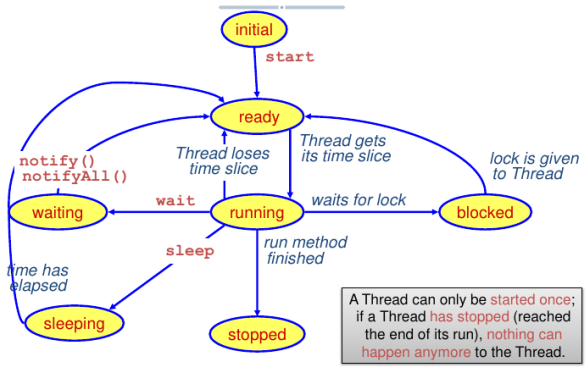
Java Threads Cheat Sheet

What is a thread?

It is the smallest executable unit of a process. A process usually has multiple threads. Multiple threads can be run in parallel.

Thread life cycle



How to create a thread

The inheritance way: the start method starts the thread and returns immediately

```
class CanBeRunParallel extends Thread {
    //...
    public void run() {
        //Thread statements
    }
}

CanBeRunParallel parObj = new
    CanBeRunParallel();

parObj.start(); //Calls the .run() method
```

The interface way is a general method, to be used when the class extends another class.

```
class CanBeRunParallel implements Runnable {
    //...
    public void run() {
        //Thread statements
    }
}

CanBeRunParallel parObj = new
    CanBeRunParallel();

Thread myThread = new Thread(parObj);
myThread.start(); //Calls the .run() method
```

The synchronized keyword

The `synchronized` keyword instruct Java to **lock** the object until the method terminates.
If the object is already locked when calling the method, the callee gets blocked until the lock is released.

```
//It can synchronize a whole method:
class CanBeRunParallel extends Thread {
    synchronized public void nowSafeMethod()
    {
        //Synchronized statements
        //The lock is obtained on the object
    }

    public void notSynchronized() {
        /*or it can synchronize just some
    statements*/
        //Not-syncronized statements
        synchronized(this) {
            //Synchronized statements
            //The lock is obtained on the
    object passed to the synchronized block
        }
    }
}
```

The `synchronized` keyword lets you obtain a lock on a specific object, **not on primitive types!** If you need to lock a primitive type variable, create a different object, and synchronize that one.

Liveliness problems

Deadlock: Two or more threads are locked forever, each one waiting for the other to release a lock.
Starvation: One thread has a hard time trying to get access to resources because other greedy tasks constantly invoke long synchronized methods.
Livelock: Cyclic sequence of operation that gets the program stuck in an useless loop.

Thread methods

`void run()`: contains the main body of the thread.
`void start()`: causes this thread to begin execution, by calling its run method.
`void sleep(long millis)`: Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds

Executors

Executors are used in large scale application, as they offer an alternative way to concurrently run different objects that implement the Runnable interface, without instantiating several Thread objects.

```
public class ExecutorClass implements
    Executable {...}

public class Runnie implements Runnable {...}

Runnie r1 = new Runnie();
Runnie r2 = new Runnie();

//Classic way
(new Thread(r1)).start();
(new Thread(r2)).start();

//Executor way
e = ExecutorClass();
e.execute(r1);
e.execute(r2);
```

Guarded blocks

`void wait() throws InterruptedException`: causes the current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object.
`void notify()`: Wakes up a single thread that is waiting on this object's monitor. The thread is chosen in a non-deterministic way.
`void notifyAll()`: Wakes up **all** threads that are waiting on this object's monitor.
These methods should only be called by a thread that is the owner of this object's monitor.

Lock interface

Implementations of the `Lock` provide additional functionality over the use of synchronized methods and statements by providing a non-blocking attempt to acquire a lock: `boolean tryLock()`.

```
Lock lock = ...;
if (lock.tryLock()) {
    try {
        // manipulate protected state
    } finally {
        lock.unlock();
    }
} else {/*alternative action*/ }
```