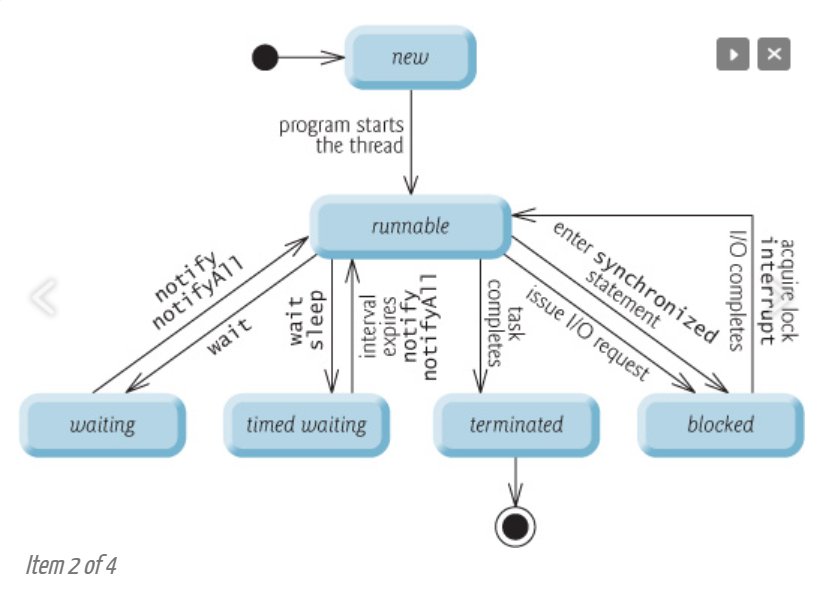
# Thread là gì?

**Thread** (luồng) về cơ bản là một tiến trình con (sub-process). Một đơn vị xử lý nhỏ nhất của máy tính có thể thực hiện một công việc riêng biệt. Trong Java, các luồng được quản lý bởi máy ảo Java (JVM).

# Multi-thread là gì?

**Multi-thread** (đa luồng) là một tiến trình thực hiện nhiều luồng đồng thời. Một ứng dụng Java ngoài luồng chính có thể có các luồng khác thực thi đồng thời làm ứng dụng chạy nhanh và hiệu quả hơn.

# Vòng đời (các trạng thái) của một Thread trong java



Vòng đời của thread trong java được kiểm soát bởi JVM. Java định nghĩa các trạng thái của luồng trong các thuộc tính static của lớp **Thread.State**:

* **NEW**: Đây là trạng thái khi luồng vừa được khởi tạo bằng phương thức khởi tạo của lớp Thread nhưng chưa được start(). Ở trạng thái này, luồng được tạo ra nhưng chưa được cấp phát tài nguyên và cũng chưa chạy. Nếu luồng đang ở trạng thái này mà ta gọi các phương thức ép buộc stop,resume,suspend … sẽ là nguyên nhân sảy ra ngoại lệ IllegalThreadStateException .
* **RUNNABLE**: Sau khi gọi phương thức start() thì luồng test đã được cấp phát tài nguyên và các lịch điều phối CPU cho luồng test cũng bắt đầu có hiệu lực. Ở đây, chúng ta dùng trạng thái là Runnable chứ không phải Running, vì luồng không thực sự luôn chạy mà tùy vào hệ thống mà có sự điều phối CPU khác nhau.
* **WAITING** : Thread chờ không giới hạn cho đến khi một luồng khác đánh thức nó.
* **TIMED\_WAITING** : Thread chờ trong một thời gian nhất định, hoặc là có một luồng khác đánh thức nó.
* **BLOCKED**: Đây là 1 dạng của trạng thái “Not Runnable”, là trạng thái khi Thread vẫn còn sống, nhưng hiện tại không được chọn để chạy. Thread chờ một monitor để unlock một đối tượng mà nó cần.
* **TERMINATED**: Một thread ở trong trạng thái terminated hoặc dead khi phương thức run() của nó bị thoát.

# Đa nhiệm có thể đạt được bằng hai cách:

## Đa nhiệm dựa trên đơn tiến trình (Process) – Đa tiến trình (Multiprocessing).

* Mỗi tiến trình có địa chỉ riêng trong bộ nhớ, tức là mỗi tiến trình phân bổ vùng nhớ riêng biệt.
* Tiến trình là nặng.
* Sự giao tiếp giữa các tiến trình có chi phí cao.
* Chuyển đổi từ tiến trình này sang tiến trình khác đòi hỏi thời gian để đăng ký việc lưu và tải các bản đồ bộ nhớ, các danh sách cập nhật, …

## Đa nhiệm dựa trên luồng (Thread) – Đa luồng (MultiThreading).

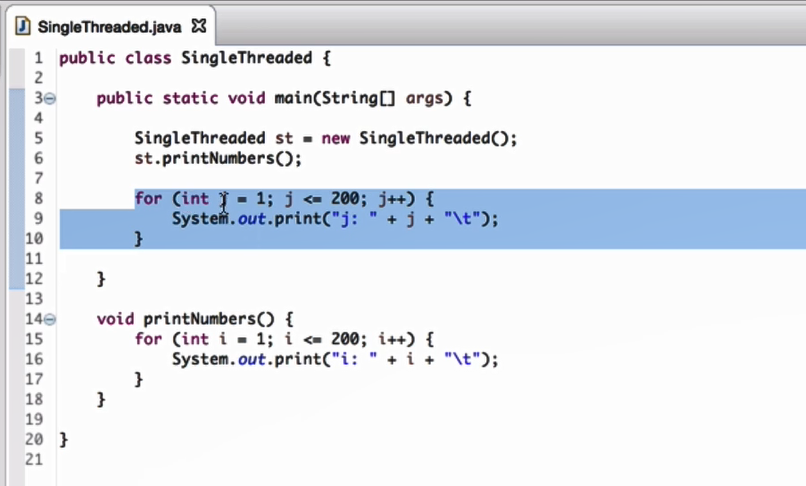
* Các luồng chia sẻ không gian địa chỉ ô nhớ giống nhau.
* Luồng là nhẹ.
* Sự giao tiếp giữa các luồng có chi phí thấp.

# Đa luồng Multithreading

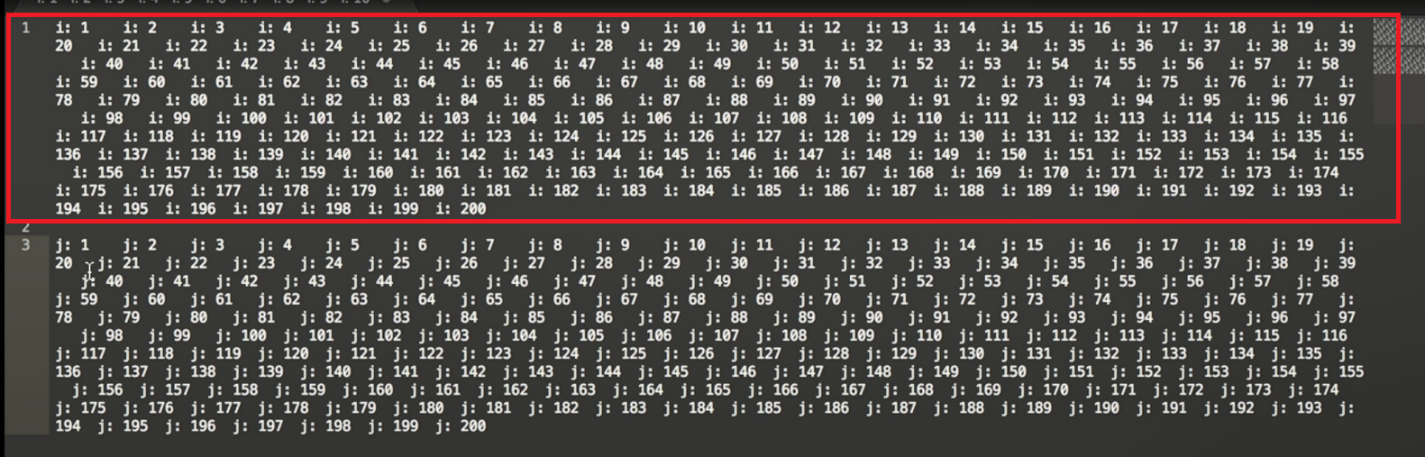
## Single Threaded Example

Luồng chạy theo trình tự, luồng a chạy xong mới đến luồng b chạy

**Example**



Kết quả:

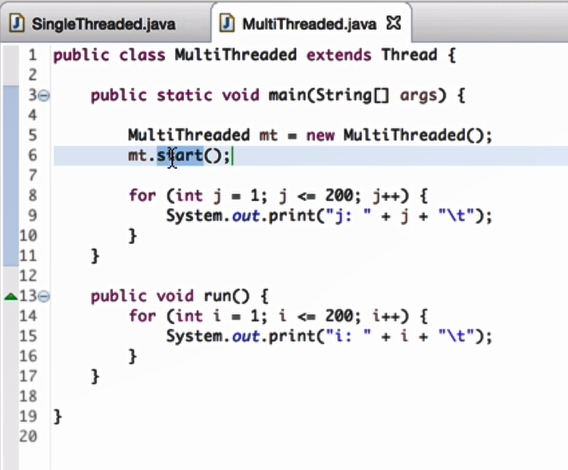


Hàm printnumber i=1 … chạy xong

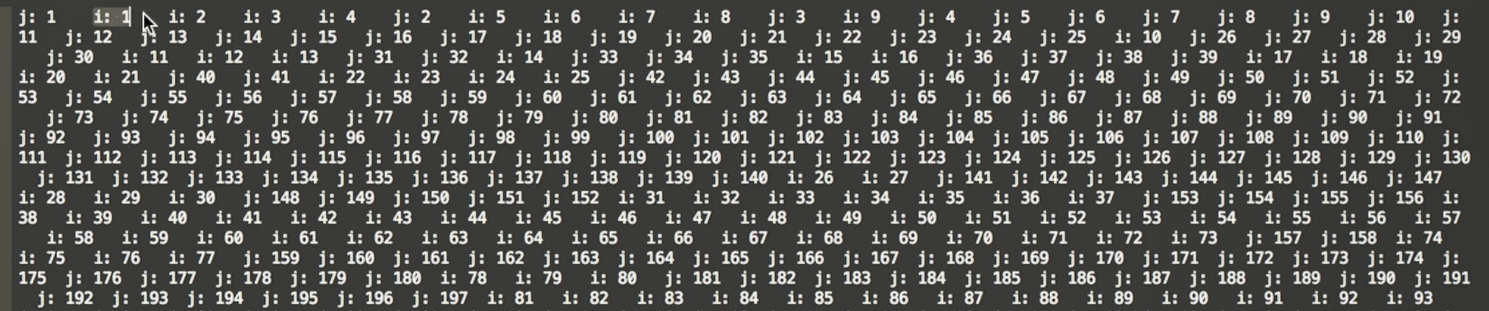
Thì mới đến hàm main() j=1…

## Multi Threading in Action

* Nhiều luồng chạy đồng thời cùng một lúc



Kết quả:



## 3. Sleep Method

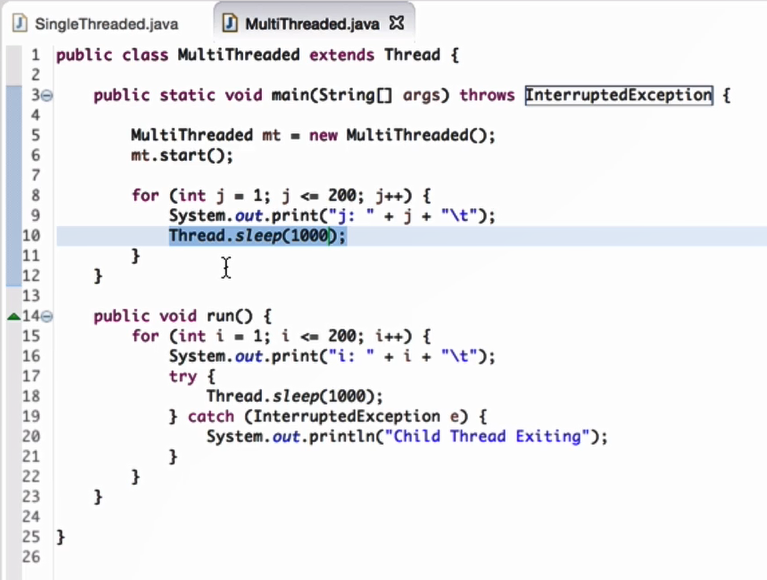
Phương thức sleep() của lớp Thread được sử dụng để tạm ngứng một thread cho một khoảng thời gian nhất định.

Cú pháp của phương thức sleep trong java

Lớp Thread cung cấp hai phương thức để tạm ngưng một thread:

* public static void sleep(long miliseconds)throws InterruptedException
* public static void sleep(long miliseconds, int nanos)throws InterruptedException

Như bạn biết rõ rằng tại một thời điểm chỉ có một thread được thực thi. Nếu bạn tạm ngưng một thread trong một khoảng thời gian nhất định, thì thread scheduler chọn một thread khác để thực thi.



<https://viettuts.vn/java-thread/phuong-thuc-sleep-trong-java>

## Example download file

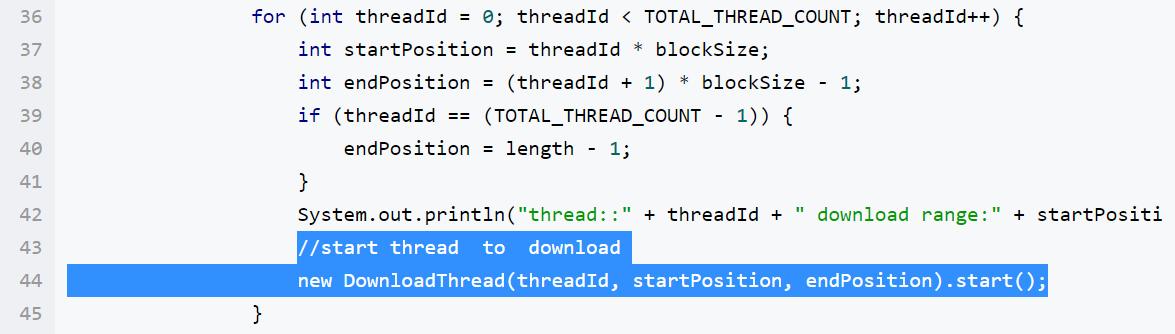
Bài toán: Nhiều người cùng tải file cùng lúc, trong qua trình tải file mất rất nhiều thời gian nên phải sử dụng Multi Threading ở đây.

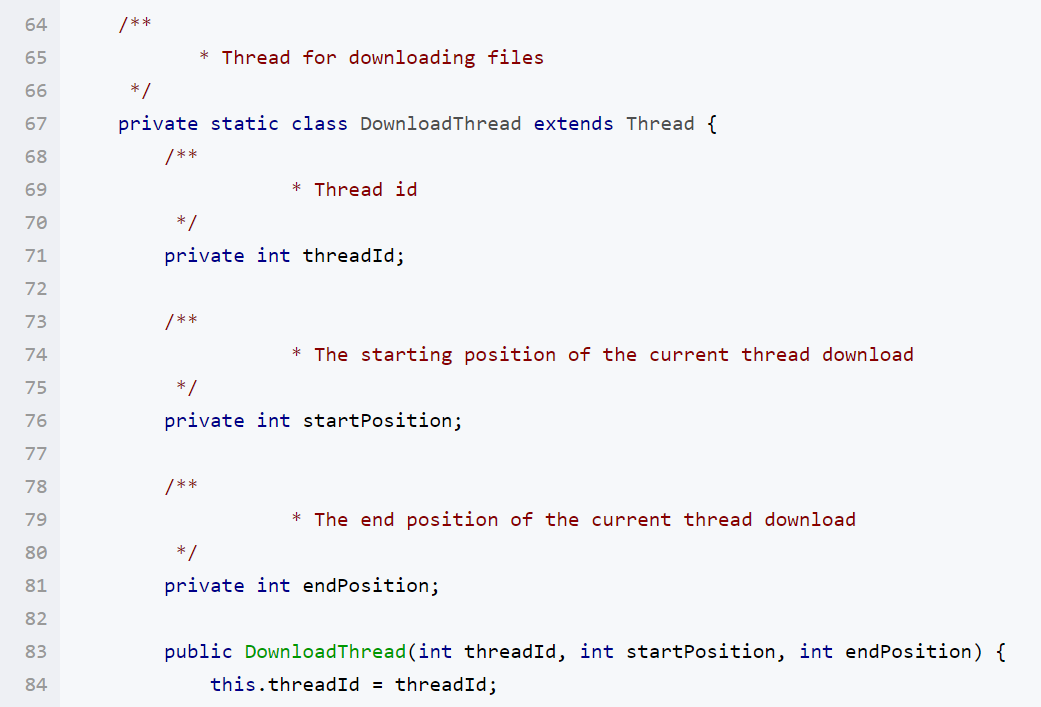
Nếu không dùng multi threading, mà dùng single threading thì mỗi lần tải file, thì chờ người A tải xong mới đến người B tải, thì làm cho người b phải đợi mất rất nhiều thời gian.

