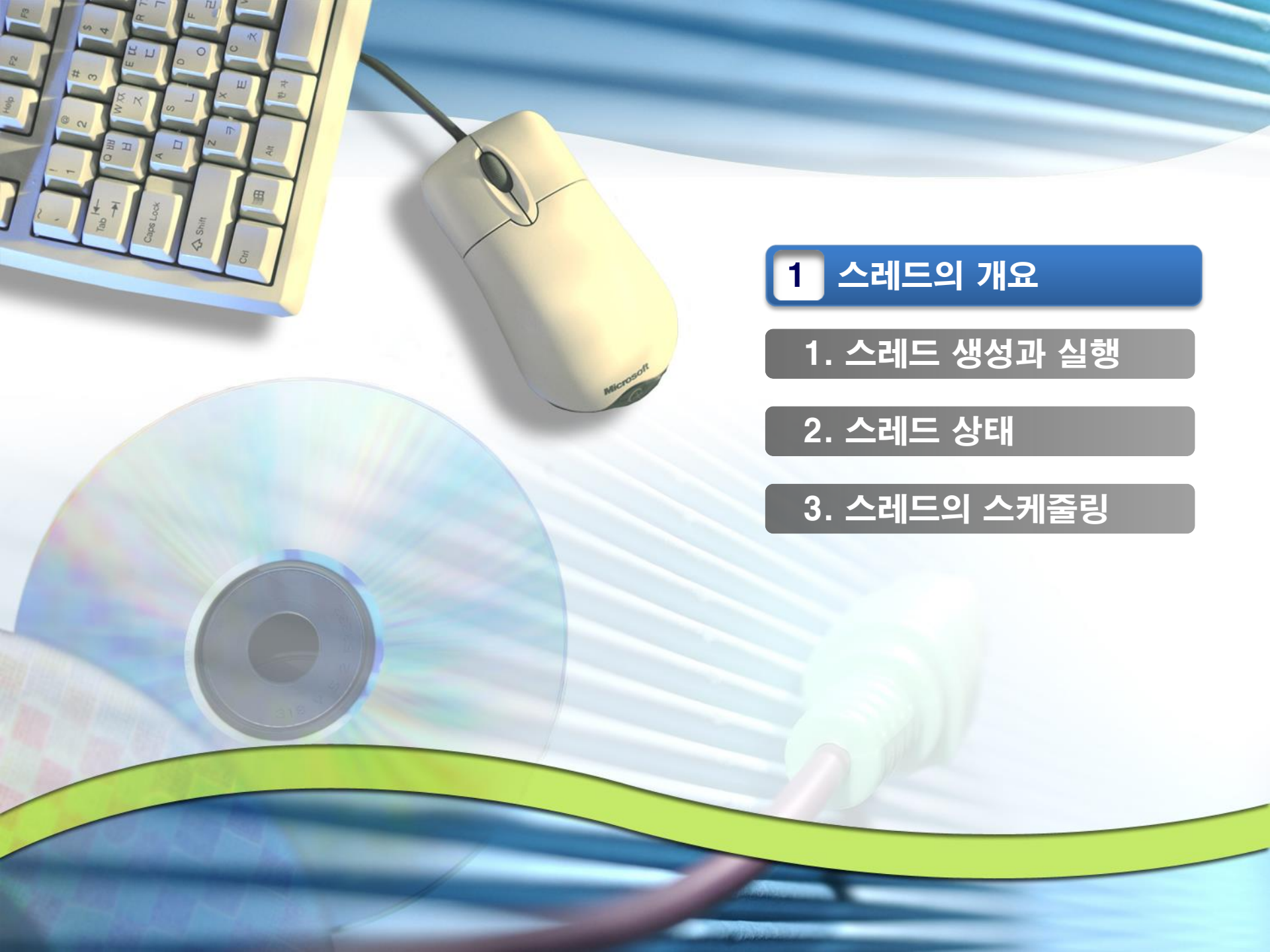

JAVA
스레드



1 스레드의 개요

1. 스레드 생성과 실행

2. 스레드 상태

3. 스레드의 스케줄링

프로세스(process)

- 프로세스(Process)

: OS로부터 자원을 할당받아 동작하는 독립된 프로그램을 의미한다. 두 개 이상의 프로세스가 실행되는 것을 멀티 프로세스라 하며 그 멀티 프로세스를 실행하여 일을 처리하는 것을 멀티 태스킹(두 가지 이상의 작업을 동시에 처리하는 것)이라 한다.

- 멀티 프로세스

: 독립적으로 프로그램들을 실행하고 여러 가지 작업 처리

- 멀티 스레드

: 한 개의 프로그램을 실행하고 내부적으로 여러 가지 작업 처리

멀티 태스킹

병렬작업(multi-tasking)은 여러 개의 작업을 동시에 실행시켜서 컴퓨터 시스템의 성능을 높이기 위한 기법이다.

음악을 들으면서
운동을 할 수 있다.

인쇄를 하면서
문서 편집을 할 수 있다.

운영체제가 CPU의 시간을 쪼개서 각 작업에 할당하여서 작업들이 동시에 수행되는 것처럼 보이게 하기 때문이다.

멀티 태스킹

동시에 여러 가지 일을 병행해서 하는 멀티태스킹은 프로세스 기반과 스레드 기반 두 가지 유형이 있다. 프로세스란 실행 중인 프로그램을 의미하며 미디어 프로그램, 문서작성 프로그램, 메신저 프로그램을 동시에 실행하는 작업은 프로세스기반의 멀티태스킹이다.

프로세스 기반은 여러 프로그램이 병행 실행되는 것이지만, 스레드 기반은 하나의 프로그램 내에서 여러 작업을 병행하는 것이다. 하나의 프로그램 내에서 동시에 실행되는 작업단위를 스레드라고 한다.

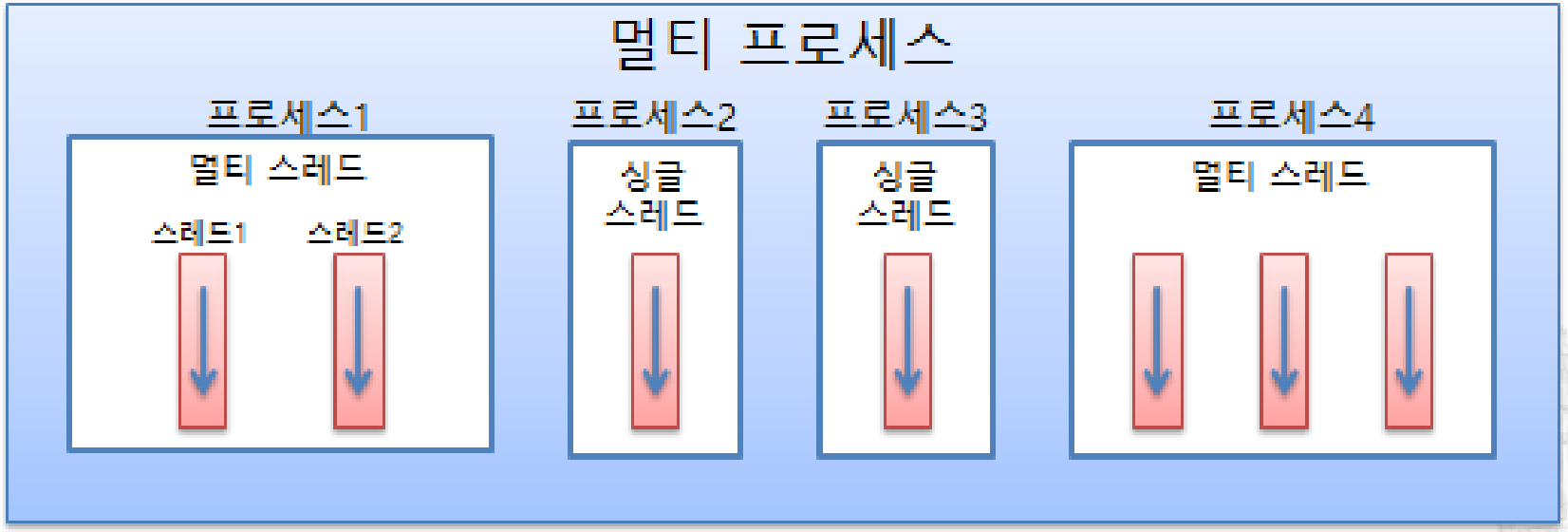
스레드란?

스레드란 프로세스 내에서 실행되는 세부 작업 단위이다.

멀티 스레딩(multi-threading)은 하나의 프로그램이 동시에 여러 가지 작업을 할 수 있도록 하는 것이다.

