Study java

Contents

[I. Kỹ thuật lập trình căn bản 1](#_Toc164235271)

[1) Các kiểu dữ liệu 1](#_Toc164235272)

[2) Các câu lệnh if, for, while, do-while, case switch… Cho ví dụ minh họa, sự khác nhau. 1](#_Toc164235273)

[II. Lập trình hướng đối tượng 2](#_Toc164235274)

[1. Class và object là gì? 2](#_Toc164235275)

[1. Lớp (Class) là gì? 2](#_Toc164235276)

[2. Đối tượng (Object) là gì? 2](#_Toc164235277)

[3. Sự khác biệt giữa class và object: 3](#_Toc164235278)

[2. Lập trình hướng đối tượng (OOP) là gì? 3](#_Toc164235279)

[a. Tính đa hình: 3](#_Toc164235280)

[b. Tính đóng gói 5](#_Toc164235281)

[c. Tính kế thừa 6](#_Toc164235282)

[d. Tính trừu tượng 6](#_Toc164235283)

[III. Xử lý ngoại lệ (Exeption) trong java 9](#_Toc164235284)

[1. Cơ chế try-catch 9](#_Toc164235285)

[2. Throw ngoại lệ 10](#_Toc164235286)

[3. Chuyển tiếp ngoại lệ 10](#_Toc164235287)

# Kỹ thuật lập trình căn bản

## Các kiểu dữ liệu

* Nguyên thủy: Byte, short, int, long, float, double, char, boolean
* Không nguyên thủy: String, Array, classes, interface, enum, wrapper

## Các câu lệnh if, for, while, do-while, case switch… Cho ví dụ minh họa, sự khác nhau.

* if - else:

int x = 10;

if(x < 100){

System.out.println(x)

}else{

System.out.println(x + “lớn hơn 100”);

}

* for:

for(int i = 0; i < 5; i++){

System.out.println(“Giá trị i: ” + i);

}

## Nhập trong java

### Scanner

* Đọc chuỗi trong màn console bằng phương thức nextLine()
* Đọc số từ màn console bằng:
  + nextInt()
  + nextDouble()
  + nextFloat()

### Buffered Reader

* Đọc chuỗi từ màn hình bằng realLine()
* Đọc số bằng:
  + Integer.parseInt()
  + Double.parseDouble()

## Các kiểu dữ liệu

| **Kiểu** | **Mô tả** | **Kích cỡ** | **Tối thiểu** | **Tối đa** | **Lớp bao** |
| --- | --- | --- | --- | --- | --- |
| byte | số nguyên một byte | 8 bit | -128 | 127 | Byte |
| short | số nguyên ngắn | 16 bit | -2^15 | 2^15-1 | Short |
| int | số nguyên | 32 bit | -2^31 | 2^31-1 | Integer |
| long | số nguyên dài | 64 bit | -2^63 | -2^63-1 | Long |
| float | kiểu thực với độ chính xác đơn | 32 bit | IEEE754 | IEEE754 | Float |
| double | Double-precision floating point | 64 bit | IEEE754 | IEEE754 | Double |
| char | kiểu kí tự | 16 bit | Unicode 0 | Unicode 2^16-1 | Character |
| boolean | kiểu logic | true hoặc false | - | - | Boolean |
| void | - | - | - | - | Void |

## Hàm và phương thức

* Hàm có kết quả trả về sẽ không có void
* Hàm không có kết quả trả về sẽ có void

## Ép kiểu trong java

* (kiểu dữ liệu) biểu thức
* Ví dụ:
  + (float) 3 \* 5;

## Chuyển kiểu dữ liệu

* Chuyển String sang int
  + int i = Integer.parseInt("200");
* Chuyển đổi int thành String trong java
  + String.valueOf() hoặc Integer.toString()

## Toán tử 3 ngôi

* Ví dụ: String result = a % 2 == 0 ? "a là số chẵn" : "a là số lẻ";

# Collections trong Java

## Collection là gì?

* Collection là gì: Trong lập trình Java, Collections (Tập hợp) là một khái niệm quan trọng và mạnh mẽ cho việc lưu trữ và quản lý một nhóm các đối tượng. Chúng ta có thể thực hiện nhiều thao tác trên các tập hợp như tìm kiếm, sắp xếp, chèn, xóa và nhiều thao tác khác
* Là một khung (framework) cung cấp kiến trúc để lưu trữ và thao tác trên các tập hợp đối tượng

## List trong java

* Trong Java, chúng ta phải import package là java.util.List để sử dụng được List.
  + ArrayList
    - Thêm phần tử vào ArrayList: add(“adb”)
    - Thêm phần tử ArrayList này qua ArrayList khác: addAll(mảng)
    - Truy cập tới phần tử trong ArrayList: mảng.get(0)
    - Thay đổi phần tử trong ArrayList: mảng.set(1, “bcv”);
    - Xóa phần tử mảng: mảng.remove(1);
    - Xóa tất cả phần tử: mảng.clear();
    - Kiểm tra độ dài của mảng: mảng.size();

## Set trong java

* Thêm phần từ set.add();
* Xóa phần tử set.remove(“”);
* Xóa tất cả trong set: set.clear();

## Map trong java

* Trong java, map được sử dụng để lưu trữ và truy xuất dữ liệu theo cặp khóa (key) và giá trị (value). Mỗi cặp key và value được gọi là entry.
* Chèn liên kết put(K, V), nếu đã có K thì V sẽ thay thế giá trị cũ: put(k,v);
* Chèn tất cả map đã chỉ định vào map hiện tại: putAll()
* Trả về value với k tương ứng: get(k)
* Thay thế value của key K với value V được chỉ định mới nhất: replace(k, v)
* Thay thế value của key K với value mới là newValue nếu key K được liên kết với value oldValue: replace(K, oldValue, newValue)
* Xóa mục được đại diện bởi key K khỏi map: remove(K)
* Xóa mục có key K liên kết với value V khỏi map: remove(K, V)
* Trả về một tập hợp gồm tất cả các key có trong map: keyset()
* Trả về tập hợp tất gồm cả các value có trong map: value()
* Trả về tập hợp gồm tất cả cặp key/value có trong map: entryset()

