|  |  |
| --- | --- |
|  | **BỘ GIÁO DỤC VÀ ĐÀO TẠO**  **TRƯỜNG ĐẠI HỌC CNTT GIA ĐỊNH**  **KHOA CÔNG NGHỆ THÔNG TIN** |

**BÁO CÁO**

**ĐỀ TÀI CUỐI KÌ**

**TÊN ĐỀ TÀI**

**THREAD**

**Giảng viên hướng dẫn: TRẦN VĂN MINH**

**Nhóm thực hiện: 12**

**Tên thành viên:**

**BÙI QUỐC KHÁNH-1731102060**

**TRẦN HÔNG LĨNH-1731102076**

**VŨ NGUYỄN HOÀNG PHÁT-**

**TẠ VĂN TRỰC-**

**Lớp: 11DHCT2A-Nhóm 2**

**Khóa: 11**

*Thành phố Hồ Chí Minh, tháng 11 năm 2018*

**LỜI CẢM ƠN**

Qua kì học vừa rồi được sự chỉ bảo và giảng dạy nhiệt tình của thầy Minh đã truyền đạt cho em những kiến thức về lý thuyết và thực hành trong suốt thời gian học vừa qua.

Sau một khoảng thơi gian tìm hiểu và thực hiện đề tài "Thread" đã hoàn thành, ngoài sự nỗ lực của bản thân, còn là sự hợp tác, giúp đở lẫn nhau của cả nhóm.

Em xin cảm ơn Thầy Trần Văn Minh đã hướng dẫn những thứ cần thiết để giúp chúng em hoàn thành đề tài này.

Trong quá trình làm bài báo cáo đề tài, khó tránh khỏi sai sót, rất mong Thầy bỏ qua. Đồng thời do trình độ lý luận cũng như kinh nghiệm thực tiễn còn hạn chế nên bài báo cáo không thể tránh khỏi những thiếu sót, em rất mong nhận được ý kiến đóng góp của Thầy để em học thêm được nhiều kinh nghiệm và sẽ hoàn thành tốt hơn các bài sau nữa.

Em xin chân thành cảm ơn!

**`**

MỤC LỤC

[PHẦN I: GIỚI THIỆU VỀ THREAD](#_Toc503453948) 1

[I.1 Thread là gì? Multi-thread là gì](#_Toc503453949) 1

[I.2 Đa nhiệm (Multitasking)](#_Toc503453950) 2

[I.3 Ưu điểm của đa luồng](#_Toc503453951) 3

[I.4 Nhược điểm của đa luồng:](#_Toc503453962) 3

**[Phần II: VÒNG ĐỜI (CÁC TRẠNG THÁI) CỦA MỘT THREAD TRONG JAVA 4](#_Toc503453963)**

[**Phần III: CÁCH TẠO LUỒNG TRONG JAVA**](#_Toc503453964) **5**

[III.1 Tạo luồng bằng cách extend từ lớp Thread: 6](#_Toc503453965)

[III.2 Tạo luồng bằng cách implement từ lớp Interface Runnable: 7](#_Toc503453966)

[III.3 Khi nào implements từ interface Runnable? 8](#_Toc503453967)

[III.4 Ví dụ minh họa sử dụng đa luồng:](#_Toc503453968) 9

[III.4.1 Tạo luồng bằng cách extend từ class Thread: 9](#_Toc503453970)

[III.4.2 Tạo luồng bằng cách implement từ class Interface Runnable: 12](#_Toc503453971)

**[Phần IV: CÁC PHƯƠNG THỨC CỦA LỚP THREAD THƯỜNG HAY SỬ DỤNG 16](#_Toc503453972)**

**Phần V: PHÂN BIỆT GIỮA MÔ HÌNH XỬ LÝ DÙNG MULTITHREAD VÀ NHIỀU PROCESS 18**

**Phần VI: BÀI THỰC HÀNH 20**

**Phần VII: TÀI LIỆU THAM KHẢO 26**

**Phần I: GIỚI THIỆU VỀ THREAD**

# I.1 Thread là gì? Multi-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 (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.

# VD: Trình duyệt web hay các chương trình chơi nhạc là 1 ví dụ điển hình về đa luồng.

# + Khi duyệt 1 trang web, có rất nhiều hình ảnh, CSS, javascript… được tải đồng thời bởi các luồng khác nhau.

# + Khi play nhạc, chúng ta vẫn có thể tương tác được với nút điều khiển như: Play, pause, next, back … vì luồng phát nhạc là luồng riêng biệt với luồng tiếp nhận tương tác của người dùng.

# I.2 Đa nhiệm (Multitasking) :

# Multitasking: Là khả năng chạy đồng thời một hoặc nhiều chương trình cùng một lúc trên một hệ điều hành. Hệ điều hành quản lý việc này và sắp xếp lịch phù hợp cho các chương trình đó. Ví dụ, trên hệ điều hành Windows chúng ta có làm việc đồng thời với các chương trình khác nhau như: Microsoft Word, Excel, Media Player, …

# Chúng ta sử dụng đa nhiệm để tận dụng tính năng của CPU.

# Đa nhiệm có thể đạt được bằng hai cách:

# Đa nhiệm dựa trên 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 tiến trình (multiprocessing) và đa luồng (multithreading) cả hai được sử dụng để tạo ra hệ thống đa nhiệm (multitasking). Nhưng chúng ta sử dụng đa luồng nhiều hơn đa tiến trình bởi vì các luồng chia sẻ một vùng bộ nhớ chung. Chúng không phân bổ vùng bộ nhớ riêng biệt để tiết kiệm bộ nhớ, và chuyển đổi ngữ cảnh giữa các luồng mất ít thời gian hơn tiến trình.

**I.3 Ưu điểm của đa luồng:**

# Nó không chặn người sử dụng vì các luồng là độc lập và bạn có thể thực hiện nhiều công việc cùng một lúc.

# Mỗi luồng có thể dùng chung và chia sẻ nguồn tài nguyên trong quá trình chạy, nhưng có thể thực hiện một cách độc lập.

# Luồng là độc lập vì vậy nó không ảnh hưởng đến luồng khác nếu ngoại lệ xảy ra trong một luồng duy nhất.

# Có thể thực hiện nhiều hoạt động với nhau để tiết kiệm thời gian. Ví dụ một ứng dụng có thể được tách thành : luồng chính chạy giao diện người dùng và các luồng phụ nhiệm gửi kết quả xử lý đến luồng chính.

