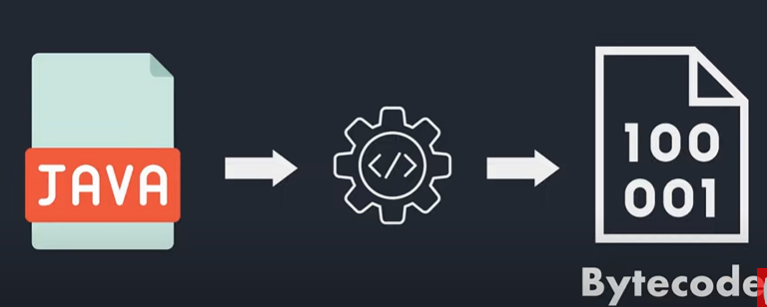
**LEARN JAVA**

# **JVM , JRE , JDK**

JVM, JRE và JDK là ba khái niệm quan trọng trong ngôn ngữ lập trình Java mà bất kỳ lập trình viên nào cũng cần biết.

JVM (Java Virtual Machine) là một máy ảo, chịu trách nhiệm chuyển đổi bytecode (được biên dịch từ mã nguồn Java) thành mã máy của hệ thống cụ thể, giúp ứng dụng Java có thể chạy trên nhiều nền tảng khác nhau mà không cần chỉnh sửa. Giúp chạy trên bất kì hệ điều hành nào.



JRE (Java Runtime Environment) cung cấp môi trường cần thiết để thực thi ứng dụng Java, bao gồm JVM và các thư viện đi kèm.

JDK (Java Development Kit) không chỉ chứa JRE mà còn bổ sung các công cụ phát triển như trình biên dịch, giúp lập trình viên có thể viết và biên dịch ứng dụng Java một cách hiệu quả.

# **Autoboxing/Unboxing (Object wrapper class)**

"Boxing": convert từ primitive value (giá trị nguyên thủy) => chuyển thành object (wrapper class)

"unboxing": là quá trình ngược lại, từ object => giá trị nguyên thủy

=> Công việc Boxing/Unboxing do compiler "tự động làm" (nên gọi là autoboxing/unboxing :v)

Link tham khảo:

[**https://docs.oracle.com/javase/tutorial/java/data/autoboxing.html**](https://docs.oracle.com/javase/tutorial/java/data/autoboxing.html)

# **Access Modifier (private/public/protected)**

Access Modifier : quyền truy cập sử dụng/sửa đổi

Trong java, có 4 thuộc tính hay dùng nhất:

- private: chỉ được sử dụng trong nội bộ class

+ public : công khai hoàn toàn (100%)

(default): là public đối với class trong cùng package. là private khi khác package

# protected: được sử dụng với lớp cha/con (tính chất kế thừa)

=> public và private là 2 thuộc tính được sử dụng nhiều nhất

Mức độ che dấu tăng dần theo mũi tên: public -> protected -> {default} -> private

# **Encapsulation (tính đóng gói)**

Để tăng tính bảo mật cho dữ liệu, bạn cần:

- hạn chế quyền "modify" data (java hỗ trợ như private, protected, public )

- che dấu data : public/private...

Thường ta sẽ sử dụng getter // setter : giúp che dấu code

**Lợi ích :**

-Hạn chế được các truy xuất không hợp lệ đến các thuộc tính của đối tượng

-Giúp cho trạng thái của các đối tượng luôn đúng

- Giúp ẩn đi những thông tin không cần thiết về đối tượng

- Cho phép thay đổi cấu trúc bên trong lớp mà không ảnh hưởng tới lớp khác.

# **Polymorphism (Tính đa hình)**

<https://www.javatpoint.com/runtime-polymorphism-in-java>

<https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html>

A screenshot of a computer program

Description automatically generated

Tính Đa Hình hay còn gọi là Polymorphism là một khái niệm quan trọng trong lập trình hướng đối tượng cho phép các đối tượng của các loại khác nhau được xử lý như là các đối tượng của một superclass (lớp cha) chung. Điều này có nghĩa là bạn có thể viết code chung và có thể tái sử dụng được với các đối tượng khác nhau mà không cần biết loại cụ thể tại compile time.

Một trong những lợi ích chính của tính đa hình là nó cho phép bạn viết code linh hoạt hơn. Bạn có thể tạo các lớp mới kế thừa từ một superclass chung và triển khai phiên bản riêng của các phương thức, mà có thể được sử dụng tương đương với các đối tượng của superclass. Điều này giúp dễ dàng thêm chức năng mới vào code của bạn mà không cần thay đổi mã hiện có.

