<https://www.youtube.com/watch?v=1vor_Kd0I-4&index=10&list=PLJbBHp6iPUiGAiqJn0oYboX5WDWyWb4bO>

# 1. Volatile:

<http://huongdanjava.com/noi-ve-bien-volatile-trong-java.html>

# 2. Deadlock:

## Class TaiKhoan:

**package** demo.com;

**public** **class** TaiKhoan {

**int** tongTien = 10000;

**public** **void** guiTien(**int** soTienGui){

tongTien += soTienGui;

}

**public** **void** rutTien(**int** soTienRut){

tongTien -= soTienRut;

}

**public** **int** getTongTien() {

**return** tongTien;

}

**public** **void** setTongTien(**int** tongTien) {

**this**.tongTien = tongTien;

}

**public** **void** giaoDich(TaiKhoan taiKhoan1, TaiKhoan taiKhoan2, **int** soTien){

taiKhoan1.guiTien(soTien);

taiKhoan2.rutTien(soTien);

}

}

## Class Runner:

**package** demo.com;

**import** java.util.Random;

**public** **class** Runner {

TaiKhoan taiKhoan1 = **new** TaiKhoan();

TaiKhoan taiKhoan2 = **new** TaiKhoan();

**public** **void** tienTrinhMot() {

Random rd = **new** Random();

**for** (**int** i = 0; i < 10000; i++) {

taiKhoan1.giaoDich(taiKhoan1, taiKhoan2, rd.nextInt(100));

}

}

**public** **void** tienTrinhHai() {

Random rd = **new** Random();

**for** (**int** i = 0; i < 10000; i++) {

taiKhoan1.giaoDich(taiKhoan2, taiKhoan1, rd.nextInt(100));

}

}

**public** **void** ketQua() {

System.***out***.println("So du cua tai khoan 1: " + taiKhoan1.getTongTien());

System.***out***.println("So du cua tai khoan 2: " + taiKhoan2.getTongTien());

System.***out***.println("Tong tien cua 2 tai khoan: " + (taiKhoan1.getTongTien() + taiKhoan2.getTongTien()));

}

}

## Class testThread:

**package** demo.com;

**public** **class** testThread {

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

Runner runner = **new** Runner();

Thread thread1 = **new** Thread(**new** Runnable() {

@Override

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

runner.tienTrinhMot();

}

});

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

@Override

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

runner.tienTrinhHai();;

}

});

thread1.start();

thread2.start();

**try** {

thread1.join();

thread2.join();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

runner.ketQua();

}

}

## Explanation:

Ta thấy sử dụng thread1.join và thread2.join là để cho 2 thread cùng hoạt động. Đáng lý ra tổng tiền phải bằng 10000 không đổi nhưng vì có 2 thread cùng hoạt động nên xảy ra xung đột, dẫn đến kết quả sai.

## Result:



## Sử dụng đồng bộ để khắc phục:

**package** demo.com;

**import** java.util.Random;

**import** java.util.concurrent.locks.\*;

**public** **class** Runner {

TaiKhoan taiKhoan1 = **new** TaiKhoan();

TaiKhoan taiKhoan2 = **new** TaiKhoan();

Lock lock1 = **new** ReentrantLock();

Lock lock2 = **new** ReentrantLock();

**public** **void** tienTrinhMot() {

Random rd = **new** Random();

**for** (**int** i = 0; i < 10000; i++) {

lock1.lock();

lock2.lock();

taiKhoan1.giaoDich(taiKhoan1, taiKhoan2, rd.nextInt(100));

lock2.unlock();

lock1.unlock();

}

}

**public** **void** tienTrinhHai() {

Random rd = **new** Random();

**for** (**int** i = 0; i < 10000; i++) {

lock1.lock();

lock2.lock();

taiKhoan1.giaoDich(taiKhoan2, taiKhoan1, rd.nextInt(100));

lock2.unlock();

lock1.unlock();

}

}

**public** **void** ketQua() {

System.***out***.println("So du cua tai khoan 1: " + taiKhoan1.getTongTien());

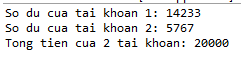
System.***out***.println("So du cua tai khoan 2: " + taiKhoan2.getTongTien());

System.***out***.println("Tong tien cua 2 tai khoan: " + (taiKhoan1.getTongTien() + taiKhoan2.getTongTien()));

}

}

## Result:



Ta thấy kết quả đã hiển thị chính xác.