Bước 1: HttpUrlConnection để yêu cầu máy chủ lấy độ dài của tệp con.getContentLength (), tên file, thông tin file

Bước 2: Tải file

* Xây dựng một method riêng, gọi đến method DownloadThread





**Ý tưởng chung như sau:**

1 HttpUrlConnection để yêu cầu máy chủ lấy độ dài của tệp con.getContentLength ()

2 Tạo RandomAcessFile trống để nhận và chỉ định độ dài setLength vừa lấy được

3 Mở N luồng để tính độ dài của mỗi luồng sẽ được tải xuống

4 Trước khi tải xuống, hãy kiểm tra xem tệp lưu tiến trình đã tải xuống có tồn tại hay không. Nếu nó tồn tại, hãy lấy tiến trình tải xuống từ tệp

5 Bắt đầu tải xuống tệp

6 Xóa tệp tạm thời Đóng tài nguyên

**new DownloadThread(threadId, startPosition, endPosition).start();**

|  |
| --- |
| package demo;  import java.io.BufferedReader;  import java.io.File;  import java.io.FileInputStream;  import java.io.InputStream;  import java.io.InputStreamReader;  import java.io.RandomAccessFile;  import java.net.HttpURLConnection;  import java.net.URL;  class MultiDownloaFile {      public static final String path = "http://192.168.217.1:8080/androidsimpleserver/HTTP.pdf";      public static final int TOTAL\_THREAD\_COUNT = 3;      public static int runningThreadCount = 0;      public static void main(String[] args) {          try {              long start = System.currentTimeMillis();              URL url = new URL(path);              HttpURLConnection conn = (HttpURLConnection) url.openConnection();              conn.setRequestMethod("GET");              int code = conn.getResponseCode();              if (code == 200) {                  int length = conn.getContentLength();                  System.out.println("file length:" + length);                  //create  a  null  file  to  save  its  length                  RandomAccessFile raf = new RandomAccessFile(getDownloadFileName(path), "rw");                  raf.setLength(length);                  raf.close();                  //parse  every  thread  that  they  need  how  much  room  to  download                  int blockSize = length / TOTAL\_THREAD\_COUNT;                  System.out.println("every block size:" + blockSize);                  runningThreadCount = TOTAL\_THREAD\_COUNT;                  for (int threadId = 0; threadId < TOTAL\_THREAD\_COUNT; threadId++) {                      int startPosition = threadId \* blockSize;                      int endPosition = (threadId + 1) \* blockSize - 1;                      if (threadId == (TOTAL\_THREAD\_COUNT - 1)) {                          endPosition = length - 1;                      }                      System.out.println("thread::" + threadId + " download range:" + startPosition + "~~" + endPosition);                      //start thread  to  download                      new DownloadThread(threadId, startPosition, endPosition).start();                  }              } else {                  System.out.println(" connection  error ");              }          } catch (Exception e) {              e.printStackTrace();          }      } |

|  |
| --- |
| /\*\*       \* Get the file name from the network path       \*       \* @param  path network path       \* @return  file name       \*/      private static String getDownloadFileName(String path) {          return path.substring(path.lastIndexOf("/") + 1);      }      /\*\*             \* Thread for downloading files       \*/      private static class DownloadThread extends Thread {          /\*\*                     \* Thread id           \*/          private int threadId;          /\*\*                     \* The starting position of the current thread download           \*/          private int startPosition;          /\*\*                     \* The end position of the current thread download           \*/          private int endPosition;          public DownloadThread(int threadId, int startPosition, int endPosition) {              this.threadId = threadId;              this.startPosition = startPosition;              this.endPosition = endPosition;          }          @Override          public void run() {              System.out.println("thread:" + threadId + " begin working");              // lest thread download it's self range data              try {                  File finfo = new File(TOTAL\_THREAD\_COUNT + getDownloadFileName(path) + threadId + ".txt");  //                http                  if (finfo.exists() && finfo.length() > 0) {                      System.out.println("Broken point resume start");                      FileInputStream fis = new FileInputStream(finfo);                      BufferedReader br = new BufferedReader(new InputStreamReader(fis));                      String lastPosition = br.readLine();                      // This thread download data before times;                      int intLastPosition = Integer.parseInt(lastPosition);                      startPosition = intLastPosition;                      fis.close();                  }                  URL url = new URL(path);                  HttpURLConnection conn = (HttpURLConnection) url.openConnection();                  conn.setRequestMethod("GET");                  System.out.println("begin and end:" + threadId + " range of download: " + startPosition + "~~" + endPosition);                  conn.setRequestProperty("Range", "bytes=" + startPosition + "-" + endPosition);                  // Download Resource from server                  int code = conn.getResponseCode();                  if (code == 206) {                      InputStream is = conn.getInputStream();  //                    RandomAccessFile raf = new RandomAccessFile(getDownloadFileName(path), "rw");                      RandomAccessFile raf = new RandomAccessFile(getDownloadFileName(path), "rw");                      // vary important, position of begin to write                      raf.seek(startPosition);                      byte[] buffer = new byte[1024 \* 100];                      int len = -1;                      int total = 0; // downloaded data of current thread in this times;                      while ((len = is.read(buffer)) != -1) {                          raf.write(buffer, 0, len);                          // record position of current thread to downloading                          total += len;                          RandomAccessFile inforaf = new RandomAccessFile(TOTAL\_THREAD\_COUNT + getDownloadFileName(path) + threadId + ".txt", "rwd");                          // save position of current thread                          inforaf.write(String.valueOf(startPosition + total).getBytes());                          inforaf.close();                      }                      is.close();                      raf.close();                      System.out.println("thread:" + threadId + " download complete...");                  } else {                      System.out.println("request download failed.");                  }              } catch (Exception e) {                  e.printStackTrace();              } finally {                  synchronized (MultiDownloaFile.class) {                      runningThreadCount--;                      if (runningThreadCount <= 0) {                          System.out.println("  all  multi thread download complete.   success!!!");                          for (int i = 0; i < TOTAL\_THREAD\_COUNT; i++) {                              File finfo = new File(TOTAL\_THREAD\_COUNT + getDownloadFileName(path) + i + ".txt");                              System.out.println(finfo.delete());                          }                      }                  }              }          }      }  } |

<https://www.programmersought.com/article/82205159740/>

<https://tuhocict.com/thread-multi-threading-lap-trinh-da-luong/>

## Example upload file multi-threading

Vấn đề: Phía máy chủ chỉ có thể xử lý một máy khách, khi máy khách B để yêu cầu thời gian truyền, máy chủ nếu yêu cầu không được xử lý A, B chỉ có thể chờ đợi.

V: máy chủ đa luồng giải quyết nhiều yêu cầu của khách hàng: Hình ảnh của:

(1) khách hàng, phù hợp với trước đó.

(2) máy chủ, sử dụng đa luồng: xử lý đồng thời nhiều yêu cầu của khách hàng.

Dùng @Override

|  |
| --- |
| package com.yidongxueyuan.tcp;    import java.io.File;  import java.io.FileOutputStream;  import java.io.IOException;  import java.io.InputStream;  import java.io.OutputStream;  import java.net.Socket;    public class PicThread implements Runnable {        private Socket s; // process different clients:      // tiếp nhận thông tin      public PicThread( Socket s){          this.s=s;      }        int count = 1; // count      @Override      public void run() {               // define a stream: get the data          try {              String ip= s.getInetAddress().getHostAddress();                InputStream in= s.getInputStream();               // encapsulates a parent path:              File parent = new File("C:/Users/Mrzhang/javaEE/javaSE-07/javaSE-52");               // whole path:                File file = new File(parent, ip+(count++)+".jpg");                 while (file.exists ()) // add judgment, the file exists, the file name re-assignment.                     file = new File(parent, ip+(count++)+".jpg");                FileOutputStream fos = new FileOutputStream(file);              byte b[]= new byte[1024];              int len =0 ;              while((len= in.read(b))!=-1){                  fos.write(b,0,len);              }                s.shutdownInput();                OutputStream out = s.getOutputStream();               out.write ( "uploaded successfully" .getBytes ());              } catch (IOException e) {              e.printStackTrace();          }        }    } |