## Enum trong java

* Enum là một từ khóa trong Java, là một kiểu dữ liệu đặc biệt được sử dụng để đại diện cho hằng số cố định. Một enum có thể chứa các trường, phương thức và constructor

# Lập trình hướng đối tượng

## Class và object là gì?

### Lớp (Class) là gì?

* Trong Java, một lớp là một thực thể xác định hành vi mà một đối tượng có và sẽ có. Nói cách khác, một lớp chỉ là một bản thiết kế hoặc một tập hợp các hướng dẫn để xây dựng các đặc tính của một đối tượng cụ thể sau này.
* **Cách tạo một lớp trong java:**

class <class\_name>{

field;

method;

}

Trong đó:

* Class: từ khóa để tạo một class
* <class\_name>: tên class bạn định tạo nó sẽ được viết liền và viết HOA chữ cái đầu tiên của từng từ
* Field: biến đối tượng
* Method: phương thức

### Đối tượng (Object) là gì?

* Một Object có thể chứa các thành phần như các phương thức (method) và thuộc tính (thuộc tính) để tạo ra các kiểu dữ liệu hữu ích. Đối tượng xác định hành vi của lớp. Khi bạn gửi tin nhắn đến một đối tượng, bạn bắt buộc phải gọi đối tượng hoặc thực hiện một trong các phương thức của nó.
* **Cách tạo một object:**

<class\_name> ReferenceVariable = new <class\_name>();

Trong đó:

* <class\_name>: kiểu dữ liệu của đối tượng
* ReferenceVariable: tên tham chiếu của đối tượng
* new: từ khóa dùng tạo đối tượng
* <class\_name>(): class dùng để tạo bạn đối tượng

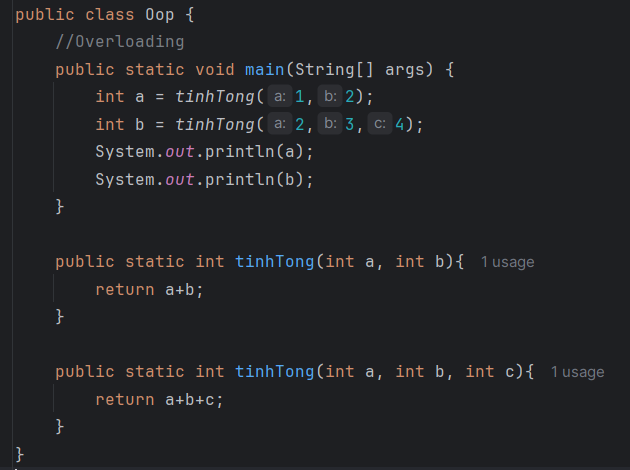
### Sự khác biệt giữa class và object:

|  |  |  |
| --- | --- | --- |
| STT | Lớp (Class) | Đối tượng (Object) |
| 1 | Là bản thiết kế hoặc khuôn mẫu mà từ đó đối tượng được tạo ra | Là một thể hiện của lớp |
| 2 | Là tập hợp những đối tượng có thuộc tính giống nhau | Là một thực thể trong thế giới thực như bút, máy tính, giường,… |
| 3 | Chỉ cần khai báo 1 lần | Khai báo bao nhiêu lần cũng được |
| 4 | Được khai báo với từ khóa Class | Đối tượng được khai báo với từ khóa “new” hoặc tên lớp |
| 5 | Lớp không được cấp phát bộ nhớ khi được tạo | Đối tượng được phân bộ nhớ mỗi khi khởi tạo |
| 6 | Thực thể logic | Thực thể vật lý |
| 7 | Như lớp động vật (Animal) | Đối tượng là một thực thể như con mèo (cat), con chó (dog) |

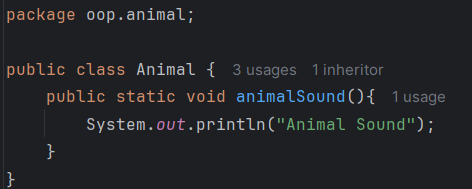
## Lập trình hướng đối tượng (OOP) là gì?

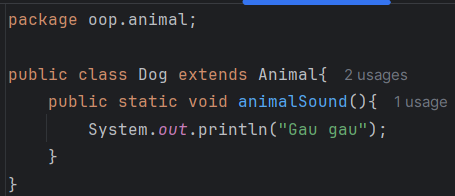
### Tính đa hình:

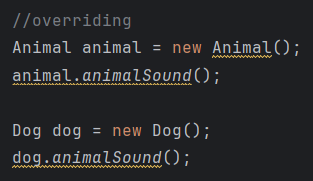
* OOP cho phép bạn sử dụng tính đa hình, nghĩa là bạn có thể sử dụng các đối tượng cùng một lớp một cách đa dạng.
  + Ví dụ, bạn có thể sử dụng một phương thức "di chuyển" cho nhiều loại đối tượng khác nhau, như “Người” và “Xe hơi”.
* **Overloading (Nạp chồng phương thức) là gì?**
  + Cho phép một class có nhiều phương thức cùng thêm nhưng khác nhau về tham số truyền vào, hoặc kiểu dữ liệu tham số:
  + Ví dụ:

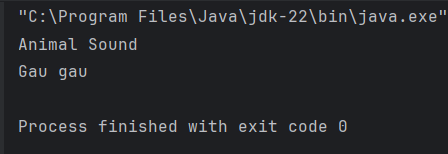


* **Overriding (Ghi đè phương thức) là gì?**
  + Overrding (tên đầy đủ là Method Overriding), được sử dụng trong trường hợp lớp con kế thừa từ lớp cha và muốn định nghĩa lại một phương thức đã có mặt ở lớp cha
  + Ví dụ:

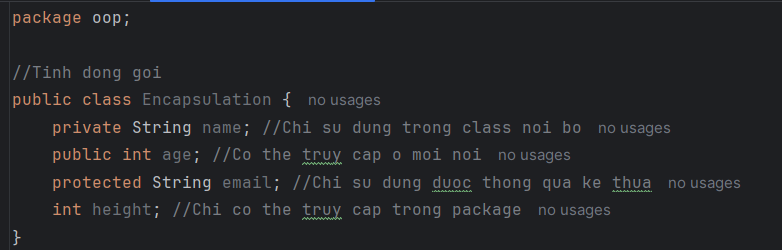






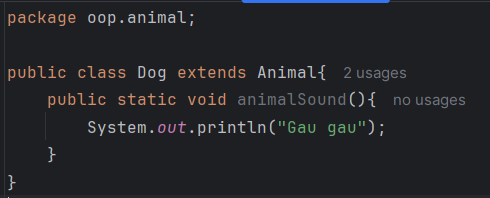


### Tính đóng gói



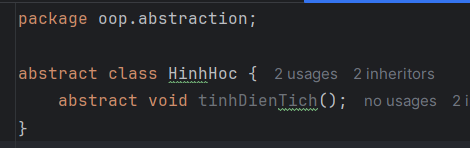
* private: chỉ sử dụng trong class nội bộ
* public: truy cập ở mọi nơi
* protected: sử dụng khi được kế thừa
* default: chỉ sử dụng trong packge

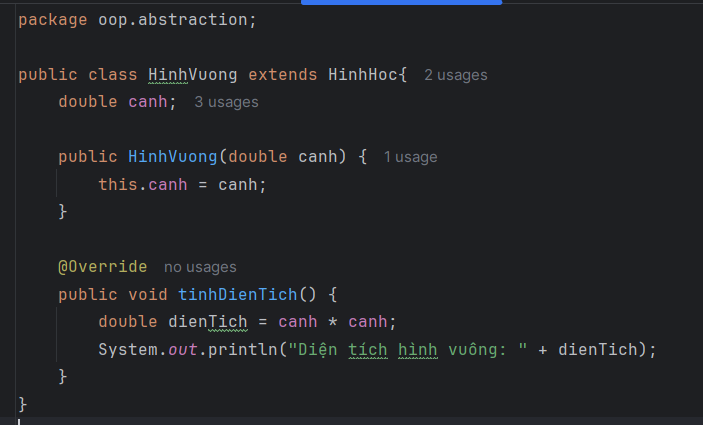
### Tính kế thừa

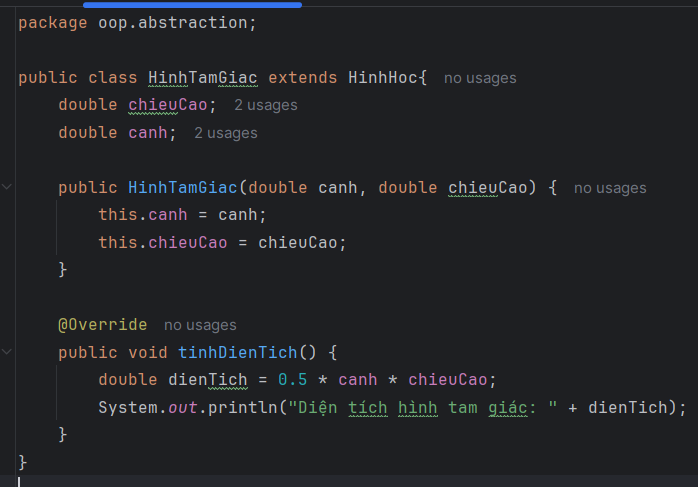
* Tính kế thừa giúp tái sử dụng mã nguồn và tạo ra cấu trúc phân cấp cho các lớp. Điều này cho phép bạn xây dựng các lớp mới trên cơ sở của các lớp đã có, tiết kiệm thời gian và công sức, đồng thời duy trì tính nhất quán trong mã nguồn. 

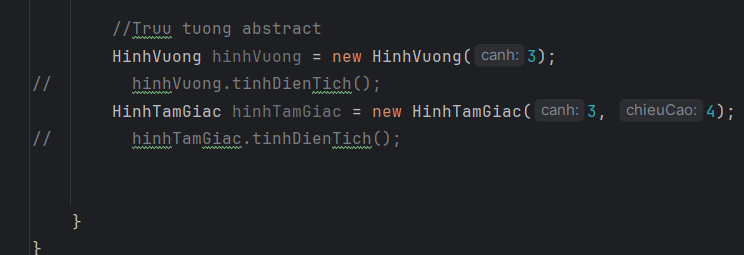
### Tính trừu tượng

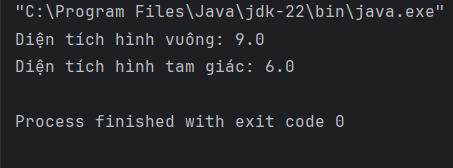
* Abstract nó cho phép bạn tạo ra các lớp và đối tượng trừu tượng, tập trung vào các tính năng quan trọng mà bạn quan tâm và che giấu chi tiết phức tạp bên trong
* Ví dụ:



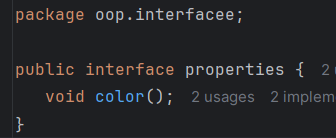


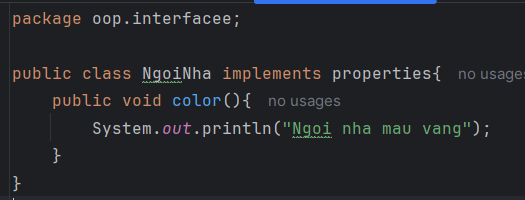


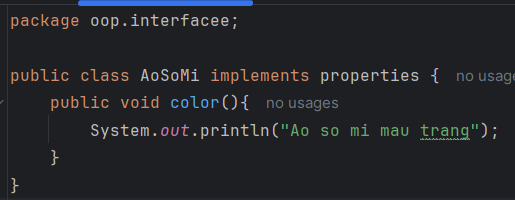


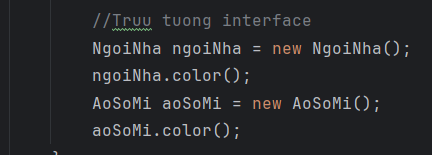


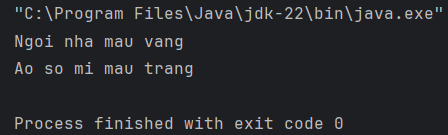
* Interface là một bản thiết kế của một lớp. Nó chỉ có các phương thức trừu tượng. Interface là một kỹ thuật để thu được tính trừu tượng hoàn toàn và đa kế thừa trong Java
* Ví dụ:











# Xử lý ngoại lệ (Exeption) trong java

* Exception là một tình trạng bất thường xảy ra trong quá trình thực thi chương trình, phá vỡ luồng chuẩn của chương trình.
* Checked Exception: Đây là các ngoại lệ mà chúng ta phải kiểm tra tại thời điểm biên dịch. Checked Exception thường xảy ra do người dùng và không thể dự đoán được bởi lập trình viên. Ví dụ, IOException, SQLException.
* Unchecked Exception: Đây là các ngoại lệ không cần kiểm tra tại thời điểm biên dịch và thường xảy ra do lỗi trong mã nguồn, chẳng hạn như NullPointerException, ArithmeticException.
* Error: Là các ngoại lệ không thường được đặt bẫy, và chúng thường xảy ra do lỗi vượt quá tầm kiểm soát của lập trình viên, ví dụ như OutOfMemoryError, StackOverflowError.

## Cơ chế try-catch

* Cú pháp của try-catch trong java

Try{

<mã lệnh có thể xảy ra ngoại lệ>

} catch (Exception\_Type e) {

<Xử lý ngoại lệ ở đây>

}

* try: Khối mã lệnh mà bạn muốn kiểm tra ngoại lệ.
* catch: Khối mã lệnh sẽ được thực thi nếu ngoại lệ xảy ra. Exception\_Type là kiểu ngoại lệ mà bạn muốn bắt.

## Throw ngoại lệ

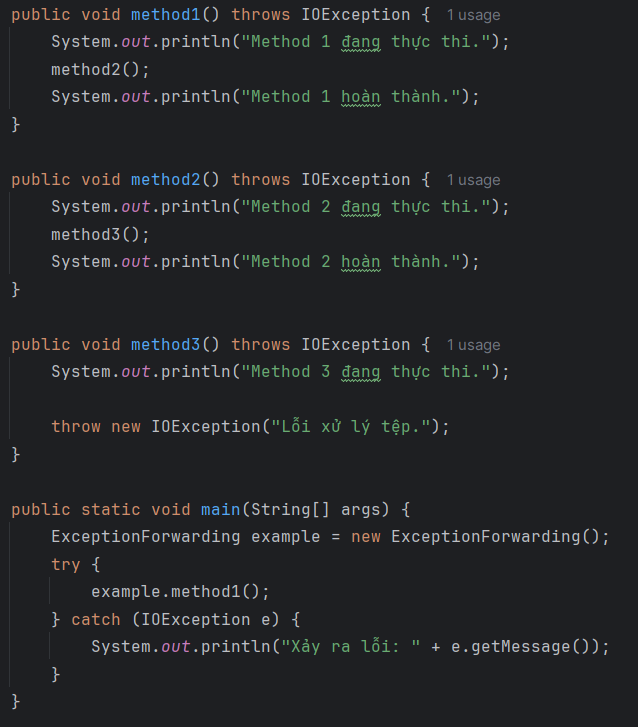
* Từ khóa throw trong Java được sử dụng để ném ra một exception tường minh. Chúng ta có thể ném cả Checked Exception và Unchecked Exception trong Java bằng từ khóa throw. Từ khóa throw thường được sử dụng để ném các Custom Exception
* Cú pháp throw trong Java như sau:

throw exception;

* Ví dụ về việc sử dụng throw để ném ra một IOException:

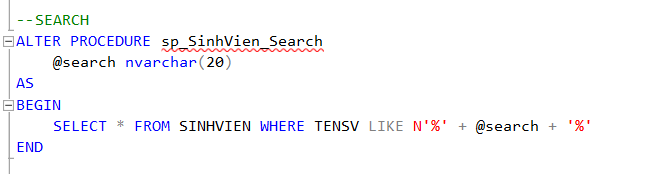
throw new IOException("File không tồn tại");

## Chuyển tiếp ngoại lệ

* Trong Java, chuyển tiếp ngoại lệ (exception forwarding) là quá trình mà một phương thức không xử lý hoặc không có khả năng xử lý một ngoại lệ, nhưng nó chuyển tiếp (forward) ngoại lệ đó đến một phương thức gọi khác để xử lý.
* 

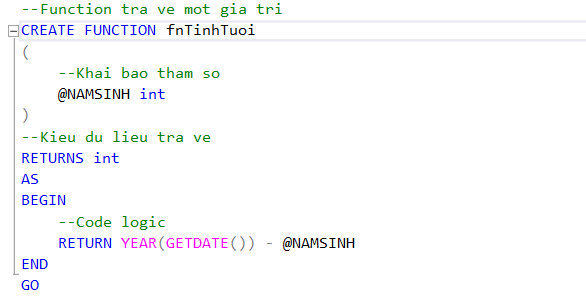
# Lập trình xử lý với database

## Stored Procedure

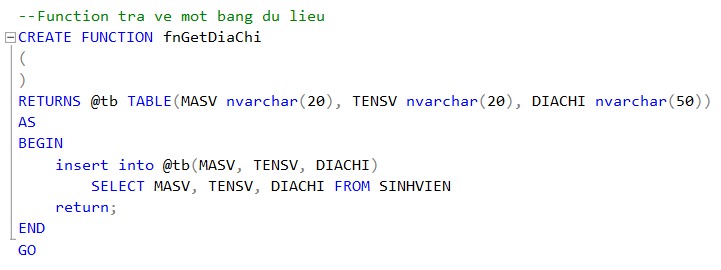
* Một Stored Procedure là bao gồm các câu lệnh Transact-SQL và được lưu lại trong cơ sở dữ liệu. Các lập trình viên chỉ cần gọi ra và thực thi thông qua SQL Server Management Studio hoặc ngay trong ứng dụng đang phát triển.
* 