**I.4 Nhược điểm của đa luồng:**

# Càng nhiều luồng thì xử lý càng phức tạp.

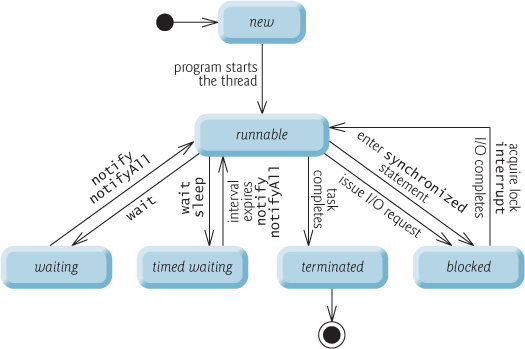
# Xử lý vấn đề về tranh chấp bộ nhớ, đồng bộ dữ liệu khá phức tạp.

# Cần phát hiện tránh các luồng chết (dead lock), luồng chạy mà không làm gì trong ứng dụng cả.

# “ Deadlock (Khoá chết) là gì? Deadlock xảy ra khi 2 tiến trình đợi nhau hoàn thành, trước khi chạy. Kết quả của quá trình là cả 2 tiến trình không bao giờ kết thúc.”

### 

**Phần II: 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.

# BLOCKED : Đây là 1 dạng của trạng thái “Not Runnable”. Thread chờ một monitor để unlock một đối tượng mà nó cần.

# 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 trạng thái khi thread vẫn còn sống, nhưng hiện tại không được chọn để chạy.

# TERMINATED : Một thread ở trong trạng thái terminated hoặc dead khi phương thức run() của nó bị thoát.

**Phần III: CÁCH TẠO LUỒNG TRONG JAVA**

# Trong java ta có thể tạo ra một luồng bằng một trong hai cách sau: tạo 1 đối tượng của lớp được extend từ class Thread hoặc implements từ interface Runnable.

**III.1 Tạo luồng bằng cách extend từ lớp Thread:**

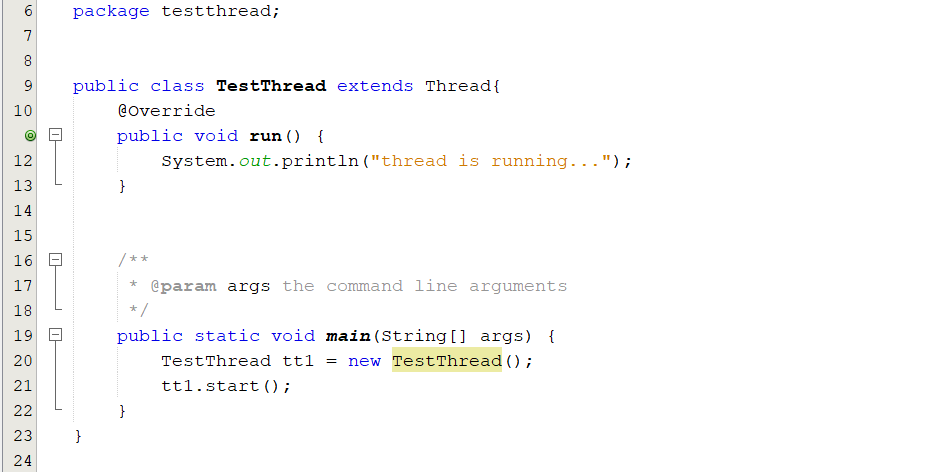
# Để tạo luồng bằng cách tạo lớp kế thừa từ lớp Thread, ta phải làm các công việc sau :

# Khai báo 1 lớp mới kế thừa từ lớp Thread

# Override lại phương thức run ở lớp này, những gì trong phương thức run sẽ được thực thi khi luồng bắt đầu chạy. Sau khi luồng chạy xong tất cả các câu lệnh trong phương thức run thì luồng cũng tự hủy.

# Tạo 1 thể hiện (hay 1 đối tượng) của lớp ta vừa khai báo.

# Sau đó gọi phương thức start() của đối tượng này để bắt đầu thực thi luồng.



# Lưu ý :

# Tuy ta khai báo những công việc cần làm của luồng trong phương thức run() nhưng khi thực thi luồng ta phải gọi phương thức start(). Vì đây là phương thức đặc biệt mà java xây dựng sẵn trong lớp Thread, phương thức này sẽ cấp phát tài nguyên cho luồng mới rồi chạy phương thức run() ở luồng này. Vì vậy, nếu ta gọi phương thức run() mà không gọi start() thì cũng tương đương với việc gọi 1 phương thức của 1 đối tượng bình thường và phương thức vẫn chạy trên luồng mà gọi phương thức chứ không chạy ở luồng mới tạo ra, nên vẫn chỉ có 1 luồng chính làm việc chứ ứng dụng vẫn không phải là đa luồng.

# 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.

**III.2 Tạo luồng bằng cách implement từ lớp Interface Runnable:**

# Để tạo luồng bằng cách hiện thực từ Interface Runnable, ta phải làm các công việc sau :

# Khai báo 1 lớp mới implements từ Interface Runnable

# Hiện thực phương thức run() ở lớp này, những gì trong phương thức run() sẽ được thực thi khi luồng bắt đầu chạy. Sau khi luồng chạy xong tất cả các câu lệnh trong phương thức run thì luồng cũng tự hủy.

