Overview of How Concurrent Programs are Developed in Java (Part 1)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

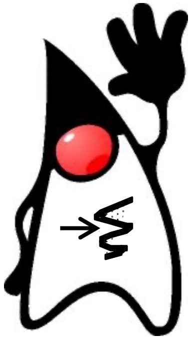
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

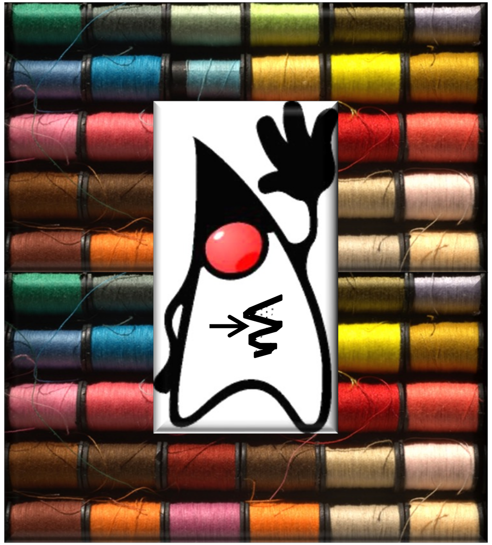
Learning Objectives in this Part of the Lesson

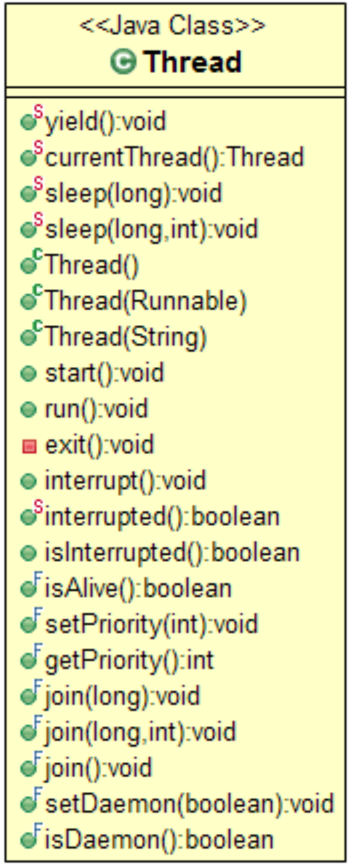
- Understand the meaning of key concurrent programming concepts
- Recognize how Java supports concurrent programming concepts



Learning Objectives in this Part of the Lesson

- Understand the meaning of key concurrent programming concepts
- Recognize how Java supports concurrent programming concepts, e.g.
  - Thread objects





