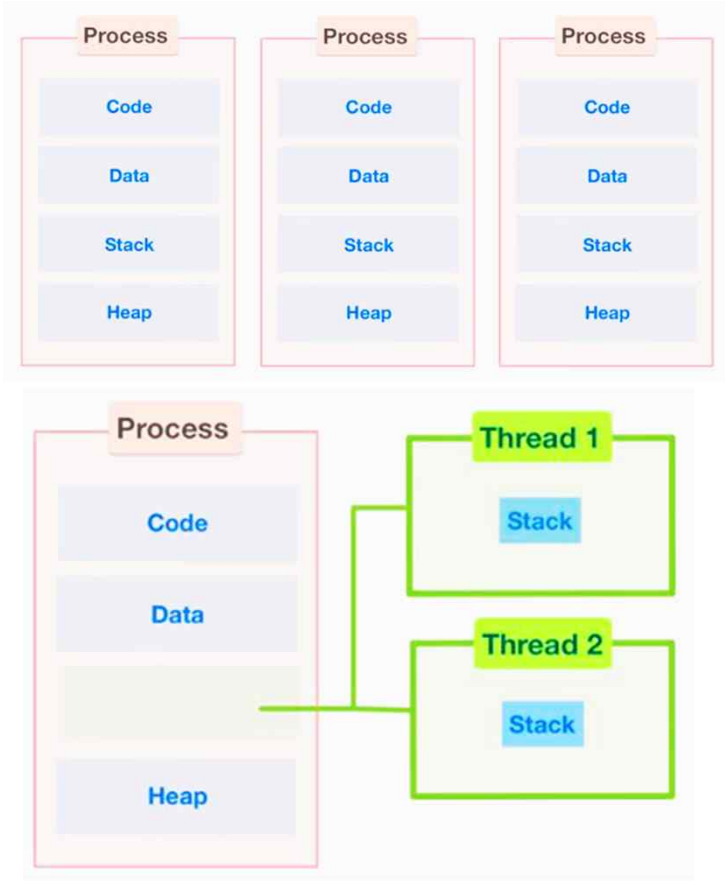
쓰레드
thread

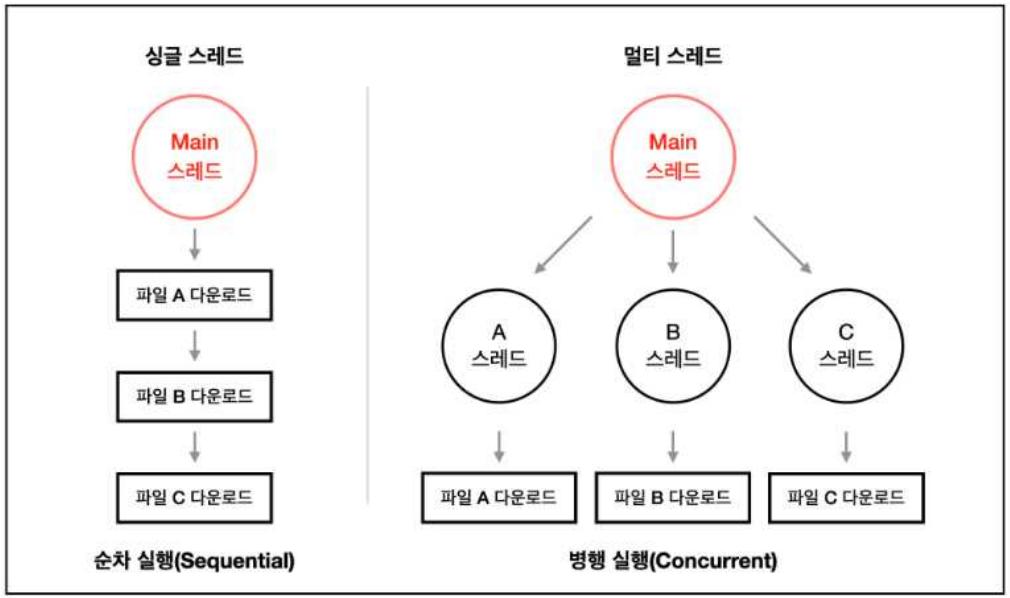
thread의 개념

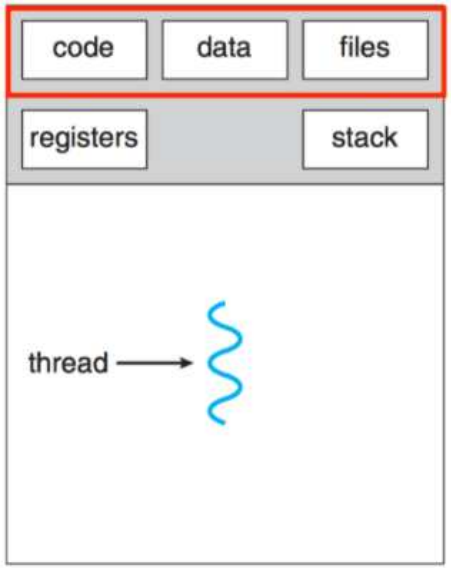
- Program
  - 컴퓨터에서 어떤 작업을 위해 실행할 수 있는 '정적인 상태'의 파일
- Process
  - 프로그램이 실행되서 돌아가고 있는 상태, 컴퓨터에서 연속적으로 실행되고 있는 '동적인 상태'의 컴퓨터 프로그램
  - 운영체제가 메모리 등의 필요한 자원을 할당해 준 '실행중인 프로그램'
- 스레드(thread)
  - 프로세스가 할당받은 자원을 이용하는 실행 단위
  - 프로세스의 특정한 수행 경로
  - 프로세스 내에서 실행되는 여러 흐름의 단위



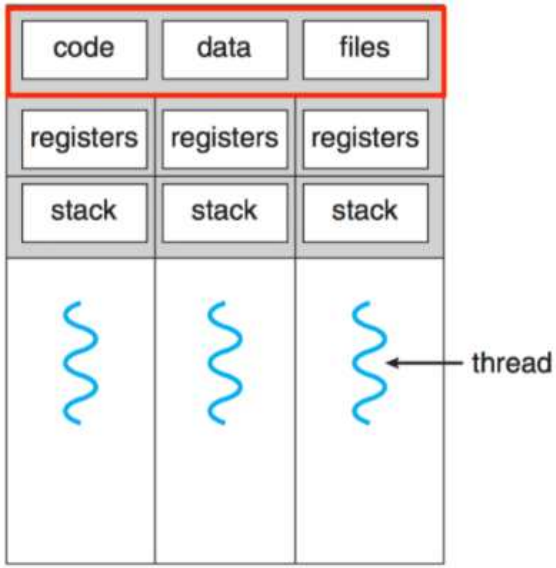
thread

- single thread
- multi-thread





single-threaded process



multithreaded process

Thread

Thread	Single	Multi
장점	• 문맥 교환(context switch) 작업 불필요 • 동기화 불필요 • 자원 소모 코스트가 낮다 • 프로그래밍 난이도가 낮다	• 응답성 • 경제성 • 멀티프로세서의 활용
단점	• 멀티프로세서의 사용불가 • 연산량이 많은 작업으로 인한 딜레이 • 에러 처리 불가 시 작업 종료	• 스레드 생성 시간이 단일 스레드보다 느리다. • 동기화 • 프로그래밍 난이도가 높다 • 자원 소모

Thread의 생성

- Runnable 인터페이스를 구현하는 방법

```
public class Test01 implements Runnable {
    @Override
    public void run() {
                    /* 스레드 실행코드 */