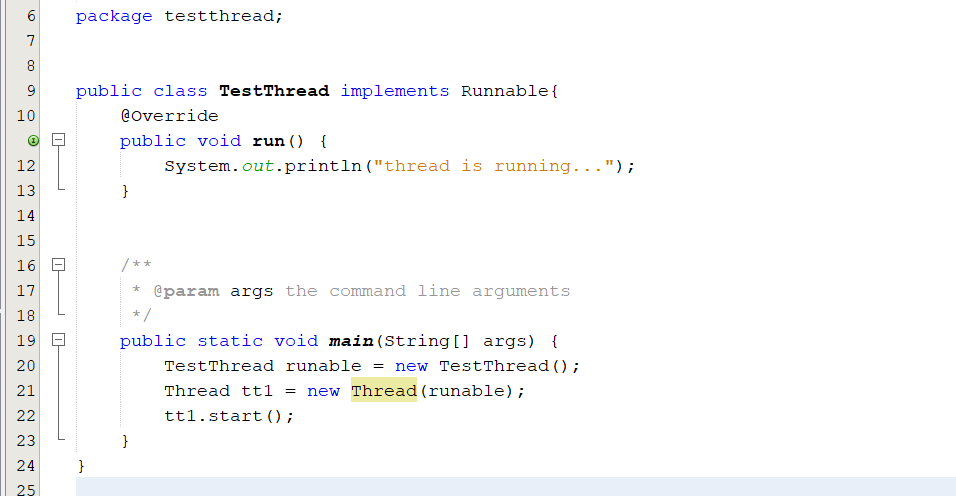
# Tạo 1 thể hiện (hay 1 đối tượng) của lớp ta vừa khai báo. (VD : Tên đối tượng là r1)

# Tạo 1 thể hiện của lớp Thread bằng phương thức khởi tạo : Thread(Runnable target)

# Runnable target: Là 1 đối tượng thuốc lớp được implements từ giao diện Runnable.

# Ví dụ: Thread t1 = new Thread(r1);

# Gọi phương thức start() của đối tượng t1.



**III.3 Khi nào implements từ interface Runnable?**

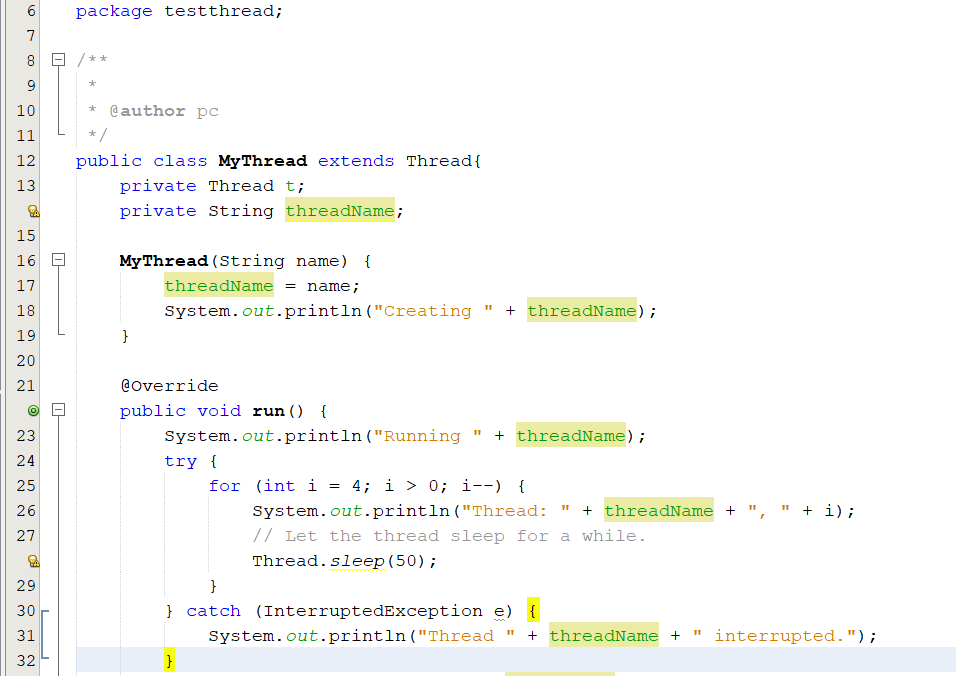
# -Cách hay được sử dụng và được yêu thích là dùng ****interface Runnable****, bởi vì nó không yêu cầu phải tạo một lớp kế thừa từ lớp Thread. Trong trường hợp ứng dụng thiết kế yêu cầu sử dụng đa kế thừa, chỉ có interface mới có thể giúp giải quyết vấn đề. Ngoài ra, ****Thread Pool**** rất hiểu quả và có thể được cài đặt, sử dụng rất hơn giản.

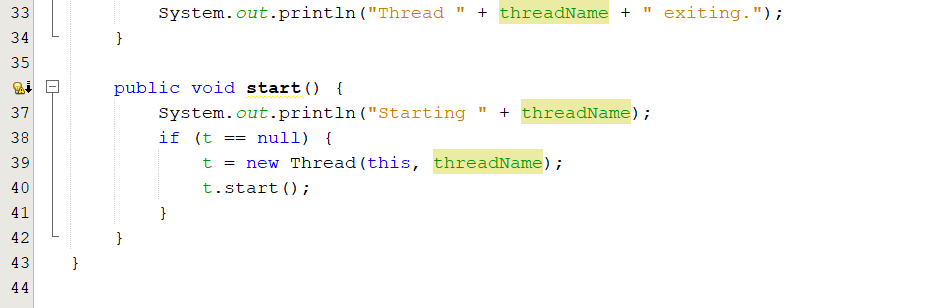
# -Trong trường hợp còn lại ta có thể kế thừa từ lớp Thread.

