



# java 21강 - 쓰레드

---

양 명 속

[[now4ever7@gmail.com](mailto:now4ever7@gmail.com)]



# 목차

---

- 프로세스와 쓰레드

# 프로세스

- 자바 프로그램의 실행이 요청되면 다음의 형태로 메모리 공간이 할당되고, 이 메모리를 기반으로 프로그램이 실행된다.
- 프로세스(Process) - 할당된 메모리 공간을 기반으로 **실행 중에 있는 프로그램**



- 메소드 영역 : 클래스(\*.class)에 대한 정보, static 변수
- 스택 영역 : 지역변수, 매개변수 (메서드의 작업에 필요한 메모리 공간을 제공)
- 힙 영역 : 인스턴스

프로세스에 할당된 메모리  
(자바 가상머신의 메모리 모델)

# 프로세스와 쓰레드

- 프로세스(process)란
  - 실행 중인 프로그램(program)
  - 프로그램을 실행하면 OS로부터 실행에 필요한 자원(메모리)을 할당받아 프로세스가 됨



- 프로세스 = 프로그램을 수행하는 데 필요한 데이터와 메모리 등의 자원 + 쓰레드로 구성
- 쓰레드 - 프로세스내에서 실제로 작업을 수행
  - 모든 프로세스에는 최소한 하나 이상의 쓰레드가 존재함
  - 둘 이상의 쓰레드를 가진 프로세스를 멀티쓰레드 프로세스라고 함

쓰레드 - 프로세스라는 작업공간(공장)에서 작업을 처리하는 일꾼  
별도의 실행 흐름  
하나의 프로세스 내에서 동시에 수행되는 작업단위

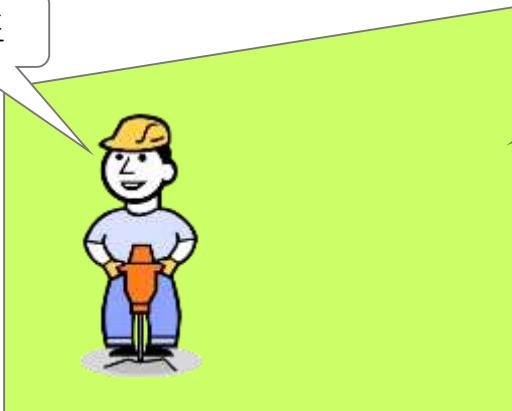
# 프로세스와 쓰레드

- 멀티 태스킹(Multi-tasking, 다중작업) - 여러 개의 프로세스가 동시에 실행되는 것처럼 보이는 것
- 멀티 쓰레딩 - 하나의 프로세스 내에서 여러 쓰레드가 동시에 작업을 수행하는 것

싱글 쓰레드 프로세스 = 자원 + thread

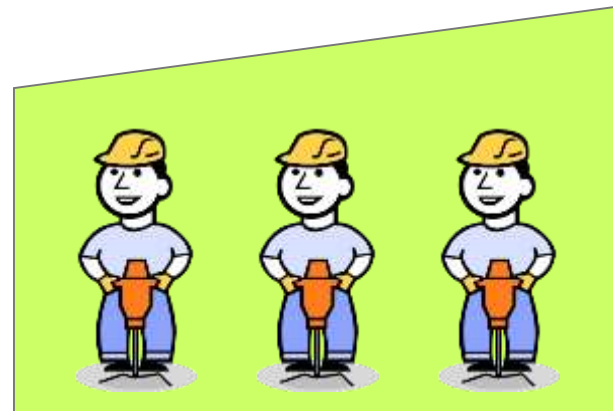
멀티 쓰레드 프로세스 = 자원 + thread + thread +....

쓰레드



싱글 쓰레드 프로세스

프로세스



멀티 쓰레드 프로세스



# 프로세스와 쓰레드

- 싱글 쓰레드 - 실행 흐름이 하나밖에 없으므로 한 번에 하나의 작업밖에 하지 못함
- 멀티 쓰레드 - 하나의 응용 프로그램에 두 개 이상의 쓰레드가 동시에 실행될 수 있음
  - 하나의 응용 프로그램이 두 개의 작업을 병렬적으로 처리하는 것이 가능
  - 백그라운드에서 틈틈이 해야 할 작업, 시간이 오래 걸리는 계산작업, 인쇄를 위한 스푼링 작업 등
  - 백그라운드의 쓰레드가 필요한 작업을 하는 동안 주 쓰레드는 계속 사용자의 입력을 받아 처리할 수 있으므로 프로그램의 반응성이 좋아지고, 여러 가지 작업을 동시 다발적으로 처리할 수 있음
- 예) 메신저 - 채팅하면서 파일 다운로드 받거나, 음성 대화 나눌 수 있는 것이 가능한 이유
  - <= 멀티쓰레드로 작성되어 있기 때문
- 예) 워드 - 자동교정(맞춤법검사), 자동저장, 워드프로세서 작업 등이 동시에 가능



# 프로세스와 쓰레드

- 멀티 쓰레딩의 장점

- CPU의 사용률을 향상시킨다
- 자원을 보다 효율적으로 사용할 수 있다
- 사용자에게 대한 응답성이 향상된다
- 작업이 분리되어 코드가 간결해진다

- 멀티쓰레딩의 단점

- 멀티쓰레드 프로세스는 여러 쓰레드가 같은 프로세스 내에서 자원을 공유하면서 작업을 하기 때문에 발생할 수 있는 동기화(synchronization), 교착상태(deadlock)와 같은 문제들을 고려해서 신중히 프로그래밍해야 함
- 교착상태 - 두 쓰레드가 자원을 점유한 상태에서 서로 상대방이 점유한 자원을 사용하려고 기다리느라 진행이 멈춰있는 상태를 말함



# 쓰레드의 구현과 실행

- 쓰레드를 구현하는 방법
  - 1. Thread 클래스를 상속받는 방법
  - 2. Runnable 인터페이스를 구현하는 방법
    - 쓰레드 클래스가 상속해야 할 또 다른 클래스가 존재하는 경우 사용

## 1. Thread 클래스를 상속

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

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



# 쓰레드의 구현과 실행

## 2. Runnable 인터페이스를 구현

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

```
public interface Runnable{  
    public abstract void run();  
}
```

```
MyThread r = new MyThread();  
Thread t = new Thread(r); //Thread의 생성자 : Thread(Runnable target)  
t.start();
```

- Runnable 인터페이스
  - run() 메서드만 정의되어 있는 간단한 인터페이스
- 쓰레드 구현
  - 쓰레드를 통해 작업하고자 하는 내용으로 run()의 몸통을 채우기만 하면 됨



## 예제

```
main메서드입니다.  
현재 실행중인 스레드 이름: main  
실행중인 스레드 갯수: 1  
변경된 스레드 이름: Prime
```

```
public class ThreadTest {  
    public static void main(String[] args) {  
        System.out.println("main메서드입니다.");  
        Thread current=Thread.currentThread();  
        System.out.println("현재 실행중인 스레드 이름: "+current.getName());  
        int cnt=Thread.activeCount();  
        System.out.println("실행중인 스레드 갯수: "+cnt);  
  
        Thread.currentThread().setName("Prime");  
        System.out.println("변경된 스레드 이름: "+Thread.currentThread().getName());  
    }  
}
```

# 예제 1

- Runnable 인터페이스를 구현한 경우 인스턴스 생성방법  
Runnable 인터페이스를 구현한 클래스의 인스턴스를 생성한 다음,  
이 인스턴스를 가지고 Thread 클래스의 인스턴스를 생성할 때  
생성자의 매개변수로 제공해야 함

```
class ThreadEx1 {
    public static void main(String args[]) {
        MyThread1 t1 = new MyThread1("나의 쓰레드1");
        MyThread2 r = new MyThread2();
        Thread t2 = new Thread(r); // Thread의 생성자: Thread(Runnable target)

        t1.start();
        t2.start();
        System.out.println("main 쓰레드 종료!");
    }
}

class MyThread1 extends Thread {
    public void run() {
        for(int i=0; i < 5; i++) {
            System.out.println(getName()); // 조상인 Thread의 getName()을 호출
        }
    }
}

class MyThread2 implements Runnable {
    public void run() {
        for(int i=0; i < 5; i++) {
            // Thread.currentThread() - 현재 실행중인 Thread를 반환한다.
            System.out.println(Thread.currentThread().getName());
        }
    }
}
```

```
public MyThread1(String name){
    super(name);
    //setName(name);
}
```

- Thread 클래스를 상속받으면, Thread 클래스의 메서드를 직접 호출  
할 수 있다 => getName()

```
main 쓰레드 종료!
Thread-0
나의 쓰레드1
나의 쓰레드1
나의 쓰레드1
나의 쓰레드1
나의 쓰레드1
Thread-0
Thread-0
Thread-0
Thread-0
```

