Đa luồng trong Java

Java là một *ngôn ngữ chương trình đa luồng (multithreaded)*, nghĩa là chúng ta có thể phát triển chương trình đa luồng sử dụng Java. Một chương trình đa luồng chứa hai hoặc nhiều phần mà có thể chạy đồng thời và mỗi phần có thể xử lý tác vụ khác nhau tại cùng một thời điểm, để sử dụng tốt nhất các nguồn có sẵn, đặc biệt khi máy tính của bạn có nhiều CPU.

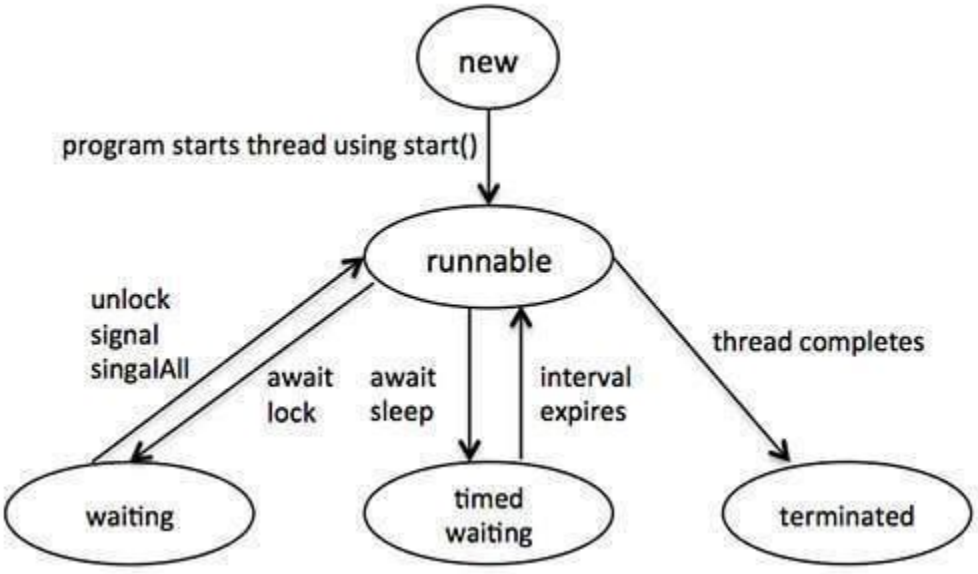
Theo định nghĩa, đa nhiệm (multitasking) là khi nhiều tiến trình chia sẻ nguồn xử lý chung ví dụ như một CPU. Đa luồng kế thừa ý tưởng của đa nhiệm trong các ứng dụng để bạn có thể chia nhỏ các hoạt động riêng biệt bên trong một ứng dụng đơn thành các luồng (thread) riêng lẻ. Mỗi một thread có thể chạy song song. OS phân chia thời gian xử lý không chỉ trong các ứng dụng khác nhau, mà còn trong mỗi luồng bên trong một ứng dụng.

Đa luồng cho bạn khả năng viết một chương trình mà có nhiều hoạt động có thể phát sinh đồng thời.

Để hiểu sâu hơn các khái niệm được trình bày trong chương này, mời bạn tham khảo loạt bài: **Ví dụ về Thread trong Java**.

Vòng đời của một Thread trong Java

Một thread đi qua các giai đoạn khác nhau trong vòng đời của nó. Ví dụ, một thread được sinh ra, được bắt đầu, chạy và sau đó hủy. Sơ đồ sau biểu diễn một vòng đời đầy đủ của một thread.



Các giai đoạn trên được giải thích như sau:

- **New:** Một thread mới bắt đầu vòng đời của nó trong trạng thái new. Nó tồn tại trong trạng thái này tới khi chương trình bắt đầu thread này. Nó cũng được xem như là một thread mới sinh.

- **Runnable:** Sau khi một thread mới sinh ra được bắt đầu, thread trở thành runnable. Một thread trong trạng thái này được xem như đang thực hiện tác vụ của nó.

- **Waiting:** Đôi khi, một thread quá độ qua trạng thái waiting trong khi thread đợi cho thread khác thực hiện một tác vụ. Một thread chuyển về trạng thái runnable chỉ khi thread khác ra hiệu cho thread đang đợi để tiếp tục thực thi.

- **Timed waiting:** Một thread trong trạng thái runnable có thể đi vào trạng thái timed waiting trong một khoảng thời gian nào đó. Một thread trong trạng thái này chuyển về trạng thái runnable khi khoảng thời gian đó kết thúc hoặc khi sự kiện nó đang đợi xuất hiện.

- **Terminated:** Một thread trong trạng thái runnable có thể đi vào trạng thái terminated khi nó hoàn thành tác vụ của nó hoặc nó chấm dứt.

Quyền ưu tiên của thread trong Java

Mỗi thread trong Java có một quyền ưu tiên mà giúp hệ điều hành xác định thứ tự thread nào được ghi lịch trình.

