# Java.util.concurrent.Semaphore class in Java

**Prerequisite :** Semaphores in Java

```
public class Semaphore
extends Object
implements Serializable
```

Conceptually, a semaphore maintains a set of permits. Each acquire() blocks if necessary until a permit is available, and then takes it. Each release() adds a permit, potentially releasing a blocking acquirer. However, no actual permit objects are used; the Semaphore just keeps a count of the number available and acts accordingly.

**Methods:**

1. **void acquire()** : This method acquires a permit, if one is available and returns immediately, reducing the number of available permits by one.If the current thread is interrupted while waiting for a permit then InterruptedException is thrown.

   ```
   Syntax :
   public void acquire() throws InterruptedException
   Parameters :
   NA
   Returns :
   NA
   Throws:
   InterruptedException - if the current thread is interrupted
   ```

2. **void acquire(int permits)** : This method acquires the given number of permits, if they are available, and returns immediately, reducing the number of available permits by the given amount.If the current thread is interrupted while waiting for a permit then InterruptedException is thrown.

   ```
   Syntax :
   public void acquire(int permits) throws InterruptedException
   Parameters :
   permits - the number of permits to acquire
   Returns :
   NA
   ```

```
Throws:
InterruptedException - if the current thread is interrupted
IllegalArgumentException - if permits is negative
```

3. **void acquireUninterruptibly()** : This method acquires a permit, if one is available and returns immediately, reducing the number of available permits by one.If the current thread is interrupted while waiting for a permit then it will continue to wait,

```
Syntax :
public void acquireUninterruptibly()
Parameters :
NA
Returns :
NA
```

4. **void acquireUninterruptibly(int permits)** : This method the given number of permits, if they are available, and returns immediately, reducing the number of available permits by the given amount.If the current thread is interrupted while waiting for a permit then it will continue to wait,

```
Syntax :
public void acquireUninterruptibly(int permits)
Parameters :
permits - the number of permits to acquire
Returns :
NA
Throws:
IllegalArgumentException - if permits is negative
```

5. **boolean tryAcquire()** : This method acquires a permit, if one is available and returns immediately, with the value true, reducing the number of available permits by one. If no permit is available then this method will return immediately with the value false.

```
Syntax :
public boolean tryAcquire()
Parameters :
NA
Returns :
true if a permit was acquired and false otherwise
```

6. **boolean tryAcquire(int permits)** : This method acquires the given number of permits, if they are available, and returns immediately, with the value true, reducing the number of available permits by the given amount.If insufficient permits are available then this method will return immediately with the value false.

```
Syntax :
public boolean tryAcquire(int permits)
Parameters :
permits - the number of permits to acquire
Returns :
true if the permits were acquired and false otherwise
```

```
Throws:
IllegalArgumentException - if permits is negative
```

7. **boolean tryAcquire(long timeout, TimeUnit unit)** : This method acquires a permit, if one is available and returns immediately, with the value true, reducing the number of available permits by one. If the specified waiting time elapses then the value false is returned. If the time is less than or equal to zero, the method will not wait at all.

```
Syntax :
public boolean tryAcquire(long timeout, TimeUnit unit)
     throws InterruptedException
Parameters :
timeout - the maximum time to wait for a permit
unit - the time unit of the timeout argument
Returns :
true if a permit was acquired and
false if the waiting time elapsed before a permit was acquired
Throws:
InterruptedException - if the current thread is interrupted
```

8. **boolean tryAcquire(int permits, long timeout, TimeUnit unit)** : This method acquires the given number of permits, if they are available and returns immediately, with the value true, reducing the number of available permits by the given amount. If the specified waiting time elapses then the value false is returned. If the time is less than or equal to zero, the method will not wait at all.Any permits that were to be assigned to this thread, are instead assigned to other threads trying to acquire permits.

```
Syntax :
public boolean tryAcquire(int permits, long timeout, TimeUnit unit)
throws InterruptedException
Parameters :
permits - the number of permits to acquire
timeout - the maximum time to wait for a permit
unit - the time unit of the timeout argument
Returns :
true if all permits were acquired and
false if the waiting time elapsed before all
permits were acquired
Throws:
InterruptedException - if the current thread is interrupted
IllegalArgumentException - if permits is negative
```

9. **void release()** : This method releases a permit, increasing the number of available permits by one. If any threads are trying to acquire a permit, then one is selected and given the permit that was just released.

```
Syntax :
public void release()
Parameters :
NA
```

```
Returns :
NA
```

10. **void release(int permits)** : This method releases the given number of permits, increasing the number of available permits by that amount.If any threads are trying to acquire permits, then one is selected and given the permits that were just released. If the number of available permits satisfies that thread's request then that thread is (re)enabled for thread scheduling purposes; otherwise the thread will wait until sufficient permits are available.

```
Syntax :
public void release(int permits)
Parameters :
permits - the number of permits to release
Returns :
NA
Throws :
IllegalArgumentException - if permits is negative
```

11. **int availablePermits()** : This method returns the current number of permits available in this semaphore.This method is typically used for debugging and testing purposes.

```
Syntax :
public int availablePermits()
Parameters :
NA
Returns :
the number of permits available in this semaphore
```

12. **int drainPermits()** : This method acquires and returns all permits that are immediately available.

```
Syntax :
public int drainPermits()
Parameters :
NA
Returns :
the number of permits acquired
```

