Chapter 17: Java Threads

- Thread represents a separate path of execution of a group of statements.

- Thread Applications:
  - Threads are used in server-side programs because a server should be multi-threaded, so that various clients can be handled at the same time.
  - Threads can also be used to create games and animation.

Thread States and Transitions:

- Thread may be in one of the five states:
  - New
  - Runnable
  - Running
  - Blocked state(Waiting / Sleeping)
  - Terminated / Dead state

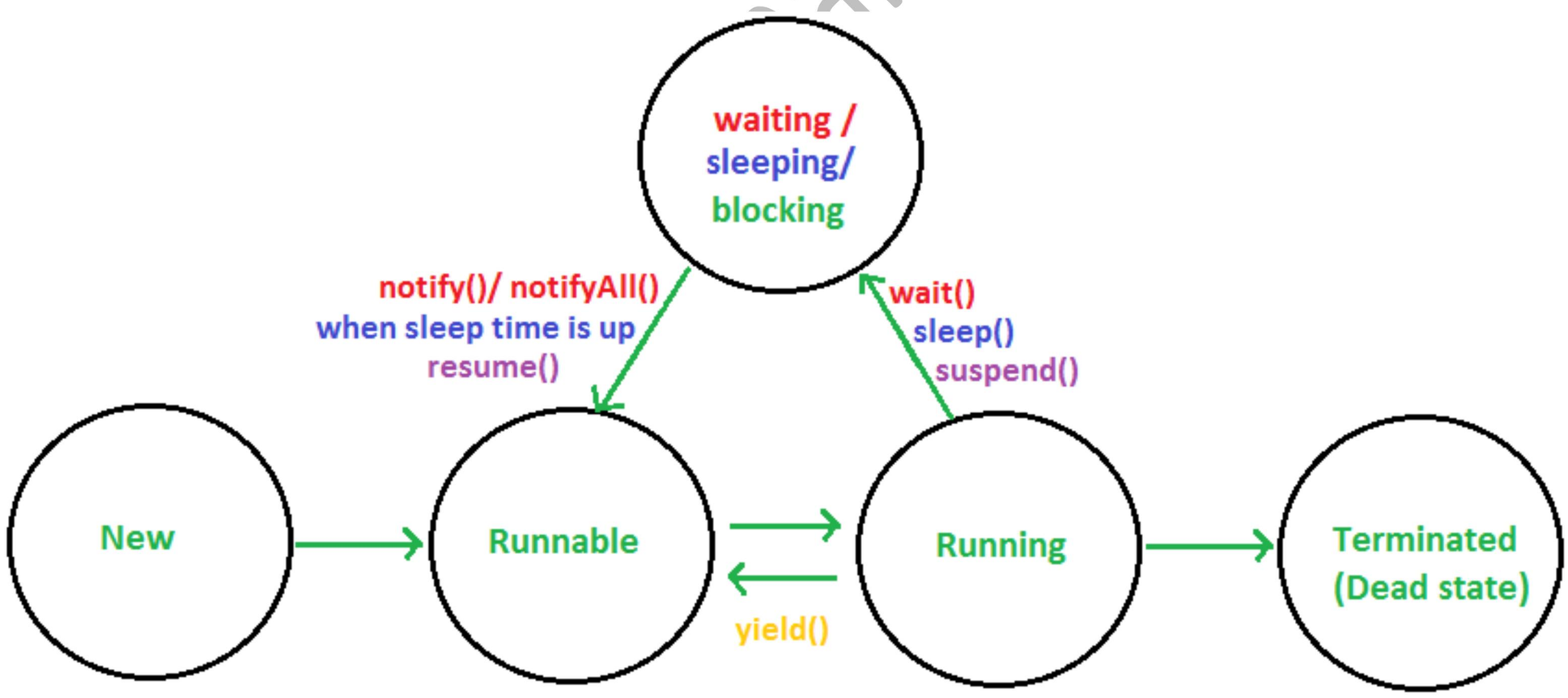

Fig. THREAD STATES

New State:

- ❑ This is the state the thread is in after the Thread instance has been created, but the start() method has not been invoked on the thread.