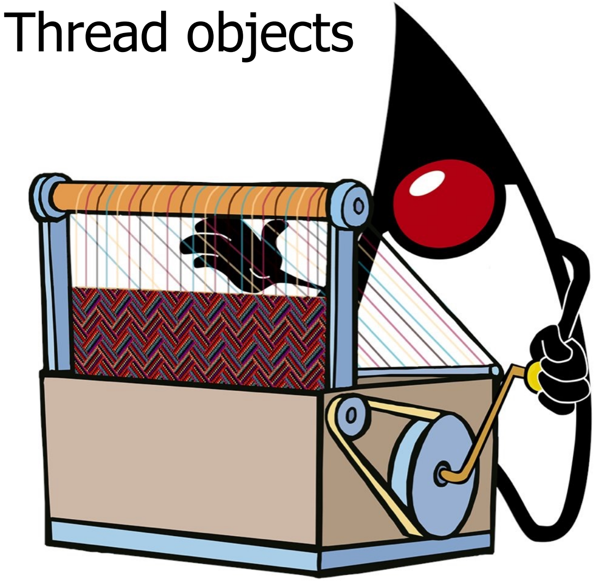
<<Java Class>>
Thread

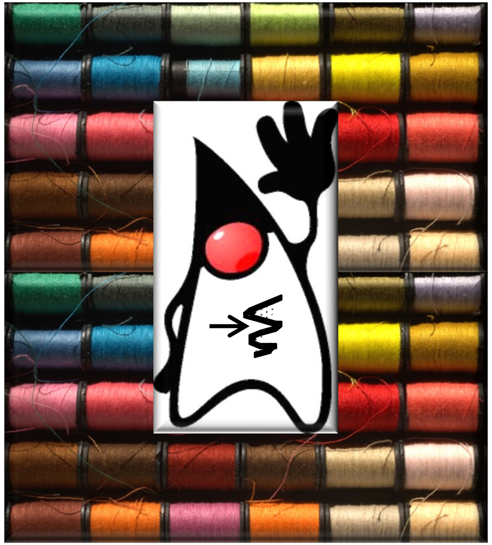
- yield():void
- currentThread():Thread
- sleep(long):void
- sleep(long,int):void
- Thread()
- Thread(Runnable)
- Thread(String)
- start():void
- run():void
- exit():void
- interrupt():void
- interrupted():boolean
- isInterrupted():boolean
- isAlive():boolean
- setPriority(int):void
- getPriority():int
- join(long):void
- join(long,int):void
- join():void
- setDaemon(boolean):void
- isDaemon():boolean

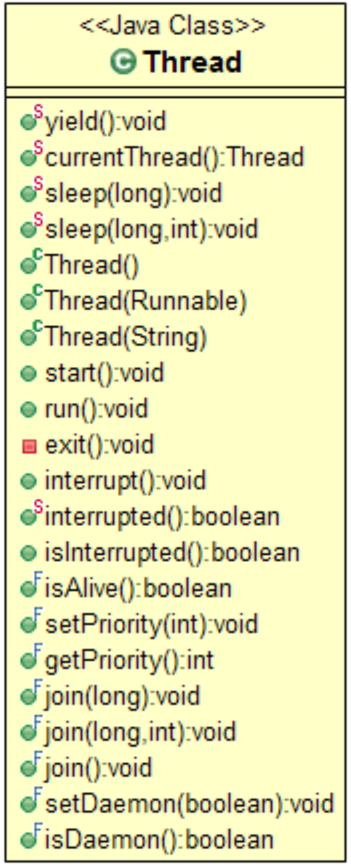
See docs.oracle.com/javase/8/docs/api/java/lang/Thread.html

Learning Objectives in this Part of the Lesson

- Understand the meaning of key concurrent programming concepts
- Recognize how Java supports concurrent programming concepts, e.g.
  - Thread objects







<<Java Class>>
Thread

yield():void
currentThread():Thread
sleep(long):void
sleep(long,int):void
Thread()
Thread(Runnable)
Thread(String)
start():void
run():void
exit():void
interrupt():void
interrupted():boolean
isInterrupted():boolean
isAlive():boolean
setPriority(int):void
getPriority():int
join(long):void
join(long,int):void
join():void
setDaemon(boolean):void
isDaemon():boolean