## Tuy nhiên nếu:

Nếu chúng ta đảo ngược lock1 và lock2 ở tiến trình 2, thì chương trình sẽ treo, đây gọi là deadlock

**package** demo.com;

**import** java.util.Random;

**import** java.util.concurrent.locks.\*;

**public** **class** Runner {

TaiKhoan taiKhoan1 = **new** TaiKhoan();

TaiKhoan taiKhoan2 = **new** TaiKhoan();

Lock lock1 = **new** ReentrantLock();

Lock lock2 = **new** ReentrantLock();

**public** **void** tienTrinhMot() {

Random rd = **new** Random();

**for** (**int** i = 0; i < 10000; i++) {

lock1.lock();

lock2.lock();

taiKhoan1.giaoDich(taiKhoan1, taiKhoan2, rd.nextInt(100));

lock2.unlock();

lock1.unlock();

}

}

**public** **void** tienTrinhHai() {

Random rd = **new** Random();

**for** (**int** i = 0; i < 10000; i++) {

lock2.lock();

lock1.lock();

taiKhoan1.giaoDich(taiKhoan2, taiKhoan1, rd.nextInt(100));

lock1.unlock();

lock2.unlock();

}

}

**public** **void** ketQua() {

System.***out***.println("So du cua tai khoan 1: " + taiKhoan1.getTongTien());

System.***out***.println("So du cua tai khoan 2: " + taiKhoan2.getTongTien());

System.***out***.println("Tong tien cua 2 tai khoan: " + (taiKhoan1.getTongTien() + taiKhoan2.getTongTien()));

}

}

## Để tránh tình trạng:

Để tránh tình trạng deadlock như thế, ta thêm vào cái private void như sau:

**package** demo.com;

**import** java.util.Random;

**import** java.util.concurrent.locks.\*;

**public** **class** Runner {

TaiKhoan taiKhoan1 = **new** TaiKhoan();

TaiKhoan taiKhoan2 = **new** TaiKhoan();

Lock lock1 = **new** ReentrantLock();

Lock lock2 = **new** ReentrantLock();

**private** **void** avoidDeadLock(Lock lock1, Lock lock2) {

**while** (**true**) {

**boolean** getTinhTrangLock1 = **false**;

**boolean** getTinhTrangLock2 = **false**;

**try** {

getTinhTrangLock1 = lock1.tryLock();

getTinhTrangLock2 = lock2.tryLock();

} **finally** {

**if** (getTinhTrangLock1 && getTinhTrangLock2) {

**return**;

}

**if** (getTinhTrangLock1) {

lock1.unlock();

}

**if** (getTinhTrangLock2) {

lock2.unlock();

}

}

**try** {

Thread.*sleep*(1);

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

**public** **void** tienTrinhMot() {

Random rd = **new** Random();

**for** (**int** i = 0; i < 10000; i++) {

avoidDeadLock(lock1, lock2);

taiKhoan1.giaoDich(taiKhoan1, taiKhoan2, rd.nextInt(100));

lock2.unlock();

lock1.unlock();

}

}

**public** **void** tienTrinhHai() {

Random rd = **new** Random();

**for** (**int** i = 0; i < 10000; i++) {

avoidDeadLock(lock1, lock2);

taiKhoan1.giaoDich(taiKhoan2, taiKhoan1, rd.nextInt(100));

lock1.unlock();

lock2.unlock();

}

}

**public** **void** ketQua() {

System.***out***.println("So du cua tai khoan 1: " + taiKhoan1.getTongTien());

