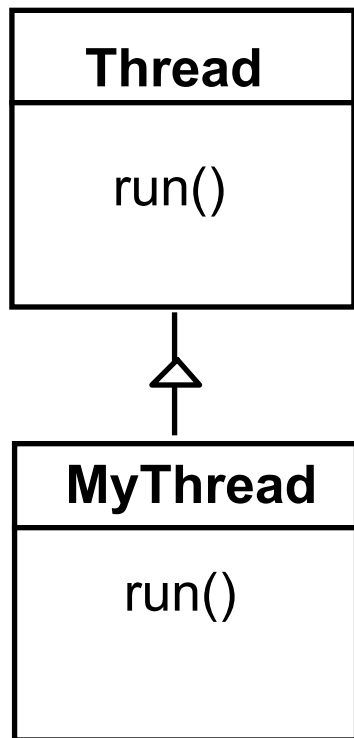


threads in Java

A Thread class manages a single sequential thread of control. Threads may be created and deleted dynamically.



The Thread class executes instructions from its method run(). The actual code executed depends on the implementation provided for run() in a derived class.

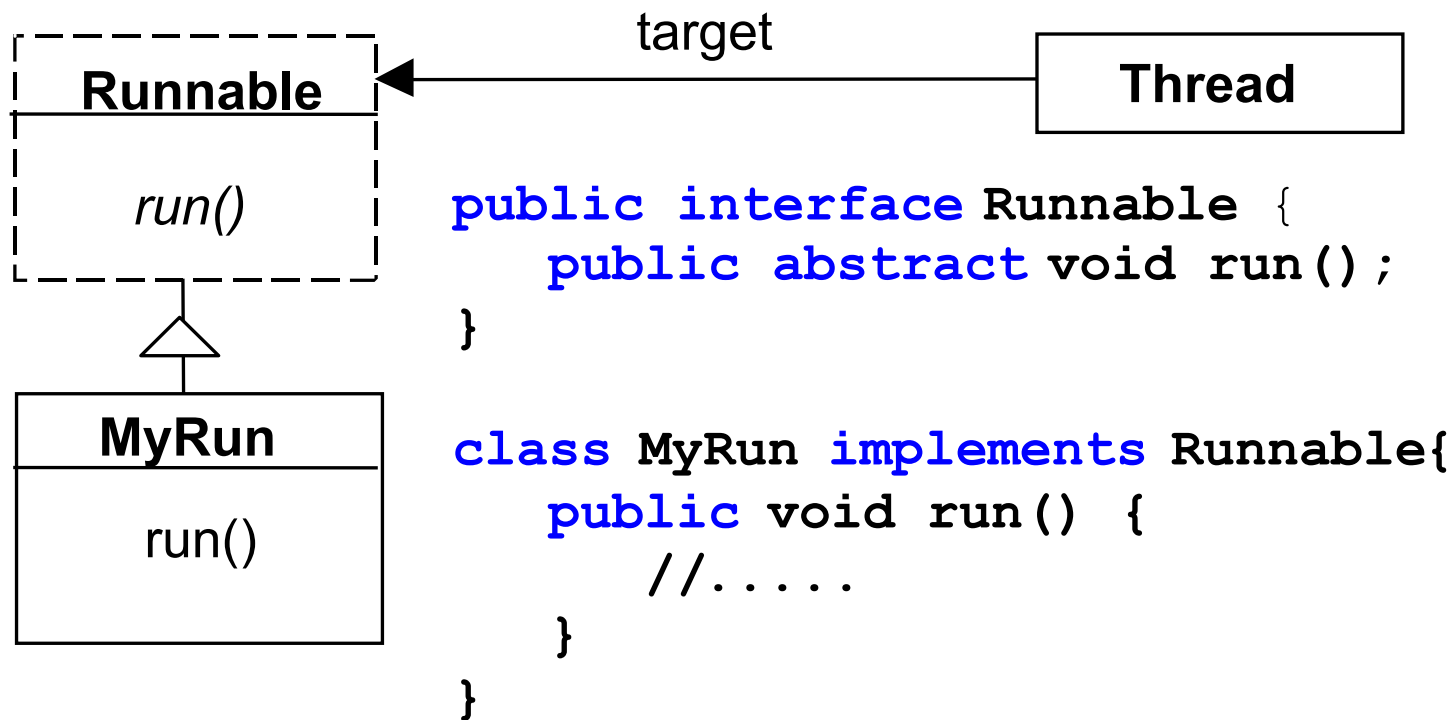
```
class MyThread extends Thread {
    public void run() {
        //.....
    }
}
```

Creating a thread object:

```
Thread a = new MyThread();
```

threads in Java

Since Java does not permit multiple inheritance, we often implement the **run()** method in a class not derived from Thread but from the interface Runnable.