    }
}
```

- Thread 클래스를 상속받는 방법

```
public class Test01 extends Thread{
    @Override
    public void run() {
                    /* 스레드 실행코드 */
    }
}
```

thread 생성자

생성자	내용
Thread()	일반적인 스레드 객체 생성 Thread-n 이런 이름을 가진 스레드가 만들어진다
Thread(Runnable target)	run 메서드를 가지는 객체를 인자값으로 할당하기
Thread(Runnable target, String name)	run 메서드를 가지는 객체와 스레드 이름을 인자값으로 할당하기
Thread(String name)	스레드 생성하면서 스레드 이름 지어주기

thread 메서드

void sleep(long msec) throws Interrupted Exception	msec에 지정된 밀리초 동안 대기
String getName()	스레드의 이름을 s로 설정
void setName(String s)	스레드의 이름을 s로 설정
void start()	스레드를 시작 run() 메소드 호출
int getPriority()	스레드의 우선 순위를 반환
void setpriority(int p)	스레드의 우선순위를 p값으로
boolean isAlive()	스레드가 시작되었고 아직 끝나지 않았으면 true 끝났으면 false 반환
void join() throws InterruptedException	스레드가 끝날 때 까지 대기
void run()	스레드가 실행할 부분 기술 (오버라이딩 사용)
void suspend()	스레드가 일시정지 resume()에 의해 다시시작 할 수 있다.
void resume()	일시 정지된 스레드를 다시 시작.
void yield()	다른 스레드에게 실행 상태를 양보하고 자신은 준비 상태로