## Function

* Là đoạn chương trình kịch bản (programming scripts) với các câu lệnh SQL nhúng (embedded SQL) được lưu dưới dạng đã được biên dịch và thi hành thực tiếp bởi MySQL server.
* Function trả về một dữ liệu:



* Function trả về bảng dữ liệu:



## Trigger

* Hiểu đơn giản thì Trigger là một stored procedure không có tham số. Trigger thực thi một cách tự động khi một trong ba câu lệnh Insert, Update, Delete làm thay đổi dữ liệu trên bảng có chứa trigger
* Có 2 loại trigger:
  + Thực hiện sau khi có 1 câu lệnh I/U/D trên bảng dữ liệu thực hiện xong
  + Trigger sẽ thực hiện thay thế cho các câu lệnh I/U/D
* Gắn với sự kiện I/U/D

## Sequence

* SEQUENCE là một tập hợp các số nguyên được tạo thứ tự theo yêu cầu. Các SEQUENCE thường được sử dụng trong cơ sở dữ liệu vì đáp ứng yêu cầu của nhiều ứng dụng là đòi hỏi mỗi hàng trong một bảng chứa một giá trị duy nhất tương tự như khóa chính

## Partition

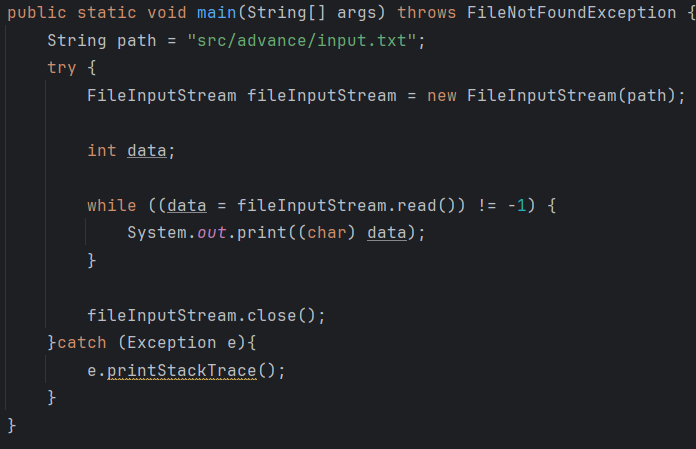
* Partition là quá trình phân chia table thành các phân vùng nhỏ theo một quy tắc nào đó được gọi là partition function.

# Java Core Advance

## Java IO

### FileInputStream

* Dùng để đọc dữ liệu từ file
* Constructors:
  + FileInputStream(File file)
  + FileInputStream(FileDescriptor fdObj)
  + FileInputStream(String name)



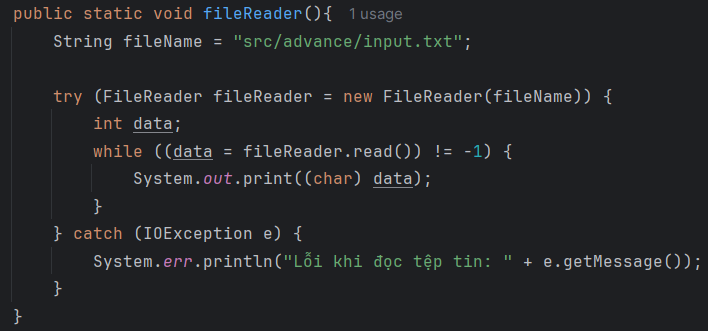
### BufferendlnputStream

* Đọc dữ liệu bằng cách sử dụng bộ nhớ tạm, buffer, trong bộ nhớ
  + Sử dụng buffer sẽ nhanh hơn so với thông thường
* Constructors:
  + BufferedInputStream(InputStream int)
  + BufferedInputStream(InputStream int, int size)
* Một số phương thức
  + int available()
  + void mark(int readlimit)
  + void reset ()
  + boolean markSupported ()
  + void close ()
  + int read(byte [] b, int off, int len)
  + long skip(long n)



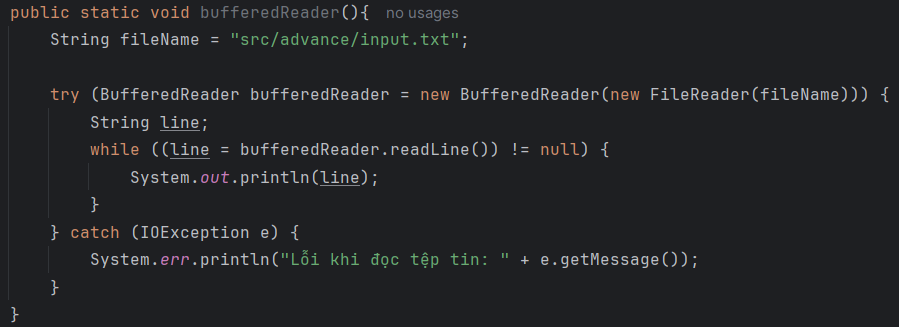
### FileReader

* Dùng để đọc 1 file
* Constructor:
  + FileReader (File file)
  + FileReader (FileDescriptor fd)
  + FileReader (String fileName)



### BufferedRender

* Dùng để đọc file
* Constructor:
  + BufferedReader(Reader in)
  + BufferedReader(Reader in, int sz)



