**Introduction to Suspend Resume Thread**

When the **sleep()** method time is over, the thread becomes implicitly active. **sleep()** method is preferable when the inactive time is known earlier. Sometimes, the inactive time or blocked time may not be known to the programmer earlier; to come to the task here comes **suspend()** method. The suspended thread will be in blocked state until **resume()** method is called on it. These methods are deprecated, as when not used with precautions, the thread locks, if held, are kept in inconsistent state or may lead to deadlocks.

**Note:** You must have noticed, in the earlier **sleep()** method, that the thread in blocked state retains all its state. That is, attribute values remains unchanged by the time it comes into runnable state.

**Suspend Resume Thread: Program explaining the usage of suspend() and resume() methods**

public class SRDemo extends Thread

{

public static void main( String args[ ] )

{

SRDemo srd1 = new SRDemo();

SRDemo srd2 = new SRDemo();

srd1.setName("First");

srd2.setName("Second");

srd1.start();

srd2.start();

try

{

Thread.sleep( 1000 );

srd1.suspend();

System.out.println("Suspending thread First");

Thread.sleep( 1000 );

srd1.resume();

System.out.println("Resuming thread First");

Thread.sleep(1000);

srd2.suspend();

System.out.println("Suspending thread Second");

Thread.sleep(1000);

srd2.resume();

System.out.println("Resuming thread Second");

}

catch(InterruptedException e)

{

e.printStackTrace();

}}

public void run()

{

try

{

for(int i=0; i<7; i++)

{

Thread.sleep(500);

System.out.println( this.getName() + ":  " + i );

}}

catch(InterruptedException e)

{

e.printStackTrace();

}

}

}

|  |  |
| --- | --- |
|  |  |

