

14장. 스레드



Thread



멀티 쓰레드

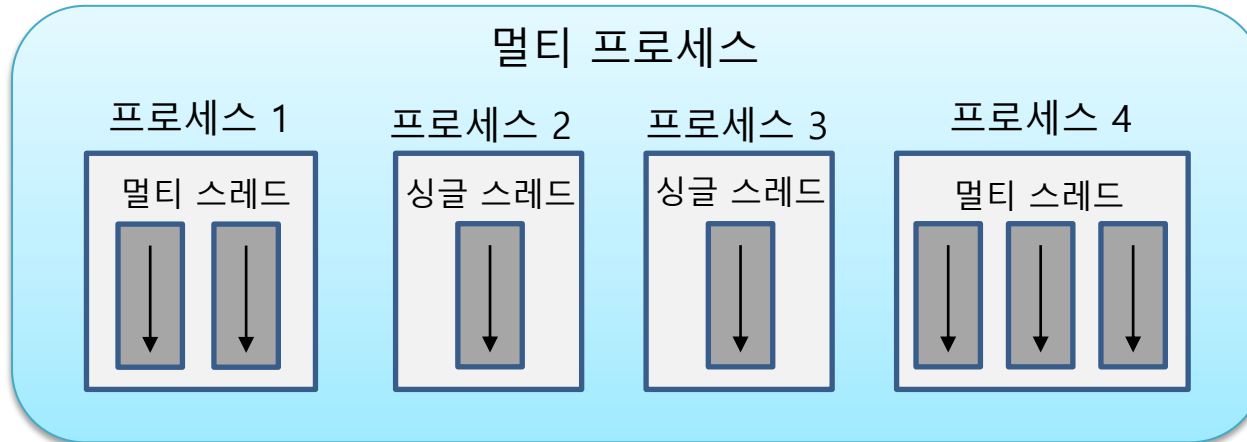
■ 프로세스와 쓰레드

프로세스(Process)

- 실행 중인 하나의 프로그램을 말한다.(하드디스크 -> 주기억장치)

멀티 태스킹(multi tasking)

- 두가지 이상의 작업을 동시에 처리하는 것.
- 멀티 프로세스 : 독립적으로 프로그램들을 실행하고 여러가지 작업처리.
- 멀티 스레드 : 한 개를 프로그램을 실행하고 내부적으로 여러 가지 작업 처리



작업 관리자

파일(F) 옵션(O) 보기(V)

프로세스 성능 앱 기록 시작프로그램 사용자 세부 정보 서비스

이름	상태	3%	36%	1%	0%
		CPU	메모리	디스크	네트워크
앱 (3)					
Microsoft PowerPoint(32비트)		0.3%	59.3MB	0MB/s	0Mbps
작업 관리자		0.8%	23.5MB	0MB/s	0Mbps
합계 도구		0%	3.0MB	0MB/s	0Mbps
백그라운드 프로세스 (47)					
AhnLab Safe Transaction Appli...		0.3%	1.6MB	0MB/s	0Mbps
AhnLab Safe Transaction Appli...		0.3%	0.6MB	0MB/s	0Mbps
AMD External Events Client Mo...		0.1%	1.6MB	0MB/s	0Mbps
AMD External Events Service M...		0%	1.0MB	0MB/s	0Mbps
AMD ReLive: Desktop Overlay		0%	1.0MB	0MB/s	0Mbps
AMD ReLive: Host Application		0%	2.3MB	0MB/s	0Mbps
ASDF Service Application		0%	5.7MB	0.1MB/s	0Mbps
CCDaemon.exe(32비트)		0%	1.0MB	0MB/s	0Mbps

