Thread

program & process

프로그램은 실행 가능한 파일이고, 프로세스는 실행 중인 프로그램으로 자원과 쓰레드로 구성되어 있다.

Thread

프로세스의 자원을 이용해서 실제로 작업을 수행하는 것이다.
모든 프로세스는 최소한 하나의 쓰레드를 가지고 있으며, 쓰레드가 하나라면 싱글 쓰레드, 둘 이상의 쓰레드라면 멀티 쓰레드이다.

멀티 쓰레드

멀티 태스킹과 차이는 멀티 태스킹은 동시에 여러 프로세스를 실행시키는 것이고 멀티 쓰레딩은 하나의 프로세스 내에 동시에 여러 쓰레드를 실행시키는 것이다.
프로세스를 생성하는 것보다 쓰레드를 생성하는 비용이 적으며 같은 프로세스 내의 쓰레드들은 서로 자원을 공유한다.

멀티 쓰레딩의 장단점은 다음과 같다.

- 장점
    - CPU의 사용률을 향상시킨다.
    - 자원을 보다 효율적으로 사용할 수 있다.
    - 사용자에 대한 응답성이 향상된다.
    - 작업이 분리되어 코드가 간결해진다.
- 단점
    - 동기화에 주의해야 한다.
    - 교착 상태가 발생하지 않도록 주의해야 한다.
    - 각 쓰레드가 효율적으로 고르게 실행될 수 있게 해야 한다.

쓰레드의 구현과 실행

쓰레드를 구현하는 방법에는 두 가지가 있다.

- Thread 클래스를 상속

```
class MyThread extends Thread {
    public void run() { /* 작업 내용 */ } // Thread 클래스의 run()을 오버라이딩
한다.
}
```

- Runnable 인터페이스를 구현

```
class MyThread implements Runnable {
    public void run() { /* 작업 내용 */ } // Runnable 인터페이스의 추상 메서드
run()을 구현한다.
}
```

다음 중 **Runnable 인터페이스를 구현하는 것**이 재사용성이 높고 코드의 일관성을 유지할 수 있다.
Thread 클래스를 상속받으면 자손 클래스에서 Thread 클래스의 메서드를 직접 호출할 수 있지만,
Runnable을 구현하면 Thread 클래스의 static 메서드인 currentThread()를 호출하여 쓰레드에 대한 참조를 얻어와야 호출이 가능하다.

실행하는 방법은 다음과 같다.

- Thread 클래스를 상속

```
MyThread t = new MyThread();
t.start();
```

- Runnable 인터페이스를 구현

```
Runnable r = new MyThread();
Thread t = new Thread(r);
t.start();
```

하나의 쓰레드는 start() 한 번만 호출될 수 있으므로 작업을 한 번 더 수행해야 한다면 새로운 쓰레드를 생성 후에 start()를 호출한다.

실행되는 순서는 다음과 같다.

1. main메서드에서 쓰레드의 start()를 호출한다.
2. start()는 새로운 쓰레드를 생성하고, 쓰레드가 작업하는데 사용될 호출 스택을 생성한다.
3. 새로 생성된 호출 스택에 run()이 호출되어, 쓰레드가 독립된 공간에서 작업을 수행한다.
4. 스케줄러가 정한 순서에 의해서 번갈아 가면서 실행된다.

쓰레드의 실행 순서는 OS 스케줄러가 결정한다.

쓰레드의 종류

쓰레드는 사용자 쓰레드와 데몬 쓰레드 두 종류가 있다.
실행 중인 사용자 쓰레드가 하나도 없을 때 프로그램은 종료된다.
main 쓰레드는 main메서드의 코드를 수행하는 쓰레드로 사용자 쓰레드이다.

싱글 쓰레드와 멀티 쓰레드

싱글 코어에서는 싱글 쓰레드보다 멀티 쓰레드로 작업한 시간이 더 걸린다.
왜냐하면 쓰레드간의 작업 전환에 시간이 걸리고, 한 쓰레드가 작업하는 것을 기다리는데 발생하는 대기시간때문이다.

두 쓰레드가 서로 다른 자원을 사용하는 작업의 경우에는 싱글 쓰레드보다 멀티 쓰레드가 더 효율적이다.
예를 들어, 싱글 쓰레드에서 I/O blocking이 일어나는데 멀티 쓰레드에선 다른 쓰레드가 작업을 한다.

쓰레드의 우선 순위

작업의 중요도에 따라 쓰레드의 우선 순위를 다르게 하여 특정 쓰레드가 더 많은 작업 시간을 갖게 할 수 있는데 희망사항이다.
왜냐하면 우선 순위는 OS 스케줄러가 결정하기 때문이다.
설정하면 일찍 끝날 확률이 높아지는 것뿐이므로 쓰레드에 우선 순위를 부여하는 대신 작업에 우선 순위를 두어 PriorityQueue에 저장해 놓는 것도 생각하자.

