java, đó là:

1. try
2. catch
3. finally
4. throw
5. throws

Chúng ta sẽ học cách sử dụng các từ khóa này trong các bài tiếp theo...

3.9 Khối lệnh try-catch trong java

3.9.1. Khối lệnh try trong java

Khối lệnh try trong java được sử dụng để chứa một đoạn code có thể xảy ra một ngoại lệ. Nó phải được khai báo trong phương thức.

Sau một khối lệnh try bạn phải khai báo khối lệnh catch hoặc finally hoặc cả hai.

1. Cú pháp của khối lệnh try-catch trong java

```
try {
    // code có thể ném ra ngoại lệ
} catch(Exception_class_Name ref) {
    // code xử lý ngoại lệ
}
```

2. Cú pháp của khối lệnh try-finally trong java

```
try {
```

```
// code có thể ném ra ngoại lệ
} finally {
    // code trong khối này luôn được thực thi
}
```

3.9.2. Khối lệnh catch trong java

Khối catch trong java được sử dụng để xử lý các Exception. Nó phải được sử dụng sau khối try.

Bạn có thể sử dụng nhiều khối catch với một khối try duy nhất.

3.9.3. Vấn đề không có ngoại lệ xử lý

```
public class TestTryCatch1 {
    public static void main(String args[]) {
        int data = 50 / 0; // ném ra ngoại lệ ở đây
        System.out.println("rest of the code...");
    }
}
```

Output:

```
Exception in thread "main" java.lang.ArithmetricException: / by zero
at vn.tpv.exception1.TestTryCatch1.main(TestTryCatch1.java:5)
```

Trong ví dụ trên, phần còn lại của code không được thực thi (dòng chữ "rest of the code..." không được in ra màn hình). Tất cả các lệnh không được thực thi sau khi xảy ra ngoại lệ.

3.9.4. Giải quyết bằng xử lý ngoại lệ

```
public class TestTryCatch2 {
    public static void main(String args[]) {
        try {
            int data = 50 / 0;
        } catch (ArithmetricException e) {
            System.out.println(e);
        }
    }
}
```

```

        System.out.println("rest of the code...");  

    }  

}

```

Output:

```

java.lang.ArithmetricException: / by zero  

rest of the code...

```

Trong ví dụ này, phần còn lại của code được thực thi nghĩa là dòng chữ "rest of the code..." được in ra màn hình.

3.10. Đa khối lệnh catch trong java

Nếu bạn phải thực hiện các tác vụ khác nhau mà ở đó có thể xảy ra các ngoại lệ khác nhau, hãy sử dụng đa khối lệnh catch trong java.

Hãy xem ví dụ sau về việc sử dụng nhiều khối lệnh catch trong java

```

public class TestMultipleCatchBlock {  

    public static void main(String args[]) {  

        try {  

            int a[] = new int[5];  

            a[5] = 30 / 0;  

        } catch (ArithmetricException e) {  

            System.out.println("task1 is completed");  

        } catch (ArrayIndexOutOfBoundsException e) {  

            System.out.println("task 2 completed");  

        } catch (Exception e) {  

            System.out.println("common task completed");  

        }  

        System.out.println("rest of the code...");  

    }  

}

```

Output:

```

task1 is completed  

rest of the code...

```

Quy tắc: Vào một thời điểm chỉ xảy ra một ngoại lệ và tại một thời điểm chỉ có một khối catch được thực thi.

Quy tắc: Tất cả các khối catch phải được sắp xếp từ cụ thể nhất đến chung nhất, tức là phải khai báo khối lệnh catch để xử lý lỗi ArithmeticException trước khi khai báo catch để xử lý lỗi Exception.

Ví dụ:

```
public class TestMultipleCatchBlock1 {
    public static void main(String args[]) {
        try {
            int a[] = new int[5];
            a[5] = 30 / 0;
        } catch (Exception e) {
            System.out.println("common task completed");
        } catch (ArithmetricException e) {
            System.out.println("task1 is completed");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("task2 is completed");
        }
        System.out.println("rest of the code...");
    }
}
```

Output:

Compile-time error

Chương trình trên bị lỗi tại compile-time là vì khi có ngoại lệ xảy ra thì các khối lệnh catch (ArithmetricException e) và catch (ArrayIndexOutOfBoundsException e) không bao giờ được thực thi, do khối catch (Exception e) đã bắt tất cả các ngoại lệ rồi.

3.11. Khối lệnh try lồng nhau trong java

Khối try trong một khối try được gọi là khối try lồng nhau trong java.

3.11.1. Tại sao phải sử dụng khối try lồng nhau

Đôi khi một tình huống có thể phát sinh khi một phần của một khối lệnh có thể xảy ra một lỗi và toàn bộ khối lệnh chính nó có thể xảy ra một lỗi khác. Trong những trường hợp như vậy, trình xử lý ngoại lệ phải được lồng nhau.

```
try {
    statement 1;
    statement 2;
    try {
        statement 1;
        statement 2;
    }
    catch(Exception e)  {

    }
}
catch(Exception e)  {

}
```

3.11.2. Ví dụ về khối try lồng nhau trong java

Hay xem ví dụ đơn giản sau về khối lệnh try lồng nhau.

```
public class TestException {
    public static void main(String args[]) {
        try {
            try {
                System.out.println("Thuc hien phep chia");
                int b = 39 / 0;
            } catch (ArithmetricException e) {
                System.out.println(e);
            }
        }
        try {
            int a[] = new int[5];
            a[5] = 4;
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println(e);
        }
    }
}
```

```

        System.out.println("khoi lenh khac");
    } catch (Exception e) {
        System.out.println("xy ly ngoai le");
    }

    System.out.println("tiep tuc chuong trinh..");
}
}

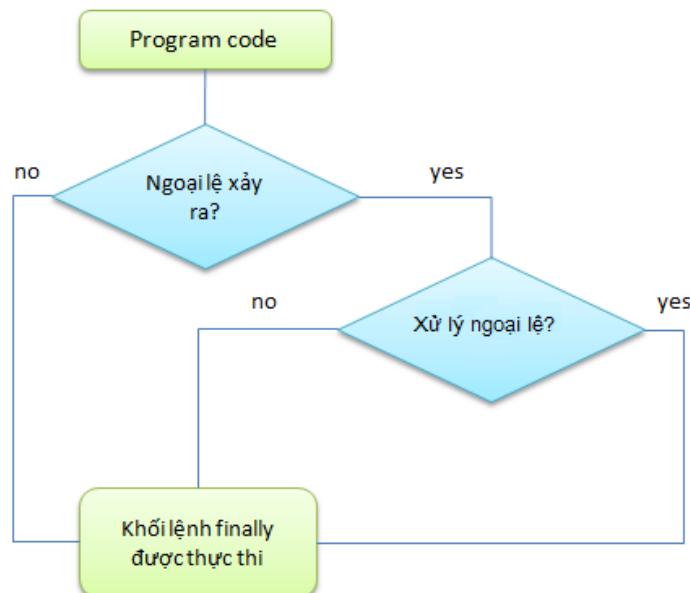
```

3.12. Khối lệnh finally trong java

Khối lệnh finally trong java được sử dụng để thực thi các lệnh quan trọng như đóng kết nối, đóng cá stream,...

Khối lệnh finally trong java luôn được thực thi cho dù có ngoại lệ xảy ra hay không hoặc gặp lệnh return trong khối try.

Khối lệnh finally trong java được khai báo sau khối lệnh try hoặc sau khối lệnh catch.



Note: Nếu bạn không xử lý ngoại lệ, trước khi kết thúc chương trình, JVM thực thi khối finally (nếu có).

3.12.1. Tại sao phải sử dụng khối finally

Khối finally có thể được sử dụng để chèn lệnh "cleanup" vào chương trình như việc đóng file, đóng các kết nối,...

3.12.2. Cách sử dụng khối finally trong java

Dưới đây là các trường hợp khác nhau về việc sử dụng khối finally trong java.

TH1:

sử dụng khối lệnh finally nơi ngoại lệ không xảy ra.

```
public class TestFinallyBlock {  
    public static void main(String args[]) {  
        try {  
            int data = 25 / 5;  
            System.out.println(data);  
        } catch (NullPointerException e) {  
            System.out.println(e);  
        } finally {  
            System.out.println("finally block is always executed");  
        }  
        System.out.println("rest of the code...");  
    }  
}
```

Output:

```
5  
finally block is always executed  
rest of the code...
```

TH2:

sử dụng khối lệnh finally nơi ngoại lệ xảy ra nhưng không xử lý.

```
public class TestFinallyBlock1 {  
    public static void main(String args[]) {  
        try {  
    
```

```

        int data = 25 / 0;
        System.out.println(data);
    } catch (NullPointerException e) {
        System.out.println(e);
    } finally {
        System.out.println("finally block is always executed");
    }
    System.out.println("rest of the code...");
}
}

```

Output:

```

finally block is always executed
Exception in thread "main" java.lang.ArithmetiException: / by
zero

```

TH3:

sử dụng khối lệnh finally nơi ngoại lệ xảy ra và được xử lý.

```

public class TestFinallyBlock2 {
    public static void main(String args[]) {
        try {
            int data = 25 / 0;
            System.out.println(data);
        } catch (ArithmetiException e) {
            System.out.println(e);
        } finally {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code...");
    }
}

```

Output:

```

java.lang.ArithmetiException: / by zero
finally block is always executed
rest of the code...

```

TH4:

Sử dụng khôi lệnh finally trong trường hợp trong khôi try có lệnh return.

```
public class TestFinallyBlock3 {
    public static void main(String args[]) {
        try {
            int data = 25;
            if (data % 2 != 0) {
                System.out.println(data + " is odd number");
                return;
            }
        } catch (ArithmetricException e) {
            System.out.println(e);
        } finally {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code...");
    }
}
```

Output:

```
25 is odd number
finally block is always executed
```

3.13. Từ khóa throw trong java

3.13.1. Từ khóa throw trong java

Từ khoá throw trong java được sử dụng để ném ra một ngoại lệ cụ thể.

Chúng ta có thể ném một trong hai ngoại lệ checked hoặc unchecked trong java bằng từ khóa **throw**. Từ khóa throw chủ yếu được sử dụng để ném ngoại lệ tùy chỉnh (ngoại lệ do người dùng tự định nghĩa). Chúng ta sẽ học ngoại lệ tùy chỉnh trong bài sau.

Cú pháp từ khóa throw:

```
throw exception;
```

Ví dụ về throw IOException.

```
throw new IOException("File không tồn tại");
```

3.13.2. Ví dụ về từ khóa throw trong java

Ví dụ 1: throw ra ngoại lệ nhưng không xử lý

Trong ví dụ này, chúng ta tạo ra phương thức validate() với tham số truyền vào là giá trị integer. Nếu tuổi dưới 18, chúng ta ném ra ngoại lệ ArithmeticException nếu không in ra một thông báo "welcome".

```
public class TestThrow1 {
    static void validate(int age) {
        if (age < 18)
            throw new ArithmeticException("not valid");
        else
            System.out.println("welcome");
    }

    public static void main(String args[]) {
        validate(13);
        System.out.println("rest of the code...");
    }
}
```

Output:

```
Exception in thread "main" java.lang.ArithmetricException: not
valid
```

Ví dụ 1: throw ra ngoại lệ nhưng có xử lý

```
public class TestThrow2 {
    static void validate(int age) {
        try {
            if (age < 18)
                throw new ArithmeticException("not valid");
            else
                System.out.println("welcome");
        } catch (ArithmetricException ex) {
            System.out.println(ex.getMessage());
        }
    }

    public static void main(String args[]) {
```

```

        validate(13);
        System.out.println("rest of the code...");
    }
}

```

Output:

```

not valid
rest of the code...

```

3.14. Từ khóa throws trong java

3.14.1. Từ khóa throws trong java

Từ khóa throws trong java được sử dụng để khai báo một ngoại lệ. Nó thể hiện thông tin cho lập trình viên rằng có thể xảy ra một ngoại lệ, vì vậy nó là tốt hơn cho các lập trình viên để cung cấp các mã xử lý ngoại lệ để duy trì luồng bình thường của chương trình.

Exception Handling chủ yếu được sử dụng để xử lý ngoại lệ checked. Nếu xảy ra bất kỳ ngoại lệ unchecked như NullPointerException, đó là lỗi của lập trình viên mà anh ta không thực hiện kiểm tra trước khi code được sử dụng.

Cú pháp của throws trong java

```

return_type method_name() throws exception_class_name {
    / /method code
}

```

Ngoại lệ nào nên được khai báo

Chỉ ngoại lệ checked, bởi vì:

- **Ngoại lệ unchecked:** nằm trong sự kiểm soát của bạn.
- **error:** nằm ngoài sự kiểm soát của bạn, ví dụ bạn sẽ không thể làm được bất kì điều gì khi các lỗi VirtualMachineError hoặc StackOverflowError xảy ra.

3.14.2. Lợi ích của từ khóa throws trong java

- Ngoại lệ checked có thể được ném ra ngoài và được xử lý ở một hàm khác.
- Cung cấp thông tin cho caller của phương thức về các ngoại lệ.

3.14.3. Ví dụ về từ khóa throws trong java

Dưới đây là ví dụ về mệnh đề throws trong java mô tả rằng ngoại lệ checked có thể được truyền ra bằng từ khóa throws.

```
import java.io.IOException;

public class TestThrows1 {
    void m() throws IOException {
        throw new IOException("Loi thiet bi");// checked exception
    }

    void n() throws IOException {
        m();
    }

    void p() {
        try {
            n();
        } catch (Exception e) {
            System.out.println("ngoai le duoc xu ly");
        }
    }

    public static void main(String args[]) {
        TestThrows1 obj = new TestThrows1();
        obj.p();
        System.out.println("luong binh thuong...");
    }
}
```

Output:

```
ngoai le duoc xu ly
luong binh thuong...
```

Quy tắc: Nếu bạn đang gọi một phương thức khai báo throws một ngoại lệ, bạn phải bắt hoặc throws ngoại lệ đó.

Có hai trường hợp:

- **TH1:** Bạn đã bắt ngoại lệ, tức là xử lý ngoại lệ bằng cách sử dụng try/catch.

- **TH2:** Bạn khai báo ném ngoại lệ, tức là sử dụng từ khóa throws với phương thức.

3.14.3.1. TH1: xử lý ngoại lệ với try/catch

Trong trường hợp bạn xử lý ngoại lệ, code sẽ được thực thi tốt cho dù ngoại lệ có xuất hiện trong chương trình hay không.

```
import java.io.IOException;

class M {
    void method() throws IOException {
        throw new IOException("Loi thiet bi");
    }
}

public class TestThrows2 {
    public static void main(String args[]) {
        try {
            M m = new M();
            m.method();
        } catch (Exception e) {
            System.out.println("Ngoai le duoc xu ly");
        }

        System.out.println("Luong binh thuong...");
    }
}
```

Output:

```
Ngoai le duoc xu ly
Luong binh thuong...
```

3.14.3.2. TH2: Khai báo throws ngoại lệ

- A) Trong trường hợp bạn khai báo throws ngoại lệ, nếu ngoại lệ không xảy ra, code sẽ được thực hiện tốt.
- B) Trong trường hợp bạn khai báo throws ngoại lệ, nếu ngoại lệ xảy ra, một ngoại lệ sẽ được ném ra tại runtime vì throws nên không xử lý ngoại đó.

A) Ngoại lệ không xảy ra

```
import java.io.IOException;

class M {
    void method() throws IOException {
        System.out.println("Thiet bi dang hoat dong tot");
    }
}

public class TestThrows2 {
    public static void main(String args[]) throws IOException {
        M m = new M();
        m.method();
        System.out.println("Luong binh thuong...");
    }
}
```

Output:

```
Thiet bi dang hoat dong tot
```

```
Luong binh thuong...
```

Ngoại lệ xảy ra

```
import java.io.IOException;

class M {
    void method() throws IOException {
        throw new IOException("Thiet bi");
    }
}

public class TestThrows2 {
    public static void main(String args[]) throws IOException {
        M m = new M();
        m.method();
        System.out.println("Luong binh thuong...");
    }
}
```

```

    }
}

```

Output:

```
Exception in thread "main" java.io.IOException: Thiet bi
```

3.14.4. Sự khác nhau giữa throw và throws trong java

Có một vài sự khác nhau giữa từ khóa throw và throws trong java được mô tả trong bảng dưới đây.

No.	throw	throws
1)	Từ khóa throw trong java được sử dụng để ném ra một ngoại lệ rõ ràng.	Từ khóa throws trong java được sử dụng để khai báo một ngoại lệ.
2)	Ngoại lệ checked không được truyền ra nếu chỉ sử dụng từ khóa throw.	Ngoại lệ checked được truyền ra ngay cả khi chỉ sử dụng từ khóa throws.
3)	Sau throw là một instance.	Sau throws là một hoặc nhiều class.
4)	Throw được sử dụng trong phương thức.	Throws được khai báo ngay sau dấu đóng ngoặc đơn của phương thức.
5)	Bạn không thể throw nhiều exceptions.	Bạn có thể khai báo nhiều exceptions, Ví dụ: public void method() throws IOException,SQLException.