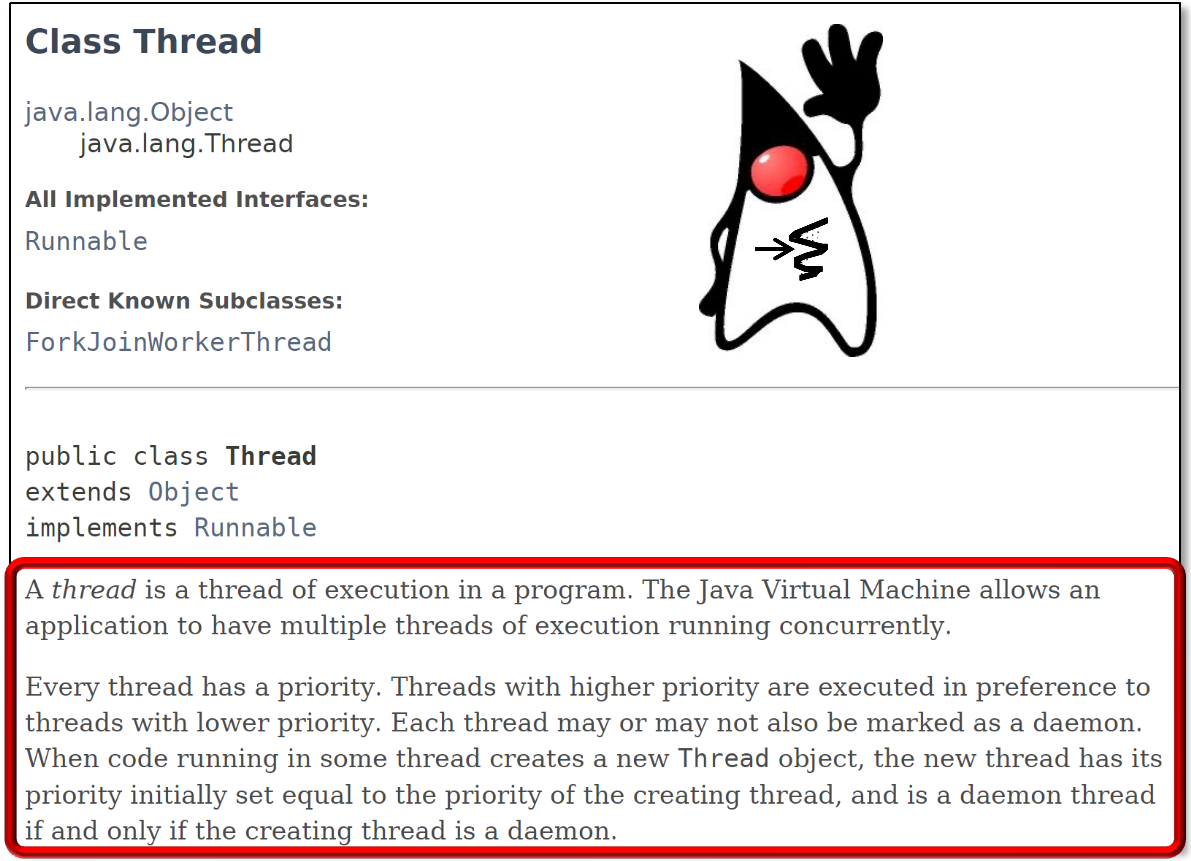
Java threads are under-going major changes as part of Project Loom

See wiki.openjdk.java.net/display/loom/Main

An Overview of Java Threads

An Overview of Java Threads

- A Java Thread is an object



Class Thread

java.lang.Object
    java.lang.Thread

All Implemented Interfaces:

`Runnable`

Direct Known Subclasses:

`ForkJoinWorkerThread`

```
public class Thread
extends Object
implements Runnable
```

A *thread* is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.

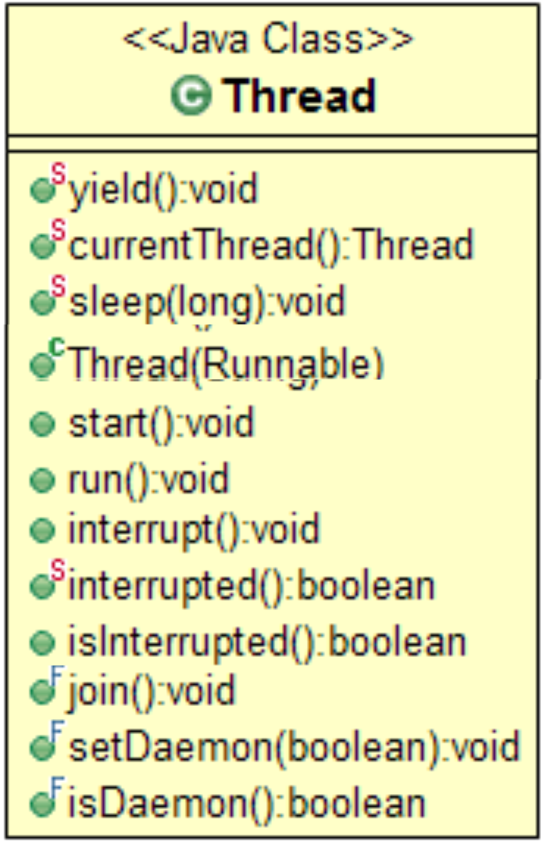
Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority. Each thread may or may not also be marked as a daemon. When code running in some thread creates a new `Thread` object, the new thread has its priority initially set equal to the priority of the creating thread, and is a daemon thread if and only if the creating thread is a daemon.

