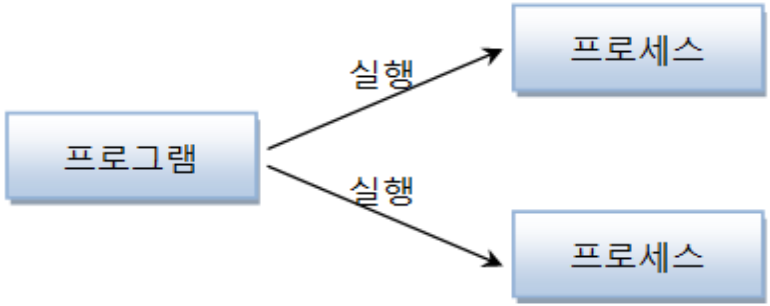
12장. 멀티 스레드

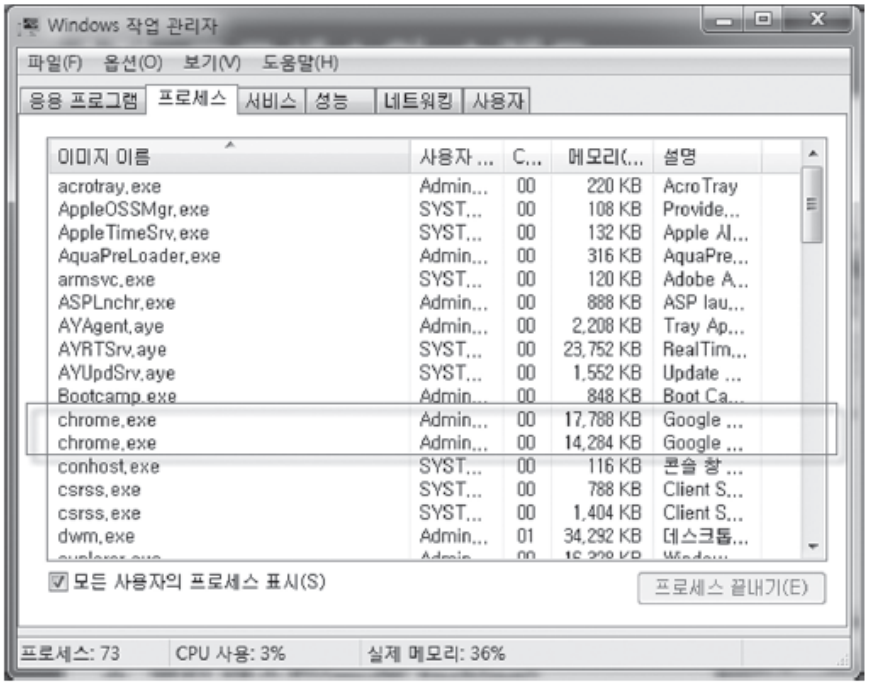
Contents

1절. 프로세스와 스레드

❖ 프로세스(process)

- 실행 중인 하나의 프로그램(애플리케이션)
- 사용자가 애플리케이션을 실행하면 운영체제(O/S)로부터 실행에 필요한 메모리를 할당받아 애플리케이션의 코드를 실행하는 것
- 하나의 프로그램이 다중 프로세스 만들기도 함





1절. 프로세스와 스레드

❖ 멀티 태스킹(multi tasking)

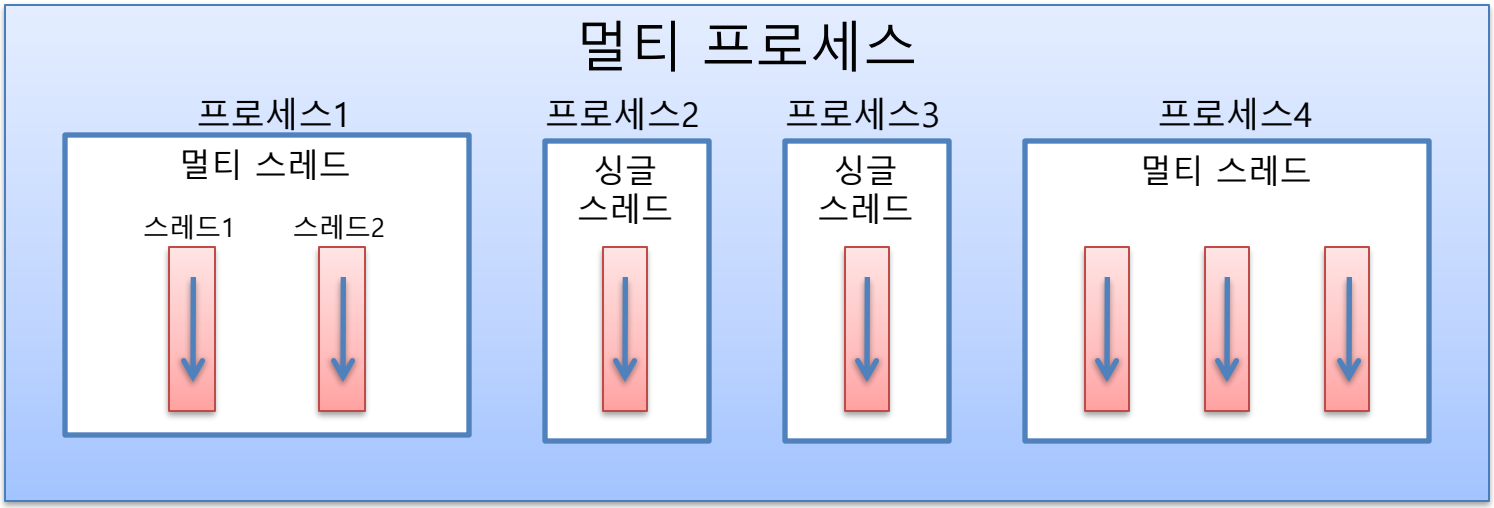
- 두 가지 이상의 작업을 동시에 처리하는 것

- 멀티 프로세스
  - 독립적으로 프로그램들을 실행하고 여러 가지 작업 처리

- 멀티 스레드

  스레드(thread) : 한 가지 작업을 실행하기 위해 순차적으로 실행할 코드

  - 한 개의 프로그램을 실행하고 내부적으로 여러 가지 작업 처리



1절. 프로세스와 스레드

❖ 멀티 프로세스

멀티프로세스 : 애플리케이션 단위의 멀티 태스킹

- 운영체제에서 할당받은 자신의 메모리를 가지고 실행하기 때문에 서로 독립적이다
- 하나의 프로세스에서 오류가 발생해도 다른 프로세스에게 영향을 미치지 않는다
- 워드와 엑셀을 동시에 사용하던 도중, 워드에 오류가 발생하더라도 엑셀은 여전히 사용이 가능하다

❖ 멀티 스레드

멀티스레드 : 한 개의 애플리케이션 내부에서의 멀티 태스킹

- 하나의 프로세스 내에서 여러가지 작업을 처리
- 메신저의 경우 파일을 전송하는 스레드에서 예외가 발생되면 메신저 프로세스 자체가 종료되기 때문에 채팅 스레드도 같이 종료된다
- 멀티스레드에서는 예외 처리에 만전을 기해야 한다
- 활용
  - 대용량 데이터의 처리 시간을 줄이기 위해 데이터를 분할해서 병렬처리 할 때
  - UI를 가지고 있는 애플리케이션에서 네트워크 통신을 할 때
  - 다수 클라이언트의 요청을 처리하는 서버를 개발할 때