Tổng thể, tính đa hình là một khái niệm căn bản trong lập trình hướng đối tượng và rất quan trọng để tạo ra mã có tính mô-đun, dễ bảo trì và có khả năng mở rộng.

Cha cha = new Con();

# **Throw / throws**

**THROW :** được sử dụng để tạo ra một ngoại lệ cụ thể và đẩy nó ra khỏi phương thức hiện tại.

-Viết trong hàm

-Ném một lỗi

* Dùng để tự ném ngoại lệ

A computer screen with text

Description automatically generated

**THROWS** : được sử dụng khi khai báo một phương thức để nói rằng phương thức có thể gây ra một hoặc nhiều loại ngoại lệ. (cuối đầu né tránh)

-Viết ngoài hàm

-Ném nhiều lỗi

<https://docs.oracle.com/javase/tutorial/essential/exceptions/declaring.html>

<https://stackoverflow.com/questions/18491020/what-does-throws-do-and-how-is-it-helpful>

# **Super**

* Sử dụng keyword super để truy cập tới thuộc tính và method của cha
* Giúp class con lấy được hàm của class cha = super.tenHam()

Lưu ý : super luôn gọi tới class cha.

-Trong mối quan hệ kế thừa thì super() ⬄ constructor lớp cha không có tham số

- Mặc định sẽ tự tạo.

# **Method Overriding / Orverloading**

**+ Overriding ( ghi đè phương thức)**

***Khái niệm :***

Method overriding xảy ra khi một lớp con cung cấp một triển khai cụ thể của một phương thức đã được định nghĩa trong lớp cha của nó.

***Điều kiện :***

-Phương thước trong lớp con phải cung tên, cùng kiểu trả về và cùng danh sách tham số như phương thức trong lớp cha

-Lớp con phải sử dụng từ khóa @Override trong Java để chú thích đó là phương thức ghi đè

***Mục đích:***

-Để thay đổi hoặc mở rộng chức năng của phương thức được thừa kế từ lớp cha

-Tạo ra một phiên bản cụ thể của phương thức cho lớp con.

**+ Method Overloading (Nạp chồng phương thức)**

***Định nghĩa :***

Method overloading xảy ra khi trong cùng một lớp, có nhiều phương thức có cùng tên nhưng khác nhau về danh sách tham số (số lượng, kiểu dữ liệu hoặc thứ tự tham số).

***Điều kiện :***

Các phương thức phải có cùng tên nhưng khác nhau về danh sách tham số.

Các phương thức có thể khác nhau về kiểu trả về, nhưng điều này không đủ để phân biệt chúng.

**Mục đích**:

Tạo ra các phiên bản khác nhau của một phương thức để xử lý các loại tham số hoặc số lượng tham số khác nhau.

**Tóm lại :**

* **Overriding**: Xảy ra giữa các lớp có quan hệ kế thừa (lớp cha và lớp con).
* **Overloading**: Xảy ra trong cùng một lớp.

# **Abstract- Tính trừu tượng**

Tài liệu tham khảo :

<https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html>

Lưu ý :

-Lớp trừu tượng không thể khởi tạo hay không thể new.

Trong Java, từ khóa abstract được sử dụng để khai báo một lớp hoặc một phương thức là trừu tượng

**Lớp Trừu Tượng (Abstract Class)**

1. **Định nghĩa**:
   * Lớp trừu tượng là một lớp không thể được khởi tạo và thường được sử dụng để cung cấp một cơ sở cho các lớp con.
2. **Đặc điểm**:
   * Lớp trừu tượng có thể có các phương thức trừu tượng (phương thức không có thân) và các phương thức thông thường (phương thức có thân).
   * Một lớp trừu tượng có thể có các thuộc tính và phương thức như các lớp thông thường.
3. **Mục đích**:
   * Để tạo ra một lớp cơ sở mà các lớp con có thể kế thừa và triển khai các phương thức trừu tượng theo cách cụ thể của riêng mình.

**Phương Thức Trừu Tượng (Abstract Method)**

1. **Định nghĩa**:
   * Phương thức trừu tượng là một phương thức được khai báo mà không có thân phương thức (body). Nó chỉ có phần định nghĩa (signature) và được các lớp con triển khai cụ thể.
2. **Đặc điểm**:
   * Phương thức trừu tượng phải được khai báo trong một lớp trừu tượng.
   * Các lớp con kế thừa từ lớp trừu tượng phải triển khai tất cả các phương thức trừu tượng hoặc chúng cũng phải được khai báo là trừu tượng.
3. **Mục đích**:
   * Để ép buộc các lớp con phải cung cấp một triển khai cụ thể cho các phương thức trừu tượng.