System.***out***.println("So du cua tai khoan 2: " + taiKhoan2.getTongTien());

System.***out***.println("Tong tien cua 2 tai khoan: " + (taiKhoan1.getTongTien() + taiKhoan2.getTongTien()));

}

}

## Result:



# 3. Tao va su dung Thread

## Cach tao ke thua tu Thread:

**package** DemoThread1;

**class** RunThread **extends** Thread{

@Override

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

**for** (**int** i = 0; i < 10; i++) {

System.***out***.println("Xin chao cac ban: " + i);

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

**public** **class** MyThread1 {

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

RunThread t1 = **new** RunThread();

t1.start();

}

}

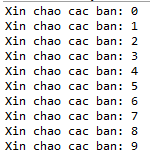
Lưu ý là phải có t1.start() để thread nó chạy, muốn stop thì t1.stop()

## Result:

Sau 1 giay no se in ra 1 dong:



Cho den 10 giay nhu the nay:



## Cach tao implement tu Runnable:

**package** DemoThread2;

**class** RunThread **implements** Runnable{

@Override

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

**for** (**int** i = 0; i < 10; i++) {

System.***out***.println("Xin chao cac ban: " + i);

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

**public** **class** MyThread2 {

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

Thread t1 = **new** Thread(**new** RunThread());

t1.start();

}

}

## Result:

Y chang bên trên

## Cách mà 2 thread chạy song song với nhau:

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

Thread t1 = **new** Thread(**new** RunThread());

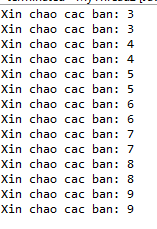
Thread t2 = **new** Thread(**new** RunThread());

t1.start();

t2.start();

}

## Result:



## Cach tạo thead kiểu thứ 3, kiểu trực tiếp:

**package** DemoThread3;

**public** **class** MyThread3 {

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

Thread t1 = **new** Thread(**new** Runnable() {

@Override

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

**for** (**int** i = 0; i < 10; i++) {

System.***out***.println("Xin chao cac ban: " + i);

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

});

t1.start();

}

}

## Result:

Y chang cách 1 và 2

# 4. Đồng bộ cơ bản:

**package** DemoSync;

**import** java.util.Scanner;

**class** MyRun **extends** Thread{

**boolean** stop = **false**;

**public** **void** StopThread(){

stop = **true**;

}

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

**while** (!stop) {

System.***out***.println("Xin chao minh le Che Cong Binh");

**try** {

Thread.*sleep*(1000);

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

}

**public** **class** MyThreadSync {

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

MyRun t1 = **new** MyRun();

t1.start();

Scanner sNhap = **new** Scanner(System.***in***);

sNhap.nextLine();

t1.StopThread();

}

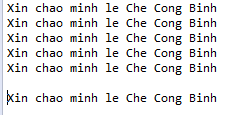
}

## Explanation:

Ở đây vì chưa chạy cái void StopThread nên sẽ in ra mãi.

Đến khi người dùng click hoặc enter (ta sử dụng Scanner để làm điều này), thì thread sẽ dừng.

## Result:



Địa chỉ: https://www.youtube.com/watch?v=GJE4JYfM5js&list=PLJbBHp6iPUiGAiqJn0oYboX5WDWyWb4bO&index=4

# 5. Đồng bộ Synchronized:

## Code bình thường:

**package** demo.com;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Random;

**public** **class** MyThreadBlock {

List<Integer> ds1 = **new** ArrayList<>();

List<Integer> ds2 = **new** ArrayList<>();

Random rd = **new** Random();

**public** **void** setDs1(){

**try** {

Thread.*sleep*(1);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

ds1.add(rd.nextInt(100));

}

**public** **void** setDs2(){

**try** {

Thread.*sleep*(1);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

ds2.add(rd.nextInt(100));

}

**public** **void** xuLy(){

**for** (**int** i = 0; i < 1000; i++) {

setDs1();

setDs2();

}

}

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

MyThreadBlock my = **new** MyThreadBlock();

System.***out***.println("Dang xu ly...");

my.xuLy();

System.***out***.println("Danh sach 1: " + my.ds1.size());

System.***out***.println("Danh sach 2: " + my.ds2.size());

}

}

## Result:



Vì không có thread nên ta thấy kết quả bình thường, k có chuyện gì xảy ra.

