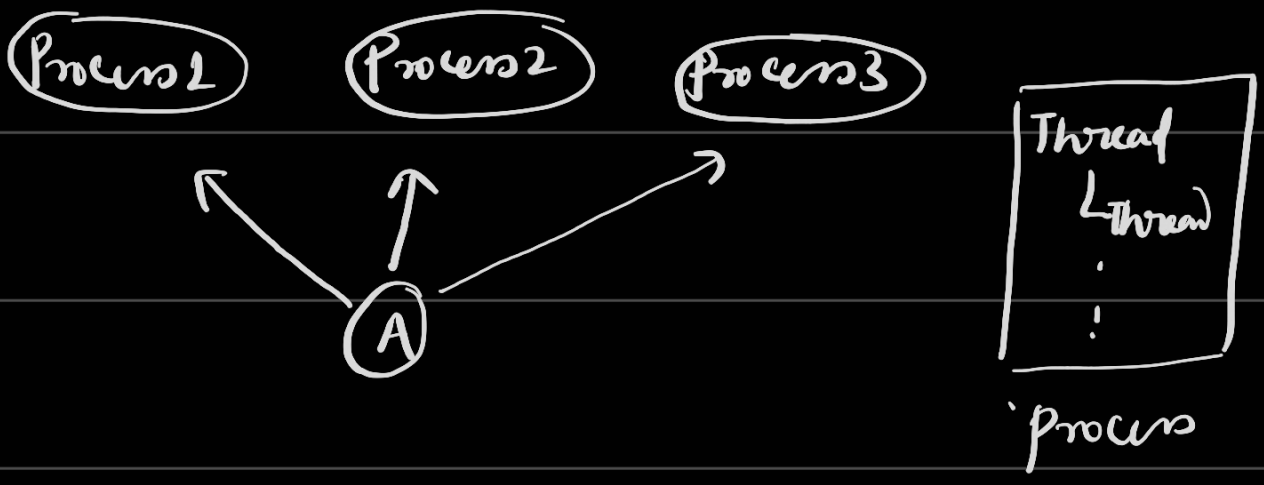
• Multithreading in JAVA –

⇒ Multithreading and multiprocessing both are used to achive multitasking.



In a nut-shell –

- threads use shared memory area.
- threads has faster context switching.
- threads are light weight whereas process are heavy weight.

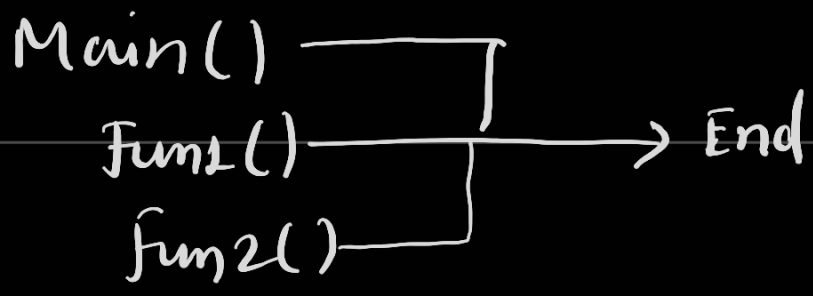
eg - When we write on a board then spell checking, indentation, saving are running in background by threads.

Flow of Control in JAVA :-

- Without threading :-

Main() ⇒ Fun1() ⇒ Fun2() ⇒ End

- With threading :-



In threading threads runs concurrently.

Creating a threading :-

There are two ways :-

① By extending thread class -

② By implementing Runnable interface.

creating Thread by extending Thread class -

```
class MyThread1 extends Thread {
    @Override
    public void run(){
        int i=0;
        while (i<100) {
            Sout("Thread 1 is running");
            i++;
        }
    }
class My Thread2 extends Thread{
        // code exactly same as above
}
```

⇒ main method -

```
MyThread1 t1 = new Thread();
MyThread2 t2 = new Thread();
t1.start();
t2.start();
```

Creating Thread by implementing Runnable interface

```
class T1 implements Runnable {
    @Override
    public void run(){
        int i=0;
        while (i<=1000){
            Sout("Thread1");
            i++;
        }
    }
}

class T2 implements Runnable {
    @Override
    //same as above
}
```