1절. 프로세스와 스레드

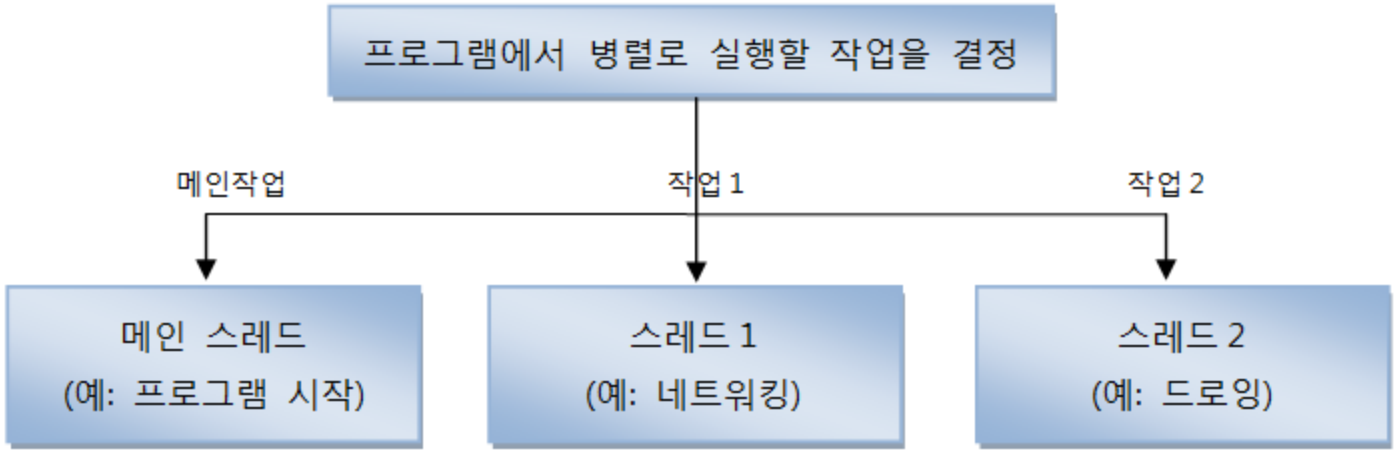
❖ 메인(main) 스레드

- 모든 자바 프로그램은 메인 스레드가 main() 메소드 실행하며 시작
- main() 메소드의 첫 코드부터 아래로 순차적으로 실행
- 실행 종료 조건
  - 마지막 코드 실행
  - return 문을 만나면
- main 스레드는 작업 스레드들을 만들어 병렬로 코드들 실행
  - 멀티 스레드 생성해 멀티 태스킹 수행
- 프로세스의 종료
  - 싱글 스레드: 메인 스레드가 종료하면 프로세스도 종료
  - 멀티 스레드: 실행 중인 스레드가 하나라도 있다면, 프로세스 미종료

2절. 작업 스레드 생성과 실행

❖ 멀티 스레드로 실행하는 어플리케이션 개발

- 몇 개의 작업을 병렬로 실행할지 결정하는 것이 선행되어야



2절. 작업 스레드 생성과 실행

❖ 작업 스레드 생성 방법

- Thread 클래스로부터 직접 생성
  - Runnable 인터페이스를 매개값으로 갖는 Thread 생성자 호출

```
Runnable beepTask = new BeepTask();
Thread   thread   = new Thread(beepTask);
thread.start();
```

- Thread 하위 클래스로부터 생성
  - Thread 클래스 상속 후 run 메소드 재정의 해 스레드가 실행할 코드 작성

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

2절. 작업 스레드 생성과 실행

❖ 두 개의 문장을 출력하는 예제

```
public class BeepPrintEx {
    public static void main(String[] args) throws InterruptedException {

        for (int i = 0; i < 5; i++) {
            System.out.println("삐~");

        }

        for (int i = 0; i < 5; i++) {
            System.out.println("***");

        }
    }
}
```

2절. 작업 스레드 생성과 실행

❖ 두 개의 문장을 출력하는 예제

```
public class BeepPrintEx {
    public static void main(String[] args) throws InterruptedException {

        for (int i = 0; i < 5; i++) {
            System.out.println("삐~");
            Thread.sleep(500);
        }

        for (int i = 0; i < 5; i++) {
            System.out.println("***");
            Thread.sleep(500);
        }
    }
}
```

2절. 작업 스레드 생성과 실행

❖ 두 개의 문장을 출력하는 예제

```
public class BeepPrintEx {
    public static void main(String[] args) throws InterruptedExc

        for (int i = 0; i < 5; i++) {
            System.out.println("삑~");
            Thread.sleep(500);
        }

        for (int i = 0; i < 5; i++) {
            System.out.println("***");
            Thread.sleep(500);
        }
    }
}
```

```
삑~
삑~
삑~
삑~
삑~
***
***
***
***
***
```

2절. 작업 스레드 생성과 실행

❖ Thread 클래스로부터 직접 생성

- Runnable을 매개값으로 갖는 생성자 호출

```
public class BeepTask implements Runnable {

    @Override
    public void run() {
        for (int i=0; i<5; i++) {
            System.out.println("삑~");
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) { }
        }
    }
}
```

```
public class BeepTaskEx {
    public static void main(String[] args) throws InterruptedException {
        Runnable beepTask = new BeepTask();
        Thread   thread   = new Thread(beepTask);
        thread.start();

        for (int i=0; i<5; i++) {
            System.out.println("*****");
            Thread.sleep(500);
        }
    }
}
```

2절. 작업 스레드 생성과 실행

❖ Thread 클래스로부터 직접 생성

- Runnable을 매개값으로 갖는 생성자 호출

```
public class BeepTask implements Runnable {

    @Override
    public void run() {
        for (int i=0; i<5; i++) {
            System.out.println("삑~");
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) { }
        }
    }
}
```

```
public class BeepTaskEx {
    public static void main(String[] args) throws InterruptedException {
        Runnable beepTask = new BeepTask();
        Thread   thread   = new Thread(beepTask);
        thread.start();

        for (int i=0; i<5; i++) {
            System.out.println("*****");
            Thread.sleep(500);
        }
    }
}
```

```
*****
삑~
*****
삑~
*****
삑~
*****
삑~
삑~
*****
```

2절. 작업 스레드 생성과 실행

❖ Thread 클래스로부터 직접 생성

- Runnable을 매개값으로 갖는 생성자 호출 (익명 객체 이용)

```
public class BeepTaskEx2 {
    public static void main(String[] args) throws InterruptedException {
        Thread thread = new Thread(new Runnable() {

            @Override
            public void run() {
                for (int i=0; i<5; i++) {
                    System.out.println("삡~");
                    try {
                        Thread.sleep(500);
                    } catch (InterruptedException e) { }
                }
            }
        });

        thread.start();

        for (int i=0; i<5; i++) {
            System.out.println("***");
            Thread.sleep(500);
        }
    }
}
```

2절. 작업 스레드 생성과 실행

❖ Thread 하위 클래스로부터 생성

- Thread 클래스 상속 후 run 메소드 재정의 해 스레드가 실행할 코드 작성

```
public class BeepThread extends Thread {

    @Override
    public void run() {
        for (int i=0; i<5; i++) {
            System.out.println("삡~");
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) { }
        }
    }
}
```

```
public class BeepThreadEx {
    public static void main(String[] args) throws InterruptedException {
        Thread thread = new BeepThread();
        thread.start();

        for (int i=0; i<5; i++) {
            System.out.println("*****");
            Thread.sleep(500);
        }
    }
}
```

2절. 작업 스레드 생성과 실행

❖ 스레드의 이름

- 메인 스레드 이름: main
- 작업 스레드 이름 (자동 설정) : Thread-n

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

- 작업 스레드 이름 변경

```
thread.setName("스레드  이름");
```

- 코드 실행하는 현재 스레드 객체의 참조 얻기

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

2절. 작업 스레드 생성과 실행

❖ 스레드의 이름 예제

```
public class ThreadA extends Thread {
    public ThreadA() {
        setName("ThreadA");
    }

    @Override
    public void run() {
        for (int i=0; i<5; i++)
            System.out.println(getName() + " 작동 중");
    }
}
```

```
public class ThreadB extends Thread {
    public ThreadB() {
        setName("ThreadB");
    }

    @Override
    public void run() {
        for (int i=0; i<5; i++)
            System.out.println(getName() + " 작동 중");
    }
}
```

2절. 작업 스레드 생성과 실행

❖ 스레드의 이름 예제

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

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

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

2절. 작업 스레드 생성과 실행

❖ 스레드의 이름 예제

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

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

        ThreadB threadB = new Thread
        threadB.start();
        System.out.println("작업 스레드 
    }
}
```

