

스레드

예제 소스 코드는 파일과 연결되어 있습니다.

editplus(유료), notepad++(무료)와 같은 편집 도구를 미리 설치하여 PPT를 슬라이드 쇼로 진행할 때 소스 파일과 연결하여 보면 강의하실 때 편리합니다.

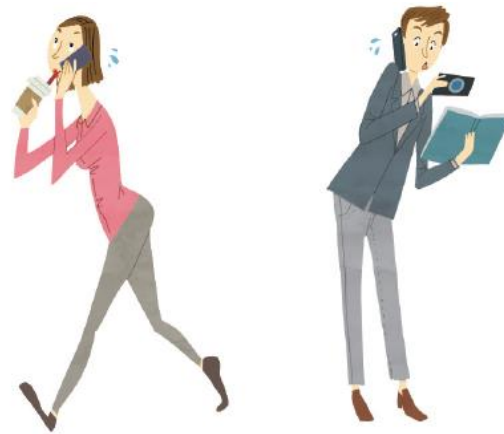
프로세스와 스레드

■ 프로세스

- 실행 중인 프로그램
- 디스크에 있는 프로그램이 메모리에 적재되어 운영체제의 제어를 받는 상태를 의미
- 자신만의 자원을 가지기 때문에 프로세스끼리는 서로 독립적

■ 멀티태스킹

- 대부분의 운영체제는 시스템 자원을 효율적으로 사용할 수 있도록 멀티태스킹을 지원
- 멀티 코어 CPU라면 실제로 다수의 애플리케이션을 동시에 병렬처리, 싱글 코어 CPU라면 운영체제가 다수의 애플리케이션을 병행처리
- 심지어 하나의 애플리케이션에서도 동시에 수행할 수 있는 다수의 코드 블록이 있을 수 있기 때문에 멀티태스킹 가능 -> 멀티스레딩



프로세스와 스레드

■ 스레드 의미

- 하나의 실행 흐름으로 프로세스 내부에 존재
- 하나의 프로세스는 하나 이상의 실행 흐름을 포함하기 때문에 프로세스는 적어도 하나의 스레드를 가진다
- 프로세스에 비해 실행 환경에 필요한 자원이 매우 적기 때문에 경량 프로세스라고도 한다
- 메모리와 파일 등 모든 자원을 프로세스 자원과 공유한다.



(a) 단일 스레드 프로세스



(b) 멀티스레드 프로세스

프로세스와 스레드

■ 스레드 개념

- 자바 애플리케이션의 실행 환경인 JVM은 하나의 프로세스로 실행
- 예 : 하나의 운영체제에서 각각 1개와 2개의 스레드로 구성된 2개의 애플리케이션을 실행한다면



스레드 생성과 실행

■ 스레드 생성 방법

- 자바 애플리케이션은 적어도 메인 스레드라는 하나의 스레드를 가짐
- 다수의 스레드로 실행하려면 다음 방법 중 하나를 사용하여 작업 스레드를 별도로 생성해야 함

Runnable 구현 클래스 정의

```
class MyRunnable implements Runnable {  
    // 스레드가 실행할 코드  
}
```

Thread 자식 클래스 정의

```
class WorkerThread extends Thread {  
    // 스레드가 실행할 코드  
}
```

Thread 객체 생성

```
Thread thread  
= new Thread(new RunnableClass());
```

Thread 자식 객체 생성

```
Thread thread  
= new WorkerThread();
```

스레드 실행

```
thread.start();
```

스레드 생성과 실행

■ 스레드 생성 방법

- Thread 클래스가 제공하는 주요 생성자

생성자	설명
<code>Thread()</code>	스레드 객체를 생성한다.
<code>Thread(Runnable target)</code>	Runnable 구현 객체를 사용해 스레드 객체를 생성한다.
<code>Thread(Runnable target, String name)</code>	Runnable 구현 객체를 사용해 스레드 이름이 name인 스레드 객체를 생성한다.