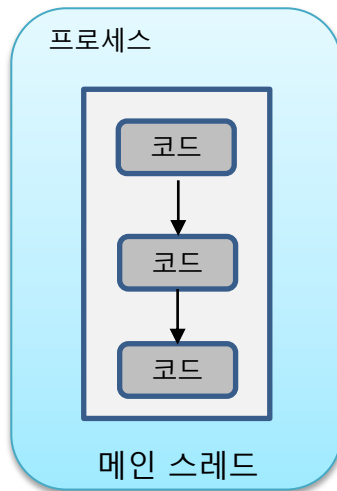


멀티 쓰레드

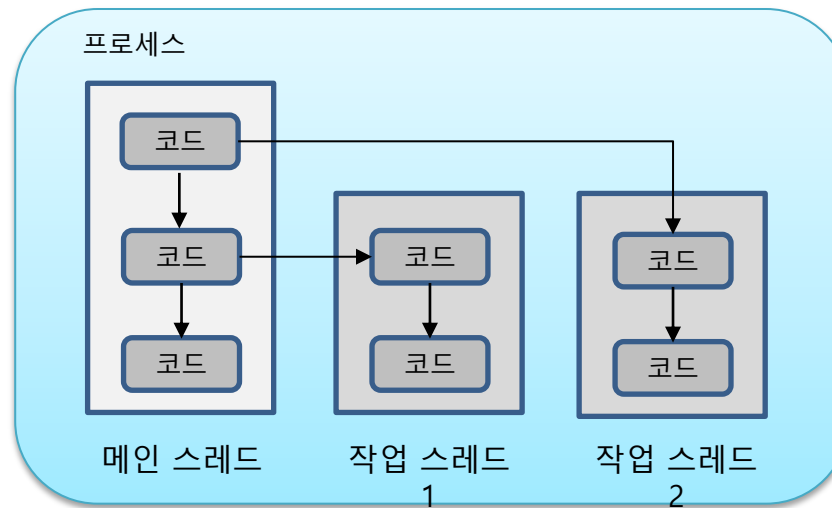
메인(main) 쓰레드

- 모든 자바 프로그램은 메인 쓰레드가 main() 메소드를 실행하면서 시작된다.
- main() 메소드의 첫 코드부터 아래로 순차적으로 실행한다.
- main()메소드의 마지막 코드를 실행하거나, return문을 만나면 실행이 종료된다.

싱글 쓰레드 애플리케이션



멀티 쓰레드 애플리케이션



프로세스의 종료

- 싱글 쓰레드 : 메인 쓰레드가 종료되면 프로세스도 종료된다.
- 멀티 쓰레드 : 실행 중인 쓰레드가 하나라도 있다면, 프로세스는 종료되지 않는다.
(메인 쓰레드가 작업스레드보다 먼저 종료되는 경우도 있다.)



멀티 스레드

Thread 클래스

- 자바에서는 작업 스레드도 객체로 생성되기 때문에 클래스가 필요하다.
- Java.lang.Thread 클래스를 직접 객체화하거나, Thread를 상속해서 하위 클래스를 만들어 생성

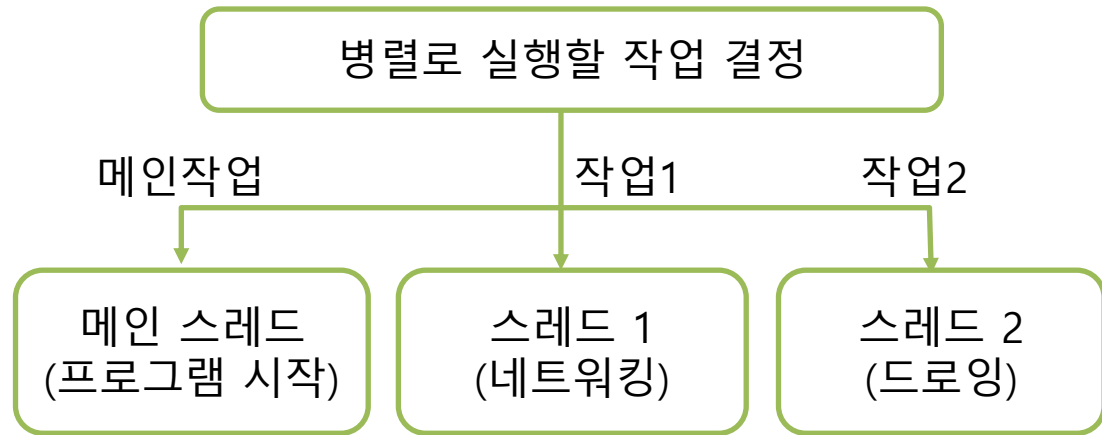
Class Thread

java.lang.Object
java.lang.Thread

All Implemented Interfaces:
Runnable

Direct Known Subclasses:
ForkJoinWorkerThread

```
public class Thread  
extends Object  
implements Runnable
```

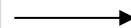


멀티 쓰레드

방법 1) Runnable 인터페이스를 구현한 Task 클래스

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

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



thread.start()

쓰레드 시작(실행)

방법 2) 익명 구현 객체(구현 클래스 없이 직접 생성)

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



thread.start()



멀티 스레드

메인 스레드만 이용한 경우

```
public class BeepPrintTest {  
    public static void main(String[] args) {  
        //메인 스레드만 실행  
        //"띵" 문자를 5번 출력하기 -> 1초 대기 간격  
        for(int i=0; i<5; i++) {  
            System.out.println("띵");  
            try {  
                Thread.sleep(1000); //1000ms -> 1s(1초)  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
  
        //"띵" 소리를 5번 재생하기  
        Toolkit toolkit = Toolkit.getDefaultToolkit();  
        for(int i=0; i<5; i++) {  
            toolkit.beep();  
            try {  
                Thread.sleep(1000);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```

Module java.desktop

Package java.awt

Class Toolkit

java.lang.Object

java.awt.Toolkit

public abstract class Toolkit

extends Object



멀티 스레드

비프음을 들려주는 작업 정의

