**BÁO CÁO THỰC TẬP**

****

Họ tên : Trần Quang Thắng.

Vị trí: Thực tập sinh Java.

Thời gian bắt đầu: 14/12/2022.

Mục lục:

**Mục lục**

[Phần 1 : JAVA CORE BASIC. 4](#_Toc122967331)

[1. Kĩ thuật lập trình căn bản. 4](#_Toc122967332)

[a Download và cài đặt công cụ lập trình 4](#_Toc122967333)

[b Các kiểu dữ liệu 5](#_Toc122967334)

[c Các câu lệnh if, for, while, do-while, case switch 5](#_Toc122967335)

[d Một số giải thuật làm việc với mảng: sắp xếp (các loại sắp xếp cơ bản), thêm, xóa 1 phần tử trong mảng, … 6](#_Toc122967336)

[e Streams Input/Out put, làm việc với file mức cơ bản. 6](#_Toc122967337)

[2. Lập trình hướng đối tượng. 7](#_Toc122967338)

[a Các khái niệm cơ bản: 7](#_Toc122967339)

[b Các đặc điểm 9](#_Toc122967340)

[c Cách khai báo lớp thuộc tính, phương thức, constructor, phân chia các class 10](#_Toc122967341)

[3. Exception and handling 12](#_Toc122967342)

[a Khai báo ngoại lệ. 12](#_Toc122967343)

[b Cơ chế try-catch 12](#_Toc122967344)

[c Throw ngoại lệ 12](#_Toc122967345)

[d Chuyển tiếp ngoại lệ 13](#_Toc122967346)

[4. Lập trình xử lý với database 14](#_Toc122967347)

[a Cài đặt MySQL 14](#_Toc122967348)

[b Sử dụng MySQL 14](#_Toc122967349)

[c Kết nối MySQL 15](#_Toc122967350)

[5. Học cách làm quen và sử dụng file config trong các module: 15](#_Toc122967351)

[6. Đóng gói chương trình, run file trên windows, linux … 15](#_Toc122967352)

[7. Bài tập 15](#_Toc122967353)

[Chương 2: JAVA CORE ADVANCE 16](#_Toc122967354)

[1. Streams Input/out put, làm việc với file mức nâng cao (tự nghiên cứu sử dụng các thư viện nâng cao trong Java), ưu nhược điểm. 16](#_Toc122967355)

[2. Thread 17](#_Toc122967356)

[a. Khái niệm 17](#_Toc122967357)

[b. Có 2 cách khởi tạo thread : 17](#_Toc122967358)

[c. Thread Priority : mức độ ưu tiên của thread . 18](#_Toc122967359)

[d. Vòng đời của 1 thread, các trạng thái 19](#_Toc122967360)

[e. Deadlock : khóa chết 20](#_Toc122967361)

[f. Timer, Time Task, Schedule 23](#_Toc122967362)

[3. Lập trình mạng 25](#_Toc122967363)

[a. Socket 25](#_Toc122967364)

[b. TCP : 26](#_Toc122967365)

[c. UDP : 27](#_Toc122967366)

[d. TCP vs UDP : 28](#_Toc122967367)

[4. MultiThread . 28](#_Toc122967368)

[a. Khái niệm 28](#_Toc122967369)

[b. Tìm hiểu Queue là gì? các loại? cách làm việc với queue 29](#_Toc122967370)

[c. Thread Pool: Khái niệm, cách sử dụng, ưu nhược điểm 31](#_Toc122967371)

[d. Thế nào là đồng bộ và bất đồng bộ, làm thế nào để đồng bộ khi nhiều thread cùng sử dụng 1 tài nguyên? Các vấn đề cần xử lý khi chương trình có nhiều thread? 32](#_Toc122967372)

[5. Logger 34](#_Toc122967373)

# Phần 1 : JAVA CORE BASIC.

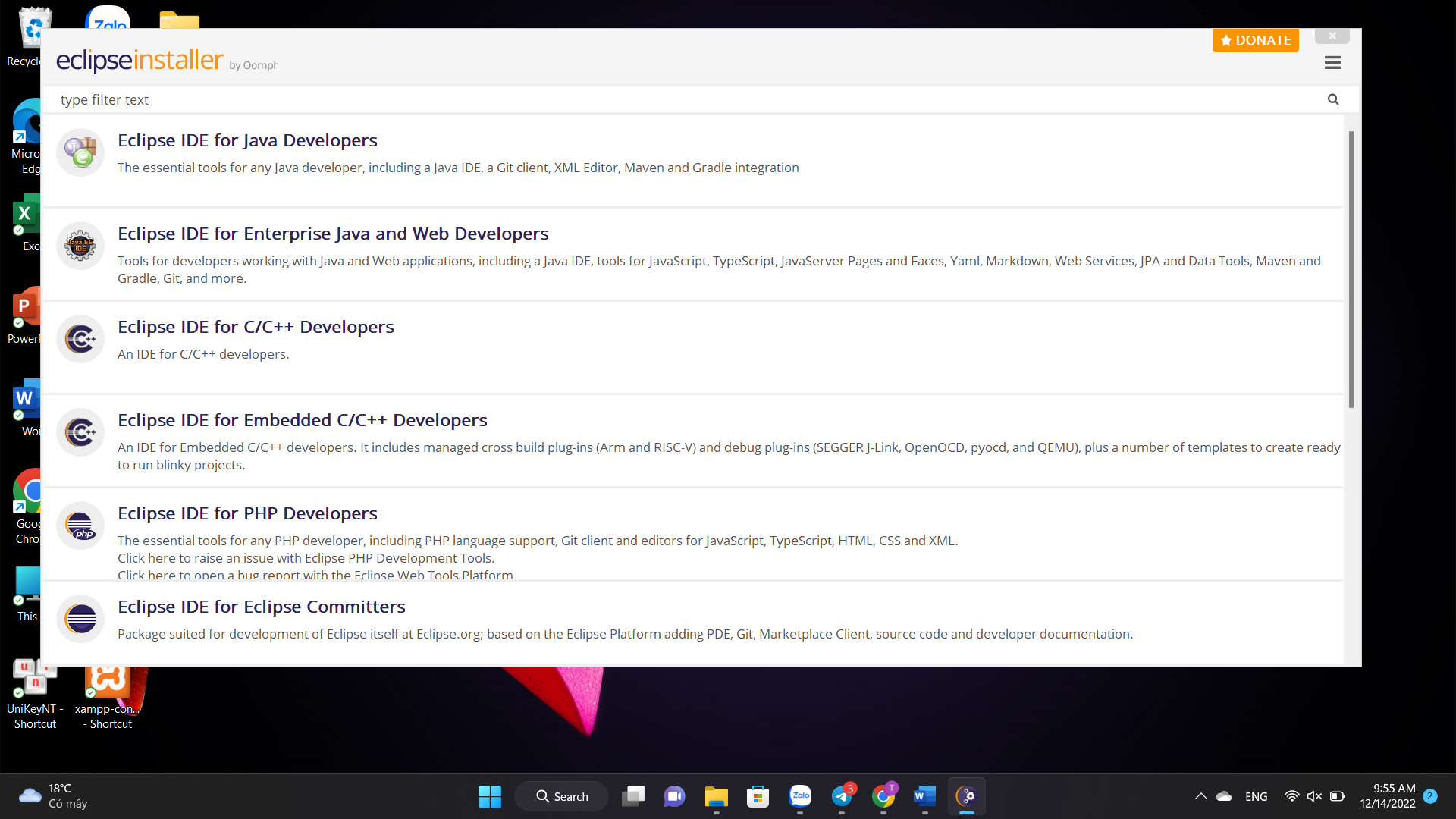
## Kĩ thuật lập trình căn bản.