**III.4 Ví dụ minh họa sử dụng đa luồng:**

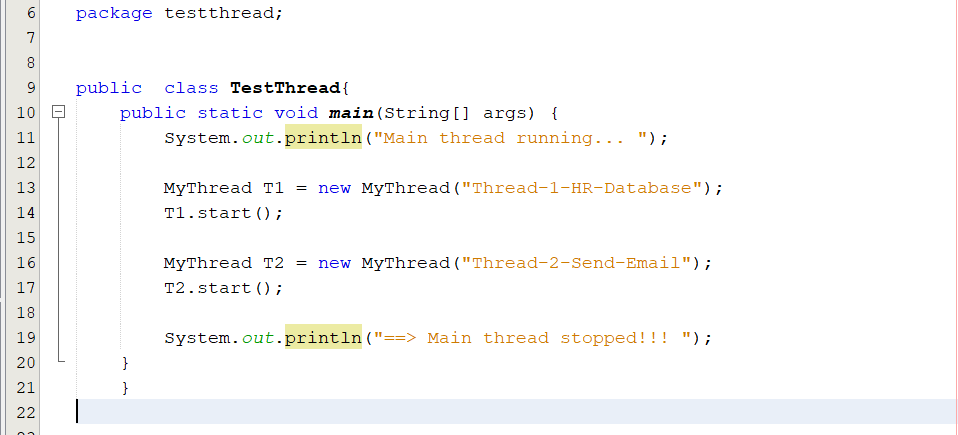
III.4.1 Tạo luồng bằng cách extend từ class Thread:

Tạo luồng extend từ class Thread

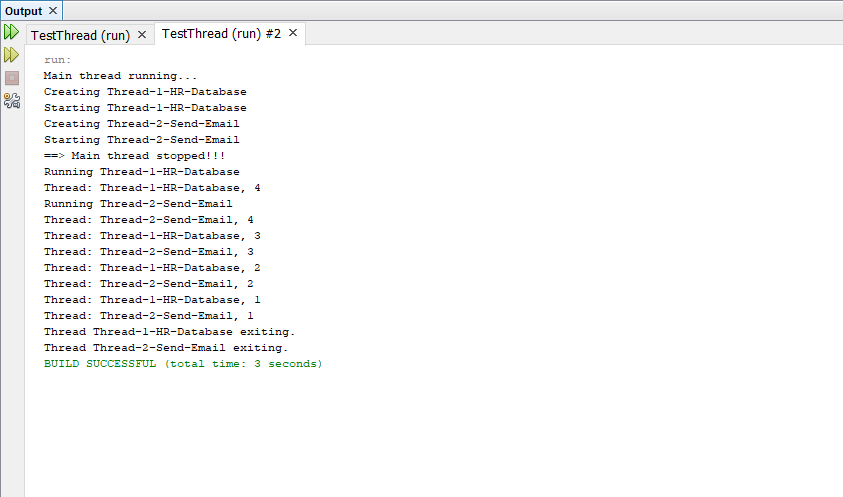




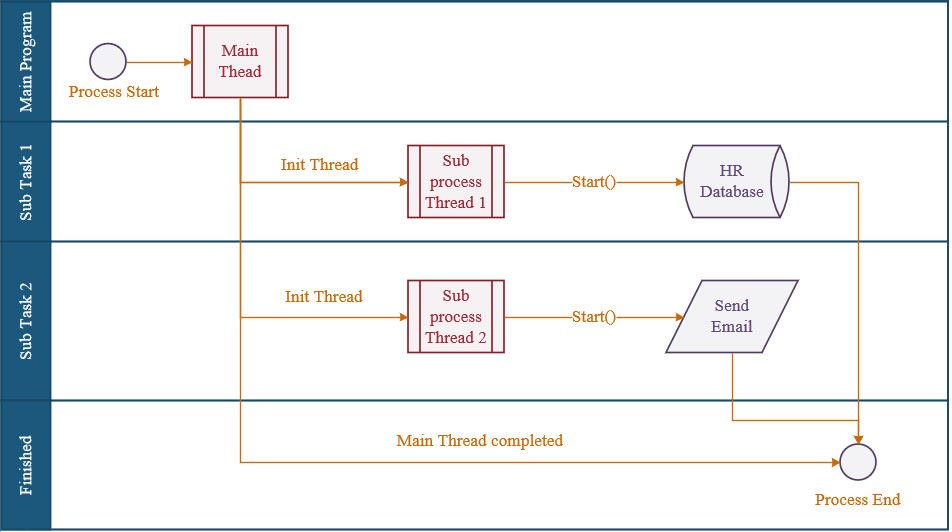
Chương trình sử dụng đa luồng:



Kết quả thực thi trương trình trên

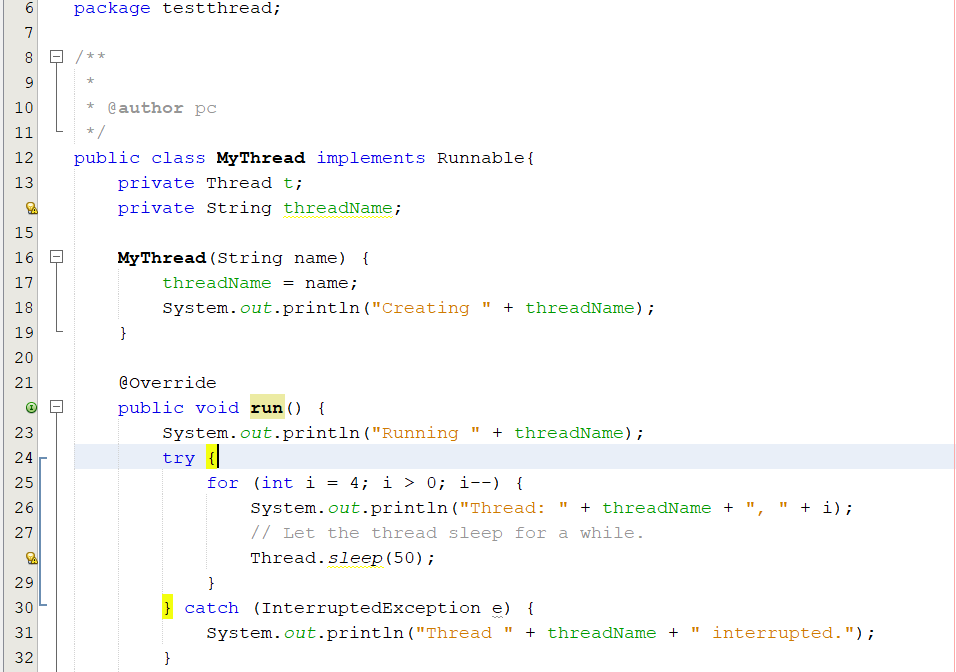


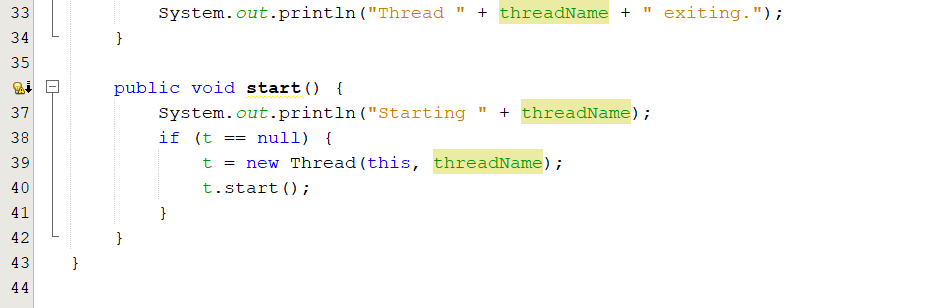
Kết quả được giải thích qua hình:



III.4.2 Tạo luồng bằng cách implement từ class Interface Runnable:

Tạo luồng implement từ Interface Runable:

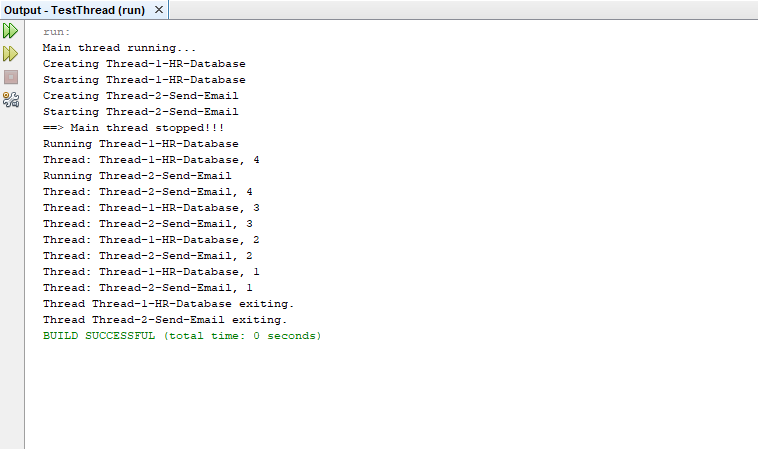




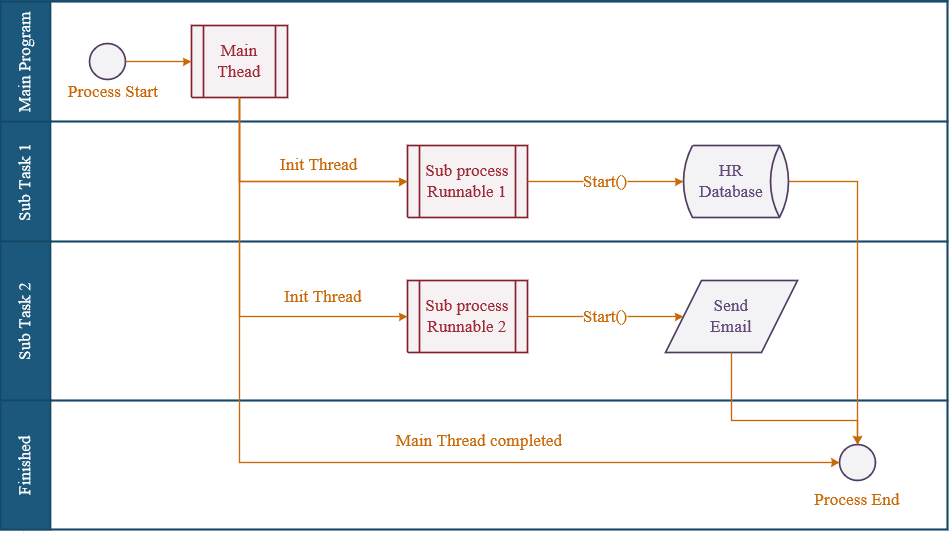
Chương trình sử dụng đa luồng



Kết quả thực thi chương trình trên:



Kết quả được giải thích qua hình dưới đây :



**Phần IV: CÁC PHƯƠNG THỨC CỦA LỚP THREAD THƯỜNG HAY SỬ DỤNG**

# suspend() : Đây là phương thức làm tạm dừng hoạt động của 1 luồng nào đó bằng các ngưng cung cấp CPU cho luồng này. Để cung cấp lại CPU cho luồng ta sử dụng phương thức resume(). Cần lưu ý 1 điều là ta không thể dừng ngay hoạt động của luồng bằng phương thức này. Phương thức suspend() không dừng ngay tức thì hoạt động của luồng mà sau khi luồng này trả CPU về cho hệ điều hành thì không cấp CPU cho luồng nữa.

# resume() : Đây là phương thức làm cho luồng chạy lại khi luồng bị dừng do phương thức suspend() bên trên. Phương thức này sẽ đưa luồng vào lại lịch điều phối CPU để luồng được cấp CPU chạy lại bình thường.

# stop() : Luồng này sẽ kết thúc phương thức run() bằng cách ném ra 1 ngoại lệ ThreadDeath, điều này cũng sẽ làm luồng kết thúc 1 cách ép buộc. Nếu giả sử, trước khi gọi stop() mà luồng đang nắm giữa 1 đối tượng nào đó hoặc 1 tài nguyên nào đó mà luồng khác đang chờ thì có thể dẫn tới việc sảy ra deadlock.

# destroy() : dừng hẳn luồng.

# isAlive() : Phương thức này kiểm tra xem luồng còn active hay không. Phương thức sẽ trả về true nếu luồng đã được start() và chưa rơi vào trạng thái dead. Nếu phương thức trả về false thì luồng đang ở trạng thái “New Thread” hoặc là đang ở trạng thái “Dead”

# yeild() : Hệ điều hành đa nhiệm sẽ phân phối CPU cho các tiến trình, các luồng theo vòng xoay. Mỗi luồng sẽ được cấp CPU trong 1 khoảng thời gian nhất định, sau đó trả lại CPU cho hệ điều hành (HĐH), HĐH sẽ cấp CPU cho luồng khác. Các luồng sẽ nằm chờ trong hàng đợi Ready để nhận CPU theo thứ tự. Java có cung cấp cho chúng ta 1 phương thức khá đặc biệt là yeild(), khi gọi phương thức này luồng sẽ bị ngừng cấp CPU và nhường cho luồng tiếp theo trong hàng chờ Ready. Luồng không phải ngưng cấp CPU như suspend mà chỉ ngưng cấp trong lần nhận CPU đó mà thôi.