- Main Method -

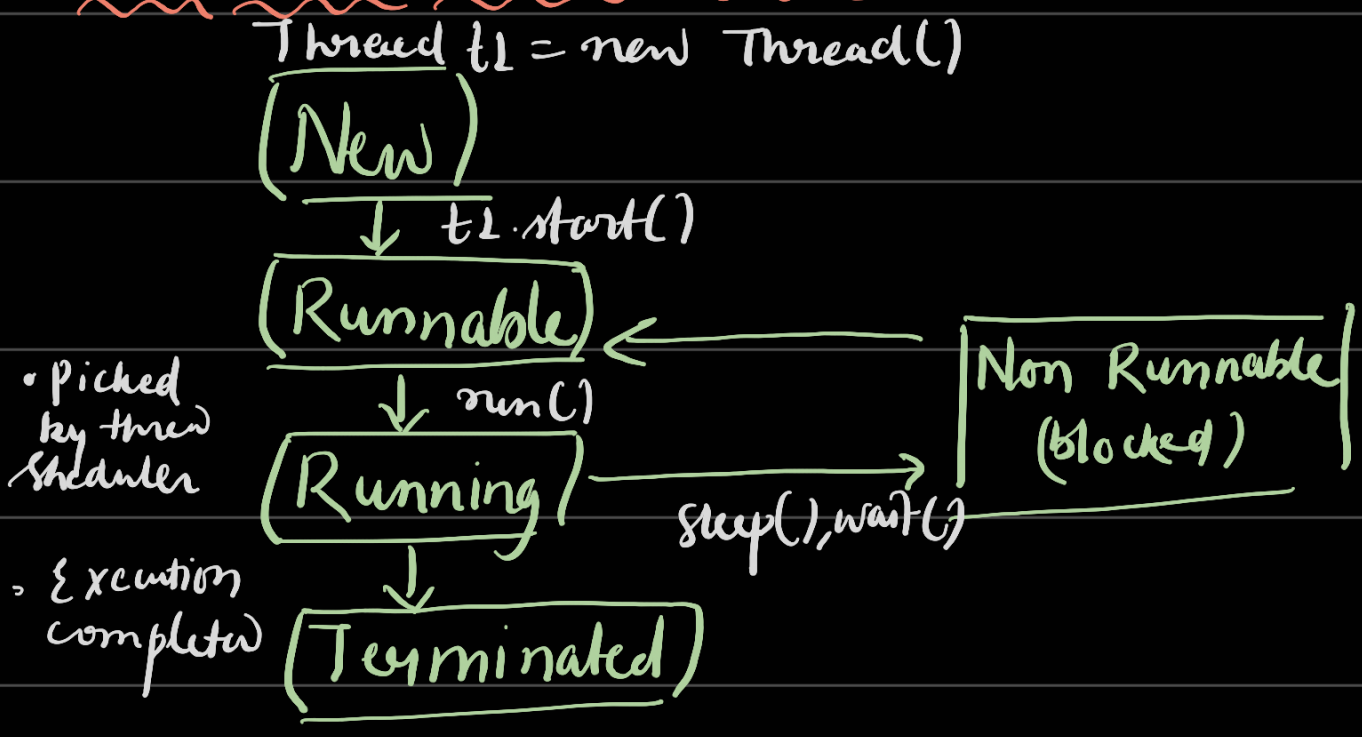
```
T2 bullet1 = new T2();
Thread gun1 = new Thread(bullet1);

T2 bullet2 = new T2();
Thread gun2 = new Thread(bullet2);
gun1.start(); gun2.start();
```

Runnable interface is preferred over extending the Thread class because -

i) Multiple inheritance is not supported in JAVA, it is impossible to extend thread class if class has already extend some other class.

ii) More memory is required while extending the Thread Class, because each thread creates a unique object.

Java Thread Life Cycle :-



Constructors from Thread class in JAVA

- Commonly used constructors -

① Thread()
② Thread (String)
③ Thread (Runnable r)
④ Thread (Runnable r, String name)

eg -

```
class MyThr extends Thread {
    public MyThr (String name) {
        super(name);
    }
    public void run(){
        sout("Thank you");
    }
}
```

main method –

```
MyThr t1 = new MyThr ("Ajay");
t1.start();
```

- To get id of the thread –

```
t1.getId();
```

- To get name of the thread –

```
t2.getName();
```

Java Thread Priorities –

=> The programmer can explicitly assign the priority to the thread. Otherwise, JVM automatically assigns priority while creating the thread.

- Integer constants defined in the Thread class –

(i) MIN_PRIORITY :- Minimum priority, value = 1

(ii) NORM_PRIORITY :- Default priority, value = 5

(iii) MAX_PRIORITY :- Maximum priority, value = 10.

Priority Methods –

① setPriority(range) :- range [0 - 10]

```
t1.setPriority(Thread.MIN_PRIORITY);
```

② getPriority() :-

```
sout(t1.getPriority());
```

Java Thread Methods :-

join() method –

- join() method in java allows one thread to wait until the execution of some other specified thread is complete.
- join(millis) (x) ⇒ I will wait for x second then I will leave.

Syntax –

```
try{
    t1.join();
}
catch(Exception e){
    sout(e);
}
```

Sleep() method -

→ The sleep() method in Java is useful to put a thread to sleep for a specified amount of time.

Syntax - Thread.sleep(long milliseconds)

```
try {
    for (int i=1;  i<=5;  i++){
        Thread.sleep(2000);
        sout(i);
    }
}
catch (Exception e){
    sout(e);
}
}
```

Interrupt() method –

→ A thread in sleeping or waiting state can be interrupted by another thread by interrupt() method of the Thread class.

→ The interrupt() method throws InterruptedExecution.

→ The interrupt() method will not throw InterruptedException if the thread is not in the sleeping/blocked state.

eg –

```
try {
    for (int i=0; i<5; i++){
        sout(i);
        Thread.sleep(4000);
    }
}
catch(InterruptedException e){
    sout("Child Interrupted Thread");
}
```

```
// t1.start();
t1.interrupt();
```