See docs.oracle.com/javase/8/docs/api/java/lang/Thread.html

An Overview of Java Threads

- A Java Thread is an object, e.g.
  - It contains methods & (internal) fields

```
public class Thread
       implements Runnable {
    private volatile char name[];
    private int priority;
    private boolean daemon = false;
    private Runnable target;
    ThreadLocal.ThreadLocalMap
       threadLocals = null;
    private long stackSize;
    private long tid;
    ...
```



<<Java Class>>

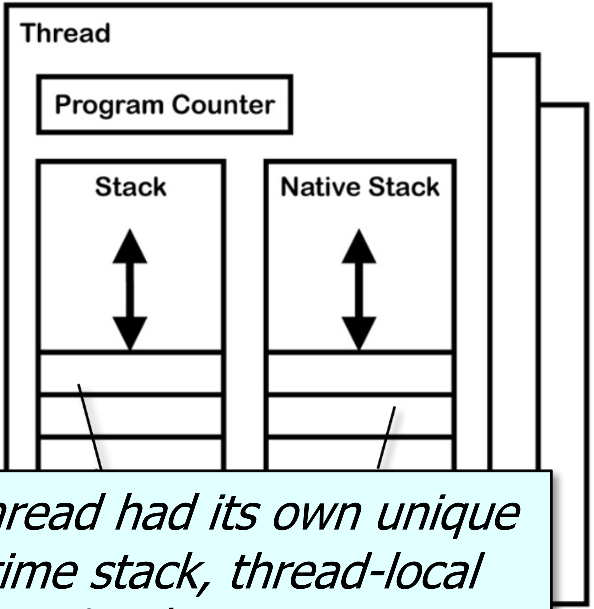
Thread

yield():void
currentThread():Thread
sleep(long):void
Thread(Runnable)
start():void
run():void
interrupt():void
interrupted():boolean
isInterrupted():boolean
join():void
setDaemon(boolean):void
isDaemon():boolean

