
Class java.lang.Thread

```
java.lang.Object
   |
   +----java.lang.Thread
```

---

public class **Thread**
extends Object
implements Runnable

A Thread is a single sequential flow of control within a process. This simply means that while executing within a program, each thread has a beginning, a sequence, a point of execution occurring at any time during runtime of the thread and of course, an ending. Thread objects are the basis for multi–threaded programming. Multi–threaded programming allows a single program to conduct concurrently running threads that perform different tasks.

To create a new thread of execution, declare a new class which is a subclass of Thread and then override the run() method with code that you want executed in this Thread. An instance of the Thread subclass should be created next with a call to the start() method following the instance. The start() method will create the thread and execute the run() method. For example:

```
class PrimeThread extends Thread {
    public void run() {
        // compute primes...
    }
}
```

To start this thread you need to do the following:

```
PrimeThread p = new PrimeThread();
p.start();
...
```

Another way to create a thread is by using the Runnable interface. This way any object that implements the Runnable interface can be run in a thread. For example:

```
class Primes implements Runnable {
    public void run() {
        // compute primes...
    }
```

```
}
```

To start this thread you need to do the following:

```
Primes p = new Primes();
new Thread(p).start();
...
```

The virtual machine runs until all Threads that are not daemon Threads have died. A Thread dies when its run() method returns, or when the stop() method is called.

When a new Thread is created, it inherits the priority and the daemon flag from its parent (i.e.: the Thread that created it).

See Also:
Runnable
Version:
1.43, 09/22/95

Variable Index

o **MAX_PRIORITY**
The maximum priority that a Thread can have.
o **MIN_PRIORITY**
The minimum priority that a Thread can have.
o **NORM_PRIORITY**
The default priority that is assigned to a Thread.

Constructor Index

o **Thread**()
Constructs a new Thread.
o **Thread**(Runnable)
Constructs a new Thread which applies the run() method of the specified target.
o **Thread**(ThreadGroup, Runnable)
Constructs a new Thread in the specified Thread group that applies the run() method of the specified target.
o **Thread**(String)
Constructs a new Thread with the specified name.
o **Thread**(ThreadGroup, String)
Constructs a new Thread in the specified Thread group with the specified name.
o **Thread**(Runnable, String)
Constructs a new Thread with the specified name and applies the run() method of the specified target.
o **Thread**(ThreadGroup, Runnable, String)
Constructs a new Thread in the specified Thread group with the specified name

and applies the run() method of the specified target.

Method Index

- **activeCount**()
  Returns the current number of active Threads in this Thread group.
- **checkAccess**()
  Checks whether the current Thread is allowed to modify this Thread.
- **countStackFrames**()
  Returns the number of stack frames in this Thread.
- **currentThread**()
  Returns a reference to the currently executing Thread object.
- **destroy**()
  Destroy a thread, without any cleanup, i.e.
- **dumpStack**()
  A debugging procedure to print a stack trace for the current Thread.
- **enumerate**(Thread[])
  Copies, into the specified array, references to every active Thread in this Thread's group.
- **getName**()
  Gets and returns this Thread's name.
- **getPriority**()
  Gets and returns the Thread's priority.
- **getThreadGroup**()
  Gets and returns this Thread group.
- **interrupt**()
  Send an interrupt to a thread.
- **interrupted**()
  Ask if you have been interrupted.
- **isAlive**()
  Returns a boolean indicating if the Thread is active.
- **isDaemon**()
  Returns the daemon flag of the Thread.
- **isInterrupted**()
  Ask if another thread has been interrupted.
- **join**(long)
  Waits for this Thread to die.
- **join**(long, int)
  Waits for the Thread to die, with more precise time.
- **join**()
  Waits forever for this Thread to die.
- **resume**()
  Resumes this Thread execution.
- **run**()
  The actual body of this Thread.
- **setDaemon**(boolean)
  Marks this Thread as a daemon Thread or a user Thread.
- **setName**(String)

Sets the Thread's name.

