Java Thread Cheat Sheet

javaconceptoftheday.com

Basic Definitions

What is thread?

Thread is a smallest executable unit of a process. Thread has its own path of execution in a process. A process can have multiple threads.

What is process?

Process is an executing instance of an application. For example, when you double click MS Word icon in your computer, you start a process that will run MS word application.

What is application?

Application is a program which is designed to perform a specific task. For example : MS Word, Google Chrome, a video or audio player etc.

What is multithreaded programming?

In a program or in an application, when two or more threads execute their task simultaneously then it is called multithreaded programming. Java supports multithreaded programming.

Types Of Threads

There are two types of threads in Java.

1) User Threads :

User threads are threads which are created by the application or user. They are high priority threads. JVM will not exit until all user threads finish their execution. JVM wait for user threads to finish their task. These threads are foreground threads.

2) Daemon Threads :

Daemon threads are threads which are mostly created by the JVM. These threads always run in background. These threads are used to perform some background tasks like garbage collection. These threads are less priority threads. JVM will not wait for these threads to finish their execution. JVM will exit as soon as all user threads finish their execution.

Thread Priority

MIN_PRIORITY :

It defines the lowest priority that a thread can have and it's value is 1.

NORM_PRIORITY :

It defines the normal priority that a thread can have and it's value is 5.

MAX_PRIORITY :

It defines the highest priority that a thread can have and it's value is 10.

The default priority of a thread is same as that of it's parent. We can change the priority of a thread at any time using setPriority() method.

Thread States

There are six thread states - NEW, RUNNABLE, BLOCKED, WAITING, TIMED _WAITING and TERMINATED. At any point of time, a thread will be in any one of these states.

NEW : A thread will be in this state before calling start() method.

RUNNABLE : A thread will be in this state after calling the start() method.

BLOCKED : A thread will be in this state when a thread is waiting for object lock to enter into synchronized method/block or a thread will be in this state if deadlock occurs.

WAITING : A thread will be in this state when wait() or join() method is called.

TIMED_WAITING : A thread will be in this state when sleep() or wait() with timeOut or join() with timeOut is called.

TERMINATED : A thread will be in this state once it finishes it's execution.

How do you create threads in Java?

There are two ways to create threads in Java.

1) By extending java.lang.Thread class

```
class MyThread extends Thread
{
   @Override
   public void run()
   {
      //Keep the task to be performed here
   }
}

//Creating and starting MyThread
MyThread myThread = new MyThread();
myThread.start();
```

2) By implementing java.lang.Runnable interface

```
class MyRunnable implements Runnable
{
   @Override
   public void run()
   {
      //Keep the task to be performed here
   }
}

//Creating and starting MyRunnable
Thread t = new Thread(new MyRunnable());
t.start();
```

Thread Synchronization

Through synchronization, we can make the threads to execute a particular method or block in sync not simultaneously. Synchronization in Java is achieved using **synchronized** keyword.

When a method or block is declared as synchronized, only one thread can enter into that method or block.

The synchronization in Java is built around an entity called object lock or monitor.

Any thread wants to enter into synchronized methods or blocks of any object, they must acquire object lock associated with that object and release the lock after they are done with the execution.

```
synchronized void synchronizedMethod()
{
      //Synchronized Method
}
```

Deadlock

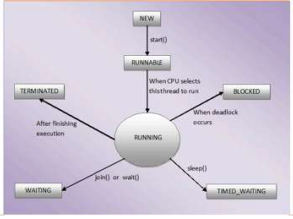
Deadlock in Java is a condition which occurs when two or more threads get blocked waiting for each other for an infinite period of time to release the resources (Locks) they hold.

Lock ordering and lock timeout are two methods which are used to avoid the deadlock in Java.

Lock Ordering : In this method of avoiding the deadlock, some predefined order is applied for threads to acquire the locks they need.

Lock Timeout : It is another deadlock preventive method in which we specify the time for a thread to acquire the lock. If it fails to acquire the specified lock in the given time, then it should give up trying for a lock and retry after some time.

Thread Life Cycle



Java.lang.Thread Methods

start() :

It starts execution of a thread.

run() :

It contains main task to be performed by the thread.

sleep() :

It makes the currently executing thread to pause it's execution for a specified period of time. When the thread is going for sleep, it does not release the locks it holds.

join() :

Using this method, you can make the currently executing thread to wait for some other threads to finish their task.

yield() :

It causes the currently executing thread to temporarily pause its execution and allow other threads to execute.

wait() :

It makes the currently executing thread to release the lock of this object and wait until some other thread notifies it.

notify() :

It wakes up one thread randomly which is waiting for this object's lock.

notifyAll() :

It wakes up all thread which are waiting for this object's lock. But, only one thread will acquire lock of this object depending upon the priority.

isAlive() :

It checks whether a thread is alive or not.

isDaemon() :

It checks whether a thread is daemon thread or user thread.

setDaemon() :

It sets daemon status of a thread.

currentThread() :

It returns a reference to currently executing thread.

interrupt() :

It is used to interrupt a thread.

isInterrupted() :

It checks whether a thread is interrupted or not.

getId() :

It returns ID of a thread.

getState() :

It returns current state of a thread.

getName() and setName() :

Getter and setter for name of a thread.

getPriority() and setPriority() :

Getter and setter for priority of a thread.

getThreadGroup() :

It returns a thread group to which this thread belongs to.

Inter Thread Communication

Threads in Java communicate with each other using wait(), notify() and notifyAll() methods.

wait() : This method tells the currently executing thread to release the lock of this object and wait until some other thread acquires the same lock and notify it using either notify() or notifyAll() methods.

notify() : This method wakes up one thread randomly that called wait() method on this object.

notifyAll() : This method wakes up all the threads that called wait() method on this object. But, only one thread will acquire lock of this object depending upon the priority.