# sleep(long) : tạm dừng luồng trong một khoảng thời gian millisecond.

# join() : thông báo rằng hãy chờ thread này hoàn thành rồi thread cha mới được tiếp tục chạy.

# join(long) : Thread cha cần phải đợi millisecond mới được tiếp tục chạy, kể từ lúc gọi join(long). Nếu tham số millis = 0 nghĩa là đợi cho tới khi luồng này kết thúc.

# getName() : Trả về tên của thread.

# setName(String name) : Thay đổi tên của thread.

# getId() : Trả về id của thread.

# getState(): trả về trạng thái của thread.

# currentThread() : Trả về tham chiếu của thread đang được thi hành.

# getPriority() : Trả về mức độ ưu tiên của thread.

# setPriority(int) : Thay đổi mức độ ưu tiên của thread.

# isDaemon() : Kiểm tra nếu thread là một luồng Daemon.

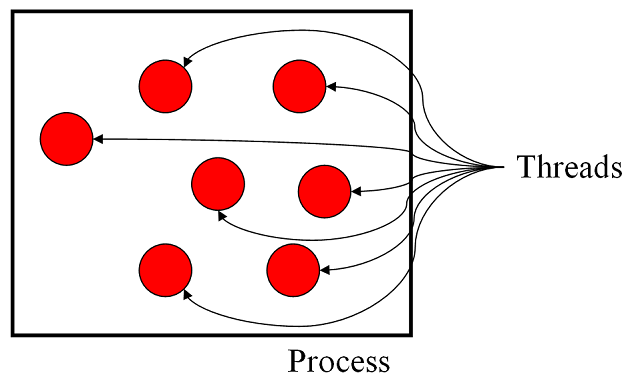
# setDaemon(boolean): xác định thread là một luồng Daemon hay không.

# interrupt() : làm gián đoạn một luồng trong java. Nếu thread nằm trong trạng thái sleep hoặc wait, nghĩa là sleep() hoặc wait() được gọi ra. Việc gọi phương thức interrupt() trên thread đó sẽ phá vỡ trạng thái sleep hoặc wait và ném ra ngoại lệ InterruptedException. Nếu thread không ở trong trạng thái sleep hoặc wait, việc gọi phương thức interrupt() thực hiện hành vi bình thường và không làm gián đoạn thread nhưng đặt cờ interrupt thành true.

# isInterrupted() : kiểm tra nếu thread đã bị ngắt.

# interrupted() : kiểm tra nếu thread hiện tại đã bị ngắt.

**Phần V: PHÂN BIỆT GIỮA MÔ HÌNH XỬ LÝ DÙNG MULTITHREAD VÀ NHIỀU PROCESS**



## Process và Thread

## -Process là quá trình hoạt động của một ứng dụng. Điều đó nghĩa là gì? Có thể lấy một ví dụ như sau, khi bạn click đúp chuột vào biểu tượng MS Word, một process chạy ứng dụng Word được khởi tạo. Thread là một bước điều hành bên trong một process. Một process dĩ nhiên có thể chứa nhiều thread bên trong nó. Khi chúng ta chạy ứng dụng Word, hệ điều hành tạo ra một process  và bắt đầu chạy các thread chính của process đó.

# -Điểm quan trọng nhất cần chú ý là một thread có thể làm bất cứ nhiệm vụ gì một process có thể làm. Tuy nhiên, vì một process có thể chứa nhiều thread, mỗi thread có thể coi như là một process nhỏ. Vậy, điểm khác biệt mấu chốt giữa thread và process là công việc mỗi cái thường phải làm.

# -Một điểm khác biệt nữa đó là nhiều thread nằm trong cùng một process dùng một không gian bộ nhớ giống nhau, trong khi process thì không. Điều này cho phép các thread đọc và viết cùng một kiểu cấu trúc và dữ liệu, giao tiếp dễ dàng giữa các thread với nhau. Giao thức giữa các process, hay còn gọi là IPC (inter-process communication) thì tương đối phức tạp bởi các dữ liệu có tính tập trung sâu hơn.

## Đa luồng (Mutilple threading)

# -Thread, dĩ nhiên cho phép chạy đa luồng. Minh hoạ dễ hiểu cho tính ưu việt của sự đa luồng là trình xử lý Word có thể vừa in tài liệu sử dụng một thread nền, vừa cùng lúc chạy một thread khác nhận dữ liệu vào từ người dùng để gõ một văn bản mới.

# -Nếu bạn đang làm việc với ứng dụng sử dụng một thread duy nhất, mà ứng dụng đó chỉ có thể làm một việc đơn lẻ vào một thời điểm – thì việc vừa in văn bản vừa tiếp nhận thông tin người dùng là bất khả thi trong ứng dụng đơn luồng này.

# -Mỗi một process có một vùng nhớ riêng của chúng, song các thread trong cùng một process thì dùng chung địa chỉ nhớ. Và các thread cũng dùng chung bất cứ tài nguyên nào nằm trong process đấy. Có nghĩa là rất dễ để chia sẻ dữ liệu giữa các thread, nhưng cũng rất dễ làm thread này nhảy sang thread khác, dẫn đến một số kết quả tồi tệ.

# -Các chương trình đa luồng cần được lập trình cẩn thận để tránh việc nhảy cóc như trên xảy ra.  Đoạn mã lệnh thay đổi cấu trúc dữ liệu chia sẻ giữa các đa luồng này được gọi là những đoạn quan trọng. Khi một đoạn quan trọng đang chạy ở thread này, cần đảm bảo không thread khác nào được phép sử dụng đoạn quan trọng này. Đây là quy đình đồng bộ hoá, nhằm tránh không bị dừng chương trình một cách bất ngờ ở đây. Và đó cũng là lý do tại sao đa luồng đòi hỏi lập trình một cách rất cẩn thận.

