**Sử dụng CountDownLatch trong Java**

**1. CountDownLatch**

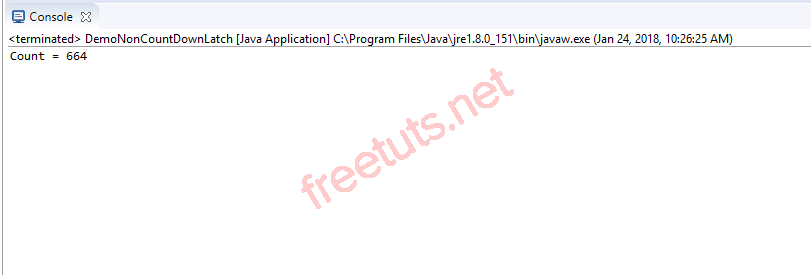
CountDownLatch là một lớp trong Java chuyên dùng để đếm. Tại sao Java lại tạo ra một lớp riêng để đếm trong khi chúng ta hoàn toàn có thể thực hiện việc đếm bằng cách khác? Bởi vì trong lập trình đa luồng, việc đếm này sẽ rất khó và phức tạp, thường khi muốn đếm thì chúng ta phải sử dụng đến đồng bộ phương thức nếu không việc đếm sẽ không chính xác, vì lúc này trong hệ thống các luồng đang được xử lý song song và các biến sẽ được sử dụng một cách không tuần tự lúc đó kết quả đếm sẽ không chính xác và không như chúng ta mong muốn.

Để hiểu rõ hơn các bạn theo dõi ví dụ sau:

**DemoNonCountDownLatch.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | package countdownlatch;    public class DemoNonCountDownLatch {        private int count = 2000;        public void dem() {          Thread thread1 = new Thread(new Runnable() {              public void run() {                  for (int i = 0; i < 1000; i++) {                      count--;                  }              }          });            Thread thread2 = new Thread(new Runnable() {              public void run() {                  for (int i = 0; i < 1000; i++) {                      count--;                  }              }          });            thread1.start();          thread2.start();            try {              thread1.join();              thread2.join();          } catch (InterruptedException ignored) {          }            System.out.println("Count = " + count);      }        public static void main(String[] args) {          DemoNonCountDownLatch demoNonCountDownLatch = new DemoNonCountDownLatch();          demoNonCountDownLatch.dem();      }  } |

Kết quả sau khi biên dịch chương trình:



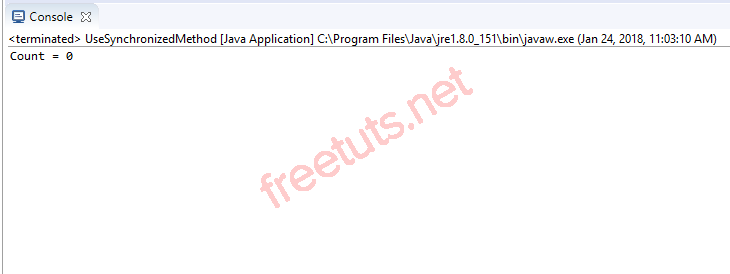
Với đoạn code trên thì khi thực thi chương trình, kết quả mà chúng ta mong muốn là giá trị của biến count sẽ bằng 0, nhưng vì thread1 và thread2 lúc này là 2 luồng chạy song song nên kết quả biến count sau mỗi lần biên dịch sẽ khác nhau. Vì biến count được cả hai thread1 và thread2 cùng đồng thời tác động và thay đổi giá trị nên sẽ xảy ra trường hợp thread1 chưa xử lý xong (*giảm count đi 1*) thì thread2 đã thực hiện tác động tới biến count này nên giá trị của count sẽ thay đổi và không theo ý muốn của chúng ta.

Vì vậy, để giải quyết trường hợp này, chúng ta có thể sử dụng từ khóa synchronized để đảm bảo rằng việc giảm biến count đi 1 sẽ được xử lý một cách tuần tự, tức là tại 1 thời điểm chỉ có duy nhất một Thread được thao tác với biến count:

**UseSynchronizedMethod.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | package countdownlatch;    import java.util.concurrent.CountDownLatch;    public class DemoCountDownLatch {        public CountDownLatch count = new CountDownLatch(2000);        public void doWork() {          Thread thread1 = new Thread(new Runnable() {              public void run() {                  for (int i = 0; i < 1000; i++) {                      count.countDown();  // giảm giá trị của biến count                  }              }          });            Thread thread2 = new Thread(new Runnable() {              public void run() {                  for (int i = 0; i < 1000; i++) {                      count.countDown();                  }              }          });            thread1.start();          thread2.start();            try {              count.await();          } catch (InterruptedException ignored) {            }          System.out.println("Count = " + count.getCount());      }        public static void main(String[] args) {          DemoCountDownLatch demo = new DemoCountDownLatch();          demo.doWork();      }    } |

Kết quả sau khi biên dịch chương trình:



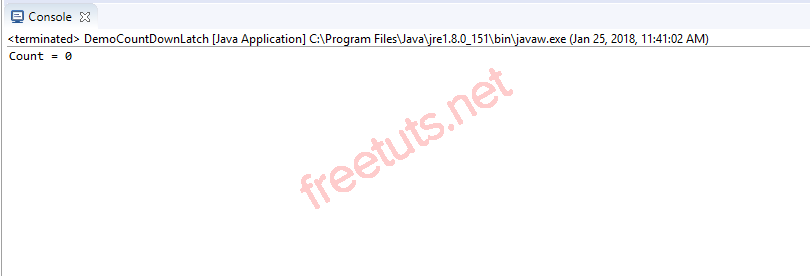
Trong đoạn code trên, từ khóa synchronized ở phương thức giamCount() đã bảo đảm rằng tại một thời điểm chỉ có duy nhất 1 Thread được gọi vào hàm để xử lý.

Ngoài ra, như nội dung chính của phần này đó là chúng ta có thể dùng CountDownLatch để giải quyết vấn đề này:

**DemoCountDownLatch.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | package countdownlatch;    import java.util.concurrent.CountDownLatch;    public class DemoCountDownLatch {        public CountDownLatch count = new CountDownLatch(2000);        public void doWork() {          Thread thread1 = new Thread(new Runnable() {              public void run() {                  for (int i = 0; i < 1000; i++) {                      count.countDown();  // giảm giá trị của biến count đi 1                  }              }          });            Thread thread2 = new Thread(new Runnable() {              public void run() {                  for (int i = 0; i < 1000; i++) {                      count.countDown();                  }              }          });            thread1.start();          thread2.start();            try {              count.await();          } catch (InterruptedException ignored) {            }          System.out.println("Count = " + count.getCount());      }        public static void main(String[] args) {          DemoCountDownLatch demo = new DemoCountDownLatch();          demo.doWork();      }    } |

Kết quả sau khi biên dịch chương trình



Giải thích hoạt động của chương trình trên:

Đoạn code public CountDownLatch count = new CountDownLatch(2000); sẽ khởi tạo một đối tượng CountDownLatch có tên là count với giá trị là 2000. Còn để giảm dần giá trị của count thì lớp CountDownLatch cung cấp cho chúng ta phương thức countDown(): count.countDown();. Ngoài ra, chúng ta còn thấy dòng code count.await();, phương thức await() sẽ đợt tất cả các tiến trình xử lý của CountDownLatch hoàn thành (*tương tự như phương thức join() trong Thread*).