- Runnable을 구현하면 Thread 클래스의 static 메서드인 currentThread()  
를 호출하여 쓰레드에 대한 참조를 얻어 와야만 Thread 클래스의 메서드  
호출 가능



# 예제 1

## ■ Thread 클래스의 메서드

Thread **currentThread()** – 현재 실행중인 스레드의 참조를 반환  
String **getName()** – 스레드의 이름을 반환

## ■ 스레드의 이름을 지정 또는 변경하는 방법

Thread(Runnable target, String name)  
Thread(String name)  
void setName(String name)

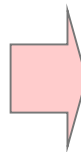
## ■ 스레드의 이름을 지정하지 않으면 'Thread-번호'의 형식으로 이름이 정해짐

- **main** 메서드가 수행을 마쳤더라도 다른 스레드가 아직 작업을 마치지 않은 상태라면 프로그램이 종료되지 않음
- 실행 중인 사용자 스레드가 하나도 없을 때 프로그램은 종료됨

# 예제 1

- start()
  - 쓰레드를 생성한 다음에는 start()를 호출해야만 비로소 작업을 시작하게 됨
- 한 번 사용한 쓰레드는 다시 재사용할 수 없다
  - 즉, 하나의 쓰레드에 대해 start()가 한 번만 호출될 수 있다는 뜻
- 쓰레드의 작업이 한 번 더 수행되기를 원한다면 새로운 쓰레드를 생성한 다음에 start()를 호출해야 함
  - 하나의 쓰레드에 대해 start()를 두 번 이상 호출하면 실행시에 IllegalStateException 이 발생함

```
ThreadEx1 t1 = new ThreadEx1();  
t1.start();  
t1.start(); //exception 발생
```



```
ThreadEx1 t1 = new ThreadEx1();  
t1.start();  
  
t1 = new ThreadEx1();  
t1.start();
```



# start()와 run()

## ■ run() 메서드 직접 호출

- 단순한 메서드의 호출일 뿐, 쓰레드의 생성으로 이어지지 않음
- 쓰레드는 자신만의 메모리 공간을 할당 받아서 별도의 실행흐름을 형성함

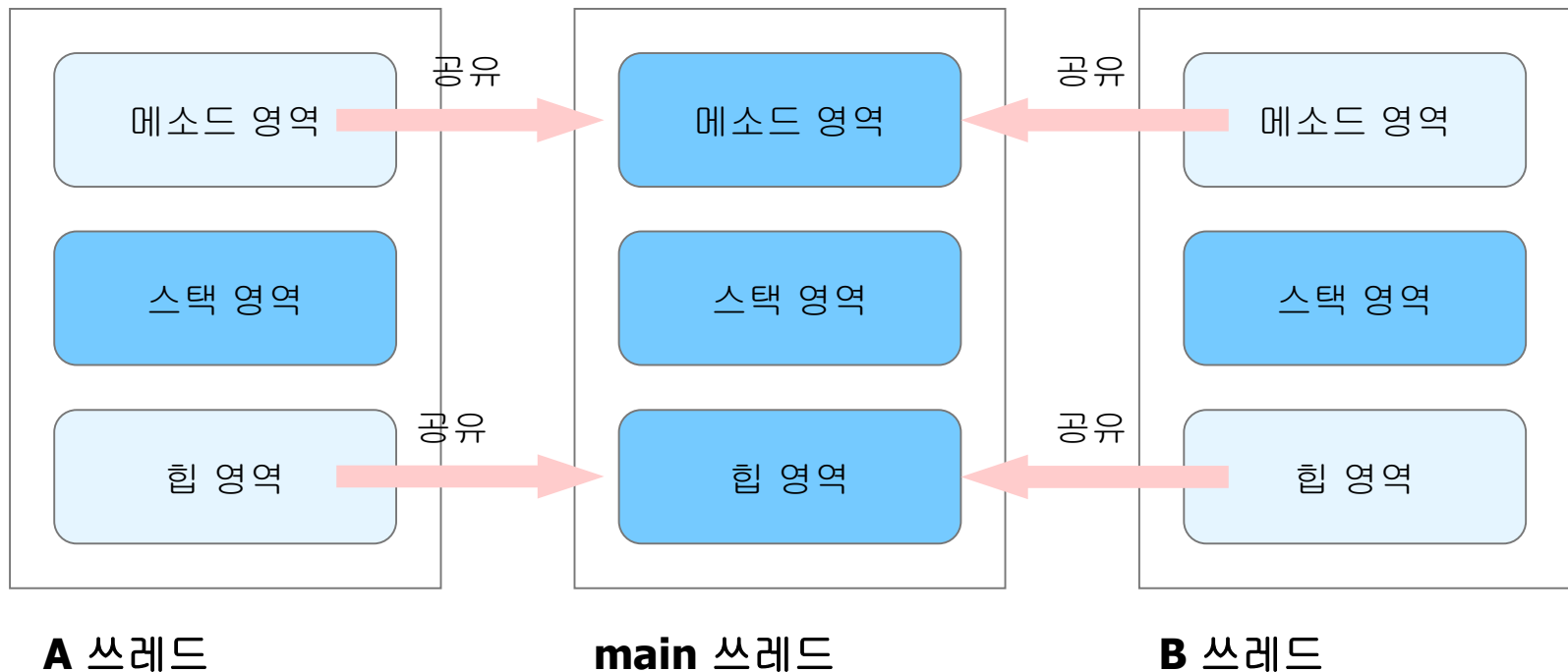
## ■ start() 메서드

- 메모리 공간의 할당 등 쓰레드의 실행을 위한 기반을 마련한 다음에 run 메서드를 대신 호출해 줌

- 모든 쓰레드는 cpu를 공유함
- run 메서드의 실행이 완료되면, 해당 쓰레드는 종료되고, 소멸됨
- 쓰레드 - 별도의 실행흐름을 형성하기 위해서 자바 가상머신에 의해 만들어진 모든 리소스와 각종 정보들(메모리 공간의 할당, CPU를 나눠쓰기 위한 각종 정보들)을 총칭해서 쓰레드라 함

# 쓰레드의 메모리 구성

- 쓰레드가 생성되면 가상머신은 쓰레드의 실행을 위한 별도의 메모리 공간을 할당함
- 쓰레드 - 별도의 실행흐름 형성
  - main 메서드와는 다른 실행흐름을 형성하기 위해서는 **별도의 스택이 쓰레드에게 할당**되어야 함
  - 예) main 쓰레드 이외에 두 개의 쓰레드가 추가로 생성된 경우 쓰레드에 할당되는 메모리





# 쓰레드의 메모리 구성

---

- 모든 쓰레드는 자신의 스택을 할당 받는다
- 그러나 힙과 메소드 영역은 모든 쓰레드가 공유한다
- 힙 영역이 공유됨
  - 모든 쓰레드가 동일한 힙 영역에 접근 가능
  - "A 쓰레드가 만든 인스턴스의 주소값만 알면 B 쓰레드도 A 쓰레드가 만든 인스턴스에 접근 가능하다."

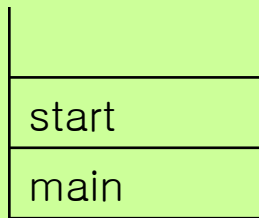


# start()와 run()

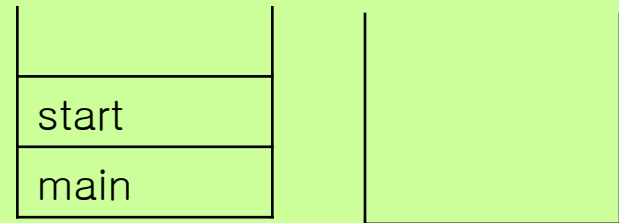
## ■ 쓰레드가 실행되는 과정

1. main 메서드에서 쓰레드의 start 메서드를 호출한다
2. start 메서드는 쓰레드가 작업을 수행하는데 사용될 새로운 호출스택을 생성한다
3. 생성된 호출스택에 run 메서드를 호출해서 쓰레드가 작업을 수행하도록 한다
4. 이제는 호출스택이 2개이기 때문에 스케줄러가 정한 순서에 의해서 번갈아 가면서 실행된다