- Thread 클래스가 제공하는 메서드

메서드 및 상수	설명
<code>static Thread currentThread()</code>	현재 실행 중인 스레드 객체의 참조 값을 반환한다.
<code>String getName()</code>	스레드의 이름을 반환한다.
<code>int getPriority()</code>	스레드의 우선순위 값을 반환한다.
<code>boolean isInterrupted()</code>	스레드가 인터럽트를 당했는지 여부를 반환한다.
<code>void setName()</code>	스레드의 이름을 설정한다.
<code>void setPriority()</code>	스레드의 우선순위를 설정한다.

스레드 생성과 실행

■ Runnable 구현 클래스에 스레드 실행 코드 추가

```
class MyRunnable implements Runnable {  
    public void run() {  
        // 스레드가 실행할 코드  
    }  
}
```

Runnable 구현 클래스이다.

스레드가 아니라
스레드가 수행할 코드

```
Thread thread = new Thread(new MyRunnable());  
thread.start();
```

Runnable 구현 클래스인
MyRunnable 객체이다.

스레드를 실행한다.

● Runnable 인터페이스의 익명 구현 객체화

```
Thread thread = new Thread(new Runnable() {  
    public void run() {  
        // 스레드가 실행할 코드  
    }  
});  
thread.start();
```

MyRunnable 객체와 동일하며 Runnable
인터페이스의 무명 구현 객체이다.

스레드 생성과 실행

■ Runnable 구현 클래스에 스레드 실행 코드 추가

- 익명 Thread 객체화

```
new Thread(new Runnable() {  
    public void run() {  
        // 스레드가 실행할 코드  
    }  
}).start();
```

- 람다식을 이용한 익명 Thread 객체화

```
new Thread(() -> {  
    // 스레드가 실행할 코드  
}).start();
```

람다식

- 예제 : [sec02/Thread1Demo](#), [sec02/Thread2Demo](#), [sec02/Thread3Demo](#)

안녕. 잘가. 잘가. 안녕. 잘가. 안녕. 잘가. 안녕. 잘가. 안녕.

스레드 생성과 실행

■ Thread 자식 클래스에 스레드 실행 코드 추가

```
class MyThread extends Thread {  
    public void run() {  
        // 스레드가 실행할 코드  
    }  
}
```

Thread의 자식 클래스이다.

Thread 클래스의 run() 메서드를 오버라이딩한다.

```
Thread thread = new MyThread();  
thread.start();
```

● 무명 Thread 객체화

```
new Thread() {  
    public void run() {  
        // 스레드가 실행할 코드  
    }  
}.start();
```

● 예제 : [sec02/Thread4Demo](#), [sec02/Thread5Demo](#)

스레드 생성과 실행

■ 스레드 풀을 이용한 스레드 실행

- 스레드의 개수가 많아지면 그에 따른 스레드 객체 생성과 스케줄링 등으로 CPU와 메모리에 많은 부하가 발생 -> 동시에 실행하는 스레드 개수를 제한할 필요
- 스레드 풀은 제한된 개수의 스레드를 JVM에 관리하도록 맡기는 방식
- JDK 5부터 ExecutorService 인터페이스와 Executors 클래스를 포함하는 java.util.concurrent 패키지를 이용해 지원
- Executor를 사용하는 대표적인 코드

```
Runnable task = () -> {    // 스레드가 수행할 코드  
};
```

```
Executor exec = Executors.newCachedThreadPool();  
exec.execute(task);
```

