Unit6: THREAD

**+, Phương thức join()** chờ một thread chết. Nói cách khác, nó làm cho các thread đang chạy ngừng hoạt động cho đến khi luồng mà nó tham gia hoàn thành nhiệm vụ của nó.

Text

Description automatically generated

+, **ThreadPool**: được dùng để giới hạn số lượng Thread được chạy bên trong ứng dụng của chúng ta trong cùng một thời điểm. Nếu chúng ta không có sự giới hạn này, mỗi khi có một Thread mới được tạo ra và được cấp phát bộ nhớ bằng từ khóa new thì sẽ có vấn đề về bộ nhớ và hiệu suất, có thể dẫn đến lỗi crash chương trình.

Text

Description automatically generated with medium confidence

Unit7: ENCODING AND DECODING TRONG JAVA

Encode và Decode là những khái niệm dùng để chỉ đến việc chuyển đổi dữ liệu cho hai mục đích khác nhau lưu trữ và sử dụng. Encode thông thường sẽ tạo ra dữ liệu phục phụ cho việc lưu trữ, ngược lại Decode sẽ chuyển đổi dữ liệu lưu trữ thành dữ liệu sử dụng trong chương trình.

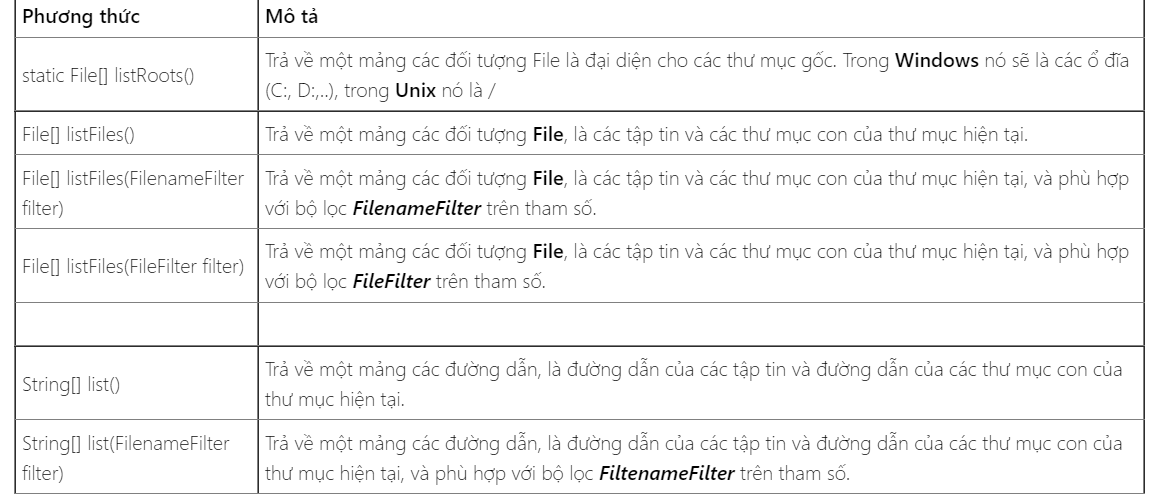
+, Byte : 1 Byte là 8 bit và có thể giữ các giá trị từ -128 đến 127.

+, File: Package **java.io** có lớp **File** cho phép bạn làm việc với các file (tệp). Thường để bắt đầu, cần tạo ra một đối tượng File bằng khởi tạo với tham số đường dẫn

A picture containing table

Description automatically generated

+, ListFile():



AtomicLong: Một nguyên tử được sử dụng trong các ứng dụng như số trình tự tăng nguyên tử và không thể được sử dụng để thay thế cho một thời gian dài. Tuy nhiên, lớp này mở rộng số lượng để cho phép truy cập thống nhất bằng các công cụ và tiện ích liên quan đến các lớp dựa trên số.

+ stream.: Streams là một bổ sung mới trong thư viện Collections API của Java 8. Streams là cách thức mới để xử lý tập hợp dữ liệu - Collections data bên cạnh các cách thức thông thường là vòng lặp - for, bộ lặp - iterator.

+, getAndSet: Java.util.concien.atomic.atomicinteger.getandset () là một phương thức sẵn có trong Java đặt giá trị đã cho cho giá trị được truyền trong tham số và trả về giá trị trước khi cập nhật loại dữ liệu của int.

+, InputStream đọc dữ liệu theo một chuỗi byte có thứ tự.

+, **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.

+, **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.