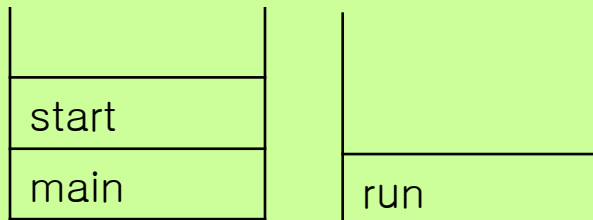
### 1. Call stack



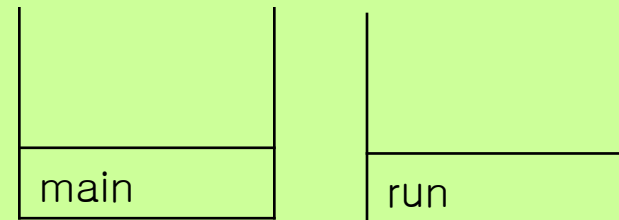
### 2. Call stack



### 3. Call stack

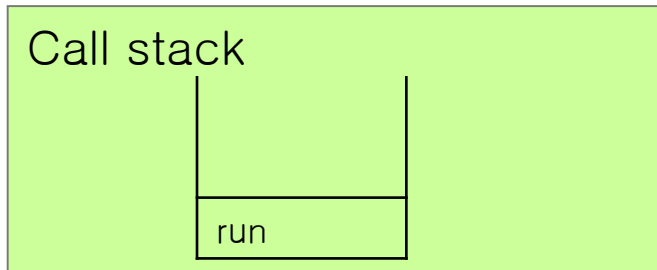


### 4. Call stack





# start()와 run()



main 메서드가 종료된 후의 호출스택

실행 중인 사용자 스레드가 하나도 없을 때 프로그램은 종료된다



# 싱글쓰레드와 멀티쓰레드

- 두 개의 작업을 하나의 쓰레드(th1)로 처리하는 경우와 두 개의 쓰레드(th1, th2)로 처리하는 경우를 가정해보자
  - 하나의 쓰레드로 두 작업을 처리하는 경우 - 한 작업을 마친 후에 다른 작업을 시작
  - 두 개의 쓰레드로 작업하는 경우 - 짧은 시간동안 **2개의 쓰레드가 번갈아가면서 작업을 수행**해서 동시에 두 작업이 처리되는 것과 같이 느끼게 함
- 하나의 쓰레드로 두개의 작업을 수행한 시간과 두개의 쓰레드로 두 개의 작업을 수행한 시간은 거의 같다
- 오히려 두 개의 쓰레드로 작업한 시간이 싱글쓰레드로 작업한 시간보다 더 걸리게 됨
  - <= 이유 : 쓰레드간의 작업전환에 시간이 걸리기 때문
  - 단순히 CPU만을 사용하는 계산작업이라면 오히려 멀티쓰레드보다 싱글쓰레드로 프로그래밍하는 것이 더 효율적
- 작업전환할 때는 현재 진행중인 작업의 상태(예:다음에 실행해야할 위치(프로그램 카운터) 등의 정보를 저장하고 읽어오는 시간이 소요됨

[illegible]

```
class ThreadEx4 {
    public static void main(String args[]) {
        long startTime = System.currentTimeMillis();
        for(int i=0; i < 300; i++) {
            System.out.print("-");
        }
        System.out.println();
        System.out.println(" [소요시간1: " +(System.currentTimeMillis()- startTime)+" ] \n");

        for(int i=0; i < 300; i++) {
            System.out.print("2");
        }
        System.out.println();
        System.out.println(" [소요시간2: "+(System.currentTimeMillis() - startTime) +"] ");
    }
}
```

- '-'를 출력하는 작업과 '2'를 출력하는 작업을 하나의 쓰레드가 연속적으로 처리하는 시간을 측정하는 예제

- 

1. **Identify the main components of the system.**

[illegible]

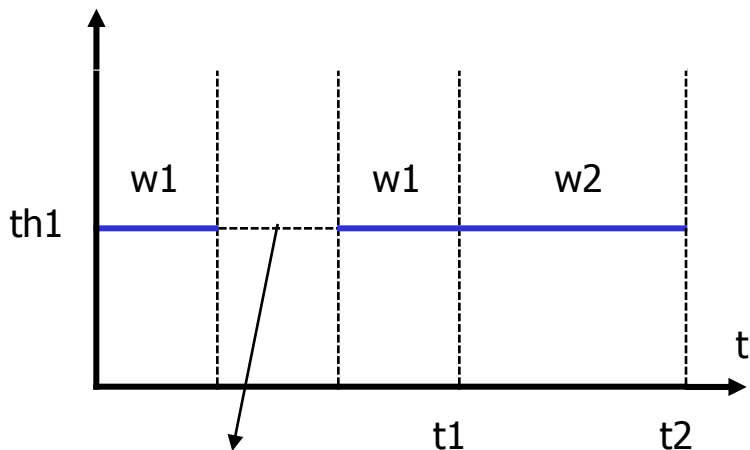
두 작업이 아주 짧은 시간동안 번갈아가면서 실행  
거의 동시에 작업이 완료되었음

# 싱글쓰레드 프로세스와 멀티쓰레드 프로세스의 비교

- CPU 이외의 자원을 사용하는 경우 - 싱글쓰레드 프로세스보다 멀티쓰레드 프로세스가 더 효율적
  - 예) 사용자로부터 데이터를 입력 받는 작업, 네트워크로 파일을 주고받는 작업, 프린터로 파일을 출력하는 작업과 같이 외부기기와의 입출력을 필요로 하는 경우

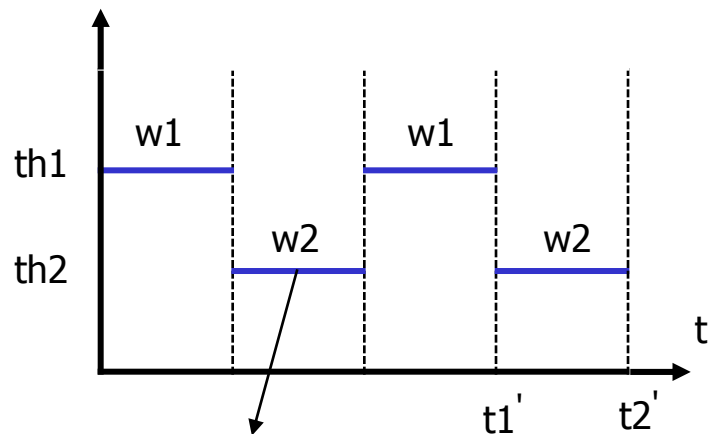
사용자로부터 입력 받는 작업(w1), 화면에 출력하는 작업(w2)

- 하나의 쓰레드로 두 개의 작업을 수행



사용자로부터 입력을 기다리는 구간. 아무 일도 하지 않는다

- 두개의 쓰레드로 두 개의 작업을 수행



사용자로부터 입력을 기다리는 구간. th2가 수행된다



## 싱글쓰레드 프로세스와 멀티쓰레드 프로세스의 비교

---

- 사용자로부터 입력 받는 작업(w1)과 화면에 출력 하는 작업(w2)을 하나의 쓰레드로 처리한다면 왼쪽 그래프처럼 사용자가 입력을 마칠 때까지 아무 일도 하지 못하고 기다리기만 해야 함
- 두 개의 쓰레드로 처리한다면 **사용자의 입력을 기다리는 동안 다른 쓰레드가 작업을 처리할 수 있기 때문에 보다 효율적인 CPU의 사용이 가능**
  - 작업을 더 빨리 마침

## 예제 4

```
import javax.swing.JOptionPane;
```

```
class ThreadEx6
```

```
{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        String input = JOptionPane.showInputDialog("아무 값이나 입력하세요");
```

```
        System.out.println("입력하신 값은 " + input + "입니다.");
```

```
        for(int i=10; i > 0; i--) {
```

```
            System.out.println(i);
```

```
            try {
```

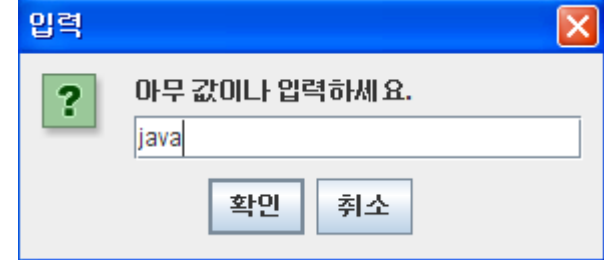
```
                Thread.sleep(1000); //지정된 시간동안 스레드를 일시정지시킴
```

```
            } catch(Exception e ) {}
```

```
        }
```

```
    }
```

```
}
```

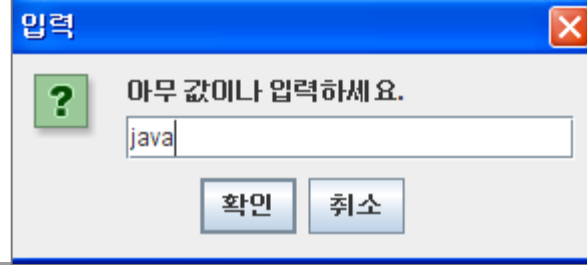


```
입력하신 값은 java입니다.
10
9
8
7
6
5
4
3
2
1
계속하려면 아무 키나 누르십시오...
```

하나의 스레드로 사용자의 입력을 받는 작업과 화면에 숫자를 출력하는 작업을 처리  
⇒ 사용자가 입력을 마치기 전까지는 화면에 숫자가 출력되지 않다가  
사용자가 입력을 마치고 나서야 화면에 숫자가 출력됨



