**Hướng dẫn cách sử dụng sleep() và join()**

**1. sleep()**

Phương thức sleep() của lớp Thread được sử dụng để tạm ngưng một Thread đang hoạt động trong một khoảng thời gian nhất định. Để sử dụng phương thức sleep(), Java cung cấp cho chúng ta 2 cú pháp như sau:

**Cú pháp**

Thread.sleep(long millis);  // tạm dừng Thread với khoảng thời gian dừng tính bằng millisecond

Thread.sleep(long millis, int nanos);   // tạm dừng Thread với khoảng thời gian dừng tính bằng thời gian millis (tính bằng milliseconds) cộng với thời gian nanos (tính bằng nanoseconds và nằm trong khoảng từ 0-999999)

, trong đó millis là khoảng thời gian tính bằng milliseconds và nanos là khoảng thời gian tính bằng nanoseconds.

**Lưu ý:** Trong khi sử dụng phương thức sleep() này thì trình biên dịch sẽ bắt buộc chúng ta sinh ra đoạn try...catch bao bọc bên ngoài. Tạm thời các bạn đừng để ý đến nó mà chỉ cần hiểu đây là điều bắt buộc khi muốn sử dụng sleep(). Chi tiết về try...catch tôi sẽ trình bày trong chương sau.

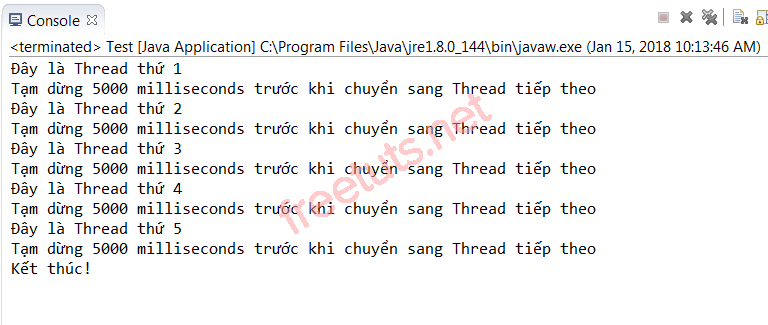
Để minh họa cách sử dụng phương thức sleep(), tôi sẽ đưa ra ví dụ minh họa đơn giản như sau:

**DemoSleep.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | package phuongthucsleep;    public class DemoSleep extends Thread {        public void run() {          super.run();          for (int i = 1; i <= 5; i++) {              System.out.println("Đây là Thread thứ " + i);              System.out.println("Tạm dừng 5000 milliseconds trước khi chuyển sang Thread tiếp theo");              if (i == 5) {                  System.out.println("Kết thúc!");              }              try {                  Thread.sleep(5000);              } catch (InterruptedException e) {                  // TODO Auto-generated catch block                  e.printStackTrace();              }   // tạm dừng 5000 milliseconds trước khi in ra câu tiếp theo          }      }    } |

**Test.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | package phuongthucsleep;    public class Test {        public static void main(String[] args) {          DemoSleep t1 = new DemoSleep();          t1.start();      }    } |

Kết quả sau khi biên dịch chương trình: 

**2. join()**

Phương thức join() được sử dụng để đảm bảo cho quá trình thực thi của Thread đang chạy không bị gián đoạn bởi các Thread khác. Nói một cách khác, nếu một Thread đang trong quá trình được thực thi thì các Thread khác sẽ phải chờ đợi cho đến khi Thread đó thực thi xong. join() được sử dụng khi trong một chương trình Java có nhiều hơn một Thread và chúng ta cần đảm bảo các Thread thực thi và kết thúc đúng theo thứ tự mà chúng đã được khởi tạo.

Để hiểu hơn vai trò của join(), các bạn hãy theo dõi 2 ví dụ sau: Ví dụ thứ nhất là ví dụ không sử dụng join() và ví dụ thứ hai là ví dụ có sử dụng phương thức join():

**Ví dụ không sử dụng join()**

**NoneJoinThread.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package phuongthucjoin;    public class NoneJoinThread extends Thread {        public void run() {          super.run();          System.out.println(Thread.currentThread().getName() + " đang chạy.");          try {              Thread.sleep(5000);          } catch (InterruptedException e) {              e.printStackTrace();          }          System.out.println("Kết thúc " + Thread.currentThread().getName());      }    } |

**TestNoneJoinThread.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | package phuongthucjoin;    public class TestNoneJoinThread {        public static void main(String[] args) {          NoneJoinThread thread1 = new NoneJoinThread();          thread1.setName("Thread 1");          NoneJoinThread thread2 = new NoneJoinThread();          thread2.setName("Thread 2");          NoneJoinThread thread3 = new NoneJoinThread();          thread3.setName("Thread 3");            thread1.start();          thread2.start();          thread3.start();      }    } |

Kết quả sau khi biên dịch chương trình:



**Ví dụ sử dụng join()**

**JoinThread.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package phuongthucjoin;    public class JoinThread extends Thread {        public void run() {          super.run();          System.out.println(Thread.currentThread().getName() + " đang chạy.");          try {              Thread.sleep(5000);          } catch (InterruptedException e) {              e.printStackTrace();          }          System.out.println("Kết thúc " + Thread.currentThread().getName());      }    } |

**TestJoinThread.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 | package phuongthucjoin;    public class TestJoinThread {        public static void main(String[] args) {          JoinThread thread1 = new JoinThread();          thread1.setName("Thread 1");          thread1.start();    // khởi chạy thread 1          try {              thread1.join();          } catch (InterruptedException e) {              e.printStackTrace();          }            JoinThread thread2 = new JoinThread();          thread2.setName("Thread 2");          thread2.start();    // khởi chạy thread2          try {              thread2.join();          } catch (InterruptedException e) {              e.printStackTrace();          }            JoinThread thread3 = new JoinThread();          thread3.setName("Thread 3");          thread3.start();    // khởi chạy thread3          try {              thread3.join();          } catch (InterruptedException e) {              e.printStackTrace();          }      }    } |

Kết quả sau khi biên dịch chương trình:



**Giải thích hoạt động của hai chương trình trên**

Trong ví dụ 1 (*ví dụ không sử dụng join()*) thì chúng ta thấy rằng 3 thread1, thread2, thread3 được khởi chạy theo đúng thứ tự thread1 → thread2 → thread3, nhưng khi kết thúc thì 3 Thread này không theo thứ tự thread1 → thread2 → thread3. Ở mỗi thời điểm chạy chương trình có thể nhận được các kết quả khác nhau.

Còn khi chúng ta sử dụng join() (*trong ví dụ 2*) thì khi thread1 đang thực thi thì thread2 và thread3 chưa khởi chạy ngay cho dù có lệnh thread2.start() và thread3.start() mà phải đợi cho đến khi thread1 thực thi xong. Tương tự khi thread2 đang chạy thì thread3 vẫn phải đợi cho đến khi thread2 thực thi xong. Do đó mà kết quả in ra màn hình theo đúng thứ tự thread1 → thread2 → thread3 mà chúng đã được khởi chạy.