```java
public static void main(String[] args) {
    // 여기서는 스레드로 만들 내용을 두개 준비한다.
    Thread t1 = new Thread(new SecondThreadTest( name: "t1"));
    Thread t2 = new Thread(new SecondThreadTest( name: "t2"));

    // 스케쥴러는 2개의 스레드를 등록한다.
    t1.start();
    t2.start();

    // 현재 아직 대장 프로세스에게 할당된 시간이 남아있으므로
    // 대장 프로세스는 여기까지 실행하게 된다.
    System.out.println("main()프로세스 실행중");

    //main()프로세스 실행중
    //t2:88
    //t1:55
    //t1:66
    //t2:11

    // 위의 결과에서 파악 해야 하는 내용이
    // 1. 실제 스레드를 실행한다고 바로 실행되는 것이 아니라는 점
    // 2. 현재 sleep()이 0.5초인데
    //    스케쥴링은 압도적으로 빠른 속도로 일어난다.
    //    그런데 왜 t1이 2번 왔는가?
    //   ps-ef를 했을때 나오는 여러 프로세스들과 경쟁을 하다가
    //   채택된 것이 t1일뿐(이것은 상황에 따라 결과가 변할 수 있다.)
}
```

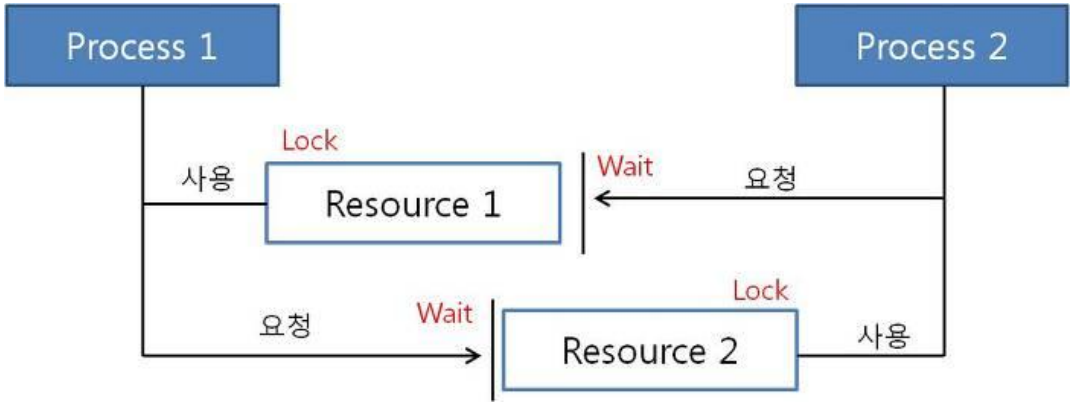
<-스레드의 결과값은 여러 프로세스들의 경쟁으로
t1 ,t2가 랜덤으로 나온다고 했지만 아래 ThridThreadTest.java에서 결과값은 계속 t1,t2로 고정되서 진행되는건가요?
네모칸 try sleep을 전체 스레드 완료후 진행하면 랜덤값이 나오지만 중간에 sleep시키면

```java
public static void main(String[] args) {
    Thread t1 = new Thread(new ThirdThreadTest( name: "t1"));
    Thread t2 = new Thread(new ThirdThreadTest( name: "t2"));

    // 데몬프로세스:init(1) 프로세스가 직접 개입하여
    //           심판을 내리기 전까지는 죽지 않는 불멸의 프로세스
    //          (그래서 서비스 운영에서 밥먹듯이 사용됨)
    t2.setDaemon(true);

    t1.start();

    // main()을 sleep() 하고자 한다면
    // 반드시 Thread의 동작을 모두 완료한 이후에 sleep()시키도록 한다.
    try {
        Thread.sleep( millis: 800);
    }catch (InterruptedException e){}


    t2.start();
    System.out.println("main() 실행중");
}
}
```

서로 경쟁으로 하더라도 t1 ,t2 t3순으로
고정시켜서 진행할수도 있는건가요?

데드락 (Dead lock) 이란?

'교착 상태' 라고도 하며 한정된 자원을 여러 곳에서 사용하려고 할 때 발생할 수 있다.



- 멀티 프로그래밍 환경에서 한정된 자원을 사용하려고 서로 경쟁하는 상황이 발생 할 수 있다.

- 어떤 프로세스가 자원을 요청 했을 때 그 시각에 그 자원을 사용할 수 없는 상황이 발생할 수 있고 그 때는 프로세스가 대기 상태로 들어 간다.

- 대기 상태로 들어간 프로세스들이 실행 상태로 변경 될 수 없을 때 이러한 상황을 교착 상태라 한다.

데드락 (Dead lock)의 발생 조건

- 교착 상태는 한 시스템 내에서 다의 네 가지 조건이 동시에 성립 할 때 발생한다.
- 따라서, 아래의 네 가지 조건 중 하나라도 성립하지 않도록 만든다면 교착 상태를 해결할 수 있다.

상호 배제 (Mutual exclusion)	- 자원은 한 번에 한 프로세스만이 사용할 수 있어야 한다.
점유 대기 (Hold and wait)	- 최소한 하나의 자원을 점유하고 있으면서 다른 프로세스에 할당되어 사용하고 있는 자원을 추가로 점유하기 위해 대기하는 프로세스가 있어야 한다.
비선점 (No preemption)	- 다른 프로세스에 할당된 자원은 사용이 끝날 때까지 강제로 빼앗을 수 없어야 한다.
순환 대기 (Circular wait)	- 프로세스의 집합 {P0, P1, ,…Pn}에서 P0는 P1이 점유한 자원을 대기하고 P1은 P2가 점유한 자원을 대기하고 P2…Pn-1은 Pn이 점유한 자원을 대기하며 Pn은 P0가 점유한 자원을 요구해야 한다.