```
프로그램 시작 스레드 이름 : Thread[main,5,main]
작업 스레드 이름 - ThreadA
ThreadA 작동 중...
ThreadA 작동 중...
ThreadA 작동 중...
ThreadA 작동 중...
ThreadA 작동 중...
작업 스레드 이름 - ThreadB
ThreadB 작동 중...
ThreadB 작동 중...
ThreadB 작동 중...
ThreadB 작동 중...
ThreadB 작동 중...
```

2절. 작업 스레드 생성과 실행

❖ 멀티 스레드 예제

```
public class MyThread extends Thread {
    private int x;

    public MyThread(int x) {
        this.x = x;
    }

    @Override
    public void run() {
        System.out.println(x + "번째 스레드입니다.");
    }
}
```

```
public class MyThreadEx {
    public static void main(String[] args) {
        for (int i=0; i<100; i++) {
            MyThread thread = new MyThread(i+1);
            thread.start();
        }
    }
}
```

2절. 작업 스레드 생성과 실행

❖ 멀티 스레드 예제

```
public class MyThread extends Thread {
    private int x;

    public MyThread(int x) {
        this.x = x;
    }

    @Override
    public void run() {
        System.out.println(x + "번째 스레드입니다.");
    }
}
```

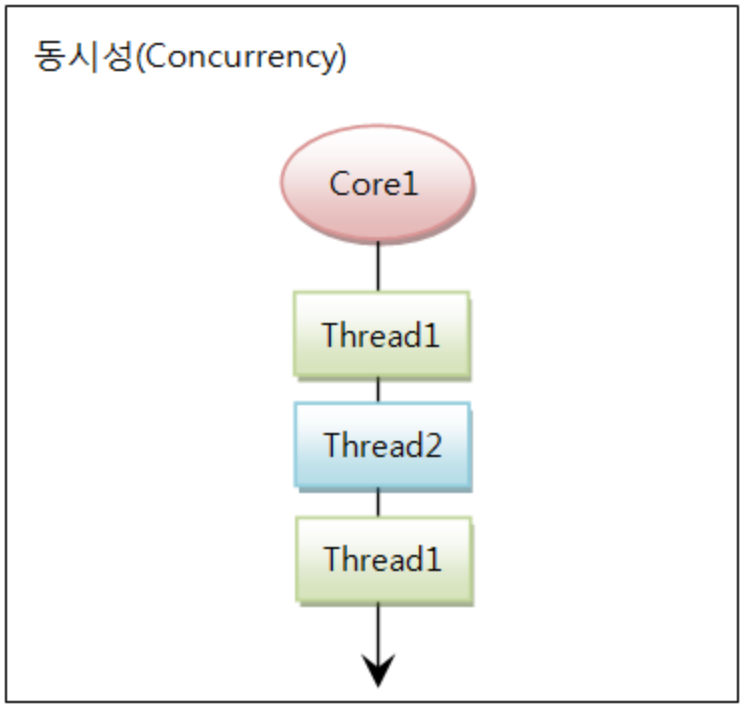
```
public class MyThreadEx {
    public static void main(String[] args) {
        for (int i=0; i<100; i++) {
            MyThread thread = new MyThread(i+1);
            thread.start();
        }
    }
}
```

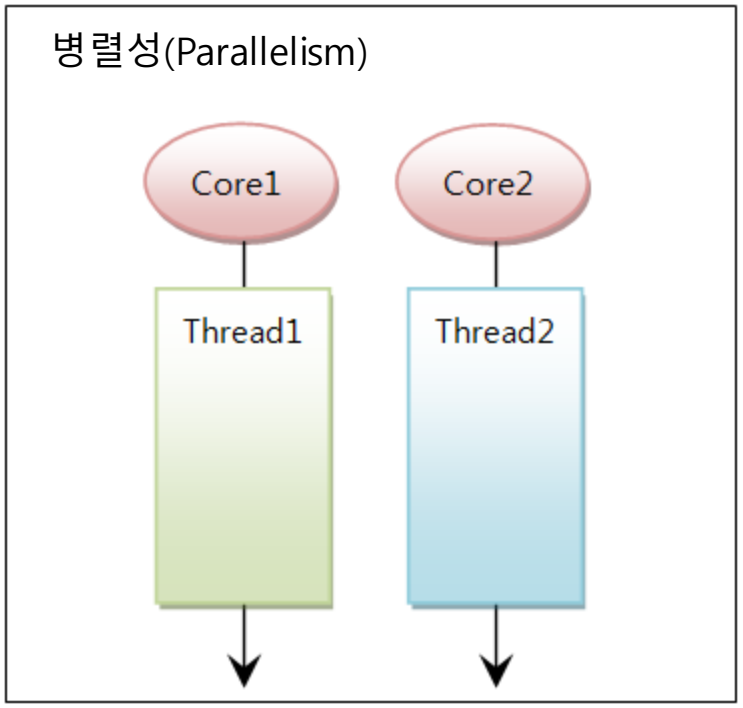
```
0번째 스레드입니다.
1번째 스레드입니다.
2번째 스레드입니다.
3번째 스레드입니다.
4번째 스레드입니다.
5번째 스레드입니다.
6번째 스레드입니다.
8번째 스레드입니다.
9번째 스레드입니다.
7번째 스레드입니다.
10번째 스레드입니다.
11번째 스레드입니다.
12번째 스레드입니다.
13번째 스레드입니다.
14번째 스레드입니다.
16번째 스레드입니다.
```

3절. 스레드 우선 순위

❖ 동시성과 병렬성

- 동시성(Concurrency)
  - 멀티 작업 위해 하나의 코어에서 멀티 스레드가 번갈아 가며 실행하는 성질
- 병렬성(Parallelism)
  - 멀티 작업을 위해 멀티 코어에서 개별 스레드를 동시에 실행하는 성질

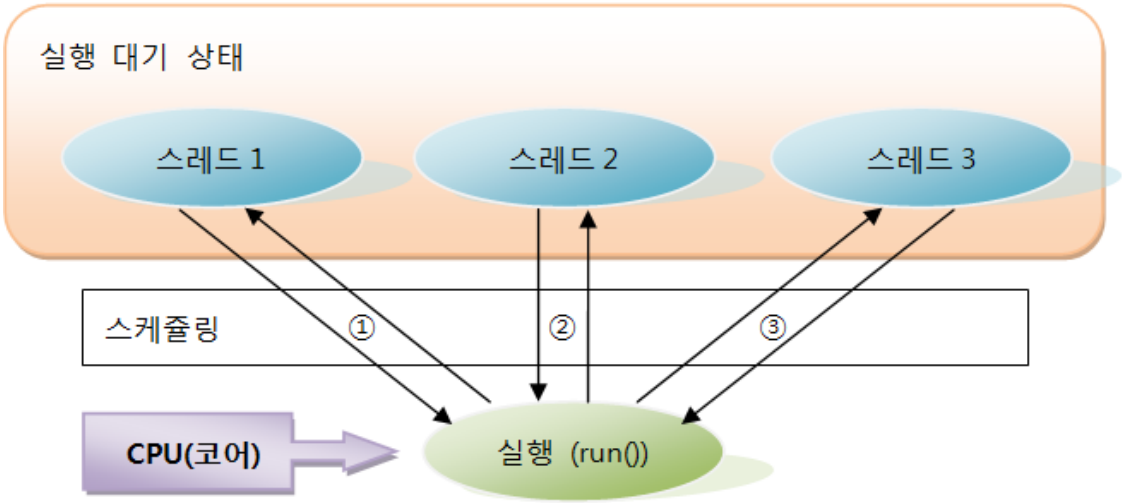




3절. 스레드 우선 순위

❖ 스레드 스케줄링

- 스레드의 개수가 코어의 수보다 많을 경우
  - 스레드를 어떤 순서로 동시성으로 실행할 것인가 결정→ 스레드 스케줄링
  - 스케줄링 의해 스레드들은 번갈아 가며 run() 메소드를 조금씩 실행



3절. 스레드 우선 순위

❖ 자바의 스레드 스케줄링

- 우선 순위(Priority) 방식과 순환 할당(Round-Robin) 방식 사용

- 우선 순위 방식 (코드로 제어 가능)
  - 우선 순위가 높은 스레드가 실행 상태를 더 많이 가지도록 스케줄링
  - 1~10까지 값을 가질 수 있으며 기본은 5

- 순환 할당 방식 (코드로 제어할 수 없음)
  - 시간 할당량(Time Slice) 정해서 하나의 스레드를 정해진 시간만큼 실행

3절. 스레드 우선 순위

❖ 우선 순위 방식 (코드로 제어 가능)

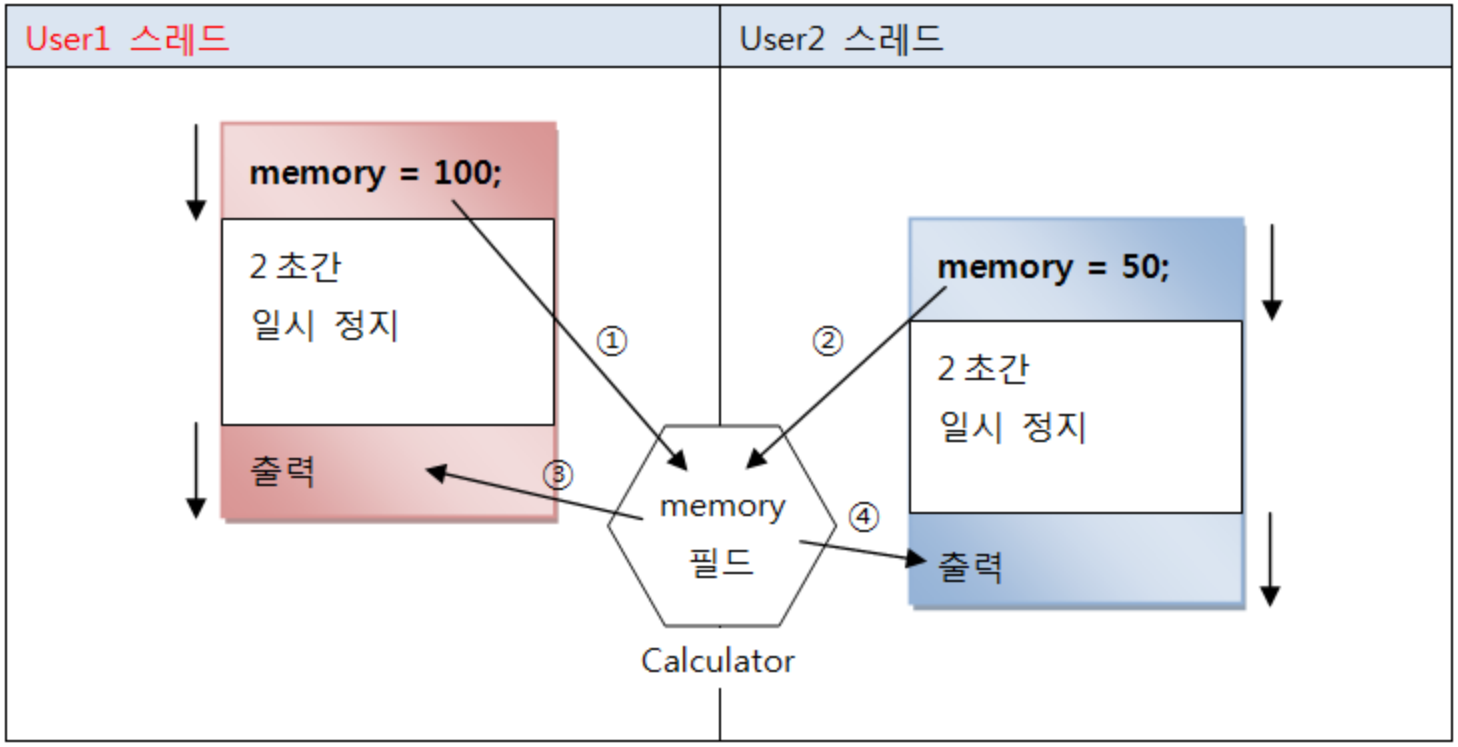
```
public class LoopThread extends Thread {
    public LoopThread(String name) {
        setName(name);
    }