<https://www.programmersought.com/article/57202078785/>

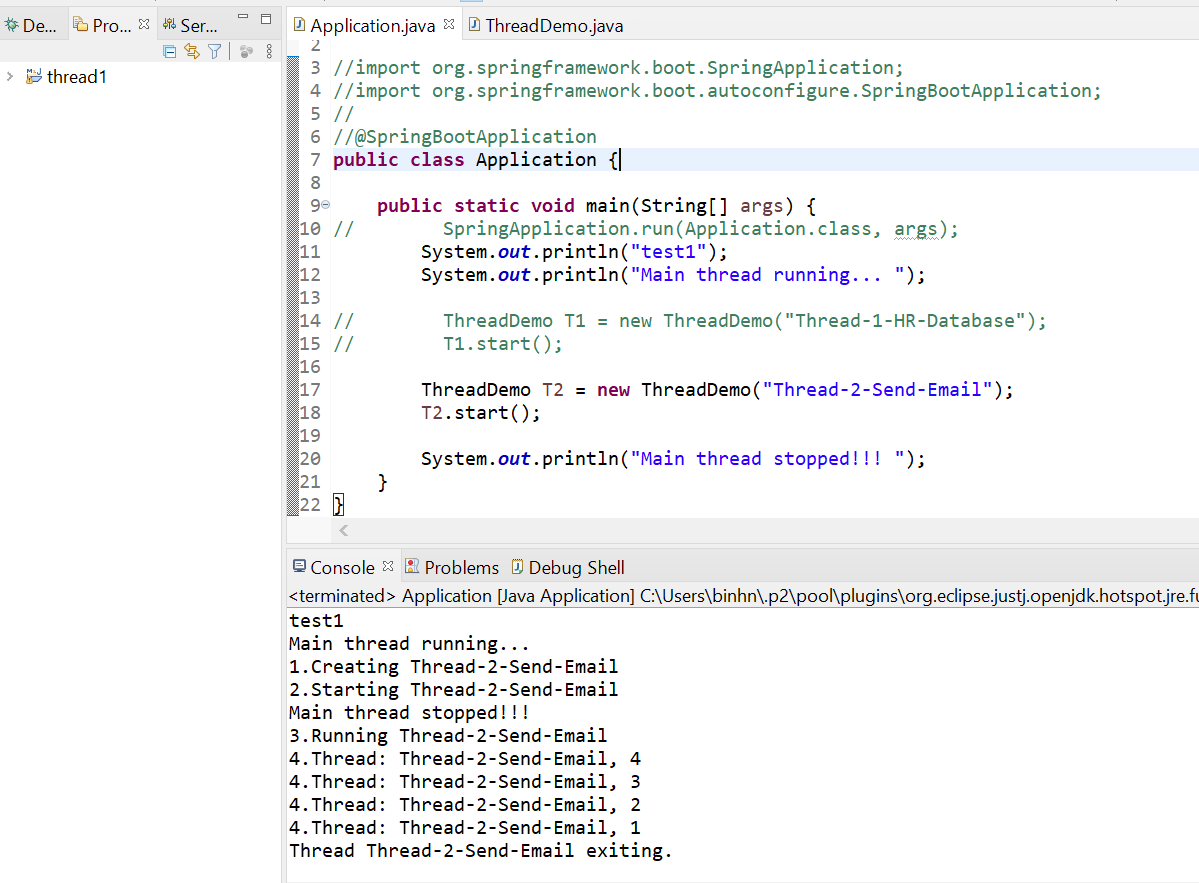
# example sử dụng đa luồng

## example1 Tạo luồng bằng cách extend từ class Thread

### Th1: start thread

|  |
| --- |
| **package** com.thread1;  **public** **class** ThreadDemo **extends** Thread {  **private** Thread t;  **private** String threadName;    ThreadDemo(String name) {  threadName = name;  System.***out***.println("1.Creating " + threadName);  }    @Override  **public** **void** run() {  System.***out***.println("3.Running " + threadName);  **try** {  **for** (**int** i = 4; i > 0; i--) {  System.***out***.println("4.Thread: " + threadName + ", " + i);  // Let the thread sleep for a while.  Thread.*sleep*(50);  }  } **catch** (InterruptedException e) {  System.***out***.println("Thread " + threadName + " interrupted.");  }  System.***out***.println("Thread " + threadName + " exiting.");  }    **public** **void** start() {  System.***out***.println("2.Starting " + threadName);  **if** (t == **null**) {  t = **new** Thread(**this**, threadName);  t.start();  }  }    } |

|  |
| --- |
| **package** com.thread1;  //import org.springframework.boot.SpringApplication;  //import org.springframework.boot.autoconfigure.SpringBootApplication;  //  //@SpringBootApplication  **public** **class** Application {    **public** **static** **void** main(String[] args) {  // SpringApplication.run(Application.class, args);  System.***out***.println("test1");  System.***out***.println("Main thread running... ");    // ThreadDemo T1 = new ThreadDemo("Thread-1-HR-Database");  // T1.start();    ThreadDemo T2 = **new** ThreadDemo("Thread-2-Send-Email");  T2.start();    System.***out***.println("Main thread stopped!!! ");  }  } |



Quy trình: Để chạy một thread:

Bước 1: new thread

Bước 2: thread.start()

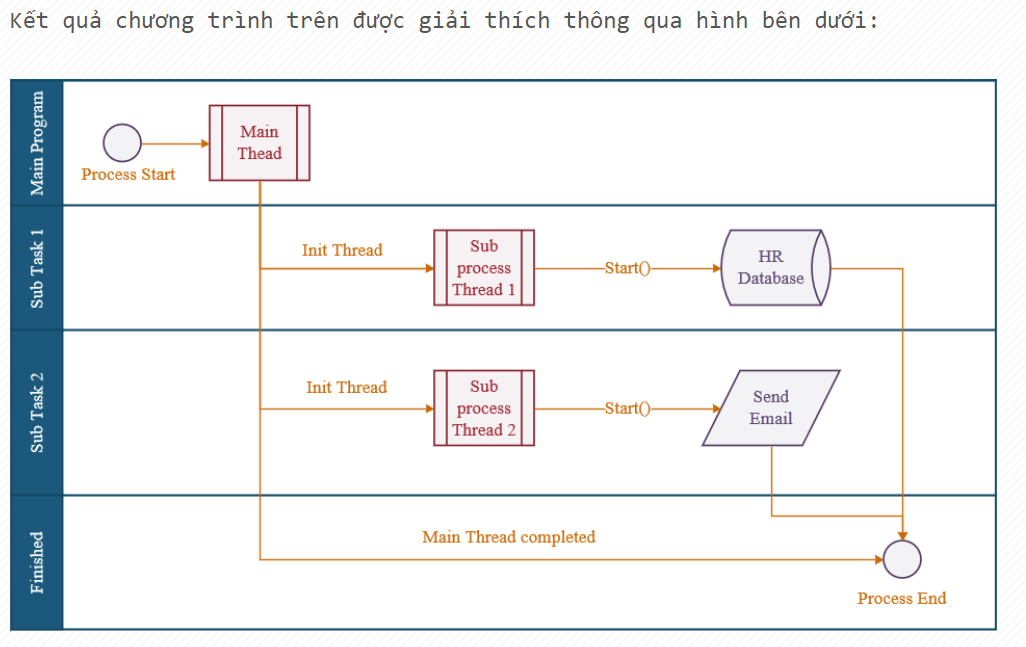
Thực thi:

|  |
| --- |
| ThreadDemo(String name) {  threadName = name;  System.***out***.println("1.Creating " + threadName);  } |

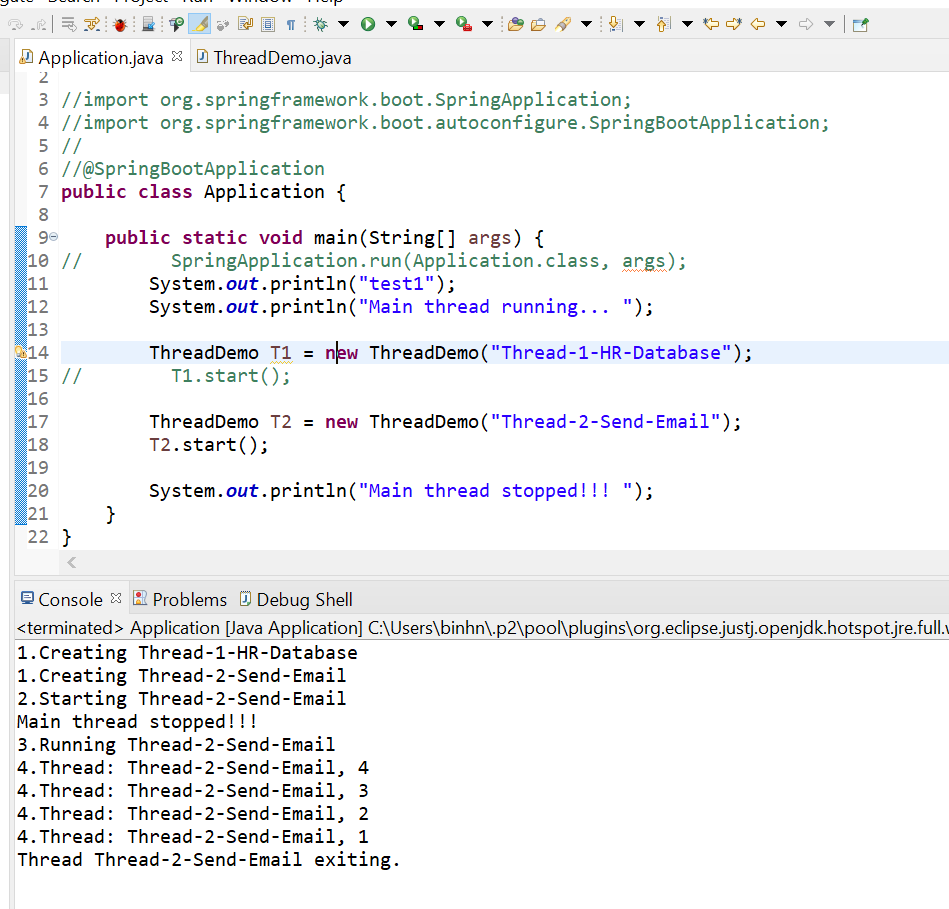
|  |
| --- |
| **public** **void** start() {  System.***out***.println("2.Starting " + threadName);  **if** (t == **null**) {  t = **new** Thread(**this**, threadName);  t.start();  }  } |

