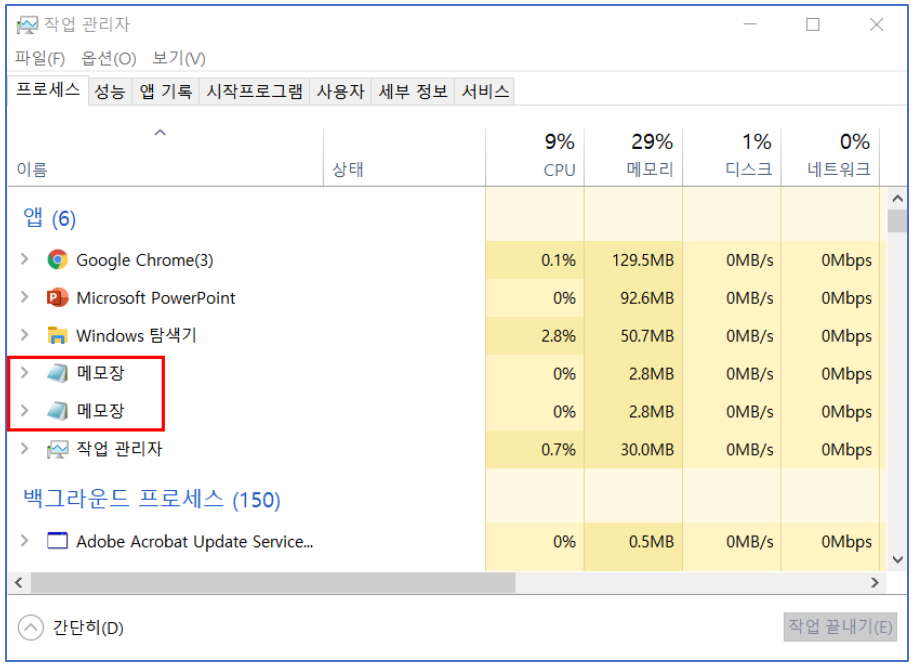
스레드

애플리케이션을 실행하면 운영체제로부터 실행에 필요한 메모리를 할당받아 애플리케이션이 실행되는데, 이것을 프로세스(process)라고 한다. 그리고 프로세스 내부에서 코드의 실행흐름을 스레드(thread)라고 한다.

하나의 애플리케이션은 멀티 프로세스를 만들기도 한다. 예를 들어 메모장 애플리케이션을 2개 실행했다면 2개의 메모장 프로세스가 생성된 것이다.



스레드

운영체제는 두 가지 이상의 작업을 동시에 처리하는 멀티 태스킹을 할 수 있도록 메모리 자원을 프로세스마다 적절히 할당해주고, 병렬로 실행시킨다. 예를들어, 워드로 문서작업을 하면서 동시에 윈도우 미디어 플레이어로 음악을 들을 수 있다.

멀티태스킹은 꼭 멀티 프로세스를 뜻하는 것은 아니다. 한 프로세스 내에서 멀티 태스킹을 할 수 있도록 만들어진 애플리케이션도 있다.

미디어 플레이어는 동영상 재생과 음악재생이라는 두 가지 작업을 동시에 처리하고, 메신저는 채팅 기능을 제공하면서 동시에 파일 전송기능을 수행하기도 한다.

스레드

어떻게 하나의 프로세스가 두 가지 이상의 작업을 처리할 수 있을까?
그 비밀은 멀티스레드에 있다.

스레드

스레드는 사전적 의미로 한 가닥의 실이라는 뜻인데, 한 가지 작업을 실행하기 위해 순차적으로 실행할 코드를 실처럼 이어놓았다고 해서 유래된 이름이다.
하나의 스레드는 하나의 코드실행흐름이기 때문에 한 프로세스에 스레드가 2개라면 2개의 코드 실행흐름이 생긴다는 의미이다.

멀티 프로세스는 운영체제에서 할당받은 자신의 메모리를 가지고 실행하기 때문에 각 프로세스는 서로 독립적이다. 따라서 하나의 프로세스에서 오류가 발생해도 다른 프로세스에 영향을 미치지 않는다.

하지만 멀티스레드는 하나의 프로세스 내부에 생성되기 때문에 하나의 스레드가 예외를 발생시키면 프로세스 자체가 종료될 수 있다.

스레드

예를 들어 멀티프로세스인 워드와 엑셀을 동시에 사용하던 도중, 워드에 오류가 생겨 먹통이 되더라도 엑셀은 여전히 사용 가능하다.

그러나 멀티스레드로 동작하는 메신저의 경우 파일을 전송하는 스레드에서 예외가 발생하면 메신저프로세스 자체가 종료되므로 채팅 스레드도 같이 종료된다.

그렇기 때문에 멀티스레드에서는 예외처리에 만전을 기해야 한다.