    @Override
    public void run() {

            System.out.println(getName());

    }
}
```

```
public class LoopThreadEx {
    public static void main(String[] args) {
        for (int i=1; i<=10; i++) {
            Thread thread = new LoopThread("thread_"+i);
            if (i != 10)
                thread.setPriority(Thread.MIN_PRIORITY);
            else
                thread.setPriority(Thread.MAX_PRIORITY);

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

4절. 동기화 메소드와 동기화 블록

❖ 공유 객체를 사용할 때의 주의할 점

- 멀티 스레드가 하나의 객체를 공유해서 생기는 오류



4절. 동기화 메소드와 동기화 블록

❖ 공유 객체를 사용하는 경우

```
public class Calculator {
    private int memory;

    public int getMemory() {
        return memory;
    }

    public void setMemory(int memory) throws InterruptedException {
        this.memory = memory;

        Thread.sleep(2000);
        System.out.println(Thread.currentThread().getName()+":"+this.memory);
    }
}
```

4절. 동기화 메소드와 동기화 블록

❖ 공유 객체를 사용하는 경우

```
public class CalUser1 extends Thread {
    private Calculator calculator;

    public void setCalculator(Calculator calculator) {
        setName("User1");
        this.calculator = calculator;
    }

    @Override
    public void run() {
        try {
            calculator.setMemory(100);
        } catch (InterruptedException e) { }
    }
}
```

```
public class CalUser2 extends Thread {
    private Calculator calculator;

    public void setCalculator(Calculator calculator) {
        setName("User2");
        this.calculator = calculator;
    }

    @Override
    public void run() {
        try {
            calculator.setMemory(50);
        } catch (InterruptedException e) { }
    }
}
```

4절. 동기화 메소드와 동기화 블록

❖ 공유 객체를 사용하는 경우

```
public class CalEx {
    public static void main(String[] args) {
        Calculator calculator = new Calculator();

        CalUser1 user1 = new CalUser1();
        user1.setCalculator(calculator);
        user1.start();

        CalUser2 user2 = new CalUser2();
        user2.setCalculator(calculator);
        user2.start();
    }
}
```

4절. 동기화 메소드와 동기화 블록

❖ 공유 객체를 사용하는 경우

```
public class CalEx {
    public static void main(String[] args) {
        Calculator calculator = new Calculator();

        CalUser1 user1 = new CalUser1();
        user1.setCalculator(calculator);
        user1.start();

        CalUser2 user2 = new CalUser2();
        user2.setCalculator(calculator);
        user2.start();
    }
}
```

```
User1 : 50
User2 : 50
```

4절. 동기화 메소드와 동기화 블록

❖ 동기화 메소드 및 동기화 블록 – synchronized

- 스레드가 사용 중인 객체를 다른 스레드가 변경할 수 없도록 하기 위해 스레드 작업이 끝날 때까지 객체에 잠금 장치 적용
- 단 하나의 스레드만 실행할 수 있는 메소드 또는 블록
- 다른 스레드는 메소드나 블록 실행이 끝날 때까지 대기해야
- 동기화 메소드

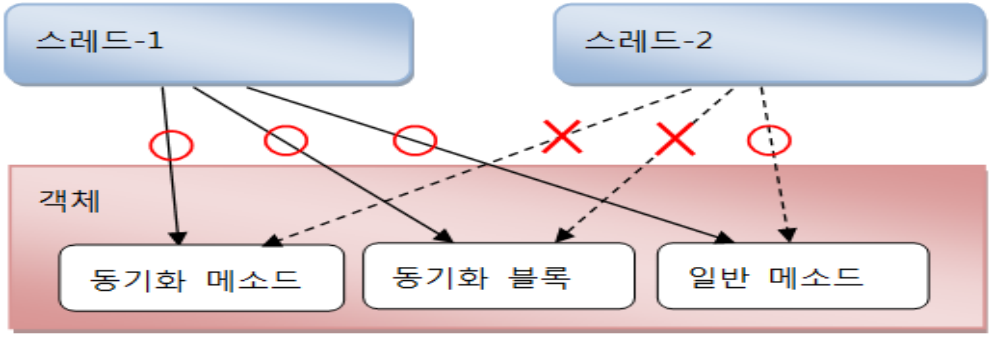
```
public synchronized void method() {
    임계 영역;  //단 하나의 스레드만 실행
}
```