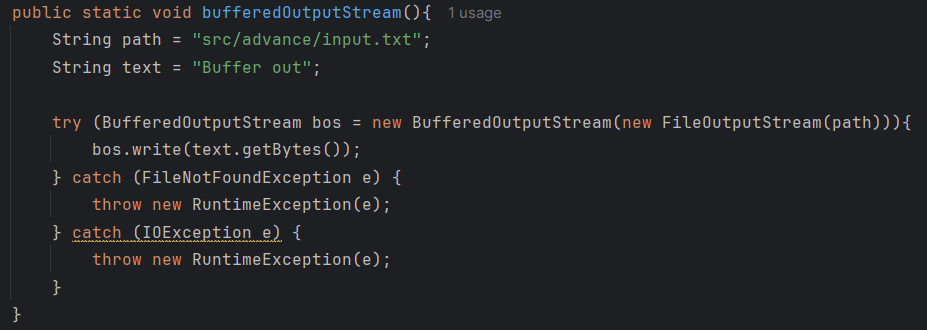
### FileOutputStream

* Dùng để ghi dữ liệu
* Constructor:
  + FileOutputStream(File file, boolean append)
  + FileOutputStream(FileDescriptor fdObj)
  + FileOutputStream(String name)
  + FileOutputStream(String name, boolean append)



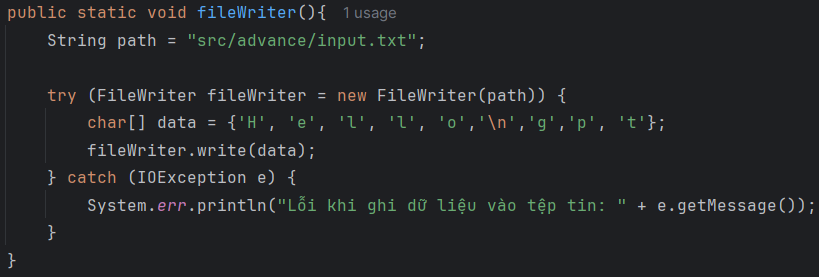
### BufferedOutputStream

* Dùng để ghi dữ liệu
* Constructor:
  + BufferedOutputStream(OutputStream out)
  + BufferedOutputStream(OutputStream out, int size)



### FileWriter

* Cho phép ghi các ký tự lên file.
* Một số constructors của lớp FileWriter:
  + FileWriter (Tệp tệp)
  + FileWriter (Tệp tệp, nối thêm boolean)
  + Trình ghi tệp (Bộ mô tả tệp fd)
  + FileWriter (Tên tệp chuỗi, nối thêm boolean)

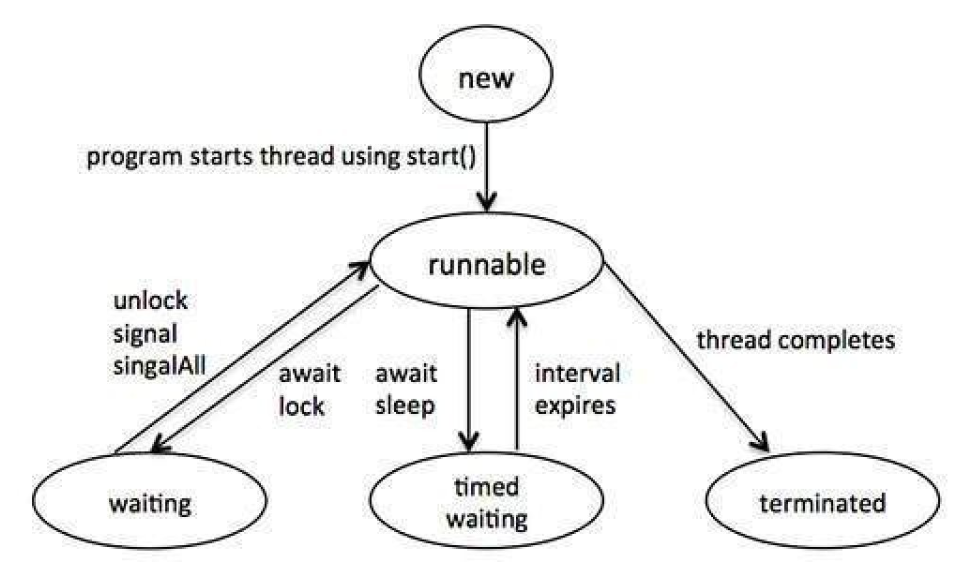
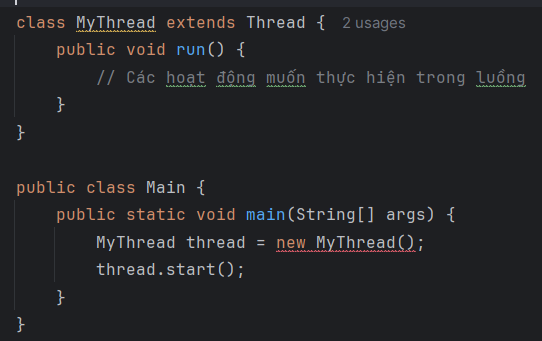
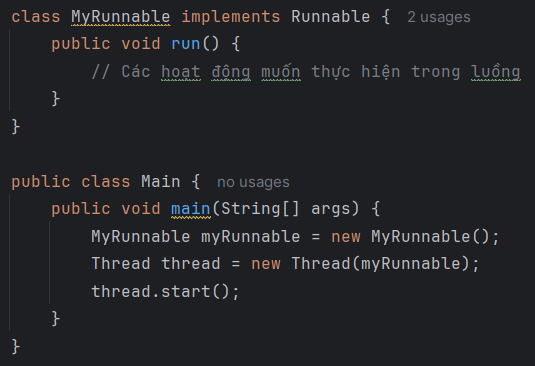