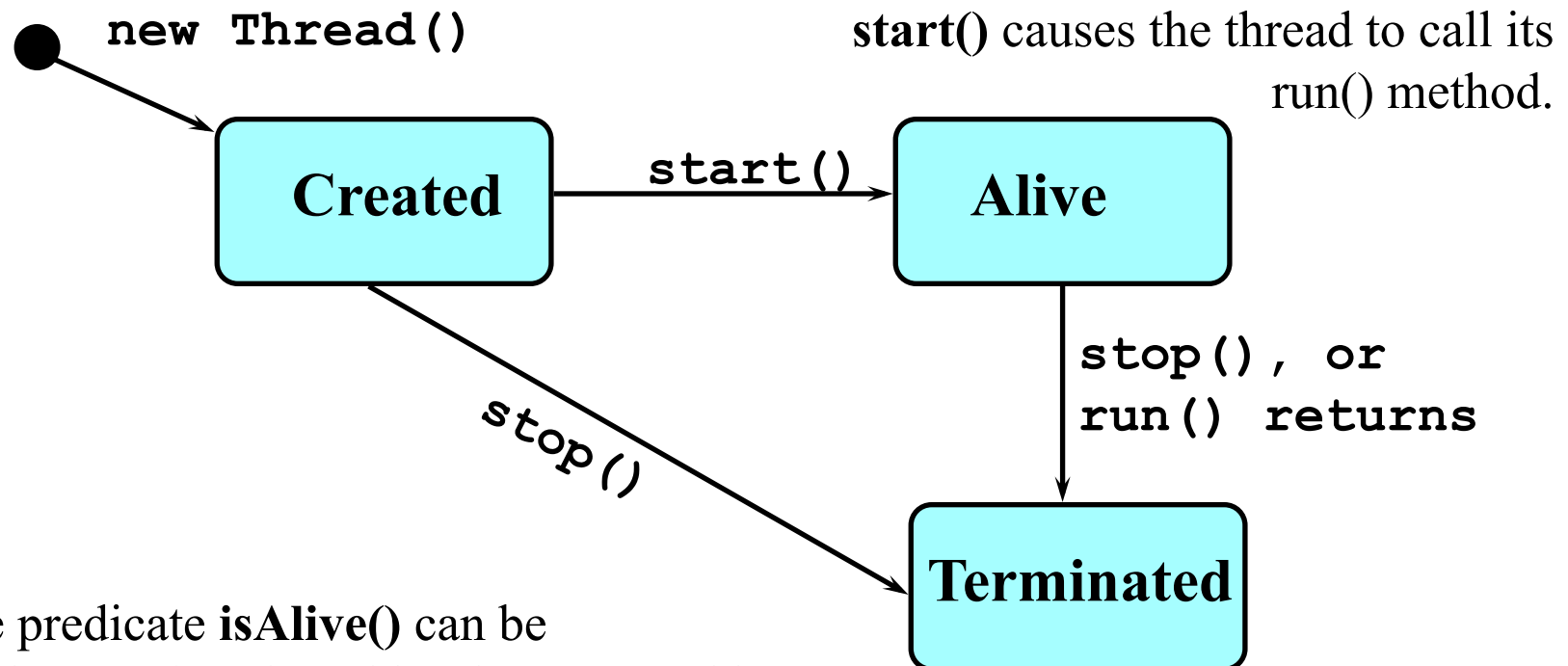


Creating a thread object:

```
Thread b = new Thread(new MyRun());
```

thread life-cycle in Java

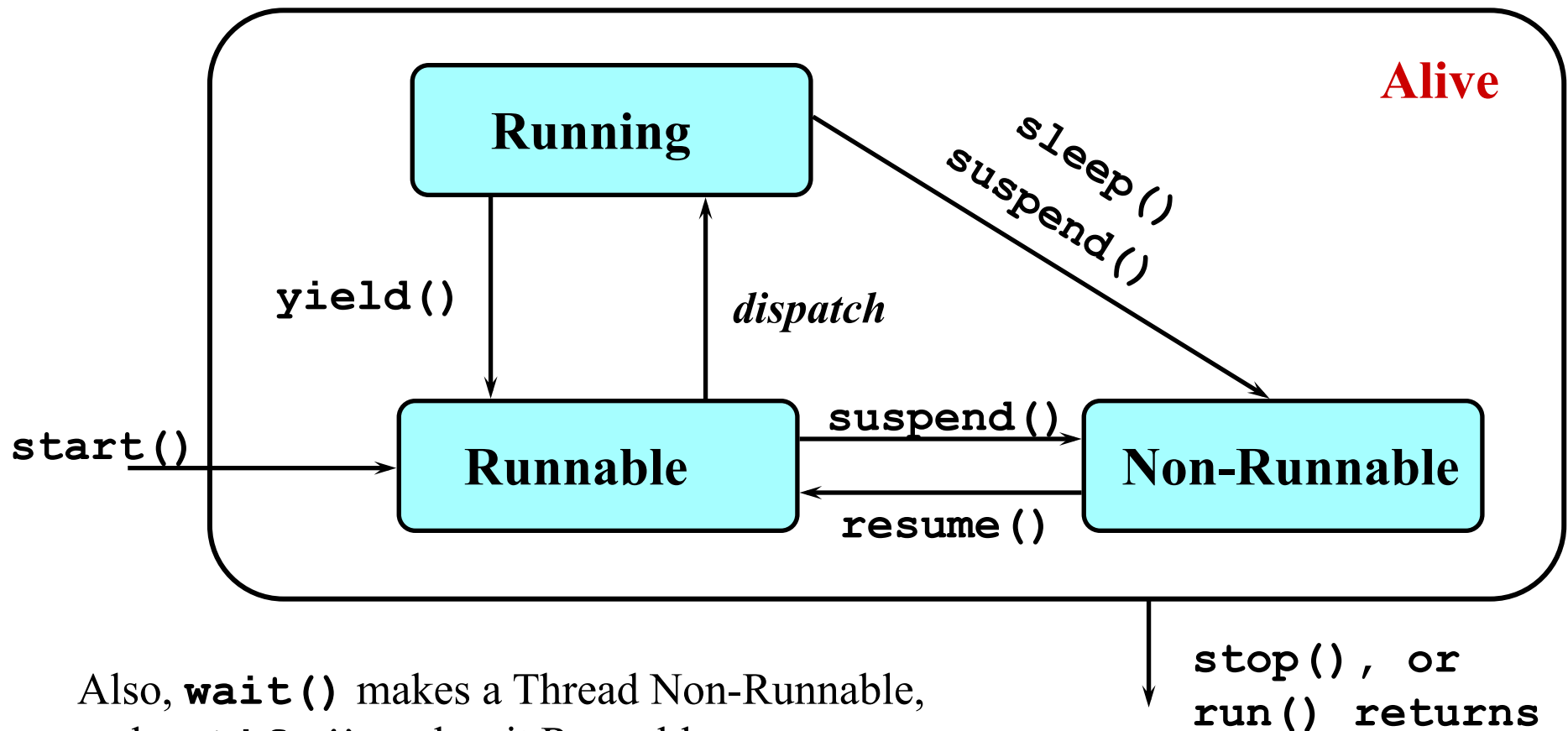
An overview of the life-cycle of a thread as state transitions:



The predicate `isAlive()` can be used to test if a thread has been started but not terminated. Once terminated, it cannot be restarted (cf. mortals).

thread **alive** states in Java

Once started, an **alive** thread has a number of substates :