### Download và cài đặt công cụ lập trình

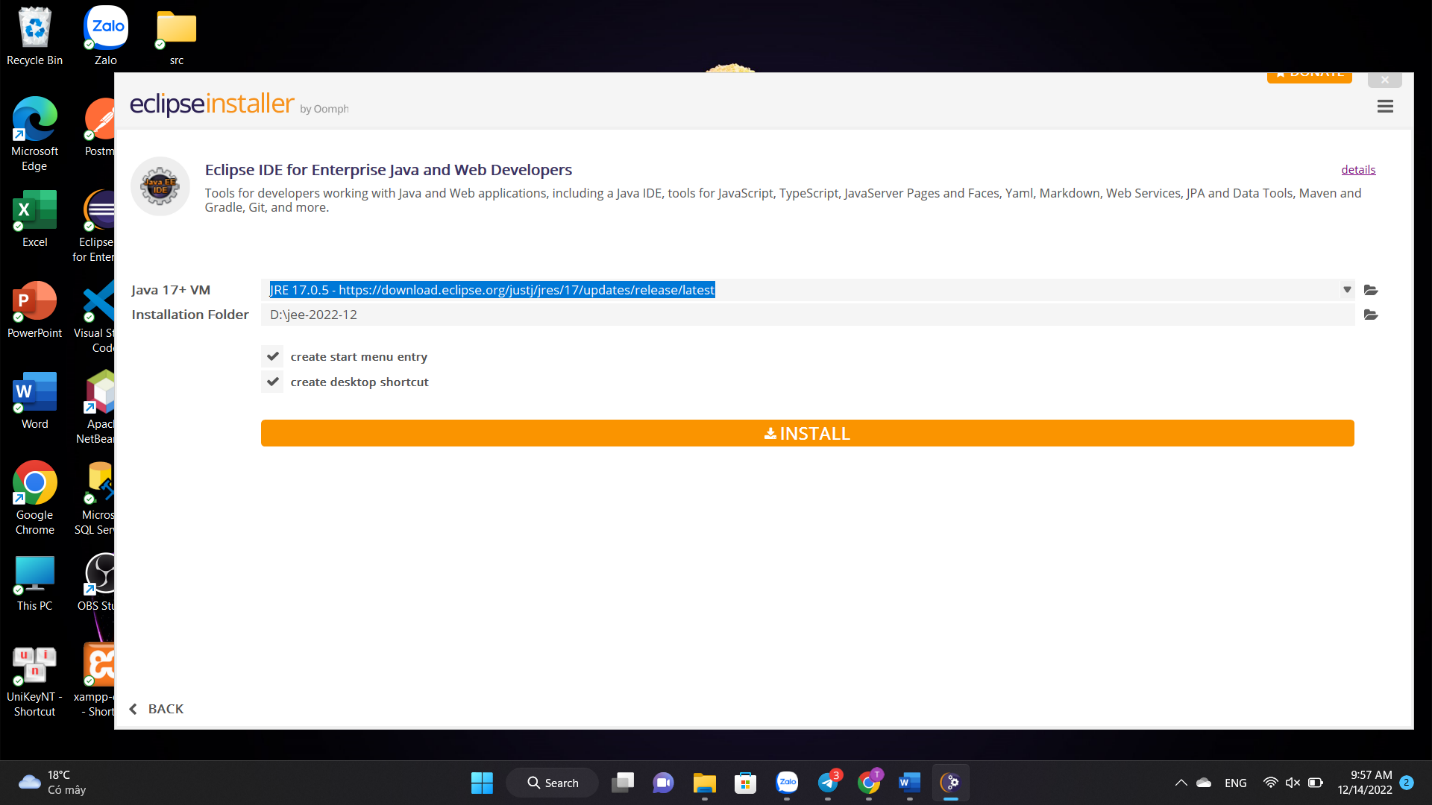
B1: Truy cập link : <https://www.eclipse.org/downloads/download.php?file=/oomph/epp/2022-12/R/eclipse-inst-jre-win64.exe&mirror_id=1287> để cài đặt bộ cài Eclipse.

B2: Mở file vừa mới tải về.

B3: chọn gói sẽ cài ( chọn ‘Eclipse IDE for Enterprise Java and Web Deverloper’ để phát triển Java Back-end , ‘Eclipse IDE for Java Deverlopers’ để phát triển Java core , Java Swing , … )



B4: chọn đỉa chỉ lưu bản cài đặt trong máy và chọn JDK phù hợp



### Các kiểu dữ liệu

Java có 2 kiểu dữ liệu:

* Kiểu nguyên thủy: int(4), float(4),double(8),long(8), boolean(1),byte(1), short(2), char(2)
* Kiểu đổi tượng : Integer, Float, … hoặc do người dùng tự định nghĩa.

### Các câu lệnh if, for, while, do-while, case switch

If : sử dụng khi cần xác định đoạn mã cần chạy với điều kiện là 1 mệnh đề trả về true false.

For: sử dụng khi cần làm một công việc nhiều lần và đã xác định được số lần thực hiện. dùng lặp mảng . Phiên bản cải tiến for-each có thể dùng để lặp list hiệu quả.

While: sử dụng khi cần làm một công việc nhiều và chưa xác định số lần chạy cụ thể

Do-While: sử dụng khi cần làm một công việc nhiều và chưa xác định số lần chạy và cần thực hiện 1 lần trước khi chạy.

Break : sử dụng để kết thúc vòng lặp và chạy đoạn mã sau vòng lặp. Continue : sử dụng để bỏ qua đoạn mã trong khổi và khởi chạy điều kiện tiếp theo.

Switch-case : sử dụng khi cần kiểm tra nhiều điều kiện.

### Một số giải thuật làm việc với mảng: sắp xếp (các loại sắp xếp cơ bản), thêm, xóa 1 phần tử trong mảng, …

Một số giải thuật :

* Bubble sort :
* Insert sort
* Quick sort

### Streams Input/Out put, làm việc với file mức cơ bản.

Sử dụng đối tượng FileOutputStream để tạo file và ghi dữ liệu vào file

VD: tạo ra file data.txt tại ổ D ( nếu chưa tồn tại) , ghi đoạn chữ ‘Xin Chao’ vào file data.txt

try {

FileOutputStream fos = new FileOutputStream(new File("D:\\data.txt"));

String data = "Xin Chao";

byte[] rs = data.getBytes();

fos.write(rs);

} catch (Exception ex) {

ex.printStackTrace();

}

Sử dụng đối tượng FileInputStream để đọc dữ liệu trong file

VD: đọc dữ liệu trong file data.txt và in ra console . Nếu file không tồn tại sẽ ném ra ngoại lệ.

try {

FileInputStream fis = new FileInputStream(new File("D:\\data.txt"));

String data = "";

int rs;

while ((rs = fis.read()) != -1) {

data += (char) rs;

}

System.out.printf("%s",data);

} catch (Exception ex) {

ex.printStackTrace();

}

## Lập trình hướng đối tượng.

### Các khái niệm cơ bản:

* Class : là khuôn mẫu thiết kế để tạo ra đối tượng . 1 file java chỉ có 1 public class và có thể có nhiều default-class, inner class. Trong class có : Thuộc tính, phương thức, constructor , khối lênh, Inner class và interface .