사용

- run()메소드가 종료하면 스레드는 종료된다.
- 스레드를 계속 실행 하려면 run()메소드를 무한루프 속에 실행 되어야 합니다.
- 한번 종료한 스레드는 다시 시작할 수 없어 스레드 객체를 다시 생성해야 합니다.

start() vs run()

- start()
  - thread 생성/추가 등의 기본 설정을 실행하고 run()
- run()
  - 입력받은 target:Runnable의 run()을 실행

- run()을 호출하면 Multi-threading (Call Stack 생성 등)도 진행하지 않고, Runnable의 함수를 호출
- start()를 호출하면 native method를 호출한다.

우선순위

- 2개 이상의 스레드가 동작 중일 때 우선 순위를 부여하여 우선 순위가 높은 스레드에게 실행의 우선권을 부여
- 우선 순위를 지정하기 위한 상수
  - static final int MAX_PRIORITY : 우선순위 10 - 가장 높은 우선 순위
  - static final int MIN_PRIORITY : 우선순위 1 - 가장 낮은 우선 순위
  - static final int NORM_PRIORITY : 우선순위 5 - 보통의 우선 순위
- 스레드 우선 순위는 변경 가능
  - void setPriority(int priority)
  - int getPriority()
  - main()스레드의 우선 순위 값은 초기값이 5
- JVM의 스케쥴링 규칙
  - 철저한 우선 순위 기반
  - 가장 높은 우선 순위의 스레드가 우선적으로 스케쥴링
  - 동일한 우선 순위의 스레드는 돌아가면서 스케쥴링(라운드 로빈 방식)

Thread 동기화

- 동기화(synchronized)
  - 하나의 작업이 완전히 완료된 후 다른 작업 수행
  - 비동기식: 하나의 작업 명령 이후(완료와 상관없이) 바로 다른 작업 명령을 수행
- Thread A가 실행중(Running)에 synchronized 선언된 블록을 만나게 되면 object's Lock Pool 로 이동하고 락(lock)을 획득
- 다른 Thread는 Running 중 Blocked 되어 실행가능한(Runnable) 상태
- Thread A의 synchronized 블록 실행이 종료된다면 A는 락을 해제
- A가 락을 해제하면 다른 스레드 B가 실행가능한(Runnable)상태에서 실행(Running)
- 만약 스레드 B가 synchronized 선언된 블록을 만나면 B가 object's Lock Pool 로 이동하고 락(lock)을 획득한다. 이후 과정은 앞과 동일

임계영역 설정

- 동기화 메서드
  - 접근제한자와 리턴형태 사이에 synchronized 키워드를 명시

```
public synchronized void methodA(){
}
```

- 동기화 블록
  - 임계영역으로 설정할 부분을 synchronized(공유객체){ } 코드로 포함
  - 메서드 전체를 동기화하기에 코드가 많거나 특정 부분에만 동기화가 필요한 경우에 사용하는 방법

```
public void methodA(){
        synchronized(공유객체){
                //공유객체가 자기자신이라면 () 안에 키워드 this
        }
}
```

