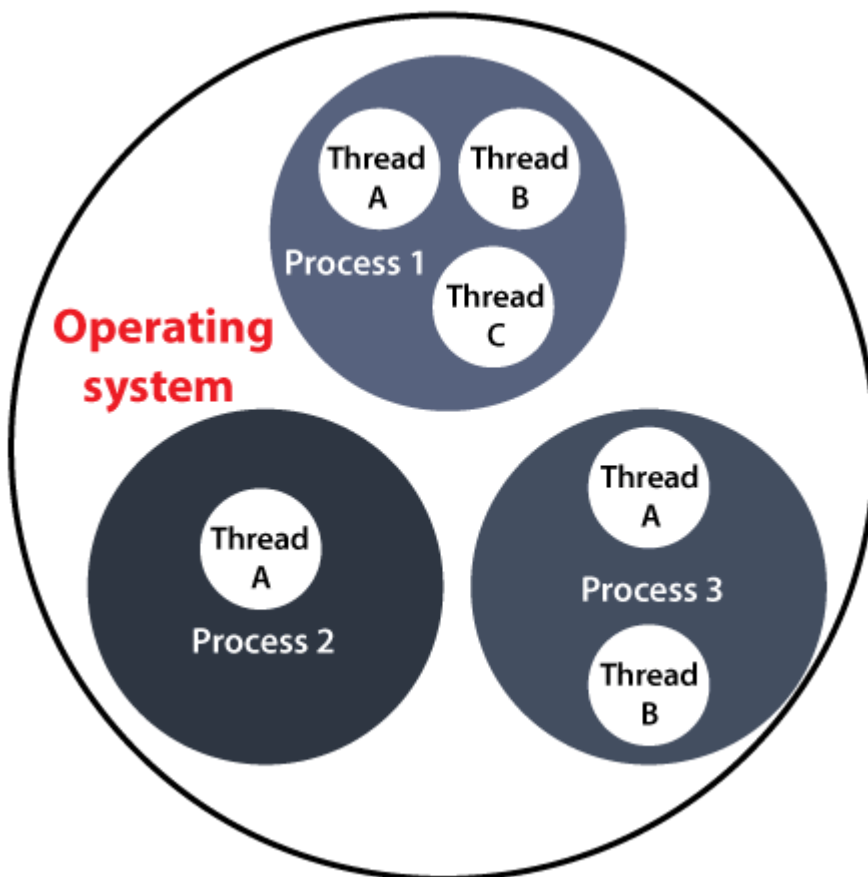


Thread Concept in Java

Before introducing the **thread concept**, we were unable to run more than one task in parallel. It was a drawback, and to remove that drawback, **Thread Concept** was introduced.

A **Thread** is a very light-weighted process, or we can say the smallest part of the process that allows a program to operate more efficiently by running multiple tasks simultaneously.

All the tasks are executed without affecting the main program. In a program or process, all the threads have their own separate path for execution, so each thread of a process is independent.



Another benefit of using **thread** is that if a thread gets an exception or an error at the time of its execution, it doesn't affect