Constructor : là phương thức cùng tên với class , không có kiểu dữ liệu trả về và được sử dụng để khởi tạo đối tượng tứ class chứa.

Access modifier với class

|  |  |
| --- | --- |
| Access-Modifier | Description |
| Public | Có thể truy cập từ các class khác, chỉ có duy nhất trong 1 file java |
| default | Chỉ có thể truy cập trong cùng pakage |

|  |  |
| --- | --- |
| Non-Modifier | Description |
| final | Không cho phép class khác kế thừa |
| abstract | Chỉ định là abstract class ( có thể chứa các abstract method) |

Access modifier với method, attribute

|  |  |
| --- | --- |
| Access-Modifier | Description |
| public | Có thể truy cập từ các class khác trong tất cả các Pakage |
| protected | Có thể truy cập từ các class khác trong cùng Pakage và class con khác gói |
| default | Có thể truy cập từ các class khác trong cùng Pakage |
| private | Có thể truy cập bên trong class đó |

|  |  |
| --- | --- |
| Non-Modifier | Description |
| final | Method : không cho phép override  Attribute : không cho phép thay đổi giá trị |
| abstract | Method chỉ định là abstract method ( không có thân ) |
| static | Là các method , attribute dùng chung trong toàn chương trình. Có thể truy xuất trực tiếp mà không cần khởi tạo đối tượng. |

VD: khai báo class MyObject với những thuộc tính : id,name,value ; những phương thức getter setter tương ứng và constructor với đầy đủ tham số.

package JavaCore;

/\*\*

\*

\* @author thang

\*/

public class MyObject {

public Long id;

public String name;

public Integer value;

// constructor duoc su dung de khoi tao doi tuong

public MyObject(Long id, String name, Integer value) {

this.id = id;

this.name = name;

this.value = value;

}

@Override

public String toString() {

return "MyObject{" + "id=" + id + ", name=" + name + ", value=" + value + '}';

}

}

* Đối tượng : là một thực thể thể hiện của một class, với những giá trị cụ thể

VD: Khởi tạo đối tượng obj từ class MyObject trên : trong đó (id : 3, name : ‘Xtel’,value:5)

MyObject object =new MyObject(3l, "Xtel", 5);

### Các đặc điểm

Các đặc điểm : Kế thừa, đa hình, trừu tượng, đóng gói.

* Kế thừa: cho phép class khác sử dụng những phương thức , thuộc tính ,constructor public , protected, và khi cùng gói.Java chỉ hỗ trợ đơn kế thừa class nhưng có thể đa kế thừa interface. Có thể định nghĩa kiểu đối tượng con với kiểu dữ liệu của class cha. -> tái sử dụng code
* Trừu tượng: thể hiện qua abstract class và interface. Trong đó ta sẽ định nghĩa những phương thức không có thân. Khi cần thực thi thì ta sẽ định nghĩa nghiệp vụ cụ thể cho các phương thức đó.-> tăng khả năng mở rộng của hệ thống, tránh việc phụ thuộc trực tiếp giữa các class với nhau
* Đa hình: phương thức có thể được xử lý theo nhiều cách khác nhau. thể hiện qua overloading và overriding. Overloading thì định nghĩa những phương thức cùng tên nhưng khác tham số truyền vào, thứ tự, Overriding định nghĩa lại nội dung xử lý phương thức (phải có cùng tham số truyền vào , cùng kiểu dữ liệu trả về hoặc class con , cùng ngoại lệ hoặc ngoại lệ con)
* Đóng gói: các thuộc tính sẽ được đặt phạm vi truy cập là private và được sử dụng thông qua getter và setter. Có thể tùy chỉnh các trường là : chỉ đọc và ta có thể kiểm tra dữ liệu đầu vào trước khi việc thay đổi thuộc tính diễn ra.

So sánh overloading và overriding

|  |  |
| --- | --- |
| **Overloading** | **Overloading** |
| Định nghĩa những phương thức cùng tên nhưng khác tham số truyền vào, thứ tự. | Định nghĩa lại nội dung xử lý phương thức (phải có cùng tham số truyền vào , cùng kiểu dữ liệu trả về hoặc class con , cùng ngoại lệ hoặc ngoại lệ con) |
| Thể hiện tính đa hình lúc compiler | Thể hiện tính đa hình lúc runtime |
| Phạm vi : cùng class hoặc được kế thừa | Phạm vi : phải được kế thừa. |

So sánh abstract class với interface

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| Có thể chứa abstract method và non-abstract method | Chỉ chứa abstract method. Từ Java 8 trở đi có thêm static và default. |
| Có thể chưa các biến toàn cục chỉ khai báo,khởi tạo, | Chỉ chứa các biến toàn cục đã khởi tạo giá trị . Từ Java 8 trở đi có thêm static và default. |
| Có constructor | Không có constructor -> không thể khởi tạo. |
| Thực thi qua từ khóa extends | Thực thi qua từ khóa Implements |
| Có thể extends class khác , implement interface khác | Có thể extends nhiều interface |
| Chỉ hỗ trợ đơn kế thừa | Hỗ trợ đa kế thừa |
| Biến và phương thức có thể có phạm vi truy cập : public, protected, private | Biến và phương thức chỉ có phạm vi truy cập là public |

### Cách khai báo lớp thuộc tính, phương thức, constructor, phân chia các class

VD:

package JavaCore;

public class MyObject {

// khai bao thuoc tinh

public Long id;

public String name;

public Integer value;

public InnerClass innerClass;

// constructor dduuoc su dung de khoi tao doi tuong

public MyObject(Long id, String name, Integer value) {

this.id = id;

this.name = name;

this.value = value;

innerClass = new InnerClass(name);

}

// khai báo phuong thuc

public void print(){

System.out.println(this.toString());

}

@Override

public String toString() {

return "MyObject{" + "id=" + id + ", name=" + name + ", value=" + value + '}';

}

// inner class

class InnerClass{

public String data;

public InnerClass(String name) {

this.data = name;

}

}

}

## Exception and handling

### Khai báo ngoại lệ.

Exception là các trường hợp bất thường xảy ra trong lúc runtime làm gián đoạn hoạt động của chương trình. Java cung cấp cơ chế xử lý exception qua try-catch.

Có 2 loại Exception : Built-in Exception và User-defined Exceeption.

User-defined Exceeption : Khai báo ngoại lệ ta cần phải cho class đó extends class Exception.

VD :

public class CustomException extends Exception{

private String name;

public CustomException(String name) {

super("Exception : " +name);

this.name = name;

}

}

### Cơ chế try-catch

Có 2 lỗi : exception và error . Cả 2 là class con của class Throwable.Error là vấn đề nghiêm trọng và không thể xử lý với try-catch . Exception là vấn đề ít nghiêm trọng hơn và có thể xử lý bằng try-catch.

Có 2 loại exception :

CheckedException : xảy ra lúc compiler . Cần xử lý mới có thể chạy chương trình được

UncheckedException : xảy ra lúc runtime.