## 예제5



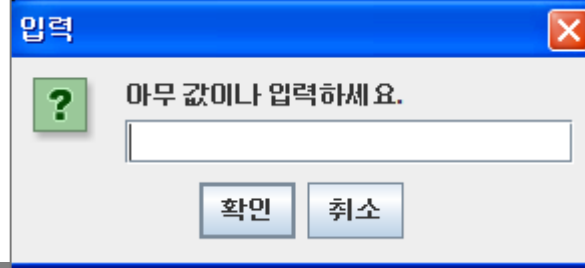
```
import javax.swing.JOptionPane;
```

```
class ThreadEx7 {  
    public static void main(String[] args){  
        MyThread th = new MyThread();  
        th.start();  
  
        String input = JOptionPane.showInputDialog("아무 값이나 입력하세요.");  
        System.out.println("입력하신 값은 " + input + "입니다.");  
    }  
}  
  
class MyThread extends Thread {  
    public void run() {  
        for(int i=10; i > 0; i--) {  
            System.out.println(i);  
            try {  
                sleep(1000);  
            } catch(Exception e) {}  
        }  
    }  
}
```

```
10  
9  
8  
입력하신 값은 java입니다.  
7  
6  
5  
4  
3  
2  
1  
계속하려면 아무 키나 누르십시오
```

사용자의 입력을 받는 부분과 화면에 숫자를 출력하는 부분을 두 개의 스레드로 나누어서 처리  
⇒사용자가 입력을 마치지 않았어도 화면에 숫자가 출력

## 예제6



```
main 쓰레드 종료!!
9
10 초안에 값을 입력해야 합니다.
8
7
6
5
4
3
2
1
0
10 초 동안 값이 입력되지 않아 종료합니다.
```

```
import javax.swing.JOptionPane;
```

```
class ThreadEx8 {
    static boolean inputCheck = false;

    public static void main(String[] args){
        MyThread1 th1 = new MyThread1();
        MyThread2 th2 = new MyThread2();
        th1.start();
        th2.start();
        System.out.println("main 쓰레드 종료!!");
    }
}

class MyThread1 extends Thread {
    public void run() {
        System.out.println("10초안에 값을 입력해야 합니다.");
        String input = JOptionPane.showInputDialog("아무 값이나 입력하세요.");
        ThreadEx8.inputCheck = true;
        System.out.println("입력값은 " + input + "입니다.");
    }
}
```

10초 동안 사용자가 입력하지 않으면 종료하는 예제



## 예제6

---

```
class MyThread2 extends Thread {  
    public void run() {  
        for(int i=9; i >= 0; i--) {  
            if(ThreadEx8.inputCheck) return;  
            System.out.println(i);  
  
            try {  
                sleep(1000);  
            } catch(InterruptedException e ) {}  
        }  
        System.out.println("10초 동안 값이 입력되지 않아 종료합니다.");  
        System.exit(0);  
    }  
}
```

# 쓰레드의 우선순위

- 쓰레드의 스케줄링(Scheduling)과 쓰레드의 우선순위
  - 둘 이상의 쓰레드가 생성될 수 있기 때문에 자바 가상 머신은 쓰레드의 실행을 스케줄링(컨트롤)해야 함
    - 1) 우선순위가 높은 쓰레드의 실행을 우선한다.
    - 2) 동일한 우선순위의 쓰레드가 둘 이상 존재할 때는 CPU의 할당시간을 분배해서 실행한다.
- 쓰레드의 우선순위
  - 가상머신에 의해서 우선적으로 실행되어야 하는 쓰레드의 순위
  - 가장 높은 우선순위 : 10
  - 가장 낮은 우선 순위 : 1

int getPriority() : 쓰레드의 우선순위를 반환

void setPriority(int newPriority) : 쓰레드의 우선순위를 지정한 값으로 변경

public static final int MAX\_PRIORITY=10 //최대 우선순위

static int MIN\_PRIORITY=1 //최소 우선순위

static int NORM\_PRIORITY=5 //보통 우선순위



# 쓰레드의 우선순위

---

- 쓰레드
  - 우선순위(priority) 속성(멤버변수)를 가지고 있음
  - 우선순위의 값에 따라 쓰레드가 얻는 실행시간이 달라짐
  - 쓰레드가 수행하는 작업의 중요도에 따라 쓰레드의 우선순위를 서로 다르게 지정하여 특정 쓰레드가 더 많은 작업시간을 갖도록 할 수 있음
- 예) 파일 전송기능이 있는 메신저
  - 파일다운로드를 처리하는 쓰레드보다 채팅내용을 전송하는 쓰레드의 우선순위가 더 높아야 사용자가 채팅하는 데 불편함이 없을 것임
  - 대신 파일다운로드 작업에 걸리는 시간은 더 길어질 것임
- 시각적인 부분이나 사용자에게 빠르게 반응해야 하는 작업을 하는 쓰레드의 우선순위는 다른 작업을 수행하는 쓰레드에 비해 높아야 함



# 쓰레드의 우선순위

---

- 쓰레드의 우선순위에 따른 할당되는 시간의 차이
  - 두 쓰레드의 우선순위가 같다면, 각 쓰레드에게 거의 같은 양의 실행시간이 주어짐
  - 우선 순위가 다르다면 우선순위가 높은 쓰레드에게 상대적으로 더 많은 양의 실행시간이 주어지고 결과적으로 더 빨리 작업이 완료될 수 있음
- 쓰레드의 우선순위는 쓰레드를 생성한 쓰레드로부터 상속 받는다
  - main 메서드를 수행하는 쓰레드는 우선순위가 5 이므로 main 메서드 내에서 생성하는 쓰레드의 우선순위는 자동적으로 5가 됨

# 예제

```
class ThreadEx9 {
    static long startTime = 0;
    public static void main(String args[]) {
        startTime = System.currentTimeMillis();
```

```
        MyThread1 th1 = new MyThread1();
        MyThread2 th2 = new MyThread2();
```

```
        th2.setPriority(7);//;
```

```
        System.out.println("Priority of th1(-) : " + th1.getPriority() );
```

```
        System.out.println("Priority of th2(2) : " + th2.getPriority() );
```

```
        th1.start();
```

```
        th2.start();
```

```
    }
}
```

```
class MyThread1 extends Thread {
    public void run() {
        for(int i=0; i < 10000; i++) {
            System.out.print("-");
        }
    }
}
```

```
        System.out.println(" [MyThread1 소요시간 (-): "
            +(System.currentTimeMillis()- ThreadEx9.startTime)+"] \n");
```

```
    }
}
```

```
Priority of th1(-) : 5
Priority of th2(2) : 7
```

```
계속하려면 아무 키나 누르십시오 . . .
```

동일한 우선순위 쓰레드들은 **CPU**의 할당시간을 골고루 나눠서 실행됨

- **th1**과 **th2** 모두 **main** 메서드에서 생성하였기 때문에 **main** 메서드를 실행하는 쓰레드의 우선순위인 **5**을 상속 받았음
- 그 다음에 **th2.setPriority(7)**로 **th2**의 우선순위를 **7**로 변경한 다음에 **start()**를 호출해서 쓰레드를 실행시킴
- **쓰레드를 실행하기 전에만 우선순위를 변경할 수 있음**
- 우선순위가 높은 **th2**의 실행시간이 **th1**에 비해 상당히 늘어난 것을 알 수 있음
- **th2**가 더 빨리 작업이 완료됨



## 예제

---

```
class MyThread2 extends Thread {  
    public void run() {  
        for(int i=0; i < 10000; i++) {  
            System.out.print("2");  
  
        }  
  
        System.out.println(" [MyThread2 소요시간 (2): "  
            +(System.currentTimeMillis()- ThreadEx9.startTime)+"]  \n");  
    }  
}
```



# 실습

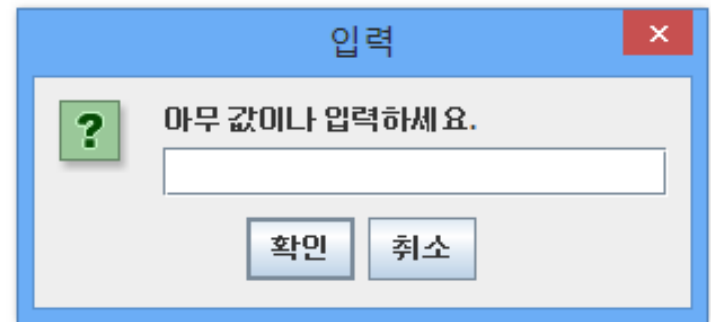
- Mythread1 쓰레드에서는 사용자로부터 숫자를 하나 입력 받아서 그 숫자까지의 합을 구하고,
  - 1) JOptionPane.showInputDialog() 이용
  - 2) Scanner 이용
- Mythread2 쓰레드에서는 3초마다 현재 시간을 출력하기 (5번 반복)

main 쓰레드 종료!