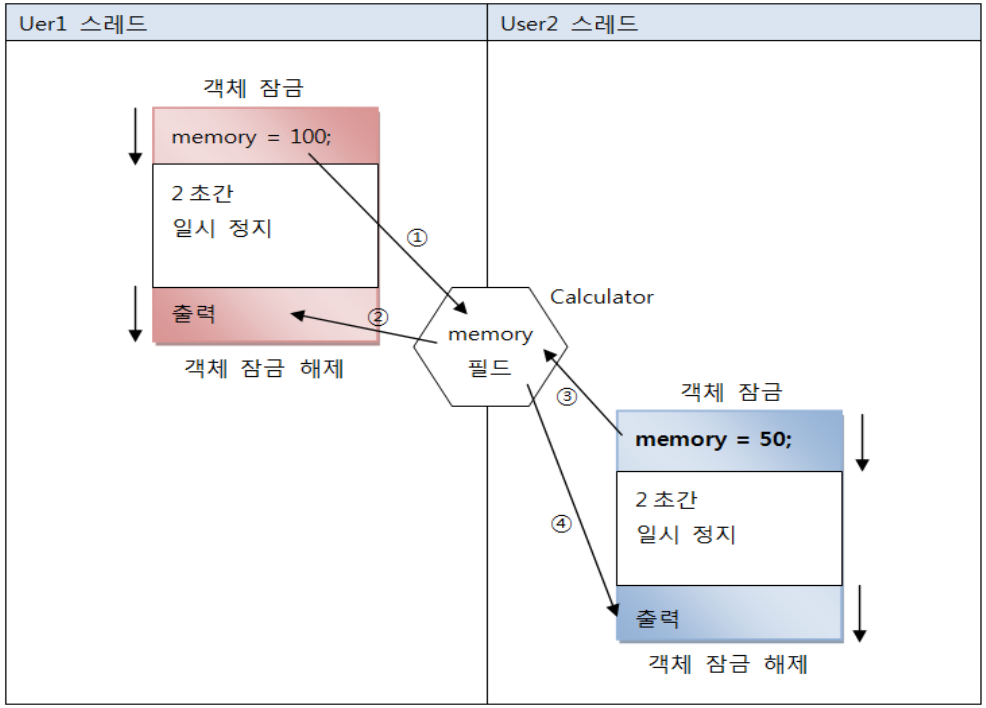
- 동기화 블록

```
public void method () {
  //여러 스레드가 실행 가능 영역
  ...
  synchronized(공유객체) {
    임계 영역 //단 하나의 스레드만 실행
  }
  //여러 스레드가 실행 가능 영역
  ...
}
```

4절. 동기화 메소드와 동기화 블록

❖ 동기화 메소드 및 동기화 블록





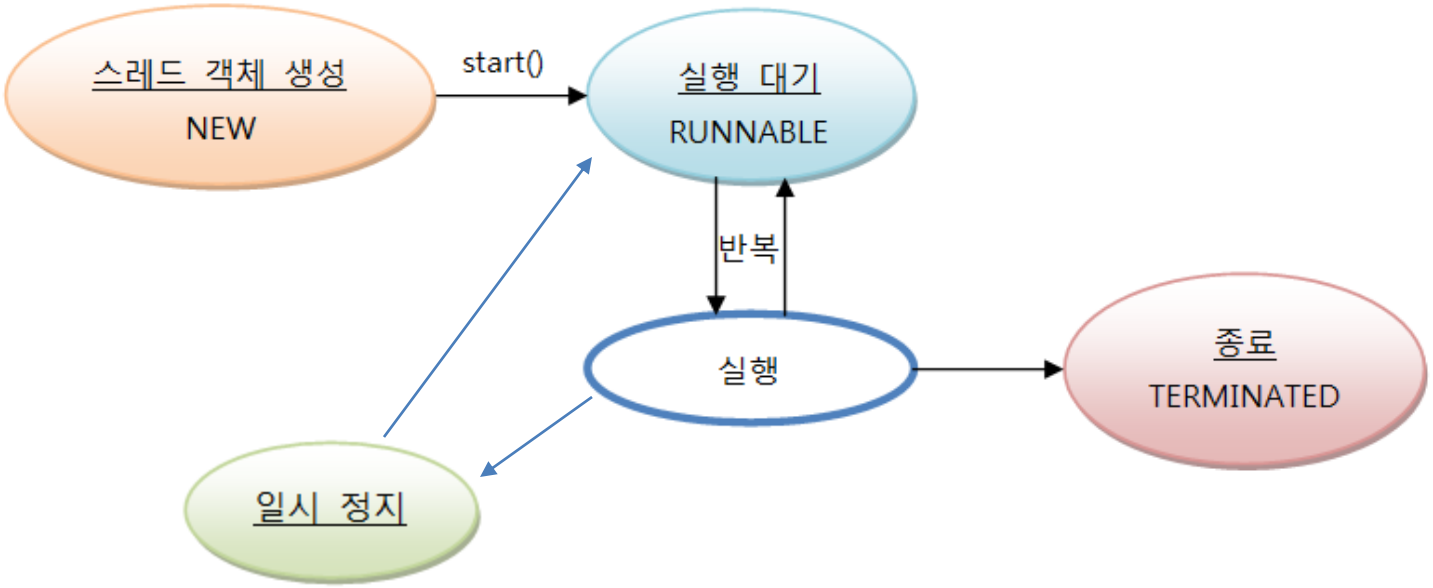
4절. 동기화 메소드와 동기화 블록

❖ 동기화 메소드 및 동기화 블록

```
public class Calculator {
    private int memory;

    public int getMemory() {
        return memory;
    }

    public synchronized void setMemory(int memory) {
        this.memory = memory;

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

        System.out.println(Thread.currentThread().getName()+":"+this.memory);
    }
}
```