Được sử dụng để xử lý exception . Khối try chứa đoạn mã có thể gây ra exception, khối catch sẽ chỉ định loại exception sẽ xảy ra và nghiệp vụ xử lý. Exception con phải được catch trước exception cha. Nếu exception ở khối catch trên đã xảy ra thì nó sẽ bỏ qua exception ở khối catch dưới.

### Throw ngoại lệ

Có 2 từ khóa để ném ra ngoại lệ :

throw: ném ra 1 exception từ block, bên trong method. Nó có thể ném ra CheckedException và UncheckedException.Sau khi ném ra exception , Java sẽ dừng luồng thực thi (dừng việc thực hiện công việc bên dưới ) và tìm khổi catch phù hợp với kiểu exception và chạy vào đó nếu không thì chương trình xử lý exception của java sẽ tạm dừng chương trình.

VD :

public void method1(){

try {

int i = 1/0;

} catch (Exception e) {

throw new RuntimeException();

}

}

throws: ném ra một hoặc nhiều ngoại lệ khai báo trên method. Nơi nào gọi method thì phải xử lý try-catch hoặc tiếp tục throws . Nhưng cuối cùng sẽ phải có 1 xử lý try-catch. Throws chỉ được yêu cầu cho checkedException.

public void method2() throws RuntimeException {

int i = 1 / 0;

}

### Chuyển tiếp ngoại lệ

Chuyển tiếp exception sang xử lý ở phương thức khác ( phương thức) qua việc throw exception .

VD:

// Chuyen tiep ngoai le

public void method2() throws RuntimeException {

int i = 1 / 0;

}

// Xu ly chuyen tiep ngoai le

public void method3(){

try {

method2();

} catch (Exception e) {

// xu lý

System.out.println("can not divide 0");

}

}

## Lập trình xử lý với database

### Cài đặt MySQL

### Sử dụng MySQL

Tạo database (CREATE TABLE database\_name),

Tạo bảng : CREATE TABLE table\_name( column type\_data constraint, column2 type\_data constraint)

Tạo khóa chính: ALTERT TABLE table\_name ADD PRIMARY KEY (column) : được dùng để định danh bản ghi đó , không được null và phải unique

Tạo khóa ngoại : ALTERT TABLE table\_name ADD CONSTRAINT name\_constraint FOREIGH KEY (column) REFERENCES reference\_table(

reference \_columns) : được dùng để kết nối dữ liệu giữa 2 bảng , tăng tính nhất quan của các bảng.

Store procedure :

DELIMITER $$

CREATE PROCEDURE `getStudents` ()

BEGIN

--query

END$$

DELIMITER ;

Có thể không cần trả về giá trị. Nó định nghĩa tập hợp câu lệnh sql và được biên dịch và lưu trữ trong CSDL. Nó giúp tăng tốc truy vấn ( không phải biên dịch lại), giảm lưu lượng giao tiếp giữa client với database server, đơn giản hơn so với việc thực hiện bên client .

Function : là một loại stored procedure. Có nhiệm vụ xác định và bắt buộc phải trả về một giá trị đơn. Không chứa câu lệnh Insert, Update, Delete một bảng, view trong CSDL.

Views : là bảng ảo dựa trên kết quả của câu truy vấn, các trường trong nó là các trường của 1 hay nhiều bảng thật trong DB và nó có thể tương tác

### Kết nối MySQL

## Học cách làm quen và sử dụng file config trong các module:

## Đóng gói chương trình, run file trên windows, linux …

## Bài tập

# Chương 2: JAVA CORE ADVANCE

## Streams Input/out put, làm việc với file mức nâng cao (tự nghiên cứu sử dụng các thư viện nâng cao trong Java), ưu nhược điểm.

Java có các class làm việc với I/O stream lằm trong pakage java.io

Trong Java có 2 loại luồng chính : Byte stream và Character stream .

* Byte stream:

Đọc ghi dữ liệu theo từng byte ( 8bits ). Thường được sử dụng để đọc ghi dữ liệu nhị phân được kế thừa từ abstract class Inputstream và outputstream.

Một số class thường gặp :

FileOutputStream/FileInputStream : được sử dụng để tạo luồng đọc ghi file từ với đạng chuỗi byte (byte thô) như ảnh .

BufferedInputStream-BufferedOutputStream: không làm việc trực tiếp với file mà làm việc với dữ liệu ở vùng bộ nhớ buffer. Đọc nhiều byte một lúc -> Tăng tốc độ đọc ghi file.

DataInputStream-DataOutputStream : thường được sử dụng để đọc ghi dữ liệu nguyên thủy của Java một cách linh hoạt.

ObjectInputStream – ObjectOutputStream : được đùng để đọc, ghi các đối tượng trong Java . Để có thể ghi đối tượng ra file thì class tạo ra đối tượng đó phải implement serializable .

* Character stream:

Đọc ghi dữ liệu theo từng ký tự (thường là 16bits ). Thường được sử dụng để đọc ghi dữ liệu ký tự. được kế thừa từ abstract class Reader và Writer.

Một số class thường gặp :

FileReader – FileWriter: được dùng để đọc ghi file với định dạng kí tự . với việc ghi file thì có thể ghi trực tiếp chuỗi mà không cần chuyển thành mảng byte.

BufferedReader – BufferedWriter: làm việc với dữ liệu vùng bộ nhớ đệm ( ta có thể quy định kích thước ) có thể đọc nhiều character trong 1 lần -> tăng hiệu suất

InputStreamReader – InputStreamWriter:được sử dụng để đọc ghi file từ kiểu character sang kiểu byte và ngược lại. Nó được coi là cầu nối giữa Byte stream và Character stream.

Ưu điểm :

* Byte stream : Mạnh mẽ linh hoạt
* Character stream : đọc ghi file hỗ trợ unicode.

Nhược điểm :

* Byte stream : hỗ trợ đọc ghi file unicode kém.
* Character stream : chậm hơn, tốn bộ nhớ hơn.

## Thread

### Khái niệm

Thread là một tiểu trình nhỏ ( light process) thực hiện 1 nhiệm vụ cụ thể. Các thread tồn tại trong một tiến trình (process) và có thể chia sẻ tài nguyên với các thread khác. Các thread hoạt động độc lập, nếu xảy ra ngoại lệ sẽ không ảnh hưởng tới thread khác

Có 2 loại Thead : User Thread và Daemon Thead ( thread chạy nền). JVM sẽ chỉ chờ user thread chay xong để tiếp tục thread khác chứ không chờ daemon thread.

### Có 2 cách khởi tạo thread :

extends class Thread hoặc implemets interface Runable.

|  |  |
| --- | --- |
| Thread | Runnable |
| Extends Thread, Không cho phép extends class khác nữa | Implements Runable, Có thể extends class khác |
| Là một thread | Không phải là thread |
| Có thể override thêm một số phương thức khác : isLive(), start() ,… | Chỉ có thể override duy nhất phương thức run() . |
| Khi khởi tạo thread thì sẽ khởi tạo đối tượng extends Thread | Khi khởi tạo thread thì sẽ khởi tạo đối tượng implement runnable và truyền qua constructor khởi tạo Thread:  Thread(runnable);  Thread(runnable,name); …. |
| Kém linh động. | Linh động hơn. |
| Đổi tượng được tạo là duy nhất | Đổi tượng được tạo có thể được chia sẻ với nhiều luồng khác nhau. |

Main thread : là thread được JVM tạo ra , bắt đầu ngay khi chương trình Java chạy và sẽ hoàn thành cuối cùng .Từ main các thread con mới có thể sinh ra

### Thread Priority : mức độ ưu tiên của thread .

Thread scheduler là 1 thành phần trong JVM .Việc quản lý việc chạy hay đợi của các Thread được quyết định bởi Thread scheduler. Tại 1 thời điểm chỉ có thể chạy và xử lý 1 thread duy nhất.