### File

* Lớp File cho phép làm việc trực tiếp với các file và thư mục. Lớp này cung cấp các phương thức để tạo, xóa và đổi tên file, cũng như thao tác với các thư mục.
* Lớp File bổ sung thêm các thao tác đối với tập tin/thư mục mà các lớp thể hiện stream còn chưa cung cấp.
* Constructor:
  + File(String dirpath)
  + File(String dirpath, String filename)
  + File(File fileobj, String filerſame)
  + File(URL urlobj)
* exists(): kiểm tra tồn tại
* getName(): lấy tên thư mục
* getAbsolutePath(): lấy đường dẫn tuyệt đối
* mkdir(): tạo thư mục mới
* createNewFile(): tạo thư mục mới, nếu đã tồn tại thì tạo mới sẽ thất bại
* delete(): xóa thư mục

|  |  |  |
| --- | --- | --- |
| Lớp | Ưu điểm | Nhược điểm |
| FileInputStream | - Đơn giản và dễ sử dụng.  - Có thể sử dụng để đọc dữ liệu từ file byte theo byte. | - Không thực hiện bất kỳ loại đệm nào, có thể dẫn đến hiệu suất không tốt khi đọc dữ liệu từ file lớn. |
| BufferedInputStream | - Sử dụng bộ đệm giúp tăng hiệu suất đọc dữ liệu từ file bằng cách đọc dữ liệu vào bộ đệm trước khi xử lý.  - Cung cấp các phương thức hỗ trợ đọc dữ liệu từ bộ đệm một cách hiệu quả hơn. | - Tăng sử dụng bộ nhớ.  - Có thể cần phải gọi phương thức close() để giải phóng tài nguyên đúng cách. |
| FileReader | - Đơn giản và dễ sử dụng cho việc đọc các dữ liệu ký tự từ file. | - Không hỗ trợ các phương thức đọc byte.  - Không đảm bảo hiệu suất tốt khi đọc dữ liệu từ file lớn. |
| BufferedReader | - Sử dụng bộ đệm giúp tăng hiệu suất đọc dữ liệu từ file ký tự bằng cách đọc dữ liệu vào bộ đệm trước khi xử lý.  - Cung cấp các phương thức hỗ trợ đọc dữ liệu từ bộ đệm một cách hiệu quả hơn. | - Tăng sử dụng bộ nhớ.  - Có thể cần phải gọi phương thức close() để giải phóng tài nguyên đúng cách. |
| FileOutputStream | - Đơn giản và dễ sử dụng.  - Có thể sử dụng để ghi dữ liệu vào file byte theo byte. | - Không thực hiện bất kỳ loại đệm nào, có thể dẫn đến hiệu suất không tốt khi ghi dữ liệu vào file lớn. |
| BufferedOutputStream | - Sử dụng bộ đệm giúp tăng hiệu suất ghi dữ liệu vào file bằng cách ghi dữ liệu vào bộ đệm trước khi ghi vào file.  - Cung cấp các phương thức hỗ trợ ghi dữ liệu từ bộ đệm một cách hiệu quả hơn. | - Tăng sử dụng bộ nhớ.  - Có thể cần phải gọi phương thức close() để giải phóng tài nguyên đúng cách. |
| FileWriter | - Đơn giản và dễ sử dụng cho việc ghi các dữ liệu ký tự vào file. | - Không hỗ trợ các phương thức ghi byte.  - Không đảm bảo hiệu suất tốt khi ghi dữ liệu vào file lớn. |
| File | - Cung cấp các phương thức để tạo, xóa và đổi tên file, cũng như thao tác với các thư mục.  - Bổ sung thêm các thao tác đối với tập tin/thư mục mà các lớp thể hiện stream còn chưa cung cấp. | - Không thể đọc hoặc ghi trực tiếp dữ liệu từ hoặc vào file.  - Không có các phương thức trực tiếp để đọc hoặc ghi dữ liệu từ file. |

# Thread (luồng)

* Khái niệm:
  + Thread(luồng) là đơn vị nhỏ nhất trong java, mỗi luồng sẽ thực hiện một công việc riêng biệt được quản lý bảo máy ảo java
  + Java là ngôn ngữ lập trình đa luồng
  + Mỗi thread có thể chạy song song riêng biệt với nhau
* Vòng đời của một thread:
  + 
  + Thread đi qua các giai đoạn khác nhau, được sinh ra, bắt đầu, chạy và sẽ tự động hủy ngay sau đó
    - New: Đây là trạng thái đầu tiên của một Thread khi mới bắt đầu vòng đời của nó, tồn tại khí chương trình bắt đầu chạy
    - Runnable: Khi bắt đầu chạy nó sẽ thành runnable, trạng thái này của nó là đang thực hiện tác vụ của mình
    - Waiting: thread sẽ ở trạng thái waiting(chờ đợi) để một thread khác thực hiện một tác vụ nào đó, nó sẽ tự động chuyển thành runnable chỉ khi thread trc đó ra hiệu
    - Timed waiting: một thread đang ở trạng thái runnable có thể chuyển thành trạng thái timed waiting trong 1 khoảng tg nào đó. tg này phụ thuộc vào tốc độ xử lý của thread trc đó
    - Terminated: Một thread trong trạng thái runnable có thể đi vào trạng thái terminated khi nó đã hoàn thành xong tất cả các tác vụ của nó.
* Có 2 cách chính để khởi tạo 1 thread:
  + Kế thừa lớp Thread:
    - 
  + Implement interface Runnable:
    - 

|  |  |  |
| --- | --- | --- |
| Cách khởi tạo | Ưu điểm | Nhược đi |
| Kế Thread | - Dễ dàng triển khai với cú pháp đơn giản.  - Có thể truy cập trực tiếp các phương thức và thuộc tính của lớp Thread. | - Không thể kế thừa từ bất kỳ lớp nào khác, vì Java không hỗ trợ đa kế thừa.  - Gây ra mối quan tâm về việc kế thừa và phân biệt giữa việc mở rộng lớp và thực hiện giao diện. |
| Implement Runnable | - Phân chia tốt hơn giữa nhiệm vụ và luồng, giúp tăng tính linh hoạt.  - Cho phép một lớp thực hiện nhiều giao diện khác nhau. | - Phức tạp hơn một chút so với việc kế thừa lớp Thread.  - Đôi khi cần phải tạo một đối tượng Thread bổ sung để quản lý luồng, làm tăng độ phức tạp của mã. |