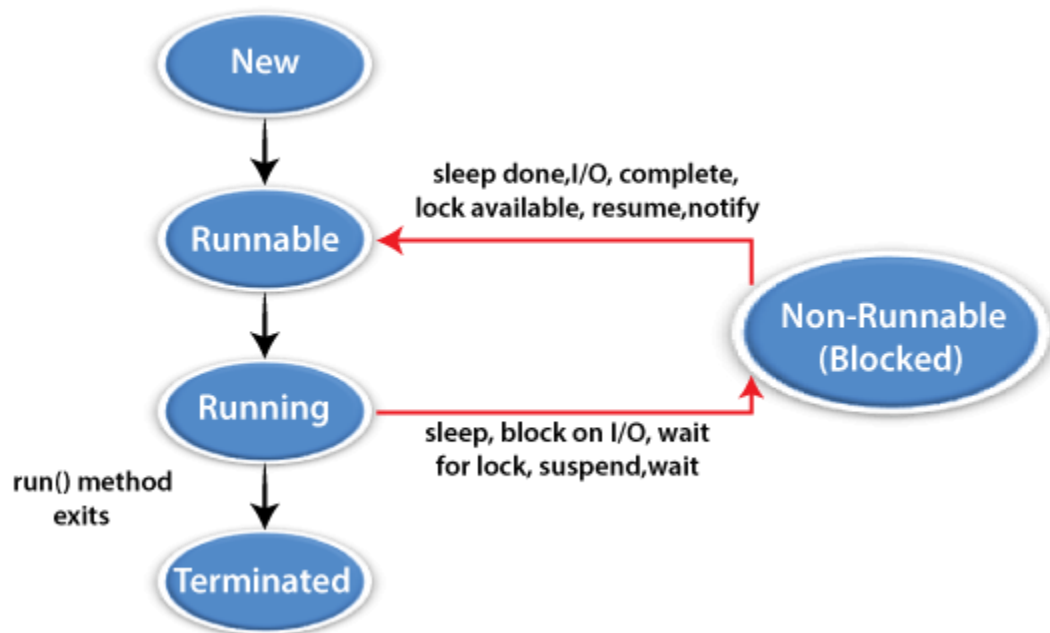
the execution of the other threads. All the threads share a common memory and have their own stack, local variables and program counter. When multiple threads are executed in parallel at the same time, this process is known as Multithreading

In a simple way, a Thread is a:

- Feature through which we can perform multiple activities within a single process.
- Lightweight process.
- Series of executed statements.
- Nested sequence of method calls.

Thread Model

Just like a process, a thread exists in several states. These states are as follows:



1) New (Ready to run)

A thread is in **New** when it gets CPU time.

2) Running

A thread is in a **Running** state when it is under execution.

3) Suspended

A thread is in the **Suspended** state when it is temporarily inactive or under execution.

4) Blocked

A thread is in the **Blocked** state when it is waiting for resources.

5) Terminated

A thread comes in this state when at any given time, it halts its execution immediately.

Creating Thread

A thread is created either by "creating or implementing" the **Runnable Interface** or by extending the **Thread class**. These are the only two ways through which we can create a thread.

A **Thread class** has several methods and constructors which allow us to perform various operations on a thread. The **Thread** class extends the **Object** class. The **Object** class implements the **Runnable** interface. The thread class has the following constructors that are used to perform various operations.

- **Thread()**
- **Thread(Runnable, String name)**
- **Thread(Runnable target)**
- **Thread(ThreadGroup group, Runnable target, String name)**
- **Thread(ThreadGroup group, Runnable target)**
- **Thread(ThreadGroup group, String name)**
- **Thread(ThreadGroup group, Runnable target, String name, long stackSize)**

Runnable Interface(run() method)

The Runnable interface is required to be implemented by that class whose instances are intended to be executed by a thread. The runnable interface gives us the **run()** method to perform an action for the thread.

start() method

The method is used for starting a thread that we have newly created. It starts a new thread with a new callstack. After executing the **start()** method, the thread changes the state from New to Runnable. It executes the **run() method** when the thread gets the correct time to execute it.

Let's take an example to understand how we can create a [Java](#)

thread by extending the Thread class:

Thread1.java

// Implementing runnable interface by extending Thread class

```
public class Thread1 extends Thread {
```

```
    // run() method to perform action for thread.
```

```
    public void run()
```

```
    {
```

```
        int a= 10;
```

```
        int b=12;
```

```
        int result = a+b;
```

```
        System.out.println("Thread started running..");
```

```
        System.out.println("Sum of two numbers is: "+ result);
```

```
    }
```

```
    public static void main( String args[] )
```

```
    {
```

```
        // Creating instance of the class extend Thread class
```