Method run chạy sau cùng

|  |
| --- |
| @Override  **public** **void** run() {  System.***out***.println("3.Running " + threadName);  **try** {  **for** (**int** i = 4; i > 0; i--) {  System.***out***.println("4.Thread: " + threadName + ", " + i);  // Let the thread sleep for a while.  Thread.*sleep*(50);  }  } **catch** (InterruptedException e) {  System.***out***.println("Thread " + threadName + " interrupted.");  }  System.***out***.println("Thread " + threadName + " exiting.");  } |



### Th2: no start thread



Thì sẽ không chạy hàm start và run

|  |
| --- |
| **public** **void** start() {  System.***out***.println("2.Starting " + threadName);  **if** (t == **null**) {  t = **new** Thread(**this**, threadName);  t.start();  }  } |

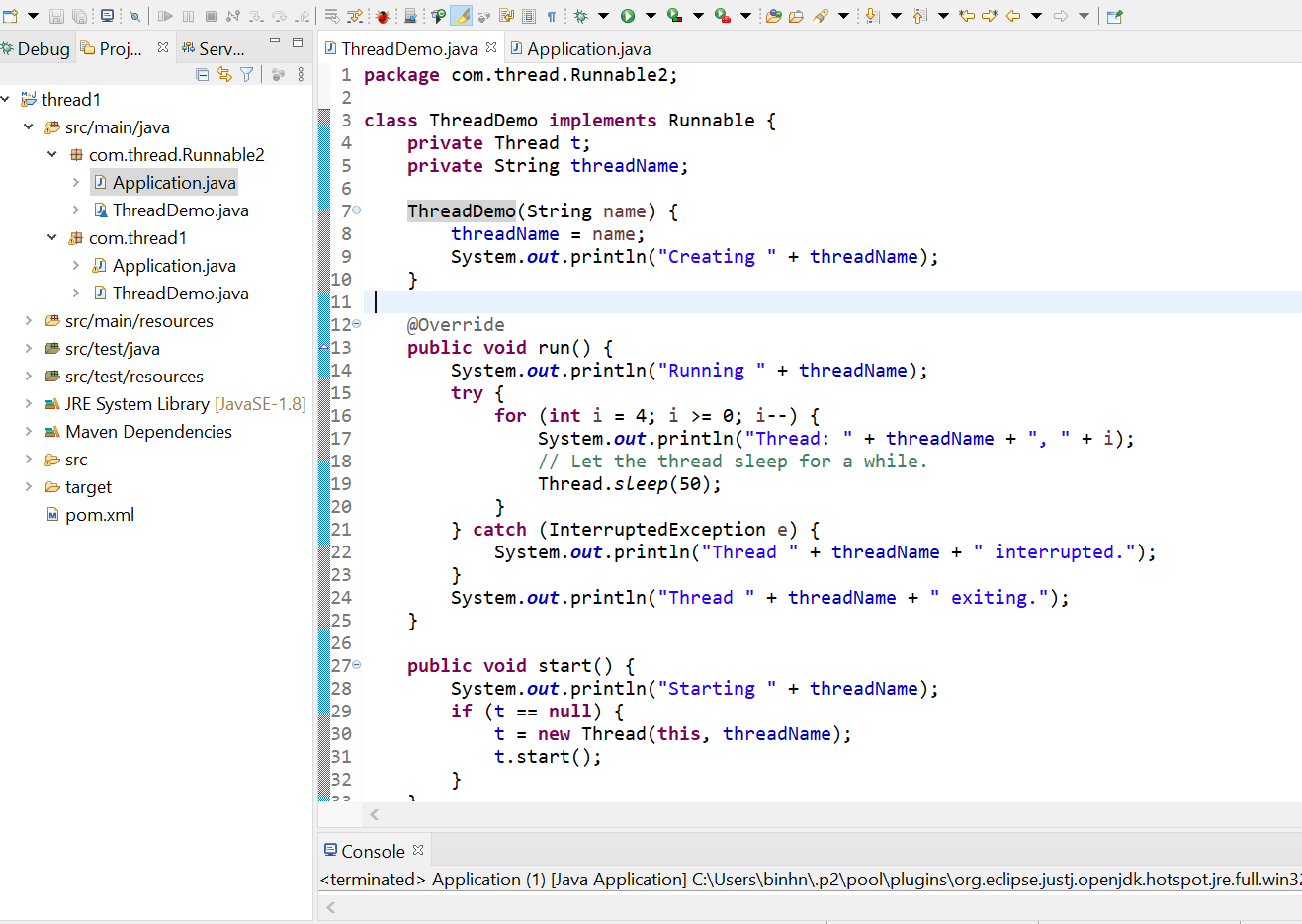
Method run chạy sau cùng

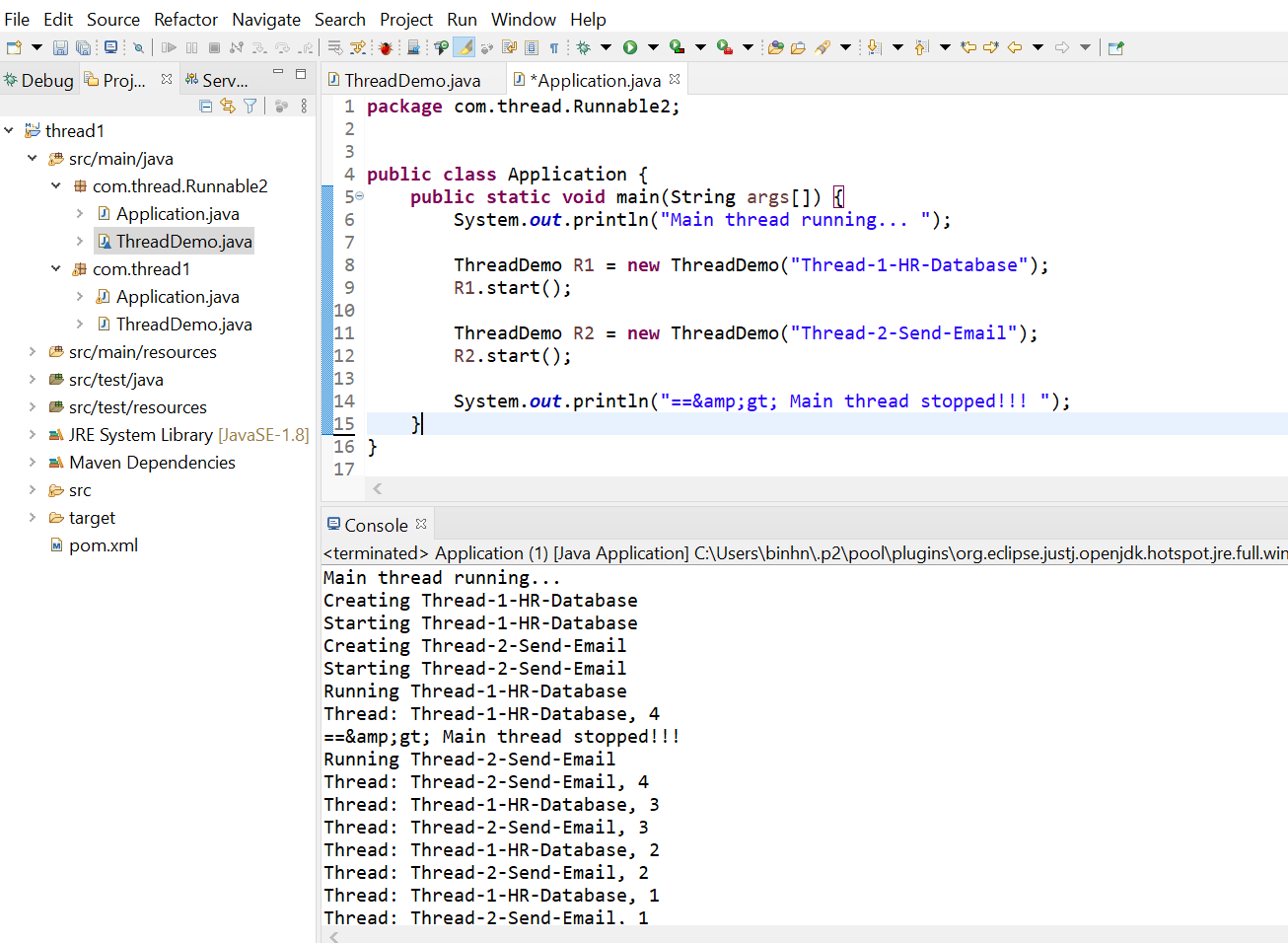
|  |
| --- |
| @Override  **public** **void** run() {  System.***out***.println("3.Running " + threadName);  **try** {  **for** (**int** i = 4; i > 0; i--) {  System.***out***.println("4.Thread: " + threadName + ", " + i);  // Let the thread sleep for a while.  Thread.*sleep*(50);  }  } **catch** (InterruptedException e) {  System.***out***.println("Thread " + threadName + " interrupted.");  }  System.***out***.println("Thread " + threadName + " exiting.");  } |