## Khi có 2 thread chạy song song:

**package** demo.com;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Random;

**public** **class** MyThreadBlock {

List<Integer> ds1 = **new** ArrayList<>();

List<Integer> ds2 = **new** ArrayList<>();

Random rd = **new** Random();

**public** **void** setDs1(){

**try** {

Thread.*sleep*(1);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

ds1.add(rd.nextInt(100));

}

**public** **void** setDs2(){

**try** {

Thread.*sleep*(1);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

ds2.add(rd.nextInt(100));

}

**public** **void** xuLy(){

**for** (**int** i = 0; i < 1000; i++) {

setDs1();

setDs2();

}

}

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

MyThreadBlock my = **new** MyThreadBlock();

System.***out***.println("Dang xu ly...");

Thread t1 = **new** Thread(**new** Runnable() {

@Override

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

my.xuLy();

}

});

Thread t2 = **new** Thread(**new** Runnable() {

@Override

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

my.xuLy();

}

});

t1.start();

t2.start();

**try** {

t1.join();

t2.join();

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("Danh sach 1: " + my.ds1.size());

System.***out***.println("Danh sach 2: " + my.ds2.size());

}

}

## Result:



Đáng lý ra kết quả phải là 2000 2000 mới đúng, nhưng vì 2 thread chạy song song nên có lúc sẽ add cùng 1 giá trị, vì thế bị mất giá trị trong danh sách.

## Sử dụng synchronized để đồng bộ:

**public** **synchronized** **void** setDs1(){

**public** **synchronized** **void** setDs2(){

## Result:



## Sử dụng synchronized với lock:

**package** demo.com;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Random;

**public** **class** MyThreadBlock {

List<Integer> ds1 = **new** ArrayList<>();

List<Integer> ds2 = **new** ArrayList<>();

Random rd = **new** Random();

Object lock1 = **new** Object();

Object lock2 = **new** Object();

**public** **void** setDs1(){

**synchronized** (lock2) {

**try** {

Thread.*sleep*(1);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

ds1.add(rd.nextInt(100));

}

**public** **synchronized** **void** setDs2(){

**synchronized** (lock1) {

**try** {

Thread.*sleep*(1);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

ds2.add(rd.nextInt(100));

}

**public** **void** xuLy(){

**for** (**int** i = 0; i < 1000; i++) {

setDs1();

setDs2();

}

}

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

MyThreadBlock my = **new** MyThreadBlock();

System.***out***.println("Dang xu ly...");

Thread t1 = **new** Thread(**new** Runnable() {

@Override

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

my.xuLy();

}

});

Thread t2 = **new** Thread(**new** Runnable() {

@Override

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

my.xuLy();

}

});

t1.start();

t2.start();

**try** {

t1.join();

t2.join();

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("Danh sach 1: " + my.ds1.size());

System.***out***.println("Danh sach 2: " + my.ds2.size());

}

}

## Result:

Kết quả ra 2000 2000 như bên trên

# 6. Callable và Future:

Sau khi luồng xử lý xong thì trả lại 1 giá trị mà chúng ta muốn, ta sử dụng Callable và Future.