Quyền ưu tiên của thread trong Java là một dãy giữa MIN_PRIORITY (hằng số 1) và MAX_PRIORITY (hằng số 10). Theo mặc định, mỗi thread được cung cấp một quyền ưu tiên NORM_ PRIORITY (hằng số 5).

Các thread với quyền ưu tiên cao hơn là quan trọng hơn với một chương trình và nên được cấp phát thời gian bộ vi xử lý trước các thread có quyền ưu tiên thấp hơn. Tuy nhiên, các quyền ưu tiên của thread bảo đảm thứ tự trong đó các thread thực thi và phụ thuộc rất nhiều vào platform.

Tạo thread bởi triển khai Runnable Interface trong Java

Nếu class của bạn có ý định để được thực thi như một thread, thì bạn có thể đạt được điều này bởi triển khai **Runnable** Interface. Bạn sẽ cần theo 3 bước cơ bản sau:

Bước 1:

Tại bước này, bạn cần triển khai một phương thức run() được cung cấp bởi **Runnable**Interface. Phương thức này cung cấp điểm đi vào (entry) cho thread. Sau đây là cú pháp đơn giản của phương thức run() trong Java:

```
public void run( )
```

Bước 2:

Tại đây, bạn sẽ thuyết minh một đối tượng **Thread**, sử dụng constructor sau trong Java:

```
Thread(Runnable threadObj, String threadName);
```

Ở đây, *threadObj* là một instance của một lớp mà triển khai Runnable Interface và*threadName* là tên được cung cấp cho thread mới.

Bước 3

Khi đối tượng Thread được tạo, bạn có thể bắt đầu nó bởi gọi phương thức **start()** trong Java, mà thực thi một triệu hồi tới phương thức run(). Sau đây là cú pháp đơn giản của phương thức start() trong Java:

```
void start( );
```

Ví dụ:

Ví dụ sau để tạo một thread mới và bắt đầu chạy:

```
class RunnableDemo implements Runnable {
   private Thread t;
   private String threadName;


   RunnableDemo( String name){
       threadName = name;
       System.out.println("Creating " +  threadName );
   }
   public void run() {
      System.out.println("Running " +  threadName );
      try {
         for(int i = 4; i > 0; i--) {
```

```
            System.out.println("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("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }

}

public class TestThread {
   public static void main(String args[]) {

      RunnableDemo R1 = new RunnableDemo( "Thread-1");
      R1.start();

      RunnableDemo R2 = new RunnableDemo( "Thread-2");
      R2.start();
   }
}
```

Nó sẽ cho kết quả:

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

Tạo Thread bởi kế thừa lớp Thread trong Java

Cách thứ hai để tạo một thread mới là kế thừa lớp **Thread** theo hai bước đơn giản. Cách tiếp cận này linh động hơn trong việc xử lý nhiều thread được tạo bởi sử dụng các phương thức có sẵn trong lớp Thread trong Java.

Bước 1

Bạn sẽ cần override phương thức **run()** có sẵn trong lớp Thread. Phương thức này cung cấp điểm entry cho thread. Sau đây là cú pháp đơn giản của phương thức run() trong Java:

```
public void run( )
```

Bước 2

Khi đối tượng Thread được tạo, bạn có thể bắt đầu nó bởi gọi phương thức **start()** trong Java, mà thực thi một triệu hồi tới phương thức run(). Sau đây là cú pháp đơn giản của phương thức start() trong Java:

```
void start( );
```

Ví dụ:

Đây là ví dụ ở trên được viết lại để kế thừa lớp Thread:

```
class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;

   ThreadDemo( String name){
       threadName = name;
       System.out.println("Creating " +  threadName );
   }
   public void run() {
      System.out.println("Running " +  threadName );
      try {
         for(int i = 4; i > 0; i--) {
            System.out.println("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("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
```

```
      }
   }

}

public class TestThread {
   public static void main(String args[]) {

      ThreadDemo T1 = new ThreadDemo( "Thread-1");
      T1.start();

      ThreadDemo T2 = new ThreadDemo( "Thread-2");
      T2.start();
   }
}
```

Nó sẽ cho kết quả sau:

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

Các phương thức của lớp Thread trong Java

Bảng dưới đây liệt kê các phương thức quan trọng có sẵn trong lớp Thread của Java:

STT	Phương thức và Miêu tả
1	**public void start()** Bắt đầu thread trong một path riêng rẽ, sau đó triệu hồi phương thức run() trên đối tượng Thread này
2	**public void run()** Nếu đối tượng Thread này được thuyết minh bởi sử dụng một đối tượng Runnable, phương thức run() được triệu hồi trên đối tượng Runnable đó
3	**public final void setName(String name)** Thay đổi tên của đối tượng Thread. Cũng có một phương thức getName() để thu nhận tên này
4	**public final void setPriority(int priority)** Thiết lập quyền ưu tiên của đối tượng Thread này. Giá trị có thể có nằm trong khoảng từ 1 tới 10
5	**public final void setDaemon(boolean on)** Một tham số true chứng tỏ Thread này như là một Daemon thread.
6	**public final void join(long millisec)** Thread hiện tại triệu hồi phương thức này trên thread thứ hai, làm cho Thread hiện tại block tới khi thread thứ hai kết thúc hoặc sau một số lượng mili giây đã xác định
7	**public void interrupt()** Ngắt thread này, làm cho nó tiếp tục thực thi nếu nó bị block vì bất cứ lý do gì

