스레드 기본 - API

이찬형

목차

Contents

01. sleep()

- 지정된 시간동안 현재 스레드의 실행을 일시 정지(대기 상태)

- 시스템 콜(네이티브 메서드)을 통해 커널모드에서 수행 후 유저 모드로 전환

```
//지정된 밀리초만큼 스레드 수면(대기) 상태
staic void sleep(long millis) throws InterruptedException

//지정된 밀리초 + 나노 초 동안 스레드 수면(대기) 상태
staic void sleep(long millis,  int nanos) throws InterruptedException
```

InterruptedException
- 스레드 수면(대기)시 인터럽트 발생
- 예외 발생 시, 스레드는 수면(대기) 상태 → 실행 대기 상태로 전환된다.

Ex).
interrupt() → sleep()
sleep() → interrupt()

01. sleep()

sleep(0)

Thread.currentThread().sleep(0)

- 대기 중인 스레드 중 동일한 우선순위인 스레드 CPU 할당
  → Context Switching, 모드 전환

- 동일한 우선순위 없으면, 다시 현재 스레드에 CPU 할당
  →모드 전환만 발생

VS

sleep(n)

Thread.currentThread().sleep(1)

- 스케줄러 조건 상관없이 대기 상태인 스레드 CPU 할당
  → Context Switching, 모드 전환

- 다른 스레드에게 명확하게 실행을 양보하기 위함은 sleep(1)을 사용해라!

01. sleep()

[작동 방식]

- synchronized 영역에서 sleep()인 스레드는 획득한 모니터나 락을 잃지 않고 계속 유지한다.
→ wait()은 모니터나 락을 놓아버리지만 sleep()는 놓지 않는 차이점이 있다.

- 수면 시간은 OS 스케줄러 및 시스템 기능에 따라 제한되어 정확성을 보장하지 않습니다.
→ 시스템 부하가 많고 적음에 따라 지정한 수면 시간과 차이가 날 수 있다.

02. join()

- 한 스레드가 다른 스레드가 종료될 때까지 실행을 중지하고 대기상태

- 대상 스레드가 종료되면 실행 대기 상태로 전환

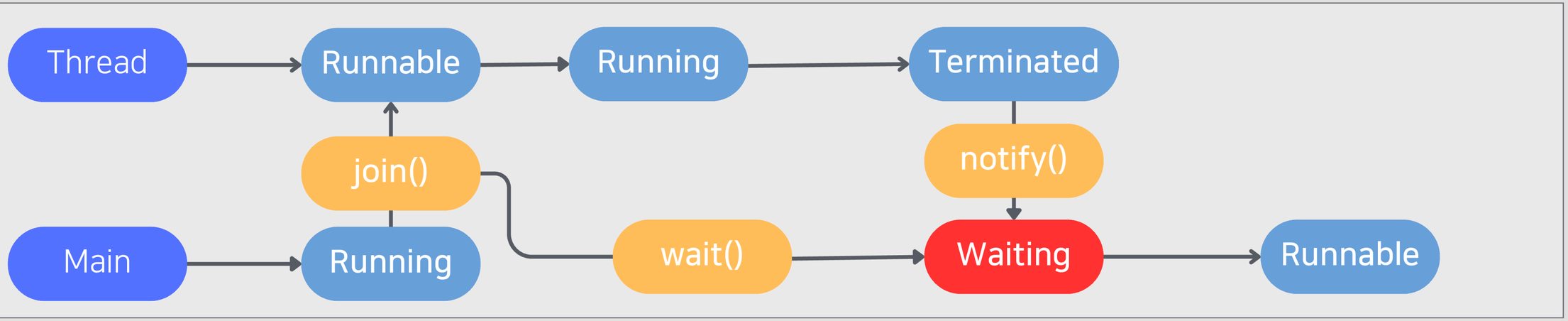
```
//스레드의 작업 종료까지 대기 상태 유지
static  join() throws InterruptedException

//지정된 밀리초 동안 스레드 수면(대기) 상태
static join(long millis) throws InterruptedException

//지정된 밀리초 + 나노 초 동안 스레드 수면(대기) 상태
static join(long millis,  int nanos) throws InterruptedException
```

02. join()

[기본 예제]

```
public static void main(String[] args) {
   Runnable r = new MyRunnable();
   Thread thread = new Thread(r);
   thread.start();
   try {
         thread.join();
   }catch (InterruptedException e) {
   }
}
```