### link:

<https://gpcoder.com/3484-lap-trinh-da-luong-trong-java-java-multi-threading/>

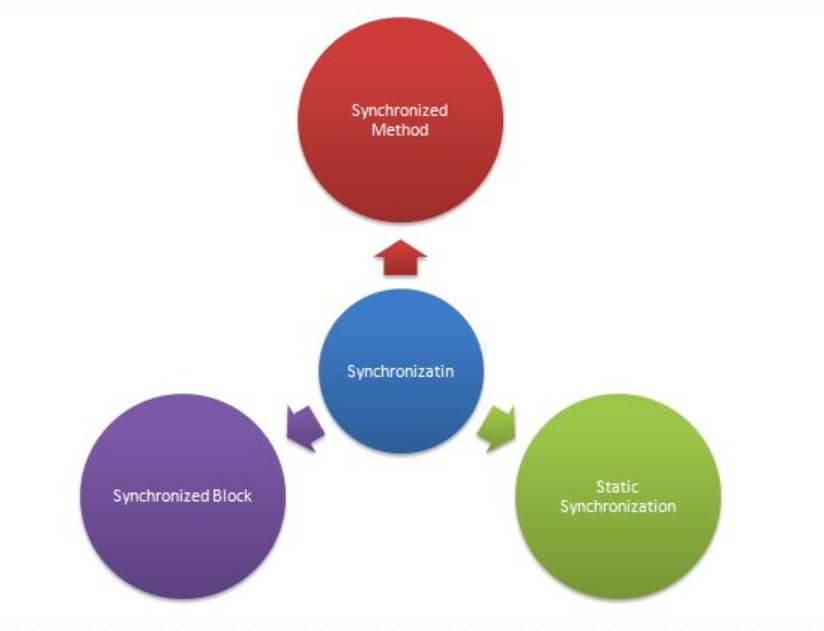
## example2 Ví dụ Tạo luồng bằng cách implement từ Interface Runnable





# Các cách để đồng bộ trong Java

 Có 3 cách để đồng bộ là: synchronized methods và synchronized statements, static synchronized method.



## 1.Synchronized methods

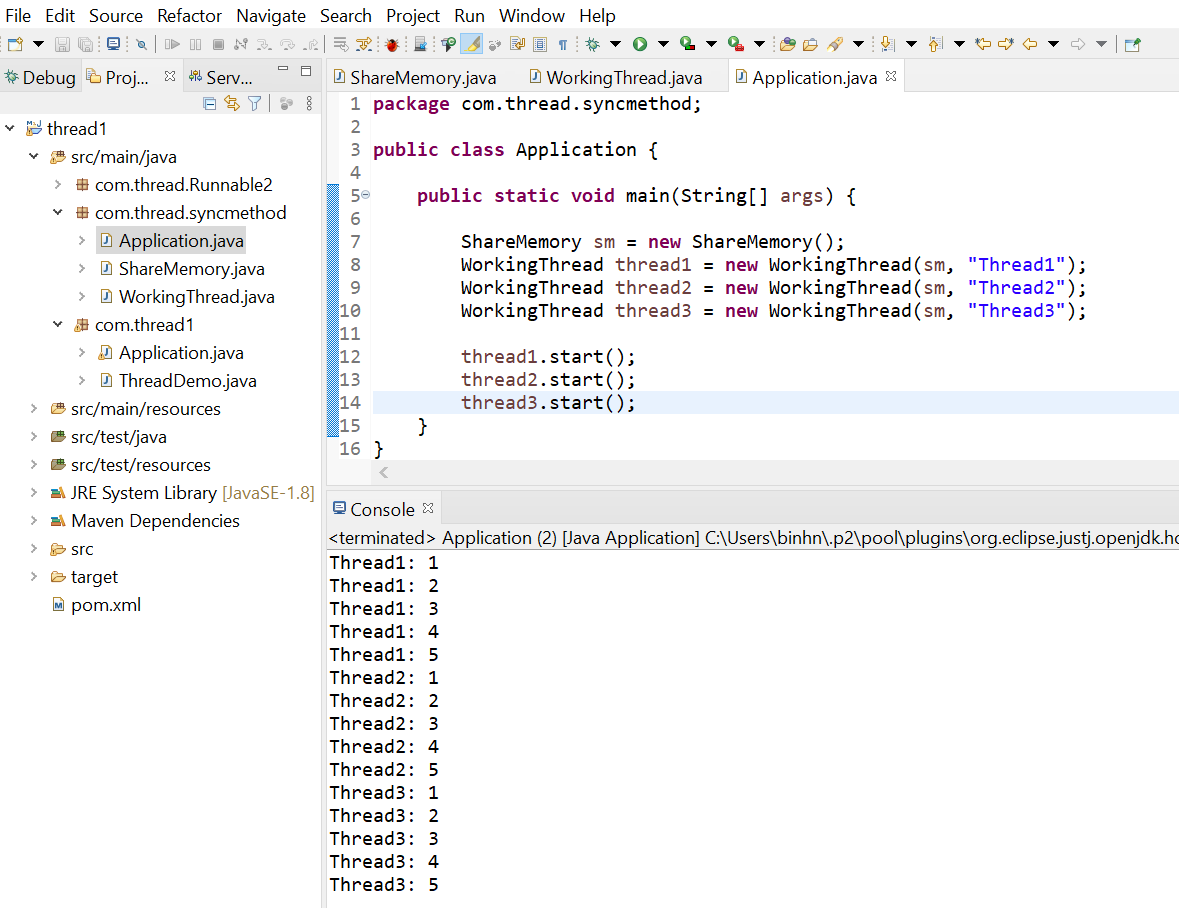
### Th1: synchronized trong methods

|  |
| --- |
| **package** com.thread.syncmethod;  **public** **class** ShareMemory {  **public** **synchronized** **void** printData(String threadName) {  **for** (**int** i = 1; i <= 5; i++) {  System.***out***.println(threadName + ": " + i);  }  }  } |

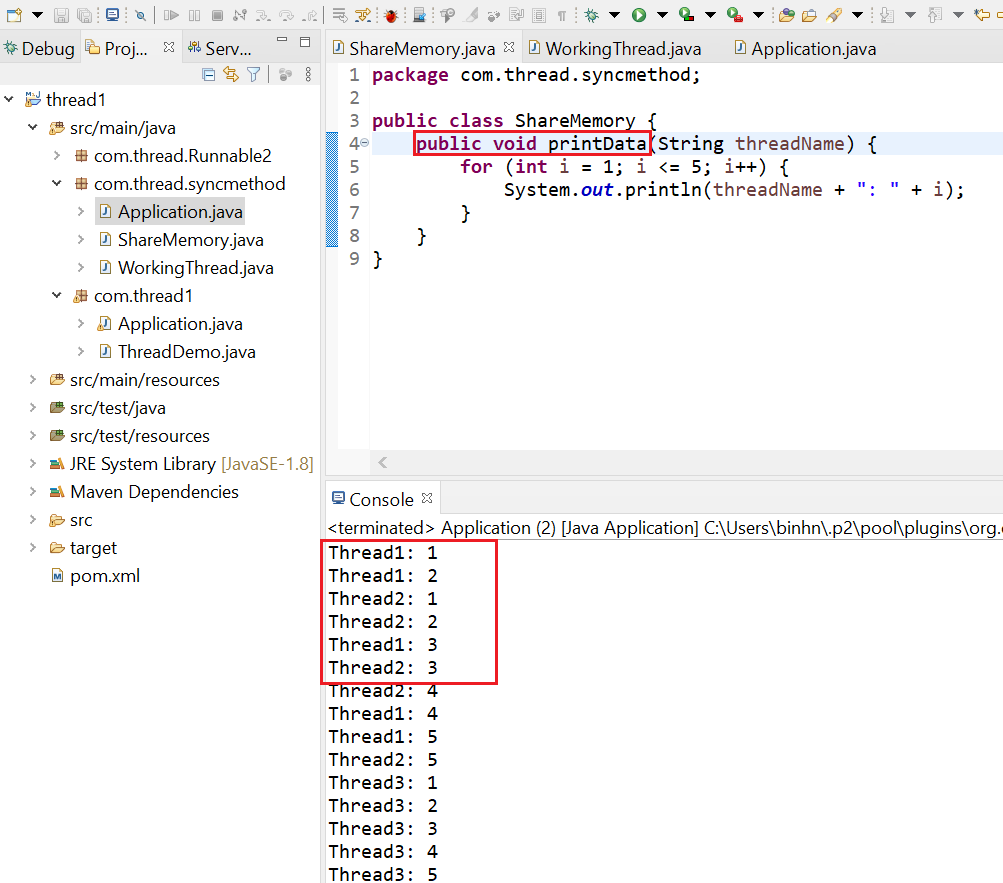
|  |
| --- |
| **package** com.thread.syncmethod;  **public** **class** WorkingThread **extends** Thread {  **private** ShareMemory mShareMemory;  **private** String mThreadName;    **public** WorkingThread(ShareMemory sm, String threadName) {  **this**.mShareMemory = sm;  **this**.mThreadName = threadName;  }    @Override  **public** **void** run() {  mShareMemory.printData(mThreadName);  }  } |

|  |
| --- |
| **package** com.thread.syncmethod;  **public** **class** Application {  **public** **static** **void** main(String[] args) {  ShareMemory sm = **new** ShareMemory();  WorkingThread thread1 = **new** WorkingThread(sm, "Thread1");  WorkingThread thread2 = **new** WorkingThread(sm, "Thread2");  WorkingThread thread3 = **new** WorkingThread(sm, "Thread3");    thread1.start();  thread2.start();  thread3.start();  }  } |