Ví dụ về throw trong java

```
void m() {
    int n;
    if (n < 0) {
        throw new ArithmeticException("sorry");
    }
}
```

Ví dụ về throws trong java

```
void m() throws ArithmeticException {
    //method code
}
```

Ví dụ về throw và throws trong java

```
void m() throws ArithmeticException {
    throw new ArithmeticException("sorry");
}
```

3.14.5. Sự khác nhau giữa final, finally và finalize

Sự khác nhau giữa final, finally và finalize được thể hiện trong bảng dưới đây:

No.	final	finally	finalize
1)	Final được sử dụng để áp dụng các hạn chế về class, phương thức và biến. Lớp final không thể được kế thừa, phương thức final không thể được ghi đè và giá trị biến final không thể thay đổi.	Finally được sử dụng để thực thi code quan trọng, nó luôn được thực thi cho dù ngoại lệ được xử lý hay không.	Finalize được sử dụng để thực hiện quá trình xử lý xóa ngay trước khi đối tượng thu gom rác.
2)	Final là một từ khóa.	Finally là một khối (block).	Finalize là một phương thức.

Ví dụ về final trong java

```
class FinalExample{
class FinalExample {
    public static void main(String[] args) {
        final int x = 100;
        x = 200; // Compile Time Error
    }
}
```

Ví dụ về finally trong java

```
class FinallyExample {
    public static void main(String[] args) {
        try {
            int x = 300;
        } catch (Exception e) {
            System.out.println(e);
        } finally {
            System.out.println("finally block is executed");
        }
    }
}
```

Ví dụ về finalize trong java

```
class FinalizeExample {
    public void finalize() {
        System.out.println("finalize called");
    }

    public static void main(String[] args) {
        FinalizeExample f1 = new FinalizeExample();
        FinalizeExample f2 = new FinalizeExample();
        f1 = null;
        f2 = null;
        System.gc();
    }
}
```

3.14.6. Exception handling với overriding phương thức trong java

Có một vài quy tắc về xử lý ngoại lệ (exception handling) với overriding phương thức trong java như sau:

- Nếu phương thức của lớp cha không khai báo ném ra exception

- Nếu phương thức của lớp cha không khai báo ném ra exception, phương thức được ghi đè của lớp cha không thể khai báo ném ra ngoại lệ checked, nhưng ngoại lệ unchecked thì có thể.
- **Nếu phương thức của lớp cha khai báo ném ra exception**
 - Nếu phương thức của lớp cha khai báo ném ra exception, phương thức được ghi đè của lớp cha có thể khai báo ném ra ngoại lệ tương tự, ngoại lệ con, nhưng không thể khai báo ném ra ngoại lệ cha.

Nếu phương thức của lớp cha không khai báo ném ra exception

1) Quy tắc: Nếu phương thức của lớp cha không khai báo ném ra exception, phương thức được ghi đè của lớp cha không thể khai báo ném ra ngoại lệ checked.

```

1  import java.io.*;
2
3  class Parent {
4
5      void msg() {
6          System.out.println("parent");
7      }
8  }
9
10 class TestExceptionChild extends Parent {
11
12     void msg() throws IOException {
13         System.out.println("TestExceptionChild");
14     }
15 }
16
17 public static void main(String args[]) {
18     Parent p = new TestExceptionChild();
19     p.msg();
20 }
```

Output:

Compile Time Error

2) Quy tắc: Nếu phương thức của lớp cha không khai báo ném ra exception, phương thức được ghi đè của lớp cha không thể khai báo ném ra ngoại lệ checked, nhưng ngoại lệ unchecked thì có thể.

```

3 import java.io.*;
4
5 class Parent {
6     void msg() {
7         System.out.println("parent");
8     }
9 }
10
11 class TestExceptionChild1 extends Parent {
12     void msg() throws ArithmeticException {
13         System.out.println("child");
14     }
15 }
16
17 public static void main(String args[]) {
18     Parent p = new TestExceptionChild1();
19     p.msg();
20 }

```

Output:

child

Nếu phương thức của lớp cha khai báo ném ra exception

Quy tắc: Nếu phương thức của lớp cha khai báo ném ra exception, phương thức được ghi đè của lớp cha có thể khai báo ném ra ngoại lệ tương tự, ngoại lệ con, nhưng không thể khai báo ném ra ngoại lệ cha.

Ví dụ về TH phương thức ghi đè của lớp cha khai báo ném ra ngoại lệ cha.

```

3 class Parent {
4     void msg() throws ArithmeticException {
5         System.out.println("parent");
6     }
7 }
8
9 class TestExceptionChild2 extends Parent {
10    void msg() throws Exception {
11        System.out.println("child");
12    }
13 }
14
15 public static void main(String args[]) {
16     Parent p = new TestExceptionChild2();
17     try {
18         p.msg();
19     } catch (Exception e) {
20     }
21 }

```

Exception class is parent of ArithmeticException class

Output:

Compile Time Error

Ví dụ về TH phương thức ghi đè của lớp cha khai báo ném ra ngoại lệ tương tự.

```

3 class Parent {
4     void msg() throws Exception {
5         System.out.println("parent");
6     }
7 }
8
9 class TestExceptionChild3 extends Parent {
▲10    void msg() throws Exception {
11        System.out.println("child");
12    }
13 }
14 public static void main(String args[]) {
15     Parent p = new TestExceptionChild3();
16     try {
17         p.msg();
18     } catch (Exception e) {
19     }
20 }
21 }
```

Output:

child

Ví dụ về TH phương thức ghi đè của lớp cha khai báo ném ra ngoại lệ con.

```

3 class Parent {
4     void msg() throws Exception {
5         System.out.println("parent");
6     }
7 }
8
9 class TestExceptionChild4 extends Parent {
▲10    void msg() throws ArithmeticException {
11        System.out.println("child");
12    }
13 }
14 public static void main(String args[]) {
15     Parent p = new TestExceptionChild4();
16     try {
17         p.msg();
18     } catch (Exception e) {
19     }
20 }
21 }
```

Output:

child

Ví dụ về TH phương thức ghi đè của lớp cha không khai báo ném ra ngoại lệ nào.

```

3 class Parent {
4     void msg() throws Exception {
5         System.out.println("parent");
6     }
7 }
8
9 class TestExceptionChild5 extends Parent {
10    void msg() { // Declares no exception
11        System.out.println("child");
12    }
13
14 public static void main(String args[]) {
15     Parent p = new TestExceptionChild5();
16     try {
17         p.msg();
18     } catch (Exception e) {
19     }
20 }
21 }
```

Output:

child

3.14.7. Exception tùy chỉnh trong java

Nếu bạn tạo ngoại lệ riêng của mình được biết đến như ngoại lệ tùy chỉnh (exception tùy chỉnh) hoặc ngoại lệ do người dùng định nghĩa. Các ngoại lệ tùy chỉnh trong Java được sử dụng để tùy chỉnh ngoại lệ theo nhu cầu của người dùng.

Sử dụng ngoại lệ tùy chỉnh, bạn có thể có ngoại lệ và message của riêng bạn.

Dưới đây là ví dụ đơn giản về exception tùy chỉnh trong java.

File: InvalidAgeException.java

```

1 class InvalidAgeException extends Exception {
2     InvalidAgeException(String s) {
3         super(s);
4     }
5 }
```

File: TestCustomException1.java

```

class TestCustomException1 {

    static void validate(int age) throws InvalidAgeException {
        if (age < 18)
            throw new InvalidAgeException("not valid");
        else
            System.out.println("welcome to vote");
    }

    public static void main(String args[]) {
        try {
            validate(13);
        } catch (Exception m) {
            System.out.println("Exception occurred: " + m);
        }

        System.out.println("rest of the code...");
    }
}

```

Output:

```

Output:Exception occurred: InvalidAgeException:not valid
      rest of the code...

```

3.14.8. Truyền Exception cho caller

Một ngoại lệ đầu tiên được ném ra từ phía trên của call stack (stack chứa các phương thức gọi đến nhau) và nếu nó không được catch, nó sẽ giảm xuống ngăn xếp đến phương thức trước, nếu không được catch ở đó, ngoại lệ lại giảm xuống phương thức trước, và cứ như vậy cho đến khi chúng được catch hoặc cho đến khi chúng chạm đến đáy của stack. Điều này được gọi là truyền ngoại lệ (truyền exception trong java).

Quy tắc: Theo mặc định, Các ngoại lệ unchecked được chuyển tiếp trong chuỗi gọi (được truyền).

Ví dụ về truyền exception trong java

```

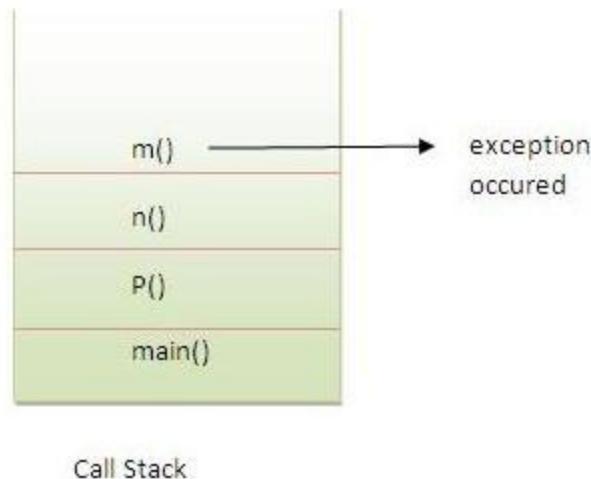
class TestExceptionPropagation1 {
    void m() {

```

```
int data = 50 / 0;  
}  
  
void n() {  
    m();  
}  
  
void p() {  
    try {  
        n();  
    } catch (Exception e) {  
        System.out.println("exception handled");  
    }  
}  
  
public static void main(String args[]) {  
    TestExceptionPropagation1 obj = new  
    TestExceptionPropagation1();  
    obj.p();  
    System.out.println("normal flow...");  
}
```

Output:

```
Output:exception handled  
      normal flow...
```



Trong ví dụ về ngoại lệ trên xảy ra trong phương thức `m()` nơi nó không được xử lý, do đó nó được truyền đến phương thức `n()` nhưng nó không được xử lý, một lần nữa nó được truyền đến phương thức `p()` trong đó ngoại lệ được xử lý.

Ngoại lệ có thể được xử lý trong bất kỳ phương thức nào trong call stack hoặc trong phương thức `main()`, phương thức `p()`, phương thức `n()` hoặc `m()`.

Quy tắc: Theo mặc định, Các ngoại lệ checked không được chuyển tiếp trong các hàm.

Chương trình dưới đây mô tả ngoại lệ đã checked không được truyền

```
class TestExceptionPropagation2 {
    void m() {
        throw new java.io.IOException("device error");// checked exception
    }

    void n() {
        m();
    }

    void p() {
        try {
            n();
        } catch (Exception e) {
            System.out.println("exception handled");
        }
    }
}
```

```
    }  
}  
  
public static void main(String args[]) {  
    TestExceptionPropagation2 obj = new TestExceptionPropagation2();  
    obj.p();  
    System.out.println("normal flow");  
}  
}
```

Output:

Output:Compile Time Error

CHƯƠNG 4. Java I/O

Gói **java.io** chứa gần như tất cả các lớp bạn cần để thực hiện input và output (I/O) trong Java. Tất cả những stream này biểu diễn một nguồn input và một đích đến output. Stream trong **java.io** package hỗ trợ nhiều như liệu như các kiểu gốc, Object, các ký tự nội bộ, ...

Một stream có thể được định nghĩa như là một dãy liên tục dữ liệu. **InputStream** được sử dụng để đọc dữ liệu từ một nguồn và **OutputStream** được sử dụng để ghi dữ liệu tới một đích đến.

Java cung cấp sự hỗ trợ mạnh mẽ nhưng linh hoạt cho I/O liên quan tới các File và các mạng nhưng trong phần hướng dẫn này, chúng tôi chỉ bàn luận tính năng cơ bản liên quan tới các stream và I/O. Chúng ta sẽ xem xét từng ví dụ được sử dụng phổ biến nhất.

4.1 Input/Output trong java

Java I/O hay Input/Output trong java được sử dụng để xử lý đầu vào và đầu ra trong java.

Java sử dụng khái niệm stream để làm cho hoạt động I/O nhanh hơn. Gói **java.io** chứa tất cả các lớp cần thiết cho hoạt động input và output.

4.1.1 Khái niệm về stream

Một stream là một dãy dữ liệu. Trong java, một stream bao gồm các byte. Nó được gọi là stream (dòng chảy) vì nó giống như một dòng nước chảy liên tục.

Trong java, 3 stream được tạo cho chúng ta một cách tự động. Tất cả các stream này được gắn với console.

1) **System.out**: output stream tiêu chuẩn

2) **System.in**: input stream tiêu chuẩn

3) **System.err**: error stream tiêu chuẩn

Chúng ta hãy xem đoạn mã để in thông báo **output** và **error** tới console.

```
System.out.println("simple message");
System.err.println("error message");
```

Còn đoạn code dưới đây lấy giá trị **input** từ console

```
int i=System.in.read(); //tra ve ma ASCII cua ky tu dau tien
```

```
System.out.println((char)i); //in ky tu lay duoc ra man hinh
```

4.1.2 Các lớp OutputStream với InputStream

Giải thích về các lớp OutputStream và InputStream được trình bày như dưới đây:

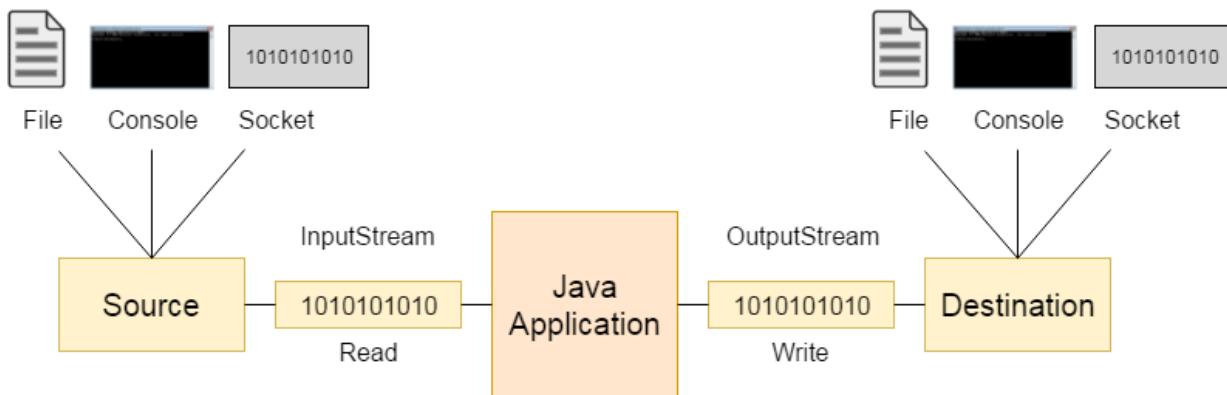
OutputStream

Ứng dụng Java sử dụng một output stream để ghi dữ liệu đến đích, nó có thể là một tệp tin, một mảng, thiết bị ngoại vi hoặc socket.

InputStream

Ứng dụng Java sử dụng một input stream để đọc dữ liệu từ một nguồn, nó có thể là một tệp tin, một mảng, thiết bị ngoại vi hoặc socket.

Hoạt động của của Java OutputStream và InputStream được mô tả trong hình dưới đây.



4.2 Lớp OutputStream

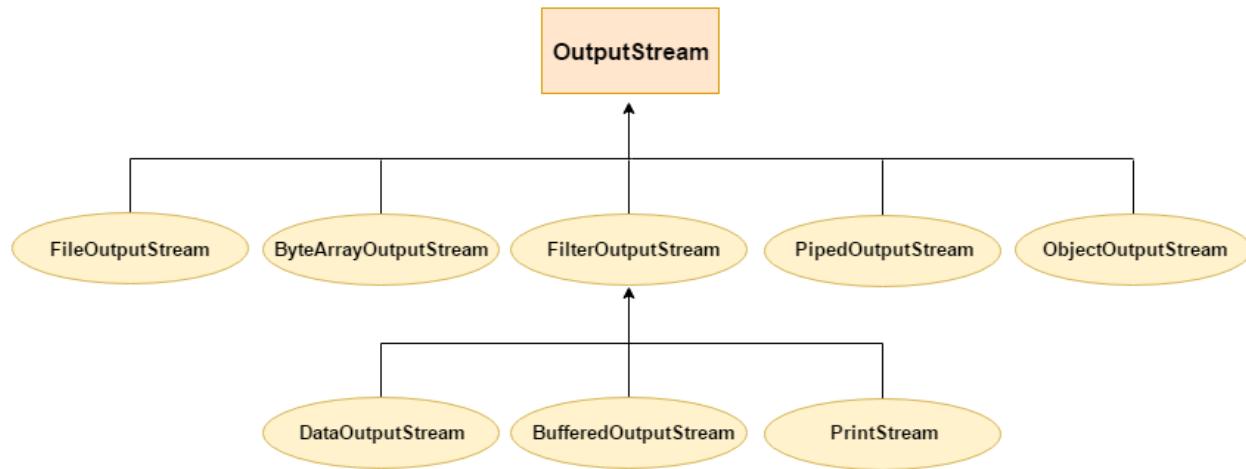
Lớp OutputStream là một lớp trừu tượng. Nó là super class của tất cả các lớp đại diện cho một output stream của các byte. Một output stream chấp nhận output các byte và gửi chúng đến một nơi có thể chứa.