# **Câu hỏi phỏng vấn**

**Abstract khác với Interface như thế nào ?**

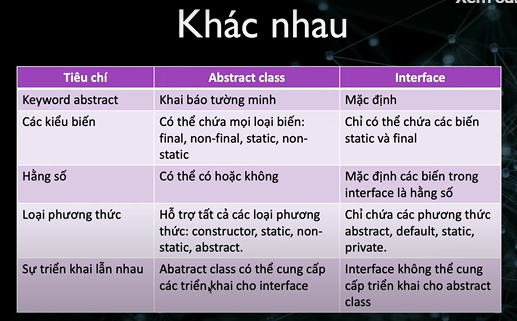
Giống nhau :

-Đều được sử dụng để đạt được tính trừu tượng trong OOP

-Đều sử dụng trong kế thừa

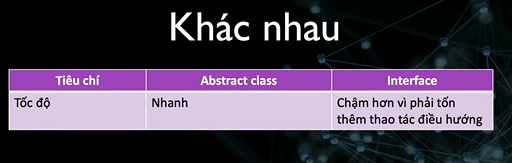
-Đều không thể tạo đối tượng với keyword new.

Khác nhau:



A screenshot of a computer screen

Description automatically generated

\

Abstract class dùng để TỔNG QUÁT (Generalize) hóa các hành vi của các đối tượng. Interface dùng để CHUẨN HÓA (Standardize) hành vi của các đối tượng.

**Khi nào sử dụng abstract hay interface?**

**Abstract :**

Nếu bạn muốn có một số phương thức được định nghĩa sẵn trong lớp cha và một số phương thức khác phải được cài đặt trong các lớp con, bạn nên sử dụng lớp trừu tượng.

Lớp trừu tượng cho phép bạn khai báo các biến thành viên và các phương thức không trừu tượng.

Khi muốn kế thừa và chia sẻ code chung : Lớp trừu tượng cho phép các lớp con kế thừa và sử dụng các phương thức đã được cài đặt sẵn, giảm thiểu việc lặp lại code.

**Interface:**

Java không hỗ trợ đa kế thừa với các lớp, nhưng một lớp có thể thực hiện nhiều interface.

Interface định nghĩa các phương thức mà không cung cấp triển khai. Các lớp thực hiện interface phải cung cấp triển khai cho các phương thức này.

Interface thường được sử dụng để nhóm các phương thức có liên quan với nhau, mà không quan tâm đến các chi tiết triển khai cụ thể.

Tóm lại:

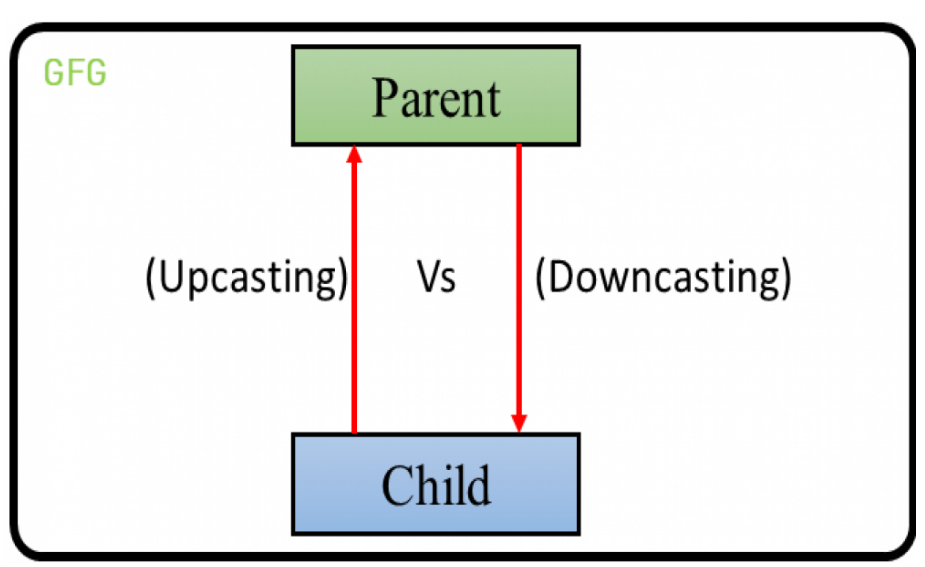
 Sử dụng abstract class khi bạn cần chia sẻ code chung giữa các lớp và muốn kết hợp cả các phương thức đã triển khai và chưa triển khai.

 Sử dụng interface khi bạn cần định nghĩa các hành vi mà không quan tâm đến chi tiết triển khai, và khi bạn muốn tận dụng đa kế thừa.