- **setPriority**(int)
  Sets the Thread's priority.
- **sleep**(long)
  Causes the currently executing Thread to sleep for the specified number of milliseconds.
- **sleep**(long, int)
  Sleep, in milliseconds and additional nanosecond.
- **start**()
  Starts this Thread.
- **stop**()
  Stops a Thread by tossing an object.
- **stop**(Object)
  Stops a Thread by tossing an object.
- **suspend**()
  Suspends this Thread's execution.
- **toString**()
  Returns a String representation of the Thread.
- **yield**()
  Causes the currently executing Thread object to yield.

Variables

- **MIN_PRIORITY**

```
public final static int MIN_PRIORITY
```

  The minimum priority that a Thread can have. The most minimal priority is equal to 1.

- **NORM_PRIORITY**

```
public final static int NORM_PRIORITY
```

  The default priority that is assigned to a Thread. The default priority is equal to 5.

- **MAX_PRIORITY**

```
public final static int MAX_PRIORITY
```

  The maximum priority that a Thread can have. The maximal priority value a Thread can have is 10.

Constructors

- **Thread**

```
public Thread()
```

Constructs a new Thread. Threads created this way must have overridden their run() method to actually do anything. An example illustrating this method being used is shown.

```
import java.lang.*;

class plain01 implements Runnable {
  String name;
  plain01() {
    name = null;
  }
  plain01(String s) {
    name = s;
  }
  public void run() {
    if (name == null)
      System.out.println("A new thread created");
    else
      System.out.println("A new thread with name " + name + " created");
  }
}
class threadtest01 {
  public static void main(String args[] ) {
    int failed = 0 ;

   Thread t1 = new Thread();
    if(t1 != null) {
      System.out.println("new Thread() succeed");
    } else {
       System.out.println("new Thread() failed");
       failed++;
    }

}
```

- **Thread**

```
public Thread(Runnable target)
```

Constructs a new Thread which applies the run() method of the specified target.

Parameters:

target – the object whose run() method is called

- **Thread**

```
public Thread(ThreadGroup group,
              Runnable target)
```

Constructs a new Thread in the specified Thread group that applies the run() method of the specified target.

Parameters:

group – the Thread group

target – the object whose run() method is called

- **Thread**

```
public Thread(String name)
```

Constructs a new Thread with the specified name.
Parameters:
name – the name of the new Thread

- **Thread**

```
public Thread(ThreadGroup group,
              String name)
```

Constructs a new Thread in the specified Thread group with the specified name.
Parameters:
group – the Thread group
name – the name of the new Thread

- **Thread**

```
public Thread(Runnable target,
              String name)
```

Constructs a new Thread with the specified name and applies the run() method of the specified target.
Parameters:
target – the object whose run() method is called
name – the name of the new Thread

- **Thread**

```
public Thread(ThreadGroup group,
              Runnable target,
              String name)
```

Constructs a new Thread in the specified Thread group with the specified name and applies the run() method of the specified target.
Parameters:
group – the Thread group
target – the object whose run() method is called
name – the name of the new Thread

Methods

- **currentThread**

```
public static Thread currentThread()
```

Returns a reference to the currently executing Thread object.

o **yield**

```
public static void yield()
```

Causes the currently executing Thread object to yield. If there are other runnable Threads they will be scheduled next.

o **sleep**

```
public static void sleep(long millis) throws InterruptedException
```

Causes the currently executing Thread to sleep for the specified number of milliseconds.

Parameters:

millis – the length of time to sleep in milliseconds

o **sleep**

```
public static void sleep(long millis,
                         int nanos) throws InterruptedException
```

Sleep, in milliseconds and additional nanosecond.

Parameters:

millis – the length of time to sleep in milliseconds

nanos – 0–999999 additional nanoseconds to sleep

o **start**

```
public synchronized void start()
```

Starts this Thread. This will cause the run() method to be called. This method will return immediately.

Throws: IllegalThreadStateException

If the thread was already started.

See Also:

run, stop

o **run**

```
public void run()
```

The actual body of this Thread. This method is called after the Thread is started. You must either override this method by subclassing class Thread, or you must create the Thread with a Runnable target.