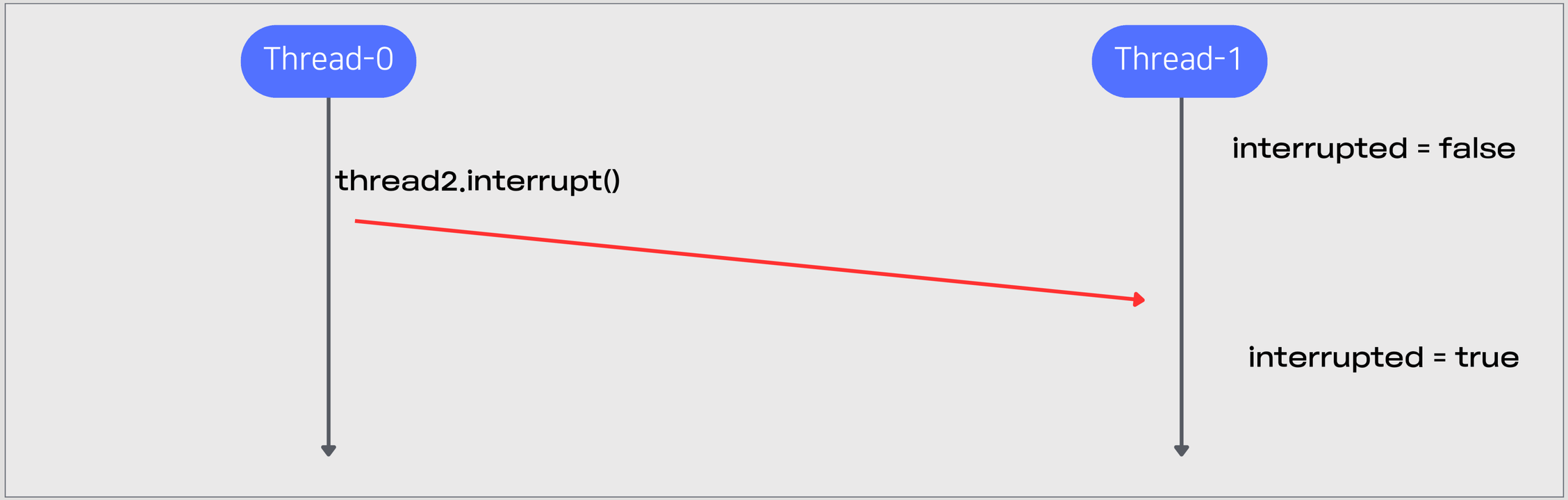
[작동 방식]



03. Interrupt()

- 특정 스레드에게 신호를 전달하여, 실행 중단, 작업 취소, 강제 종료 등으로 사용

- 현재 실행 흐름을 멈추고, 인터럽트 이벤트를 먼저 처리하도록 시그널 보내는 장치



03. Interrupt()

[상태]

- interrupt 상태 값 존재(false, true)

static booelan interrupted()

- 현재 인터럽트 상태를 반환
→ 인터럽트 상태를 false 로 초기화 하므로 인터럽트를 해제하는 역할을 한다.

- 인터럽트를 해제하는 경우 다른 곳에서 스레드에 대한 인터럽트 상태를 체크하는 곳이 있다면 별도의 처리가 필요할 수 있다

- InterruptedException이 발생할 경우 상태를 초기화하는 것과 동일한 효과
→ Thread.sleep(), Thread.join(), Object.wait(),
→ Future.get()(비동기 작업 완료 대기), BlockingQueue.take()(블로킹 연산 Queue 데이터 얻기)

boolean isInterrupted()

- 이 메서드는 스레드의 인터럽트 상태를 변경하지 않고 계속 유지한다.

- 상태를 확인하는 용도로만 사용할 경우 interrupted() 보다 이 메서드를 사용!!

04. name() / currentThread() / isAlive()

[Thread Name]

- 각 스레드의 작업을 파악할 수 있기 때문에 스레드 이름을 정하는 것은 매우 도움이 된다.

[자동 생성]

```
Thread myThread = new Thread(task);
```

→ 메인 스레드 이름 main, 이후 Thread 0 ··· N 순차적 생성 (별도의 setName()으로 n개를 만들면 0 + n번 부터 시작)

[사용자 정의]

```
thread.setName("myThread");
Thread myThread = new Thread([ThreadGroup], [Runnable], "myThread");
```

[이름 조회]

```
thread.getName();
```

04. name() / currentThread() / isAlive()

currentThread()

Thread.currentThread()

if(Thread.currentThread() == thread)

- 현재 실행 중인 스레드 개체에 대한 참조를 반환

→익명 Thread/ Runnable 구현에는
Thread.currentThread()으로 처리를 해야 한다.

isAlive()

thread.isAlive()

-스레드 살아 있는 여부 확인

→ true : 활성화, false : 비활성화

05. Priority

- 자바 런타임은 고정 우선순위 선점형 스케줄링으로 매우 단순하고 결정적인 스케줄링 알고리즘 지원

- 실행 대기 상태의 스레드 중에 상대적인 우선순위에 따라 스레드 예약

- 1 ~ 10까지 존재, 높을수록 우선순위 높다.
→스레드 생성시 Default : 5 우선순위 부여

- 우선 순위 높은 스레드 실행 중 중지/양보/실행 불가능할 때 우선 순위 낮은 스레드 실행 시작

- 우선 순위 같을 경우, 라운드 로빈 스케줄링 방식에 의해 다음 스레드 선택!
→ 여기서 라운드 로빈은 실행 대기 순서(순환 할당)가 아니라, 모두 일정한 타임 슬라이스를 가지고 CPU 할당을 한다는 이야기입니다.

- 반드시 우선순위가 높아도 실행 보장 X
→ OS마다 정책이 다를 수 있음, 기아 상태를 피하기 위해 더 낮은 스레드 선택 가능

05. Priority

[상수]

최소

public static int MIN_PRIORITY

Priority : 1

중간

public static int MORM_PRIORITY

Priority : 5

최대

public static int MAX_PRIORITY

Priority : 10

```
// 1 ~ 10 인수 유효, 우선 순위 변경
void setPriority(int newPriority)

//우선순위 정보 얻기
int getPriority()
```

THANK YOU

감사합니다.