현재 시간 : 2015-03-22 03:05:46 오후

현재 시간 : 2015-03-22 03:05:49 오후

현재 시간 : 2015-03-22 03:05:52 오후





# 데몬 쓰레드(daemon thread)

- 데몬 쓰레드(daemon thread)
  - 다른 일반 쓰레드(데몬 쓰레드가 아닌 쓰레드)의 작업을 돕는 보조적인 역할을 수행하는 쓰레드
  - 일반 쓰레드가 모두 종료되면 데몬 쓰레드는 강제적으로 자동 종료됨
  - 데몬 쓰레드의 예) 가비지 컬렉터, 워드프로세서의 자동 저장, 화면 자동갱신 등

setDaemon(boolean on)은 반드시 **start()**를 호출하기 전에 실행되어야 함  
그렇지 않으면 `IllegalThreadStateException`이 발생함

# 예제

```
import java.util.*;
import java.text.*;
```

```
class JoinTest {
```

```
    public static void main(String[] args) {
        MyThread1 th1 = new MyThread1();
        th1.setDaemon(true); //일반 쓰레드가 모두 종료되면 th1은 강제로 자동 종료
        th1.start();
```

```
        MyThread2 r = new MyThread2();
        Thread th2 = new Thread(r);
        th2.start();
```

```
        for (int i=0;i<100 ;i++ )
        {
```

```
            System.out.print(i + " ");
```

```
            try {
```

```
                Thread.sleep(100);
```

```
            } catch (InterruptedException e ) {e.printStackTrace();}
```

```
        }
```

```
        System.out.println("[main 쓰레드 종료!]);
```

```
    }
```

```
}//class
```

C:\WINDOWS\system32\cmd.exe

0 숫자를 입력하세요

[현재 시간 : 2012-06-04 03:33:29 오후]

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

[현재 시간 : 2012-06-04 03:33:31 오후]

19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36

[현재 시간 : 2012-06-04 03:33:33 오후]

37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52

[0~100까지의 합 : 4950]

53 54 55

[현재 시간 : 2012-06-04 03:33:35 오후]

56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73

[현재 시간 : 2012-06-04 03:33:37 오후]

74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91

[현재 시간 : 2012-06-04 03:33:39 오후]

92 93 94 95 96 97 98 99 [main 쓰레드 종료!]

계속하려면 아무 키나 누르십시오 . . .

// 이 부분이 없으면 종료되지 않는다.



# 예제

---

```
class MyThread1 extends Thread {
    public void run() {
        while(true)
        {
            Date date = new Date();
            SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss a");
            String str = df.format(date);
            System.out.println("\n[현재 시간 : " + str + "]);

            try {
                sleep(2000);
            } catch (InterruptedException e ) {e.printStackTrace();}

        }
    }
}

class MyThread2 implements Runnable {
    public void run() {
        Scanner sc = new Scanner(System.in);
        System.out.println("숫자를 입력하세요");
        int cnt = sc.nextInt();
        int sum=0;
        for (int i=0;i<cnt ;i++ )
        {
            sum += i;
        }
        System.out.println("\n[0~" + cnt + "까지의 합 : " + sum + "]);
    }
}
```

계속하려면 아무 키나 누르십시오 . . .

37

## 예제2

```
class ThreadEx11 implements Runnable {
    static boolean autoSave = false;

    public static void main(String[] args) {
        Thread t = new Thread(new ThreadEx11());
        t.setDaemon(true);    // 이 부분이 없으면 종료되지 않는다.
        t.start();

        for(int i=1; i <= 20; i++) {
            try{
                Thread.sleep(1000);
            } catch(InterruptedException e) {}
            System.out.println(i);

            if(i==5)
                autoSave = true;
        }

        System.out.println("프로그램을 종료합니다.");
    }
}
```

```
1
2
3
4
5
6
7 작업파일이 자동저장되었습니다.
8
9
10
11 작업파일이 자동저장되었습니다.
12
13
14 작업파일이 자동저장되었습니다.
15
16
17
18 작업파일이 자동저장되었습니다.
19
20
프로그램을 종료합니다.
```



## 예제2

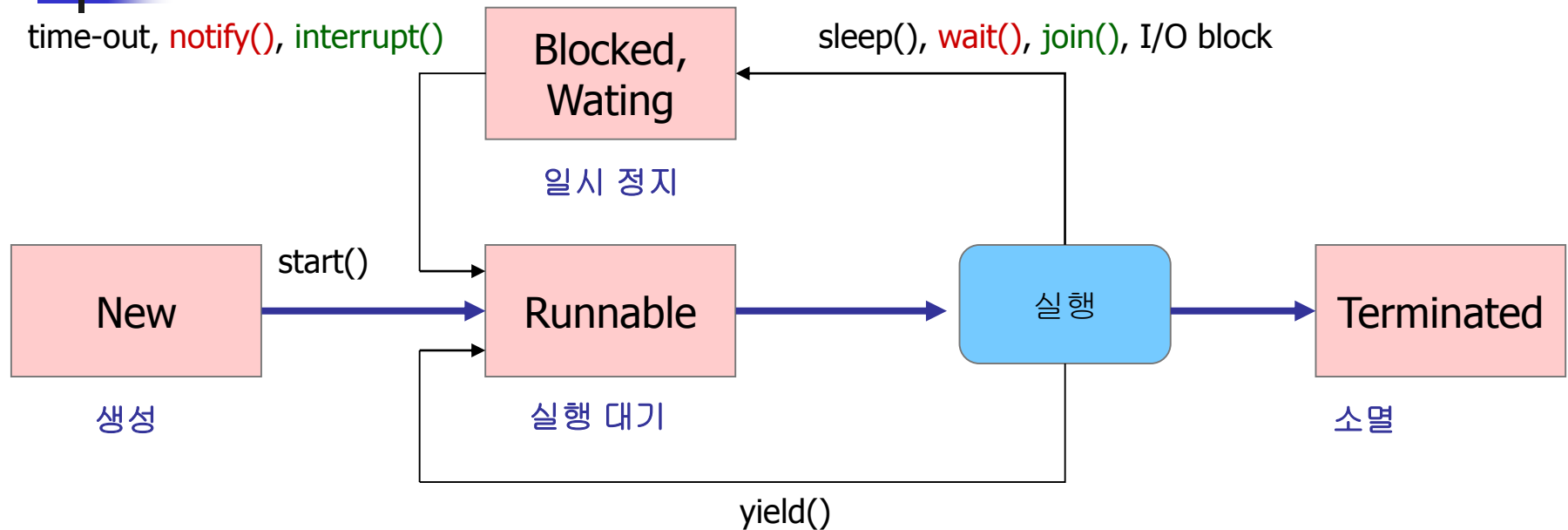
---

```
public void run() {
    while(true) {
        try {
            Thread.sleep(3 * 1000);           // 3초마다
        } catch (InterruptedException e) {}

        // autoSave의 값이 true이면 autoSave()를 호출한다.
        if(autoSave) {
            autoSaveFunc();
        }
    }
}

public void autoSaveFunc() {
    System.out.println("작업파일이 자동저장되었습니다.");
}
}
```