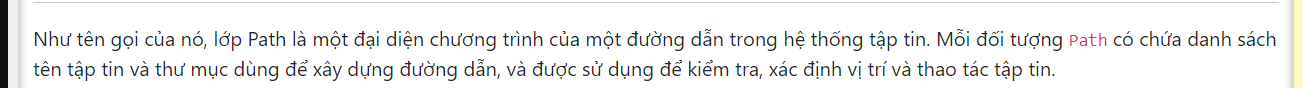
+, **Lớp ObjectOutputStream trong java** được sử dụng để ghi các kiểu dữ liệu nguyên thuỷ và các đối tượng Java vào một OutputStream. Chỉ có các đối tượng implements giao tiếp java.io.Serializable mới có thể được ghi vào stream.

 +, **FileChannel** có thể được sử dụng để sao chép tệp trong Java một cách nhanh chóng thông qua transferFrom() method.

+, **rewind()**: Phương thức tua lại () của lớp java.nio.bytebuffer được sử dụng để tua lại bộ đệm này. Vị trí được đặt thành 0 và nhãn hiệu bị loại bỏ. Gọi phương thức này trước một chuỗi các hoạt động-viết kênh hoặc nhận được, giả sử rằng giới hạn đã được đặt một cách thích hợp. Gọi phương thức này không thay đổi cũng như loại bỏ giá trị của Mark.

+, Release(): phóng thích. Phát hành giấy phép, trả lại cho Semaphore. Phát hành giấy phép, tăng số lượng giấy phép có sẵn. Nếu bất kỳ chủ đề nào đang cố gắng để có được giấy phép, thì một chủ đề được chọn và được cấp giấy phép vừa được phát hành.

+, **NIO**:



**+, DirectoryStream:**

**Thư mục là đối tượng để lặp lại các mục trong một thư mục. Một luồng thư mục cho phép sử dụng thuận tiện cho cấu trúc for-each để lặp lại trên một thư mục.**

**+, ByteBuffer: Một bytebuffer là một bộ đệm cung cấp cho các byte chuyển từ nguồn sang đích. Ngoài việc lưu trữ như một mảng bộ đệm, nó cũng cung cấp các trừu tượng như vị trí hiện tại, giới hạn, công suất, v.v.**

**+, ByteBuffer.allocate: Phương thức phân bổ () của lớp java.nio.bytebuffer được sử dụng để phân bổ bộ đệm byte mới. Vị trí của bộ đệm mới sẽ bằng không, giới hạn của nó sẽ là công suất của nó, nhãn hiệu của nó sẽ không được xác định và mỗi phần tử của nó sẽ được khởi tạo về 0. Nó sẽ có một mảng sao lưu và phần bù mảng của nó sẽ bằng không.**

**+, SimpleFileVisitor: Một khách truy cập của các tập tin. Việc triển khai giao diện này được cung cấp cho các phương thức File.WalkFiletree để truy cập từng tệp trong một cây tệp. Ví dụ sử dụng: Giả sử chúng ta muốn xóa một cây tệp. Trong trường hợp đó, mỗi thư mục nên bị xóa sau khi các mục trong thư mục bị xóa.**

**+, PathMatcher là một giao diện được giới thiệu trong JDK 7. Nó có một phương thức khớp () phù hợp với mẫu với đường dẫn đã cho. PathMatcher được khởi tạo bởi GetPathMatcher () là phương pháp của Java**

**Unit8:**

**+, GetPort: Trả về số hiệu cổng sử dụng trong một chuỗi đĩa chỉ URL**

Graphical user interface, text, application, email

Description automatically generated

+, URlConnection: Mở một kết nối tới URL, cho phép một client giao tiếp với nguồn

+, **InputStream** : **là một lớp nằm trong package java.io, nó là một lớp cơ sở đại diện cho một dòng chảy của các bytes (stream of bytes) có được khi đọc một nguồn dữ liệu nào đó, chẳng hạn file**.

+, **Getheaderfield** (int n) là phương pháp của lớp httpurlconnection. Phương pháp này được sử dụng để có trường tiêu đề N. Nó trả về 0 khi nó là trường tiêu đề đặc biệt, tức là, trường tiêu đề 0. Phương pháp này cũng có thể được sử dụng với phương thức GetheaderFieldKey để có được tất cả các tiêu đề của tin nhắn.

+, **Biconsumer** là một giao diện chức năng, đại diện cho một toán tử chấp nhận hai tham số đầu vào và không trả về không có gì.

+, **DataOutputStream** thường được sử dụng để ghi các dữ liệu nguyên thuỷ (primitive data) vào một **OutputStream** khác. Sau đó, một ứng dụng có thể sử dụng **DataInputStream** để đọc dữ liệu trở lại.