Các phương thức của lớp OutputStream

Method	Description
1) public void write(int) throws IOException	được sử dụng để ghi một byte đến output stream hiện tại.
2) public void write(byte[]) throws IOException	được sử dụng để ghi một mảng các byte đến

	output stream hiện tại.
3) public void flush()throws IOException	flush output stream hiện tại.
4) public void close()throws IOException	được sử dụng để đóng output stream hiện tại.

OutputStream Hierarchy



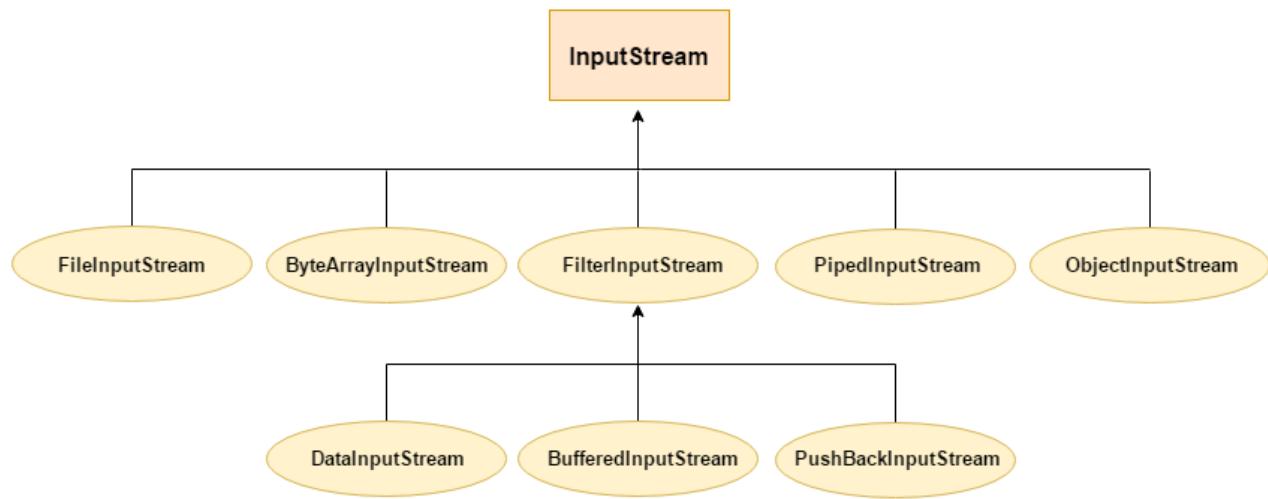
4.3 Lớp InputStream

Lớp InputStream là một lớp trừu tượng. Nó là super class của tất cả các lớp đại diện cho một input stream của các byte.

Các phương thức của lớp InputStream

Method	Description
1) public abstract int read()throws IOException	Đọc byte kế tiếp của dữ liệu từ input stream. Nó trả về -1 khi đọc đến vị trí cuối tập tin.
2) public int available()throws IOException	Trả về một ước tính về số byte có thể đọc được từ input stream hiện tại.
3) public void close()throws IOException	được sử dụng để đóng input stream hiện tại.

InputStream Hierarchy



Bạn có biết?

- Làm thế nào để ghi một dữ liệu chung cho nhiều tập tin bằng cách sử dụng một stream duy nhất?
- Làm thế nào chúng ta có thể truy cập nhiều file bằng một stream duy nhất?
- Làm thế nào chúng ta có thể cải thiện hiệu suất của hoạt động Input và Output?
- Chúng ta có thể đọc được dữ liệu từ bàn phím trong java như thế nào?
- Lớp Console là gì?
- Cách nén và giải nén dữ liệu của một file trong java?

Tìm hiểu thêm mã nguồn

ĐỌC GHI FILE TRONG JAVA

4.4 Byte Stream trong Java

Byte Stream trong Java được sử dụng để thực hiện input và output của các byte (8 bit). Mặc dù có nhiều lớp liên quan tới byte stream nhưng các lớp thường được sử dụng nhất là: **FileInputStream** và **FileOutputStream**. Sau đây là một ví dụ sử dụng hai lớp này để sao chép một input file vào trong một output file:

```
import java.io.*;

public class CopyFile {
    public static void main(String args[]) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;

        try {
            in = new FileInputStream("input.txt");
            out = new FileOutputStream("output.txt");

            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

Bây giờ giả sử chúng ta có một file là *input.txt* có nội dung sau:

Day là ví dụ về sao chép file.

Trong bước tiếp theo, biên dịch chương trình trên và thực thi nó, sẽ cho kết quả là tạo một file là *output.txt* có cùng nội dung như chúng ta có trong *input.txt*. Vì thế, bạn đặt code trên vào trong *CopyFile.java* file và làm như sau:

```
$javac CopyFile.java
$java CopyFile
```

4.5 Character Stream trong Java

Byte Stream trong Java được sử dụng để thực hiện input và output của các byte (8 bit), trong khi đó, **Character** Stream trong Java được sử dụng để thực hiện input và output cho Unicode 16 bit. Mặc dù có nhiều lớp liên quan tới character stream nhưng các lớp thường dùng nhất là **FileReader** và **FileWriter**... Mặc dù trong nội tại, **FileReader** sử dụng **FileInputStream** và **FileWriter** sử dụng **FileOutputStream**, nhưng có một điểm khác biệt lớn ở đây là **FileReader** đọc hai byte cùng một thời điểm và **FileWriter** ghi 2 byte cùng một lúc.

Chúng ta có thể viết lại ví dụ trên mà sử dụng hai lớp này để sao chép một input file (có các ký tự Unicode) vào trong một output file.

```
import java.io.*;

public class CopyFile {
    public static void main(String args[]) throws IOException
    {
        FileReader in = null;
        FileWriter out = null;

        try {
            in = new FileReader("input.txt");
            out = new FileWriter("output.txt");

            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
        }finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

```

    }
}
```

Giả sử chúng ta có **input.txt** có nội dung sau:

```
Day la vi du ve sao chep file.
```

Trong bước tiếp theo, biên dịch chương trình trên và thực thi nó, sẽ cho kết quả là tạo một file là output.txt có cùng nội dung như chúng ta có trong input.txt. Vì thế, bạn đặt code trên vào trong CopyFile.java file và làm như sau:

```
$javac CopyFile.java
$java CopyFile
```

4.6 Standard Stream trong Java

Tất cả các Ngôn ngữ lập trình cung cấp sự hỗ trợ cho I/O chuẩn, tại đây chương trình của người sử dụng có thể nhận đầu vào từ một bàn phím và sau đó tạo kết quả trên màn hình máy tính. Nếu bạn đã biết về các ngôn ngữ C/C++, thì bạn phải biết về 3 thiết bị chuẩn là STDIN, STDOUT, và STDERR. Theo cách tương tự, Java cung cấp 3 Standard Stream sau:

- Đầu vào chuẩn (Standard Input):** Nó được sử dụng để truyền dữ liệu tới chương trình của người dùng và thường thì một bàn phím được sử dụng như là đầu vào chuẩn và được biểu diễn như **System.in**.
- Đầu ra chuẩn (Standard Output):** Nó được sử dụng để hiển thị kết quả đầu ra từ chương trình của người dùng và thường thì một màn hình máy tính được sử dụng như là đầu ra chuẩn và được biểu diễn như là **System.out**.
- Lỗi chuẩn (Standard Error):** Được sử dụng để hiển thị các lỗi trong chương trình của người dùng và thường thì một màn hình máy tính được sử dụng như là lỗi chuẩn và được biểu diễn như là **System.err**.

Sau đây là một chương trình đơn giản tạo **InputStreamReader** để đọc luồng đầu vào chuẩn tới khi người sử dụng gõ một "q".

```
import java.io.*;

public class ReadConsole {
    public static void main(String args[]) throws IOException
    {
        InputStreamReader cin = null;

        try {
            cin = new InputStreamReader(System.in);
            System.out.println("Nhập các ký tự, 'q' để thoát.");
            char c;
```

```

do {
    c = (char) cin.read();
    System.out.print(c);
} while(c != 'q');
}finally {
    if (cin != null) {
        cin.close();
    }
}
}
}

```

Giữ code trên trong ReadConsole.java file và thực thi và biên dịch nó như dưới đây. Chương trình này tiếp tục đọc và hiển thị kết quả tới khi chúng ta nhấn phím "q".

```

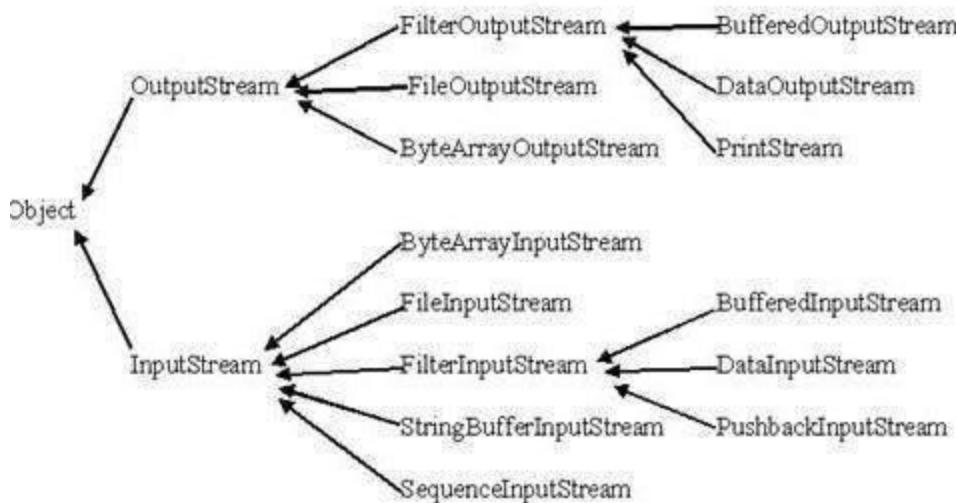
$javac ReadConsole.java
$java ReadConsole
Nhap cac ky tu, 'q' de thoat.
1
1
e
e
q
q

```

4.7 Đọc và Ghi File trong Java

Như đã miêu tả trước đó, một Stream có thể được định nghĩa như là một dãy liên tục của dữ liệu. **InputStream** được sử dụng để đọc dữ liệu từ một nguồn và **OutputStream** được sử dụng để ghi dữ liệu tới một đích.

Dưới đây là một cấu trúc có thứ tự của các lớp để xử lý các luồng Input và Output.



Hai luồng quan trọng nhất là **FileInputStream** và **FileOutputStream**, sẽ được bàn luận sau đây:

4.8 FileInputStream trong Java:

Luồng này được sử dụng để đọc dữ liệu từ các file. Các đối tượng có thể được tạo bởi sử dụng từ khóa new và có một số kiểu constructor có sẵn.

Constructor sau đây nhận tên file như là một chuỗi để tạo một đối tượng Input Stream để đọc file:

```
InputStream f = new FileInputStream("C:/java/hello");
```

Constructor sau nhận một đối tượng File để tạo một đối tượng Input Stream để đọc file. Đầu tiên chúng ta tạo một đối tượng file bởi sử dụng phương thức File() như sau:

```
File f = new File("C:/java/hello");
InputStream f = new FileInputStream(f);
```

Khi chúng ta có đối tượng **InputStream**, thì khi đó có một danh sách các phương thức có thể được sử dụng để đọc stream hoặc để thực hiện hoạt động nào khác trên stream này.

STT	Phương thức và Miêu tả
1	public void close() throws IOException{} Phương thức này đóng output stream. Giải phóng bất kỳ nguồn hệ thống nào liên kết với file. Ném một IOException
2	protected void finalize()throws IOException {} Phương thức này xóa sự kết nối tới File đó. Bảo đảm rằng phương thức close

	của output stream này được gọi khi không có tham chiếu nào nữa tới stream này. Ném một IOException
3	public int read(int r) throws IOException{} Phương thức này đọc byte dữ liệu đã xác định từ InputStream. Trả về một int. Trả về byte dữ liệu tiếp theo và -1 sẽ được trả về nếu kết thúc file.
4	public int read(byte[] r) throws IOException{} Phương thức này đọc r byte từ input stream vào trong một mảng. Trả về tổng số byte đã đọc. Nếu kết thúc file, -1 được trả về.
5	public int available() throws IOException{} Cung cấp số byte mà được đọc từ input stream này. Trả về một int

Có một số input stream quan trọng khác có sẵn, để biết thêm chi tiết, bạn tham khảo theo link sau:

- [ByteArrayInputStream](#)
- [DataInputStream](#)

4.9 FileOutputStream trong Java

FileOutputStream được sử dụng để tạo một file và ghi dữ liệu vào trong nó. Luồng này sẽ tạo một file, nếu nó đã không tồn tại, trước khi mở nó để ghi output.

Dưới đây là hai constructor mà có thể được sử dụng để tạo một đối tượng FileOutputStream trong Java.

Constructor sau nhận một tên file như là một chuỗi để tạo một đối tượng input stream để ghi file.

```
OutputStream f = new FileOutputStream("C:/java/hello")
```

Constructor sau nhận một đối tượng file để tạo một đối tượng output stream để ghi file.

```
File f = new File("C:/java/hello");
OutputStream f = new FileOutputStream(f);
```

Khi bạn có đối tượng **OutputStream** này, thì sau đây có các phương thức có thể được sử dụng để ghi stream hoặc để thực hiện các hoạt động khác trên stream này:

STT	Phương thức và Miêu tả
1	public void close() throws IOException{}

	Phương thức này đóng output stream. Giải phóng bất kỳ nguồn hệ thống nào liên kết với file. Ném một IOException
2	protected void finalize()throws IOException {} Phương thức này xóa sự kết nối tới File đó. Bảo đảm rằng phương thức close của output stream này được gọi khi không có tham chiếu nào nữa tới stream này. Ném một IOException
3	public void write(int w)throws IOException{} Phương thức này ghi byte đã xác định tới output stream
4	public void write(byte[] w) Ghi w byte từ mảng byte đã đề cập tới OutputStream.

Ngoài ra cũng có một số output stream quan trọng khác, bạn tham khảo theo link sau:

- [ByteArrayOutputStream](#)
- [DataOutputStream](#)

Ví dụ:

Ví dụ sau minh họa InputStream và OutputStream:

```
import java.io.*;

public class FileStreamTest{

    public static void main(String args[]){
        try{
            byte bWrite [] = {11,21,3,40,5};
            OutputStream os = new FileOutputStream("test.txt");
            for(int x=0; x < bWrite.length ; x++){
                os.write( bWrite[x] ); // writes the bytes
            }
            os.close();

            InputStream is = new FileInputStream("test.txt");
            int size = is.available();
```

```

for(int i=0; i< size; i++){
    System.out.print((char)is.read() + " ");
}
is.close();
}catch(IOException e){
    System.out.print("Exception");
}
}
}
}

```

Code trên tạo test.txt file và sẽ ghi các số đã cho trong định dạng nhị phân. Kết quả tương tự trên màn hình stdout.

4.10 Điều hướng file và I/O trong Java

Có một số lớp khác chúng ta cần tìm hiểu để biết các khái niệm cơ bản về Điều hướng file và I/O.

- [Lớp File trong Java](#)
- [Lớp FileReader trong Java](#)
- [Lớp FileWriter trong Java](#)

Thư mục trong Java

Một thư mục là File mà có thể giữ một danh sách các file và thư mục khác. Bạn sử dụng đối tượng **File** để tạo các thư mục, để liệt kê các file có sẵn trong một thư mục. Để hiểu chi tiết, bạn kiểm tra các phương thức mà bạn có thể gọi trên đối tượng File và những gì liên quan tới thư mục.

Để hiểu sâu hơn các khái niệm được trình bày trong tập bài tập chương này.

Tạo thư mục trong Java

Có hai phương thức tiện ích **File** hữu ích, mà có thể được sử dụng để tạo các thư mục:

- Phương thức **mkdir()** tạo một thư mục, trả về true nếu thành công và false nếu thất bại. Việc thất bại là do đường truyền đã xác định trong đối tượng File đã tồn tại, hoặc là do thư mục không thể được tạo vì cả đường truyền không tồn tại.
- Phương thức **mkdirs()** tạo cả một thư mục và tất cả các thư mục cha của nó.