See blog.jamesdbloom.com/JVMInternals.html

An Overview of Java Threads

- A Java Thread is an object, e.g.
  - It contains methods & (internal) fields

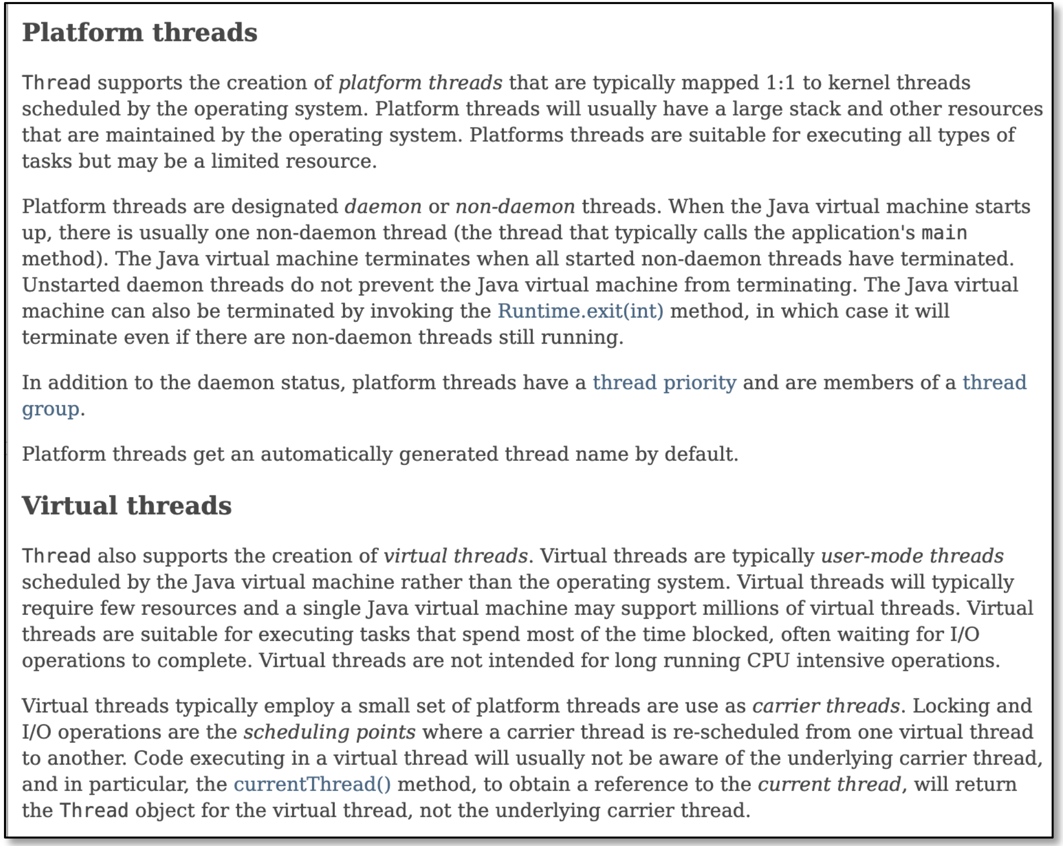


Historically each Java Thread had its own unique id, name, priority, runtime stack, thread-local storage, instruction pointer, & other registers, etc.

See blog.jamesdbloom.com/JVMInternals.html

An Overview of Java Threads

- A Java Thread is an object, e.g.
  - It contains methods & (internal) fields



Platform threads

Thread supports the creation of *platform threads* that are typically mapped 1:1 to kernel threads scheduled by the operating system. Platform threads will usually have a large stack and other resources that are maintained by the operating system. Platforms threads are suitable for executing all types of tasks but may be a limited resource.

Platform threads are designated *daemon* or *non-daemon* threads. When the Java virtual machine starts up, there is usually one non-daemon thread (the thread that typically calls the application's `main` method). The Java virtual machine terminates when all started non-daemon threads have terminated. Unstarted daemon threads do not prevent the Java virtual machine from terminating. The Java virtual machine can also be terminated by invoking the Runtime.exit(int) method, in which case it will terminate even if there are non-daemon threads still running.

In addition to the daemon status, platform threads have a thread priority and are members of a thread group.

Platform threads get an automatically generated thread name by default.

Virtual threads

Thread also supports the creation of *virtual threads*. Virtual threads are typically *user-mode threads* scheduled by the Java virtual machine rather than the operating system. Virtual threads will typically require few resources and a single Java virtual machine may support millions of virtual threads. Virtual threads are suitable for executing tasks that spend most of the time blocked, often waiting for I/O operations to complete. Virtual threads are not intended for long running CPU intensive operations.

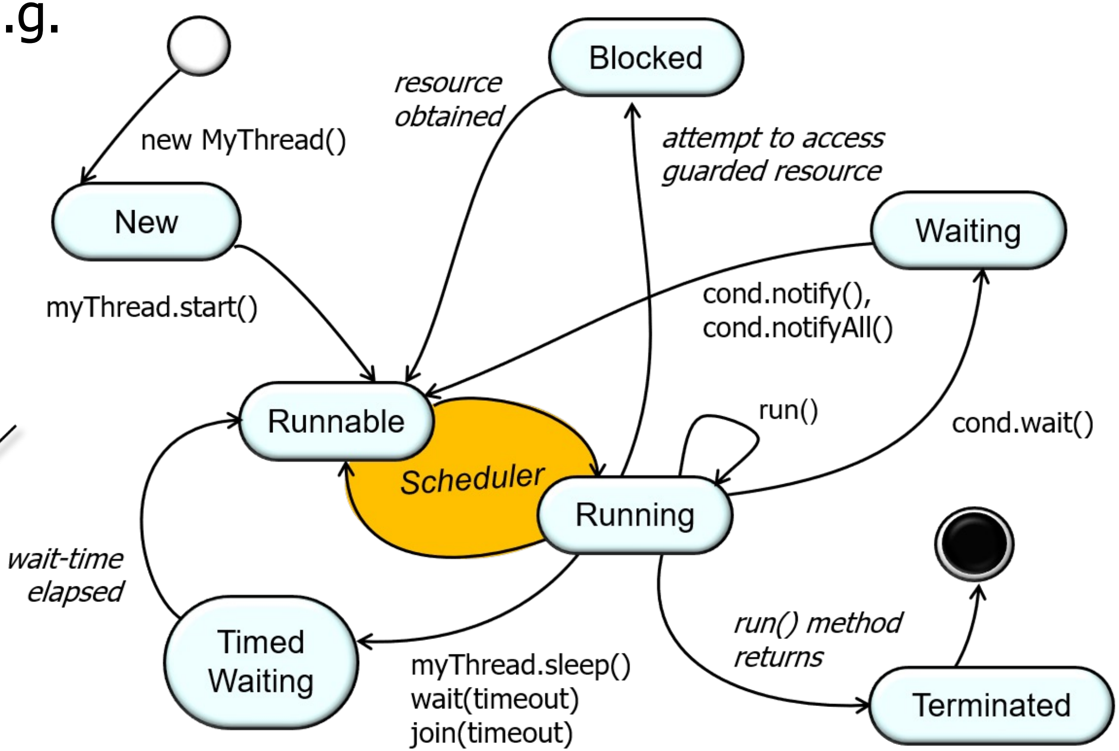
Virtual threads typically employ a small set of platform threads are use as *carrier threads*. Locking and I/O operations are the *scheduling points* where a carrier thread is re-scheduled from one virtual thread to another. Code executing in a virtual thread will usually not be aware of the underlying carrier thread, and in particular, the currentThread() method, to obtain a reference to the *current thread*, will return the Thread object for the virtual thread, not the underlying carrier thread.