## Code:

**package** demo.com;

**import** java.util.\*;

**import** java.util.concurrent.\*;

**public** **class** TestCallableFuture {

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

ExecutorService pool = Executors.*newCachedThreadPool*();

Future<Integer> getTong = pool.submit(**new** Callable<Integer>() {

@Override

**public** Integer call() **throws** Exception {

**int** tong = 0;

Random rd1 = **new** Random();

Random rd2 = **new** Random();

**int** a = rd1.nextInt(400);

**int** b = rd2.nextInt(400);

System.***out***.println("Gia tri random 1: "+a);

System.***out***.println("Gia tri random 2: "+b);

tong = a + b;

**return** tong;

}

});

pool.shutdown();

**try** {

System.***out***.println("Tong la: "+getTong.get());

} **catch** (InterruptedException e) {

e.printStackTrace();

} **catch** (ExecutionException e) {

e.printStackTrace();

}

}

}

## Result:



# 7. CountDownLock:

Countdown lock dùng để quy định số tuyến trình đầu vào

## Code:

**package** demo.com;

**import** java.util.concurrent.\*;

**class** Runner **implements** Runnable{

CountDownLatch l;

**public** Runner(CountDownLatch l) {

**this**.l = l;

}

@Override

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

System.***out***.println("Dang xu ly...");

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***out***.println("Da xu ly xong");

l.countDown();

}

}

**public** **class** MyCountDown {

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

CountDownLatch latch = **new** CountDownLatch(3); // Cho phep truyen vao toi da 3 tuyen trinh

ExecutorService pool = Executors.*newFixedThreadPool*(3); //Xu ly cung 1 luc bao nhieu tuyen trinh

**for** (**int** i = 0; i < 6; i++) {

pool.submit(**new** Runner(latch));

}

**try** {

latch.await();

} **catch** (InterruptedException e) {

e.printStackTrace();

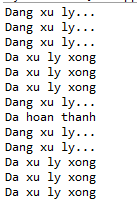
}

System.***out***.println("Da hoan thanh");

}

}

## Result:



T thấy xử lý xong 3 tiến trình thì mới xuất ra câu: Da hoan thanh

# 8. Thread Pool:

## code:

**package** demo.com;

**import** java.util.concurrent.\*;

**class** RunPool **implements** Runnable{

**int** id;

@Override

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

System.***out***.println("Dang xu ly tien trinh "+id);

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***out***.println("Da xu ly tien trinh "+id);

}

**public** RunPool(**int** id) {

**this**.id = id;

}

}

**public** **class** MyThreadPool {

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

ArrayBlockingQueue<Runnable> hangDoi = **new** ArrayBlockingQueue<>(100);

ThreadPoolExecutor pool = **new** ThreadPoolExecutor(5, 5, 1, TimeUnit.***SECONDS***, hangDoi);

**for** (**int** i = 0; i < 20; i++) {

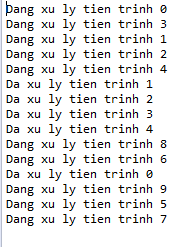
pool.execute(**new** RunPool(i));

}

}

}

## Result:



Ta thấy là chương trình sẽ xử lý cùng lúc 5 tiến trình.

## Cách 2 với Thread Pool:

**public** **class** MyThreadPool {

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

ExecutorService pool = Executors.*newFixedThreadPool*(5);

**for** (**int** i = 0; i < 20; i++) {

pool.submit(**new** RunPool(i));

}

**try** {

pool.awaitTermination(1, TimeUnit.***DAYS***);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

pool.shutdown();

}

}

## Result:

Như trên

# 9. ProducerCustomer:

BlockingQueue giống như hàng đợi tạm thời, ta có thể đem giá trị từ luồng này và qua luồng kia để xử lý.

