15장. 스레드

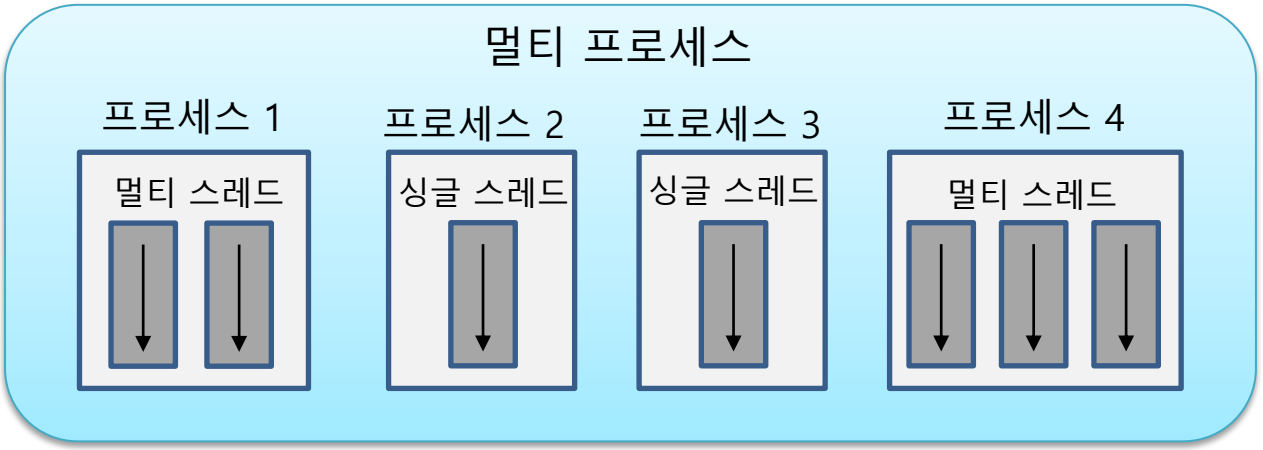
Thread

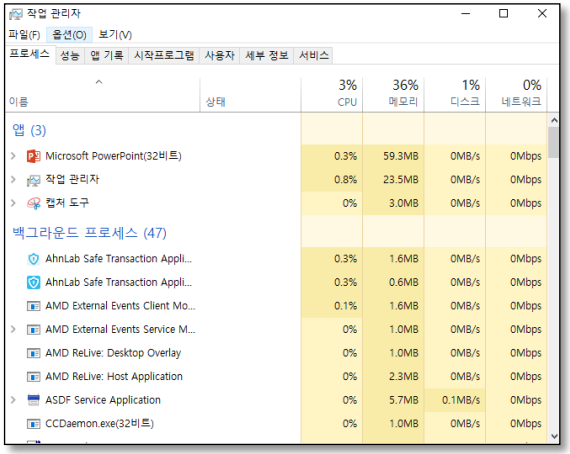
멀티 쓰레드

- **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



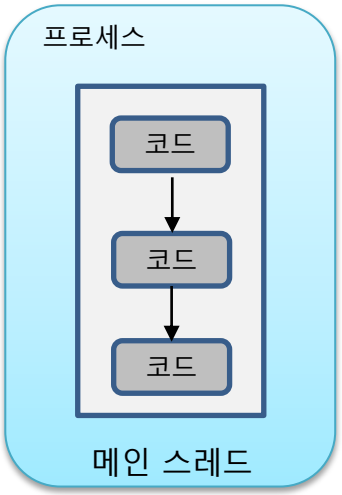


멀티 쓰레드

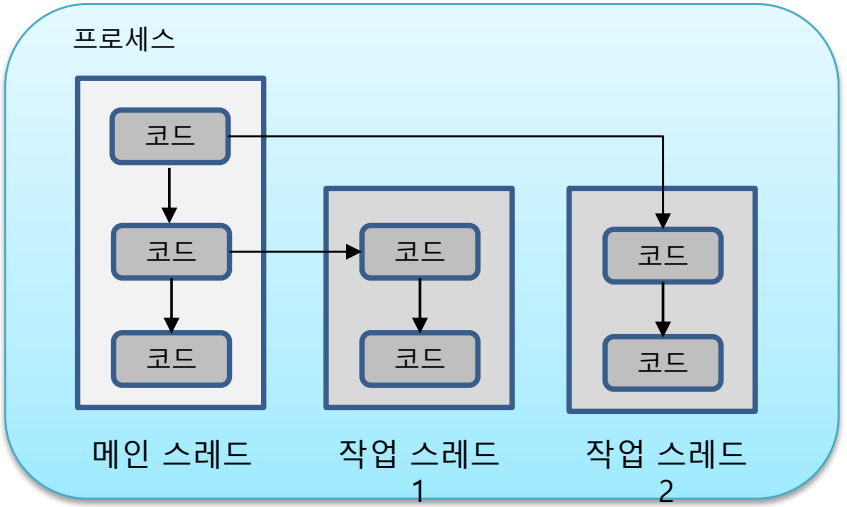
메인(main) 스레드

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

싱글 스레드 애플리케이션



멀티 스레드 애플리케이션



프로세스의 종료

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

Thread 클래스

작업 스레드 생성과 실행

- 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

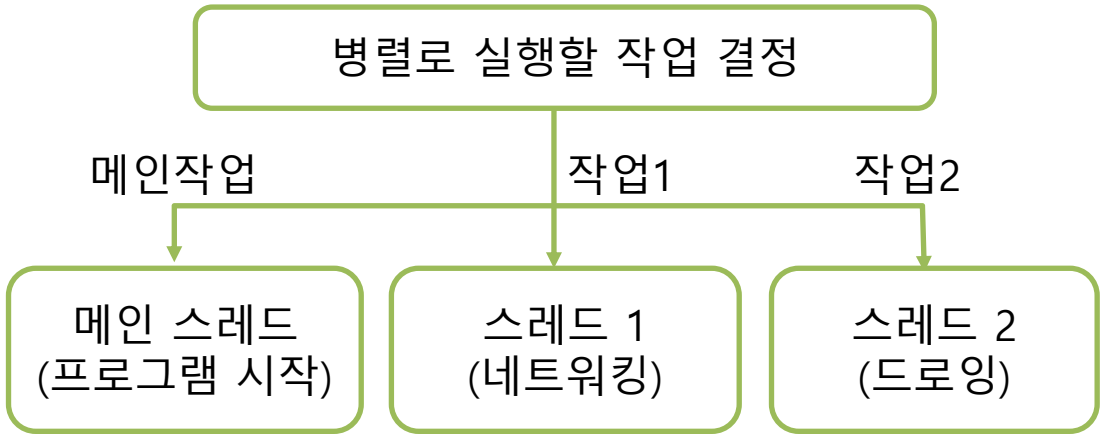
All Implemented Interfaces:

Runnable

Direct Known Subclasses:

ForkJoinWorkerThread

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



Thread 클래스

방법 1) Thread 클래스로 부터 직접 생성

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

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

→ **thread.start()**

쓰레드 시작(실행)

Thread 클래스

방법 2) Thread 하위 클래스로 부터 생성 (상속)

```
public class 스레드 클래스 extends Thread{
    …
}

Thread thread = new 스레드클래스();
```

→ **thread.start()**

쓰레드 시작(실행)

멀티 쓰레드

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 쓰레드

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 쓰레드

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("띵");
            try {
                Thread.sleep(1000); //1000ms -> 1s(1초)
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
띵
띵
띵
띵
띵
```

멀티 쓰레드

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("띵");