Thread scheduler căn cứ vào : priority(+) và arrival time ( thời gian đến của luồng)

Trong trường hợp xảy ra tranh chấp tài nguyên thì thread có độ ưu tiên cao hơn sẽ ưu tiên được thực hiện .

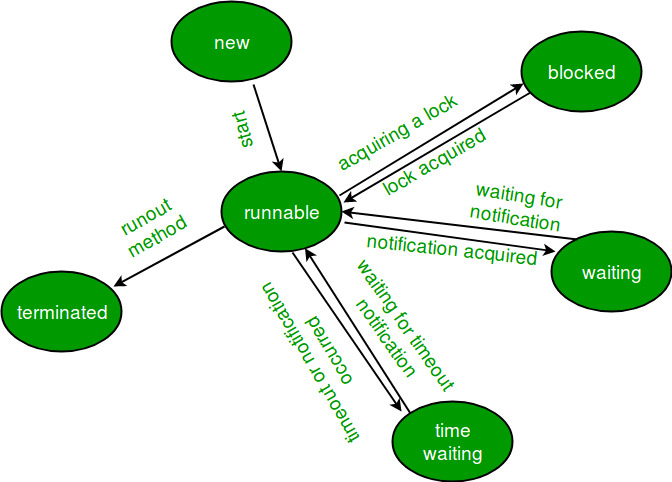
Mức độ ưu tiên của thread có giá trị từ 1 – 10; Java có quy định một số hằng số ưu tiên như MIN\_PRIORITY(1), NORM\_PRIORITY(5), MAX\_PRIORITY(10)

Độ ưu tiên thread phụ thuộc vào thread tạo ra nó . mặc định của thread là NORM\_PRIORITY(5) .

Có thể tùy biến độ ưu tiên của Thread bằng phương thuc setPriority(int priority)

### Vòng đời của 1 thread, các trạng thái

* Vòng đời :



* Các trạng thái của Thread

New : khi thread đã được tạo ra nó ở trạng thái . Thread vẫn chưa bắt đầu chạy ở trạng thái

Runnable : thread đang được chạy hoặc sẵn sàng được chạy bất cứ lúc nào. Tuy nhiên nó không chạy ngay sau khi start mà còn phụ thuộc vào hoạt động điều phối của Thread scheduler.

Blocked / Waiting : thread được chuyển vào trạng thái dừng lại tạm thời và sẽ hoạt động trở lại theo thời gian đã được thiết lập. Sau thời gian chờ sẽ chuyển sang runable

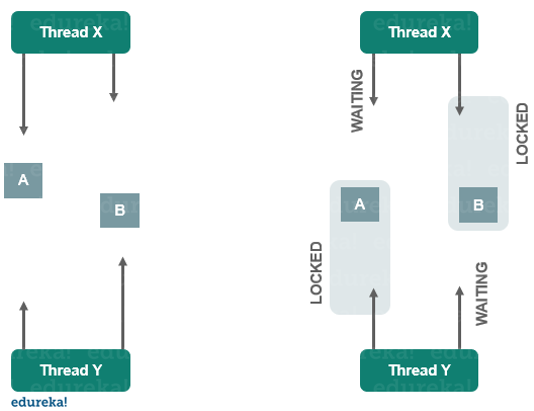
Blocked : khi một thread cố gắng truy cập vào tài nguyên mà một số thread khác đang khóa và sử dụng. Khi phần tài nguyên được mở khóa Thread scheduler sẽ chọn một trong các Thread bị blocked và chuyển nó sang trạng thái runable.

Waiting : khi một thread đang chờ thông báo từ một thread khác. Khi gọi wait,join

Timed Waiting : thread rơi vào trạng thái chờ đợi cho đến khi hết thời gian chờ . khi gọi sleep, wait, join

Terminated : Thread rơi vào trạng thái ngưng hoạt động sau khi đã hoàn thành phương thức run() hoặc gặp một số exception chưa được xử lý.

### Deadlock : khóa chết



Xảy ra khi 2 thread chờ đợi lẫn nhau ( thread này chờ thread kia thực hiện xong mới mới tiếp tục thực hiện công việc của mình).Do vậy các thread sẽ luôn ở trạng thái waiting mãi mãi.

VD :

String data1 = "thread1";

String data2 = "thread2";

Thread thread = **new** Thread() {

**public** **void** run() {

// khoa data1

**synchronized** (data1) {

System.***err***.println(**this**.getName() + " looked " + data1);

**try** {

**this**.*sleep*(100);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***err***.println("Waiting" + data2);

**synchronized** (data2) {

System.***err***.println(**this**.getName() + " looked " + data2);

}

}

};

};

Thread thread2 = **new** Thread() {

**public** **void** run() {

// khoa data2

**synchronized** (data2) {

System.***err***.println(**this**.getName() + ": looked " + data2);

**try** {

**this**.*sleep*(100);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***err***.println("Waiting" + data1);

**synchronized** (data1) {

System.***err***.println(**this**.getName() + ": looked " + data1);

}

}

};

};

thread.setName("Thread A ");

thread2.setName("Thread B ");

thread.start();

thread2.start();

Thread A khóa tài nguyên data1 và chờ data2 được mở khóa trong khi đó

Thread B khóa tài nguyên data2 và chờ data1 được mở khóa ->2 thread mãi chờ nhau

### Timer, Time Task, Schedule

Timer : là class cung cấp các phương thức được sử dụng để lên lịch cho 1 công việc sau một khoảng thời gian.Mỗi công việc được lên lich để chạy 1 lần hoặc nhiều lần lặp lại.

TimerTask : là abstract class định nghĩa công việc có thể được lên lịch trình để chạy 1 hay nhiều lần lặp lại.Để khởi tạo 1 TimerTask cần phải extends TimerTask và định nghĩa phương thức run().

Schedule : là phương thức được cung cấp bởi timer có nhiệm vụ lên lịch trình chạy cho 1 công việc

Timer và TimerTask là hai class trong package java.util được sử dụng để lên lịch cho một công việc và chạy nó trong background với 1 thread.

[TimerTask](http://docs.oracle.com/javase/8/docs/api/java/util/TimerTask.html) là công việc được thực thi còn [Timer](http://docs.oracle.com/javase/8/docs/api/java/util/Timer.html) là lịch trình thực thi.