각각의 작업은 스레드(thread)라고 불린다.

- 싱글 스레드 프로세스(Single-Thread Process) : 프로세스 내부에 하나의 스레드가 동작
- 멀티 스레드 프로세스(Multi-Thread Process) : 프로세스 내부에 여러 개의 스레드가 동작



스레드란?

우리가 목표로 하는 것은 프로그램 단위의 멀티태스킹이 아니라 하나의 프로그램 내에서 실행되는 멀티메소드이다.

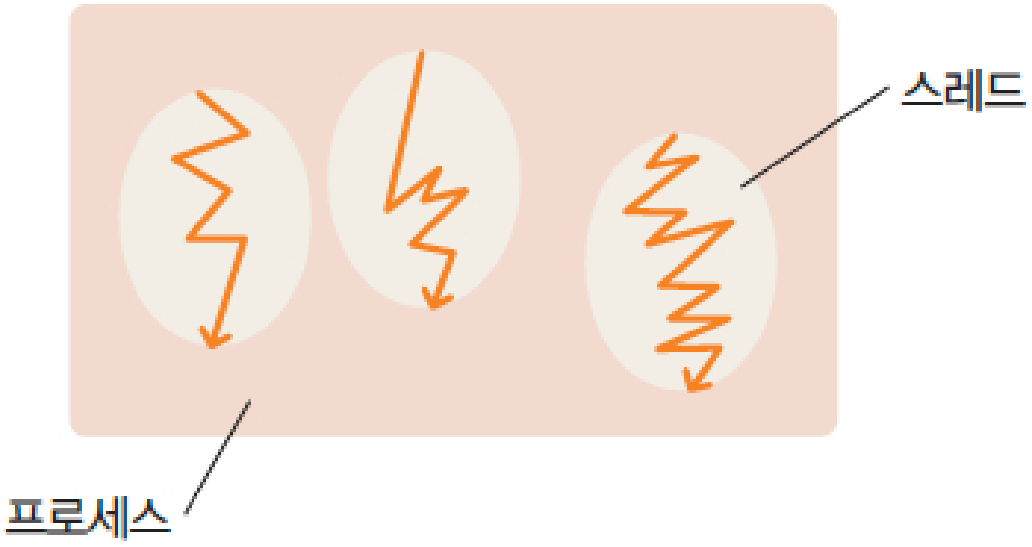
즉 하나의 프로세스 내에서 여러 개의 메소드를 동시에 실행하기를 원한다는 것이다. 그래서 두 개의 메소드가 존재하고 서로 독립된 메소드로써 동시에 작업을 진행할 수 있다면 이때 이 메소드들을 스레드라고 부를 수 있다.

그래서 프로세스의 경우 한 순간에 하나의 메소드만이 실행되지만 스레드의 기법을 이용하면 여러 개의 메소드를 동시에 실행 시킬 수 있다.

프로세스와 스레드

가장 기본적인 차이점

- 프로세스는 같은 프로그램을 여러 개 실행한다고 해도, 그들이 할당 받은 자원은 서로 공유하지 않는다는 점이다.
- 스레드는 스레드끼리 자원을 공유하며 실행 할 수 있다는 특징이 있다.



프로세스와 스레드

- 프로세스

프로세스는 독자적으로 실행이 가능한 환경이다. 프로세스는 일반적으로 완전한 자기만의 자원을 가진다. 특히 각각의 프로세스의 메모리 공간은 분리되어 있다. 프로그램이 실행되면 프로세스가 된다. 대부분의 자바 가상 기계는 하나의 프로세스로 실행된다.

프로세스와 스레드

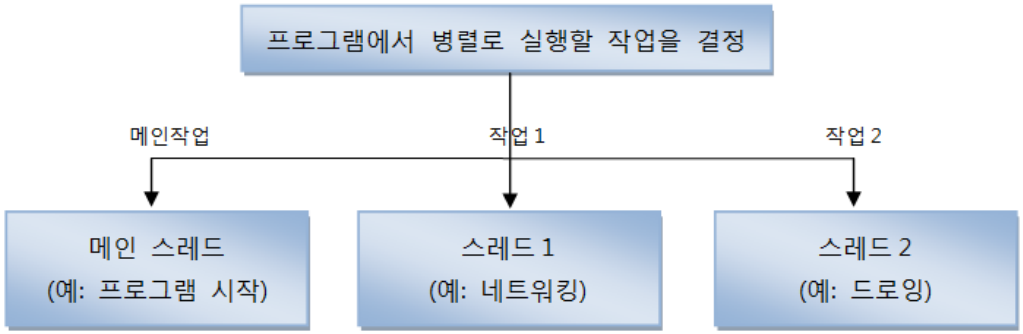
- 스레드

완벽하게 독립적으로 실행되는 프로세스가 아니고 하나의 프로세스 안에서 동작하는 작은 프로세스이기 때문에 “경량 프로세스(Light Weight Process) " 라고도 한다.

스레드들은 프로세스 안에 존재한다. 스레드들은 메모리와 파일을 포함하여 프로세스의 모든 자원을 공유한다. 모든 자바 프로그램은 적어도 하나의 스레드를 가고 있다. 즉 main() 스레드라고 부르는 하나의 스레드로 출발한다.

프로세스와 스레드

- 메인(main) 스레드
  - 모든 자바 프로그램은 메인 스레드가 main() 메소드 실행하며 시작
  - main() 메소드의 첫 코드부터 아래로 순차적으로 실행
  - 실행 종료 조건
    - 마지막 코드 실행
    - return 문을 만나면
  - 메인 스레드는 자바 프로그램 시작 시 자동으로 생성되며 main() 메서드를 실행한다. main() 메서드의 명령문 실행이 완료되면 메인 스레드는 종료되고 자바 프로그램도 종료된다.
- 멀티 스레드로 실행하는 어플리케이션 개발



프로세스와 스레드

- 스레드 프로그램을 할 때 주의해야 할 사항

① 우선권 : 어떤 메소드에게 작업할 권한을 많이 줄 것인지에 대한 것

② 동기화 : 공유자원을 상대로 순서대로 작업이 이루어지는 것을 동기화가 보장된다고 말한다.

스레드 사용의 예

스레드 사용의 예는 다음과 같다.

- 웹 브라우저에서 웹 페이지를 보면서 동시에 파일을 다운로드할 수 있도록 한다.
- 워드 프로세서에서 문서를 편집하면서 동시에 인쇄한다.
- 게임 프로그램에서는 응답성을 높이기 위하여 많은 스레드를 사용한다.
- GUI에서는 마우스와 키보드 입력을 다른 스레드를 생성하여 처리한다.

스레드 사용의 예

스레드 사용의 예는 다음과 같다.

- 웹 브라우저에서 웹 페이지를 보면서 동시에 파일을 다운로드할 수 있도록 한다.
- 워드 프로세서에서 문서를 편집하면서 동시에 인쇄한다.
- 게임 프로그램에서는 응답성을 높이기 위하여 많은 스레드를 사용한다.
- GUI에서는 마우스와 키보드 입력을 다른 스레드를 생성하여 처리한다.

스레드를 생성하는 방법

하나의 프로세스를 여러 스레드로 나누어 번갈아 가면서 실행하는 다중 스레드 환경에서는 하나의 스레드가 독립적으로 실행되는 기본 단위가 되어야 한다. 즉 스레드는 하나의 독립적인 작업을 실행하는 코드와 개별 데이터를 가지고 있어야 한다. 이는 작업을 번갈아 가면서 수행하더라도 해당 데이터가 다른 스레드에 의해 변경되거나 하면 코드가 정상적으로 실행되지 않기 때문이다. **스레드를 자바에서 생성하려면 객체 단위로 생성해야 한다.**

스레드 생성 방법
- Thread 클래스를 상속하는 방법
- Runnable 인터페이스를 구현하는 방법

스레드 생성과 실행

스레드는 Thread 클래스가 담당한다.

메소드	설명
Thread()	매개 변수가 없는 기본 생성자
Thread(String name)	이름이 name인 Thread 객체를 생성.
Thread(Runnable target, String name)	Runnable을 구현하는 객체로부터 스레드를 생성.
static int activeCount()	현재 활동중인 스레드의 개수를 반환한다.
String getName()	스레드의 이름을 반환한다.
String setName(String name)	스레드의 이름을 설정한다.
String getPriority()	스레드의 우선 순위를 반환한다.
String setPriority(int priority)	스레드의 우선 순위를 설정한다.

스레드 생성과 실행

스레드는 Thread 클래스가 담당한다.