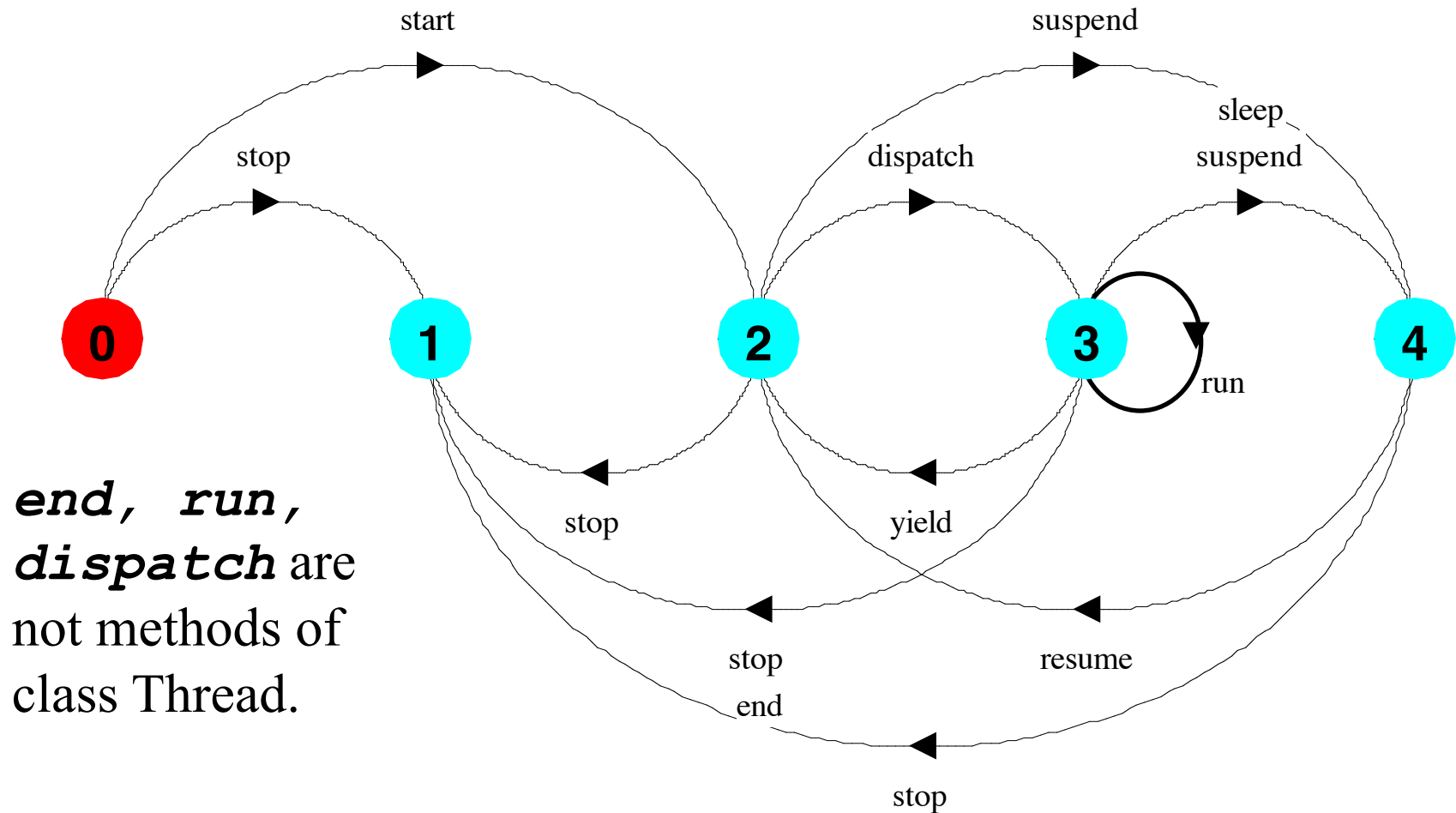


Also, `wait()` makes a Thread Non-Runnable, and `notify()` makes it Runnable (used in later chapters).

Java thread lifecycle - an FSP specification

```
THREAD          = CREATED ,
CREATED         = (start          ->RUNNABLE
                  |stop          ->TERMINATED) ,
RUNNING        = ({suspend,sleep}->NON_RUNNABLE
                  |yield         ->RUNNABLE
                  |{stop,end}    ->TERMINATED
                  |run           ->RUNNING) ,
RUNNABLE       = (suspend        ->NON_RUNNABLE
                  |dispatch      ->RUNNING
                  |stop          ->TERMINATED) ,
NON_RUNNABLE   = (resume         ->RUNNABLE
                  |stop          ->TERMINATED) ,
TERMINATED     = STOP.
```

Java thread lifecycle - an FSP specification



States 0 to 4 correspond to **CREATED**, **TERMINATED**, **RUNNABLE**, **RUNNING**, and **NON-RUNNABLE** respectively.