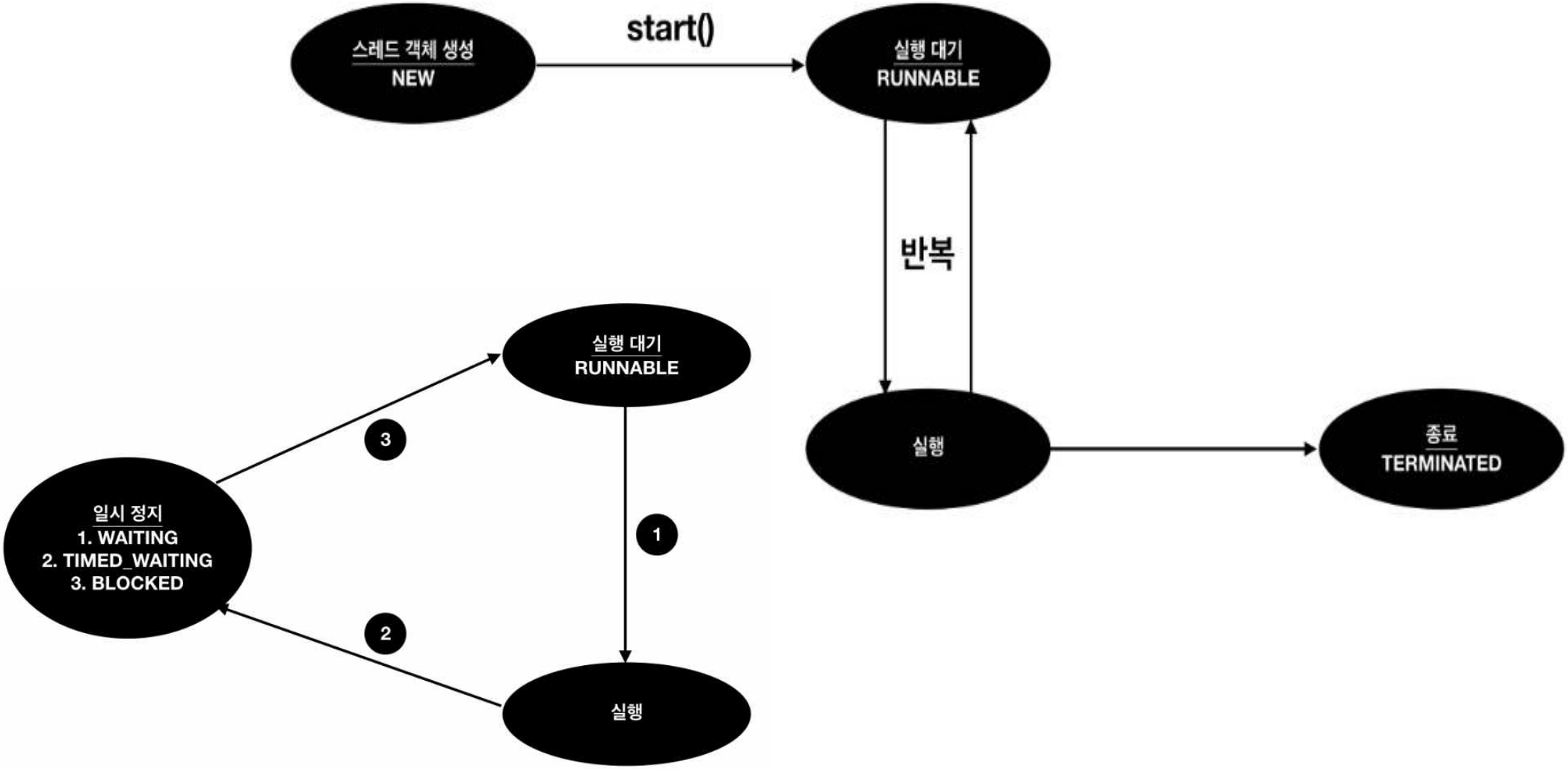
동기화의 방법

- 동시에 두 개의 Thread가 동기화 메서드 사용불가
- 동시에 두 개의 Thread가 동기화 블록 사용불가

thread의 state

- getState() : 스레드의 상태를 알 수 있도록 해주는 메소드
- Thread.State 열거 상수

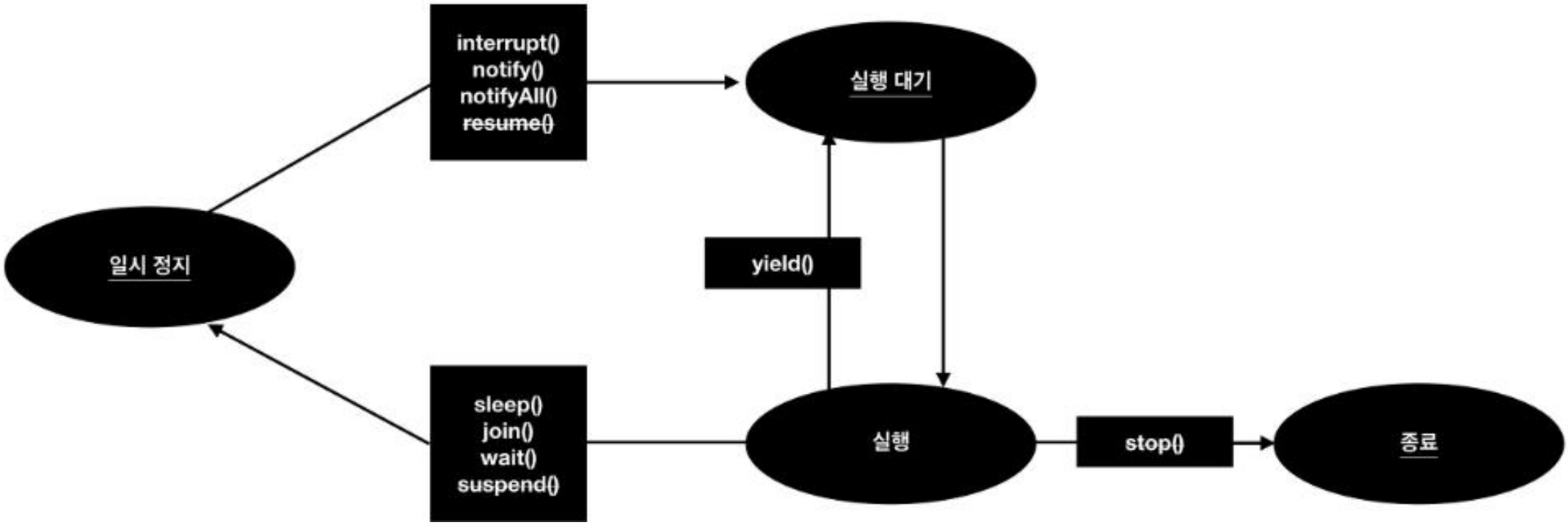
상태	열거 상수	설명
객체 생성	NEW	스레드 객체가 생성, 아직 start() 메소드가 호출되지 않은 상태
실행 대기	RUNNABLE	실행 상태로 언제든지 갈 수 있는 상태
일시 정지	WAITING	다른 스레드가 통지할 때까지 기다리는 상태
	TIMED_WAITING	주어진 시간 동안 기다리는 상태
	BLOCKED	사용하고자 하는 객체의 락이 풀릴 때까지 기다리는 상태
종료	TERMINATED	실행을 마친 상태