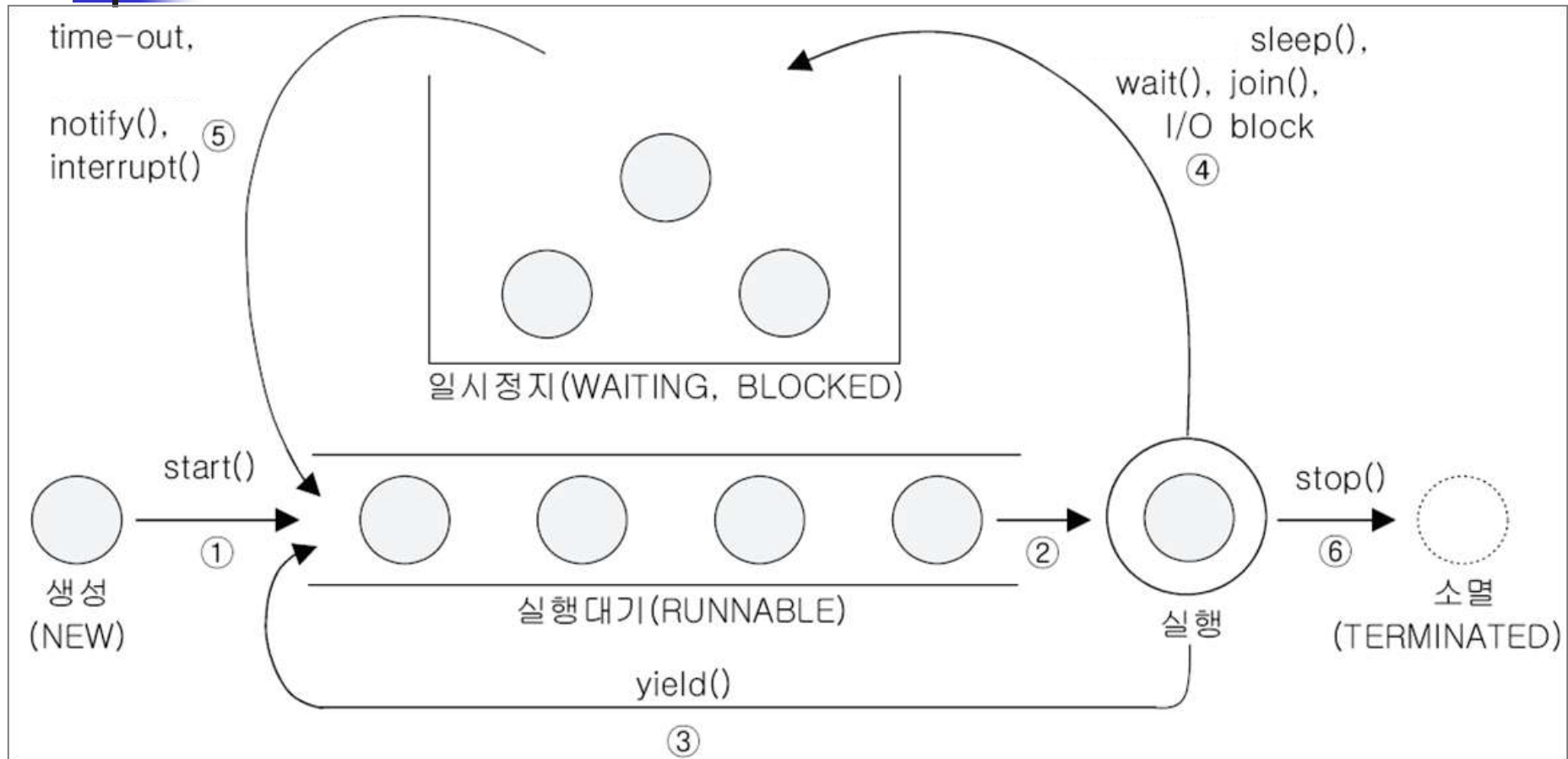
# 쓰레드의 라이프 사이클(Life Cycle)



상태	설명
NEW	쓰레드가 생성되고 아직 <code>start()</code> 가 호출되지 않은 상태
RUNNABLE	실행 중 또는 실행 가능한 상태
BLOCKED	동기화블럭에 의해서 일시정지된 상태(lock이 풀릴 때까지 기다리는 상태)
WAITING, TIMED_WAITING	쓰레드의 작업이 종료되지는 않았지만 실행가능하지 않은(unrunnable) 일시정지상태. TIMED_WAITING은 일시정지시간이 지정된 경우를 의미한다.
TERMINATED	쓰레드의 작업이 종료된 상태



# 쓰레드의 라이프 사이클(Life Cycle)





# 쓰레드의 상태

---

## ■ Runnable 상태

- 모든 실행의 준비를 마치고, 스케줄러에 의해서 선택되어 실행될 수 있기만을 기다리는 상태.

## ■ Blocked 상태

- 실행중인 쓰레드가 sleep, join 메서드를 호출하거나, CPU의 할당이 필요치 않는 입출력 연산을 하게 되면, CPU를 다른 쓰레드에게 양보하고, 본인은 Blocked 상태가 됨

## ■ Terminated 상태

- run 메서드의 실행이 완료되어서 run 메서드를 빠져 나오게 되면, 해당 쓰레드는 Terminated 상태가 됨
- 쓰레드의 실행을 위해서 할당 받았던 메모리를 비롯해서 각종 쓰레드 관련 정보가 완전히 사라지는 상태

interrupt() - InterruptedException 을 발생시켜서 sleep(), join(), wait()에 의해 일시정지상태인 스레드를 실행대기상태로 만든다.

## 스레드의 스케줄링과 관련된 메서드

생성자 / 메서드	설 명
void interrupt()	sleep()이나 join()에 의해 일시정지상태인 스레드를 실행대기상태로 만든다. 해당 스레드에서는 InterruptedException이 발생함으로써 일시정지상태를 벗어나게 된다.
void join() void join(long millis) void join(long millis, int nanos)	지정된 시간동안 스레드가 실행되도록 한다. 지정된 시간이 지나거나 작업이 종료되면 join()을 호출한 스레드로 다시 돌아와 실행을 계속한다.
static void sleep(long millis) static void sleep(long millis, int nanos)	지정된 시간(천분의 일초 단위)동안 스레드를 일시정지시킨다. 지정한 시간이 지나고 나면, 자동적으로 다시 실행대기상태가 된다.
static void yield()	실행 중에 다른 스레드에게 양보(yield)하고 실행대기상태가 된다.

sleep() - 항상 현재 실행중인 스레드에 대해 작동 => Thread.sleep(1000);

# 예제

```
class ThreadEx13 {
    static long startTime = 0;

    public static void main(String args[]) {
```

```
MyThread1 th1 = new MyThread1();
MyThread2 th2 = new MyThread2();

th1.start();
th2.start();
startTime = System.currentTimeMillis();
```

```
try {
    th1.join(); // th1의 작업이 끝날 때까지 기다린다.
    th2.join(); // th2의 작업이 끝날 때까지 기다린다.
} catch (InterruptedException e) {}
```

```
");    System.out.print(" [소요시간:" + (System.currentTimeMillis() - ThreadEx13.startTime)+"")
);
    System.out.print(" [main 스레드 종료!] ");
} // main
```

```
[소요시간:141] [main 쓰레드 종료!] 계속하려면 아무 키나 누르십시오 . . .
```

- join()을 사용하지 않으면 main 쓰레드는 바로 종료
- join() 사용해서 쓰레드 th1, th2의 작업을 마칠때까지 main 쓰레드가 기다리도록.



# 예제

---

```
class MyThread1 extends Thread {  
    public void run() {  
        for(int i=0; i < 3000; i++) {  
            System.out.print("-");  
        }  
    } // run()  
}
```

```
class MyThread2 extends Thread {  
    public void run() {  
        for(int i=0; i < 3000; i++) {  
            System.out.print("2");  
        }  
    } // run()  
}
```

# Thread-start, stop(interrupted)

```
public class LifeCycleTest extends Frame
    implements ActionListener, Runnable{
    int x=30, y=100;
    Button btStart, btStop;
    public LifeCycleTest(){
        setSize(500,300);
        setLayout(new BorderLayout());
        Panel p=new Panel();
        add(p,"South");
        p.add(btStart=new Button("시작"));
        p.add(btStop=new Button("중지"));

        btStart.addActionListener(this);
        btStop.addActionListener(this);
    }
    public void run(){
        System.out.println("width:"+this.getWidth());
        while(true){
            if(x>this.getWidth()){
                x=0;
            }
        }
    }
}
```

- 스레드에서 해야 할 작업
  - 1) x좌표값 증가시키기
  - 2) 다시 적용하여 그리기(repaint)



```

        x+=5; //x값 증가시키기
        repaint(); //다시 그리기
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            System.out.println(e);
            //중지 클릭하면, java.lang.InterruptedException: sleep interrupted
            break;
        }
    } //while
}

Thread tr;
public void actionPerformed(ActionEvent e){
    Object o=e.getSource();
    System.out.println("o="+o+", btStart="+btStart);
    if(o==btStart){
        tr=new Thread(this);
        //스레드 시작
        tr.start();
    }else if(o==btStop){
        //스레드 중지
        tr.interrupt();
    }
}
}

```

```

public void paint(Graphics g){
    g.setColor(Color.ORANGE);
    g.fillOval(x,y,70,70); //x, y, width, height
}

public static void main(String[] args) {
    LifeCycleTest f = new LifeCycleTest();
    f.setVisible(true);
}
}

```

# 쓰레드의 동기화

- 멀티쓰레드 프로세스의 경우 여러 쓰레드가 같은 프로세스 내의 자원을 공유해서 작업을 하기 때문에 서로의 작업에 영향을 주게 됨
- 쓰레드 A가 작업하던 도중에 다른 쓰레드B에게 제어권이 넘어갔을 때, 쓰레드A가 작업하던 공유데이터를 쓰레드B가 임의로 변경하였다면, 원래 의도했던 것과 다른 결과를 얻게 됨
- 쓰레드의 동기화(synchronized)
  - 한 번에 하나의 쓰레드만 객체에 접근할 수 있도록 객체에 락(lock)을 걸어서 데이터의 일관성을 유지하는 것

```
1. 특정한 객체에 lock을 걸고자 할 때
   synchronized(객체의 참조변수){
       ....
   }
2. 메서드에 lock을 걸고자 할 때
   public synchronized void func1(){
       ...
   }
```