13. **void reducePermits(int reduction)** : This method shrinks the number of available permits by the indicated reduction. This method can be useful in subclasses that use semaphores to track resources that become unavailable. This method differs from acquire in that it does not block waiting for permits to become available.

```
Syntax :
protected void reducePermits(int reduction)
Parameters :
reduction - the number of permits to remove
Returns :
NA
Throws :
IllegalArgumentException - if reduction is negative
```

14. **boolean isFair()** : This method returns true if this semaphore has fairness set true.

```
Syntax :
public boolean isFair()
Parameters :
NA
Returns :
true if this semaphore has fairness set true
```

15. **final boolean hasQueuedThreads()** : This method queries whether any threads are waiting to acquire. Note that because cancellations may occur at any time, a true return does not guarantee that any other thread will ever acquire. This method is designed primarily for use in monitoring of the system state.

```
Syntax :
public final boolean hasQueuedThreads()
Parameters :
NA
Returns :
true if there may be other threads waiting to acquire the lock
```

16. **final int getQueueLength()** : This method returns an estimate of the number of threads waiting to acquire. The value is only an estimate because the number of threads may change dynamically while this method traverses internal data structures. This method is designed for use in monitoring of the system state, not for synchronization control.

```
Syntax :
public final int getQueueLength()
Parameters :
NA
Returns :
the estimated number of threads waiting for this lock
```

17. **Collection getQueuedThreads()** : This method returns a collection containing threads that may be waiting to acquire. Because the actual set of threads may change dynamically while constructing this result, the returned collection is only a best-effort estimate. The elements of the returned collection are in no particular order.

```
Syntax :
protected Collection getQueuedThreads()
Parameters :
NA
Returns :
the collection of threads
```

18. **String toString()** : This method Returns a string identifying this semaphore, as well as its state. The state, in brackets, includes the String “Permits =” followed by the number of permits.

**Syntax :**
public String toString()
**Parameters :**
NA
**Returns :**
a string identifying this semaphore, as well as its state
**Overrides:**
toString in class Object

**Example explaining methods :** Note that output is not same all the time.

```
// Java program to demonstrate
// methods of Semaphore class
import java.util.concurrent.*;

class MyThread extends Thread
{
    Semaphore sem;
    String threadName;
    public MyThread(Semaphore sem, String threadName)
    {
        super(threadName);
        this.sem = sem;
        this.threadName = threadName;
    }

    @Override
    public void run() {

            // First, get a permit.
            System.out.println(threadName + " is waiting for a permit.");

               try {
                 // acquire method
                sem.acquire();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

               System.out.println(threadName + " gets a permit");

            // Now, critical section
            // other waiting threads will wait, until this
               // thread release the lock
            for(int i=0; i < 2; i++)
            {
                 // hasQueuedThreads() methods
                 boolean b = sem.hasQueuedThreads();
                 if(b)
                     // getQueuedLength() methods
                     System.out.println("Length of Queue : " + sem.getQueueLength())    ;

                 // Now, allowing a context switch -- if possible.
                 try {
                    Thread.sleep(10);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }


            // Release the permit.
            System.out.println(threadName + " releases the permit.");

            // release() method
```

```
            sem.release();
        }
}

// Driver class
public class SemaphoreDemo
{

    public static void main(String args[]) throws InterruptedException
    {
        // creating a Semaphore object
        // with number of permits 3 and fairness true
        Semaphore sem = new Semaphore(3, true);

        //isFair() method
        System.out.println("is Fairness enabled : " + sem.isFair());

        // Main thread try to acquire 2 permits
        // tryAcquire(int permits) method
        sem.tryAcquire(2);

        // availablePermits() method
        System.out.println("Available permits : " + sem.availablePermits());

        //drainPermits() method
        System.out.println("number of permits drain by Main thread : "
                                    + sem.drainPermits());

        // permit released by Main thread
        sem.release(1);

        // creating two threads with name A and B
        MyThread mt1 = new MyThread(sem, "A");
        MyThread mt2 = new MyThread(sem, "B");


        // starting threads A
        mt1.start();

        // starting threads B
        mt2.start();

        // toString method
        System.out.println(sem.toString());


        // waiting for threads A and B
        mt1.join();
        mt2.join();
    }
}
```

Run on IDE

Output:

```
is Fairness enabled : true
Available permits : 1
number of permits drain by Main thread : 1
java.util.concurrent.Semaphore@7852e922[Permits = 1]
B is waiting for a permit.
B gets a permit
A is waiting for a permit.
Length of Queue : 1
B releases the permit.
```

```
A gets a permit
A releases the permit.
```

This article is contributed by **Gaurav Miglani**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeks-forgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Java | Java - util package | Java-Multithreading

Login to Improve this Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

## Recommended Posts:

CountDownLatch in Java
Deadlock in Java Multithreading
Semaphore in Java
Daemon thread in Java
Callable and Future in Java
IntStream reduce(IntBinaryOperator op) in Java with Examples
Stream builder() in Java with Examples
Stream empty() in Java with Examples
Stream toArray() in Java with Examples
DoubleStream reduce(double identity, DoubleBinaryOperator op) in Java with Examples

(Login to Rate)

4.7 Average Difficulty : **4.7/5.0**
Based on **4** vote(s)

Basic | Easy | Medium | Hard | Expert

Add to TODO List

Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Share this post!