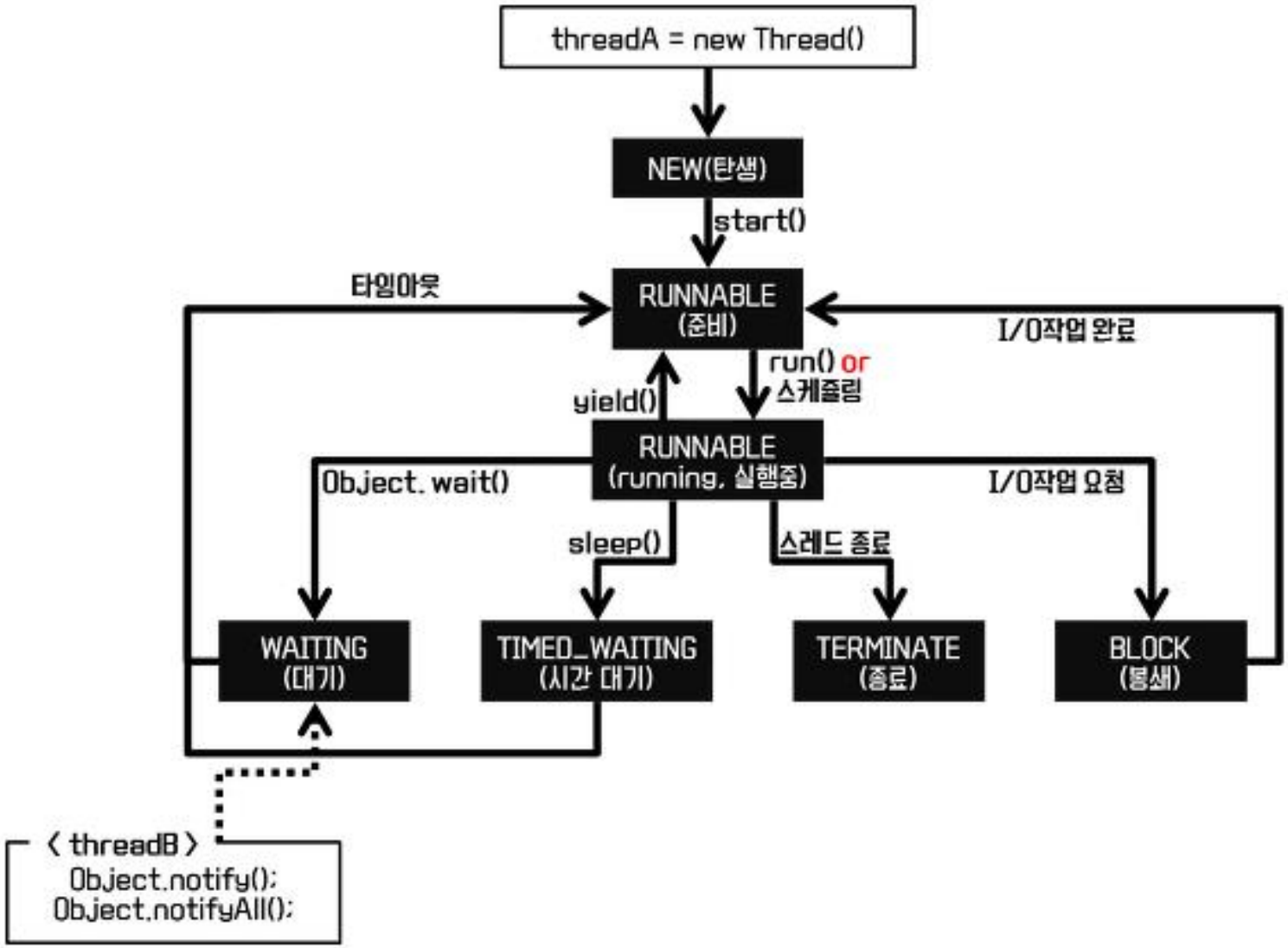

스레드 객체 생성
NEW
start()
실행 대기
RUNNABLE
반복
실행
종료
TERMINATED
실행 대기
RUNNABLE
3
일시 정지
1. WAITING
2. TIMED_WAITING
3. BLOCKED
1
2
실행

쓰레드의 상태 제어

메소드	설명
interrupt()	일시 정지 상태의 스레드에서 InterruptedException 예외를 발생시켜, 예외 처리 코드(catch)에서 실행 대기 상태로 가거나 종료 상태로 갈 수 있도록 한다.
notify() notifyAll()	동기화 블록 내에서 wait() 메소드에 의해 일시 정지 상태에 있는 스레드를 실행 대기 상태로 만든다.
resume()	suspend(0 메소드에 의해 일시 정지 상태에 있는 스레드를 실행 대기 상태로 만든다. - Deprecated (대신 notify(), notifyAll() 사용)
sleep(long millis) sleep(long millis, int nanos)	주어진 시간 동안 스레드를 일시 정지 상태로 만든다. 주어진 시간이 지나면 자동적으로 실행 대기 상태가 된다.
join() join(long millis) join(long millis, int nanos)	join() 메소드를 호출한 스레드는 일시 정지 상태가 된다. 실행 대기 상태로 가려면, join() 메소드를 멤버로 가지는 스레드가 종료되거나, 매개값으로 주어진 시간이 지나야 한다.
wait() wait(long millis) join(long millis, int nanos)	동기화(synchronized) 블록 내에서 스레드를 일시 정지 상태로 만든다. 매개값으로 주어진 시간이 지나면 자동적으로 실행 대기 상태가 된다. 시간이 주어지지 않으면 notify(), notifyAll() 메소드에 의해 실행 대기 상태로 갈 수 있다.
suspend()	스레드를 일시 정지 상태로 만든다. resume() 메소드를 호출하면 다시 실행 대기 상태가 된다. - Deprecated (대신 wait() 사용)
yield()	실행 중에 우선순위가 동일한 다른 스레드에게 실행을 양보하고 실행 대기 상태가 된다.
stop()	쓰레드 즉시 종료 – Deprecated