See Also:

start, stop

o **stop**

```
public final void stop()
```

Stops a Thread by tossing an object. By default this routine tosses a new instance of ThreadDeath to the target Thread. ThreadDeath is not actually a subclass of Exception, but is a subclass of Object. Users should not normally try to catch ThreadDeath unless they must do some extraordinary cleanup operation. If ThreadDeath is caught it is important to rethrow the object so that the thread will actually die. The top–level error handler will not print out a message if ThreadDeath falls through.

See Also:

start , run

- **stop**

```
public final synchronized void stop(Object o)
```

Stops a Thread by tossing an object. Normally, users should just call the stop() method without any argument. However, in some exceptional circumstances used by the stop() method to kill a Thread, another object is tossed. ThreadDeath, is not actually a subclass of Exception, but is a subclass of Object.

Parameters:

o – the object to be tossed

See Also:

start , run

- **interrupt**

```
public void interrupt()
```

Send an interrupt to a thread.

- **interrupted**

```
public static boolean interrupted()
```

Ask if you have been interrupted.

- **isInterrupted**

```
public boolean isInterrupted()
```

Ask if another thread has been interrupted.

- **destroy**

```
public void destroy()
```

Destroy a thread, without any cleanup, i.e. just toss its state; any monitors it has locked remain locked. A last resort.

o **isAlive**

```
public final boolean isAlive()
```

Returns a boolean indicating if the Thread is active. Having an active Thread means that the Thread has been started and has not been stopped.

o **suspend**

```
public final void suspend()
```

Suspends this Thread's execution.

o **resume**

```
public final void resume()
```

Resumes this Thread execution. This method is only valid after suspend() has been invoked.

o **setPriority**

```
public final void setPriority(int newPriority)
```

Sets the Thread's priority.
Throws: IllegalArgumentException
If the priority is not within the range MIN_PRIORITY, MAX_PRIORITY.
See Also:
MIN_PRIORITY, MAX_PRIORITY, getPriority

o **getPriority**

```
public final int getPriority()
```

Gets and returns the Thread's priority.
See Also:
setPriority

o **setName**

```
public final void setName(String name)
```

Sets the Thread's name.
Parameters:
name – the new name of the Thread
See Also:
getName

o **getName**

```
public final String getName()
```

Gets and returns this Thread's name.
See Also:
setName

o **getThreadGroup**

```
public final ThreadGroup getThreadGroup()
```

Gets and returns this Thread group.

o **activeCount**

```
public static int activeCount()
```

Returns the current number of active Threads in this Thread group.

o **enumerate**

```
public static int enumerate(Thread tarray[])
```

Copies, into the specified array, references to every active Thread in this Thread's group.
Returns:
the number of Threads put into the array.

o **countStackFrames**

```
public int countStackFrames()
```

Returns the number of stack frames in this Thread. The Thread must be suspended when this method is called.
Throws: IllegalThreadStateException
If the Thread is not suspended.

o **join**

```
public final synchronized void join(long millis) throws InterruptedException
```

Waits for this Thread to die. A timeout in milliseconds can be specified. A timeout of 0 milliseconds means to wait forever.
Parameters:
millis – the time to wait in milliseconds

o **join**

```
public final synchronized void join(long millis,
```

```
                                    int nanos) throws InterruptedException
```

Waits for the Thread to die, with more precise time.

- **join**

```
public final void join() throws InterruptedException
```

Waits forever for this Thread to die.

- **dumpStack**

```
public static void dumpStack()
```

A debugging procedure to print a stack trace for the current Thread.
See Also:
printStackTrace

- **setDaemon**

```
public final void setDaemon(boolean on)
```

Marks this Thread as a daemon Thread or a user Thread. When there are only daemon Threads left running in the system, Java exits.
Parameters:
on – determines whether the Thread will be a daemon Thread
Throws: IllegalThreadStateException
If the Thread is active.
See Also:
isDaemon

- **isDaemon**

```
public final boolean isDaemon()
```

Returns the daemon flag of the Thread.
See Also:
setDaemon

- **checkAccess**

```
public void checkAccess()
```

Checks whether the current Thread is allowed to modify this Thread.
Throws: SecurityException
If the current Thread is not allowed to access this Thread group.

- **toString**

```
public String toString()
```

Returns a String representation of the Thread.

Overrides:

toString in class Object