Kết quả: luồng được chạy theo trình tự

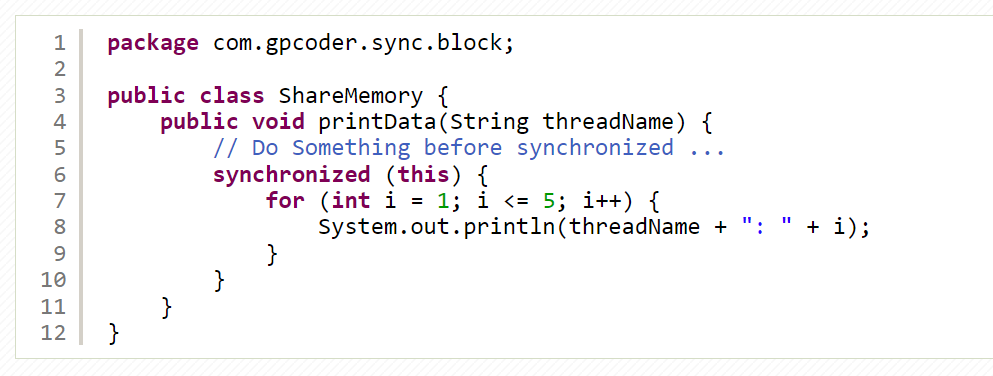


### Th2: không có synchronized trong methods, thì kết quả sẽ bị sai khi chạy đa luồng



## 2.Synchronized statements/ Synchronized Block

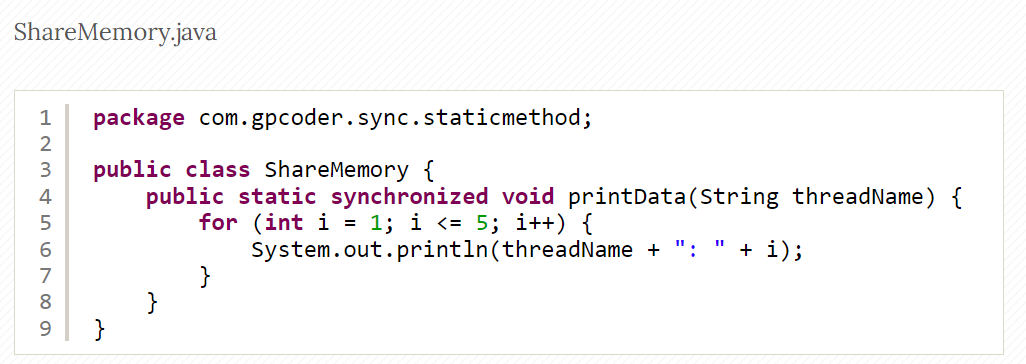
Với Synchronized methods thời gian chờ giữa các luồng khá lớn. Trong một vài trường hợp cụ thể, bạn cần hiệu suất và đồng bộ hóa cả hai cùng một lúc trong một ứng dụng. Java cung cấp cơ chế đồng bộ hóa một phần của code trong một phương thức (Synchronized statements/ Synchronized Block). Vì vậy, nhiều thread có thể truy cập vào các thông tin mà không cần phải được đồng bộ.



## 3.Static synchronized method

Ngoài 2 cách trên, chúng ta có thể synchronized một static method.

Mỗi lớp được load trong Java có một thể hiện tương ứng của java.lang.Object và được sử dụng để đại diện cho lớp đó. Một khóa của java.lang.Object được sử dụng để bảo vệ các phương thức static synchronized của lớp đó.



## Link:

<https://gpcoder.com/3514-dong-bo-hoa-cac-luong-trong-java/>

# Phương thức wait()/sleep()

## Example hệ thống rút tiền ATM

Ví dụ mô phỏng hệ thống rút tiền ATM: khách hàng chỉ được rút tiền nếu số tiền rút nhỏ hơn số tiền hiện có trong tài khoản. Nếu số tiền rút lớn hơn số tiền hiện có thì phải chờ (wait) nạp đủ tiền mới được rút, sau khi nạp tiền thì thực hiện thông báo (notify) có thể tiếp tục xử lý rút tiền.

**wait()**

Phương thức này sẽ làm cho luồng đang sở hữu monitor của đối tượng b (hay luồng đang khóa đối tượng b và nắm giữ đối tượng này) tạm thời ngưng hoạt động và trả monitor của b cho luồng khác. Sau khi trả monitor luồng 1 sẽ về trạng thái đợi (nằm ở vùng wait set. Trạng thái này java định nghĩa là **Thread.State.WAITING**).

**Notify() và notifyall()**

Sau khi luồng 2 nắm giữ monitor của b và xử lý xong những gì luồng 1 cần, thì luồng 2 sẽ gọi phươg thức notify() hoặc notifyall() trên đối tượng b để đánh thức các luồng đang chờ monitor của b và ngay sau đó luồng 2 sẽ trả lại monitor của b.

**Điểm khác nhau giữa notify() và notifyall()**là:notify()sẽ gửi thông điệp đánh thức cho 1 luồng ngẫu nhiên trong các luồng đang chờ, còn notifyall() sẽ gửi cho tất cả các luồng đang chờ b. Tuy nhiên, thông điệp gửi bởi notify() như đã nói, nó sẽ đánh thức 1 luồng bất kỳ chứ không chắc chắn là luồng 1 nên Oracle khuyến cáo nên dùng notifyall().

### Example1: trường hợp sử dụng hàm notify()

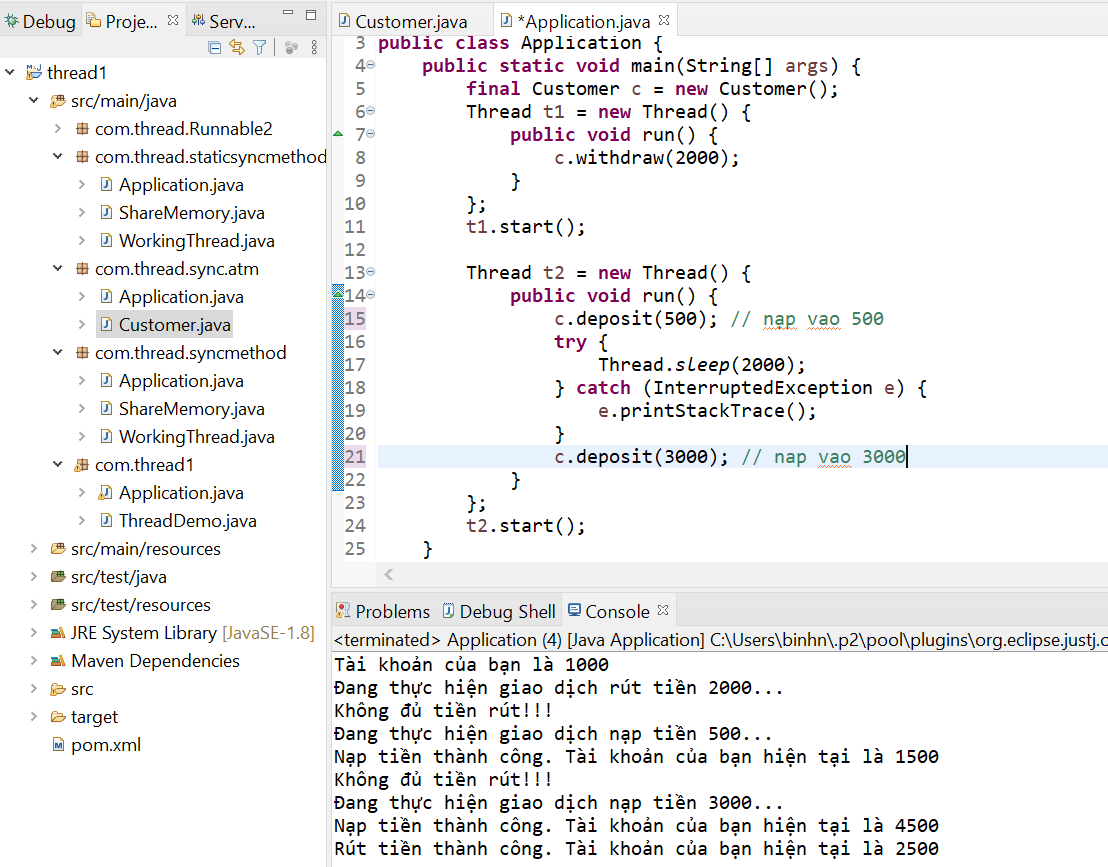
Sử dùng hàm wait(); khi không đủ tiền chờ nạp tiền