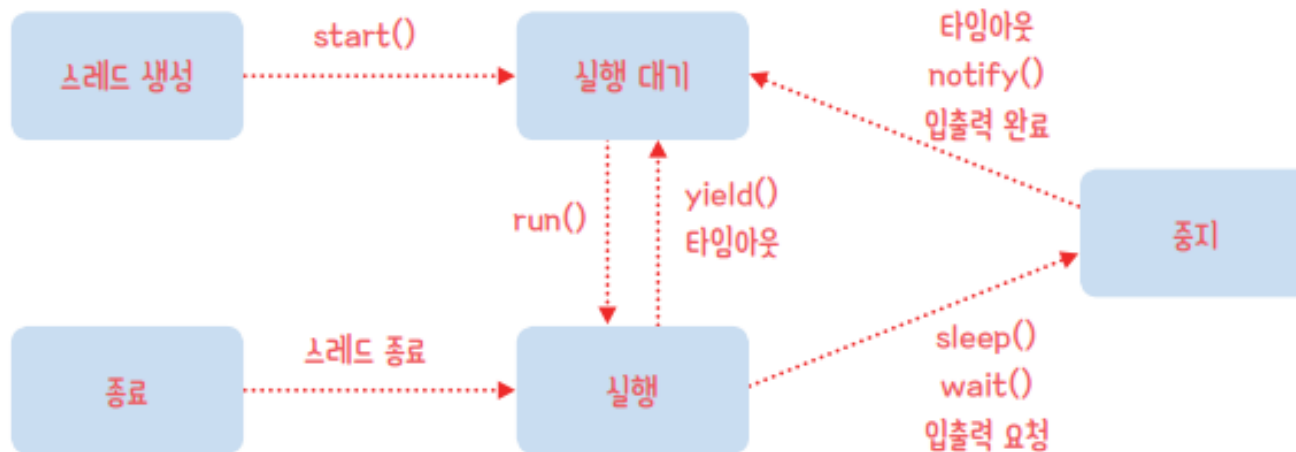
Executors 클래스의 다양한 팩토리 메서드 중 하나이다.

- 예제 : [sec02/ThreadPoolDemo](#)

스레드 상태와 제어

■ 스레드 생명 주기

- JVM은 스레드의 개수, 스레드의 상태, 우선순위 등 스레드와 관련된 모든 정보를 관리하며, 스레드 스케줄링을 수행
- 스레드의 생명 주기 흐름



스레드 상태와 제어

■ Thread 클래스의 멤버

상태	상수	설명
객체 생성	NEW	객체를 생성한 후 아직 시작하지 않은 상태
실행 대기	RUNNABLE	실행 준비가 된 상태
중지	BLOCKED	입출력 등 락(Lock)이 풀릴 때까지 기다리는 상태
	WAITING	다른 스레드가 통지할 때까지 기다리는 상태
	TIME_WAITING	주어진 시간 동안 기다리는 상태
종료	TERMINATED	실행을 마친 상태

메서드	설명
void interrupt()	실행 중인 스레드에 인터럽트를 걸어 중지시킨다.
void join()	주어진 시간이 지나거나 대응하는 스레드가 종료될 때까지 대기시킨다.
void resume()	중지 상태의 스레드를 실행 대기 상태로 전환시킨다.
static void sleep()	주어진 시간 동안 중지한다.
void start()	스레드를 실행 대기시킨다.
void stop()	스레드를 종료한다.
void suspend()	스레드를 중지한다.
static void yield()	우선순위가 동일한 스레드에 실행을 양보한다.

스레드 상태와 제어

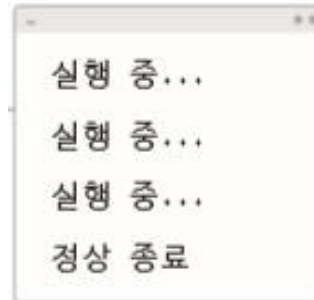
■ 스레드 종료

- run() 메서드 실행 종료를 위한 stop() 메서드
 - 호출 즉시 스레드를 종료하므로 사용 중인 자원 낭비 가능성
 - JDK 2부터는 stop() 메서드를 가급적 사용하지 않도록 권고
- 스레드를 안전하게 종료하는 방법
 - 반복문의 조건
 - interrupt() 메서드