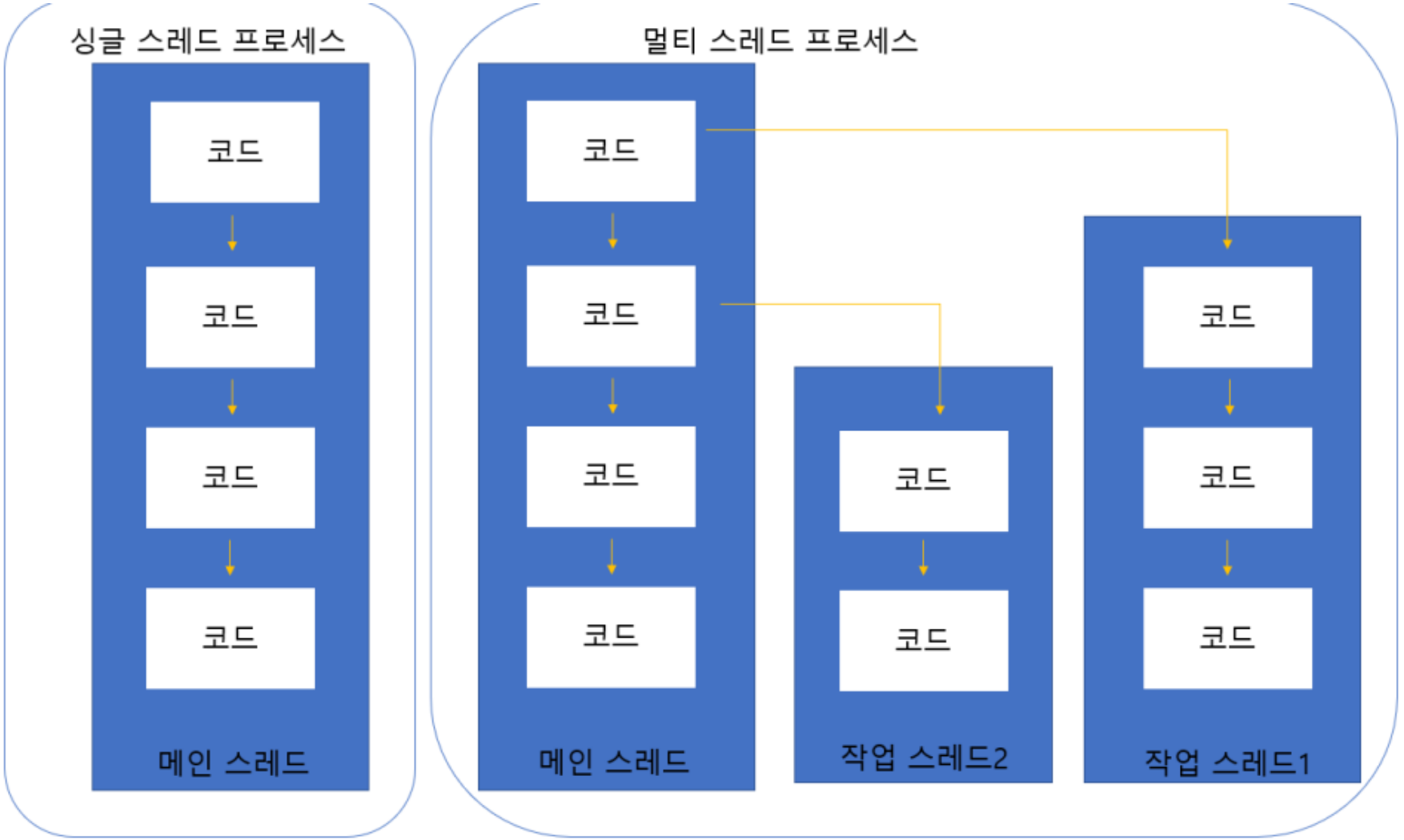
메인 스레드

모든 자바 애플리케이션은 메인 스레드가 main() 메소드를 실행하면서 시작된다.

메인 스레드는 main() 메소드의 첫 코드부터 아래로 순차적으로 실행하고, main() 메소드의 마지막 코드를 실행하거나 return문을 만나면 실행이 종료된다.

메인 스레드에서 필요에 따라 작업 스레드들을 만들어서 병렬로 코드를 실행할 수 있다. 즉 멀티 스레드를 생성해서 멀티 태스킹을 수행할 수 있다.

메인 스레드


싱글 스레드 프로세스
코드
코드
코드
코드
메인 스레드
멀티 스레드 프로세스
코드
코드
코드
코드
메인 스레드
코드
코드
작업 스레드2
코드
코드
코드
작업 스레드1

메인 스레드

싱글 스레드 애플리케이션에서는 메인 스레드가 종료하면 프로세스도 종료된다. 하지만 멀티 스레드 애플리케이션에서는 실행 중인 스레드가 하나라도 있다면, 프로세스는 종료되지 않는다.