Link tham khảo : <https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html>

# **Upcasting và Downcasting**

Chỉ cần hiểu đơn giản là ép kiểu từ class cha thành class con hoặc từ class con thành class cha. Thường là ép kiểu thấp hơn (cha -> con) việc này giúp cho có thể gọi được hàm của class con.



**Lưu ý:** khi thực hiện downcasting có thể sẽ gặp lỗi **ClassCastException** nếu không thể thực hiện downcassting được. Để an toàn chúng ta nên kiểm tra một đối tượng có phải là thể hiện của một kiểu dữ liệu cụ thể không trước khi thực hiện downcasting.

# **Interface**

Link tham khảo : <https://www.w3schools.com/java/java_interface.asp>

<https://stackoverflow.com/questions/2801878/implementing-two-interfaces-in-a-class-with-same-method-which-interface-method>

* Không thể khởi tạo đối tượng bằng từ khóa “new”
* Mặc định tất cả method của interface là “abstract” + “public”
* Khi kế thừa interface cần phải ghi đè tất cả method của interface (ngoại trừ “default” method.
* Tính đa kế thừa

***+Điểm mới trong Java8***

Có thêm method deflaut và static

<https://docs.oracle.com/javase/tutorial/java/IandI/defaultmethods.html>

<https://www.baeldung.com/java-static-default-methods>

A blue rectangle with white text

Description automatically generated

Công dụng giúp cho khi các class kế thừa interface không cần phải ghi đè, có thể sử dụng luôn.

Giúp tăng tính mềm dẻo cho interface

**Lưu ý:** Nếu ta implement quá nhiều interface và ví dụ có phương thức nào đó bị trùng tên trong cả 2 interface thì ta sử dụng : TenClassInterface.super.tenHam();

# **Final**

-Tạo ra hằng số, cấm kế thừa và cấm ghi đè (@Override)

Ví dụ lớp Math và lớp String là các final class không thể kế thừa

# **Static**

Dùng để định nghĩa các thuộc tính và phương thức thuộc về 1 lớp thay vì thuộc về các đối tượng.

Nên sử dụng tên lớp để truy cập đến thuộc tính và phương thức static

Lưu ý : không thể truy cập các thuộc tính không static từ phương thức static.

A computer screen shot of a program

Description automatically generated

# **Annotation**

@Override, @Overriding ,…. Như chú thích thôi !!!!!

# **Try -catch**

Tài liệu tham khảo :

<https://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html>

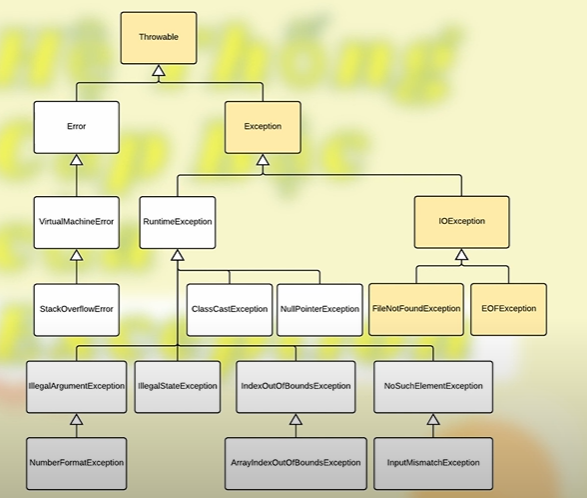
try{

//code ném Exception

}catch(Exception e){

//code muốn chạy sau khi chụp dc Exception

}



# **Exception(ngoại lệ)**

Định nghĩa thì nó là một sự kiện không mong muốn và nó sẽ làm gián đoạn chương trình đang chạy. Chương trình sẽ dừng liền ngay lập tức khi gặp Exception.

Phân loại :

<https://www.geeksforgeeks.org/exceptions-in-java/>

Chỉ xảy ra khi chạy chương trình

Link học : https://www.youtube.com/watch?v=EVoMK9Yo0fY

# **Finally**

=> Finally (settle down/onSettled) : dù có ngoại lệ hay không, thì hàm finally luôn được chạy (cho dù có sử dụng: return, continue, or break ở đâu đi nữa)

# **Try-with-resources**

https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html

Bắt đầu từ java v1.7 (version 7), thay vì dùng finally, chúng ta có thể dùng trực tiếp với try.

# **IO Stream (Đầu vào và đầu ra)**

Nguyên tắc không đổi : máy tính chỉ hiểu được ký tự 0 và 1 (không có khái niệm image/video/audio/files...)

Những khái niệm images/video/audio/files... là dành cho con người

Vậy làm sao để human và computer có thể giao tiếp với nhau ?