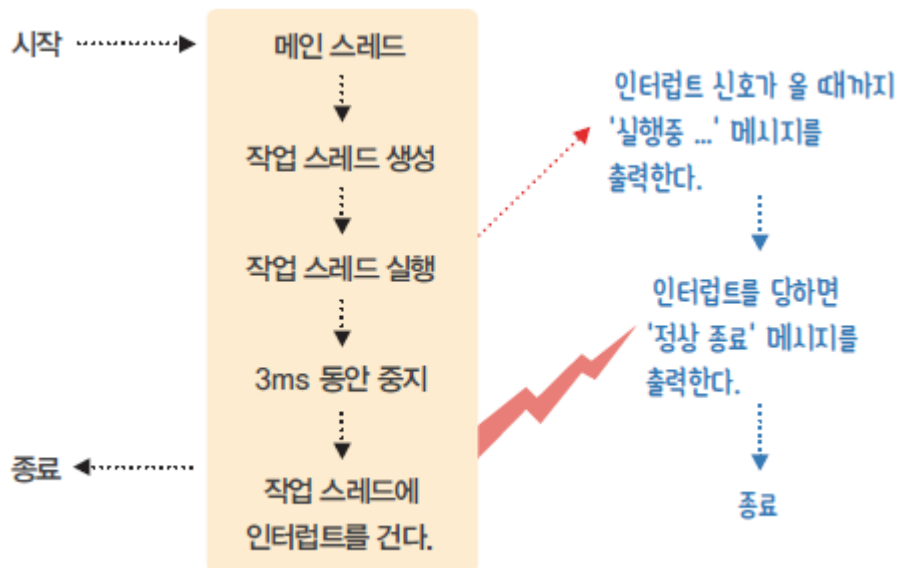
스레드 상태와 제어

■ 스레드 종료

- 예제(반복문의 조건) : [sec03/StopDemo](#)



- 예제(interrupt) : [sec03/Interrupt1Demo](#) , [sec03/Interrupt2Demo](#)



스레드 상태와 제어

■ 다른 스레드를 종료할 때까지 대기

- join() 메서드 사용(예제) : [sec03/JoinDemo](#)

- 주석이 있을 때	- 주석을 제거했을 때
총합 : 0	스레드 t가 끝날 때까지 대기...
	총합 : 5050

스레드 상태와 제어

■ 스레드 우선순위

- 멀티태스킹에서 스레드 개수가 CPU의 코어 개수보다 많다면 스레드 스케줄링 필요
- 자바는 우선순위 방식과 순환할당 방식을 사용
- Thread 클래스가 제공하는 우선순위 상수

우선순위 상수	설명
MAX_PRIORITY	최고 우선순위인 10을 나타내는 상수
NORM_PRIORITY	중간 우선순위인 5를 나타내는 상수
MIN_PRIORITY	최저 우선순위인 1을 나타내는 상수

- 예제 : [sec03/PriorityDemo](#)

```
급한 -> 느긋한 -> 급한 -> 느긋한 -> 급한 -> 느긋한 -> 느긋한 -> 급한 -> 급한 ->
느긋한 ->
```


스레드 상태와 제어

■ 데몬 스레드

- 일반적으로 스레드는 독립적으로 수행하기 때문에 메인 스레드를 종료해도 작업 스레드는 계속 실행
- 데몬 스레드 : 다른 스레드를 모두 종료하면 자동으로 종료되는 스레드
- Thread 클래스가 제공하는 데몬 스레드 관련 메서드

메서드	설명
<code>void setDaemon(boolean status)</code>	데몬 스레드 여부를 설정한다. <code>start()</code> 메서드를 호출하기 전에 사용해야 한다.
<code>boolean isDaemon()</code>	데몬 스레드 여부를 반환한다.

- 예제 : [sec03/DaemonDemo](#)

- 주석이 있을 때

메인 스레드가 끝났습니다.
작업 스레드
작업 스레드
작업 스레드

- 주석을 제거했을 때

메인 스레드가 끝났습니다.
작업 스레드

동기화와 협업

■ 스레드 동기화

- 다중 스레드 프로그래밍 환경에서 임계영역이 있다면 동기화 문제가 발생
- 자바는 임계영역을 동기화하려고 synchronized 키워드를 제공
- 스레드가 synchronized로 지정된 동기화 블록에 진입하면 락을 걸고, 그 블록을 벗어날 때 락을 푼다.
- 동기화 블록에 진입한 스레드가 코드를 실행할 동안 다른 스레드는 동기화 블록 앞에서 락이 풀릴 때까지 대기해야 한다.