**package** demo.com;

**import** java.util.Random;

**import** java.util.concurrent.\*;

**public** **class** MyThreadClass {

**static** **int** *value*;

**static** BlockingQueue<Integer> *queue* = **new** ArrayBlockingQueue<>(10); //Day la 1 block, day la hang doi, put du lieu vao chung ta mong muon

**public** **static** **void** producer(){

Random rd = **new** Random();

**while** (**true**) {

**try** {

*queue*.put(rd.nextInt(10));

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

**public** **static** **void** customer(){

Random rd = **new** Random();

**while** (**true**) {

**try** {

Thread.*sleep*(1);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**if** (rd.nextInt(10) == 0) {

**try** {

*value* = *queue*.take();

System.***out***.println("Gia tri: " + *value* + "\nChieu dai block: " + *queue*.size());

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

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

Thread t1 = **new** Thread(**new** Runnable() {

@Override

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

*producer*();

}

});

Thread t2 = **new** Thread(**new** Runnable() {

@Override

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

*customer*();

}

});

t1.start();

t2.start();

**try** {

t1.join();

t2.join();

} **catch** (InterruptedException e) {

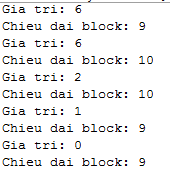
e.printStackTrace();

}

}

}

## Result:



# 10. Wait, Notify

## Class TinNhan:

**package** demo.com;

**public** **class** TinNhan {

**private** String noiDung;

**public** TinNhan() {

}

**public** TinNhan(String noiDung) {

**super**();

**this**.noiDung = noiDung;

}

**public** String getNoiDung() {

**return** noiDung;

}

**public** **void** setNoiDung(String noiDung) {

**this**.noiDung = noiDung;

}

}

## Class NguoiGuiTin:

**package** demo.com;

**public** **class** NguoiGuiTinNhan **implements** Runnable{

TinNhan tinNhan;

**public** NguoiGuiTinNhan() {

}

**public** NguoiGuiTinNhan(TinNhan tinNhan) {

**super**();

**this**.tinNhan = tinNhan;

}

@Override

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

**try** {

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**synchronized** (tinNhan) {

tinNhan.setNoiDung("Xin Chao Cac Ban");

tinNhan.notifyAll();

}

}

}

## Class NguoiNhanTin va ham main:

**package** demo.com;

**public** **class** NguoiNhanTin **implements** Runnable{

TinNhan tinNhan;

**public** NguoiNhanTin() {

}

**public** NguoiNhanTin(TinNhan tinNhan) {

**super**();

**this**.tinNhan = tinNhan;

}

@Override

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

**synchronized** (tinNhan) {

System.***out***.println("Dang lay tin nhan...");

**try** {

tinNhan.wait();

System.***out***.println("Da nhan duoc tin nhan");

System.***out***.println("Noi dung: " + tinNhan.getNoiDung());

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

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

TinNhan tn = **new** TinNhan("Xu ly");

Thread t1 = **new** Thread(**new** NguoiGuiTinNhan(tn));

Thread t2 = **new** Thread(**new** NguoiNhanTin(tn));

t1.start();

t2.start();

}

}

## Result sau 2 giây:



Chú ý là hàm Thread.sleep phải để ở ngoài synchronized

# 11. Join

Để 2 thread tuy chạy cùng lúc, nhưng cái chạy trước cái chạy sau, chứ không phải chạy song song.

## Khong co join:

**package** demo.com;

**public** **class** TestJoin {

**static** **int** *count* = 0;

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

Thread t1 = **new** Thread(**new** Runnable() {

@Override

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

**for** (**int** i = 0; i < 500; i++) {

*count*++;

}

}

});

Thread t2 = **new** Thread(**new** Runnable() {

@Override

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

**for** (**int** i = 0; i < 500; i++) {

*count*++;

}

}

});

t1.start();

t2.start();

System.***out***.println("Da dem duoc: " + *count*);

}

}

## Result:



Ta thấy ra kết quả sai vì t1 t2 chạy song song, khiến count không biết đếm như thế nào nên sai.

## Co join:

**package** demo.com;

**public** **class** TestJoin {

**static** **int** *count* = 0;

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

Thread t1 = **new** Thread(**new** Runnable() {

@Override

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

**for** (**int** i = 0; i < 500; i++) {

*count*++;

}

}

});

Thread t2 = **new** Thread(**new** Runnable() {

@Override

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

**for** (**int** i = 0; i < 500; i++) {

*count*++;

}

}

});

t1.start();

t2.start();

**try** {

t1.join();

t2.join();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***out***.println("Da dem duoc: " + *count*);

}

}

## Result:



Vì có join nên chạy đúng

# 12.Semaphore

Có thể set số truy cập tối đa truy cập vào web, tránh ngẽn.