+, **Registry**: Đăng ký là một giao diện từ xa đối với một cơ quan đăng ký đối tượng từ xa đơn giản, cung cấp các phương thức để lưu trữ và truy xuất các tham chiếu đối tượng từ xa bị ràng buộc với các tên chuỗi tùy ý.

**Unit9:**

+, Phân biệt Runnable với Thread:

Thread: tạo luồng chạy độc lập với nhau

Runable: chạy cùng lúc

+, **Frame: K**hung là một lớp của Java. Gói AWT. Một khung là một cửa sổ có thể thay đổi và di chuyển với một thanh tiêu đề và tối đa hóa, giảm thiểu và đóng các nút. Một khung là một thùng chứa có thể có menubars, menuitems và một số thành phần khác như nhãn, nút, trường văn bản, nút radio, v.v.

+, **Stream**: Dùng để thao tác với mảng dễ dàng hơn

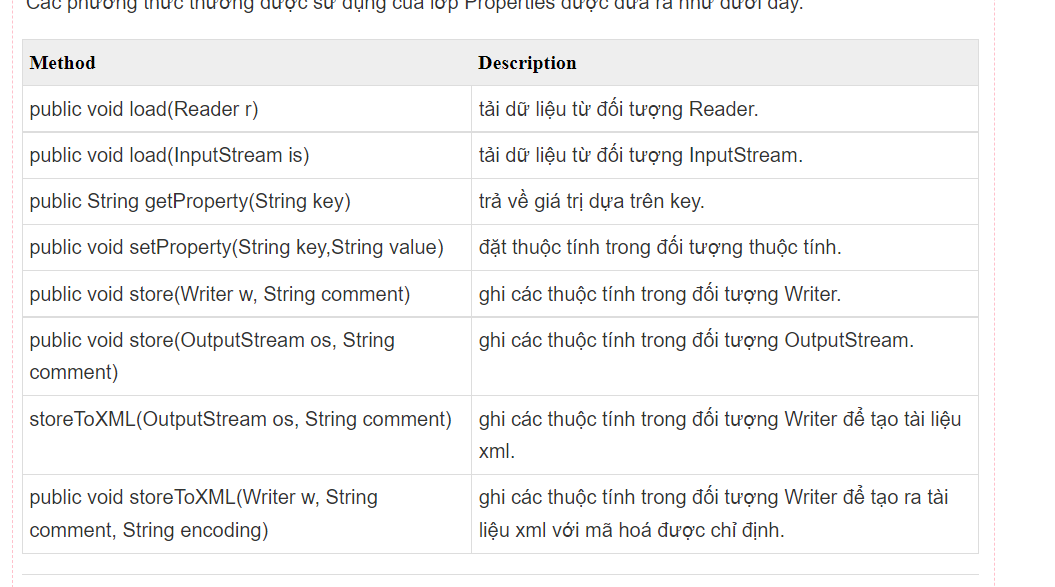
+, **getBytes():** Phương thức getBytes() trả về mảng byte của chuỗi.

+, **StringBuilder và StringBuffer:** Sử dụng khi có nhu cầu thay đổi liên tục.

