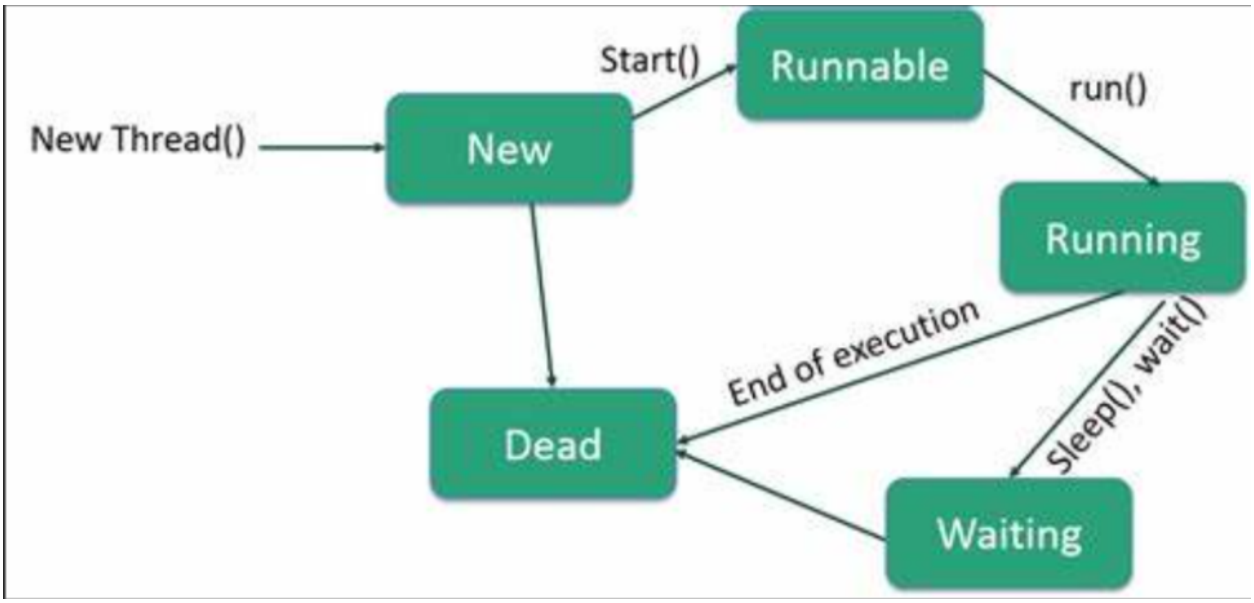
Java Threads

☑ Favorite	☐
↗ Tag	Java Network Programming

A thread is a single sequential flow of control within a program



▼ The role of threads in networking

In network programming, such as with an HTTP server or web server, threads play a crucial role in enabling concurrent connections from multiple clients. Without multithreading, only one client could connect to the server at a time, limiting the server's capacity to handle multiple requests simultaneously.

Multithreading allows the server to create separate, independent threads for each client connection request. This means that each client communicates with its own dedicated thread on the server, allowing for independent and simultaneous communication between the server and multiple clients. Overall, multithreading enhances the scalability and responsiveness of network servers by enabling them to handle multiple client requests concurrently.

▼ Runnable thread

A "runnable" thread is one that has been created and is eligible to run, but it may not be executing at that exact moment because the scheduler has not yet selected it to run on the CPU. Once the thread is chosen by the scheduler, it transitions from the "runnable" state to the "running" state, where its code is actively being executed.

- 2 ways of implementing threads in java
  - Extending java.lang.Thread class

Extending the `Thread` class:

java Copy code

```java
public class MyThread extends Thread {
    public void run() {
        // Code to be executed by the thread goes here
        System.out.println("Thread is running");
    }
}
```

To create and start a new thread:

java Copy code

```java
MyThread thread = new MyThread();
thread.start();
```

  - Implementing java.lang.Runnable interface

Implementing the `Runnable` interface:

```java
public class MyRunnable implements Runnable {
    public void run() {
        // Code to be executed by the thread goes here
        System.out.println("Thread is running");
    }
}
```

To create a `Runnable` object and pass it to a `Thread` object:

```java
MyRunnable myRunnable = new MyRunnable();
Thread thread = new Thread(myRunnable);
```

▼ Difference between start() & run() methods

1. `start()` **method**:
   - Invoking the `start()` method on a thread object creates a new thread of execution.
   - The new thread executes the code inside the `run()` method of the thread class.
   - The `start()` method returns immediately after creating the new thread and does not wait for the thread to finish its execution.
   - Calling `start()` method multiple times on the same thread object will result in an `IllegalStateException`.
2. `run()` **method**:
   - Invoking the `run()` method directly does not create a new thread; instead, it executes the code inside the `run()` method on the current thread.
   - The code inside the `run()` method will run in the context of the current thread and will not create a new thread.

- The `run()` method does not return until the code inside it has finished executing.
- Calling `run()` method multiple times on the same thread object is allowed.

In summary, the `start()` method is used to start a new thread of execution and execute the code inside the `run()` method on that new thread, while the `run()` method is used to execute the code inside the `run()` method on the current thread without creating a new thread.