**package** demo.com;

**import** java.util.concurrent.Semaphore;

**public** **class** TestConnect {

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

Semaphore se = **new** Semaphore(2);

**try** {

se.acquire();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***out***.println("Gia tri cho phep truy cap: " + se.availablePermits());

}

}

## Result:



Vì đã có 1 acquire, mà tối đa cho phép 2 luồng nên kết quả in ra 1, số luồng còn lại.

## Release

**package** demo.com;

**import** java.util.concurrent.Semaphore;

**public** **class** TestConnect {

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

Semaphore se = **new** Semaphore(2);

**try** {

se.acquire();

se.release();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***out***.println("Gia tri cho phep truy cap: " + se.availablePermits());

}

}

## Result:



Vì release đã giải phóng 1 kết nối nên tổng cộng còn 2 luồng truy cập vào.

## Áp dụng Semaphore:

### Class Connect:

**package** demo.com;

**import** java.util.concurrent.Semaphore;

**public** **class** Connect {

**public** **static** Connect *newConnect* = **new** Connect();

**int** dem = 0;

Semaphore se = **new** Semaphore(20);

**public** Connect() {

}

**public** **static** Connect openConnect(){

**return** *newConnect*;

}

**public** **void** DemSoLuongTruyCap(){

**synchronized** (**this**) {

**try** {

se.acquire();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

dem++;

System.***out***.println("So luong truy cap hien tai la: " + dem);

}

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**synchronized**(**this**){

dem--;

se.release();

}

}

}

### Class Test:

**package** demo.com;

**import** java.util.concurrent.ExecutorService;

**import** java.util.concurrent.Executors;

**import** java.util.concurrent.Semaphore;

**import** java.util.concurrent.TimeUnit;

**public** **class** TestConnect {

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

ExecutorService pool = Executors.*newCachedThreadPool*();

**for** (**int** i = 0; i < 300; i++) {

pool.submit(**new** Runnable() {

@Override

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

Connect.*openConnect*().DemSoLuongTruyCap();

}

});

}

pool.shutdown();

**try** {

pool.awaitTermination(1, TimeUnit.***DAYS***);

} **catch** (InterruptedException e) {

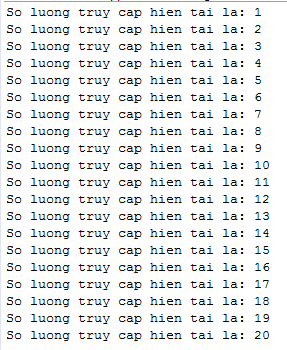
e.printStackTrace();

}

}

}

### Result:



Ta thấy chương trình chỉ cho phép tối đa 20 luồng truy cập.

# 13. Daemon Thread

**Java** chia Thread làm 2 loại một loại thông thường và **Daemon Thread**. Chúng chỉ khác nhau ở cách thức ngừng hoạt động. Trong một chương trình các luồng thông thường và Daemon chạy song song với nhau. Khi tất cả các luồng thông thường kết thúc, mọi luồng Daemon cũng sẽ bị kết thúc theo bất kể nó đang làm việc gì.

***Chú ý:***   
  
Sử dụng ***setDaemon(boolean)*** để sét đặt một luồng là Daemon hoặc không. Chú ý, bạn chỉ có thể gọi hàm ***setDaemon(boolean)*** khi thread chưa được chạy. Điều đó có nghĩa là khi thread đã chạy bạn không thể chuyển luồng từ **non-daemon** sang **daemon** và ngược lại.   
  
