JAVA.LANG.THREADGROUP.ENUMERATE METHOD

Description

The **java.lang.ThreadGroup.enumerate*Thread[]list, booleanrecurse*** method copies into the specified array every active thread in this thread group. If the **recurse** flag is true, references to every active thread in this thread's subgroups are also included.

If the array is too short to hold all the threads, the extra threads are silently ignored.

Declaration

Following is the declaration for **java.lang.ThreadGroup.enumerate** method

```
public int enumerate(Thread[] list, boolean recurse)
```

Parameters

- **list** -- This is an array into which to place the list of threads.
- **recurse** -- This is a flag indicating whether also to include threads in thread groups that are subgroups of this thread group.

Return Value

This method returns the number of threads placed into the array.

Exception

- **SecurityException** -- if the current thread does not have permission to enumerate this thread group.

Example

The following example shows the usage of java.lang.ThreadGroup.enumerate method.

```
package com.tutorialspoint;

import java.lang.*;

public class ThreadGroupDemo implements Runnable
{
   public static void main(String[] args) {
      ThreadGroupDemo tg = new ThreadGroupDemo();
      tg.func();
   }

   public void func() {
      try {
         // create a parent ThreadGroup
         ThreadGroup pGroup = new ThreadGroup("Parent ThreadGroup");

         // create a child ThreadGroup for parent ThreadGroup
         ThreadGroup cGroup = new ThreadGroup(pGroup, "Child ThreadGroup");

         // create a thread
         Thread t1 = new Thread(pGroup, this);
         System.out.println("Starting " + t1.getName() + "...");
         t1.start();

         // create another thread
         Thread t2 = new Thread(cGroup, this);
         System.out.println("Starting " + t2.getName() + "...");
```

```
         t2.start();

         // returns the number of threads put into the array with flag as true
         Thread[] list = new Thread[pGroup.activeCount()];
         int count = pGroup.enumerate(list, true);
         for (int i = 0; i < count; i++) {
            System.out.println("Thread " + list[i].getName() + " found");
         }

         // block until the other threads finish
         t1.join();
         t2.join();
      }
      catch (InterruptedException ex) {
         System.out.println(ex.toString());
      }
   }

   // implements run()
   public void run() {

      for(int i = 0; i < 1000; i++) {
         i++;
      }
      System.out.println(Thread.currentThread().getName() +
      " finished executing.");
   }
}
```

Let us compile and run the above program, this will produce the following result:

```
Starting Thread-0...
Starting Thread-1...
Thread Thread-0 found
Thread Thread-1 found
Thread-0 finished executing.
Thread-1 finished executing.
```