메소드	설명
static void sleep(int milliseconds)	현재의 스레드를 지정된 시간만큼 재운다.
void run()	스레드가 시작될 때 이 메서드가 호출된다. 스레드가 처리할 작업을 이 메서드 안에 정의한다.
void start()	스레드를 시작한다.
static void yield()	현재 스레드를 다른 스레드에 양보하게 만든다.
void interrupt()	현재 스레드를 중단한다.
void isInterrupted()	현재 스레드가 중단될 수 있는지를 검사한다.

Thread 클래스를 상속하기

첫 번째 방법은 Thread 클래스를 상속받아 서브 클래스를 만들고 run()메소드를 오버라이드하는 방법이다.

run()메소드에는 스레드가 수행하여야 할 작업이 들어간다. 이 서브 클래스의 인스턴스를 생성하고 start()메소드를 호출하면 스레드가 실행된다.

Thread 클래스를 상속하기

```
class Counting extends Thread {
        public void run() {
                for (int i = 0; i < 10; i++)
                        System.out.println(i);
        }
}
```

스레드가 실행하는 모든 작업은 이 run() 메소드 안에 있어야 한다.

Thread 클래스를 상속하기

```
public class Test {
        public static void main(String args[]) {
                Thread t = new Counting();
                t.start();
        }
}
```

start() 메소드를 호출해야만 스레드가 실행된다.

Runnable 인터페이스를 구현하는 방법

Runnable 인터페이스는 메소드 run()만을 정의한다.
Runnable 객체는 Thread 생성자에 전달되어서 실행된다.

다음과 같은 순서로 작성하면 된다.

① Runnable을 구현하는 클래스를 생성한다.
② Runnable 클래스에 run() 메소드를 작성한다.
③ Thread 클래스의 객체를 생성하고,
Runnable 객체를 Thread 생성자의 매개변수로 넘긴다.
④ Thread 객체의 start() 메소드를 호출하여야 한다.

Runnable 인터페이스를 구현하는 방법

```
class Counting  implements Runnable {
        public void run() {
                for (int i = 0; i < 10; i++)
                        System.out.println(i);
        }
}
```

스레드가 실행하는 모든 작업은 이 run() 메소드 안에 있어야 한다.

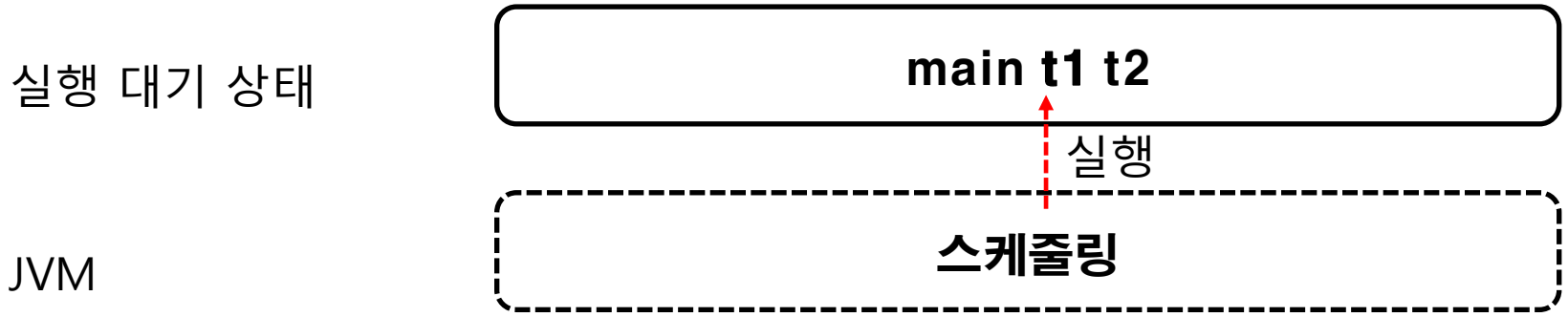
Runnable 인터페이스를 구현하는 방법

```
public class Test {
        public static void main(String args[]) {
                Counting c = new Counting();
                Thread t = new Thread(c);
                t.start();
        }
}
```

start() 메소드를 호출해야만 스레드가 실행된다.

스레드

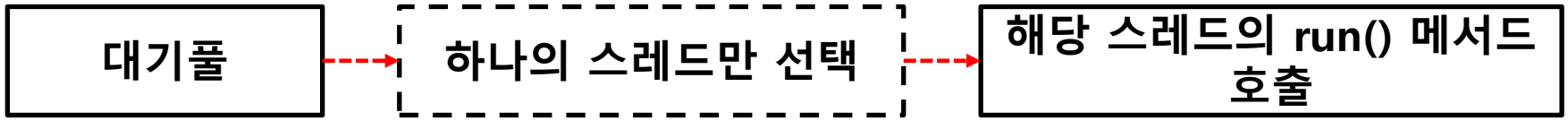
main() 메서드를 호출하여 t1, t2 스레드를 실행하면 각 스레드는 실행 대기 상태로 들어간다. 실행 대기 상태는 JVM의 스케줄링에 의해 실행할 수 있는 상태를 의미한다. 자바 프로그램 JVM에 의해 실행되는 스레드가 결정되며 이것을 스케줄링이라고 한다.



멀티태스킹은 동시에 여러 작업을 하는 것이라고 했다. 그러나 동시라는 말은 정확하지 않을 수도 있다. 만일 CPU가 하나라면 한 순간에 실행할 수 있는 프로세스 또는 스레드는 하나만 가능하기 때문이다. 그러나 동시에 여러 작업이 실행되는 것처럼 느끼는 것은 하나의 CPU가 시차를 두고 스케줄링에 의해 번갈아 가면서 실행하기 때문이다.

run() 메서드

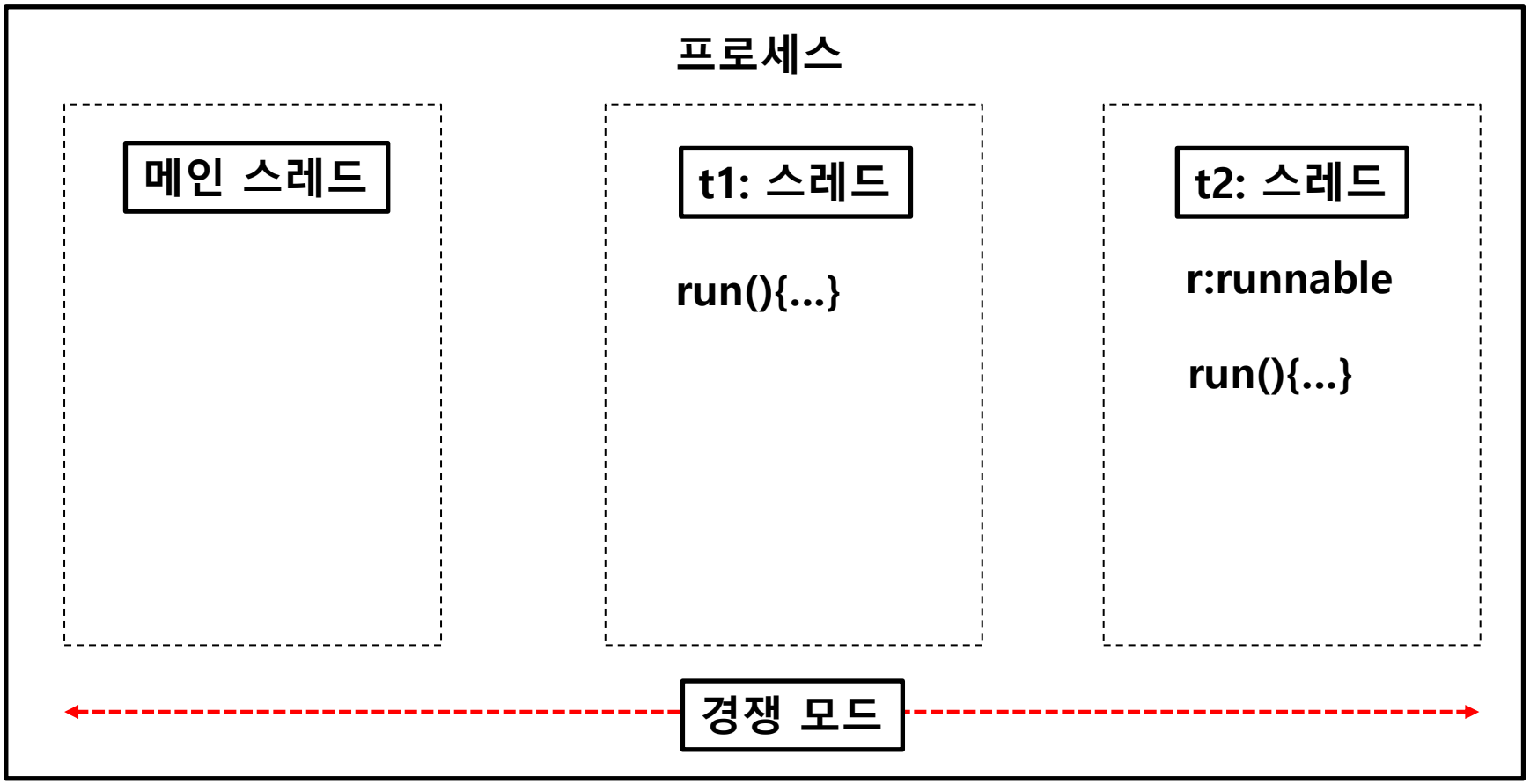
run() 메서드는 스레드의 생명 주기와 관련된 메서드로 CPU 사용권이 주어지면(실행 상태) 시스템에 의해 호출이 되도록 약속된 메서드다.
즉 스레드가 실행되기를 준비하는 장소인 대기풀(waiting pool)에 있는 스레드 중 하나의 스레드가 선택되어 실행 상태가 되면 해당 스레드의 run() 메서드가 실행된다. 그러므로 이 메서드를 재정의하면 스레드가 CPU 사용권을 가지고 있을 때 원하는 코드가 실행되도록 정의할 수 있다.