쓰레드의 우선 순위와 관련된 메서드와 상수는 다음과 같다.

```
void setPriority(int newPriority) // 쓰레드의 우선 순위를 지정한 값으로 변경한다.
int getPriority() // 쓰레드의 우선 순위를 반환한다.

MAX_PRIORITY = 10; // 최대 우선 순위
NORM_PRIORITY = 5; // 보통 우선 순위
MIN_PRIORITY = 1; // 최소 우선 순위
```

쓰레드의 우선 순위는 쓰레드를 생성한 쓰레드로부터 상속받는데 main 쓰레드는 우선 순위가 5이므로 쓰레드의 우선 순위는 자동적으로 5가 된다.

쓰레드 그룹

보안상의 이유로 도입됐으며 서로 관련된 쓰레드를 그룹으로 묶어서 다루기 위한 것이다.
쓰레드를 쓰레드 그룹에 포함시키려면 Thread의 생성자를 이용해야 한다.
모든 쓰레드는 반드시 하나의 쓰레드 그룹에 포함되어 있어야 한다.
Java가 실행이 되면, JVM은 main과 system이라는 쓰레드 그룹을 만든다.
쓰레드 그룹을 지정하지 않고 생성한 쓰레드는 main 쓰레드 그룹에 속하며 자신을 생성한 쓰레드의 그룹을 상속받는다.

ThreadGroup의 메서드는 다음과 같다.

메서드	설 명
ThreadGroup(String name)	지정된 이름의 새로운 쓰레드 그룹을 생성하는 생성자
ThreadGroup(ThreadGroup parent, String name)	지정된 쓰레드 그룹에 포함되는 새로운 쓰레드 그룹을 생성하는 생성자
int activeCount()	쓰레드 그룹에 포함된 활성 상태인 쓰레드의 수를 반환하는 메서드
int activeGroupCount()	쓰레드 그룹에 포함된 활성 상태인 쓰레드 그룹의 수를 반환하는 메서드
void checkAccess()	현재 실행중인 쓰레드가 쓰레드 그룹을 변경할 권한이 있는지 체크하는 메서드
void destory()	쓰레드 그룹과 하위 쓰레드 그룹까지 모두 삭제하는 메서드(비어있어야 삭제 가능하다.)

메서드	설 명
int enumerate(Thread[] list, [boolean recurse]) int enumerate(ThreadGroup[] list, [boolean recurse])	쓰레드 그룹에 속한 쓰레드 또는 하위 쓰레드 그룹의 목록을 지정된 배열에 담고 그 개수를 반환하는 메서드(recurse의 값을 true로 주면 하위 쓰레드도 담는다.)
int getMaxPriority()	쓰레드 그룹의 최대 우선 순위를 반환하는 메서드
String getName()	쓰레드 그룹의 이름을 반환하는 메서드
ThreadGroup getParent()	쓰레드 그룹의 상위 쓰레드 그룹을 반환하는 메서드
void interrupt()	쓰레드 그룹에 속한 모든 쓰레드를 인터럽트 상태로 전환하는 메서드
boolean isDaemon()	쓰레드 그룹이 데몬 쓰레드 그룹인지 확인하는 메서드
boolean isDestoryed()	쓰레드 그룹이 삭제되었는지 확인하는 메서드
void list()	쓰레드 그룹에 속한 쓰레드와 하위 쓰레드 그룹에 대한 정보를 출력하는 메서드
boolean parentOf(ThreadGroup g)	지정된 쓰레드 그룹의 상위 쓰레드 그룹인지 확인하는 메서드
void setDaemon(boolean daemon)	쓰레드 그룹을 데몬 쓰레드 그룹으로 설정 및 해제하는 메서드
void setMaxPriority(int put)	스레드 그룹의 최대 우선 순위를 설정하는 메서드

데몬 쓰레드

다른 일반 쓰레드의 작업을 돕는 보조적인 역할을 수행하는 쓰레드이다.
일반 쓰레드가 모두 종료되면 자동적으로 종료되며, 가비지 컬렉터, 자동 저장, 화면 자동 갱신 등에 사용된다.
무한 루프와 조건문을 이용해서 실행 후 대기하고 있다가 특정 조건이 만족되면 작업을 수행하고 다시 대기하도록 작성한다.
일반 쓰레드와 작성 및 실행 방법이 같으나 쓰레드를 생성한 다음 start() 전에 setDaemon(true)를 호출하기만 하면 된다.