Ví dụ sau tạo thư mục /tmp/user/java/bin

```

import java.io.File;

public class CreateDir {
    public static void main(String args[]) {
        String dirname = "/tmp/user/java/bin";
    }
}

```

```

File d = new File(dirname);
// Bay gio tao thu muc.
d.mkdirs();
}
}

```

Biên dịch và thực thi code trên sẽ tạo thư mục /tmp/user/java/bin

Ghi chú: Java tự động chăm sóc các bộ tách đường truyền trên UNIX và Windows theo qui ước tương ứng của từng hệ thống. Nếu bạn sử dụng một dấu gạch chéo (/) trên phiên bản Windows của Java, đường truyền sẽ vẫn giải quyết một cách chính xác.

Liệt kê thư mục trong Java

Bạn có thể sử dụng phương thức **list()** được cung cấp bởi đối tượng File để liệt kê tất cả các file và thư mục có sẵn trong một thư mục như sau:

```

import java.io.File;

public class ReadDir {
    public static void main(String[] args) {

        File file = null;
        String[] paths;

        try{
            // Tao doi tuong file moi
            file = new File("/tmp");

            // mang cac file va thu muc
            paths = file.list();

            // voi moi ten trong path array
            for(String path:paths)
            {
                // in ten file va ten thu muc
                System.out.println(path);
            }
        }catch(Exception e){
            // neu co bat cu error nao xuat hien
            e.printStackTrace();
        }
    }
}

```

{}

Nó sẽ cho kết quả sau dựa trên các thư mục và file có sẵn trong thư mục /tmp của bạn:

```
test1.txt
test2.txt
ReadDir.java
ReadDir.class
```

Ps: Về cơ bản, để đọc file binary(file ảnh, phim...) nên dùng InputStream, còn đọc file text (txt) nên dùng FileReader. Các bạn nhớ file docx không phải là file binary mà là file nhị phân.

4.11 Lớp Console trong java

Lớp Console trong java được sử dụng để lấy nội dung được nhập từ giao diện console. Nó cung cấp các phương thức để đọc văn bản và mật khẩu.

Nếu bạn đọc mật khẩu bằng cách sử dụng lớp Console, nó sẽ không được hiển thị cho người dùng.

Lớp java.io.Console được gắn với hệ thống điều khiển nội bộ. Lớp Console được giới thiệu từ jdk 1.5.

Hãy xem một ví dụ đơn giản để đọc văn bản từ console.

```
String text = System.console().readLine();
System.out.println("Text is: " + text);
```

Khai báo của lớp Console

Dưới đây là khai báo của lớp Java.io.Console:

```
public final class Console extends Object implements Flushable
```

Các phương thức của lớp Console trong java

Method	Description
Reader reader()	Nó được sử dụng để lấy ra đối tượng reader kết hợp với console

String readLine()	Nó được sử dụng để đọc một dòng văn bản từ console.
String readLine(String fmt, Object... args)	Nó cung cấp một dấu nhắc định dạng sau đó đọc dòng văn bản duy nhất từ console.
char[] readPassword()	Nó được sử dụng để đọc mật khẩu không được hiển thị trên console.
char[] readPassword(String fmt, Object... args)	Nó cung cấp một dấu nhắc định dạng sau đó đọc mật khẩu không được hiển thị trên console.
Console format(String fmt, Object... args)	Nó được sử dụng để ghi một chuỗi định dạng cho console output stream.
Console printf(String format, Object... args)	Nó được sử dụng để ghi một chuỗi vào console output stream.
PrintWriter writer()	Nó được sử dụng để lấy ra đối tượng PrintWriter kết hợp với console.
void flush()	Nó được sử dụng để xả console.

Làm thế nào để lấy ra đối tượng Console

Lớp System cung cấp một phương thức static console() trả về singleton instance của lớp Console.

```
public static Console console()
```

Hãy xem đoạn code để lấy instance của lớp Console.

```
Console c=System.console();
```

Ví dụ về lớp Console trong java

```
import java.io.Console;
```

```
public class ConsoleExample1 {  
    public static void main(String args[]) {  
        Console c = System.console();  
        System.out.println("Enter your name: ");  
        String n = c.readLine();  
        System.out.println("Welcome " + n);  
    }  
}
```

Output:

```
Enter your name: Toan Christ  
Welcome Toan Christ
```

Ví dụ về đọc password với lớp Console trong java

```
import java.io.Console;  
  
public class ConsoleExample2 {  
    public static void main(String args[]) {  
        Console c = System.console();  
        System.out.println("Enter password: ");  
        char[] ch = c.readPassword();  
        // convert char array into string  
        String pass = String.valueOf(ch);  
        System.out.println("Password is: " + pass);  
    }  
}
```

Output:

```
Enter password:  
Password is: 123456
```

4.12 Lớp FilePermission trong java

Lớp FilePermission trong java được sử dụng để cài đặt quyền (permission) cho một thư mục hoặc tập tin. Tất cả các quyền có liên quan với đường dẫn (path). Có hai loại đường dẫn:

1) **D:\IO**: Nó chỉ ra rằng cấp quyền có tác dụng với tất cả các thư mục con và các tập tin.

2) **D:\IO***: Nó chỉ ra rằng cấp quyền có tác dụng với tất cả các thư mục và các tập tin trong thư mục này trừ các thư mục con.

Khai báo của lớp FilePermission

Dưới đây là khai báo của lớp Java.io.FilePermission:

```
public final class FilePermission extends Permission implements Serializable
```

Các phương thức của lớp FilePermission

Method	Description
int hashCode()	Nó được sử dụng để trả về giá trị hash code của một đối tượng.
String getActions()	Nó được sử dụng để trả lại "biểu diễn chuỗi" của một action.
boolean equals(Object obj)	Nó được sử dụng để kiểm tra hai đối tượng FilePermission có bằng nhau không.
boolean implies(Permission p)	Nó được dùng để kiểm tra đối tượng FilePermission cho quyền được chỉ định.
PermissionCollection newPermissionCollection()	Nó được sử dụng để trả về đối tượng PermissionCollection mới để lưu trữ đối tượng FilePermission.

Ví dụ về lớp FilePermission trong java

Bạn hãy xem ví dụ đơn giản sau, trong đó quyền của một đường dẫn thư mục được cấp với quyền đọc (read) và một tập tin của thư mục này được cấp quyền ghi (write).

```
import java.io.FilePermission;
import java.io.IOException;
```

```
import java.security.PermissionCollection;

public class FilePermissionExample {
    public static void main(String[] args) throws IOException {
        // khai báo path1 là một file
        String path1 = "D:\\IO-Package\\java.txt";
        // khai báo path2 là thư mục cha của path1
        String path2 = "D:\\IO-Package\\-";

        // cấp quyền cho path1
        FilePermission file1 = new FilePermission(path2, "read");
        PermissionCollection permission = file1.newPermissionCollection();
        permission.add(file1);

        // cấp quyền cho path2
        FilePermission file2 = new FilePermission(path1, "write");
        permission.add(file2);

        // kiểm tra và in ra quyền của path1
        if (permission.implies(new FilePermission(path1, "read,write"))) {
            System.out.println("Quyền Read, Write được cấp cho path " +
path1);
        } else {
            System.out.println("Chỉ quyền Write được cấp cho path " + path1);
        }
    }
}
```

Output:

```
Quyền Read, Write được cấp cho path D:\\IO-Package\\java.txt
```

4.13 Ghi file trong java với lớp FileOutputStream

Java FileOutputStream là một output stream được sử dụng để ghi dữ liệu vào một file theo định dạng byte (byte stream).

Sử dụng lớp **FileOutputStream** trong java, nếu bạn phải ghi các giá trị nguyên thủy vào một file. Bạn có thể ghi dữ liệu theo định dạng byte hoặc định dạng ký tự thông

qua lớp FileOutputStream. Tuy nhiên, đối với các dữ liệu được ghi theo ký tự, sử dụng FileWriter thích hợp hơn FileOutputStream.

Khai báo của lớp FileOutputStream

Dưới đây là khai báo của lớp Java.io.FileOutputStream:

```
public class FileOutputStream extends OutputStream
```

Các phương thức của lớp FileOutputStream trong java

Method	Description
protected void finalize()	Nó được sử dụng để làm sạch kết nối với file output stream.
void write(byte[] ary)	Nó được sử dụng để ghi ary.length bytes từ mảng byte đến file output stream.
void write(byte[] ary, int off, int len)	Nó được sử dụng để viết len bytes từ mảng byte bắt đầu từ off tới file output stream.
void write(int b)	Nó được sử dụng để ghi byte cụ thể đến file output stream.
FileChannel getChannel()	Nó được sử dụng để trả lại đối tượng channel tập tin kết hợp với file output stream.
FileDescriptor getFD()	Nó được sử dụng để trả lại các mô tả của file liên quan đến stream.
void close()	Nó được sử dụng để đóng file output stream.

Ví dụ 1 về Java FileOutputStream : write byte

File: FileOutputStreamExample.java

```
import java.io.FileOutputStream;
import java.io.IOException;
```

```
public class FileOutputStreamExample {  
    public static void main(String args[]) throws IOException {  
        FileOutputStream fout = null;  
        try {  
            fout = new FileOutputStream("D:\\testout.txt");  
            fout.write(65);  
            fout.close();  
            System.out.println("success...");  
        } catch (Exception e) {  
            System.out.println(e);  
        } finally {  
            // close file output stream  
            fout.close();  
        }  
    }  
}
```

Output:

```
success...
```

Nội dung của file **testout.txt**

File: testout.txt

```
A
```

Ví dụ 2 về Java FileOutputStream : write String

File: FileOutputStreamExample2.java

```
import java.io.FileOutputStream;  
import java.io.IOException;  
  
public class FileOutputStreamExample2 {  
    public static void main(String args[]) throws IOException {  
        FileOutputStream fout = null;  
        try {  
            fout = new FileOutputStream("D:\\testout.txt");  
            String s = "Welcome to java.";  
            byte b[] = s.getBytes(); // converting string into byte array  
            fout.write(b);  
        }  
    }  
}
```

```

        fout.close();
        System.out.println("success...");
    } catch (Exception e) {
        System.out.println(e);
    } finally {
        fout.close();
    }
}
}

```

Output:

```
success...
```

Nội dung của file **testout.txt**

File: testout.txt

```
Welcome to java.
```

4.14 Đọc file trong java với lớp FileInputStream

Lớp FileInputStream trong java đọc được các byte từ một input file. Nó được sử dụng để đọc dữ liệu theo định dạng byte (các byte stream) như dữ liệu hình ảnh, âm thanh, video vv. Bạn cũng có thể đọc các dữ liệu có định dạng ký tự. Tuy nhiên, để đọc các dòng ký tự (các character stream), bạn nên sử dụng lớp FileReader.

Khai báo của lớp FileInputStream

Dưới đây là khai báo của lớp Java.io.FileInputStream :

```
1 public class FileInputStream extends OutputStream
```

Các phương thức của lớp FileInputStream trong java

Method	Description
int available()	Nó được sử dụng để trả về số byte ước tính có thể đọc được từ file input stream.
int read()	Nó được sử dụng để đọc byte dữ liệu từ file input stream.

int read(byte[] b)	Nó được sử dụng để đọc đến b.length byte dữ liệu từ file input stream.
int read(byte[] b, int off, int len)	Nó được sử dụng để đọc đến len byte dữ liệu từ ví trí off từ file input stream.
long skip(long x)	Nó được sử dụng để bỏ qua và loại bỏ x byte dữ liệu từ file input stream.
FileChannel getChannel()	Nó được sử dụng để trả về các đối tượng FileChannel duy nhất liên kết với file input stream.
FileDescriptor getFD()	Nó được sử dụng để trả về đối tượng FileDescriptor.
protected void finalize()	Nó được sử dụng để đảm bảo rằng phương thức close() được gọi khi không có tham chiếu đến file input stream.
void close()	Nó được sử dụng để đóng stream.

Ví dụ 1 về Java FileInputStream: đọc 1 ký tự

File: FileInputStreamExample1.java

```
import java.io.FileInputStream;
import java.io.IOException;

public class FileInputStreamExample1 {
    public static void main(String args[]) throws IOException {
        FileInputStream fin = null;
        try {
            fin = new FileInputStream("D:\\testout.txt");
            int i = fin.read();
            System.out.print((char) i);

            fin.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

```
        } finally {
            fin.close();
        }
    }
}
```

Chú ý: Trước khi chạy chương trình trên, cần phải tạo một file văn bản có tên là "testout.txt". Trong file này, chúng ta đang có nội dung sau:

File: testout.txt

```
Welcome to java.
```

Sau khi thực hiện chương trình trên, bạn sẽ nhận được một ký tự duy nhất từ file đó là 87 (dưới dạng byte). Để xem đó là ký tự gì, bạn cần phải chuyển đổi nó thành kiểu dữ liệu char.

Output:

```
W
```

Ví dụ 2 về Java FileInputStream: đọc toàn bộ nội dung file

File: FileInputStreamExample2.java

```
import java.io.FileInputStream;
import java.io.IOException;

public class FileInputStreamExample2 {
    public static void main(String args[]) throws IOException {
        FileInputStream fin = null;
        try {
            fin = new FileInputStream("D:\\testout.txt");
            int i = 0;
            while ((i = fin.read()) != -1) {
                System.out.print((char) i);
            }
            fin.close();
        } catch (Exception e) {
            System.out.println(e);
        } finally {
            fin.close();
        }
    }
}
```

```

    }
}

```

Output:

```
Welcome to java.
```

4.15 Ghi file trong java với lớp BufferedOutputStream

Lớp BufferedOutputStream trong java được sử dụng để đệm một output stream. Trong nội bộ của lớp này sử dụng bộ đệm để lưu trữ dữ liệu. Vì vậy, nó giúp hiệu suất ghi dữ liệu nhanh.

Để thêm bộ đệm vào một đối tượng OutputStream, sử dụng lớp BufferedOutputStream. Cú pháp để thêm bộ đệm vào một OutputStream như sau:

```

OutputStream os = null;
os = new BufferedOutputStream(new FileOutputStream("D:\\IO
Package\\testout.txt"));

```

Khai báo của lớp BufferedOutputStream

Dưới đây là khai báo của lớp Java.io.BufferedOutputStream:

```
public class BufferedOutputStream extends FilterOutputStream
```

Các constructor của lớp BufferedOutputStream

Constructor	Description
BufferedOutputStream(OutputStream os)	Nó tạo ra đối tượng BufferedOutputStream được sử dụng để ghi các dữ liệu vào output stream cụ thể.
BufferedOutputStream(OutputStream os, int size)	Nó tạo ra đối tượng BufferedOutputStream được sử dụng để ghi các dữ liệu vào output stream cụ thể với kích thước bộ đệm cụ thể.

Các phương thức của lớp BufferedOutputStream

Phương thức	Mô tả
-------------	-------

void write(int b)	Nó được sử dụng để ghi một byte cụ thể tới buffered output stream.
void write(byte[] b, int off, int len)	Nó ghi các byte từ byte-input stream được chỉ định vào mảng byte được chỉ định, bắt đầu bằng một giá trị cho trước off
void flush()	Xả những gì được lưu trong bộ đệm.

Ví dụ về ghi file trong java với lớp OutputStream

Trong ví dụ này, chúng ta ghi văn bản thông tin bằng đối tượng OutputStream được kết nối với đối tượng FileOutputStream. Phương thức flush() xả dữ liệu của một stream và gửi nó vào một stream khác. Nó được yêu cầu sử dụng nếu bạn đã kết nối một stream với một stream khác.