사용권은 운영체제에서 정한 순위에 따라 결정(스케줄링이라고 한다)되는데, 이 우선순위는 여러 방법에 의해서 결정되므로 어느 프로세스가 먼저 실행될지 예측하기 어렵다 이 때문에 다중 프로그래밍이 적용될 경우 랜덤 효과를 볼 수 있다.

run() 메서드

스레드로 생성된 각 객체는 메인 스레드와 경쟁 관계에 놓인다. 경쟁관계에 있는 스레드는 운영체제의 스케줄링에 따라 실행 순서가 결정된다. 실행 결과는 각각의 시스템에 따라 다르게 출력될 수 있음에 유의하자



중간 점검 문제

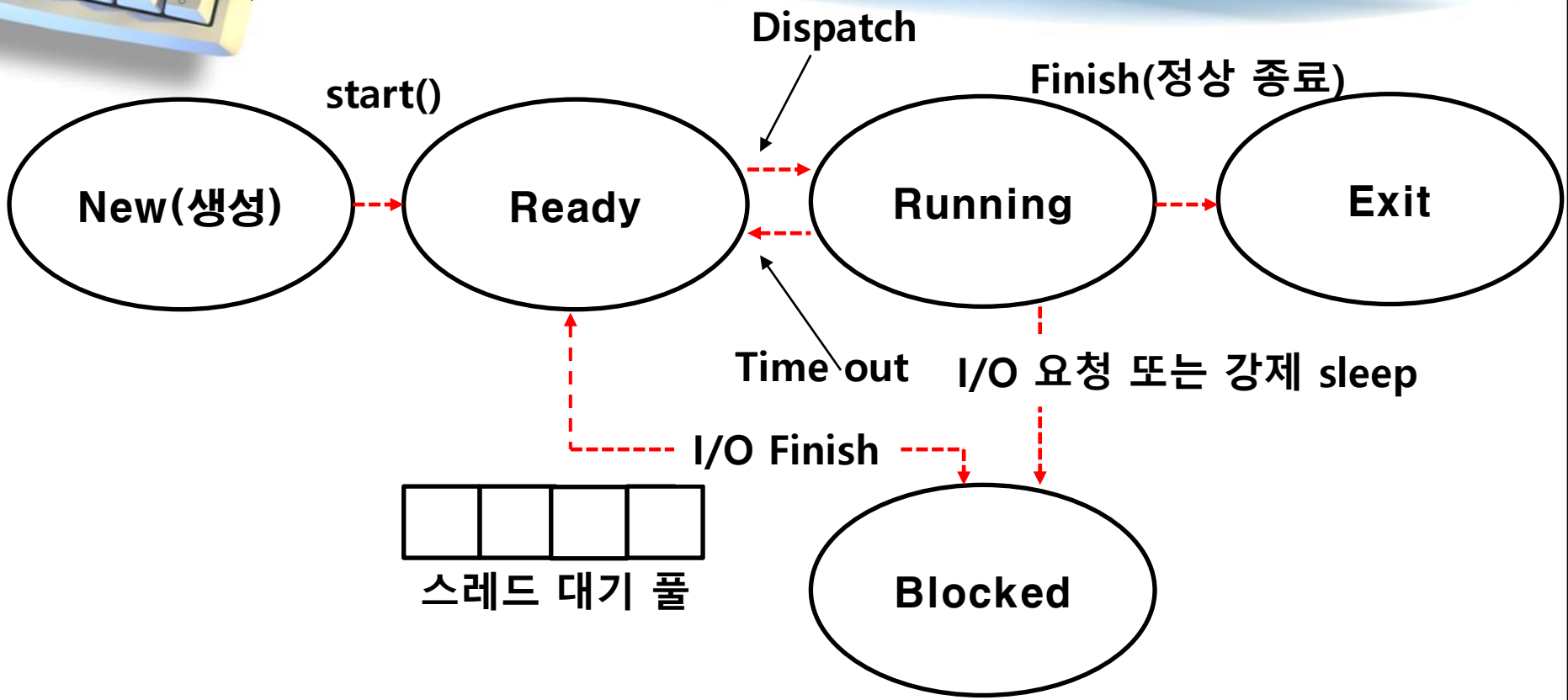
1. 스레딩을 담당하는 클래스의 이름은?

2. Thread를 상속받는 방법의 문제점은 무엇인가?

스레드의 생명 주기

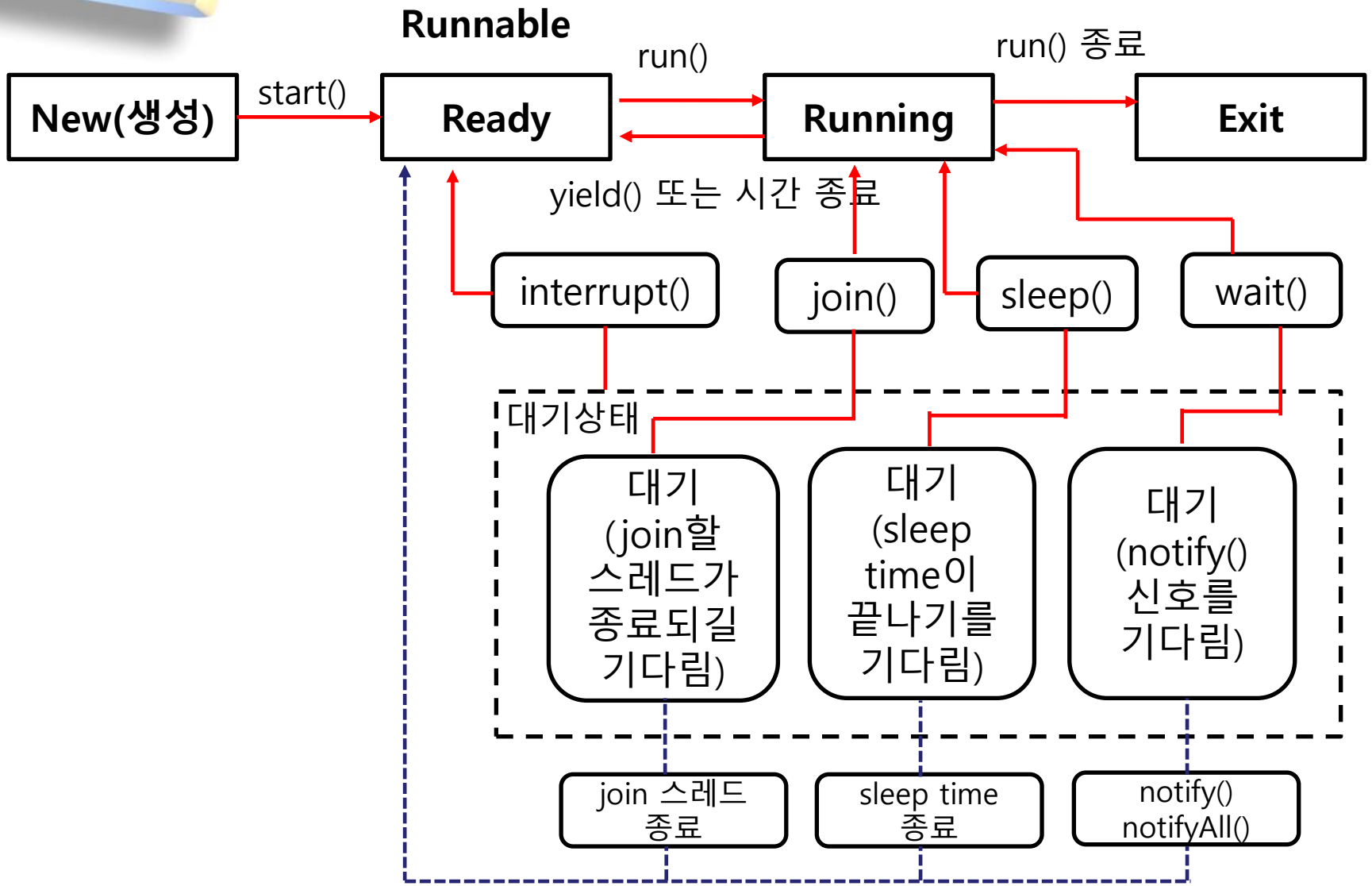
- new 상태(생성 상태): Thread 객체가 메모리에 생성만된 상태이다.
- ready 상태(준비 상태): 실행 대기 풀에서 실행 상태로 들어가려고 준비하는 상태이다. 언제든지 CPU에 의해 선점되면 실행 가능한 상태로, 런어블(runnable) 상태라 한다.
- running(실행 상태): 스레드가 CPU를 독점해 코드를 실행 중인 상태이다.
- blocked 상태(대기 상태): 이 상태에 있는 스레드는 일시적으로 스레드의 수행이 중단된 상태로서 특정 조건이 되면 다시 실행 상태로 이전된다.
- exit(종료 상태): 스레드가 정상적으로 종료된 상태이다.