StringBuilder: không có đồng bộ trong hàm

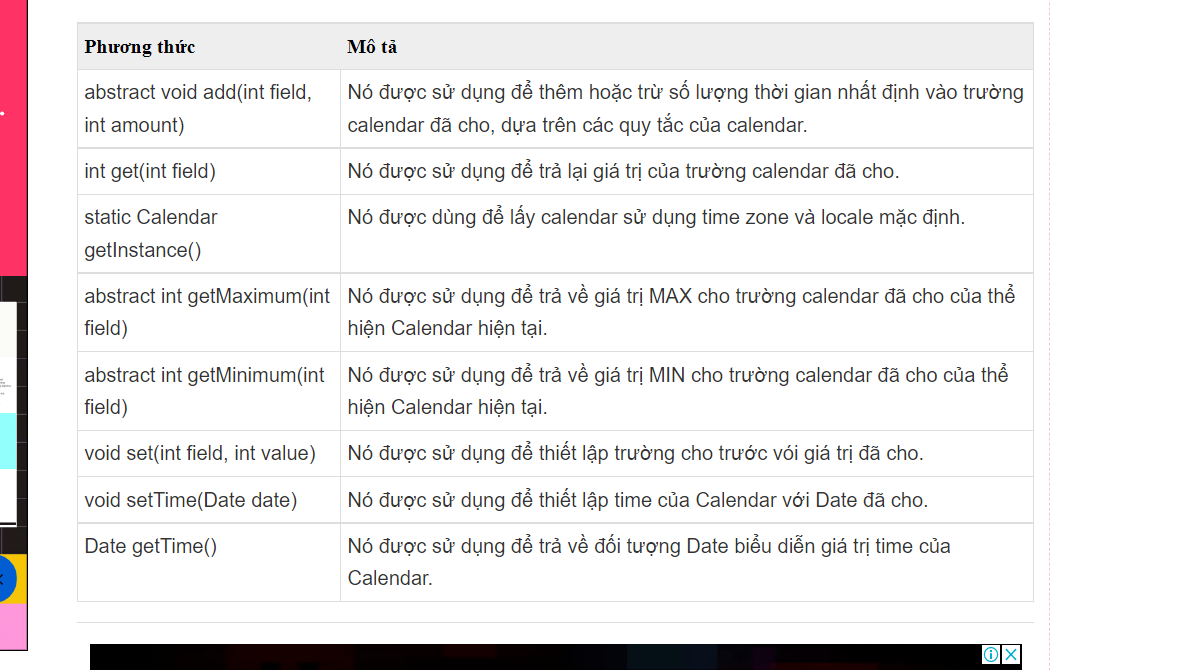
StringBuffer: có đồng bộ trong hàm

**+,** **GetProperty():** Phương thức System.getProperties() được sử dụng để lấy ra các giá trị thuộc tính của hệ thống máy tính hiện tại.



+, Locale(): Đối tượng lớp địa phương Java đại diện cho một khu vực địa lý, văn hóa hoặc chính trị cụ thể. Nó là một cơ chế để xác định các đối tượng, không phải là một container cho chính các đối tượng. Một đối tượng địa phương bao gồm các trường như ngôn ngữ, tập lệnh, quốc gia, biến thể, phần mở rộng.

+, Calendar(): Lớp **Calendar trong java** là một lớp trừu tượng (abstract) cung cấp phương thức chuyển đổi ngày giữa một thời điểm cụ thể theo thời gian và một tập hợp các trường của calendar như MONTH, YEAR, HOUR, ... Nó kế thừa lớp Object và implements giao diện Comparable.



+, **getDisplayName ()** của lớp java.util.calendar trả về chuỗi biểu diễn chuỗi của giá trị trường lịch được truyền dưới dạng tham số theo một kiểu và cục bộ nhất định. Phong cách và địa phương cũng được thông qua như tham số. Nếu không có biểu diễn chuỗi nào có sẵn, getDisplayName () trả về giá trị null. Nếu biểu diễn chuỗi có sẵn, phương thức Get () được gọi để truy cập giá trị trường lịch.

+, **Enum trong java** là một kiểu dữ liệu đặc biệt của Java được sử dụng để định nghĩa các tập hợp các hằng số. Cụ thể hơn, Java enum là một **kiểu đặc biệt** của lớp trong java. Một enum có thể chứa các trường, phương thức và Constructor.

Nó có thể được sử dụng để định nghĩa các ngày trong tuần (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY), các mùa trong năm (SPRING, SUMMER, FALL, WINTER), ...

**+, Autoboxing và chuyển đổi kiểu dữ liệu**

* **Boxing: tự động chuyển đổi từ kiểu cơ bản thành kiểu đối tượng**
* **Unboxing: chuyển từ đối tượng thành cơ bản**
* **Wraper class: chuyển đổi kiểu dữ liệu**
* **Graphical user interface, text, application

  Description automatically generated**

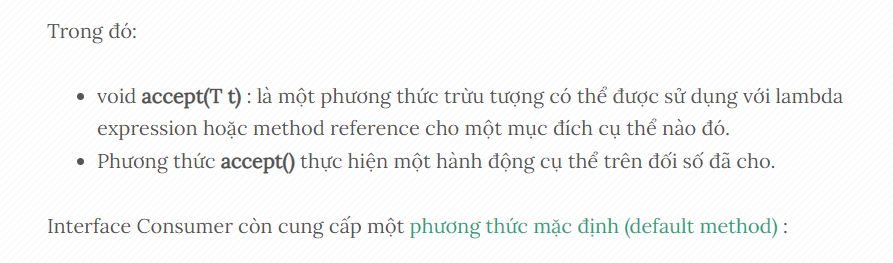
**+, AtomicInteger** là một lớp được thiết kế đặc biệt để cập nhật số nguyên theo cách an toàn theo luồng.

+, **Preferences: Lớp này cho phép các ứng dụng lưu trữ và truy xuất dữ liệu cấu hình và ưu tiên của người dùng và hệ thống**

**+**,  **Comparator trong java** được sử dụng để sắp xếp các đối tượng của lớp do người dùng định nghĩa (user-defined). Giao diện này thuộc về gói java.util và chứa hai phương thức có tên compare(Object obj1,Object obj2) và equals(Object element).