Thread State Control



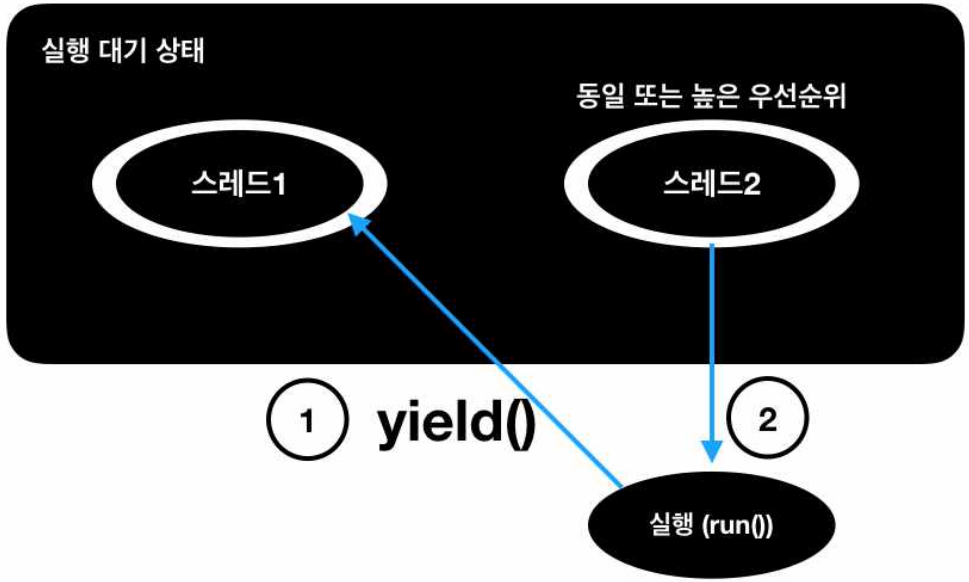

threadA = new Thread()
NEW(탄생)
start()
타임아웃
RUNNABLE
(준비)
I/O작업 완료
run() or
스케쥴링
yield()
RUNNABLE
(running, 실행중)
Object. wait()
I/O작업 요청
sleep()
스레드 종료
WAITING
(대기)
TIMED_WAITING
(시간 대기)
TERMINATE
(종료)
BLOCK
(봉쇄)
〈 threadB 〉
Object.notify();
Object.notifyAll();

sleep()

- Thread.sleep()
- 호출한 스레드는 주어진 시간 동안 일시 정지 상태가 되고, 다시 실행 대기 상태로 돌아갑니다.
  - 매개 값에는 얼마 동안 일시 정지 상태로 유지한 것인지 설정
    - 밀리세컨드(1/1000) 단위
    - Thread.sleep(3000); ->3초
  - 일시 정지 상태에서 주어진 시간이 되기 전에 interrupt() 메소드가 호출되면 InterruptedException이 발생하기 때문에 예외 처리가 필요

yield()

- Thread.yield()
- 실행 되고 있는 스레드 주체가 혼자만 작동하지 않고, 다른 스레드가 실행 기회를 양보



join()

- 스레드는 다른 스레드와 독립적으로 실행하는 것이 기본입니다.

하지만 다른 스레드가 종료될 때까지 기다렸다가 실행해야 하는 경우가 발생할 수도 있습니다.

- 다른 스레드가 join() 메서드를 호출, 그 스레드가 종료될 때까지 대기

wait(), notify(), notifyAll()

- java.lang.Object 클래스에 정의된 메서드
- 위 메서드들은 동기화(Synchronized) 처리된 코드에서 사용 가능
- wait()
  - 시간 인자값을 넣으면 주어진 시간 이후에 자동으로 실행대기 상태로 전환
- notify()
  - wait()로 일시정지 중인 스레드 1개를 실행대기 상태로 전환
- notifyAll()
  - wait()로 일시정지 중인 다른 모든 스레드를 실행대기 상태로 전환
  - 실행되는 것은 1개

Deamon Thread

- 데몬(Daemon)
  - 멀티태스킹 운영체제에서 사용자가 직접 제어하지 않고, 백그라운드에서 돌아가면서 작업을 하는 프로그램
- 일반 쓰레드가 모두 종료되면 데몬쓰레드는 강제로 종료
- 일반적으로 가비지 컬렉터, 화면 자동 갱신 등의 작업을 실행
- 무한 루프와 조건문을 이용해서 무한히 대기하고 있다가 특정 조건이 만족되면 작업을 실행하고, 다시 대기