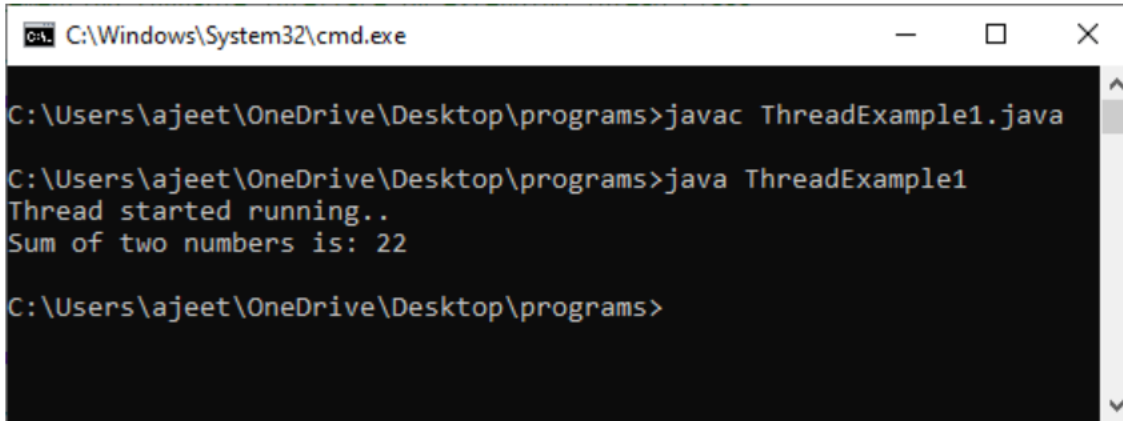
```
        Thread1 t1 = new Thread1();
```

```
        //calling start method to execute the run() method of the Thread class
```

```
        t1.start();
```

```
}  
}
```

Output:



```
C:\Windows\System32\cmd.exe  
  
C:\Users\ajet\OneDrive\Desktop\programs>javac ThreadExample1.java  
  
C:\Users\ajet\OneDrive\Desktop\programs>java ThreadExample1  
Thread started running..  
Sum of two numbers is: 22  
  
C:\Users\ajet\OneDrive\Desktop\programs>
```

Creating thread by implementing the runnable interface

In Java, we can also create a thread by implementing the runnable interface. The runnable interface provides us both the run() method and the start() method.

Let's take an example to understand how we can create, start and run the thread using the runnable interface.

Thread2.java

```
class NewThread implements Runnable {  
    String name;  
    Thread thread;  
    NewThread (String name){  
        this.name = name;  
        thread = new Thread(this, name);  
        System.out.println( "A New thread: " + thread+ "is created\n" );  
        thread.start();  
    }  
    public void run() {
```

```

try {
    for(int j = 5; j > 0; j--) {
        System.out.println(name + ": " + j);
        Thread.sleep(1000);
    }
} catch (InterruptedException e) {
    System.out.println(name + " thread Interrupted");
}
System.out.println(name + " thread exiting.");
}

class Thread2 {
    public static void main(String args[]) {
        new Thread("1st");
        new Thread("2nd");
        new Thread("3rd");
        try {
            Thread.sleep(8000);
        } catch (InterruptedException excetion) {
            System.out.println("Interruption occurs in Main Thread");
        }
        System.out.println("We are exiting from Main Thread");
    }
}

```

```
C:\Windows\System32\cmd.exe

C:\Users\ajeet\OneDrive\Desktop\programs>javac ThreadExample2.java

C:\Users\ajeet\OneDrive\Desktop\programs>java ThreadExample2
A New thread: Thread[1st,5,main]is created

A New thread: Thread[2nd,5,main]is created

A New thread: Thread[3rd,5,main]is created

3rd: 5
1st: 5
2nd: 5
3rd: 4
2nd: 4
1st: 4
1st: 3
3rd: 3
2nd: 3
1st: 2
3rd: 2
2nd: 2
3rd: 1
1st: 1
2nd: 1
3rd thread exiting.
1st thread exiting.
2nd thread exiting.
We are exiting from Main Thread

C:\Users\ajeet\OneDrive\Desktop\programs>_
```

Life cycle of a Thread (Thread States)

In Java, a thread always exists in any one of the following states. These states are:

1. New
2. Active
3. Blocked / Waiting
4. Timed Waiting
5. Terminated

Java Threads | How to create a thread in Java

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

FileName: Multi.java

```
1. class Multi extends Thread{
2. public void run(){
3. System.out.println("thread is running...");
4. }
5. public static void main(String args[]){
6. Multi t1=new Multi();
7. t1.start();
8. }
9. }
```

Output:

```
thread is running...
```

2) Java Thread Example by implementing Runnable interface

FileName: Multi3.java

```
1. class Multi3 implements Runnable{
2. public void run(){
```



```

3. System.out.println("thread is running...");
4. }
5.
6. public static void main(String args[]){
7.     Multi3 m1=new Multi3();
8.     Thread t1=new Thread(m1); // Using the constructor Thread(Runnable r)
9.     t1.start();
10. }
11. }

```

Output:

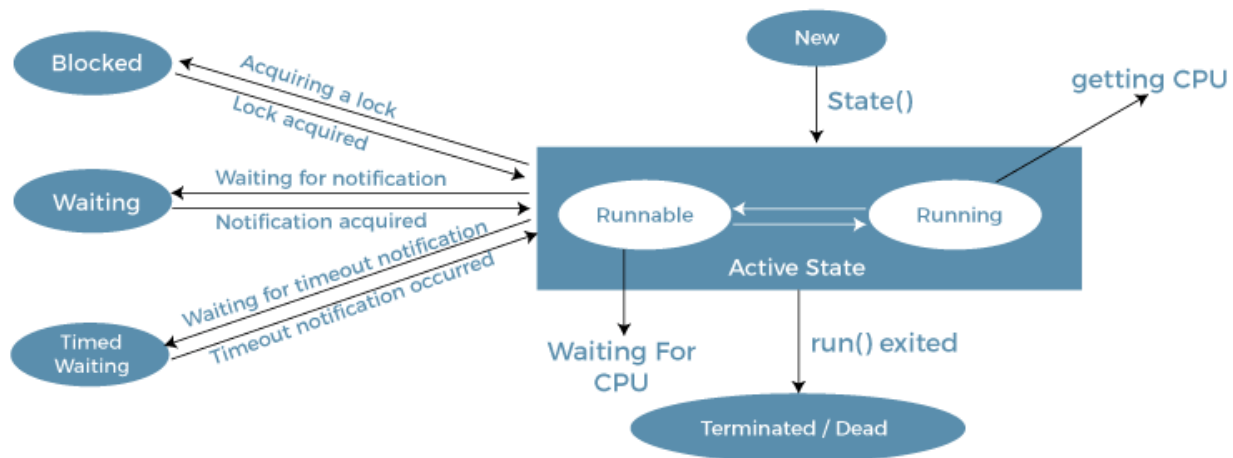
```
thread is running...
```

New: Whenever a new thread is created, it is always in the new state. For a thread in the new state, the code has not been run yet and thus has not begun its execution.

Active: When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is **runnable**, and the other is **running**.

- **Runnable:** A thread, that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time. It is the duty of the thread scheduler to provide the thread time to run, i.e., moving the thread the running state.
A program implementing multithreading acquires a fixed slice of time to each individual thread. Each and every thread runs for a short span of time and when that allocated time slice is over, the thread voluntarily gives up the CPU to the other thread, so that the other threads can also run for their slice of time. Whenever such a scenario occurs, all those threads that are willing to run, waiting for their turn to run, lie in the runnable state. In the runnable state, there is a queue where the threads lie.
- **Running:** When the thread gets the CPU, it moves from the runnable to the running state. Generally, the most common change in the state of a thread is from runnable to running and again back to runnable.

Blocked or Waiting: Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state



Life Cycle of a Thread

A thread is an independent path of execution within a program.

Many threads can run concurrently within a program

Multithreading refers to two or more tasks executing concurrently within a single program.

Every thread in Java is created and controlled by the **java.lang.Thread** class

There are two ways to create thread in java;

- Implement the Runnable interface (java.lang.Runnable)
- By Extending the Thread class (java.lang.Thread)