**+, Stream**: là class tổng hợp với các phương thức để dễ dàng làm việc với mảng

+, **Consumer<T>** là một [functional interface](https://gpcoder.com/3869-functional-interface-trong-java-8/) và do đó nó có thể được sử dụng với [lambda expression](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/) hoặc [method reference](https://gpcoder.com/3882-phuong-thuc-tham-chieu-trong-java-8-method-references/) cho một mục đích cụ thể nào đó. **Consumer<T>** chấp nhận một tham số đầu vào, và method này không trả về gì cả.

Text

Description automatically generated

## +, References là gì:

## Graphical user interface, text, application Description automatically generated

**+, Optional** trong Java 8 là một Container object, nó bao bọc (warpper) một object, object này có thể present hoặc không. Khi object là null thì Optional trả về empty. Đặc biệt Optional hỗ trợ lambda expression.

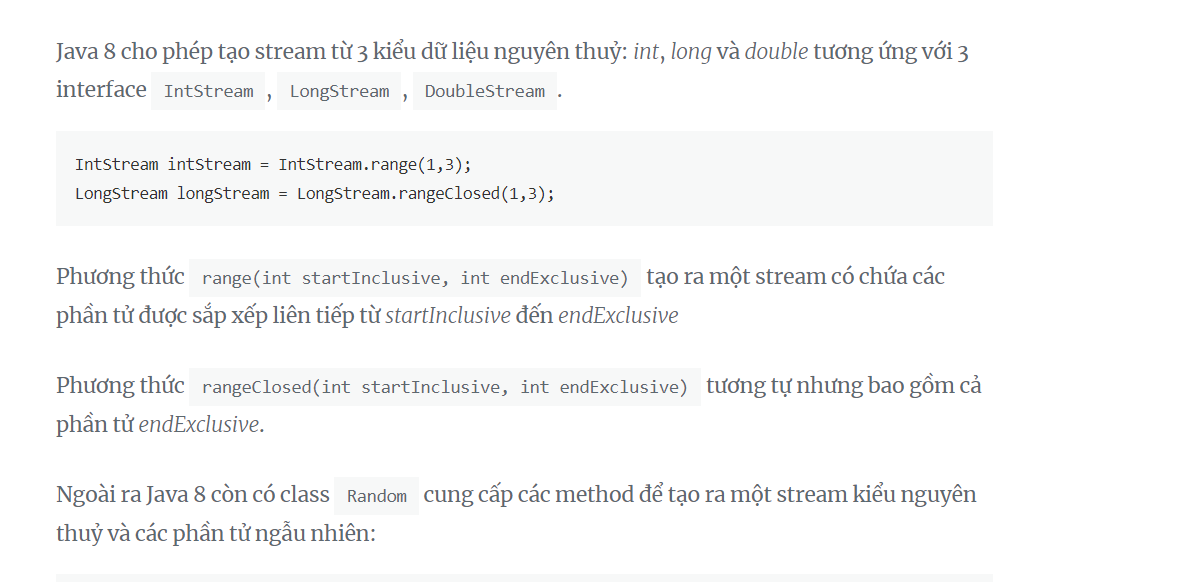
+, **Predicate** là một functional interface, đại diện cho một toán tử chấp nhận một tham số đầu vào và trả về một giá trị boolean. Ví dụ: Tạo một Predicate để kiểm tra một số có phải là số lẻ hay không.

+, filter(lamdas)

Graphical user interface, text, application, email

Description automatically generated

**+, System. currentTimeMillis(**) là một method tĩnh của class System. Nó **trả về khoảng thời gian bằng mili giây tính từ ngày 1-1-1970 cho tới thời điểm hiện tại**.



+, Vector(): Mọi hàm trong vector đã được đồng bộ hóa

+, Iterator(): lặp các phần tử và xóa trong list

+, **Iterator** trong Java **là một interface được sử dụng để thay thế Enumerations trong Java Collection Framework**. Bạn có thể sử dụng interator để: Duyệt các phần tử từ đầu đến cuối của một collection. Iterator cho phép xóa phần tử khi lặp một collection.

+, **keySet() trả về một danh sách các key chứa trong Map dưới dạng Set.**

**+, entrySet() trả về danh sách các cặp key-value chứa trong Map.**

Text

Description automatically generated

+, **Stream.collect()** là một trong các phương thức đầu cuối (terminal operation) của [**Stream API**](https://gpcoder.com/3923-gioi-thieu-ve-stream-api-trong-java-8/) trong Java 8. Nó cho phép thực hiện các thao tác có thể thay đổi trên các phần tử được lưu giữ trong Stream. Chẳng hạn như: chuyển các phần tử sang một số cấu trúc dữ liệu khác, áp dụng một số logic bổ sung, tính toán, …

+, **SystemTray** đại diện cho khay hệ thống cho máy tính để bàn. Trên Microsoft Windows, nó được gọi là "Vùng trạng thái thanh tác vụ", trên Gnome nó được gọi là "Vùng thông báo", trên KDE nó được gọi là "Khay hệ thống". Khay hệ thống được chia sẻ bởi tất cả các ứng dụng chạy trên máy tính để bàn.