Khi một luồng mới được tạo ra, nó được thừa hưởng đặc tính daemon từ luồng cha.  Như vậy khi bạn tạo một luồng trong hàm main của 1 class nó vốn là luồng non-daemon, vì vậy thread tạo ra mặc định cũng là none-daemon. Như vậy nếu bạn tạo một luồng mới trong một luồng Daemon, mặc định nó cũng sẽ là Daemon.

## Class none daemon thread:

**package** demo.com.daemonthread;

**public** **class** NoneDaemonThread **extends** Thread{

@Override

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

//Vong lap 10 lan. Luong nay se ket thuc

**for** (**int** i = 0; i < 10; i++) {

System.***out***.println("Hello from None Daemon Thread: " + i);

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

// Ghi ra thong bao luong nay ket thuc

System.***out***.println("\n==> None Daemon Thread ending\n");

}

}

## Class daemon thread:

**package** demo.com.daemonthread;

**public** **class** DaemonThread **extends** Thread{

@Override

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

**int** i = 0;

//Vong lap vo tan

**while** (**true**) {

System.***out***.println("Hello from Daemon Thread: " + i++);

**try** {

*sleep*(2000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

## Class test:

**package** demo.com.daemonthread;

**public** **class** DaemonTest {

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

System.***out***.println("==> Main Thread running..\n");

DaemonThread daemonThread = **new** DaemonThread();

daemonThread.setDaemon(**true**);

daemonThread.start();

**new** NoneDaemonThread().start();

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

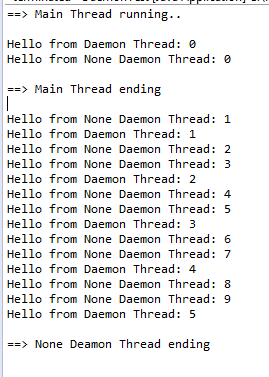
// Ghi ra thong bao nay khi ket thuc

System.***out***.println("\n==> Main Thread ending\n");

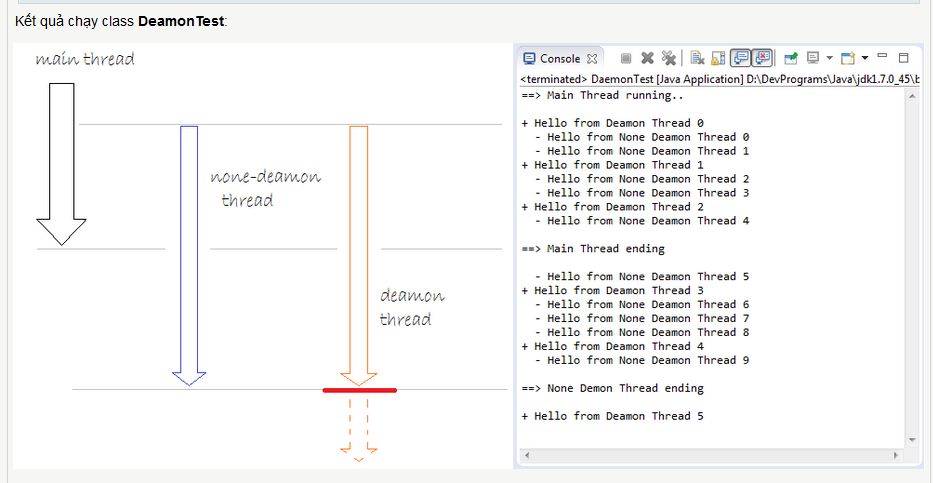
}

}

## Result:



## Explanation:



Hình minh họa trên cho thấy rằng luồng Daemon đã bị dừng lại khi tất cả các luồng thông thường đã dừng. Mặc dù code của nó là chạy vô tận.

### Luồng daemon thường dùng làm gì?

Một trong các luồng Daemon quan trọng của Java đó là luồng gom rác, nghĩa là gom các tài nguyên không còn sử dụng để giải phóng bộ nhớ. Khi tất cả các luồng người dùng không còn hoạt động nữa luồng gom rác cũng bị dừng theo.