# synchronized를 이용한 동기화

## ■ 동기화

- 한쪽에서 쓰는 동안 다른 쪽을 대기시키는 것
- 공유 자원을 액세스하는 문장을 `synchronized` 로 감싸면 동시에 두 스레드가 하나의 객체를 액세스하지 않음

- 해당 작업과 관련된 공유 데이터에 lock을 걸어서 먼저 작업 중이던 스레드가 작업을 완전히 마칠 때까지는 다른 스레드에게 제어권이 넘어가더라도 데이터가 변경되지 않도록 보호 => 스레드의 동기화를 가능하게 함

# 예제 1

```
class ThreadEx21 {  
    public static void main(String args[]) {  
        RunnableImpl r = new RunnableImpl();  
        Thread t1 = new Thread(r);  
        Thread t2 = new Thread(r);  
  
        t1.start();  
        t2.start();  
    }  
}
```

```
class RunnableImpl implements Runnable {  
    int insVar = 0; //공유
```

```
    public void run() {  
        int localVar = 0;  
        String name = Thread.currentThread().getName();  
  
        while(localVar < 3) {  
            //++localVar;  
            System.out.println(name + " Local Var:" + ++localVar);  
            System.out.println(name + "----- Instance Var:" + ++insVar);  
            System.out.println();  
        }  
    } // run()  
}
```

```
Thread-0 Local Var:1  
Thread-1 Local Var:1  
Thread-0----- Instance Var:1  
Thread-1----- Instance Var:2  
  
Thread-1 Local Var:2  
Thread-1----- Instance Var:3  
  
Thread-1 Local Var:3  
Thread-0 Local Var:2  
Thread-1----- Instance Var:4  
  
Thread-0----- Instance Var:5  
  
Thread-0 Local Var:3  
Thread-0----- Instance Var:6
```

하나의 객체(RunnableImpl인스턴스)를 두 개의 스레드가 공유 => 인스턴스변수(insVar)의 값은 두 스레드 모두 접근 가능



## 예제2

```
class ThreadEx22 {
    public static void main(String args[]) {
        Circle c = new Circle();
        MyThread t1 = new MyThread(c);
        MyThread t2 = new MyThread(c);

        t1.start();
        t2.start();
    }
}

class Circle {
    int insR = 0;
}

class MyThread extends Thread
{
    Circle c;
    MyThread(Circle c) {
        this.c = c;
    }
    public void run() {
        int localVar = 0;

        while(localVar < 3) {
            System.out.println(getName() + " Local Var:" + ++localVar);
            System.out.println(getName() + " -----Instance Var:" + ++c.insR);
            System.out.println();
        }
    }
} // run()
```

```
Thread-0 Local Var:1
Thread-1 Local Var:1
Thread-0 -----Instance Var:1
Thread-1 -----Instance Var:2

Thread-1 Local Var:2
Thread-0 Local Var:2
Thread-1 -----Instance Var:3
Thread-0 -----Instance Var:4

Thread-1 Local Var:3
Thread-0 Local Var:3
Thread-1 -----Instance Var:5
Thread-0 -----Instance Var:6
```

# 예제3

```
balance:700
balance:700
balance:500
balance:500
balance:200
balance:200
balance:200
balance:200
balance:200
balance:-100
balance:-300
```

```
class ThreadEx24 {
    public static void main(String args[]) {
        MyRunnable r = new MyRunnable();
        Thread t1 = new Thread(r);
        Thread t2 = new Thread(r);

        t1.start();
        t2.start();
    }
}
```

- 은행구좌에서 잔고를 확인하고 임의의 금액을 출금하는 예제
- 잔고가 음수  $\leq$  한 쓰레드가 if문의 조건식을 통과하고 출금하기 바로 직전에 다른 쓰레드가 끼어들어서 출금을 먼저 했기 때문

```
class Account {
    int balance = 1000;

    public void withdraw(int money){
        if(balance >= money) {
            try { Thread.sleep(1000);} catch(Exception e) {}
            balance -= money;
        }
    } // withdraw
}
```



## 예제3

---

```
class MyRunnable implements Runnable {
    Account acc = new Account();

    public void run() {
        while(acc.balance > 0) {
            // 100, 200, 300중의 한 값을 임의로 선택해서 출금(withdraw)
            int money = (int)(Math.random() * 3 + 1) * 100;
            acc.withdraw(money);
            System.out.println(Thread.currentThread().getName()
                               + ": balance="+acc.balance + ", money="+money);
        }
    } // run()
} //
```

## 예제3

- 한 스레드에 의해서 먼저 **withdraw()**가 호출되면, 종료될 때까지 다른 스레드가 **withdraw()**를 호출하더라도 대기상태에 머물게 됨
- **withdraw()**는 한 순간에 단 하나의 스레드만 사용할 수 있다.

### ■ 동기화

```
public synchronized void withdraw(int money){
    if(balance >= money) {
        try { Thread.sleep(1000);} catch(Exception e) {}
        balance -= money;
    }//if
} // withdraw
```

```
public void withdraw(int money){
    synchronized(this){
        if(balance >= money) {
            try { Thread.sleep(1000);} catch(Exception e) {}
            balance -= money;
        }//if
    }
} // withdraw
```



## wait()와 notify()에 의한 실행순서의 동기화

### ■ wait()와 notify()

- Object에 정의된 메서드 => 모든 객체에서 호출 가능
- 동기화 블록 내에서만 사용할 수 있다

- wait() – 객체의 lock을 풀고 해당 객체의 스레드를 waiting pool에 넣는다.
- notify() – waiting pool에서 대기중인 스레드 중의 하나를 깨운다. (통보)
- notifyAll() – waiting pool에서 대기중인 모든 스레드를 깨운다.

- 한 스레드가 객체에 lock을 걸고 오래 기다리는 대신 **wait()**을 호출해서 다른 스레드에게 제어권을 넘겨주고 대기 상태로 기다리다가 다른 스레드에 의해서 **notify()**가 호출되면 다시 실행상태가 되도록 하는 것 => 보다 효율적인 동기화를 가능하게 함

# 예제

```
class ThreadEx27 {
    public static void main(String args[]) {
        MyRunnable r = new MyRunnable();
        Thread t1 = new Thread(r);
        Thread t2 = new Thread(r);

        t1.start();
        t2.start();
        System.out.println("main 스레드 종료!");
    }
}

class Account {
    int balance = 1000;

    public synchronized void withdraw(int money){
        while(balance < money) {
            try {
                wait();
            } catch (InterruptedException e) {}
        } //while
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {e.printStackTrace(); }

        balance -= money;
    } // withdraw
}
```

- 출금을 위해 **withdraw()**가 호출되었을 때 잔고가 부족하면 **wait()**를 호출해서 스레드가 객체의 **lock**을 풀고 그 객체의 **waiting pool**에 들어가면서 제어권을 다른 스레드에게 양보하게 됨
- 다른 스레드에 의해서 **deposit()** 메서드가 호출되어 잔고가 증가하면서 **notify()**를 호출하면 객체의 **waiting pool**에서 기다리고 있던 스레드를 깨우게 됨





# 예제

---

```
public synchronized void deposit(int money){
    balance += money;
    notify();
} // deposit
}

class MyRunnable implements Runnable {
    Account acc = new Account();

    public void run() {
        while(acc.balance > 0) {
            int money = (int)(Math.random() * 3 + 1) * 100;
            acc.withdraw(money);
            System.out.println("balance:"+acc.balance);
        }

        acc.deposit(2000);
    } // run()
}
```

## 예제2

```
class Newspaper{
    String todayNews;
    boolean isTodayNews=false;

    public void setTodayNews(String news) {
        todayNews=news;
        isTodayNews=true;

        synchronized(this)
        {
            notifyAll();
        }
    }
    public String getTodayNews(){
        if(isTodayNews==false){
            try{
                synchronized(this)
                {
                    wait();
                }
            }
            catch(InterruptedException e){
                e.printStackTrace();
            }
        }
        return todayNews;
    }
}
```



## 예제2

---

```
class NewsWriter extends Thread
{
    Newspaper paper;

    public NewsWriter(Newspaper paper)
    {
        this.paper=paper;
    }
    public void run()
    {
        paper.setTodayNews("전국 무더위 계속..");
    }
}

class NewsReader extends Thread
{
    Newspaper paper;

    public NewsReader(Newspaper paper)
    {
        this.paper=paper;
    }
    public void run()
    {
        System.out.println("오늘의 뉴스: "+paper.getTodayNews());
    }
}
```