+, **Robot:** Lớp này được sử dụng để tạo các sự kiện đầu vào của hệ thống gốc nhằm mục đích tự động hóa kiểm tra, trình diễn tự chạy và các ứng dụng khác khi cần điều khiển chuột và bàn phím. Mục đích chính của Robot là tạo điều kiện thuận lợi cho việc kiểm tra tự động việc triển khai nền tảng Java.

**+, GENERIC: Viết code và tái sử dụng cho rất nhiều kiểu,** Thuật ngữ “**generics**” được hiểu là tham số hóa kiểu dữ liệu. Việc tham số hóa kiểu dữ liệu giúp cho lập trình viên có thể dễ bắt lỗi các kiểu dữ liệu không hợp lệ, đồng thời giúp dễ dàng hơn cho việc tạo và sử dụng các class, interface, method với nhiều kiểu dữ liệu khác nhau.

**STACK:**

**+, REGEX**

**JDBC**

+, Giao diện DatabaseMetaData trong java **cung cấp các phương thức để lấy metadata của cơ sở dữ liệu như tên sản phẩm cơ sở dữ liệu, phiên bản sản phẩm cơ sở dữ liệu, tên driver, tên của tổng số bảng, tên của tổng số các view**, ...

+, Một đối tượng JDBCROWSET là một đối tượng kết quả nâng cao. Nó duy trì kết nối với nguồn dữ liệu của nó, giống như đối tượng resultset. Sự khác biệt lớn là nó có một tập hợp các thuộc tính và cơ chế thông báo người nghe làm cho nó trở thành một thành phần Javabeans.

## +, Java JdbcRowSet

Nó là một đối tượng được cải tiến ResultSet được sử dụng để duy trì kết nối đến một nguồn dữ liệu. Nó tương tự như ResultSet, nhưng sự khác biệt lớn là nó cung cấp tập hợp các thuộc tính và listener như một JavaBeans. Mục đích chính của JdbcRowSet là làm cho một ResultSet có thể cuộn và cập nhật được.

+, Khi nào nên dùng StringBuilder và StringBuffer:

* Trong đa luồng dùng StringBuffer
* Trong đơn luồng dùng StringBuilder

Diagram

Description automatically generated

Text, letter

Description automatically generated

Diagram

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

SPRING

## @Controller

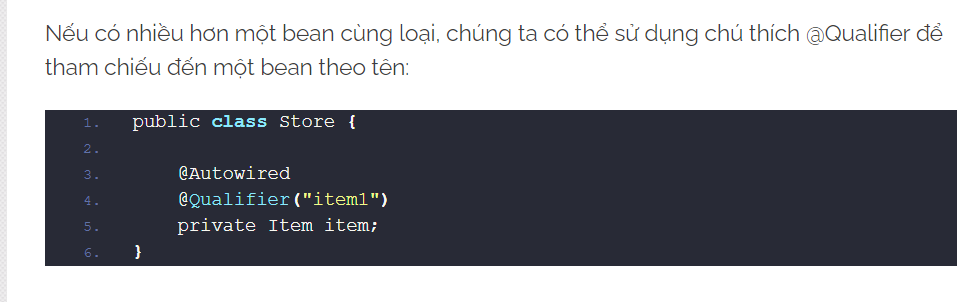


PHÂN BIỆT @COMPONENT VÀ @BEAN:

* Điều đầu tiên đó @Bean được sử dụng cho các hàm thay vì class như @Component, [hàm](https://shareprogramming.net/ham-trong-java/) này sẽ trả về một [object](https://shareprogramming.net/object-va-class-trong-java/) sẽ được đăng ký và quản lý bởi Spring IoC container.<https://shareprogramming.net/dung-lau-nhung-ban-co-phan-biet-duoc-component-va-bean-trong-spring-khong/>
* Bạn phải sử dụng @Bean trong các @Configuration class để Spring có thể quét và khởi tạo các bean được khai báo bên trong.
* Bạn có thể sử dụng @Bean trong trường hợp bạn đang sử dụng các class thư viện bên thứ ba, lúc này bạn không thể khai báo @Component trên những class này mà bắt buộc bạn phải tạo một method trả về các instance của chúng và được chú thích bởi @Bean annotation.
* Một trong những lợi ích quan trọng mà Spring mang lại là [Dependency Injection](https://shareprogramming.net/gioi-thieu-inversion-of-control-va-dependency-injection-trong-spring/) cho phép khởi tạo và quản lý các [bean](https://shareprogramming.net/tai-sao-spring-bean-duoc-goi-la-xuong-song-cua-ung-dung/) tự động. Các bean được tạo ra sẽ được quản lý bởi Spring IoC container, để sử dụng, chúng ta chỉ cần khai báo các bean với @Autowired annotation, IoC sẽ tự động tìm kiếm và tiêm chúng vào class mà chúng ta cần sử dụng.
* @Configuration annotation đại diện cho một class định nghĩa các Bean.

