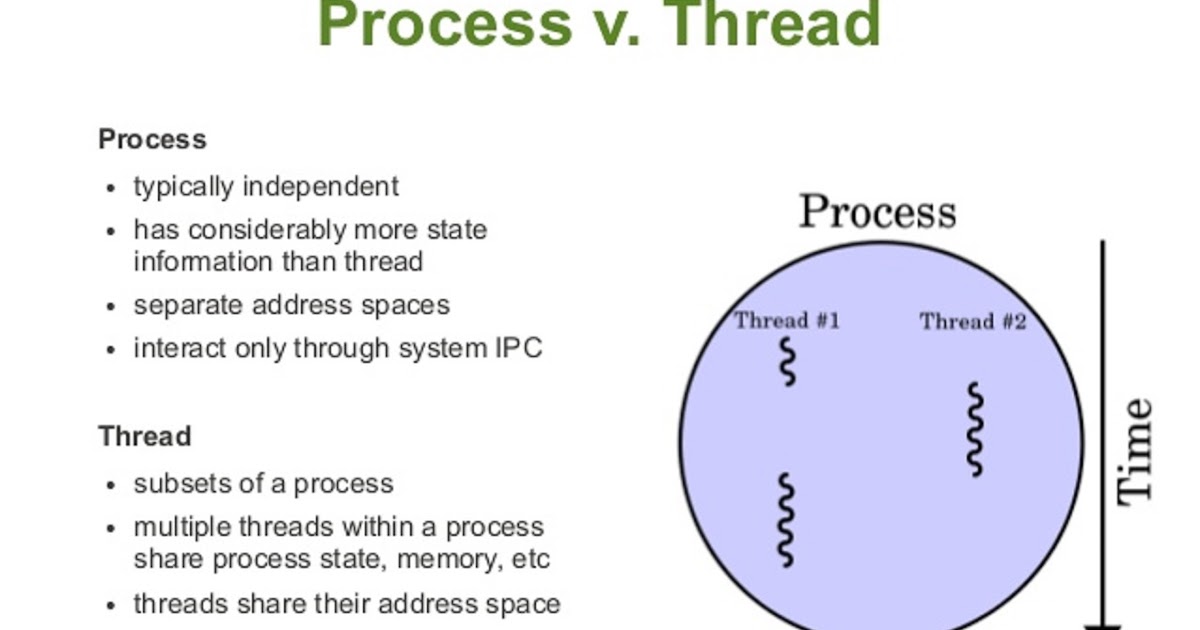
Multithreading Implementation

Thread:

* A thread is a lightweight sub process, the smallest unit of processing. *(It is a separate path of execution.)*
* Threads are independent. *(If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.)*

## Process:

* **A process is an instance of a program that is being executed.**
* When we run a program, it does not execute directly. It takes some time to follow all the steps required to execute the program, and following these execution steps is known as a process.



# 

# Multithreading in Java

* **Multithreading in**[**Java**](https://www.javatpoint.com/java-tutorial) is a process of executing multiple threads simultaneously.
* However, we use multithreading more than multiprocessing because threads use a shared memory area. *(They don't allocate separate memory area so saves memory)*

Advantages & Uses of Multithreading are

* Java Multithreading is mostly used in games, animation, etc.
* Multithreading saves time as you can perform multiple operations together.
* The threads are independent, and also, if an exception occurs in a single thread, it does not affect other threads.
* It does not block other user while performing multiple operations at the same time.

Threads can be created by using two mechanisms:

**Note:** It extends from java.lang.Thread class **(Pre-defined class)**

* Extending Thread class **[E.g. class class\_name extends Thread]**
* Implementing Runnable Interface **[E.g. class** **class\_name implements Runnable]**

Interview Questions:

1. **What is difference between Thread and Multithreading?**

* Thread is said to be an lightweight sub process or we can say it is an smallest unit of processing within process.
* Whereas Multithreading is a process of executing multiple threads simultaneously. Multithreading is used to obtain the multitasking.

### **What are the states in the lifecycle of a Thread?**

A thread can have one of the following states during its lifetime:

* **New:** In this state, a Thread class object is created using a new operator, but the thread is not alive. Thread doesn't start until we call the start() method.
* **Runnable:** In this state, the thread is ready to run after calling the start() method. However, the thread is not yet selected by the thread scheduler.
* **Running:** In this state, the thread scheduler picks the thread from the ready state, and the thread is running.
* **Non-Runnable/Waiting/Blocked:** In this state, a thread is not running but still alive, or it is waiting for the other thread to finish.
* **Dead/Terminated:** A thread is in terminated or dead state when the run() method exits.

### **Differentiate between the Thread class and Runnable interface for creating a Thread?**

* By extending the Thread class
* By implementing the Runnable interface

**However, the primary differences between both the ways are given below:**

* By extending the Thread class, we cannot extend any other class, as Java does not allow multiple inheritances while implementing the Runnable interface; we can also extend other base class (if required).
* By extending the Thread class, each of thread creates the unique object and associates with it while implementing the Runnable interface; multiple threads share the same object

### **What does join() method?**

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task. Join method is overloaded in Thread class in the following ways.

### **What is the difference between wait() and sleep() method?**

|  |  |
| --- | --- |
| **wait()** | **sleep()** |
| 1) The wait() method is defined in Object class. | The sleep() method is defined in Thread class. |
| 2) The wait() method releases the lock. | The sleep() method doesn't release the lock. |