VD:

Tạo class kế thừa TimerTask

**package** javacoreAdvance;

**import** java.util.Date;

**import** java.util.TimerTask;

**public** **class** MyTask **extends** TimerTask{

@Override

**public** **void** run() {

System.***out***.println("Do something" + **new** Date());

}

}

Tạo chương trình lên lịch

**package** javacoreAdvance;

**import** java.util.TimerTask;

**public** **class** Timer {

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

java.util.Timer timer = **new** java.util.Timer();

TimerTask task = **new** MyTask();

timer.schedule(task, 2000,5000); // (congviec,tgbatdau,tglenlich)

}

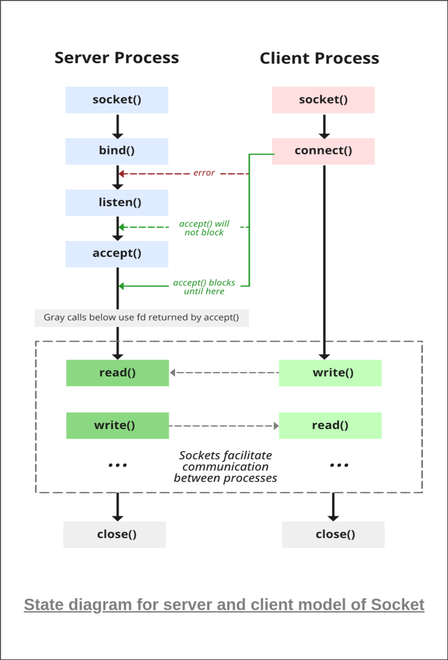
}

## Lập trình mạng

### Socket

là điểm cuối của giao tiếp liên kết 2 chiều giữa hai chương trình chạy trên Internet . Điểm cuối của liên kết này là socket ( IP Address + port).

Mô hình giao tiếp client-server



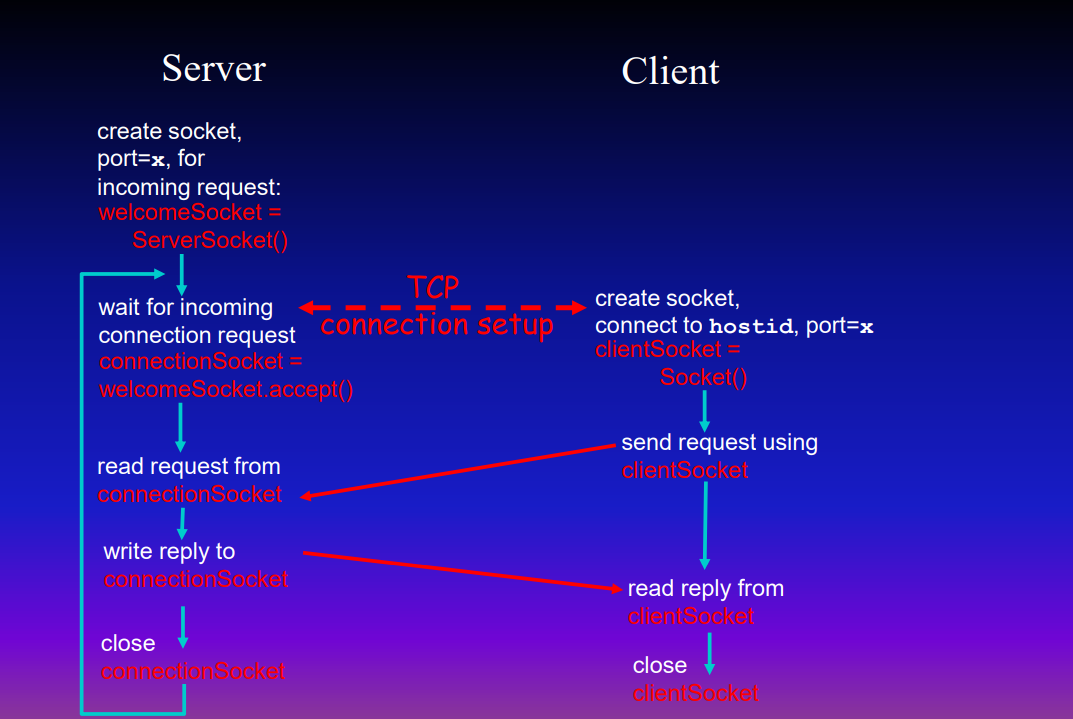
### TCP :

(Transmission Control Protocol) : là giao thức tiêu truyển tải dữ liệu 2 chiều đáng tin cậy giữa các thiết bị qua internet ( các byte được truyển tuần tự và không có lỗi hoặc thiếu sót nào).Nó là giao thức hướng kết nối . Kết hợp với IP tạo thành bộ giao thức TCP/IP .

TCP có vai trò quan trọng kiểm tra và đảm bảo sự toàn vẹn của gói tin -> khi gói tin lỗi thì sẽ gửi yêu cầu gửi lại 1 gói tin khác

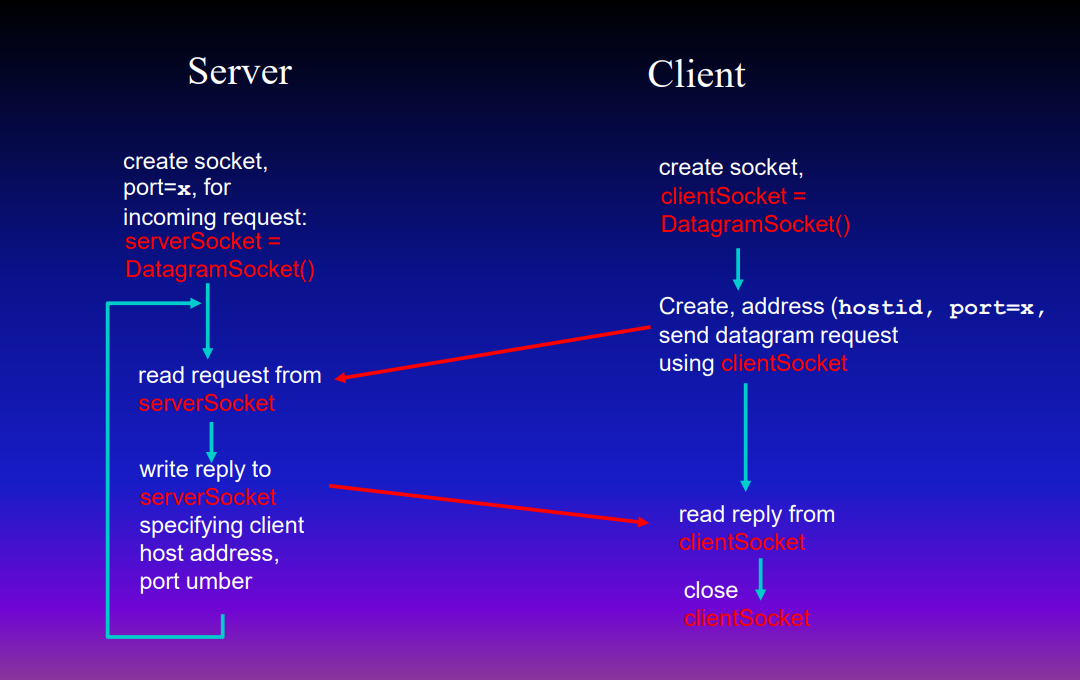
IP : cho phép gói tin được gửi dúng đích bằng cách bổ sung thông tin đường dẫn vào các gói tin

* Dùng khi cần truyền tải dữ liệu có tính toàn vẹn và không yêu cầu tốc độ : truyền file, tín hiệu điều khiển.



### UDP :

(User Datagram Protocol) : là giao thức tiêu truyển tải dữ liệu 2 chiều không đáng tin cậy giữa các thiết bị qua internet . Nó là giao thức không hướng kết nôi. UDP sử dụng để gửi các gói tin ngắn (datagram) -> tăng tốc độ truyền tải. Tuy nhiên UDP không cung cấp kiểm lỗi -> không đảm bảo toàn vẹn dữ liệu.