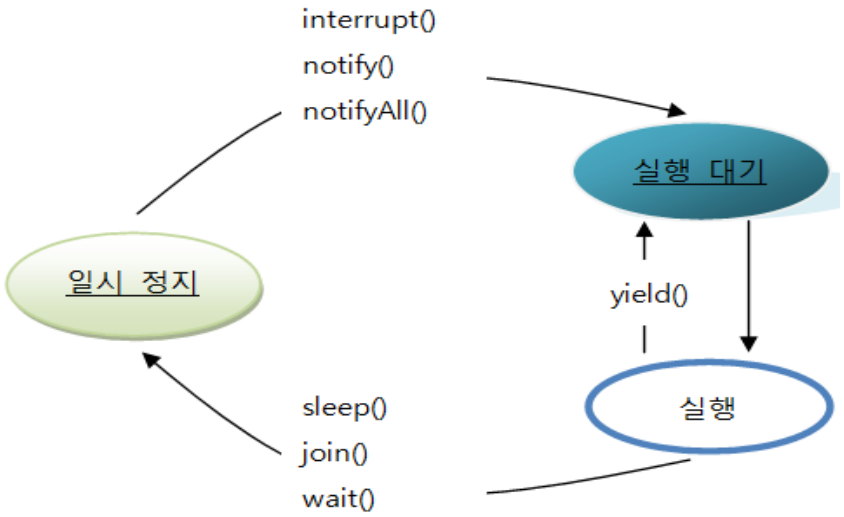
5절. 스레드 상태

❖ 스레드의 일반적인 상태



5절. 스레드 상태

❖ 스레드에 일시 정지 상태 도입한 경우

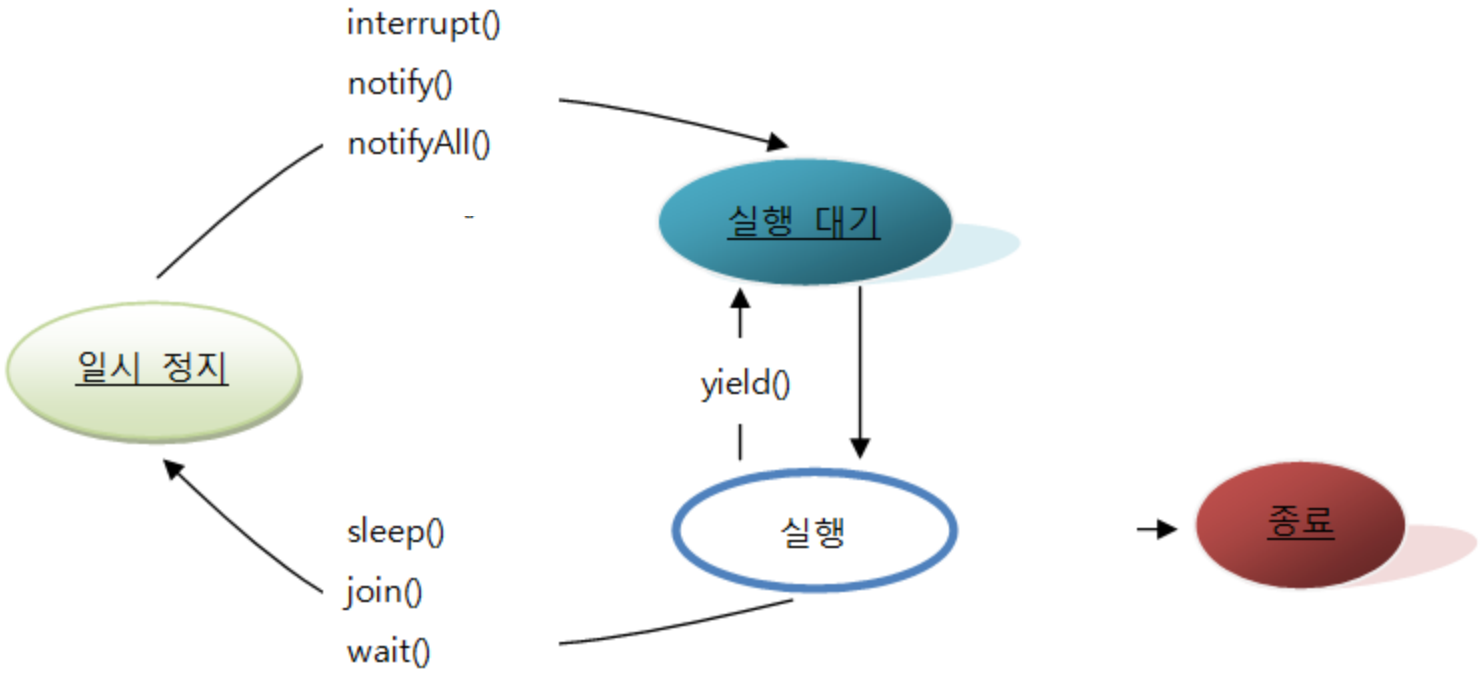


구 분	메 소 드	설 명
일시 정지로 보냄	**sleep(long millis)**	주어진 시간동안 스레드를 일시 정지 상태로 만듦. 주어진 시간이 지나면 자동으로 실행 대기 상태로 전환
	join()	join() 메소드를 호출한 스레드는 일시 정지 상태. join 메소드를 가진 스레드가 종료되면 실행 대기 상태로 전환
	wait()	동기화 블록 내에서 스레드를 일시 정지 상태로 만듦
일시 정지에서 벗어남	**interrupt()**	일시 정지 상태일 경우, exception을 발생시켜 ready 혹은 종료 상태로 전환
	notify(), notifyAll()	wait() 메소드로 일시 정지 상태인 스레드를 실행 대기 상태로 만듦
실행 대기로 보냄	**yield()**	실행 상태에서 다른 스레드에게 실행을 양보하고 실행 대기 상태로 전환

6절. 스레드 상태 제어

❖ 상태 제어

- 실행 중인 스레드의 상태를 변경하는 것
- 상태 변화를 가져오는 메소드의 종류



6절. 스레드 상태 제어

❖ 주어진 시간 동안 일시 정지 – sleep()

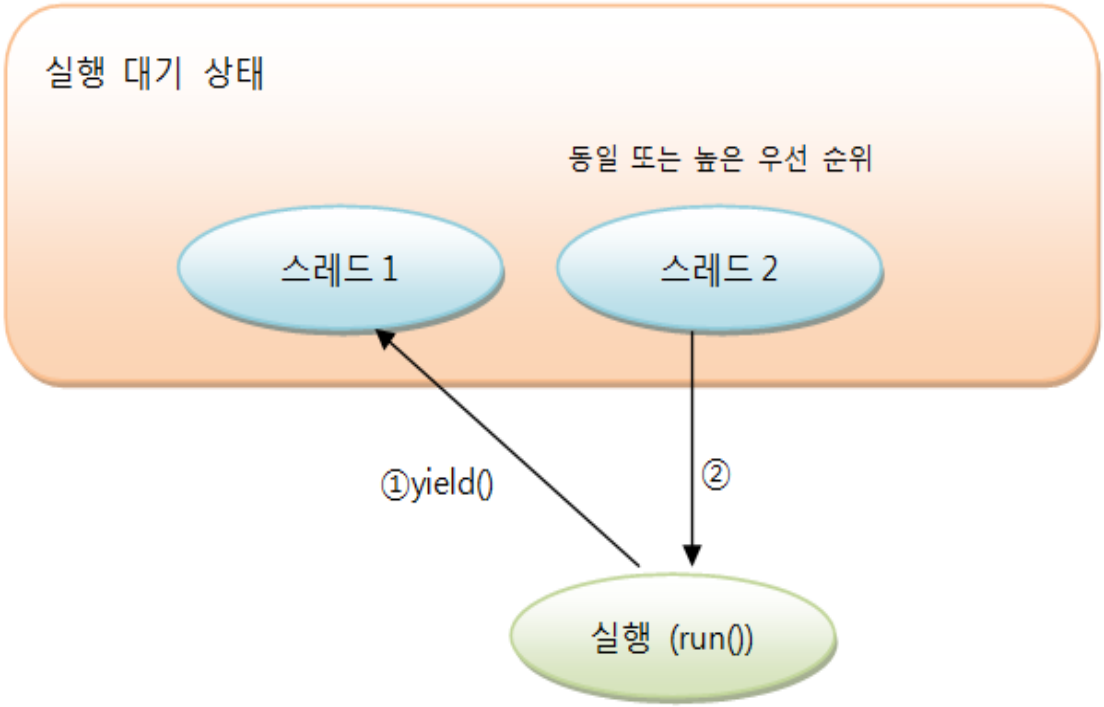
```
try {
   Thread.sleep(1000);
} catch(InterruptedException  e) {
   // interrupt()  메소드가  호출되면  실행
}
```

- 얼마 동안 일시 정지 상태로 있을 것인지 밀리 세컨드(1/1000) 단위로 지정
- 일시 정지 상태에서 주어진 시간이 되기 전에 interrupt() 메소드 호출되면
  - InterruptedException 발생
  - 그래서 sleep()는 예외 처리가 필요