Ví dụ: human: cung cấp 1 file video (input) computer: chạy và hiển thị video đấy (output)

**Bước 1:** Input ban đầu (raw) cần được biến đổi thành định dạng 0 và 1 (dạng A) => máy tính hiểu (READ)

**Bước 2:** dạng A (tại bước 1) cần biến đổi thành dạng B(images/video/audio/files...)=> Output con người hiểu (WRITE)

Để có thể thực hiện hiệu quả quá trình trên, chúng ta sử dụng "Stream"

**Ưu điểm:**

- Tiết kiệm tài nguyên (memory/cpu). File càng lớn, càng tốn nhiều tài nguyên

- Giảm thiểu thời gian loading qua network (Livestream chất lượng càng cao, load càng chậm nếu mạng yếu ...)

**Nhược điểm:** Sử dụng nhiều thao tác IO (đọc ghi nhiều lần :v)

* **Nhắc lại khái niệm về stream:** <https://docs.oracle.com/javase/tutorial/essential/io/streams.html>
* **Phân loại:**

1. **Byte Streams vs Character Streams**

https://docs.oracle.com/javase/tutorial/essential/io/bytestreams.html

https://stackoverflow.com/questions/3013996/byte-stream-and-character-stream

**2. Buffered Stream** <https://docs.oracle.com/javase/tutorial/essential/io/buffers.html>

**3. Data Streams và Object Streams (Giới thiệu) //todo**

# **Path in Java**

<https://docs.oracle.com/javase/tutorial/essential/io/path.html>

Relative Path : đường dẫn tương đối

Absolute Path : đường dẫn tuyệt đối (đường dẫn đầy đủ)

-Nếu chỉ cung cấp tên file nó sẽ hỉu là ở root

# **Processes and Threads (Đa luồng)**

<https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html>

**Application :** là chương trình chạy trên máy tính

**Task :** nhiệm vụ

**Multi- tasking :** làm nhiệm vụ cùng lúc

**Processes :** tiến trình

-Mỗi tiến trình có 1 không gian riêng để chạy => không chia sẻ memory

**Threads (luồng)**

* Thread là đơn vị nhỏ nhất để thực thi code
* 1 process cần ít nhất 1 thread (main thread)
* **Thread Objects**

<https://docs.oracle.com/javase/tutorial/essential/concurrency/threads.html>

# **I++ và ++I**

Nếu để dấu cộng trước 1 biến được gọi là prefix increment (tăng tiền tố) và ngược lại là tăng hậu tố

++i và i++ đều là toán tử tăng giá trị. Nhưng sự khác biệt chính nằm ở thời điểm mà chúng tăng giá trị.

Khi sử dụng ++i, toán tử tăng giá trị (++) sẽ được thực hiện trước, và sau đó giá trị của i (sau khi đã tăng) sẽ được sử dụng trong biểu thức hoặc được gán cho biến. Tức là cộng trước sau đó gán

Khi sử dụng i++, giá trị hiện tại của i sẽ được sử dụng hoặc được gán cho biến trước, và sau đó giá trị của i mới được tăng lên 1. Tức là gán trước sau đó cộng

**Tóm lại :**

* Chúng hoạt động giống nhau khi đứng 1 mình ví dụ ++i hay i++ thì cũng như nhau

Nhưng khi chúng dùng chung với phép tính hoặc biến khác

**Ví dụ 1:**

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

int i = 0 ;

int a = i++; => Do ++ phía sau nên gán trước rồi mới cộng nên a = 0

int a = 10 + i++; => ra 10 => Bởi ++ thực hiện sau khi tất cả các câu lệnh khác trong dòng code đã được thực hiện.

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

int i = 0 ;

int a = ++i; => Do ++ phía trước nên cộng rồi mới gán

int a = 10 + ++i => ra 11 => Bởi vì toán tử ++ thực hiện trước khi bất kì câu lệnh nào khác trong cùng dòng code được thực hiện. Tức là i đã được tăng giá trị trước khi cộng cho 10

**Ví dụ 2:**

console.log('--- Hậu tố ---');

var number1 = 1;

console.log(number1++); // 1

console.log(number1); // 2

console.log(number1++); // 2

console.log(number1); // 3

console.log('--- Tiền tố ---');

var number2 = 1;

console.log(++number2); // 2

console.log(number2); // 2

console.log(++number2); // 3

console.log(number2); // 3

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

Nếu vẫn chưa hiểu hãy xem link sau : <https://www.youtube.com/watch?v=ows3CHt6-Pk&list=PLScF8do8T751dSyEHb2GrKuNz2FPBZ0Db&index=36>