```
package thread;

import java.awt.Toolkit;

public class BeepTask implements Runnable{
    //Runnable 인터페이스를 구현한 BeepTask 클래스 생성
    //비프음을 재생하는 작업 정의
    @Override
    public void run() {
        Toolkit toolkit = Toolkit.getDefaultToolkit();
        for(int i=0; i<5; i++) {
            toolkit.beep();
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```



멀티 스레드

메인 스레드와 작업 스레드가 동시에 실행

```
public class BeepPrintTest2 {  
    public static void main(String[] args) {  
        //메인 스레드와 작업 스레드가 동시에 실행  
        Runnable beepTask = new BeepTask();  
        Thread thread = new Thread(beepTask);  
        thread.start(); //쓰레드 시작(실행)  
  
        for(int i=0; i<5; i++) {  
            System.out.println("땡");  
            try {  
                Thread.sleep(1000); //1000ms -> 1s(1초)  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```

EO EO EO EO EO



멀티 스레드

익명 객체로 구현

```
public class BeepPrintTest3 {  
    public static void main(String[] args) {  
        //메인 스레드와 작업 스레드가 동시에 실행  
        //익명 객체로 구현  
        Thread thread = new Thread(new Runnable() {  
            @Override  
            public void run() {  
                Toolkit toolkit = Toolkit.getDefaultToolkit();  
                for(int i=0; i<5; i++) {  
                    toolkit.beep();  
                    try {  
                        Thread.sleep(1000);  
                    } catch (InterruptedException e) {  
                        e.printStackTrace();  
                    }  
                }  
            }  
        });  
        thread.start(); //스레드 시작  
  
        for(int i=0; i<5; i++) {  
            System.out.println("땡");  
            try {  
                Thread.sleep(1000); //1000ms -> 1s(1초)  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```



멀티 스레드

람다식으로 구현

```
public class BeepPrintTest4 {  
    public static void main(String[] args) {  
        //메인 스레드와 작업 스레드가 동시에 실행  
        Thread thread = new Thread(()->{  
            Toolkit toolkit = Toolkit.getDefaultToolkit();  
            for(int i=0; i<5; i++) {  
                toolkit.beep();  
                try {  
                    Thread.sleep(1000);  
                } catch (InterruptedException e) {  
                    e.printStackTrace();  
                }  
            }  
        });  
        thread.start();  
  
        for(int i=0; i<5; i++) {  
            System.out.println("띵");  
            try {  
                Thread.sleep(1000); //1000ms -> 1s(1초)  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```



멀티 스레드

시계 구현

```
public class Watch {  
  
    public static void main(String[] args) {  
        // 디지털 시계  
        while(true) {  
            Calendar c = Calendar.getInstance();  
            int year = c.get(Calendar.YEAR);  
            int month = c.get(Calendar.MONTH);  
            int date = c.get(Calendar.DATE);  
            int amPm = c.get(Calendar.AM_PM);  
            int hour = c.get(Calendar.HOUR);  
            int minute = c.get(Calendar.MINUTE);  
            int second = c.get(Calendar.SECOND);  
            //int msec = c.get(Calendar.MILLISECOND);  
            String am_pm = (amPm == Calendar.AM)? "AM" : "PM";  
            //int day = c.get(Calendar.DAY_OF_WEEK);  
  
            System.out.println(year + "년 " + (month+1) + "월 " + date + "일 " +  
                               am_pm + " " + hour + "시 " + minute + "분 " + second + "초");  
            try {  
                Thread.sleep(1000);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```

```
2021년 9월 4일 PM 9시 49분 14초  
2021년 9월 4일 PM 9시 49분 15초  
2021년 9월 4일 PM 9시 49분 16초  
2021년 9월 4일 PM 9시 49분 17초  
2021년 9월 4일 PM 9시 49분 18초  
2021년 9월 4일 PM 9시 49분 19초  
2021년 9월 4일 PM 9시 49분 20초  
2021년 9월 4일 PM 9시 49분 21초  
2021년 9월 4일 PM 9시 49분 22초  
2021년 9월 4일 PM 9시 49분 23초  
2021년 9월 4일 PM 9시 49분 24초  
2021년 9월 4일 PM 9시 49분 25초
```