* Thread Priority: mức độ ưu tiên của thread từ 1 - 10
  + 3 hằng số đại diện cho mức độ ưu tiên
  + Thread.MIN\_PRIORITY: Mức độ ưu tiên tối thiểu, có giá trị là 1. Thread.NORM\_PRIORITY: Mức độ ưu tiên mặc định, có giá trị là 5. Thread.MAX\_PRIORITY: Mức độ ưu tiên tối đa, có giá trị là 10.
  + Đặt mức độ ưu tiên cho luồng
    - setPriority(int priority): Đặt mức độ ưu tiên cho luồng.
    - getPriority(): Lấy mức độ ưu tiên hiện tại của luồng.
* Deadlock là một tình huống xảy ra khi hai hoặc nhiều luồng hoặc quy trình đang chờ đợi tài nguyên được giữ bởi nhau, dẫn đến bị treo
* Timer là một công cụ cho phép bạn lên lịch thực hiện các nhiệm vụ (tasks) vào một thời điểm cụ thể trong tương lai hoặc lặp lại các nhiệm vụ theo một lịch trình cố định.
  + Các nhiệm vụ có thể là các đoạn mã được thực thi bởi một luồng (thread) hoặc một đối tượng TimerTask.

# Lập trình mạng

## Socket là gì?

* Socket được sử dụng để thiết lập kết nối giữa các máy tính thông qua mạng. Nó cung cấp một cách tiêu chuẩn để truyền và nhận dữ liệu giữa các máy tính hoặc ứng dụng trên mạng.

## Tcp là gì?

### Khái niệm

* Là giao thức chính trong giao thức TCP/IP
* Sử dụng để truyền dữ liệu qua mạng internet
* Là giao thức cấp độ kết nối, tức là nó thiết lập một kết nối ổn định giữa 2 thiết bị trước khi truyền dữ liệu

### Cơ chế

* Trước khi truyền dữ liệu, máy gửi và máy nhận cần thiết lập kết nối TCP
* Quá trình này bắt đầu bằng việc máy gửi gửi một gói tin SYN đến máy nhận. Máy nhận sau đó phản hồi bằng một gói tin SYN-ACK. Cuối cùng, máy gửi xác nhận bằng cách gửi một gói tin ACK đến máy nhận. Sau khi ba bước này hoàn tất, kết nối TCP được thiết lập và dữ liệu có thể được truyền.

## Udp là gì

### Khái niệm: UDP là một giao thức không đảm bảo, không kết nối, có nghĩa là nó không thiết lập kết nối trước khi truyền dữ liệu và không đảm bảo dữ liệu được truyền đến đích một cách chính xác và đầy đủ.

* Nó cung cấp một cách truyền dữ liệu nhanh chóng và hiệu quả hơn so với TCP, nhưng không đảm bảo về tính toàn vẹn hoặc thứ tự của dữ liệu.

### Cơ chế

* Không đảm bảo độ tin cậy (Unreliable): UDP không đảm bảo rằng dữ liệu được gửi sẽ đến đích một cách chính xác và đầy đủ.
* Không kết nối (Connectionless): UDP không thiết lập kết nối trước khi truyền dữ liệu.
* Giao diện gửi/nhận đơn giản (Simple Send/Receive Interface): UDP cung cấp một giao diện gửi/nhận đơn giản cho việc truyền dữ liệu.

|  |  |  |
| --- | --- | --- |
| Tiêu chí | TCP | UDP |
| Độ tin cậy | Tin cậy cao (sử dụng xác nhận và khôi phục lỗi) | Không tin cậy (không sử dụng xác nhận và khôi phục lỗi) |
| Xác định thứ tự | Đảm bảo dữ liệu được gửi và nhận theo thứ tự | Không đảm bảo thứ tự của dữ liệu |
| Phản hồi | Có thể cung cấp phản hồi khi truyền dữ liệu | Không cung cấp phản hồi khi truyền dữ liệu |
| Overhead | Overhead cao (bao gồm header lớn, xác nhận, retransmission) | Overhead thấp (bao gồm header nhỏ) |
| Gửi dữ liệu | Gửi dữ liệu theo dòng (stream-oriented) | Gửi dữ liệu dưới dạng gói (datagram-oriented) |
| Ứng dụng | Thích hợp cho các ứng dụng yêu cầu tin cậy như truyền file, email, web browsing | Thích hợp cho các ứng dụng yêu cầu tốc độ cao như streaming media, video games |
| Tiêu thụ băng thông | Tiêu thụ băng thông lớn hơn vì có overhead lớn | Tiêu thụ băng thông thấp hơn vì overhead nhỏ |
| Cổng kết nối | Kết nối hai chiều song song (full duplex) | Kết nối một chiều (half duplex) |
| Kết nối | Thiết lập kết nối ba bước (three-way handshake) trước khi truyền dữ liệu | Không yêu cầu thiết lập kết nối trước khi truyền dữ liệu |

# Lập trình mạng socket

* Khái niệm: là cánh của giao tiếp giữa hai tiến trình ứng dụng giữa máy chủ và máy khách đang chạy trên mạng
* Bên Server:
  + Khởi tạo server socket:
    - ServerSocket (int port, int backlog):
      * Backlog: xác định xem có bao nhiêu client đến để lưu trữ trong một hàng đợi
      * Khởi tạo một ServerSocket dựa trên port chỉ định
  + Socket accept(): chấp nhận kết nối từ client và trả về một đối tượng Socket
* Bên Client:
  + Socket(String hot, int port): kết nối với server
  + Public InputStream getInputStream(): trả về input stream của socket. Input stream được kết nối với output của socket từ xa
  + Public OutputStream getOutputStream(): trả về đầu ra của socket. Output stream được kết nối với input stream của socket từ xa
  + Public void close(): đóng socket,