# Multithreading

16 April 2025 23:13

## Process

A process is an independent program in execution. It has its own memory space system resources and at least one thread (main thread).
Example: Running to Java applications means two separate processes.(e.g., game and a calculator)

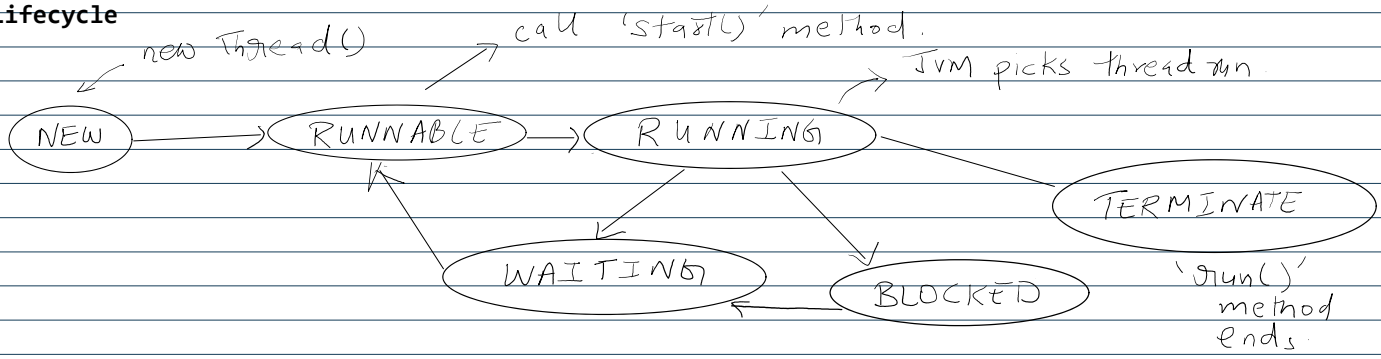
## Thread

A thread is the smallest unit of execution within a process. Multiple threads can run concurrently in the same process, sharing memory and resources.
Example: In a web browser (a process) One thread may load the page another, may play a video.

### Multiprocessing vs Multithreading

| Feature | Muti-processing | Multithreading |
|---|---|---|
| Units of execution. | Multiple processes. | Multiple threads. |
| Memory usage. | More (Each process has separate memory) | Less (threads share memory) |
| Context switching | Expensive | Faster |
| Communication | Interprocess communication (IPC) Needed. | Direct sharing of variables. |
| Stability | More stable (crash isolated) | Less stable(one thread crash may affect others). |

### Thread Lifecycle




Creating Threads in Java

① By extending Thread class

```
public class MyClass extends Thread {
    public void run() {
        System.out.println("Thread is running...");
    }
}
```

② By implementing Runnable Interface.

```
public class MyRunable implements Runnable. {
    public void run() {
```

public class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Runnable Thread is running...");
    }
}

Usage:

```
Thread t = new Thread(new MyRunnable());
t.start();
```

Common Methods

start() => Starts thread execution.
run() => Code to execute in thread.
sleep(ms) => Pause thread for milliseconds.
join() => Waits for the thread to die.
yield() => Hint to thread scheduler to pause.
isAlive() => Checks if thread is alive.
setPriority() => Sets priority(1 to 10)
getPriority() => Gets priority

**Thread Synchronization: When multiple thread access shared data. (Example of bank account.) Bears a risk of data inconsistency. Synchronization ensures only one thread access or critical section at a time.**

**Synchronized method:**

```
class  BankAccount{
   private int balance = 1000;
   public synchronized void withdraw(int amount){
      if(balance >= amount){
         System.out.println(Thread.currentThread().getName() + " is withdrawing " + amount);
         Balance -= amount;
      }
      else{
         System.out.println("insufficient balance for " + Thread.currentThread().getName());
      }
   }
}
```

**Synchronized blocks**

```
public void withdrawl(int amount){
    synchronized(this){
      // critical section
    }
}
```

**Inter thread communication.**

1. **wait()  => releases locks**
2. **notify() => wakes up a single waiting thread**
3. **notifyAll() => wakes up all waiting thread**

**Example:**

```
    synchronized(obj){
        while(!condition){
           Obj.wait();
```

```
                    //do work
                    Obj.notify();
                }
            }
```

**Problems:**

1. Deadlock
2. Race conditions
3. Starvation: Thread never gets cpu time.