메인 스레드가 작업 스레드보다 먼저 종료되더라도 작업 스레드가 계속 실행 중이라면 프로세스는 종료되지 않는다.

작업 스레드 생성과 실행

메인 스레드는 반드시 존재하기 때문에 메인 작업 이외에 추가적인 병렬작업의 수만큼 스레드를 생성하면 된다.

자바에서는 작업 스레드도 객체로 생성되기 때문에 클래스가 필요하다.

java.lang.Thread 클래스를 직접 객체화해서 생성해도 되지만, Thread 클래스를 상속해서 하위 클래스를 만들어 생성할 수도 있다.

작업 스레드 생성과 실행

java.lang.Thread 클래스로부터 작업 스레드 객체를 직접 생성하려면 Runnable 타입을 매개값으로 갖는 생성자를 호출해야 한다.

Thread thread = new Thread(Runnable target);

Runnable은 인터페이스 타입이기 때문에 구현 객체를 만들어 대입해야한다.

Runnable에는 run()메소드 하나가 정의되어 있는데, 구현 클래스는 run()메소드를 재정의해서 작업스레드가 실행할 코드를 작성해야한다.

작업 스레드 생성과 실행

다음과 같이 Runnable 구현 클래스를 작성한다.

```
class Task implements Runnable {
        public void run() {
                스레드가 실행할 코드;
        }
}
```

```
Runnable task = new Task();
Thread thread = new Thread(task);
```

Runnable 구현객체를 생성한 후, 이것을 매개값으로 해서 Thread 생성자를 호출해야 비로소 작업 스레드가 생성된다.

작업 스레드 생성과 실행

Thread 클래스를 상속하여 작업 스레드 객체를 생성할 수 있다.

Thread 클래스의 run()메소드를 재정의해서 실행할 코드를 작성하면 된다.

```
public class WorkerThread extends Thread {
        @Override
        public void run() {
                스레드가 실행할 코드;
        }
}
Thread thread = new WorkerThread();
```

start() 메소드

작업 스레드는 생성되는 즉시 실행되는 것이 아니라, start() 메소드를 호출해야만 비로소 실행된다.

start() 메소드가 호출되면, run()이 호출되면서 독립적인 호출스택이 생성된다.

스레드의 이름

스레드는 자신의 이름을 가지고 있다. 스레드의 이름이 큰 역할을 하는 것은 아니지만, 어떤 스레드가 어떤 작업을 하는지 조사할 목적으로 가끔 사용된다.

메인스레드는 “main” 이라는 이름을 가지고 있고, 우리가 직접 생성한 스레드는 자동적으로 “Thread-n”이라는 이름으로 설정된다. n은 스레드의 번호를 말하는데, Thread-n 대신 다른 이름으로 설정하고 싶다면 Thread 클래스의 setName()메소드로 변경할 수 있다.

thread.setName(“스레드 이름”);

스레드의 이름

반대로 스레드 이름을 알고 싶을 경우에는 getName() 메소드를 호출하면 된다.

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

setName()과 getName()은 Thread 클래스의 인스턴스 메소드이므로 스레드 객체의 참조가 필요하다. 스레드 객체의 참조를 가지고 있지 않다면, Thread 클래스의 정적 메소드인 currentThread() 로 현재 스레드의 참조를 얻을 수 있다.

```
Thread thread = Thread.currentThread();
```

Thread 필드

제어자 및 타입	필드	설명
public static final int	MAX_PRIORITY	스레드의 최대 우선순위로 값은 10
public static final int	MIN_PRIORITY	스레드의 최소 우선순위로 값은 1
public static final int	NORM_PRIORITY	스레드의 할당된 기본 우선순위로 값은 5

Thread 메소드

제어자 및 타입	메소드	설명
String	getName()	스레드의 이름 반환
int	getPriority()	스레드의 우선순위 반환
boolean	isAlive()	스레드의 실행상태 판단
void	join()	다른 스레드가 종료될때까지 대기
void	run()	스레드의 실행 메소드
void	start()	run() 메소드 호출하면서 스레드 시작

스레드 제어

스레드 객체를 생성하고 start() 메소드를 호출하면 바로 실행되는 것이 아니라 실행 대기상태가 된다. 실행 대기상태란 언제든지 실행할 준비가 되어있는 상태를 말한다. 스케줄러는 실행 대기상태에 있는 스레드 중에서 하나를 선택해서 실행상태로 만든다.

실행상태의 스레드는 run()메소드를 모두 실행하기 전에 다시 실행 대기상태로 돌아갈 수 있으며, 실행 대기상태에 있는 다른 스레드가 선택되어 실행상태가 되기도 한다.