# -Nhìn chung, môi trường chuyển đổi của các thread đỡ tốn kém hơn so với giữa các  process. Đặt biệt là overhead (chi phí giao thức) giữa các thread đặc biệt thấp so với các process.

# Sau đây là bảng tổng kết sự khác nhau giữa thread và process:

# Để tạo nhiều thread thì dễ dàng hơn so với process vì chúng không cần các địa chỉ nhớ riêng rẽ.

# Việc chạy đa luồng cần được lập trình một cách chi tiết vì các thread chia sẻ các cấu trúc chung mà chỉ sử dụng được bởi từng thread vào mỗi thời điểm. Khác với thread, các process không dùng chung địa chỉ nhớ.

# Thread được xếp hạng “nhẹ cân” bởi vì chúng sử dụng ít tài nguyên hơn so với các process.

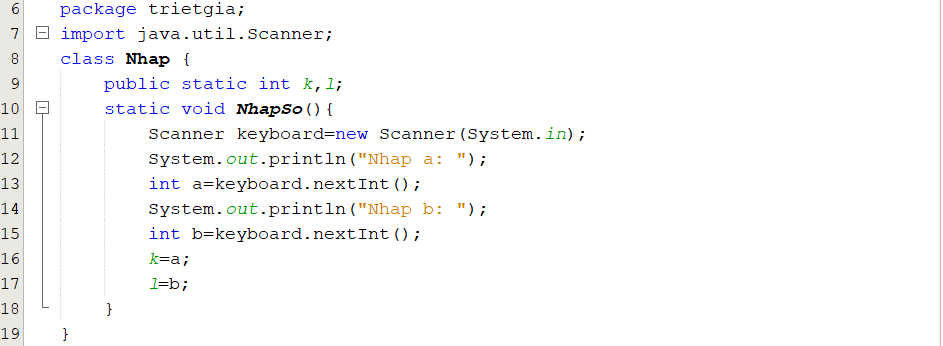
# Các process chạy độc lập với nhau. Các thread thì sử dụng chung các địa chỉ nhớ liên kết với nhau, vì thế cần thận trọng tránh việc thread này nhảy sang thread khác. (Điều đã được nhắc đến trong ý thứ 2 vừa trên)

# Một process có thể chứa nhiều thread.

**PHẦN VI: BÀI THỰC HÀNH**

Bài 14: Cho người dùng nhập vào 2 số nguyên dương a và b, sau đó dùng n thread để tìm kiếm các số nguyên tố nằm trong khoảng 2 số đã nhập. Mỗi lần tìm được một số thì sẽ tăng biến chung **number** (khởi động là 0)lên 1.

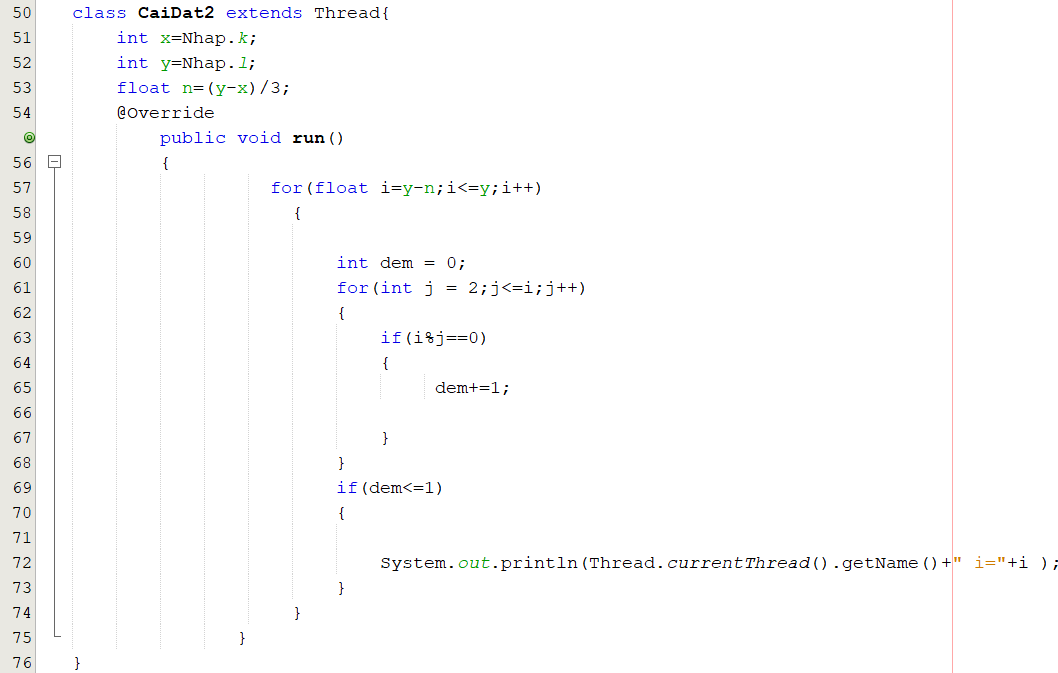
* Nhập a và b từ bàn phím:



* Sử dụng 3 thread để tìm kiếm các số nguyên tố trong khoảng a đến b:
* Thread1 từ a đến a+n; với n=(b-a)/3:



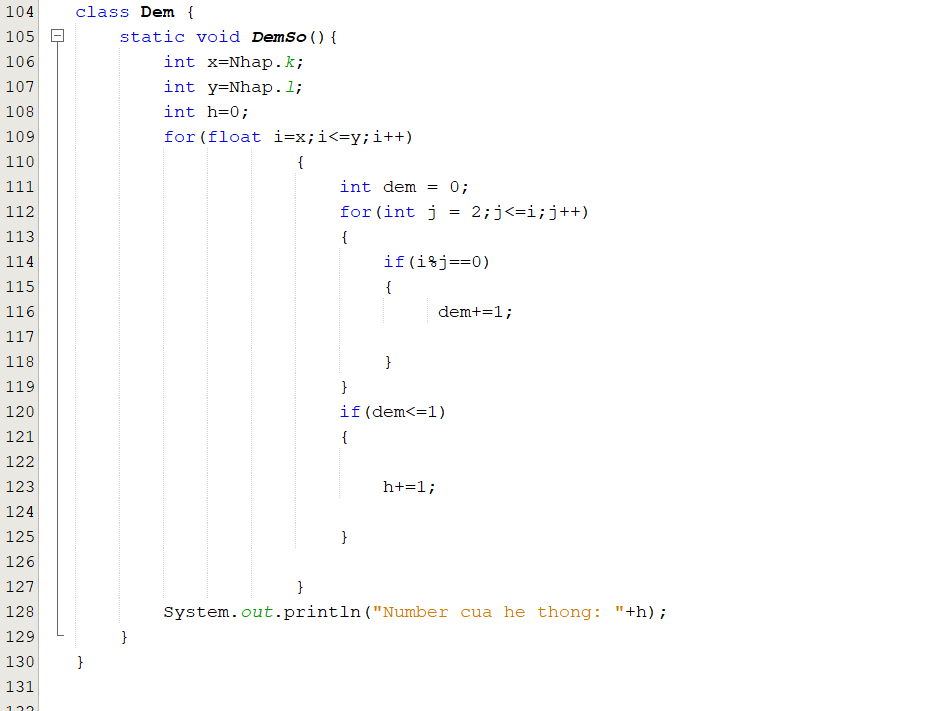
* Thread2 từ b-n đến b:



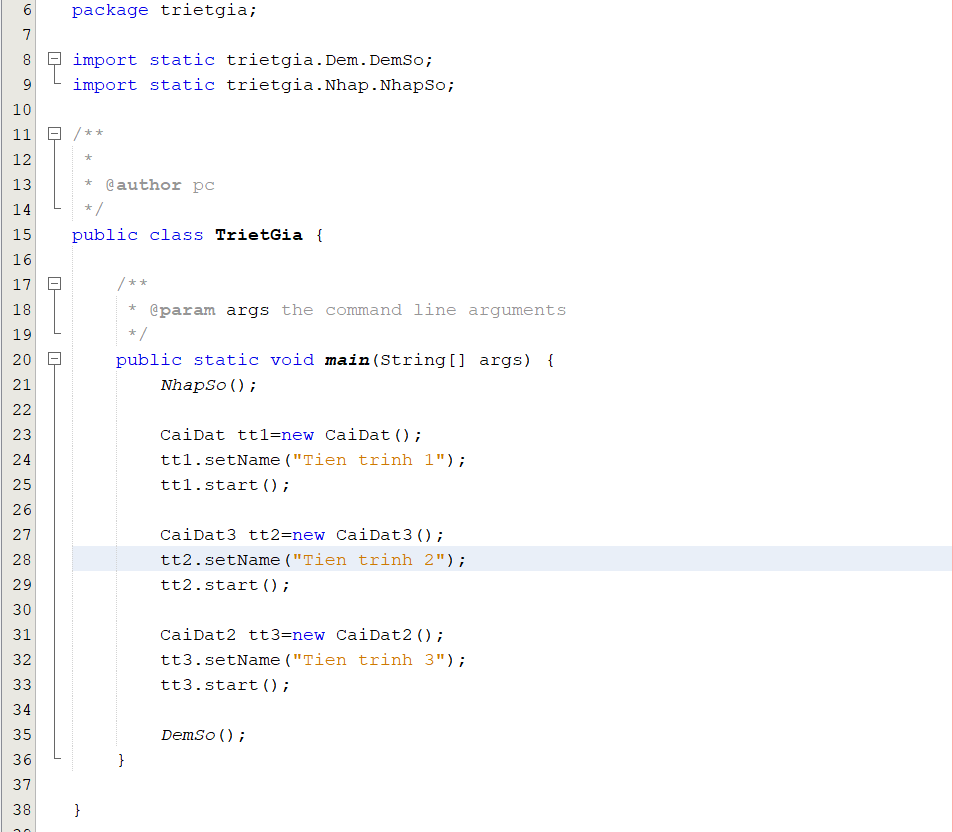
* Thread3 từ a+n đến b-n



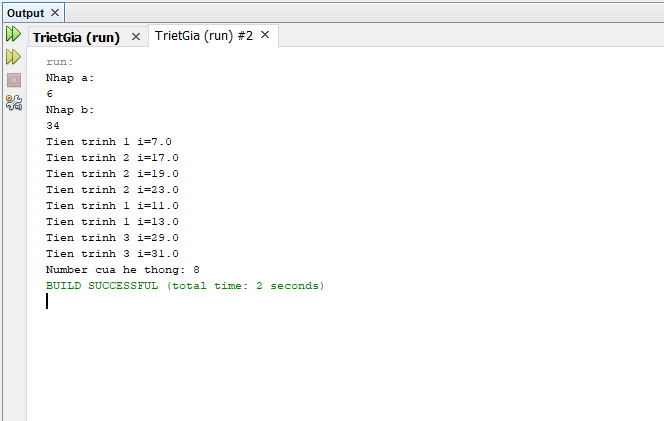
* Đếm số Number mỗi khi tìm thấy số nguyên tố:



* Đưa ra hàm main để chạy chương trình:



* Kết quả khi chạy thử với hai số a,b ngẩu nhiên:



**PHẦN VII: TÀI LIỆU THAM KHẢO**

* <https://www.google.com.vn/search?q=thread+l%C3%A0+g%C3%AC&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjfv8KAssXeAhULv7wKHWFECSUQ_AUIDigB&biw=1536&bih=750#imgrc=_>
* **http://voer.edu.vn/m/da-luong-multithreading/303a3cc3**
* <https://vietjack.com/cplusplus/da_luong_multithread_trong_cplusplus.jsp>
* <https://it.die.vn/p/process-qua-trinh/>
* **https://www.youtube.com/watch?v=8ycU4KAgBGw&t=3713s**