## Autowiring Dependencies(link: https://shareprogramming.net/gioi-thieu-inversion-of-control-va-dependency-injection-trong-spring/)

## Nếu có nhiều hơn một bean cùng loại, chúng ta có thể sử dụng chú thích @Qualifier để tham chiếu đến một bean theo tên:

* 
* Text

  Description automatically generated

Trong Spring, để tạo ra một bean chúng ta phải thông qua 2 annotation @Bean và @Component, do vậy thông thường chúng ta sẽ gặp 2 annotation này rất nhiều khi làm việc với Spring.

+, Spring bean:

## Spring Bean là gì?( https://shareprogramming.net/tai-sao-spring-bean-duoc-goi-la-xuong-song-cua-ung-dung/)

Spring Beans chính là những [Java Object](https://shareprogramming.net/object-va-class-trong-java/) mà từ đó tạo nên khung sườn của một ứng dụng Spring. Chúng được cài đặt, lắp ráp và quản lý bởi Spring IoC container.

IOC(): [Inversion of Control](https://shareprogramming.net/gioi-thieu-inversion-of-control-va-dependency-injection-trong-spring/) viết tắt là IoC là một quá trình trong đó một object định nghĩa các dependency (phụ thuộc) mà không cần phải khởi tạo chúng.Việc khởi tạo các dependency này được chuyển giao cho IoC container

+, DataSource dùng để lưu trữu các thông tin kết nối đến database

+, LEANING IN COMPANY

* Stateless and Statefull:( <https://topdev.vn/blog/stateless-va-stateful-la-gi/>)
* Stateless là design không lưu dữ liệu của client trên server. Có nghĩa là sau khi client gửi dữ liệu lên server, server thực thi xong, trả kết quả thì “quan hệ” giữa client và server bị “cắt đứt” – server không lưu bất cứ dữ liệu gì của client.
* Stateful là một design ngược với stateless, server cần lưu dữ liệu của client, điều đó đồng nghĩa với việc ràng buộc giữa client và server vẫn được giữ sau mỗi request (yêu cầu) của client. Data được lưu lại phía server có thể làm đầu vào (input parameters) cho lần kế tiếp, hoặc là dữ kiện dùng trong quá trình xử lý hay phục vụ cho bất cứ nhu cầu nào phụ thuộc vào bussiness (nghiệp vụ) cài đặt.

+, Đồng bộ và bất đồng bộ trong java (Synchronized and

Asynchronous)( https://codelearn.io/sharing/dong-bo-va-bat-dong-bo-trong-java)

## Đồng bộ trong java

* **Đồng bộ trong java** (Synchronization in java) là khả năng kiểm soát truy cập của nhiều luồng đến bất kỳ nguồn tài nguyên chia sẻ (shared resource). Giả sử có nhiều luồng muốn truy cập cùng một biến cùng một thời điểm. Ví dụ một luồng muốn đọc, trong khi luồng khác cố gắng thay đổi dữ liệu dẫn dến dữ liệu bị sai lệch.

Tại sao sử dụng Synchronization

Việc đồng bộ hóa chủ yếu được sử dụng để:

* Để tránh sự can thiệp của luồng khác.
* Để đảm bảo vấn đề nhất quán của chương trình.

Đồng bộ phương thức:

Text

Description automatically generated

Đồng bộ khối

Text

Description automatically generated

Đồng bộ lớp: Trong khi đó thì đồng bộ lớp khóa toàn bộ tài nguyên của lớp đó. Tất cả phương thức bên trong đó chỉ có thể được luồng hiện tại sử dụng. Bên dưới sẽ là ví dụ cho cách dùng này.