```
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class BufferedOutputStreamExample {
    public static void main(String args[]) throws IOException {
        FileOutputStream fout = null;
        BufferedOutputStream bout = null;

        try {
            fout = new FileOutputStream("D:\\testout.txt");
            bout = new BufferedOutputStream(fout);
            String s = "Welcome to java.";
            byte b[] = s.getBytes();
            bout.write(b);
            bout.flush();
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {
            bout.close();
            fout.close();
        }
    }
}
```

```

        System.out.println("success!");
    }
}

```

Output:

```

success!
File: testout.txt
Welcome to java.

```

4.16 Đọc file trong java với lớp BufferedInputStream

Lớp BufferedInputStream trong java được sử dụng để đọc thông tin từ stream (buffered stream). Trong nội bộ của lớp này sử dụng cơ chế đệm để làm cho hiệu suất đọc nhanh hơn.

Những điểm quan trọng về BufferedInputStream là:

- Khi các byte từ stream được bỏ qua hoặc đọc, bộ đệm nội bộ sẽ tự động nạp lại từ input stream chứa, nhiều byte tại một thời điểm.
- Khi một BufferedInputStream được tạo ra, một mảng đệm nội bộ sẽ được tạo ra.

Khai báo của lớp BufferedInputStream

Dưới đây là khai báo của lớp Java.io.BufferedInputStream:

```
public class BufferedInputStream extends FilterInputStream
```

Các constructor của lớp BufferedInputStream

Constructor	Mô tả
BufferedInputStream(InputStream IS)	Nó tạo ra đối tượng BufferedInputStream và lưu đối số của nó, input stream IS, để sử dụng sau này.
BufferedInputStream(InputStream IS, int size)	Nó tạo ra đối tượng BufferedInputStream với kích thước bộ đệm cụ thể và lưu đối số của nó, input stream IS, để sử dụng sau này.

Các phương thức của lớp BufferedInputStream

Phương thức	Mô tả
int available()	Nó trả về một số ước lượng của byte có thể đọc được từ input stream mà không bị chặn bởi phương thức gọi tiếp theo cho input stream.
int read()	Nó đọc byte tiếp theo của dữ liệu từ input stream.
int read(byte[] b, int off, int ln)	Nó đọc các byte từ từ input stream được chỉ định vào một mảng byte được chỉ định, bắt đầu với số cho trước.
void close()	Nó đóng các input stream và giải phóng bất kỳ tài nguyên hệ thống nào liên quan đến stream.
void reset()	It repositions the stream at a position the mark method was last called on this input stream.
void mark(int readlimit)	Nó được xem như đinh ước chung của phương thức đánh dấu cho input stream.
long skip(long x)	Nó bỏ qua và loại bỏ x byte dữ liệu từ input stream.
boolean markSupported()	Nó kiểm tra xem input stream có hỗ trợ các phương thức mark và reset không.

Ví dụ về đọc file trong java với lớp BufferedInputStream

Dưới đây là ví dụ đọc data từ file bằng cách sử dụng BufferedInputStream

File: BufferedInputStreamExample.java

```
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.IOException;

public class BufferedInputStreamExample {
    public static void main(String args[]) throws IOException {
        FileInputStream fin = null;
        BufferedInputStream bin = null;
```

```

try {
    fin = new FileInputStream("D:\\testout.txt");
    bin = new BufferedInputStream(fin);
    int i;
    while ((i = bin.read()) != -1) {
        System.out.print((char) i);
    }
} catch (IOException e) {
    System.out.println(e);
} finally {
    bin.close();
    fin.close();
}
}
}

```

Giả sử nội dung của file "testout.txt" file là:

Welcome to java.

Output:

Welcome to java.

4.17 Đọc file trong java với lớp SequenceInputStream

Lớp SequenceInputStream trong java được sử dụng để đọc dữ liệu từ nhiều stream. Nó đọc dữ liệu tuần tự (từng cái một).

Khai báo của lớp SequenceInputStream

Dưới đây là khai báo của lớp Java.io.SequenceInputStream:

```
public class SequenceInputStream extends InputStream
```

Các constructor của lớp SequenceInputStream

Constructor	Mô tả
SequenceInputStream(InputStream s1, InputStream s2)	Tạo ra một input stream mới bằng cách đọc dữ liệu của hai input stream theo thứ tự, đầu tiên s1 và sau đó s2.

SequenceInputStream(Enumeration e)	Tạo ra một input stream mới bằng cách đọc dữ liệu của một Enumeration mà kiểu của nó là InputStream.
------------------------------------	--

Các phương thức của lớp SequenceInputStream

Phương thức	Mô tả
int read()	Nó được sử dụng để đọc byte tiếp theo của dữ liệu từ input stream.
int read(byte[] ary, int off, int len)	Nó được sử dụng để đọc len byte dữ liệu từ dòng đầu vào vào mảng các byte tử vị trí off .
int available()	Nó được sử dụng để trả lại số byte tối đa có thể được đọc từ một input stream.
void close()	Nó được sử dụng để đóng input stream.

Ví dụ về đọc file trong java với lớp SequenceInputStream

Trong ví dụ này, chúng ta in dữ liệu của hai file testin.txt và testout.txt.

Giả sử các file testin.txt và testout.txt có nội dung như sau:

File: testin.txt

```
Welcome to Java IO.
```

File: testout.txt

```
It is the example of SequenceInputStream.
```

File: SequenceInputStreamExample.java

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.SequenceInputStream;

public class SequenceInputStreamExample {
    public static void main(String args[]) throws Exception {
        FileInputStream input1 = null;
        FileInputStream input2 = null;
        SequenceInputStream inst = null;
        try {

```

```

        input1 = new FileInputStream("D:\\testin.txt");
        input2 = new FileInputStream("D:\\testout.txt");
        inst = new SequenceInputStream(input1, input2);
        int j;
        while ((j = inst.read()) != -1) {
            System.out.print((char) j);
        }
    } catch (IOException ex) {

    } finally {
        inst.close();
        input1.close();
        input2.close();
    }
}
}

```

Output của chương trình trên:

Welcome to Java IO. It is the example of SequenceInputStream.

Ví dụ về đọc dữ liệu của 2 file và ghi vào một file khác

Trong ví dụ này, chúng ta đọc dữ liệu của từ 2 file **testin.txt** và **testout.txt** sau đó ghi dữ liệu vào file có tên là **testout2.txt**.

```

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.SequenceInputStream;

public class SequenceInputStreamExample2 {
    public static void main(String args[]) throws IOException {
        FileInputStream fin1 = new FileInputStream("D:\\testin.txt");
        FileInputStream fin2 = new FileInputStream("D:\\testout.txt");
        FileOutputStream fout = new FileOutputStream("D:\\testout2.txt");
        SequenceInputStream sis = new SequenceInputStream(fin1, fin2);
        int i;
        while ((i = sis.read()) != -1) {

```

```

        fout.write(i);
    }
    sis.close();
    fout.close();
    fin1.close();
    fin2.close();
    System.out.println("Success..");
}
}

```

Output:

Success...

Nội dung của file output **testout2.txt**

Welcome to Java IO. It is the example of SequenceInputStream.

Ví dụ về lớp SequenceInputStream đọc dữ liệu sử dụng Enumeration

Nếu chúng ta cần đọc dữ liệu từ nhiều hơn hai file, chúng ta cần sử dụng Enumeration. Có thể gọi ra thể hiện của đối tượng Enumeration bằng cách gọi phương thức elements() của lớp Vector. Trong ví dụ dưới đây chúng ta đọc dữ liệu từ 4 file: a.txt, b.txt, c.txt và d.txt.

File:

```

import java.io.FileInputStream;
import java.io.IOException;
import java.io.SequenceInputStream;
import java.util.Enumeration;
import java.util.Vector;

public class SequenceInputStreamExample3 {
    public static void main(String args[]) throws IOException {
        // tạo các đối tượng FileInputStream cho tất cả các file
        FileInputStream fin1 = new FileInputStream("D:\\a.txt");
        FileInputStream fin2 = new FileInputStream("D:\\b.txt");
        FileInputStream fin3 = new FileInputStream("D:\\c.txt");
        FileInputStream fin4 = new FileInputStream("D:\\d.txt");
        // Tạo đối tượng Vector để lưu các stram
        Vector<FileInputStream> v = new Vector<FileInputStream>();
        v.add(fin1);

```

```
v.add(fin2);
v.add(fin3);
v.add(fin4);
// Tạo đối Enumeration bởi việc gọi phương thức elements
Enumeration<FileInputStream> e = v.elements();
// Truyền đối tượng Enumeration tới constructor
SequenceInputStream bin = new SequenceInputStream(e);
int i = 0;
while ((i = bin.read()) != -1) {
    System.out.print((char) i);
}
bin.close();
fin1.close();
fin2.close();
fin3.close();
fin4.close();
}
```

Các file a.txt, b.txt, c.txt và d.txt có nội dung như dưới đây:

File: a.txt

Welcome

File: b.txt

to

File: c.txt

java

File: d.txt

programming

Output:

Welcometojavaprogramming

4.18 Ghi file trong java với lớp ByteArrayOutputStream

Lớp ByteArrayOutputStream trong java được sử dụng để ghi dữ liệu chung vào nhiều file. Trong luồng này, dữ liệu được ghi vào mảng byte có thể được ghi vào nhiều stream sau đó.

`ByteArrayOutputStream` giữ một bản sao của dữ liệu và chuyển tiếp nó đến nhiều stream.

Bộ đệm của `ByteArrayOutputStream` tự động tăng theo kích thước dữ liệu.

Khai báo của lớp `ByteArrayOutputStream`

Dưới đây là khai báo của lớp `Java.io.ByteArrayOutputStream`:

```
public class ByteArrayOutputStream extends OutputStream
```

Các constructor của lớp `ByteArrayOutputStream`

Constructor	Mô tả
<code>ByteArrayOutputStream()</code>	Tạo một <code>ByteArrayOutputStream</code> mới với dung lượng ban đầu là 32 byte, tuy nhiên kích thước của nó được tăng lên nếu cần thiết.
<code>ByteArrayOutputStream(int size)</code>	Tạo một <code>ByteArrayOutputStream</code> mới với dung lượng bộ đệm có kích thước quy định theo byte.

Các phương thức của lớp `ByteArrayOutputStream`

Phương thức	Mô tả
<code>int size()</code>	Nó được sử dụng để trả về kích thước hiện tại của một bộ đệm.
<code>byte[] toByteArray()</code>	Nó được sử dụng để tạo ra một mảng byte mới được phân bổ.
<code>String toString()</code>	Nó được sử dụng để chuyển đổi nội dung thành một chuỗi byte giải mã bằng cách sử dụng một bộ ký tự mặc định.
<code>String toString(String charsetName)</code>	Nó được sử dụng để chuyển đổi nội dung thành một chuỗi byte giải mã bằng cách sử dụng một charsetName cụ thể.
<code>void write(int b)</code>	Nó được sử dụng để ghi các byte được chỉ định.
<code>void write(byte[] b, int off, int len)</code>	Nó được sử dụng để ghi <code>len</code> byte từ mảng byte xác định bắt đầu từ <code>off</code> .

void writeTo(OutputStream out)	Nó được sử dụng để ghi nội dung hoàn chỉnh đến output stream cụ thể.
void reset()	Nó được sử dụng để thiết lập lại trường count của đối tượng ByteArrayOutputStream về giá trị 0.
void close()	Nó được sử dụng để đóng ByteArrayOutputStream.

Ví dụ về ghi file trong java với lớp ByteArrayOutputStream

Dưới đây là ví dụ đơn giản về ByteArrayOutputStream để ghi dữ liệu chung vào 2 file f1.txt và f2.txt.

```

import java.io.ByteArrayOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class ByteArrayOutputStreamExample {
    public static void main(String args[]) throws Exception {
        FileOutputStream fout1 = null;
        FileOutputStream fout2 = null;
        ByteArrayOutputStream bout = null;

        try {
            fout1 = new FileOutputStream("D:\\f1.txt");
            fout2 = new FileOutputStream("D:\\f2.txt");
            bout = new ByteArrayOutputStream();
            String data = "Hello";
            bout.write(data.getBytes());
            bout.writeTo(fout1);
            bout.writeTo(fout2);
            bout.flush();
            System.out.println("Success...");
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {
    }
}

```

```

        fout1.close();
        fout2.close();
        bout.close();
    }

}

```

Output:

Success...

Nội dung của 2 file f1.txt và f2.txt như sau:

File: f1.txt

Hello

File: f2.txt

Hello

4.19 Lớp ByteArrayInputStream trong java

ByteArrayInputStream bao gồm hai từ: ByteArray và InputStream. Tên của nó cho thấy, nó có thể được sử dụng để đọc mảng byte như là input stream.

Lớp ByteArrayInputStream trong java chứa một bộ đệm bên trong được sử dụng để đọc mảng byte dưới dạng luồng.

Bộ đệm của ByteArrayInputStream tự động tăng theo kích thước dữ liệu.

Khai báo của lớp ByteArrayInputStream

Dưới đây là khai báo của lớp Java.io.ByteArrayInputStream:

```
public class ByteArrayInputStream extends InputStream
```

Các constructor của lớp ByteArrayInputStream

Constructor	Mô tả
ByteArrayInputStream(byte[] ary)	Tạo ra một đối tượng ByteArrayInputStream mới sử dụng ary làm mảng đệm của nó.
ByteArrayInputStream(byte[] ary, int offset, int len)	Tạo ra một đối tượng ByteArrayInputStream mới sử dụng ary làm mảng đệm của nó có thể đọc lên len quy

	định byte dữ liệu từ một mảng.
--	---------------------------------------

Các phương thức của lớp ByteArrayInputStream

Methods	Description
int available()	Nó được sử dụng để trả lại số byte còn lại có thể được đọc từ input stream.
int read()	Nó được sử dụng để đọc byte tiếp theo của dữ liệu từ input stream.
int read(byte[] ary, int off, int len)	Nó được sử dụng để đọc lên đến len byte dữ liệu từ một mảng các byte trong input stream.
boolean markSupported()	Nó được sử dụng để kiểm tra input stream cho các phương thức mark() và reset().
long skip(long x)	Nó được sử dụng để bỏ qua x các byte của đầu vào từ input stream.
void mark(int readAheadLimit)	Nó được sử dụng để thiết lập vị trí đánh dấu hiện tại trong the stream.
void reset()	Nó được sử dụng để thiết lập lại bộ đệm của một mảng byte.
void close()	Nó được sử dụng để đóng ByteArrayInputStream.

Ví dụ về đọc file trong java với lớp ByteArrayInputStream

Dưới đây là ví dụ về lớp ByteArrayInputStream trong java để đọc mảng byte như một input stream.

File: ByteArrayInputStreamExample.java

```
import java.io.ByteArrayInputStream;
import java.io.IOException;
```

```

public class ByteArrayInputStreamExample {
    public static void main(String[] args) throws IOException {
        ByteArrayInputStream bis = null;

        try {
            byte[] buf = { 35, 36, 37, 38 };
            // Create the new byte array input stream
            bis = new ByteArrayInputStream(buf);
            int k = 0;
            while ((k = bis.read()) != -1) {
                // Conversion of a byte into character
                char ch = (char) k;
                System.out.println("ASCII value of Character is:" + k + " " +
                    + "Special character is: " + ch);
            }
        } finally {
            bis.close();
        }
    }
}

```

Output:

```

ASCII value of Character is:35; Special character is: #
ASCII value of Character is:36; Special character is: $
ASCII value of Character is:37; Special character is: %
ASCII value of Character is:38; Special character is: &

```

4.20 Ghi file trong jav với lớp DataOutputStream

Lớp DataOutputStream trong java cho phép một ứng dụng ghi các kiểu dữ liệu Java nguyên thủy đến output stream một cách độc lập với máy.

Ứng dụng Java thường sử dụng DataOutputStream để ghi dữ liệu mà sau này có thể được đọc bởi một DataInputStream.

Khai báo của lớp DataOutputStream

Dưới đây là khai báo của lớp Java.io.DataOutputStream:

```
public class DataOutputStream extends FilterOutputStream implements DataOutput
```

Các phương thức của lớp DataOutputStream

Phương thức	Mô tả
int size()	Nó được sử dụng để trả về số byte đã được ghi.
void write(int b)	Nó được sử dụng để ghi các byte được chỉ định cho output stream.
void write(byte[] b, int off, int len)	Nó được sử dụng để ghi len byte dữ liệu bắt đầu từ off vào output stream.
void writeBoolean(boolean v)	Nó được sử dụng để ghi Boolean đến output stream như một giá trị 1 byte.
void writeChar(int v)	Nó được sử dụng để ghi char đến output stream như một giá trị 2 byte.
void writeChars(String s)	Nó được sử dụng để ghi chuỗi cho output stream như là một dãy các ký tự.
void writeByte(int v)	Nó được sử dụng để ghi byte đến output stream như một giá trị 1 byte.
void writeBytes(String s)	Nó được sử dụng để ghi chuỗi đến output stream như là một dãy các byte.
void writeInt(int v)	Nó được sử dụng để ghi int đến output stream
void writeShort(int v)	Nó được sử dụng để ghi short đến output stream.
void writeLong(long v)	Nó được sử dụng để ghi long đến output stream.
void writeUTF(String str)	Nó được sử dụng để ghi một chuỗi đến output stream sử dụng mã hóa UTF-8 theo cách di động.
void flush()	Nó được sử dụng để xả DataOutputStream.

Ví dụ về ghi file trong java với lớp DataOutputStream

Trong ví dụ này, chúng ta đang ghi dữ liệu vào một file văn bản **testout.txt** sử dụng lớp DataOutputStream.

```
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class DataOutputStreamExample {
    public static void main(String[] args) throws IOException {
        FileOutputStream file = null;
        DataOutputStream data = null;

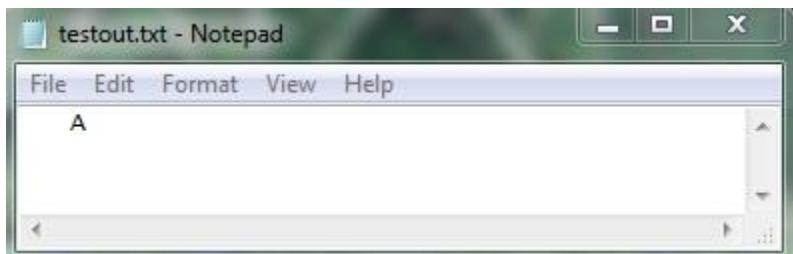
        try {
            file = new FileOutputStream("D:\\testout.txt");
            data = new DataOutputStream(file);
            data.writeInt(65);
            data.flush();
            System.out.println("Success...");
        } catch (IOException ex) {

        } finally {
            data.close();
        }
    }
}
```

Output:

```
Success...
```

Nội dung file **testout.txt**:



4.21 Đọc file trong java với lớp DataInputStream

Lớp DataInputStream trong java cho phép một ứng dụng đọc dữ liệu nguyên thủy từ luồng đầu vào một cách độc lập với máy.