스레드의 생명 주기

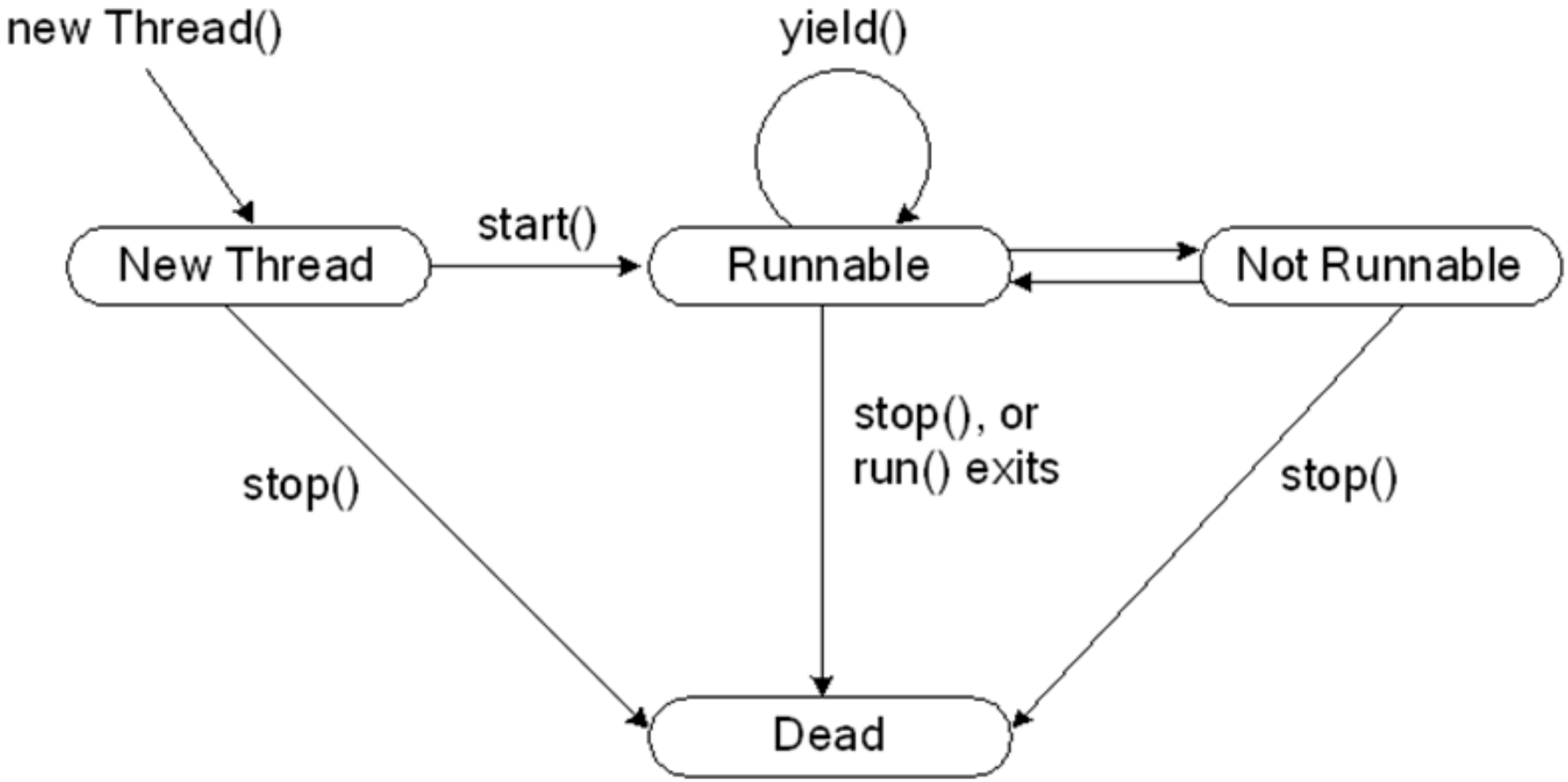


운영체제의 스케줄러가 실행 대기 풀에서 대기 중인 스레드 하나를 선택하면 해당 스레드는 CPU 사용권을 독점하여 실행 상태로 전환된다. 이때 해당 스레드는 디스패치 되었다고 말한다. 스케줄러에 의해 디스패치된 스레드는 running 상태가 되면서 Thread 클래스에 정의된 run() 메서드를 호출한다. 따라서 해당 스레드가 실행 상태에 있을 때 원하는 작업이 있으면 run() 메서드를 재정의해서 해당 기능을 수행하게 할 수 있다.

스레드 상태


Runnable
New(생성)
start()
Ready
run()
Running
run() 종료
Exit
yield() 또는 시간 종료
interrupt()
join()
sleep()
wait()
대기상태
대기 (join할 스레드가 종료되길 기다림)
대기 (sleep time이 끝나기를 기다림)
대기 (notify() 신호를 기다림)
join 스레드 종료
sleep time 종료
notify() notifyAll()

스레드 상태


new Thread()
yield()
New Thread
start()
Runnable
Not Runnable
stop()
stop(), or
run() exits
stop()
Dead

생성 상태와 실행 가능 상태

1) 생성 상태

Thread 클래스를 이용하여 새로운 스레드를 생성한 상태이다. 그러나 생성만 되었을 뿐 스레드가 아직은 시작되지 않았다.

- Thread 클래스를 이용하여 새로운 스레드를 생성
- start()는 생성된 스레드를 시작
- stop()은 생성된 스레드를 멈추게 한다.

생성 상태와 실행 가능 상태

2) 실행 가능 상태(Runnable)

- 스레드가 실행되면 일반적으로 Runnable 상태

생성 상태에서 start() 메소드를 호출하면 run()이 호출되어 스레드가 실행 가능 상태로 된다. CPU의 제어권을 할당받는 순간 running상태가 되었다가 일정 시간동안 작업을 한 후 Runnable 상태로 되돌아와서 자신의 순서가 되기를 기다린다.

실행 중지 상태

3) 실행 중지 상태

실행 가능한 상태에서 다음의 이벤트가 발생하면 실행 중지 상태로 된다.

① 스레드가 wait()를 호출하는 경우

② 스레드가 sleep()을 호출하는 경우

③ 스레드가 입출력 작업을 하기 위해 대기하는 경우

실행 중지 상태

위의 각각의 경우에 대하여 다음과 같은 이벤트가 발생하면 실행 가능 상태로 된다.

①의 경우 : notify()나 notifyall()이 호출되어야 한다.

②의 경우 : 지정된 sleep() 시간이 경과하여야 한다.

③의 경우 : 입출력 작업이 완료되어야 한다.

종료상태

4) 종료상태

스레드는 두 가지 방법으로 소멸하게 된다. 하나는 정상적인 종료를 통해서이고, 다른 하나는 강제적으로 종료되는 것이다.

- 정상적인 종료

정상적인 종료하는 것은 스레드의 메소드인 run()이 작업을 끝내고 종료하는 경우이다.