Traditional Java Thread objects are now called "platform threads", whereas new "virtual threads" are "lightweight" concurrency objects

See download.java.net/java/early_access/loom/docs/api/java.base/java/lang/Thread.html

An Overview of Java Threads

- A Java Thread is an object, e.g.
  - It contains methods & (internal) fields
  - It can also be in one of various "states"

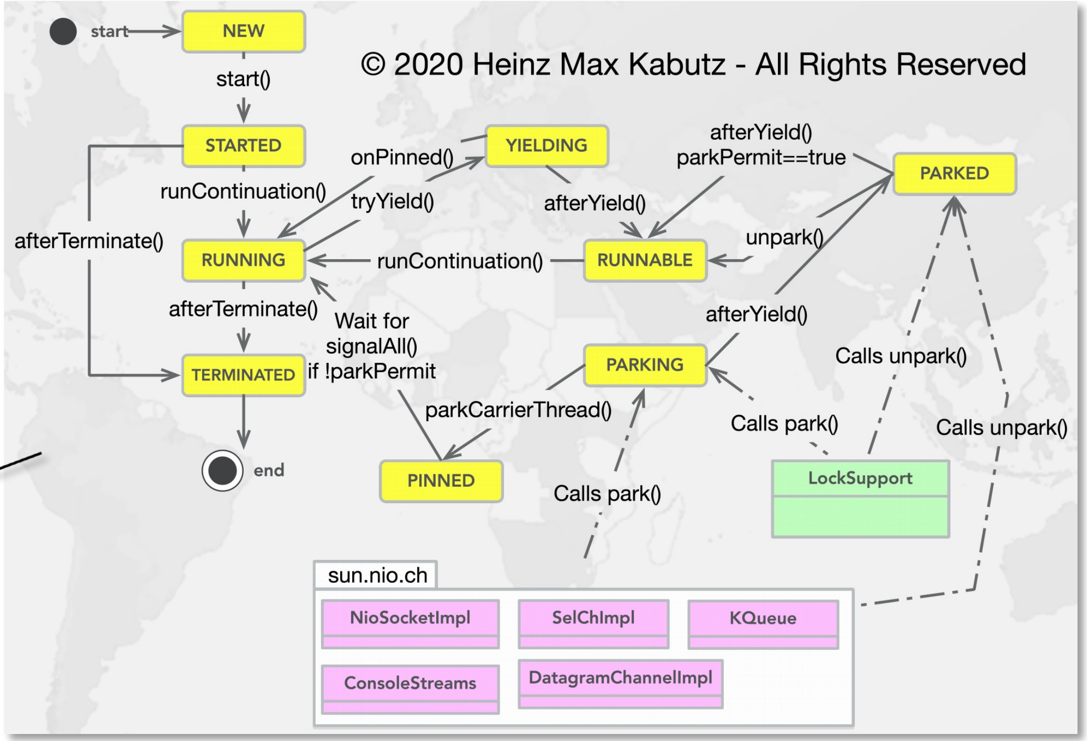


States of traditional Java (platform) threads

See docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html

An Overview of Java Threads

- A Java Thread is an object, e.g.
  - It contains methods & (internal) fields
  - It can also be in one of various "states"



States of modern Java virtual threads

See www.youtube.com/watch?v=5brCaY31y1M

End of Overview of How Concurrent Programs are Developed in Java (Part 1)