Ứng dụng Java thường sử dụng DataOutputStream để ghi dữ liệu mà sau này có thể được đọc bởi một DataInputStream.

Khai báo của lớp DataInputStream

Dưới đây là khai báo của lớp Java.io.DataInputStream:

```
public class DataInputStream extends FilterInputStream implements DataInput
```

Các phương thức của lớp DataInputStream

Phương thức	Mô tả
int read(byte[] b)	Nó được sử dụng để đọc mảng byte từ input stream.
int read(byte[] b, int off, int len)	Nó được sử dụng để đọc len byte dữ liệu bắt đầu từ off từ input stream.
int readInt()	Đó được sử dụng để đọc các byte đầu vào và trả về một giá trị int.
byte readByte()	Nó được sử dụng để đọc và trả lại một byte đầu vào.
char readChar()	Nó được sử dụng để đọc hai byte đầu vào và trả về một giá trị char.
double readDouble()	Nó được sử dụng để đọc tám byte đầu vào và trả về một giá trị double.
boolean readBoolean()	Nó được sử dụng để đọc một byte đầu vào và trả về true nếu byte khác zero, false nếu byte là zero.
int skipBytes(int x)	Nó được sử dụng để bỏ qua x các byte dữ liệu từ input stream.
String readUTF()	Nó được sử dụng để đọc một chuỗi đã được mã hóa bằng cách sử dụng định dạng UTF-8.

void readFully(byte[] b)	Nó được sử dụng để đọc byte từ input stream và lưu trữ chúng vào mảng đệm.
void readFully(byte[] b, int off, int len)	Nó được sử dụng để đọc len byte từ vị trí off từ input stream.

Ví dụ về đọc file trong java với lớp DataInputStream

```

import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;

public class DataInputStreamExample {
    public static void main(String[] args) throws IOException {
        InputStream input = null;
        DataInputStream dis = null;
        try {
            input = new FileInputStream("D:\\testout.txt");
            dis = new DataInputStream(input);
            int count = input.available();
            byte[] arr = new byte[count];
            dis.read(arr);
            for (byte bt : arr) {
                char k = (char) bt;
                System.out.print(k + "-");
            }
        } catch (IOException ex) {

        } finally {
            dis.close();
        }
    }
}

```

Giả sử file "testout.txt" có nội dung như sau:

JAVA

Output:

J-A-V-A

4.22 Ghi file trong java với lớp FileWriter

Lớp FileWriter trong java được sử dụng để ghi các dữ liệu theo định dạng ký tự vào một file.

Không giống như lớp FileOutputStream, khi ghi dữ liệu bạn không cần phải chuyển đổi chuỗi thành mảng byte vì nó cung cấp phương thức để viết chuỗi trực tiếp.

Khai báo của lớp FileWriter

Dưới đây là khai báo của lớp Java.io.FileWriter:

```
public class FileWriter extends OutputStreamWriter
```

Các constructor của lớp FileWriter

Constructor	Mô tả
FileWriter(String file)	Tạo ra một file mới với giá trị truyền vào là tên file.
FileWriter(File file)	Tạo ra một file mới với giá trị truyền vào là đối tượng file.

Các phương thức của lớp FileWriter

Phương thức	Mô tả
void write(String text)	Nó được sử dụng để ghi string vào FileWriter.
void write(char c)	Nó được sử dụng để ghi char vào FileWriter.
void write(char[] c)	Nó được sử dụng để ghi mảng char vào FileWriter.
void flush()	Nó được sử dụng để xả dữ liệu của FileWriter.
void close()	Nó được sử dụng để đóng FileWriter.

Ví dụ về ghi file trong java với lớp FileWriter

Trong ví dụ này, chúng ta ghi dữ liệu vào file **testout.txt** bằng lớp Java FileWriter.

```
import java.io.FileWriter;

public class FileWriterExample {
    public static void main(String args[]) {
        try {
            FileWriter fw = new FileWriter("D:\\testout.txt");
            fw.write("Welcome to java.");
            fw.close();
        } catch (Exception e) {
            System.out.println(e);
        }
        System.out.println("Success....");
    }
}
```

Output:

```
Success...
```

```
testout.txt:
```

```
Welcome to java
```

4.23 Đọc file trong java với lớp FileReader

Lớp FileReader trong java được sử dụng để đọc dữ liệu từ file. Nó trả về dữ liệu theo định dạng byte như lớp FileInputStream.

Đây là lớp định hướng ký tự được sử dụng để xử lý file trong java.

Khai báo của lớp FileReader

Dưới đây là khai báo của lớp Java.io.FileReader:

```
public class FileReader extends InputStreamReader
```

Các constructor của lớp FileReader

Constructor	Description
FileReader(String file)	Nó mở file với tên file ở dạng string đã cho ở chế độ đọc. Nếu tập tin không tồn tại, nó ném ra ngoại lệ FileNotFoundException.
FileReader(File file)	Nó mở file với thể hiện của file đã cho ở chế độ đọc. Nếu tập tin không tồn tại, nó ném ra ngoại lệ FileNotFoundException.

Các phương thức của lớp FileReader

Method	Description
int read()	Nó được sử dụng để trả về một ký tự trong mẫu ASCII. Nó trả về -1 vào cuối tập tin.
void close()	Nó được sử dụng để đóng lớp FileReader.

Ví dụ về đọc file trong java với lớp FileReader

Trong ví dụ này, chúng ta đọc dữ liệu từ file văn bản testout.txt sử dụng lớp Java FileReader.

```
import java.io.FileReader;

public class FileReaderExample {
    public static void main(String args[]) throws Exception {
        FileReader fr = new FileReader("D:\\testout.txt");
        int i;
        while ((i = fr.read()) != -1) {
            System.out.print((char) i);
        }
        fr.close();
    }
}
```

Giả sử file output.txt có nội dung như sau:

```
Welcome to java.
```

Output:

```
Welcome to java.
```

4.24 Ghi file trong java với lớp FilterOutputStream

Lớp FilterOutputStream trong java extends lớp OutputStream. Nó cung cấp các lớp con khác nhau như BufferedOutputStream và DataOutputStream để cung cấp các chức năng bổ sung. Vì vậy, nó ít được sử dụng riêng lẻ.

Khai báo của lớp FilterOutputStream

Dưới đây là khai báo của lớp Java.io.FilterOutputStream:

```
public class FilterOutputStream extends OutputStream
```

Các phương thức của lớp FilterOutputStream

Phương thức	Mô tả
void write(int b)	Nó được sử dụng để ghi byte được chỉ định đến output stream.
void write(byte[] ary)	It is used to ghi ary.length byte to the output stream.
void write(byte[] b, int off, int len)	Nó được sử dụng để ghi len bytes từ off đến output stream.
void flush()	Nó được sử dụng để xả output stream.
void close()	Nó được sử dụng để đóng output stream.

Ví dụ về ghi file trong java với lớp FilterOutputStream

```
import java.io.File;
```

```
import java.io.FileOutputStream;
import java.io.FilterOutputStream;
import java.io.IOException;

public class FilterOutputStreamExample {
    public static void main(String[] args) throws IOException {
        FileOutputStream file = null;
        FilterOutputStream filter = null;
        try {
            file = new FileOutputStream(new File("D:\\testout.txt"));
            filter = new FilterOutputStream(file);
            String s = "Welcome to java.";
            byte b[] = s.getBytes();
            filter.write(b);
            filter.flush();
            System.out.println("Success...");
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {
            filter.close();
            file.close();
        }
    }
}
```

Output:

```
Success...
```

Nội dung của file "testout.txt":

```
Welcome to java.
```

4.25 Đọc file trong java với lớp FilterInputStream

Lớp FilterInputStream trong java extends lớp InputStream. Nó cung cấp các lớp con khác nhau như BufferedInputStream và DataInputStream để cung cấp chức năng bổ sung. Vì vậy, nó ít được sử dụng riêng lẻ.

Khai báo của lớp FilterInputStream

Dưới đây là khai báo của lớp Java.io.FilterInputStream:

```
public class FilterInputStream extends InputStream
```

Các phương thức của lớp FilterInputStream

Phương thức	Mô tả
int available()	Nó được sử dụng để trả về một số ước lượng của byte có thể được đọc từ input stream.
int read()	Nó được sử dụng để đọc byte tiếp theo của dữ liệu từ input stream.
int read(byte[] b)	Nó được sử dụng để đọc byte.length các byte dữ liệu từ input stream.
long skip(long n)	Nó được sử dụng để bỏ qua và loại bỏ n các byte dữ liệu từ input stream.
boolean markSupported()	Nó được sử dụng để kiểm tra xem các input stream có hỗ trợ các phương thức mark() và reset() không.
void mark(int readlimit)	Nó được sử dụng để đánh dấu vị trí hiện tại trong input stream.
void reset()	Nó được sử dụng để thiết lập lại input stream.
void close()	Nó được sử dụng để đóng input stream.

Ví dụ về đọc file trong java với lớp FilterInputStream

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FilterInputStream;
import java.io.IOException;

public class FilterInputStreamExample {
    public static void main(String[] args) throws IOException {
        FileInputStream file = null;
        FilterInputStream filter = null;
```

```

try {
    file = new FileInputStream(new File("D:\\testout.txt"));
    filter = new BufferedInputStream(file);

    int k = 0;
    while ((k = filter.read()) != -1) {
        System.out.print((char) k);
    }
} catch (IOException ex) {
    ex.printStackTrace();
}
file.close();
filter.close();
}
}

```

Giả sử file "testout.txt" có nội dung như sau:

Welcome to java.

Output:

Welcome to java.

4.26 Ghi file trong java với lớp BufferedWriter

Lớp BufferedWriter trong java được sử dụng để cung cấp bộ đệm cho các các thể hiện của lớp Writer. Nó giúp hiệu suất nhanh. Nó thừa kế lớp Writer. Các ký tự đệm được sử dụng để cung cấp việc ghi dữ liệu hiệu quả với các mảng đơn, các ký tự và chuỗi.

Khai báo của lớp BufferedWriter

Dưới đây là khai báo của lớp Java.io.BufferedWriter:

```
public class BufferedWriter extends Writer
```

Các constructor của lớp BufferedWriter

Constructor	Mô tả
BufferedWriter(Writer wrt)	Nó được sử dụng để tạo ra thể hiện của BufferedWriter có sử dụng kích thước mặc định cho một bộ đệm đầu ra.

BufferedWriter(Writer wrt, int size)	Nó được sử dụng để tạo ra thể hiện của BufferedWriter mà sử dụng kích thước quy định cho một bộ đệm đầu ra.
--------------------------------------	---

Các phương thức của lớp BufferedWriter

Method	Mô tả
void newLine()	Nó được sử dụng để thêm một dòng mới với dấu xuống dòng.
void write(int c)	Nó được sử dụng để ghi một ký tự duy nhất.
void write(char[] cbuf, int off, int len)	Nó được sử dụng để ghi một phần của một mảng các ký tự.
void write(String s, int off, int len)	Nó được sử dụng để ghi một phần của một chuỗi.
void flush()	Nó được sử dụng để xả BufferedWriter .
void close()	Nó được sử dụng để đóng BufferedWriter

Ví dụ về ghi file trong java với lớp BufferedWriter

Chúng ta hãy xem ví dụ đơn giản thực hiện việc ghi dữ liệu vào file **testout.txt** bằng Java BufferedWriter.

```
import java.io.BufferedReader;
import java.io.FileWriter;

public class BufferedWriterExample {
    public static void main(String[] args) throws Exception {
        FileWriter writer = new FileWriter("D:\\testout.txt");
        BufferedWriter buffer = new BufferedWriter(writer);
        buffer.write("Welcome to java.");
        buffer.close();
    }
}
```

```

        System.out.println("Success...") ;
    }
}

```

Output:

Success...

testout.txt:

Welcome to java.

4.27 Đọc file trong java với lớp BufferedReader

Lớp BufferedReader trong java được sử dụng để đọc văn bản từ một input stream dựa trên các ký tự (character stream). Nó có thể được sử dụng để đọc dữ liệu theo dòng (line by line) bằng phương thức `readLine()`. Nó giúp hiệu suất nhanh. Nó kế thừa lớp Reader.

Khai báo của lớp BufferedReader

Dưới đây là khai báo của lớp Java.io.BufferedReader:

```
public class BufferedReader extends Reader
```

Các constructor của lớp BufferedReader

Constructor	Mô tả
<code>BufferedReader(Reader rd)</code>	Nó được sử dụng để tạo ra thể hiện của BufferedReader mà sử dụng kích thước mặc định cho một bộ đệm đầu vào.
<code>BufferedReader(Reader rd, int size)</code>	Nó được sử dụng để tạo ra thể hiện của BufferedReader có sử dụng kích thước quy định cho một bộ đệm đầu vào.

Các phương thức của lớp BufferedReader

Phương thức	Mô tả
<code>int read()</code>	Nó được sử dụng để đọc ký tự vật duy nhất.
<code>int read(char[] cbuf, int off, int len)</code>	Nó được sử dụng để đọc các ký tự thành một phần của một mảng.

len)	
boolean markSupported()	Nó được sử dụng để kiểm tra input stream có hỗ trợ các phương thức mark() và reset() không.
String readLine()	Nó được sử dụng để đọc một dòng văn bản.
boolean ready()	Nó được sử dụng để kiểm tra liệu các input stream đã sẵn sàng để được đọc.
long skip(long n)	Nó được sử dụng để bỏ qua n ký tự.
void reset()	Nó định vị lại stream tại vị trí mà phương thức đánh dấu lần cuối được gọi vào input stream này.
void mark(int readAheadLimit)	Nó được sử dụng để đánh dấu vị trí hiện tại trong một stream.
void close()	Nó đóng các dòng đầu vào và giải phóng bất kỳ tài nguyên hệ thống nào liên kết đến stream.

Ví dụ về đọc file trong java với lớp BufferedReader

Trong ví dụ này, chúng ta đọc dữ liệu từ file văn bản **testout.txt** sử dụng lớp Java BufferedReader.

```
import java.io.BufferedReader;
import java.io.FileReader;

public class BufferedReaderExample {
    public static void main(String args[]) throws Exception {
        FileReader fr = new FileReader("D:\\testout.txt");
        BufferedReader br = new BufferedReader(fr);

        int i;
        while ((i = br.read()) != -1) {
            System.out.print((char) i);
        }
    }
}
```

```

        }
        br.close();
        fr.close();
    }
}

```

Giả sử file testout.txt có nội dung như sau:

Welcome to java.

Output:

Welcome to java.

4.27 Đọc file trong java với lớp CharArrayReader

CharArrayReader gồm có hai từ: CharArray và Reader. **Lớp CharArrayReader trong java** được sử dụng để đọc mảng ký tự như là một trình đọc (Reader). Nó kế thừa lớp Reader.

Khai báo của lớp CharArrayReader

Dưới đây là khai báo của lớp Java.io.CharArrayReader:

```
public class CharArrayReader extends Reader
```

Các phương thức của lớp BufferedReader

Phương thức	Mô tả
int read()	Nó được sử dụng để đọc một ký tự duy nhất
int read(char[] b, int off, int len)	Nó được sử dụng để đọc các ký tự là một phần của một mảng.
boolean ready()	Nó được sử dụng để kiểm tra liệu đã sẵn sàng để đọc.
boolean markSupported()	Nó được sử dụng để kiểm tra xem có hỗ trợ phương thức mark() không.
long skip(long n)	Nó được sử dụng để bỏ qua n ký tự trong input stream.
void mark(int readAheadLimit)	Nó được sử dụng để đánh dấu vị trí hiện tại trong the stream.

void reset()	Nó được sử dụng để thiết lập lại stream đến vị trí được đánh dấu gần đây nhất.
void close()	Nó được sử dụng để đóng stream.

Ví dụ về đọc file trong java với lớp CharArrayReader

Hãy xem ví dụ đơn giản để đọc một ký tự sử dụng lớp Java CharArrayReader.