            try {
                Thread.sleep(1000); //1000ms -> 1s(1초)
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

멀티 쓰레드

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();
            }
        }
```

Thread 이름

Thread 이름

스레드는 자신의 이름을 가지고 있다. 메인 스레드는 'main'이라는 이름을 가지고 있고,
직접 생성한 스레드는 자동으로 Thread-n 이라는 이름으로 설정된다.
다른 이름으로 설정하고 싶다면 Thread 클래스의 setName() 메소드로 변경한다.

```
public class ThreadA extends Thread{

    public ThreadA() {
        setName("ThreadA");
    }

    @Override
    public void run() {
        for(int i=0; i<2; i++) {
            System.out.println(getName() + "가 출력한 내용");
        }
    }
}
```

```
ThreadA가 출력한 내용
작업 스레드 이름: Thread-1
Thread-1가 출력한 내용
Thread-1가 출력한 내용
```

Thread 이름

Thread 이름을 설정하지 않은 경우

```
public class ThreadB extends Thread{

    @Override
    public void run() {
        for(int i=0; i<2; i++) {
            System.out.println(getName() + "가 출력한 내용");
        }
    }
}
```

Thread 이름

```
public class ThreadExample {

    public static void main(String[] args) {
        Thread mainThread = Thread.currentThread();
        System.out.println("프로그램 시작 스레드 이름: " + mainThread.getName());

        ThreadA threadA = new ThreadA();
        System.out.println("작업 스레드 이름: " + threadA.getName());
        threadA.start();

        ThreadB threadB = new ThreadB();
        System.out.println("작업 스레드 이름: " + threadB.getName());
        threadB.start();
    }
}
```