종료상태

- 강제적인 종료

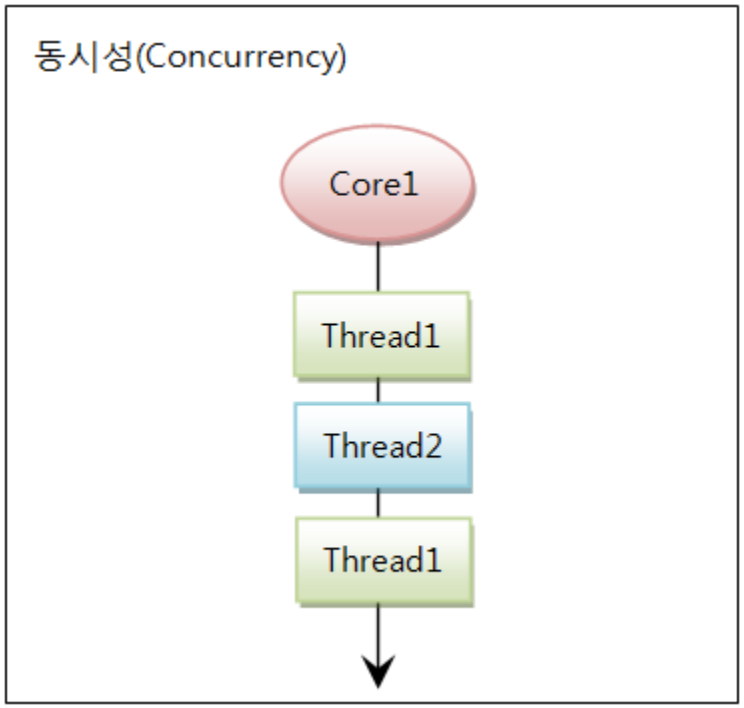
강제적인 종료는 스레드를 Thread 클래스에 정의되어 있는 stop() 메소드를 사용하여 정지시켜 종료하는 것이다.

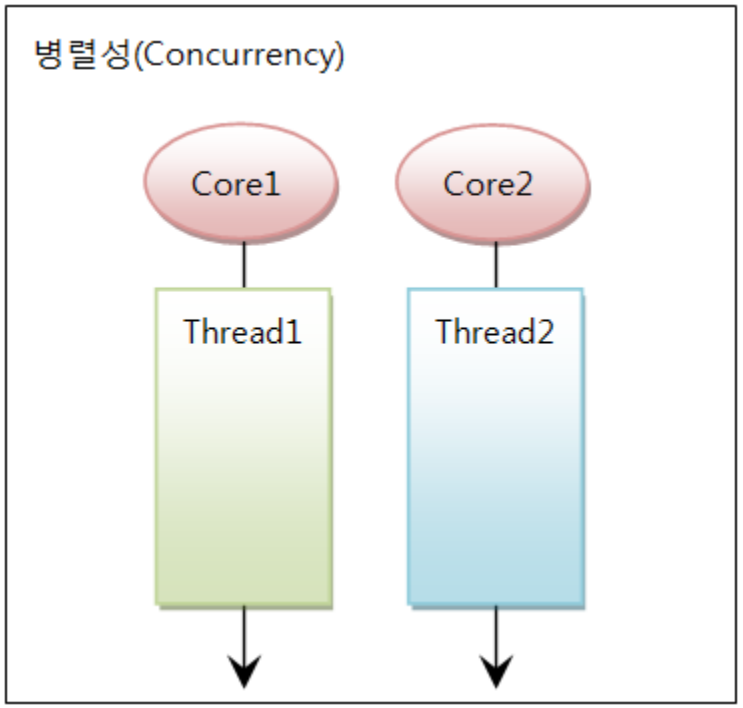
```
Thread my = new Thread();
my.start();
try {
        Thread.currentThread().sleep(1000);
} catch(InterruptedException e) {   }
my.stop();
```

스레드 우선 순위

❖동시성과 병렬성

- 동시성
  - 멀티 작업 위해 하나의 코어에서 멀티 스레드가 번갈아 가며 실행하는 성질
- 병렬성
  - 멀티 작업을 위해 멀티 코어에서 개별 스레드를 동시에 실행하는 성질





스레드 우선 순위

- 자바의 스레드 스케줄링
  - 우선 순위(Priority) 방식과 순환 할당(Round-Robin) 방식 사용
  - 우선 순위 방식 (코드로 제어 가능)
    - 우선 순위가 높은 스레드가 실행 상태를 더 많이 가지도록 스케줄링
    - 1~10까지 값을 가질 수 있으며 기본은 5
  - 순환 할당 방식 (코드로 제어할 수 없음)
    - 시간 할당량(Time Slice) 정해서 하나의 스레드를 정해진 시간만큼 실행

스레드의 우선 순위

1) 스레드의 우선 순위

스레드는 생성될 때 자신을 생성한 스레드로부터 우선순위를 상속받는다. 실행 도중에는 다음의 메소드를 이용하여 스레드의 우선순위를 얻거나 변경하는 것이 가능하다.

- void setPriority(int newPriority) : 현재 스레드의 우선순위를 변경한다.
- getPriority() : 현재 스레드의 우선순위를 반환한다.

스레드의 우선 순위

우선 순위는 Thread 클래스 정의되어 있는 상수를 이용하여 표현된다.

- Thread.MAX_PRIORITY = 10 : 최대 우선 순위
- Thread.MIN_PRIORITY = 1 : 최소 우선 순위
- Thread.NORM_PRIORITY = 5 : 기본 순위

스레드 상태 제어

2) sleep()

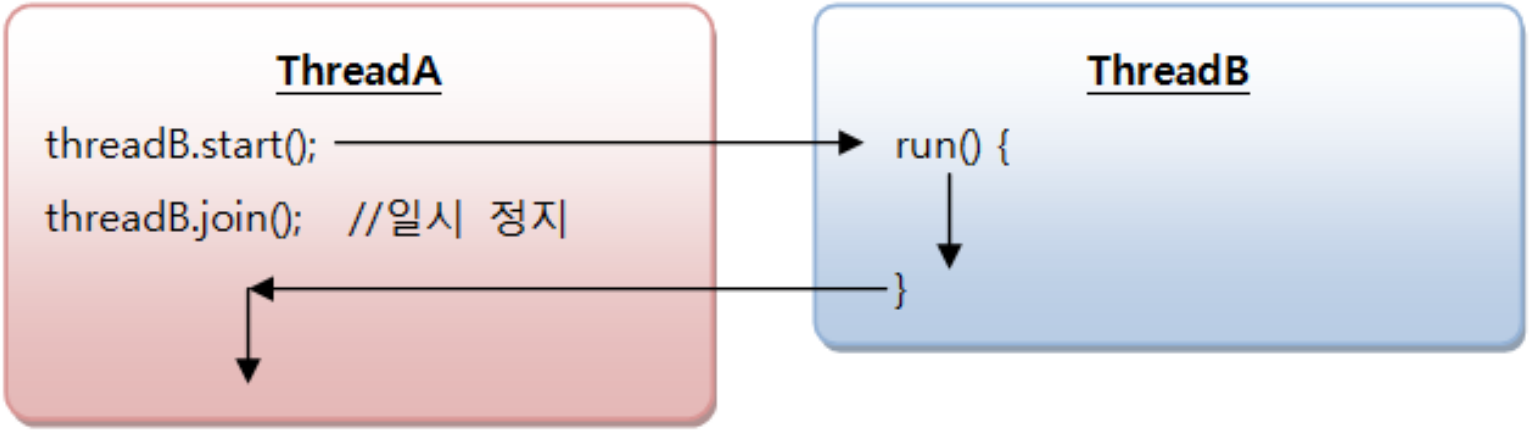
sleep() 메소드는 현재 스레드의 실행을 지정된 시간 동안 중단하게 만든다. (얼마 동안 일시 정지 상태로 있을 것인지 밀리 세컨드(1/1000) 단위로 지정)

```
public class CountDown {
        public static void main(String args[]) throws
          InterruptedException {
                for (int i = 10; i >= 0; i--) {
                        Thread.sleep(1000);
                        System.out.println(i);
                }
        }
}
```

스레드 상태 제어

3) 조인(Joins)

join()메소드는 하나의 스레드가 다른 스레드의 종료를 기다리게 하는 메소드이다.