```
import java.ioCharArrayReader;

public class CharArrayExample {
    public static void main(String[] args) throws Exception {
        char[] ary = { 'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't' };
        CharArrayReader reader = new CharArrayReader(ary);
        int k = 0;
        // Read until the end of a file
        while ((k = reader.read()) != -1) {
            char ch = (char) k;
            System.out.print(ch + " : ");
            System.out.println(k);
        }
    }
}
```

Output:

```
j : 106
a : 97
v : 118
a : 97
t : 116
p : 112
o : 111
i : 105
```

n : 110

t : 116

4.28 Ghi file trong java với lớp CharArrayWriter

Lớp CharArrayWriter trong java có thể được sử dụng để ghi dữ liệu chung cho nhiều file. Lớp này thừa kế lớp Writer. Bộ đệm của nó tự động phát triển khi dữ liệu được ghi vào stream này. Gọi phương thức close() đối với đối tượng này không có hiệu lực.

Khai báo của lớp CharArrayWriter

Dưới đây là khai báo của lớp Java.ioCharArrayWriter:

```
public class CharArrayWriter extends Writer
```

Các phương thức của lớp BufferedReader

Method	Description
int size()	Nó được sử dụng để trả về kích thước hiện tại của bộ đệm.
char[] toCharArray()	Nó được sử dụng để trả về bản sao của một dữ liệu đầu vào.
String toString()	Nó được sử dụng để chuyển đổi dữ liệu nhập vào thành một chuỗi.
CharArrayWriter append(char c)	Nó được sử dụng để nối thêm ký tự được chỉ định tới writer.
CharArrayWriter append(CharSequence csq)	Nó được sử dụng để nối thêm chuỗi ký tự được chỉ định tới writer.
CharArrayWriter append(CharSequence csq, int start, int end)	Nó được sử dụng để nối thêm dãy con của một ký tự được chỉ định tới writer.
void write(int c)	Nó được sử dụng để viết một ký tự vào bộ đệm.
void write(char[] c, int off, int len)	Nó được sử dụng để ghi một phần của chuỗi ký tự vào bộ

	đệm.
void write(String str, int off, int len)	Nó được sử dụng để ghi một phần của string vào bộ đệm.
void writeTo(Writer out)	Nó được sử dụng để ghi nội dung của bộ đệm vào stream khác nhau.
void flush()	Nó được sử dụng để xả stream.
void reset()	Nó được sử dụng để thiết lập lại bộ đệm.
void close()	Nó được sử dụng để đóng stream.

Ví dụ về ghi file trong java với lớp CharArrayWriter

Trong ví dụ này, chúng ta ghi một dữ liệu chung cho 4 tệp a.txt, b.txt, c.txt và d.txt.

```
public class CharArrayWriterExample {
    public static void main(String args[]) throws Exception {
        CharArrayWriter out = new CharArrayWriter();
        out.write("Welcome to java.");
        FileWriter f1 = new FileWriter("D:\\a.txt");
        FileWriter f2 = new FileWriter("D:\\b.txt");
        FileWriter f3 = new FileWriter("D:\\c.txt");
        FileWriter f4 = new FileWriter("D:\\d.txt");
        out.writeTo(f1);
        out.writeTo(f2);
        out.writeTo(f3);
        out.writeTo(f4);
        f1.close();
        f2.close();
        f3.close();
        f4.close();
        System.out.println("Success...");
    }
}
```

Output:

Success...

a.txt:

Welcome to java.

b.txt:

Welcome to java.

c.txt:

Welcome to java.

d.txt:

Welcome to java.

4.29 Ghi file trong java với lớp PrintStream

Lớp PrintStream trong java cung cấp các phương thức để ghi dữ liệu vào một stream khác. Lớp PrintStream tự động làm sạch dữ liệu vì vậy không cần gọi phương thức flush(). Hơn nữa, các phương thức của nó không ném ngoại lệ IOException.

Khai báo của lớp PrintStream

Dưới đây là khai báo của lớp Java.io.PrintStream:

```
public class PrintStream extends FilterOutputStream implements Closeable,
Appendable
```

Các phương thức của lớp PrintStream

Method	Description
void print(boolean b)	Nó in giá trị boolean đã chỉ định.
void print(char c)	Nó in giá trị char đã chỉ định.
void print(char[] c)	Nó in các giá trị mảng ký tự được chỉ định.
void print(int i)	Nó in giá trị int được chỉ định.
void print(long l)	Nó in các giá trị long được chỉ định.

void print(float f)	Nó in các giá trị float được chỉ định.
void print(double d)	Nó in các giá trị double được chỉ định.
void print(String s)	Nó in các giá trị string được chỉ định.
void print(Object obj)	Nó in các giá trị object được chỉ định.
void println(boolean b)	Nó in giá trị boolean đã chỉ định và xuống dòng.
void println(char c)	Nó in giá trị char đã chỉ định và xuống dòng.
void println(char[] c)	Nó in giá trị của mảng ký tự đã chỉ định và xuống dòng.
void println(int i)	INó in giá trị int đã chỉ định và xuống dòng.
void println(long l)	Nó in giá trị long đã chỉ định và xuống dòng.
void println(float f)	Nó in giá trị float đã chỉ định và xuống dòng.
void println(double d)	Nó in giá trị double đã chỉ định và xuống dòng.
void println(String s)	Nó in giá trị string đã chỉ định và xuống dòng..
void println(Object obj)	Nó in giá trị object đã chỉ định và xuống dòng..
void println()	Nó chỉ xuống dòng.
void printf(Object format, Object... args)	Nó ghi chuỗi định dạng cho stream hiện tại.
void printf(Locale l, Object format, Object... args)	Nó ghi chuỗi định dạng cho stream hiện tại theo vị trí (Locale).
void format(Object format, Object... args)	Nó ghi chuỗi định dạng cho stream hiện tại sử dụng định dạng đã chỉ định.
void format(Locale l, Object format, Object... args)	Nó ghi chuỗi định dạng cho stream hiện tại theo vị trí (Locale) sử dụng định dạng đã chỉ định.

Ví dụ về ghi file trong java với lớp PrintStream

Trong ví dụ này, chúng ta chỉ đơn giản là ghi vào file số nguyên và giá trị chuỗi.

```
import java.io.FileOutputStream;
import java.io.PrintStream;

public class PrintStreamExample {
    public static void main(String args[]) throws Exception {
        FileOutputStream fout = new FileOutputStream("D:\\testout.txt ");
        PrintStream pout = new PrintStream(fout);
        pout.println(2016);
        pout.println("Hello Java");
        pout.println("Welcome to Java");
        pout.close();
        fout.close();
        System.out.println("Success...");
    }
}
```

Output:

```
Success...
```

Nội dung file **testout.txt**:

```
2016
```

```
Hello Java
```

```
Welcome to Java
```

4.30 Ghi file trong java với lớp PrintWriter

Lớp PrintWriter trong java là bản cài đặt của lớp Writer. Nó được sử dụng để ghi các định dạng đại diện của các đối tượng vào stream hướng văn bản.

Khai báo của lớp PrintWriter

Dưới đây là khai báo của lớp Java.io.PrintWriter:

```
public class PrintWriter extends Writer
```

Các phương thức của lớp PrintWriter

Phương thức	Mô tả
void println(boolean x)	Nó được sử dụng để in giá trị boolean.
void println(char[] x)	Nó được sử dụng để in một mảng các ký tự.
void println(int x)	Nó được sử dụng để in giá trị int.
PrintWriter append(char c)	Nó được sử dụng để nối thêm ký tự được chỉ định với writer.
PrintWriter append(CharSequence ch)	Nó được sử dụng để nối thêm chuỗi ký tự được chỉ định với writer.
PrintWriter append(CharSequence ch, int start, int end)	Nó được sử dụng để nối thêm một dãy con của ký tự đã chỉ định với writer.
boolean checkError()	Nó được sử dụng để xả stream và kiểm tra tình trạng lỗi của nó.
protected void setError()	Nó được sử dụng để chỉ ra rằng một lỗi xảy ra.
protected void clearError()	Nó được sử dụng để xóa trạng thái lỗi của một stream.
PrintWriter format(String format, Object... args)	Nó được sử dụng để viết một chuỗi định dạng tới writer bằng cách sử dụng các đối số được chỉ định và định dạng chuỗi.
void print(Object obj)	Nó được sử dụng để in một object.
void flush()	Nó được sử dụng để xả stream.
void close()	Nó được sử dụng để đóng stream.

Ví dụ về ghi file trong java với lớp PrintWriter

Hãy xem ví dụ đơn giản để ghi dữ liệu trên **console** và trong **file văn bản testout.txt** sử dụng lớp Java PrintWriter.

```
import java.io.File;
import java.io.PrintWriter;

public class PrintWriterExample {
    public static void main(String[] args) throws Exception {
        // Dữ liệu được ghi trên Console sử dụng lớp PrintWriter
        PrintWriter writer = new PrintWriter(System.out);
        writer.write("Plpsoft.Vn: ");
        writer.flush();
        writer.close();

        // Dữ liệu được ghi vào File sử dụng PrintWriter
        PrintWriter writer1 = null;
        writer1 = new PrintWriter(new File("D:\\testout.txt"));
        writer1.write("Java, Spring, Hibernate, Android, PHP, ...");
        writer1.flush();
        writer1.close();
    }
}
```

Output trên console:

```
Plpsoft.Vn:
testout.txt:
Java, Spring, Hibernate, Android, PHP, ...
```

4.31 Lớp PushbackInputStream trong java

Lớp PushbackInputStream trong java ghi đè các phương thức của lớp InputStream và cung cấp thêm chức năng mở rộng cho một input stream khác. Nó có thể unread một byte đã được đọc và đẩy trở lại một byte.

Khai báo của lớp PushbackInputStream

Dưới đây là khai báo của lớp Java.io.PushbackInputStream:

```
public class PushbackInputStream extends FilterInputStream
```

Các phương thức của lớp PushbackInputStream

Phương thức	Mô tả
int available()	Nó được sử dụng để trả về số byte có thể được đọc từ input stream.
int read()	Nó được sử dụng để đọc byte kế tiếp của dữ liệu từ input stream.
boolean markSupported()	Nó được sử dụng để kiểm tra xem các phương thức mark() và reset() có được hỗ trợ không.
void mark(int readlimit)	Nó được sử dụng để đánh dấu vị trí hiện tại trong input stream.
long skip(long x)	Nó được sử dụng để bỏ qua và loại bỏ x các byte dữ liệu.
void unread(int b)	Nó được sử dụng để đẩy trở lại byte bằng cách sao chép nó vào bộ đệm pushback.
void unread(byte[] b)	Nó được sử dụng để đẩy trở lại mảng của các byte bằng cách sao chép nó vào bộ đệm pushback.
void reset()	Nó được sử dụng để thiết lập lại input stream.
void close()	Nó được sử dụng để đóng input stream.

Ví dụ về lớp PushbackInputStream trong java

```

import java.io.ByteArrayInputStream;
import java.io.PushbackInputStream;

public class PushbackInputStreamExample {
    public static void main(String[] args) throws Exception {
        String srg = "1##2#34##12";
        byte[] byteArr = srg.getBytes();
        ByteArrayInputStream array = new ByteArrayInputStream(byteArr);
        PushbackInputStream push = new PushbackInputStream(array);
    }
}

```

```

        int i;

        while ((i = push.read()) != -1) {
            if (i == '#') {
                int j;

                if ((j = push.read()) == '#') {
                    System.out.print("**");
                } else {
                    push.unread(j);
                    System.out.print((char) i);
                }
            } else {
                System.out.print((char) i);
            }
        }
    }
}

```

Output:

```
1**2#34**#12
```

4.32 Lớp PushbackReader trong java

Lớp PushbackReader trong java ghi đè các phương thức của lớp FilterReader và cung cấp thêm các chức năng mở rộng. Nó được sử dụng để đọc một luồng ký tự và có thể đẩy trở lại một ký tự vào stream.

Khai báo của lớp PushbackReader

Dưới đây là khai báo của lớp Java.io.PushbackReader:

```
public class PushbackReader extends FilterReader
```

Các phương thức của lớp PushbackReader

Phương thức	Mô tả
int read()	Nó được sử dụng để đọc một ký tự duy nhất.
void mark(int readAheadLimit)	Nó được sử dụng để đánh dấu vị trí hiện tại trong một stream.

boolean ready()	Nó được sử dụng để kiểm tra liệu luồng đã sẵn sàng để được đọc.
boolean markSupported()	Nó được sử dụng để biết liệu các stream hỗ trợ phương thức mark().
long skip(long n)	Nó được sử dụng để bỏ qua n ký tự.
void unread (int c)	Nó được sử dụng để đẩy trở lại ký tự bằng cách sao chép nó vào bộ đệm pushback.
void unread (char[] cbuf)	Nó được sử dụng để đẩy trở lại mảng các ký tự bằng cách sao chép nó vào bộ đệm pushback.
void reset()	Nó được sử dụng để thiết lập lại stream.
void close()	Nó được sử dụng để đóng stream.

Ví dụ về lớp PushbackReader trong java

```

import java.ioCharArrayReader;
import java.io.PushbackReader;

public class PushbackReaderExample {
    public static void main(String[] args) throws Exception {
        char ary[] = { '1', '-', '-', '2', '-', '3', '4', '-', '-', '-',
        '5', '6' };
        CharArrayReader reader = new CharArrayReader(ary);
        PushbackReader push = new PushbackReader(reader);
        int i;
        while ((i = push.read()) != -1) {
            if (i == '-') {
                int j;
                if ((j = push.read()) == '-') {
                    System.out.print("#*");
                } else {

```

```
        push.unread(j); // push back single character
        System.out.print((char) i);
    }
} else {
    System.out.print((char) i);
}
}
```

Output:

1#*2-34#*-56

4.33 Ghi file trong java với lớp StringWriter

Lớp StringWriter trong java là một character stream thu thập dữ liệu từ bộ đệm chuỗi, có thể được sử dụng để xây dựng một chuỗi. Lớp StringWriter kế thừa lớp Writer.

Trong lớp StringWriter, các tài nguyên hệ thống như các network socket và file không được sử dụng, do đó việc đóng StringWriter là không cần thiết.

Khai báo của lớp StringWriter

Dưới đây là khai báo của lớp `java.io.StringWriter`:

```
public class StringWriter extends Writer
```

Các phương thức của lớp StringWriter

Phương thức	Mô tả
void write(int c)	Nó được sử dụng để ghi một ký tự.
void write(String str)	Nó được sử dụng để ghi một chuỗi ký tự.
void write(String str, int off, int len)	Nó được sử dụng để ghi một phần của một chuỗi.
void write(char[] cbuf, int off, int len)	Nó được sử dụng để ghi phần của một mảng các ký tự.

String toString()	Nó được sử dụng để trả về giá trị hiện tại của bộ đệm như một chuỗi.
StringBuffer getBuffer()	Nó được sử dụng trả về bộ đệm chuỗi.
StringWriter append(char c)	Nó được sử dụng để nối thêm ký tự được chỉ định tới writer.
StringWriter append(CharSequence csq)	Nó được sử dụng để nối thêm chuỗi ký tự được chỉ định tới writer.
StringWriter append(CharSequence csq, int start, int end)	Đó được sử dụng để nối thêm dãy con của dãy ký tự được chỉ định vào writer.
void flush()	Nó được sử dụng để xả (làm sạch) stream.
void close()	Nó được sử dụng để đóng stream.

Ví dụ về ghi file trong java với lớp StringWriter

Hãy xem ví dụ đơn giản của StringWriter sử dụng BufferedReader để ghi dữ liệu vào file "**testout.txt**".

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.StringWriter;

public class StringWriterExample {
    public static void main(String[] args) throws IOException {
        char[] arr = new char[512];
        StringWriter writer = new StringWriter();
        FileInputStream input = null;
        BufferedReader buffer = null;
        input = new FileInputStream("D://testout.txt");
        buffer = new BufferedReader(new InputStreamReader(input, "UTF-8"));
        while (true) {
            int i = buffer.read();
            if (i == -1)
                break;
            arr[i] = (char) i;
        }
        writer.write(arr);
        writer.close();
    }
}
```

```

        int x;

        while ((x = buffer.read(arr)) != -1) {
            writer.write(arr, 0, x);
        }

        System.out.println(writer.toString());
        writer.close();
        buffer.close();
    }
}

```

testout.txt:

Welcome to java.

Output:

Welcome to java.

4.34 Đọc file trong java với lớp StringReader

Lớp StringReader trong java là một character stream với chuỗi như một nguồn dữ liệu. Nó lấy một chuỗi đầu vào và thay đổi nó vào character stream. Nó kế thừa lớp Reader.

Trong lớp StringReader, các tài nguyên hệ thống như các network socket và các file không được sử dụng, do đó việc đóng StringReader là không cần thiết.

Khai báo của lớp StringReader

Dưới đây là khai báo của lớp Java.io.StringReader:

```
1 public class StringReader extends Reader
```

Các phương thức của lớp StringReader

Phương thức	Mô tả
int read()	Nó được sử dụng để đọc một ký tự duy nhất.
int read(char[] cbuf, int off, int len)	Nó được sử dụng để đọc một ký tự vào một phần của một mảng.
boolean ready()	Nó được sử dụng để kiểm tra liệu stream đã sẵn sàng để được đọc.

boolean markSupported()	Nó được sử dụng để kiểm tra liệu hỗ trợ các stream có hỗ trợ phương thức mark() không.
long skip(long n)	Nó được sử dụng để bỏ qua n ký tự được chỉ định trong một stream
void mark(int readAheadLimit)	Nó được sử dụng để đánh dấu vị trí hiện tại trong một stream.
void reset()	Nó được sử dụng để thiết lập lại stream.
void close()	Nó được sử dụng để đóng stream.