6절. 스레드 상태 제어

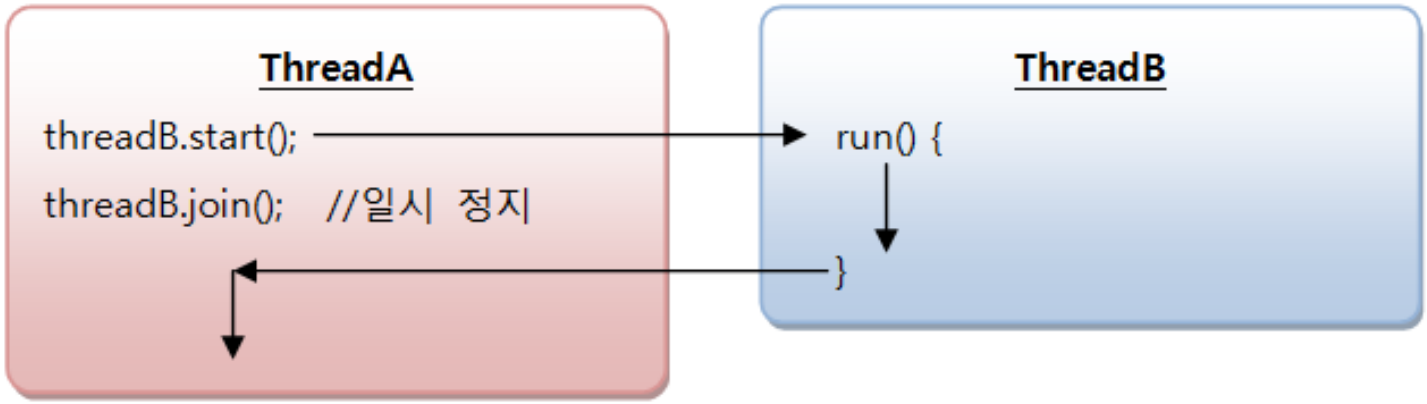
❖ 다른 스레드에게 실행 양보 – yield()

- Ex) 무의미하게 반복하는 스레드일 경우



6절. 스레드 상태 제어

❖ 다른 스레드의 종료를 기다림 – join()

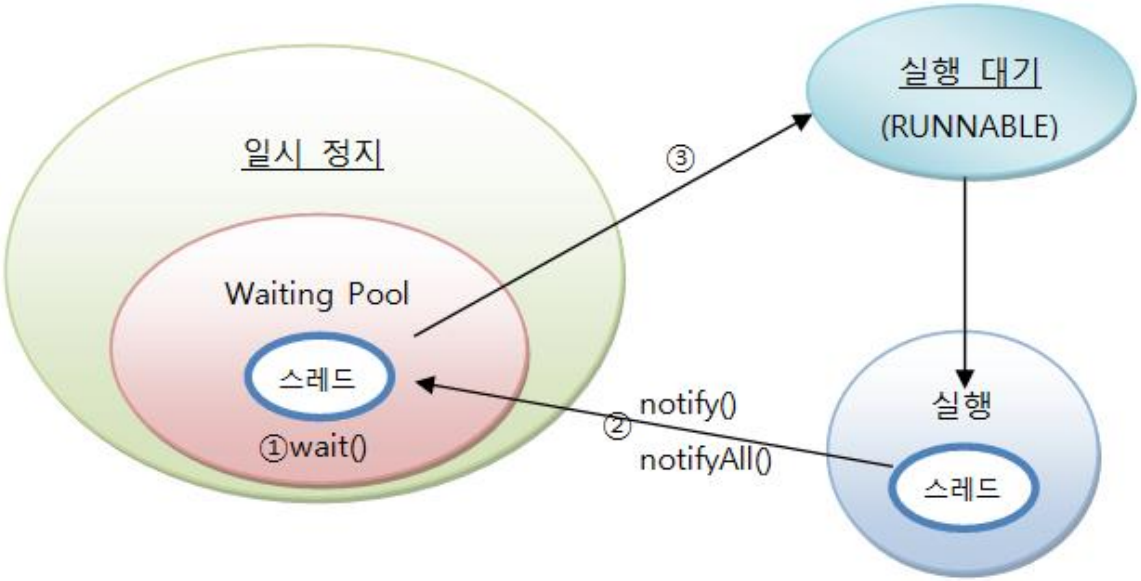


- 다른 스레드가 종료될 때까지 기다렸다가 실행해야 하는 경우
- 계산 작업을 하는 스레드가 모든 계산 작업 마쳤을 때, 결과값을 받아 이용하는 경우 주로 사용

6절. 스레드 상태 제어

❖ 스레드 간 협업 – wait(), notify(), notifyAll()

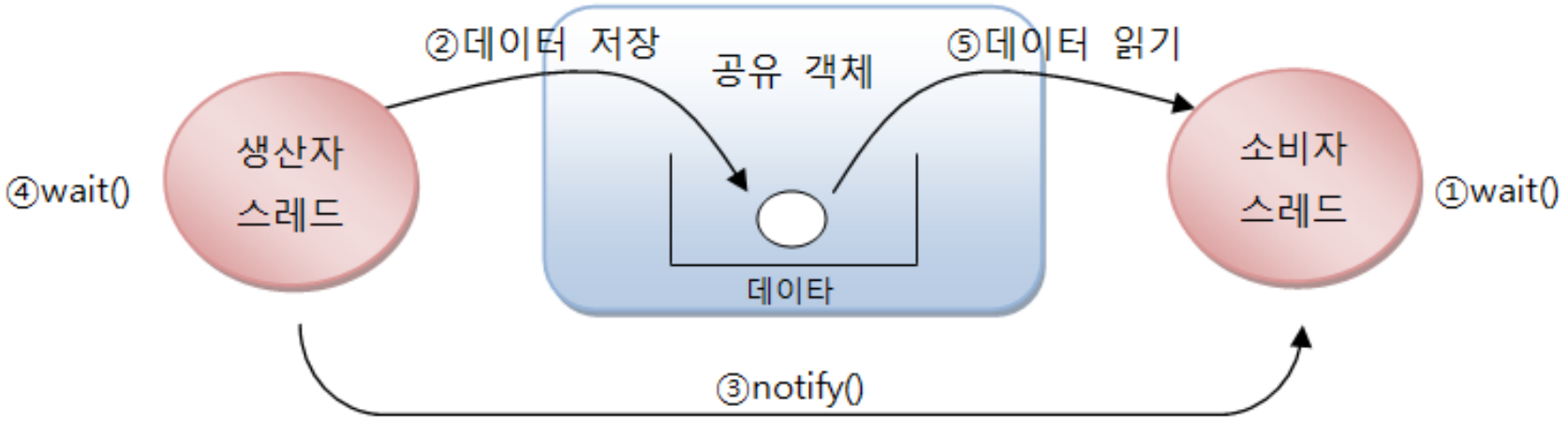
- 동기화 메소드 또는 블록에서만 호출 가능한 Object의 메소드



6절. 스레드 상태 제어

❖ 스레드 간 협업 – wait(), notify(), notifyAll()

- 두 개의 스레드가 교대로 번갈아 가며 실행해야 할 경우 주로 사용



< 생산자 소비자 문제 >

6절. 스레드 상태 제어

❖ 스레드의 안전한 종료 – stop 플래그, interrupt()

- 경우에 따라 실행 중인 스레드 즉시 종료해야 할 필요가 있을 때 사용
- stop() 메소드 사용시
  - 스레드 즉시 종료 되는 편리함
  - Deprecated(더 이상 사용하지 않음) – 사용 중이던 자원들이 불안전한 상태로 남겨짐

❖ 안전한 종료 위해 stop 플래그 이용하는 방법

- stop 플래그로 메소드의 정상 종료 유도

```
public class XXXThread extends Thread {
  private boolean stop;   //stop 플래그 필드

  public void run() {
    while( !stop ) {  ●---------- stop 이 true 가 되면 run() 이 종료된다.
      스레드가 반복 실행하는 코드;
    }
    //스레드가 사용한 자원 정리
  }
}
```

6절. 스레드 상태 제어

❖ 스레드의 안전한 종료

- interrupt() 메소드를 이용하는 방법
  - 스레드가 일시 정지 상태일 경우 InterruptedException 발생 시킴
    - 예외 처리를 통해 run() 메소드를 정상 종료시킬 수 있다.
  - 실행대기 또는 실행상태에서는 InterruptedException 발생하지 않음