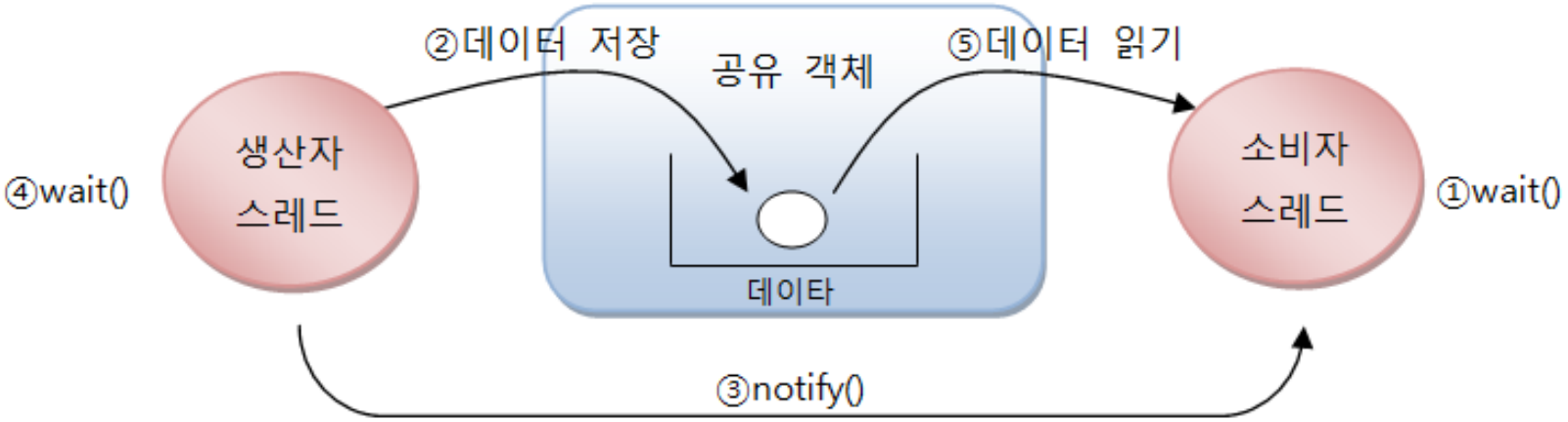


여러 스레드가 동시 작업을 할 때 스레드 간의 종속관계가 맺어지는 경우가 있다. 예를 들어 A라는 스레드 작업이 완료되어야 B라는 스레드 작업을 진행할 수 있는 경우입니다. 이럴 때 B 스레드는 A 스레드의 작업이 완료될 때까지 기다렸다가 실행되어야 하는데, 이때 사용하는 메서드가 Thread 클래스의 join() 메서드이다. **계산 작업을 하는 스레드가 모든 계산 작업 마쳤을 때, 결과값을 받아 이용하는 경우 주로 사용한다.**

스레드 상태 제어

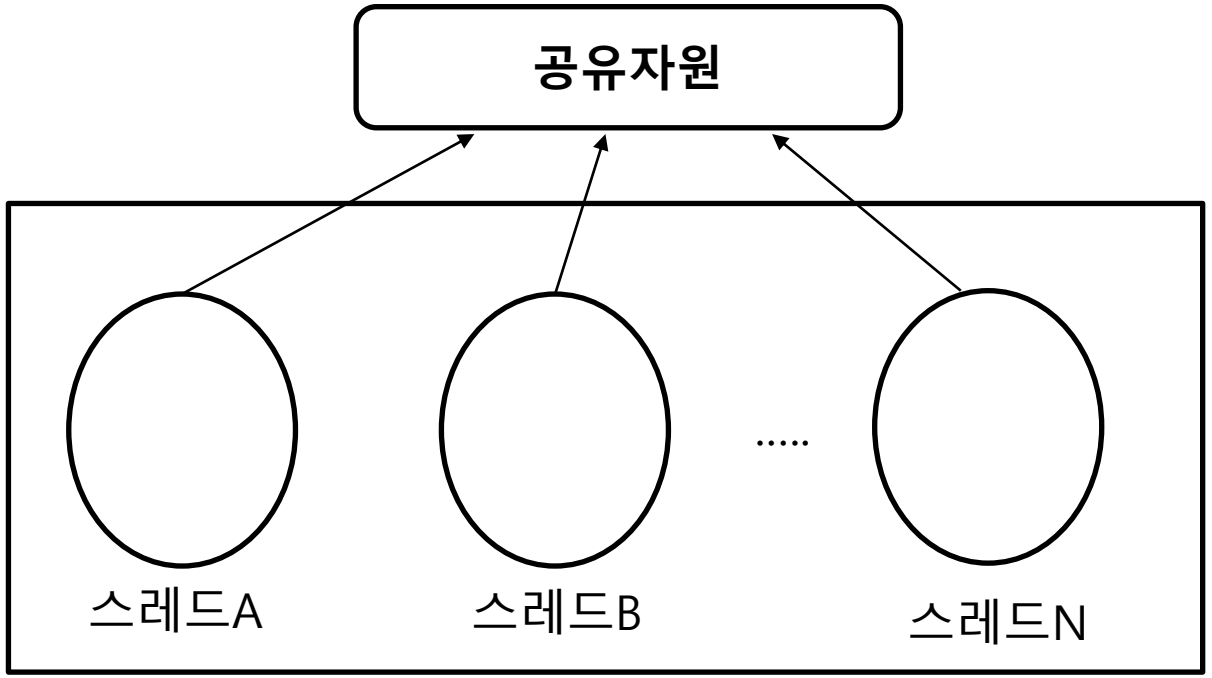
4) 스레드간 협업 – wait(), notify(), notifyAll()

- 두 개의 스레드가 교대로 번갈아 가며 실행해야 할 경우 주로 사용



동기화 문제

다중 스레드 환경에서는 독립적인 자원이 아닌 공유 자원을 이용해 작업을 수행하는 경우가 있다. 이럴 때는 서로의 작업에 영향을 미친다.



동기화 문제

다중 스레드 환경에서 공유 자원을 사용하는 작업(임계영역이라고 함)이 있으면 작업이 종료 될때까지 다른 스레드에게 사용권을 뺏기지 않도록 (비선점) 다른 스레드를 대기하게 한다. 이것을 동기화라 한다.

즉 동기화란 공유 자원에 접근할 때 한번에 하나의 스레드만 접근할 수 있도록 잠금(lock)을 걸어 데이터의 일관성을 유지하는 것이다. 주로 예약이나 예매, 금융관련 프로그램에서 사용한다.

좌석예매에 먼저 접근한 사람이 완료할 때까지 다른 사람은 접근하지 못하도록 처리하는 것을 동기화 작업이라고 한다. 동기화란 예매 시스템처럼 하나의 자원을 여러 스레드가 동시에 접근하여 사용할 때 발생할 수 잇는 오류를 방지하기 위한 작업이다.

synchronized를 이용한 동기화

스레드 동기화에 두가지 관점이 있다. **실행 순서의 동기화와 메모리 접근 동기화이다.** 전자는 실행 순서를 중요시하여 스레드의 실행 순서를 정의하고 스레드가 이 순서에 반드시 따르도록 하는 것을 말한다. 후자는 여러 스레드가 메모리에 접근할 때 동시 접근을 막는 것을 말한다. 즉 실행 순서보다 한순간에 하나의 스레드만 접근 가능하게 하는 상황을 더 중요시 한다.

둘 이상의 스레드가 동시에 실행하면 문제가 발생하는 코드 블록을 임계 영역이라고 한다. 메모리 접근에 대한 동기화는 이러한 임계 영역의 접근을 동기화하겠다는 의미이다. 자바에서는 임계 영역에 synchronized를 선언해 잠금(lock)을 설정하여 동기화할 수 있다. 이 방법은 객체와 메서드에 적용할 수 있다.

synchronized를 이용한 동기화

동기화 처리

프로그램 구현 시 동기화는 블록이나 메서드 단위로 작업할 수 있다. 동기화 작업은 매우 간단한다. 블록 또는 메서드를 선언할 때 synchronized 키워드만 선언하면 된다

블록 동기화 : synchronized(객체명) { }

메서드 동기화 : synchronized 메서드명() { }

블록 동기화

블록 동기화 작업을 할 때는 객체명에 공유 객체명을 지정한다. 만일 객체 자신이 공유된다면 this를 지정한다

메서드 동기화

메서드를 동기화 처리할 때는 메서드의 선언부에 synchronized를 선언한다. synchronized가 선언된 메서드는 한 스레드에 의해 호출되었다면 실행이 완료될 때까지 다른 스레드는 실행할 수 없다.

동기화 문제

동기화 문제의 핵심은 동시에 실행되는 스레드 사이에 있는 공유되는 데이터이다. 이때 멀티 스레드에 의해 **공유자원이 참조될 수 있는 코드의 범위를 임계영역(critical region)**이라 한다.