# **Java – Fail-fast vs Fail safe iterator(Hơi khó hiểu:** [**https://howtocodevn.wordpress.com/2015/11/03/java-fail-fast-vs-fail-safe-iterator/**](https://howtocodevn.wordpress.com/2015/11/03/java-fail-fast-vs-fail-safe-iterator/)**)**

**Sự khác biệt giữa fail-fast và fail-safe là gì?**

* Thuộc tính fail-safe của Iterator hoạt động với bản sao của collection và do đó, nó không bị ảnh hưởng bởi bất kỳ sửa đổi nào trong collection.
* Tất cả các lớp collection trong gói java.util là fail-fast, trong khi các lớp collection trong java.util.concurrent là fail-safe.
* Các Fail-fast iterators thảy ra một ConcurrentModificationException, trong khi Fail-safe iterators không bao giờ thảy một exception như vậy.

ConcurrentModificationException trong Java

* **Bất cứ khi nào chúng ta cố gắng sửa đổi đồng thời một đối tượng mà không được phép, thì lỗi ConcurrentModificationException sẽ xảy ra**. Chúng ta thường gặp phải ngoại lệ này khi chúng ta làm việc với các lớp Collection của Java. Ngoại lệ này có trong gói java. util

Có hai loại lỗi chính trong java:

* Nguy hiểm thực sự: Runtime error
* Tỏ ra nguy hiểm: Complain error

+, ThreadPool:

Thread Pool Là một trong những yếu tố chính tác động tới hiệu năng của các chương trình lớn, đòi hỏi xử lý đồng thời nhiều nhiệm vụ cùng lúc.

Một ví dụ đơn giản nhé (Trong thực tế sẽ khác, hãy coi đây là ví dụ nha):

Bây giờ, giả sử bạn có một Server Web. Nếu chúng ta nhận 1 request từ client, chúng ta sẽ xử lý mất 0.5s và trả về kết quả cho người dùng.

Thế nếu có 2 người request cùng lúc? => giải quyết bằng cách mỗi một request sẽ xử lý ở 1 thread, đơn giản.

Thế nếu có 100 người request cùng lúc? => mỗi người tạo một thre... wait a minute.... (nếu 1 tháng có 10M lượt request => tạo ra 10M thread)

Nếu bạn tạo 1-2 thread mới, chả ai trách gì bạn cả. Nhưng nếu bạn tạo liên tục và tới hàng trăm cái mới mỗi lần nhưng lại giải quyết cùng 1 vấn đề thì có lỗ hổng đấy. Vì chi phí của việc tạo 1 thread là tương đối lớn, thường dẫn tới các vấn đề về hiệu năng và cấp phát dữ liệu.

* + Với việc xử lý các tác vụ liên tục như vậy, có một giải pháp là sử dụng Thread Pool
* Cách giải quyết: Ở ví dụ trên, Bây giờ tôi sẽ chỉ sử dụng 30 thread thôi! Và đặt 30 thread này ở trạng thái không làm gì và vứt vào 1 cái Pool (1 cái bể chứa, kiểu vậy). Với mỗi request đến, tôi sẽ lấy trong Pool ra 1 thread và xử lý công việc, xử lý xong, thì cất thread vào ngược trở lại pool. Đơn giản vậy thôi, như thế chúng ta sẽ không phải tạo mới Thread nữa. Tránh tình tốn chi phí và hiệu năng.

### Executor

Executor là một class đi kèm trong gói java.util.concurrent, là một đối tượng chịu trách nhiệm quản lý các luồng và thực hiện các tác vụ Runnable được yêu cầu xử lý. Nó tách riêng các chi tiết của việc tạo Thread, lập kế hoạch (scheduling), … để chúng ta có thể tập trung phát triển logic của tác vụ mà không quan tâm đến các chi tiết quản lý Thread.

Hibernate là gì ? Hibernate framework được sử dụng nhiều nhất hiện nay để giúp lập trình viên Java có thể map các class với một csdl bất kỳ.

Hibernate là một thư viện ORM (Object Relational Mapping) mã nguồn mở giúp lập trình viên viết ứng dụng Java có thể map các objects (pojo) với hệ quản trị cơ sở dữ liệu quan hệ, và hỗ trợ thực hiện các khái niệm lập trình hướng đối tượng với cớ dữ liệu quan hệ.

Để giao tiếp với Hibernate, chúng ta sẽ tạo ra một Class đại diện cho một Table. Và mọi dữ liệu từ Table trong database sẽ được Hibernate bind vào Class đó cho chúng ta.

Đối tượng chính của việc truy xuất hay insert dữ liệu bằng Hibernate chính là Session và được tạo ra từ Session Factory.

Session Factory Là một interface giúp tạo ra session kết nối đến database bằng cách đọc các cấu hình trong một file xml và mỗi loại Database khác nhau sẽ có một cấu hình khác nhau.