8	**public final boolean isAlive()** Trả về true nếu thread này là alive, mà là bất cứ thời gian nào sau khi thread này đã được bắt đầu nhưng trước khi nó run tới khi kết thúc

Các phương thức trên được triệu hồi trên một đối tượng Thread cụ thể. Các phương thức sau trong lớp Thread là static. Triệu hồi một trong các phương thức này thực hiện hoạt động trên thread đang chạy hiện tại.

STT	Phương thức và Miêu tả
1	**public static void yield()** Làm cho thread đang chạy hiện tại chuyển tới bất kỳ thread nào khác có cùng quyền ưu tiên mà đang đợi để được ghi lịch trình
2	**public static void sleep(long millisec)** Làm cho thread đang chạy hiện tại block trong ít nhất một số lượng mili giây đã xác định
3	**public static boolean holdsLock(Object x)** Trả về true nếu thread giữ lock trên Object đã cho
4	**public static Thread currentThread()** Trả về một tham chiếu tới thread đang chạy hiện tại, mà là thread mà triệu hồi phương thức này
5	**public static void dumpStack()** In ra stack trace cho thread đang chạy hiện tại. Nó rất hữu ích khi debugging một ứng dụng đa luồng

Ví dụ:

Chương trình *ThreadClassDemo* sau minh họa một số phương thức này trong lớp Thread. Bạn để ý lớp *DisplayMessage* triển khai **Runnable**:

```
// File Name : DisplayMessage.java
// Create a thread to implement Runnable
public class DisplayMessage implements Runnable
{
   private String message;
   public DisplayMessage(String message)
   {
      this.message = message;
   }
   public void run()
   {
      while(true)
      {
         System.out.println(message);
      }
   }
}
```

Sau đây là lớp khác mà kế thừa lớp Thread:

```
// File Name : GuessANumber.java
// Create a thread to extentd Thread
public class GuessANumber extends Thread
{
   private int number;
   public GuessANumber(int number)
   {
      this.number = number;
   }
   public void run()
```

```
   {
      int counter = 0;
      int guess = 0;
      do
      {
          guess = (int) (Math.random() * 100 + 1);
          System.out.println(this.getName()
                      + " guesses " + guess);
          counter++;
      }while(guess != number);
      System.out.println("** Correct! " + this.getName()
                      + " in " + counter + " guesses.**");
   }
}
```

Dưới đây là chương trình chính mà sử dụng các lớp được định nghĩa ở trên:

```
// File Name : ThreadClassDemo.java
public class ThreadClassDemo
{
   public static void main(String [] args)
   {
      Runnable hello = new DisplayMessage("Hello");
      Thread thread1 = new Thread(hello);
      thread1.setDaemon(true);
      thread1.setName("hello");
      System.out.println("Starting hello thread...");
      thread1.start();

      Runnable bye = new DisplayMessage("Goodbye");
      Thread thread2 = new Thread(bye);
      thread2.setPriority(Thread.MIN_PRIORITY);
      thread2.setDaemon(true);
      System.out.println("Starting goodbye thread...");
```

```
      thread2.start();


      System.out.println("Starting thread3...");
      Thread thread3 = new GuessANumber(27);
      thread3.start();
      try
      {
         thread3.join();
      }catch(InterruptedException e)
      {
         System.out.println("Thread interrupted.");
      }
      System.out.println("Starting thread4...");
      Thread thread4 = new GuessANumber(75);


           thread4.start();
      System.out.println("main() is ending...");
   }
}
```

Nó sẽ cho kết quả sau. Bạn chạy thử ví dụ này nhiều lần và sẽ thấy nó cho kết quả khác nhau trong mỗi lần chạy.

```
Starting hello thread...
Starting goodbye thread...
Hello
Hello
Hello
Hello
Hello
Hello
Goodbye
Goodbye
Goodbye
```

```
Goodbye
Goodbye
.......
```

Một số khái niệm về Đa luồng chủ yếu trong Java

Trong khi lập trình đa luồng trong Java, bạn sẽ thấy rất tiện lợi khi bạn biết về các khái niệm sau:

- Thread Synchronization trong Java
- InterThread Communication trong Java
- Thread deadlock trong Java
- Thread Control trong Java

Để hiểu sâu hơn các khái niệm được trình bày trong chương này, mời bạn tham khảo loạt bài: **Ví dụ về Thread trong Java**.