Runnable State:

- ❑ This is the state a thread is in when it's eligible to run, but the scheduler has not selected it to be the running thread.
- ❑ A thread first enters the runnable state when the start() method is invoked, but a thread can also return to the runnable state after either running or coming back from a blocked, waiting, or sleeping state.

Running State:

- ❑ This is the state a thread is in when the thread scheduler selects it (from the runnable pool) to be the currently executing process.
- ❑ Thread Scheduler:
  - ❑ The component of JVM that decides the order in which threads will be executed is termed as the thread scheduler.
  - ❑ If all threads have equal priority, then they are given time slots for execution in round-robin fashion.
  - ❑ If threads priority is not same, then the one that is having the highest priority would be given first chance.
  - ❑ This process of assigning time to threads is known as time-slicing.
  - ❑ The order in which runnable threads are chosen to run is not guaranteed.

Blocked State:

- ❑ This is the state a thread is in when it's not eligible to run.
- ❑ **Sleeping / Timed Waiting State:**
- ❑ A thread may be sleeping because the thread's run code tells it to sleep for some period of time, in which case the event that sends it back to runnable is that it wakes up because its sleep time has expired.
- ❑ **Waiting State:**
- ❑ A thread may be waiting, because the thread's run code causes it to wait, in which case the event that sends it back to runnable is that another thread sends a notification that it may no longer be necessary for the thread to wait.

Terminated State:

- ❑ A thread is considered dead when its run() method completes.
- ❑ It may still be a viable Thread object, but it is no longer a separate thread of execution.
- ❑ Once a thread is dead, it can never be brought back to life!

Thread Constructors:

Constructors	Description
Thread()	Allocates a new Thread object with Automatically generated names are of the form "Thread-"+*n*, where *n* is an integer.
Thread(String threadName)	It will give a name to the thread.
Thread(Runnable target)	It is used to supply the object of the class that implements the Runnable Interface.
Thread(Runnable target, String threadName)	It is used to supply the object of the class that implements the Runnable Interface along with the thread name,

Thread Methods:

Methods	Description
public void start()	Causes this thread to begin execution;
long getId()	Returns the identifier of this Thread.
final String getName()	returns the threads name.
final void setName(String name)	sets the name of the thread.
final int getPriority()	returns the thread's priority.
final void setPriority(int newPriority)	Changes the priority of this thread.
static void sleep(long millis) throws InterruptedException	Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds
static Thread currentThread()	Returns a reference to the currently executing thread object.
static void yield()	A hint to the scheduler that the current thread is willing to yield its current use of a processor.
final void join() throws InterruptedException	Waits for this thread to die

Extending Thread Class:

- Step1: Declare the class as extending the Thread Class
- Step2: The extending class must override a method called run().
- Step3:
  - a) Create a thread object
  - OR
  - b) Pass an object that extends Thread class as a parameter to the constructor of an object of type Thread.
- Step4: Call the start() method to initiate thread execution.

Implementing Runnable interface:

- Step1: Declare the class as implementing the Runnable interface.
- Step2: Implement the run() method
- Step3: Pass an object that implements the Runnable interface as a parameter to the constructor of an object of type Thread.
- Step4: Call the threads start() method to run the thread.

Callable:

- **Callable:** It is an interface that declares on method :
  - V call() throws Exception
- It represents a task that needs to be completed by thread.
- Once the task completes it returns a value.
- If call() method cannot execute or fails, it throws Exception

Executors, ExecutorService & Future:

- To execute a task using Callable object, we need a thread pool which can be created by using **Executors** utility class.
- Few ways of creating thread pools:
  - 1) **newSingleThreadExecutor()**: it uses a single thread to execute the tasks.
  - 2) **newFixedThreadPool(int n)**: it creates a fixed thread pool of n – number of threads. **(very popular)**
- **ExecutorService:** It provides management capabilities and production of Future objects.
- **Future:** It represents objects that contain a value that is returned by thread in the future (by using get() method).