## 예제2

---

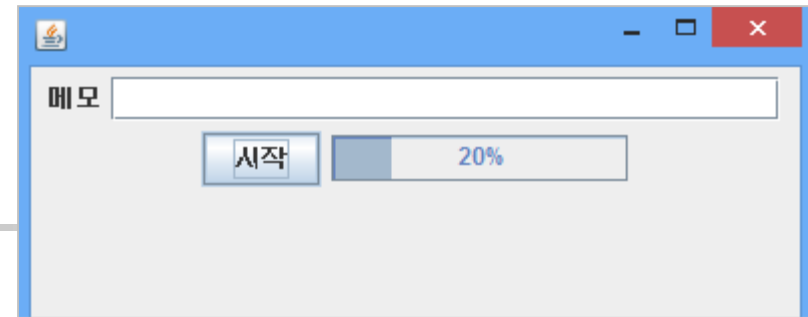
```
class SyncNewsPaper
{
    public static void main(String[] args)
    {
        NewsPaper paper=new NewsPaper();
        NewsReader reader1=new NewsReader(paper);
        NewsReader reader2=new NewsReader(paper);
        NewsWriter writer=new NewsWriter(paper);

        try
        {
            reader1.start();
            reader2.start();

            Thread.sleep(1000);
            writer.start();

            reader1.join();
            reader2.join();
            writer.join();
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
}
```

# 진행바



```
public class ProgressBarTest extends JFrame
    implements ActionListener,Runnable{
    JButton bt;
    JProgressBar bar;
    JTextField tfMemo;
    JLabel lb;
    Thread th; //버튼 클릭시 작동할 스레드
    public ProgressBarTest(){
        this.setLayout(new FlowLayout());

        lb=new JLabel("메모");
        tfMemo=new JTextField(30);

        add(lb);
        add(tfMemo);
        add(bt = new JButton("시작"));
        bar = new JProgressBar();
        bar.setStringPainted(true); //진행바에 퍼센티지가 표시됨
        add(bar);

        bt.addActionListener(this);
```

**JProgressBar-상태바, 진행바**

- 현재 작업량의 상태 표시를 시각화해서 보여주는 컴포넌트

```
new JProgressBar(JProgressBar.HORIZONTAL,0,100);
//최소값, 최대값
```

```

        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(400,300);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e){
        th = new Thread(this);
        th.start(); //runnable로 진입시킴
    }
    public void run() {
        synchronized(bar){
            System.out.println("run(), bar.getMinimum()="+bar.getMinimum()
                                +"bar.getMaximum()="+bar.getMaximum());
            for(int i=(bar.getMinimum());i<=bar.getMaximum();i+=5){
                System.out.println("i="+i);
                try {
                    Thread.sleep(200);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                bar.setValue(i);
            }//for
        }
    }
    public static void main(String[] args) {
        new ProgressBarTest();
    }
}

```

# 실습1

- main 쓰레드에서는 사용자로부터 주민번호를 입력받아서 생일과 성별을 출력
- 예) 920125-1112222
  - 생일: 1992-01-25
  - 성별 : 남
- MyThread 에서는 특정 사이트로부터 파일을 다운로드 중이라는 내용을 출력
  - 멤버변수 : String site
  - 생성자에서 site값을 매개변수로 받아와서 초기화
  - run() 메서드
    - for문에서 0~100까지, 10씩 증가하면서 다운로드중이라는 내용 출력
      - 1초간 지연시간 주기

```

http://www.nate.com에서 0% 다운로드 중...
http://www.nate.com에서 10% 다운로드 중...
http://www.nate.com에서 20% 다운로드 중...
http://www.nate.com에서 30% 다운로드 중...
http://www.nate.com에서 40% 다운로드 중...
http://www.nate.com에서 50% 다운로드 중...
http://www.nate.com에서 60% 다운로드 중...
http://www.nate.com에서 70% 다운로드 중...
주민번호 : 920125-1112222
생일 : 1992-01-25
성별 : 남
http://www.nate.com에서 80% 다운로드 중...
http://www.nate.com에서 90% 다운로드 중...
http://www.nate.com에서 100% 다운로드 중...
계속하려면 아무 키나 누르십시오 . . .
  
```

```

class MyThread1 extends Thread
{
    String site;

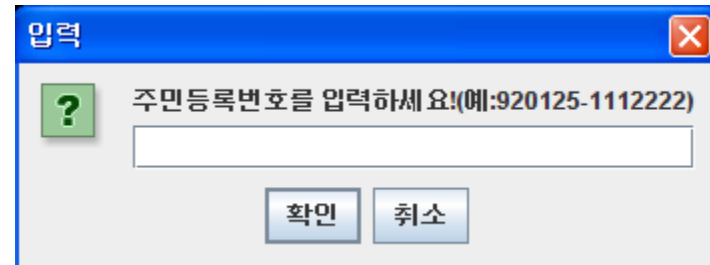
    public MyThread1(String site) {
        this.site=site;
    }

    public void run()
    { }
}
  
```

## 실습2

- 실습1을 다음과 같이 수정하기
  - MyThread 의 실행이 종료된 후 main 스레드가 실행되도록

```
http://www.nate.com에서 0% 다운로드 중...
http://www.nate.com에서 10% 다운로드 중...
http://www.nate.com에서 20% 다운로드 중...
http://www.nate.com에서 30% 다운로드 중...
http://www.nate.com에서 40% 다운로드 중...
http://www.nate.com에서 50% 다운로드 중...
http://www.nate.com에서 60% 다운로드 중...
http://www.nate.com에서 70% 다운로드 중...
http://www.nate.com에서 80% 다운로드 중...
http://www.nate.com에서 90% 다운로드 중...
http://www.nate.com에서 100% 다운로드 중...
주민번호 : 061027-4445555
생일 : 2006-10-27
성별 : 여
계속하려면 아무 키나 누르십시오 . . .
```



입력

? 주민등록번호를 입력하세요!(예:920125-1112222)

확인 취소



```
public class MyThread extends Thread{
```

```
    //bar마다 다른 값들
```

```
    int interval; //sleep interval
```

```
    int n; //bar 증가치
```

```
    JProgressBar bar;
```

```
    public MyThread(JProgressBar bar, int interval, int n){
```

```
        this.bar = bar;
```

```
        this.interval=interval;
```

```
        this.n=n;
```

```
    }
```

```
    public void run(){
```

```
        synchronized(bar){
```

```
            for(int i=bar.getMinimum();i<=bar.getMaximum();i+=n){
```

```
                try {
```

```
                    this.sleep(interval);
```

```
                    System.out.println("i="+i);
```

```
                    bar.setValue(i); //프로그레스바의 value가 각각 증가됨
```

```
                } catch (InterruptedException e) {
```

```
                    e.printStackTrace();
```

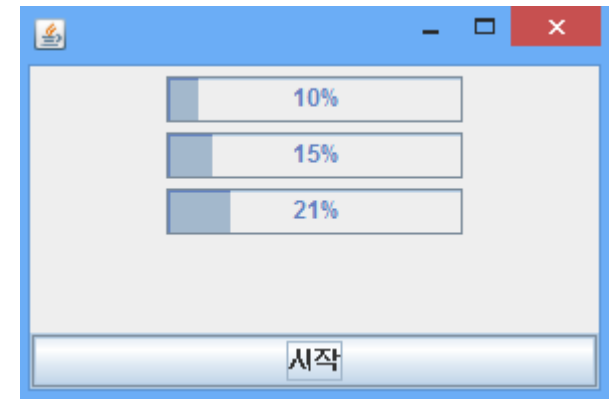
```
                }
```

```
            }//for
```

```
        }
```

```
    }
```

```
}
```



```

public class MultiProgress extends JFrame implements ActionListener{
    JButton bt;
    JProgressBar bar1,bar2,bar3;
    JPanel p;

    MyThread thread1;
    MyThread thread2;
    MyThread thread3;
    public MultiProgress(){
        bt = new JButton("시작");
        bar1 = new JProgressBar();
        bar2 = new JProgressBar();
        bar3 =new JProgressBar();
        bar1.setStringPainted(true);//진행바에 퍼센티지가 표시됨
        bar2.setStringPainted(true);
        bar3.setStringPainted(true);

        p= new JPanel();

        add(bt, BorderLayout.SOUTH);
        bt.addActionListener(this);

        p.add(bar1);
        p.add(bar2);
        p.add(bar3);
        add(p, BorderLayout.CENTER);
    }
}

```

```

        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(300,200);
        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        thread1 = new MyThread(bar1, 1000, 10); //JProgressBar, sleep interval, bar증가치
        thread2 = new MyThread(bar2, 500, 5);
        thread3 = new MyThread(bar3, 100, 1);

        thread1.start();
        thread2.start();
        thread3.start();
    }

    public static void main(String[] args) {
        new MultiProgress();
    }
}

```

- 프로그램의 첫번째 윈도우가 화면 상에 나타날 때 **AWT**에서는 특수한 쓰레드를 생성한다.
- 그리고 이 쓰레드를 사용해서 운영체제의 이벤트를 처리한다.

**main()** 이 실행되는 메인 쓰레드와, 운영체제로부터 들어오는 이벤트를 처리하고 등록된 리스너에 해당 이벤트에 대해 통지를 해주는 **AWT** 쓰레드가 기본적으로 생성됨