Cơ chế : 

* Dùng khi cần truyền tải dữ liệu thời gian thực , không cần độ chính xác dữ liệu : truyền âm thanh, hình ảnh

### TCP vs UDP :

|  |  |
| --- | --- |
| TCP | UDP |
| Đều được sử dung để truyền tải dử liệu qua Internet.  Xấy dựng trên nền IP | |
| Độ tin cậy cao | Độ tin cậy thấp |
| Gửi dữ liệu dạng luồng byte | Gửi đi Datagram |
| Hướng kết nối : phải thiết lập kết nối (connectionSocket) để liên kết client-server | Hướng không kết nối :không cần thiết lập kết nối, client phải cung cấp đúng IP và port của bên server |
| Không cho phép mất gói tin | Cho phép mất gói tin |
| Có sắp xếp thứ tự các gói tin | Không sắp xếp thứ tự các gói tin |
| Tốc độ chậm hơn | Tốc độ nhanh hơn |

## MultiThread .

### Khái niệm

MultiThread là một tiến trình ( process ) thực hiện song song đồng thời nhiều luồng (thread) , mỗi luồng sẽ làm một nhiệm vụ riêng nhằm tăng tốc độ xử lý của chương trình.

Các quá trình đang chạy dường như là đồng thời nhưng thực chất tại 1 thời điểm chỉ có thể chạy được 1 thread duy nhất. Việc quyết định chạy thread nào được quản lý bởi Thread scheduler. Thread scheduler chủ yếu sử dụng việc lập kệ hoạch ưu tiên hoặc phân chia thời gian để lên lịch các luồng.

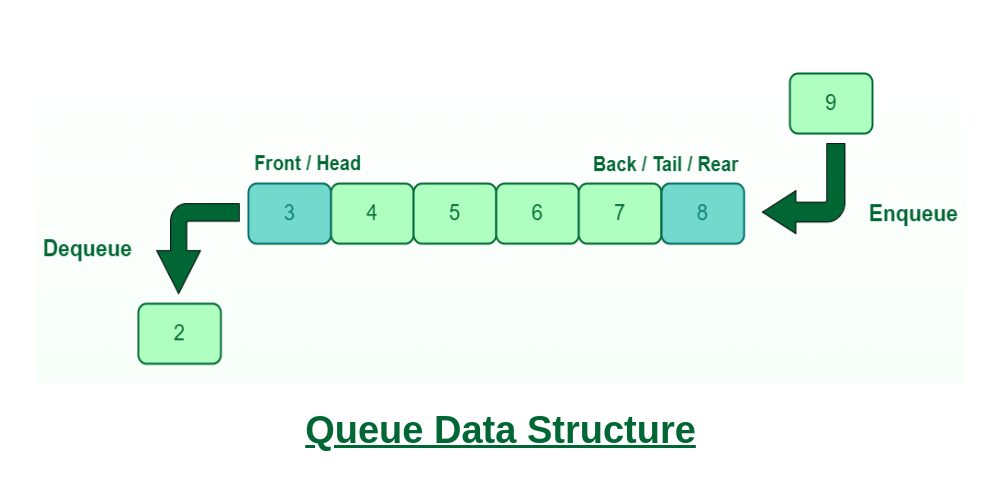
Gọi Start() vs Run() :

Start : yêu cầu JVM cấp phát tài nguyên và tạo ra luông mới khởi chạy phương thức run() trên luông này và chỉ có thể gọi 1 lần. -> đa luồng.

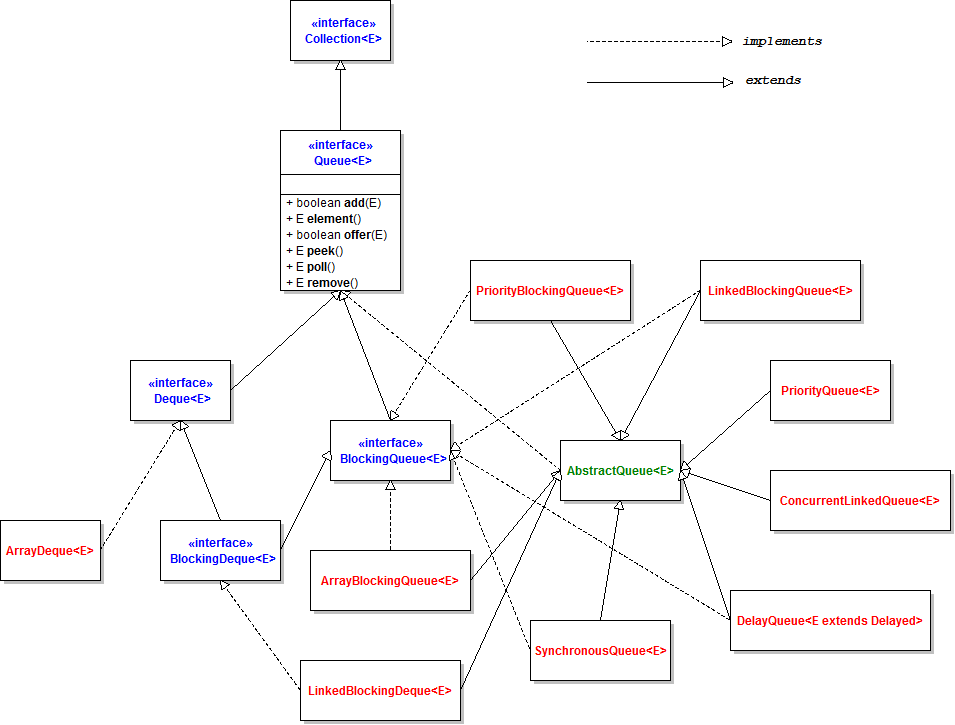
Run () : chỉ là gọi 1 phương thức thông thường và nó sẽ join vào luồng gọi phương thức đó và có thể gọi nhiều lần ->vẫn là 1 luồng .

### Tìm hiểu Queue là gì? các loại? cách làm việc với queue

Queue là một cấu trúc lưu trữ dữ liệu hoạt động theo nguyên tắc FIFO (phần tử nào vào trước sẽ ra trước).



Một số loại queue : queue trong java à interface được kế thừa từ interface Collection.



Bên dưới có một số sub-interface :

Deque : là abstract queue có 2 đầu và cho phép thêm và xóa phần tử tại 2 đầu.

BlockingQueue: là abstract queue nhưng chờ cho đến khi queue không empty khi lấy các phần tử và chờ cho đến khi queue không đầy khi thêm phần tử.

BlockingDeque: là sub-interface của BlockingQueue. Tương tự như blockingqueue nhưng nó có thể thêm vào xóa phần tử tại 2 đầu.

Việc thực thi queue được chia làm 2 nhóm :

General-purpose queue implementation :

Linked-list: implements List and Deque . sử dụng khi muốn thêm , xóa phần tử ở 2 đầu nhanh chóng. Có thể tìm theo index.

PriorityQueue: extends AbstractQueue class, các phần tử trong queue được sắp xếp theo thứ tự , sử dụng khi cần thêm phần tử ở đuôi, xóa phần tử ở đầu nhanh nhanh , kiểm tra tồn tại.

ArrayDeque: implemets Deque interface. Sử dụng khi muốn thêm , xóa phần tử ở 2 đầu nhanh chóng. Đơn giản hơn LinkedList.