멀티 스레드 프로그램에서 임계영역을 처리하지 않을 경우 심각한 문제가 발생할 수 있다. 이러한 상황을 해결할 수 있는 방법이 동기화를 이용하는 것이다. 동기화를 처리하기 위해 모든 객체에 락(lock)을 포함 시켰다. 락이란 공유 객체에 여러 스레드가 동시에 접근하지 못하도록 하기 위한 것으로 모든 객체가 힙 영역에 생성될 때 자동으로 만들어진다.

동기화 메소드

3) 동기화된 메소드 이용

동기화 메소드 및 동기화 블록 – synchronized

- 단 하나의 스레드만 실행할 수 있는 메소드 또는 블록
- 다른 스레드는 메소드나 블록이 실행이 끝날 때까지 대기해야 동기화 메소드

synchronized 키워드의 사용하면 어떤 한 순간에는 하나의 스레드만이 임계영역(공유자원이 참조될 수 있는 코드의 범위) 안에서 실행하는 것이 보장된다.

동기화 메소드

동기화 메소드

```
public synchronized void Method() {
        임계 영역; // 단 하나의 스레드만 실행
}
```

동기화 블록

```
public void Method() {
   // 여러 스레드가 실행 가능 영역
   synchronized (동기화할 객체 또는 클래스명){
        임계영역 // 단 하나의 스레드만 실행
   }
   // 여러 스레드가 실행 가능 영역
}
```

동기화 메소드

```
public synchronized void deposit(int amount) {
        balance += amount;
}
public synchronized void withdraw(int amount) {
        balance -= amount;
}
public synchronized int getBalance() {
        return balance;
}
```

한순간에 하나의 스레드만을 허용

동기화 메소드

메소드를 동기화되도록 한 것은 두 가지의 효과를 가진다.

- 동기화된 메소드는 동일한 데이터에 대하여 동시에 호출될 수 없다. 하나의 스레드가 동기화된 메소드를 실행하고 있으면 첫번째 스레드가 종료될 때까지 다른 모든 스레드는 중단된다.
- 동기화된 메소드가 종료되면 자동적으로 이후의 동기화된 메소드 호출은 올바르게 변경된 상태를 볼 수 있다.

생성자는 동기화될 수 없다. 왜냐하면 객체를 생성하는 하나의 스레드만이 생성자에 접근할 수 있기 때문이다.

스레드간의 조정

만약 두개의 스레드가 데이터를 주고 받는 경우에 발생

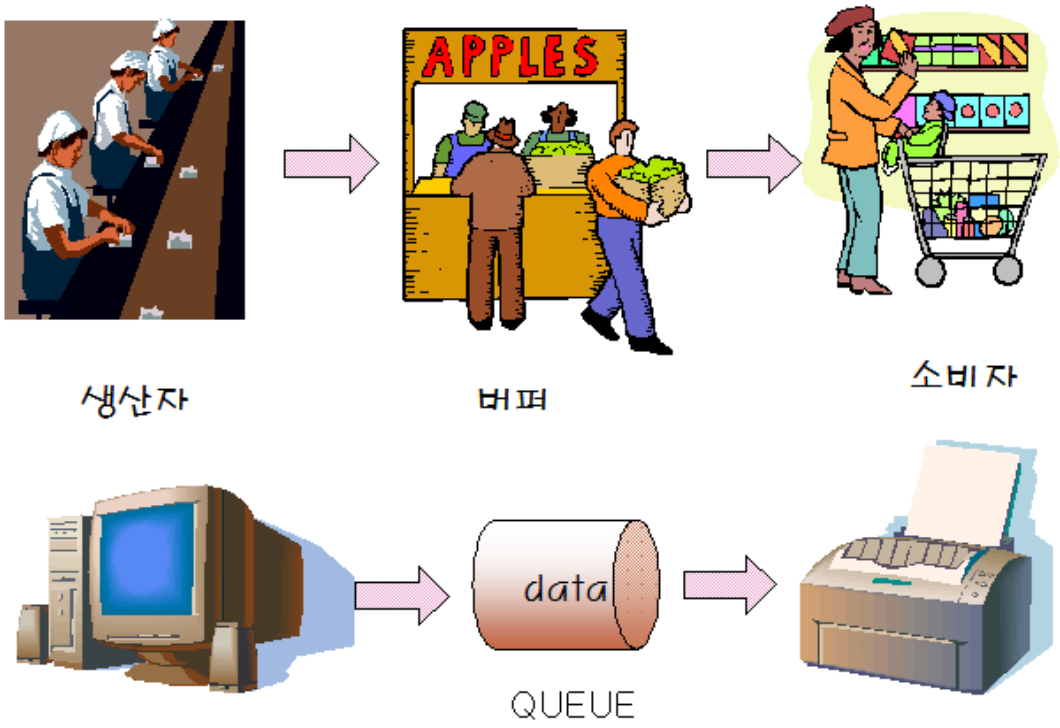


그림 16.7 생산자와 소비자 문제

wait()와 notify()

스레드 간에는 때로 서로 간에 작업을 조정할 필요가 있다. 생산자와 소비자 모델이 대표적이다. 이 모델에서 생산자는 데이터를 생산하고 소비자는 데이터를 사용해서 어떤 작업을 수행한다. 생산자와 소비자는 동시에 데이터에 대해 생산 작업과 소비 작업을 수행할 수 있으므로 각각을 스레드로 구현한다.

wait()와 notify()

- wait()
  - 다른 스레드가 notify()를 불러줄 때까지 기다린다.
- notify()
  - wait()를 호출하여 대기중인 스레드를 깨우고 RUNNABLE 상태로 만든다.
  - 2개 이상의 스레드가 대기중이라도 오직 한 스레드만 깨운다.
- notifyAll()
  - wait()를 호출하여 대기중인 모든 스레드를 깨우고 모두 RUNNABLE 상태로 만든다.

wait()와 notify()

wati() 메소드가 호출되면 스레드는 가지고 있던 잠금을 해제하고 실행을 일시적으로 중지한다. 나중에 다른 스레드가 동일한 잠금을 얻어서 notify()나 notifyAll() 메서드를 호출하면 이벤트가 발생하기를 기다리면서 일시적으로 중지된 스레드들이 깨어나게 된다.

메서드	설명
wait()	스레드를 실행하는 중에 이 메서드를 만나면 가지고 있던 잠금을 양보하고 대기 상태로 들어간다.
notify()	대기 상태인 스레드 중에서 하나의 스레드를 깨운다
notifyAll()	대기 상태인 스레드를 모두 깨운다

세 메서드는 해당 객체의 잠금에 관여한다. 그러므로 해당 객체의 잠금 영역 내에서만 사용할 수 있다. 즉 synchronized 메서드에서만 사용할 수 있다. 잠금 영역이 아닌 영역에서 호출하면 'IllegalMonitorStateException' 예외가 발생하게 된다.

wait()와 notify()

모든 객체는 대기 풀이 있다. 이곳에는 객체의 wait() 메서드를 실행하고서 실행을 멈춘 스레드들이 저장된다. 여기에 저장된 스레드는 시간이 경과하면 스스로 빠져나와 준비 상태로 갈수 있지만 대부분 notify()와 notifyAll() 메서드 또는 인터럽트에 의해 빠져나온다. wait() 메서드가 호출되면 현재 실행 중인 메서드는 wait() 메서드의 메시지를 전달받는 객체의 대기 풀로 들어가며 대기 상태가 된다. 대기 풀에 여러 스레드가 있을 때 notify() 메서드가 실행되면 그 중 하나의 스레드만이 빠져나와 준비 상태가 되며 준비 대기 풀에 들어간다. 어떤 스레드가 빠져 나올지는 JVM이 구현한 알고리즘에 따라 달라지지만 대부분 선입선출 구조인 큐를 따르는 경우가 많다.

wait()와 notify()

wait() 메소드는 어떤 일이 일어나기를 기다릴 때 사용하는 메소드이다. notify()는 반대로 어떤 일이 일어났을 때 이를 알려주는 메소드이다. 따라서 이를 이용하여 생산자는 생산이 끝나면 이를 소비자에게 알려주고 소비자는 소비가 끝나면 이를 생산자에에 알려준다.
구체적으로 wait()가 호출되면 스레드는 락(lock)을 해제하고 실행을 일시 중지한다. 다른 스레드가 락을 획득할 것이고 작업이 끝나면 notifyAll()을 호출할 것이다. 그러면 다시 첫 번째 스레드가 깨워지고 다시 실행을 계속하게 된다.

wait()와 notify()

wait(), notify() 메소드를 설명하는 이유는 자료의 일관성을 유지하기 위해 동기화를 하지만 동기화 상태에서 상황에 따라 락을 획득한 스레드가 일시 중지 하고 다른 스레드에게 제어를 넘겨할 상황이 있기 때문에 wait(), notify()를 사용해야 한다.


Thank You