Ví dụ về đọc file trong java với lớp StringReader

```
import java.io.StringReader;

public class StringReaderExample {
    public static void main(String[] args) throws Exception {
        String srg = "Hello Java! \nWelcome to java.";
        StringReader reader = new StringReader(srg);
        int k = 0;
        while ((k = reader.read()) != -1) {
            System.out.print((char) k);
        }
    }
}
```

Output:

```
Hello Java!
Welcome to java.
```

Java Serialization

Serialization trong java

1. Serialization trong java

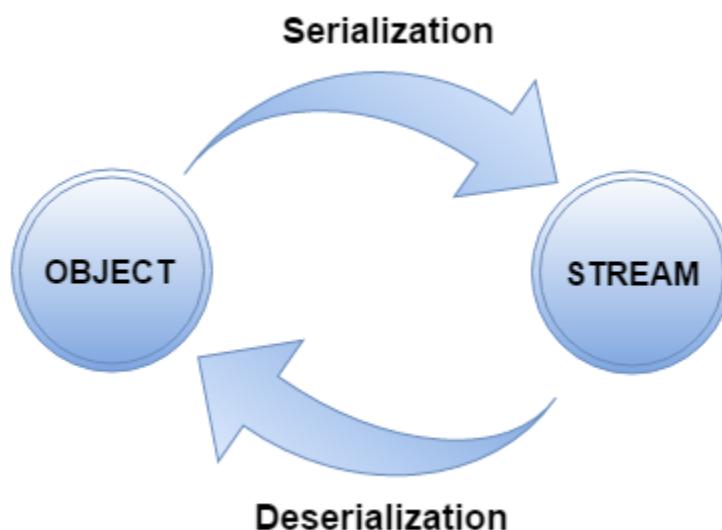
Tuần tự hoá trong java hay serialization trong java là một cơ chế để ghi trạng thái của một đối tượng vào một byte stream.

Nó chủ yếu được sử dụng trong các công nghệ Hibernate, RMI, JPA, EJB và JMS.

Hoạt động ngược lại của serialization được gọi là deserialization.

Ưu điểm của Serialization trong java

Nó chủ yếu được sử dụng để truyền trạng thái của đối tượng qua mạng (được biết đến như marshaling).



java.io.Serializable interface

Serializable là một giao diện đánh dấu (không có thành viên dữ liệu và phương thức). Nó được sử dụng để "đánh dấu" các lớp java để các đối tượng của các lớp này có thể nhận được khả năng nhất định. Cloneable và Remote cũng là những interface đánh dấu.

Nó phải được implements bởi lớp mà đối tượng của nó bạn muốn persist, bạn có thể xem thêm về [đối tượng persistent là gì?](#)

Lớp String và tất cả các lớp wrapper implements giao tiếp java.io.Serializable theo mặc định.

Hãy xem ví dụ dưới đây:

```
import java.io.Serializable;

public class Student implements Serializable {
    int id;
    String name;

    public Student(int id, String name) {
        this.id = id;
        this.name = name;
    }
}
```

Trong ví dụ trên, lớp Student implements giao tiếp Serializable. Bây giờ các đối tượng của nó có thể được chuyển đổi thành stream.

Tìm hiểu bài học về lớp [ObjectOutputStream trong java](#) để hiểu rõ hơn về serialization trong java.

2. Deserialization trong java

Deserialization là quá trình tái thiết lại các đối tượng từ trạng thái serialized. Đây là hoạt động ngược lại của serialization.

Tìm hiểu bài học về lớp [ObjectInputStream trong java](#) để hiểu rõ hơn về serialization trong java.

3. Java Serialization với thừa kế (Mối quan hệ IS-A)

Nếu một lớp implements giao tiếp Serializable thì tất cả các lớp con của nó cũng sẽ được Serializable. Hãy xem ví dụ dưới đây:

```
public class Person {
    int id;
    String name;

    Person(int id, String name) {
        this.id = id;
        this.name = name;
    }
}
```

```
public class Student extends Person {  
    String course;  
    int fee;  
  
    public Student(int id, String name, String course, int fee) {  
        super(id, name);  
        this.course = course;  
        this.fee = fee;  
    }  
}
```

Các bạn hãy thực hành ghi và đọc với lớp ObjectOutputStream trong java và lớp ObjectInputStream trong java về Java Serialization với thừa kế nhé.

4. Java Serialization với sự kết hợp (Mối quan hệ HAS-A)

Nếu một lớp có một tham chiếu của một lớp khác, tất cả các tham chiếu phải được implements giao tiếp Serializable nếu không quá trình serialization sẽ không được thực hiện. Trong trường hợp đó, NotSerializableException được ném ra khi chạy.

```
public class Address {  
    String addressLine, city, state;  
  
    public Address(String addressLine, String city, String state) {  
        this.addressLine = addressLine;  
        this.city = city;  
        this.state = state;  
    }  
}  
  
import java.io.Serializable;  
  
public class Student implements Serializable {  
    int id;  
    String name;  
    Address address;// HAS-A
```

```

public Student(int id, String name) {
    this.id = id;
    this.name = name;
}
}

```

Vì Address không implements giao tiếp Serializable nên bạn không thể serialize thẻ hiện của lớp Student.

Lưu ý: Tất cả các đối tượng trong một đối tượng phải được implements giao tiếp Serializable.

5. Java Serialization với thành viên dữ liệu static

Nếu có bất kỳ thành viên dữ liệu static trong một lớp, nó sẽ không được serialized bởi vì static là một phần của lớp chứ không phải đối tượng.

[?](#)

```

import java.io.Serializable;

public class Employee implements Serializable {
    int id;
    String name;
    static String company = "Plpsoft";// it won't be serialized

    public Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }
}

```

6. Java Serialization với array hoặc collection

Quy tắc: Trong trường hợp mảng hoặc bộ sưu tập, tất cả các đối tượng của array hoặc collection phải được tuân tự hóa. Nếu bất kỳ đối tượng không phải là serializable, serialization sẽ không thành công.

7. Externalizable trong java

Giao tiếp Externalizable cung cấp khả năng viết trạng thái của một đối tượng vào một byte stream ở định dạng nén. Nó không phải là một giao diện đánh dấu.

Giao tiếp Externalizable cung cấp hai phương thức:

- **public void writeExternal(ObjectOutput out) throws IOException**

- `public void readExternal(ObjectInput in) throws IOException`

8. Từ khóa transient trong java

Nếu bạn không muốn serialize bất kỳ thành viên dữ liệu của một lớp học, bạn có thể đánh dấu nó với từ khóa transient

Xem bài học tiếp theo để biết thêm chi tiết.

Lớp ObjectOutputStream trong java

Lớp ObjectOutputStream trong java được sử dụng để ghi các kiểu dữ liệu nguyên thuỷ và các đối tượng Java vào một OutputStream. Chỉ có các đối tượng implements giao tiếp `java.io.Serializable` mới có thể được ghi vào stream.

Khai báo của lớp ObjectOutputStream

Dưới đây là khai báo của lớp `Java.io.ObjectOutputStream`:

```
public class ObjectOutputStream
    extends OutputStream implements ObjectOutput, ObjectStreamConstants
```

Constructor của lớp ObjectOutputStream

Constructor	Mô tả
<code>public ObjectOutputStream(OutputStream out)</code>	Tạo một ObjectOutputStream ghi vào OutputStream được chỉ định.

Các phương thức của lớp ObjectOutputStream

Phương thức	Mô tả
<code>public final void writeObject(Object obj)</code>	Ghi một đối tượng được chỉ định tới ObjectOutputStream.
<code>public void flush()</code>	Làm sạch ObjectInputStream hiện tại.
<code>public void close()</code>	Đóng ObjectInputStream hiện tại.

Ví dụ về lớp ObjectOutputStream trong java

File: Student.java

```
import java.io.Serializable;

public class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private String address;
    private int age;

    public void Studet() {}

    public Student(int id, String name, String address, int age) {
        super();
        this.id = id;
        this.name = name;
        this.address = address;
        this.age = age;
    }

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getAddress() {
```

```
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String toString() {
        return "Student@[id=" + id + ", name=" + name + ", "
            + "address= " + address + ", age =" + age+ "]";
    }
}
```

File: ObjectOutputStreamExample.java

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class ObjectOutputStreamExample {
    public static void main(String[] args) throws IOException {
        ObjectOutputStream oos = null;
        try {
            oos = new ObjectOutputStream(new
FileOutputStream("D:\\testout.txt"));
            // create student
            Student student = new Student(1, "Tran Hao Phong", "Ha Noi",
17);
            // write student
            oos.writeObject(student);
            System.out.println("Success...");
        }
```

```

        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {
            oos.close();
        }
    }
}

```

Output:

Success...

testout.txt:

Lớp ObjectInputStream trong java

Lớp ObjectInputStream trong java được sử dụng để đọc các đối tượng và dữ liệu nguyên thủy mà được ghi bằng việc sử dụng lớp ObjectOutputStream.

Khai báo của lớp ObjectInputStream

Dưới đây là khai báo của lớp Java.io.ObjectInputStream :

```

public class ObjectInputStream
    extends InputStream implements ObjectInput, ObjectStreamConstants

```

Constructor của lớp ObjectInputStream

Constructor	Mô tả
public ObjectInputStream(InputStream in)	Tạo ra một ObjectInputStream đọc từ InputStream đã chỉ định.

Các phương thức của lớp ObjectInputStream

Phương thức	Mô tả
public final Object readObject()	Đọc một đối tượng từ input stream.

public void close()	Đóng ObjectOutputStream hiện tại.
---------------------	-----------------------------------

Ví dụ về lớp ObjectInputStream trong java

Trong ví dụ này, chúng ta sẽ tuần tự hóa đối tượng của lớp Student. Phương thức writeObject() của lớp ObjectOutputStream cung cấp chức năng để tuần tự hóa đối tượng. Chương trình dưới đây sẽ lưu trạng thái của đối tượng trong tệp có tên testout.txt.

Tạo lớp Student.java

```
import java.io.Serializable;

public class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private String address;
    private int age;

    public void Studet() {}

    public Student(int id, String name, String address, int age) {
        super();
        this.id = id;
        this.name = name;
        this.address = address;
        this.age = age;
    }

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
}
```

```
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getAddress() {
    return address;
}

public void setAddress(String address) {
    this.address = address;
}

public int getAge() {
    return age;
}

public void setAge(int age) {
    this.age = age;
}

public String toString() {
    return "Student@[id=" + id + " , name=" + name + ", "
           + "address= " + address + ",age =" + age+ "]";
}
```

File: ObjectInputStreamExample.java

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class ObjectInputStreamExample {
    public static void main(String[] args) throws IOException {
        ObjectInputStream ois = null;

        try {
            ois = new ObjectInputStream(new FileInputStream("D:\\testout.txt"));
            // read student
```

```

        Student student = (Student) ois.readObject();
        // show student
        System.out.println(student.toString());
    } catch (ClassNotFoundException ex) {
        ex.printStackTrace();
    } catch (IOException ex) {
        ex.printStackTrace();
    } finally {
        ois.close();
    }
}

```

Giả sử file testout.txt là output của ví dụ trong bài [lớp ObjectOutputStream trong java](#) có nội dung như sau:

Output:

```
Student@[id=1 , name=Tran Hao Phong, address= Ha Noi,age =17]
```

Từ khóa transient trong java

Từ khóa transient trong java được sử dụng trong serialization. Nếu bạn định nghĩa bất kỳ thành viên dữ liệu nào là transient, nó sẽ không được đánh dấu là tuần tự (serialize).

Ví dụ, khai báo một lớp Student, có ba thành viên dữ liệu là id, name và age. Lớp Student implements giao tiếp Serializable, tất cả các giá trị sẽ được tuần tự nhưng nếu bạn không muốn tuần tự cho một giá trị, ví dụ: age thì bạn có thể khai báo age với từ khóa transient.

Ví dụ về từ khóa transient trong java

Trong ví dụ này, chúng ta tạo ra hai lớp Student và PersistExample. Thành viên dữ liệu của lớp Student được khai báo với từ khóa transient, giá trị của nó sẽ không được tuần tự.

Khi đối tượng được giải mã tuần tự hóa, bạn sẽ nhận được giá trị mặc định của kiểu dữ liệu của biến được khai báo với từ khóa transient.

Tạo lớp Student với biến age được khai báo với từ khóa transient.

```
import java.io.Serializable;
```

```
public class Student implements Serializable {
    int id;
    String name;
    transient int age;

    public Student(int id, String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }
}
```

Bây giờ tạo lớp PersistExample và sử dụng ObjectOutputStream để ghi đối tượng student vào file **student.txt**

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class PersistExample {
    public static void main(String args[]) throws Exception {
        ObjectOutputStream oos = null;
        try {
            // tạo object ObjectOutputStream
            oos = new ObjectOutputStream(new
FileOutputStream("student.txt"));
            // tạo object student
            Student student = new Student(1001, "Tran Van A", 22);
            // ghi student vào file
            oos.writeObject(student);
            oos.flush();
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {
            oos.close();
        }
        System.out.println("success...");
    }
}
```

```
    }  
}
```

Output:

```
success...
```

Bây giờ tạo lớp DePersist để đọc đối tượng student được ghi vào file **student.txt** ở trên.

```
import java.io.FileInputStream;  
import java.io.IOException;  
import java.io.ObjectInputStream;  
  
public class DePersist {  
    public static void main(String args[]) throws Exception {  
        ObjectInputStream ois = null;  
        try {  
            ois = new ObjectInputStream(new  
FileInputStream("student.txt"));  
            Student student = (Student) ois.readObject();  
            System.out.println(student.id + " " + student.name + " " +  
student.age);  
        } catch (IOException ex) {  
            ex.printStackTrace();  
        } finally {  
            ois.close();  
        }  
    }  
}
```

Output:

```
1001 Tran Van A 0
```

Ví dụ Java IO

(Xem chi tiết quyển bài tập)

Bài này cung cấp các ví dụ Java IO. Java cung cấp các lớp i/o để hỗ trợ việc input và output thông qua byte stream, character stream, buffered stream và hệ thống file. Dưới đây là danh sách các ví dụ về java i/o bao gồm thao tác với file, file tạm thời, thư mục, đường dẫn.

Thao tác với file trong java

Tạo file trong java

Đọc file với BufferedInputStream trong java

Đọc file với BufferedReader trong java

Đọc ghi file trong java

Append nội dung vào file trong java

Delete file trong java

Delete nhiều file dựa vào phần mở rộng trong java

Tìm các file dựa vào phần mở rộng trong java

Đổi tên file trong java

Copy file trong java

Move file trong java

Check file tồn tại trong java

Check file ẩn trong java

Liệt kê tất cả file và folder trong một folder

Get thư mục hiện tại trong Java

Read properties file trong java

Thao tác với folder trong java

Check thư mục rỗng trong java

Tạo thư mục trong java

Xóa thư mục trong java

Copy thư mục trong java

Nén file trong java

ZIP file trong java

TÀI LIỆU THAM KHẢO

- [1] Giáo trình, bài giảng “Kỹ năng lập trình ứng dụng với Java”, Ths. Võ Văn Phúc, Trường đại học Nam Cần Thơ
- [2] Giáo trình, bài giảng “Lập trình Java căn bản”, Ths. Võ Văn Phúc, Trường đại học Nam Cần Thơ
- [3] Allen B. Downey & Chris Mayfield, Think Java, 2016
- [4] Lập Trình Hướng Đối Tượng Với Java – Đoàn Văn Ban, NXB KHOA HỌC VÀ KỸ THUẬT, 2005
- [5] Java Cơ Bản – Nguyễn Văn Khoa, NXB Hồng Đức, 2007
- [6] Java Core (Tiếng Việt), UDS Ebook, updateSoft.
- [7] Slide Lập Trình Java - Phạm Quang Dũng, ĐH KHTN
- [8] Lập trình hướng đối tượng – ĐH Công nghệ, ĐHQGHN
- [9] Lập Trình Java: <https://viettuts.vn/java>

Tài liệu tham khảo khác:

- [10] Cay S. Horstmann, *Core Java Volume I*, Prentice Hall, 2016.
- [11] Java Tutorial, <https://www.tutorialspoint.com/java/index.htm>
- [12] Java cơ bản (o7planning.org): <https://o7planning.org/vi/10973/java-co-ban>
- [13] Java cơ bản (vncoder.vn) <https://vncoder.vn/java/lap-trinh-java-co-ban>
- [14] Lập trình Java: <https://yellowcodebooks.com/category/lap-trinh-java/>
- [15] Java nâng cao (o7planning.org):
<https://o7planning.org/vi/10985/java-nang-cao>