Concurent queue implementation :

ArrayBlockingQueue: implements BlockingQueue và extends AbstractQueue được triển khai trên mảng.Sử dụng khi muốn giới hạn số lượng phần tử trong queue.

PriorityBlockingQueue: implements BlockingQueue và extends AbstractQueue được triển khai trên mảng.Sử dụng khi muốn giới hạn số lượng phần tử trong queue.

Làm việc với queue   
enqueue : thêm 1 phần tử vào cuối queue

Dequeue : loại bỏ phần tử đầu khỏi queue.

Peek : lấy phần tử ở đầu queue nhưng không loại bỏ phần tử đó.

FRONT : kiểm tra phần tử đầu của queue

REAR : kiểm tra phần tử cuối của queue.

* Basic queue , blockqueue , Double Ended Queue

### Thread Pool: Khái niệm, cách sử dụng, ưu nhược điểm

Khái niệm : Thread pool là nơi chứa các thread đang chờ để thực hiện các công việc giúp thread tái sử dụng nhiều lần.

Cách sử dụng :

Java cung cấp Executor framework để làm việc với ThreadPool ,trong đó Interface Executor , sub-interface ExecutorService và class ThreadPoolExecutor giúp làm việc trở lên dễ dàng.

Cơ chế hoạt động



Runable queue : là blockingqueue chứa các task được chèn vào chờ thực hiện.

Threadpool Executor : các thread chờ để kéo task từ trong blockingqueue ra để thực hiện.

Có vài cách khác nhau để giao task đến đến ThreadpoolService

Execute(runable) : thực thi chúng bất đồng bộ, -> không thu được kết quả của việc thực hiện

Submit(runable) : tương tự như Execute nhưng trả về đối tượng Future ( có thể kiểm tra runnable đã hoàn thành chưa)

Submit(callable) : tương tự như Submit(runnable) nhưng Callable với call() trả về 1 đối tượng . Kết quả thu được thông qua Future.get()

InvokeAny(Collection<?> extends Callable<T>) : trả về kết quả của 1 callable bất kì (chỉ cần 1 trong số Callable hoàn thành sẽ nhận được kết quả). 1 callable trả về kq hoặc ném ra ngoại lệ thì các Callable khác sẽ bị hủy.

InvokeAll(Collection<?> extends Callable<T>) : trả về Collection<Future<T>> từ việc thực thi các Callable. 1 Callable hoàn thành do ngoại lệ -> nó có thể không thành công trong thực hiện task và không có cách nào kiểm tra sự khác biệt trên đối tượng Futute.

Kết thúc ThreadPoolExecutor

Shutdown() : từ chối nhận task và sẽ kết thúc sau khi hoàn thành các task trong queue

ShutdownNow() : cố gắng chặn các task và loại bỏ các task trong queue.-> ko chắc sẽ tắt ngay lập tức.

awaitTemination() : chặn các luồng gọi tới threadPool cho đến khi nó được tắt hoàn toàn.

Ưu điểm : tái sử dụng thread -> không tốn thời gian và tiết kiệm tài nguyên để tạo ra thread và kết thúc . Giới hạn số lượng thread sẽ được tạo ra -> tránh trường hợp hệ thống hết bộ nhớ.

Nhược điểm :

Deadlock: các luồng đang thực thi chờ kết quả của thread bị block đang chờ trong queue do không có sẵn luồng để thực hiện.

Thread leakage: thread được lấy ra từ pool để xử lý công việc nhưng lại không trả về pool khi hoàn thành công việc . Thread ném ra exception nhưng pool không bắt exception đó -> thread sẽ kết thúc ->pool sẽ mất 1 thread, dần dần pool sẽ bị giỗng.

Resource thrashing: có quá nhiều thread trong pool hơn số công việc cần xử lý -> các thread rảnh, bộ nhớ sử dụng lớn.

### Thế nào là đồng bộ và bất đồng bộ, làm thế nào để đồng bộ khi nhiều thread cùng sử dụng 1 tài nguyên? Các vấn đề cần xử lý khi chương trình có nhiều thread?

Đồng bộ : là quá trình kiểm soát việc truy cập tài nguyên của các thread nhằm tránh gây ra kết quả không mong muốn. Chỉ 1 thread có thể truy cập tài nguyên tại 1 thời điểm.

Bất đồng bộ : ngược lại với đồng bộ , nó sẽ không kiểm soát việc các thread truy cập vào tài nguyên cùng lúc.

Monitor là công cụ hỗ trợ việc đồng bộ hóa Thread. Tại 1 thời điểm chỉ có 1 thread được vào Monitor. Để đưa thread vào Monitor ta sử dụng từ khóa synchronized.

Đồng bộ hóa thread có 2 loại :

Mutual exclusive : là cơ chế tạm khóa block để thread khác thực hiện xong thì sẽ được mở cho thread kế tiếp.

Method synchronized : đồng bộ hóa cả phương thức.

synchronized public void method(); -> khi này chỉ có 1 thread có thể truy xuất vào method(), các thread khác sẽ phải chờ.

Block synchronizded : đồng bộ hóa một đoạn mã

public void method(){

synchronized (reference ){}

} -> khi này chỉ có 1 thread có thể truy xuất vào block synchronized , các thread khác vẫn có thể chạy các đoạn mã ở trên và chỉ chờ khi đên khối synchronized.

Static synchronization :

synchronized public static void method(){} -> khi này việc đồng bộ sẽ áp dụng trên class chứ không phải đối tượng tạo ra từ class đó ( khi class được đồng bộ không phải là thread hoặc runable.)

Inter-thread :

Polling : tiến trình sẽ kiểm tra điều kiện liên tục cho đến khi đúng thì sẽ cho thực thi. -> tốn tài nguyên CPU.

Wait() : thread sẽ bỏ locked và chuyển sang chế độ nghỉ cho đến khi thread khác vào cùng monitor và gọi notify().

Notify() : đánh thức 1 thread bị gọi wait() trong cùng object nhưng nó không bỏ locked

notifyAll(): đánh thức các thread bị gọi wait() trong cùng object.

## Logger

Mục đích : ghi lại nhưng hoạt động bất thường, lỗi của chương trình xảy ra lúc runtime ra file log.

Vai trò : giúp phát hiện lỗi nhanh chóng, dự đoán lỗi có thể xảy ra, kiểm tra được hiệu năng của hệ thống, dễ dàng bảo trì, tiết kiệm thời gian chi phí.

Logger gồm 3 thành phần : Logger ( ghi log ), logging handler or appender( nơi ghi log : console, file, database… ), logging format or layout(định dạng log được ghi (java có SimpleFormatter và XMLFormatter ))

Cấp độ lỗi được xếp giảm dầm, mặt định là ở : Error. Khi đặt cấp độ , cấp độ đó và cao hơn sẽ được ghi lại.

|  |  |
| --- | --- |
| Cấp độ | Sử dụng cho |
| Fatal | Lỗi rất nghiêm trọng có thể khiến chương trình kết thúc |
| Error | Lỗi nghiêm trọng nhưng vẫn cho phép chương trình chạy |
| Warn | Vấn đề, khả năng gây hại cho chương trình |
| Info | Thông tin làm nổi bật tiến trình của chương trình |
| Debug | Sự kiện, thông tin hữu ích cho việc xử lý lỗi |
| Trace | Sự kiện, thông tin chi tiết hơn Debug |
| All | Tất cả các sự kiện |