실행상태에서 run()메소드의 내용이 모두 실행되면 스레드의 실행이 멈추고 종료상태가 된다.

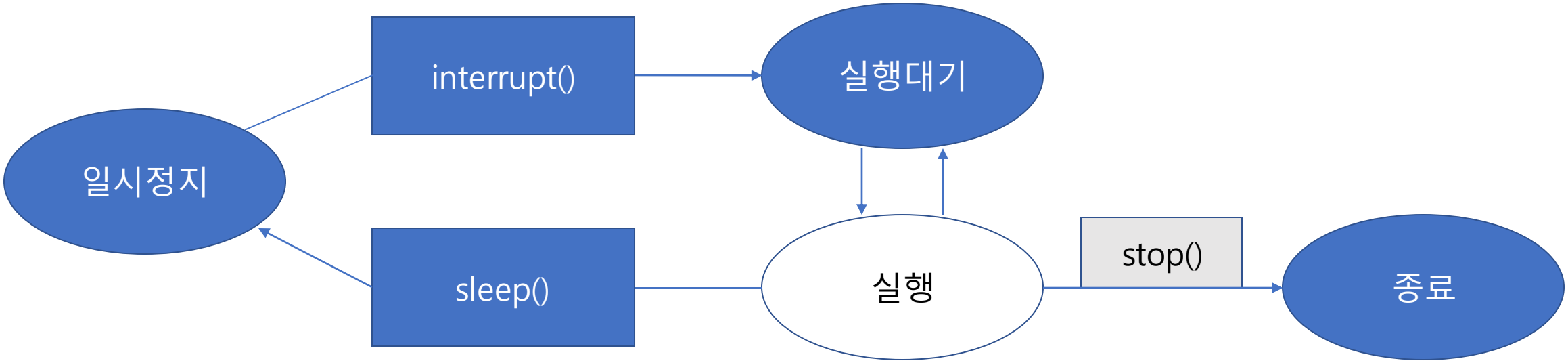
스레드 제어

사용자는 미디어 플레이어에서 동영상을 보다가 일시정지할 수도 있고, 종료할 수도 있다. 일시정지는 조금 후 다시 동영상을 보겠다는 의미이므로 미디어 플레이어는 동영상 스레드를 일시정지 상태로 만들어야 한다.

그리고 종료는 더 이상 동영상을 보지 않겠다는 의미이므로 미디어 플레이어는 스레드를 종료상태로 만들어야 한다. 이와 같이 살행중인 스레드의 상태를 변경하는 것을 스레드 상태제어라고 한다.

스레드 제어

상태변화를 가져오는 메소드의 종류



스레드 제어

메소드	설명
interrupt()	일시정지 상태의 스레드에서 InterruptedException을 발생시켜, 예외처리 코드(catch)에서 실행대기 상태로 가거나 종료 상태로 갈 수 있도록 한다.
sleep(long millis)	주어진 시간 동안 스레드를 일시정지 상태로 만든다. 주어진 시간이 지나면 자동적으로 실행대기 상태가 된다.
stop()	스레드를 즉시 종료한다. 불안전한 종료를 유발하므로 사용하지 않는 것이 좋다.

동기화 메소드

싱글 스레드 프로그램에서는 1개의 스레드가 객체를 사용하면 되지만, 멀티 스레드 프로그램에서는 스레드들이 객체를 공유해서 작업해야 하는 경우가 있다.

공유 객체를 사용할 때의 주의할 점

멀티 스레드 프로그램에서 스레드들이 객체를 공유해서 작업해야 하는 경우, 스레드A가 사용하던 객체를 스레드 B가 상태를 변경할 수 있기 때문에 스레드 A가 의도했던 것과는 다른 결과를 산출할 수 있다.

동기화 메소드

스레드가 사용중인 객체를 다른 스레드가 변경할 수 없게 하려면 스레드 작업이 끝날 때까지 객체에 잠금을 걸어 다른 스레드가 사용할 수 없도록 한다.

멀티스레드 프로그램에서 단 하나의 스레드만 실행할 수 있는 코드 영역으로 임계영역(critical section)이라고 한다. 자바는 임계영역을 지정하기 위해 동기화 메소드를 제공한다. 스레드가 객체 내부의 동기화 메소드를 실행하면 즉시 객체에 잠금을 걸어 다른 스레드가 동기화 메소드를 실행하지 못하도록 한다.

동기화 메소드

동기화 메소드를 만들려면 메소드 선언에 synchronized 키워드를 붙이면 되는데, 인스턴스와 정적메소드 어디든 붙일 수 있다.

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

동기화 메소드는 메소드 전체 내용이 임계영역이므로 스레드가 동기화 메소드를 실행하는 즉시 객체에는 잠금이 일어나고, 스레드가 동기화 메소드를 실행 종료하면 잠금이 풀린다.