Sử dụng hàm notify(); khi nạp tiền sẽ đánh thức ngẫu nhiên một hàm wait() để rút tiền

|  |
| --- |
| **package** com.thread.sync.atm;  **public** **class** Customer {  **private** **int** balance = 1000;    **public** Customer() {  System.***out***.println("Tài khoản của bạn là " + balance);  }    **public** **synchronized** **void** withdraw(**int** amount) {  System.***out***.println("Đang thực hiện giao dịch rút tiền " + amount + "...");  **while** (balance < amount) {  System.***out***.println("Không đủ tiền rút!!!");  **try** {  wait(); // Chờ nạp tiền  } **catch** (InterruptedException ie) {  System.***out***.println(ie.toString());  }  }  balance -= amount;  System.***out***.println("Rút tiền thành công. Tài khoản của bạn hiện tại là " + balance);  }    **public** **synchronized** **void** deposit(**int** amount) {  System.***out***.println("Đang thực hiện giao dịch nạp tiền " + amount + "...");  balance += amount;  System.***out***.println("Nạp tiền thành công. Tài khoản của bạn hiện tại là " + balance);  notify(); // Thông báo đã nạp tiền  }    } |

|  |
| --- |
| **package** com.thread.sync.atm;  **public** **class** Application {  **public** **static** **void** main(String[] args) {  **final** Customer c = **new** Customer();  Thread t1 = **new** Thread() {  **public** **void** run() {  c.withdraw(2000);  }  };  t1.start();    Thread t2 = **new** Thread() {  **public** **void** run() {  c.deposit(500); // nạp vao 500  **try** {  Thread.*sleep*(2000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  c.deposit(3000); // nạp vao 3000  }  };  t2.start();  }  } |

Kết quả:



### Example2: trường hợp không sử dụng hàm notify()

Sử dùng hàm wait(); khi không đủ tiền chờ nạp tiền

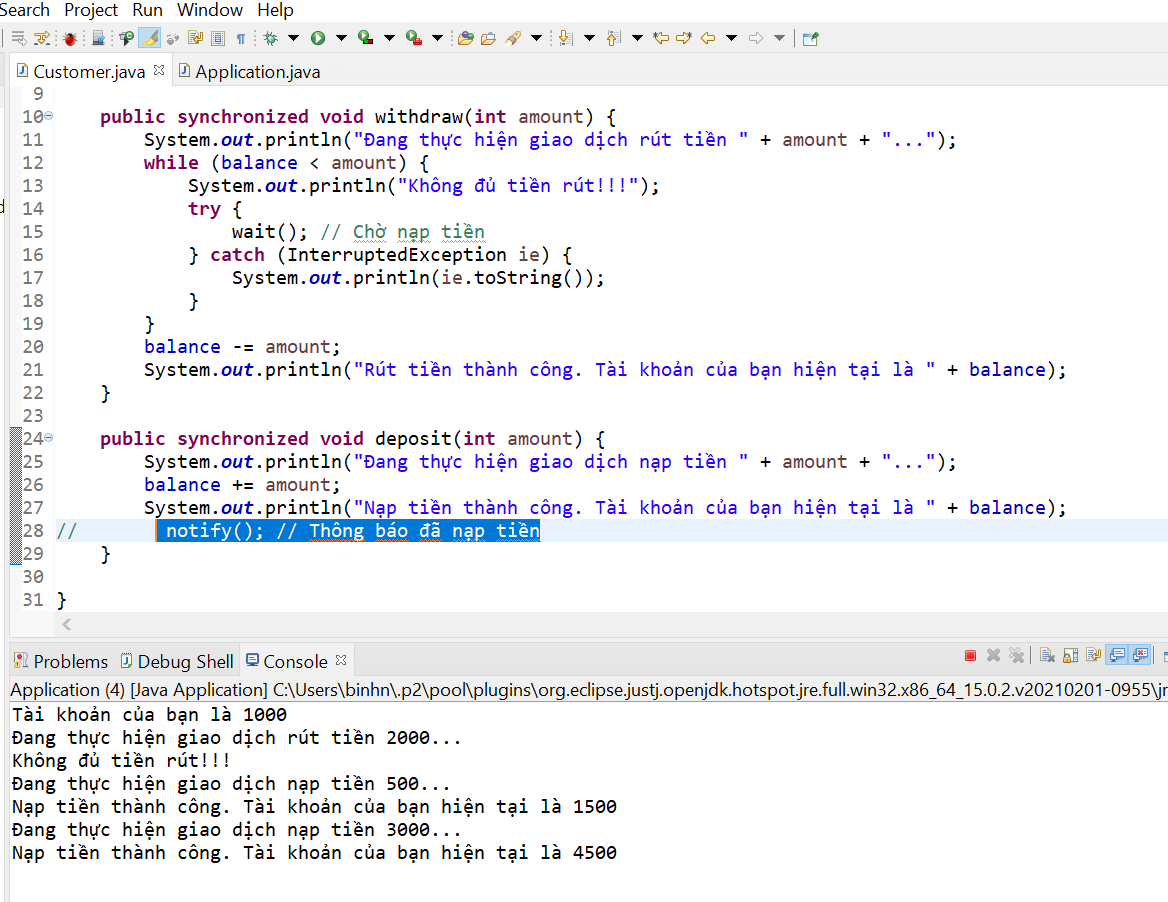
|  |
| --- |
| **package** com.thread.sync.atm;  **public** **class** Customer {  **private** **int** balance = 1000;    **public** Customer() {  System.***out***.println("Tài khoản của bạn là " + balance);  }    **public** **synchronized** **void** withdraw(**int** amount) {  System.***out***.println("Đang thực hiện giao dịch rút tiền " + amount + "...");  **while** (balance < amount) {  System.***out***.println("Không đủ tiền rút!!!");  **try** {  wait(); // Chờ nạp tiền  } **catch** (InterruptedException ie) {  System.***out***.println(ie.toString());  }  }  balance -= amount;  System.***out***.println("Rút tiền thành công. Tài khoản của bạn hiện tại là " + balance);  }    **public** **synchronized** **void** deposit(**int** amount) {  System.***out***.println("Đang thực hiện giao dịch nạp tiền " + amount + "...");  balance += amount;  System.***out***.println("Nạp tiền thành công. Tài khoản của bạn hiện tại là " + balance);  // notify(); // Thông báo đã nạp tiền }    } |

|  |
| --- |
| **package** com.thread.sync.atm;  **public** **class** Application {  **public** **static** **void** main(String[] args) {  **final** Customer c = **new** Customer();  Thread t1 = **new** Thread() {  **public** **void** run() {  c.withdraw(2000);  }  };  t1.start();    Thread t2 = **new** Thread() {  **public** **void** run() {  c.deposit(500); // nạp vao 500  **try** {  Thread.*sleep*(2000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  c.deposit(3000); // nạp vao 3000  }  };  t2.start();  }  } |

**Kết quả:**

khi nạp đủ tiền sẽ không tự động rút tiền hay là sẽ không có hàm đánh thức hàm wait()

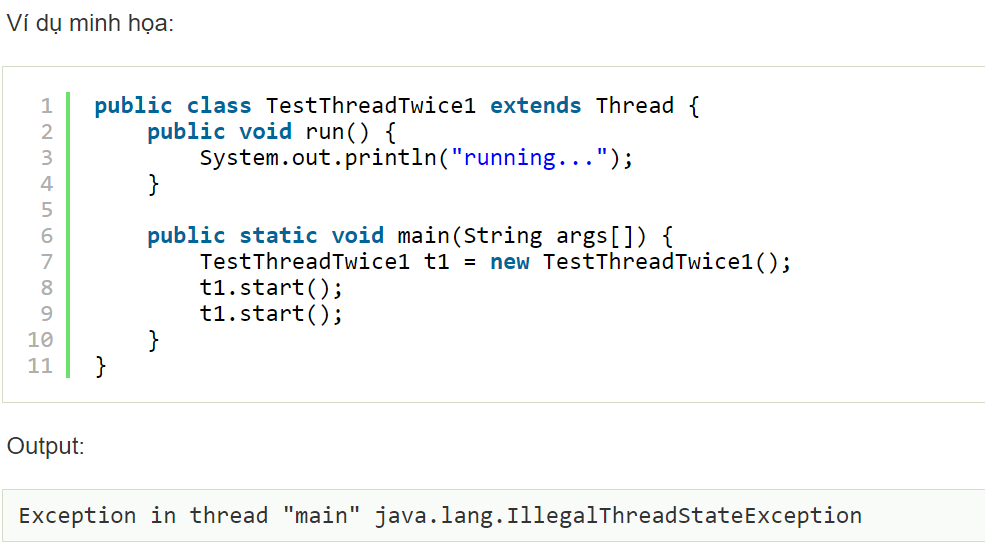
Khi này hàm wait() sẽ không có ý nghĩa



# Câu hỏi:

## Có thể start một thread hai lần không?

Không. Sau khi start một thread, nó không bao giờ có thể được start lại. Nếu bạn làm như vậy, một ngoại lệ IllegalThreadStateException sẽ xảy ra.



<https://gpcoder.com/3484-lap-trinh-da-luong-trong-java-java-multi-threading/>