sleep():

This **method** causes the currently executing **thread to sleep** for the specified number of **milliseconds.**

```
//  sleep for the specified number of milliseconds
public static void sleep(long millis) throws InterruptedException

//sleep for the specified number of milliseconds plus nano seconds
public static void sleep(long millis, int nanos)  throws InterruptedException
```

```java
public class Eg7 {

public static void main(String[] args) throws InterruptedException {

for (int i = 0; i < 5; i++) {
System.out.println(i);
Thread.sleep(2000);
}
}

}
```

join Method

java.lang.Thread class provides the **join() method.**
Which allows **one thread to wait** until **another thread completes its execution.**

There are three overloaded join Methods.

public final void join()
It will put the current thread on wait, until the thread which it is called is dead.
If thread is interrupted then it will throw **InterruptedException.**

public final synchronized void join(long millis)
It will put the current thread on wait until the thread which it is called is dead or wait for specified time **(milliseconds).**

public final synchronized void join(long millis, int nanos)
It will put the current thread on wait until the thread which it is called is dead or wait for specified time **(milliseconds + nanos).**

In the example we can see clearly

second thread **t2** starts after first thread **t1** has died
and **t3** will start its execution after second thread **t2** has died

```
public class Eg8 extends Thread {

public void run() {
for (int i = 0; i < 5; i++) {
try {
Thread.sleep(1000);
System.out.println(i);
} catch (InterruptedException e) {
e.printStackTrace();
}
}
}

public static void main(String[] args) throws InterruptedException {

Eg8 eg1 = new Eg8();
Eg8 eg2 = new Eg8();
Eg8 eg3 = new Eg8();
eg1.start();
Thread.currentThread().getName();
// eg1.join();
eg2.start();
Thread.currentThread().getName();
// eg2.join();
eg3.start();
Thread.currentThread().getName();
// seg3.join();
}
}
```

yeild Method

yield() Method will **pause** the **current executing Thread** for giving the chance of remaining **waiting Threads** of **same priority.**

If **any thread** executes **yield method,**
thread scheduler checks if there is any **thread** with **same or high priority** than this **thread.**
If **processor** finds any **thread** with **higher or same priority**
then it will move the **current thread to Ready/Runnable state**
and **processor** give chance to **other thread**
and if not – **current thread** will keep executing.

public static native void yield()

Note:
So, sometimes even after using yield() method, you may not notice any different output

```java
class A extends Thread{

@Override
public void run() {

for (int i = 0; i < 5; i++) {
System.out.println("Thread A“ + i);
}

}
}
```

```java
class B extends Thread{

@Override
public void run() {

for (int i = 0; i < 5; i++) {
System.out.println("Thread B“ + i);
}

}
}
```

```java
public class Client {

public static void main(String[] args) {

Thread a = new  A();
Thread b = new  B();

a.setPriority(Thread.MIN_PRIORITY);
b.setPriority(Thread.MAX_PRIORITY);

a.start();
b.start();
}
}
```

```
Thread B 0
Thread B 1
Thread B 2
Thread B 3
Thread B 4
Thread A 0
Thread A 1
Thread A 2
Thread A 3
Thread A 4
```

```
class A extends Thread{

@Override
public void run() {

for (int i = 0; i < 5; i++) {
System.out.println("Thread A" + i);
Thread.yield();
}

}
}
```

```
class B extends Thread{

@Override
public void run() {

for (int i = 0; i < 5; i++) {
System.out.println("Thread B " + i);
}

}
}
```

```
public class Client {

public static void main(String[] args) {

Thread a = new  A();
Thread b = new  B();

a.setPriority(Thread.MIN_PRIORITY);
b.setPriority(Thread.MAX_PRIORITY);

a.start();
b.start();
}
}
```

```
Thread B 0
Thread B 1
Thread B 2
Thread B 3
Thread B 4
Thread A 0
Thread A 1
Thread A 2
Thread A 3
Thread A 4
```