동기화와 협업

■ 스레드 동기화

- 두 가지 방법

```
public synchronized void 메서드( ) {  
    // 임계영역 코드  
}
```

(a) 동기화 메서드

```
synchronized (공유객체) {  
    // 임계영역 코드  
}
```

(b) 동기화 블록

동기화와 협업

■ 스레드 동기화

- 예제 : [sec04/SharedCar](#), [sec04/SynchroDemo](#)

- 동기화하지 않을 때

뽀지리님이 자동차에 탔습니다.
문둥이님이 자동차에 탔습니다.
문둥이님이 자동차에 탔습니다.
뽀지리님이 자동차를 운전합니다.
뽀지리님이 자동차를 운전합니다.
뽀지리님이 서울에 도착했습니다.
깡깡이님이 자동차를 운전합니다.
깡깡이님이 자동차를 운전합니다.
깡깡이님이 자동차를 운전합니다.
깡깡이님이 광주에 도착했습니다.
문둥이님이 자동차를 운전합니다.
문둥이님이 자동차를 운전합니다.
문둥이님이 부산에 도착했습니다.

- 동기화할 때

문둥이님이 자동차에 탔습니다.
문둥이님이 자동차를 운전합니다.
문둥이님이 자동차를 운전합니다.
문둥이님이 자동차를 운전합니다.
문둥이님이 부산에 도착했습니다.
깡깡이님이 자동차에 탔습니다.
깡깡이님이 자동차를 운전합니다.
깡깡이님이 자동차를 운전합니다.
깡깡이님이 광주에 도착했습니다.
뽀지리님이 자동차에 탔습니다.
뽀지리님이 자동차를 운전합니다.
뽀지리님이 자동차를 운전합니다.
뽀지리님이 서울에 도착했습니다.

동기화와 협업

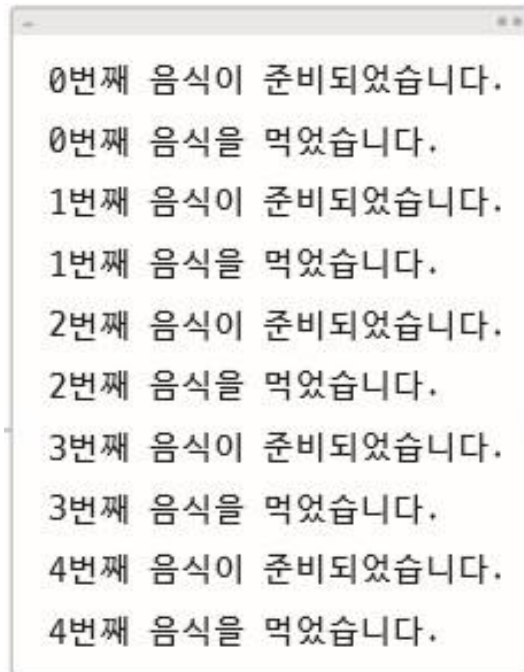
■ 대기와 통보

- 동기화 메서드나 블록을 이용하면 공유 자원은 보호할 수 있지만, 특정 스레드가 필요 없이 오랜 시간 동안 락을 걸 수 있다
- Object 클래스는 객체의 락과 관련된 `wait()`, `notify()`, `notifyAll()` 메서드를 제공해서 이런 상황을 해결
- 이 3개의 메서드는 소유한 락을 해제하거나 대기하다가 락을 소유하므로 동기화된 코드 내부에서만 의미가 있다.
- 예제 : [sec04/WaitNotifyDemo](#)

동기화와 협업

■ 스레드 협업

- 예제 : [sec04/Dish](#), [sec04/Cook](#), [sec04/Customer](#), [sec04/DishDemo](#)



0번째 음식이 준비되었습니다.
0번째 음식을 먹었습니다.
1번째 음식이 준비되었습니다.
1번째 음식을 먹었습니다.
2번째 음식이 준비되었습니다.
2번째 음식을 먹었습니다.
3번째 음식이 준비되었습니다.
3번째 음식을 먹었습니다.
4번째 음식이 준비되었습니다.
4번째 음식을 먹었습니다.