쓰레드의 실행 제어

효율적인 멀티 쓰레드 프로그램을 만들기 위해서는 보다 정교한 스케줄링을 통해 자원과 시간을 여러 쓰레드가 낭비없이 잘 사용하도록 프로그래밍해야 한다.

쓰레드의 스케줄링과 관련된 메서드는 다음과 같다.

메서드	설 명
static void sleep(long millis, [int nanos])	지정된 시간(천 분의 일 초)동안 쓰레드를 일시정지시키는 메서드 지정된 시간이 지나면 다시 실행대기 상태로 된다.

메서드	설 명
void join([long millis], [int nanos])	지정된 시간동안 쓰레드가 실행되도록 하는 메서드 지정된 시간이 지나거나 작업이 종료되면 join()을 호출한 쓰레드로 다시 돌아와 실행을 계속한다.
int interrupt()	sleep()이나 join()에 의해 일시정지 상태인 쓰레드를 깨워서 실행대기 상태로 만드는 메서드
void stop()	쓰레드를 즉시 종료시키는 메서드(deprecated)
void suspend()	쓰레드를 일시정지시키는 메서드(deprecated) resume()를 호출하면 다시 실행대기 상태가 된다.
void resume()	suspend()에 의해 일시정지 상태에 있는 쓰레드를 실행대기 상태로 만드는 메서드(deprecated)
static void yield()	실행 중에 자신에게 주어진 실행시간을 다른 쓰레드에게 양보하고 자신은 실행대기 상태가 되는 메서드

sleep()

현재 쓰레드를 지정된 시간동안 멈추는 메소드이며, InterruptedException 예외 처리를 해야한다.
특정 쓰레드를 지정해서 멈추게 하는 것은 불가능하다.

interrupt()

쓰레드의 작업을 취소한다.
주요 메서드는 다음과 같다.

- boolean isInterrupted(): 쓰레드의 interrupted 상태를 반환하는 메서드
- static boolean interrupted(): 현재 쓰레드의 interrupted 상태를 알려주고, false로 초기화하는 메서드

실행 제어 메서드

쓰레드의 실행을 일시정지, 재개, 완전 정지를 하지만, 교착 상태에 빠지기 쉽다.
suspend(), resume(), stop()은 deprecated로 직접 구현을 한다.
토글 방식으로 구현을 하며, 쉽게 바뀌는 변수를 뜻하는 volatile 키워드를 붙인다.

yield()

남은 시간을 다음 쓰레드에게 양보하고, 자신은 실행대기한다.
yield()와 interrupt()를 적절히 사용하면, 응답성과 효율을 높일 수 있다.

join()

지정된 시간동안 특정 쓰레드가 작업하는 것을 기다린다.
InterruptedException 예외처리를 해야한다.

쓰레드의 상태

쓰레드의 상태는 다음과 같다.

상 태	설 명
NEW	쓰레드가 생성되고 아직 start()가 호출되지 않은 상태
RUNNABLE	실행 중 또는 실행가능한 상태
BLOCKED	동기화 블럭에 의해서 일시정지된 상태
WAITING TIMED_WAITING	쓰레드의 작업이 종료되지는 않았지만 일시정지 상태
TERMINATED	쓰레드의 작업이 종료된 상태

쓰레드의 상태 과정은 다음과 같다.

1. 쓰레드를 생성하고 start()를 호출하면 바로 실행되는 것이 아니라 실행 대기열에 저장되어 자신의 차례가 될 때까지 기다려야 한다.
2. 실행대기 상태에 있다가 자신의 차례가 되면 실행 상태가 된다.
3. 주어진 실행 시간이 다 되거나 yield()를 만나면 다시 실행대기 상태가 되고 다음 차례의 쓰레드가 실행 상태가 된다.
4. 실행 중에 일시정지 상태가 될 수 있다.
5. 지정된 일시정지 시간이 다 되거나 notify(), resume(), interrupt()가 호출되면 일시정지 상태를 벗어나 다시 실행 대기열에 저장되어 자신의 차례를 기다리게 된다.
6. 실행을 모두 마치거나 stop()이 호출되면 쓰레드는 소멸된다.

쓰레드의 동기화

한 쓰레드가 진행 중인 작업을 다른 쓰레드가 간섭하지 못하도록 막는 것을 의미한다.
공유 데이터를 사용하는 코드 영역을 임계 영역으로 지정해놓고, lock을 획득한 단 하나의 쓰레드만 영역 내의 코드를 수행하도록 한다.

- 동기화가 필요한 이유: 멀티 쓰레드 프로세스의 경우 여러 쓰레드가 같은 프로세스 내의 자원을 공유해서 작업하기 때문에 서로의 작업에 영향을 주게 되어 의도했던 것과는 다른 결과를 얻을 수 있기 때문이다.

synchronized를 이용한 동기화

synchronized 키워드를 이용한 동기화의 방식은 두 가지가 있다.

- 메서드 전체를 임계 영역으로 지정

```
public synchronized void print() {
  // 임계 영역
}
```

- 특정한 영역을 임계 영역으로 지정

```
synchronized(객체의 참조변수) {
  // 임계 영역
```

```
  }
```

한 번에 한 쓰레드만 실행하므로 임계 영역은 최소화하여 효율적인 프로그램을 작성해야 한다.
또한, 동기화 블럭 내에 있는 변수는 private으로 해야한다.

한 번에 한 쓰레드만 실행할 수 있으므로 특정 쓰레드가 lock을 가진 상태로 오랜 시간을 보낸다면 다른 쓰레드들의 작업들이 원활히 진행하지 못 한다.

wait(), notify(), notifyAll()

동기화의 효율을 높이기 위해 사용한다.
Object클래스에 정의되어 있으며, 동기화 블록 내에서만 사용할 수 있다.

- wait(): 객체의 lock을 풀고 쓰레드를 해당 객체의 waiting pool에 넣는다.
- notify(): waiting pool에서 대기중인 쓰레드 중의 하나를 깨운다.
- notifyAll(): waiting pool에서 대기중인 모든 쓰레드를 깨운다.

wait()과 notify()는 어떤 객체를 깨울지 불분명하다.
그래서 기아 현상과 경쟁 상태가 나타나는데 이것을 해결한 것이 Lock과 Condition이다.

Lock과 Condition을 이용한 동기화

Lock클래스의 종류는 다음과 같다.

- ReentrantLock: 가장 일반적인 배타 lock이며, 재진입이 가능하다.
- ReentrantReadWriteLock: 읽기에는 공유적이고 쓰기에는 배타적인 lock이다.
- StampedLock: ReentrantReadWriteLock에 낙관적인 lock(일단 무조건 저지르고 나중에 확인하는 기능)을 추가한 것이다.

Lock을 사용하는 동기화 방법은 다음과 같다.

```
lock.lock();
try {
  // 임계 영역
} finally {
  lock.unlock();
}
```

ReentrantLock의 메서드는 다음과 같다.

메서드	설 명
ReentrantLock()	기본 생성자
ReentrantLock(boolean fair)	fair를 true로 주면 가장 오래 기다린 쓰레드가 lock을 획득할 수 있게 하는 생성자(성능 저하)
void lock()	lock을 잠구는 메서드
void unlock()	lock을 해지하는 메서드

메서드	설 명
boolean isLocked()	lock이 잠겼는지 확인하는 메서드

Condition의 생성 방법은 다음과 같다.

```
private ReentrantLock lock = new ReentrantLock();
private Condition condition = lock.newCondition();
```

Condition의 메서드는 다음과 같다.

메서드	설 명
void await()	wait()과 동일
void signal()	notify()과 동일
void signalAll()	notifyAll()과 동일

volatile

cache와 메모리간의 불일치를 해소하기 위한 키워드이다.
컴퓨터는 성능 향상을 위해 변수의 값을 core의 cache에 저장해놓고 작업을 해서
도중에 메모리에서 값이 변경되었는데도 cache에 저장된 값은 갱신되지 않아서 값이 다른 경우가 발생한다.

이러한 경우에 volatile을 사용하면 변수의 값을 읽어올 때 메모리에서 읽어온다.

fork & join 프레임워크

여러 쓰레드가 동시에 처리하는 것을 쉽게 만들어주는 프레임워크이다.

- fork(): 해당 작업을 쓰레드 풀의 작업 큐에 넣는다.(비동기 메서드)
- join(): 해당 작업의 수행이 끝날 때까지 기다렸다가, 수행이 끝나면 그 결과를 반환한다.(동기 메서드)

수행할 작업에 따라 두 클래스 중에서 하나를 상속받아 추상 메서드 compute()를 구현해야 한다.

- RecursiveAction: 반환값이 없는 작업을 구현할 때 사용
- RecursiveTask: 반환값이 있는 작업을 구현할 때 사용

fork()로 나눈 작업을 큐에 넣고, compute()를 재귀호출해서 구현한다.

다음과 같이 작업을 시작한다.

```
ForkJoinPool pool = new ForkJoinPool(); // 쓰레드 풀(코어의 개수와 동일한 개수)을 생
성
V task = new V(); // 수행할 작업을 생성
V result = pool.invoke(task);
```

작업 훔치기

자신의 작업 큐가 비어있는 쓰레드는 다른 쓰레드의 작업 큐에서 작업을 가져와서 수행한다.