6절. 스레드 상태 제어

❖ 스레드 상태 제어 예제

```
public class MyThreadEx {
    public static void main(String[] args) {
        for (int i=0; i<100; i++) {
            MyThread thread = new MyThread(i+1);
            thread.start();
            try {
                thread.join();
            } catch (InterruptedException e) { }
        }
    }
}
```

join() : 해당 스레드가 종료되기 전까지 다른 스레드를 실행하지 않는 기능

6절. 스레드 상태 제어

join() : 해당 스레드가 종료되기 전까지 다른 스레드를 실행하지 않는 기능

❖ 스레드 상태 제어 예제

```
public class MyThreadEx {
    public static void main(String[] args) {
        for (int i=0; i<100; i++) {
            MyThread thread = new MyThread(i+1);
            thread.start();
            try {
                thread.join();
            } catch (InterruptedException e) { }
        }
    }
}
```

```
public class MyThread extends Thread {
    private int x;

    public MyThread(int x) {
        this.x = x;
    }

    @Override
    public void run() {
        for (int i=0; i<5; i++)
            System.out.println(x + "번째 스레드입니다.");
    }
}
```

6절. 스레드 상태 제어

join() : 해당 스레드가 종료되기 전까지 다른 스레드를 실행하지 않는 기능

❖ 스레드 상태 제어 예제

```
public class MyThreadEx {
    public static void main(String[] args) {
        for (int i=0; i<100; i++) {
            MyThread thread = new MyThread(i+1);
            thread.start();
            try {
                thread.join();
            } catch (InterruptedException e) { }
        }
    }
}
```

```
public class MyThread extends Thread {
    private int x;

    public MyThread(int x) {
        this.x = x;
    }

    @Override
    public void run() {
        for (int i=0; i<5; i++)
            System.out.println(x + "번째 스레드입니다.");
    }
}
```

```
0번째 스레드입니다.
0번째 스레드입니다.
0번째 스레드입니다.
0번째 스레드입니다.
0번째 스레드입니다.
1번째 스레드입니다.
1번째 스레드입니다.
1번째 스레드입니다.
1번째 스레드입니다.
1번째 스레드입니다.
2번째 스레드입니다.
2번째 스레드입니다.
2번째 스레드입니다.
2번째 스레드입니다.
2번째 스레드입니다.
3번째 스레드입니다.
3번째 스레드입니다.
3번째 스레드입니다.
3번째 스레드입니다.
3번째 스레드입니다.
4번째 스레드입니다.
```

7절. 데몬 스레드

❖ 데몬(daemon) 스레드

- 주 스레드의 작업을 돕는 보조적인 역할 수행하는 스레드
- 주 스레드가 종료되면 데몬 스레드는 강제적으로 자동 종료
  - 워드프로세서의 자동저장, 미디어플레이어의 동영상 및 음악 재생, 가비지 컬렉터(GC) 등

- 스레드를 데몬 스레드로 만들기
  - 주 스레드가 데몬이 될 스레드의 setDaemon(true) 호출
  - 반드시 start() 메소드 호출 전에 setDaemon(true) 호출
    - 그렇지 않으면 IllegalThreadStateException이 발생

- 현재 실행중인 스레드가 데몬 스레드인지 구별법
  - isDaemon() 메소드의 리턴값 조사 – true면 데몬 스레드

7절. 데몬 스레드

setDaemon() 설정
- 메인 메소드는 메인 스레드로서 다른 스레드를 불러낸 후에 바로 종료되기 때문에 setDaemon(true)로 설정된 스레드는 메인 스레드가 종료되면 같이 종료된다

```
public class MyThread extends Thre
    private int x;

    public MyThread(int x) {
        this.x = x;
        setDaemon(true);
    }

    @Override
    public void run() {
        for (int i=0; i<5; i++)
            System.out.println(x + "번째 스레드입니다.");
    }
}
```

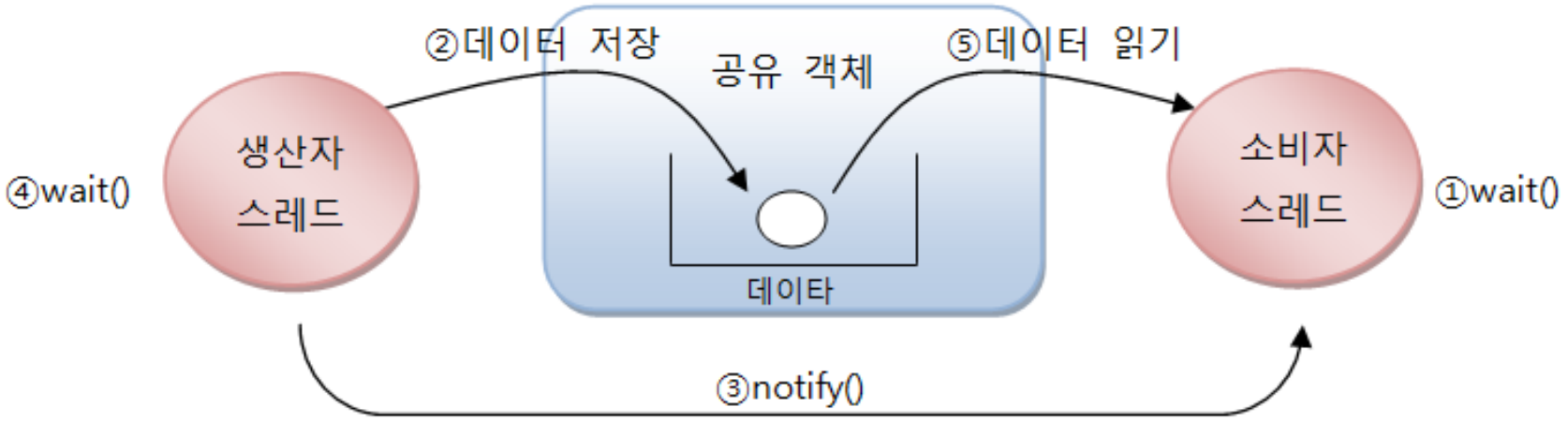
```
public class MyThreadEx {
    public static void main(String[] args) {
        for (int i=0; i<100; i++) {
            MyThread thread = new MyThread(i+1);
            thread.start();
            try {
                thread.join();
            } catch (InterruptedException e) { }
        }
    }
}
```

주석 처리 후 실행

실습 - 스레드 상태 제어

❖ 스레드간 협업 – wait(), notify(), notifyAll()

- 두 개의 스레드가 교대로 번갈아 가며 실행해야 할 경우 주로 사용



< 생산자 소비자 문제 >

실습 – 스레드 상태 제어

❖ DataBox 클래스

```
public class DataBox {
    private String data;

    public synchronized String getData() {
        if (this.data == null) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }

        String returnValue = data;
        System.out.println("ConsumerThread가 읽은 데이터 : "+returnValue);
        data = null;
        notify();
        return returnValue;
    }
    public synchronized void setData(String data) {
        if (this.data != null) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        this.data = data;
        System.out.println("ProducerThread가 생성한 데이터 : " + data);
        notify();
    }
}
```

실습 – 스레드 상태 제어

❖ DataBox 클래스

```
public class DataBox {
    private String data;

    public synchronized String getData() {
        if (this.data == null) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }

        String returnValue = data;
        System.out.println("ConsumerThread가 읽은 데이터 : "+returnValue);
        data = null;
        notify();
        return returnValue;
    }
    public synchronized void setData(String data) {
        if (this.data != null) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        this.data = data;
        System.out.println("ProducerThread가 생성한 데이터 : " + data);
        notify();
    }
}
```

data 필드가 null이면 소비자 스레드를 일시 정지 상태로 만든다.

data 필드를 null로 만들고 생산자 스레드를 실행 대기 상태로 만든다.

실습 – 스레드 상태 제어

❖ DataBox 클래스

```
public class DataBox {
    private String data;

    public synchronized String getData() {
        if (this.data == null) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }

        String returnValue = data;
        System.out.println("ConsumerThread가 읽은 데이터 : "+returnValue);
        data = null;
        notify();
        return returnValue;
    }
    public synchronized void setData(String data) {
        if (this.data != null) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        this.data = data;
        System.out.println("ProducerThread가 생성한 데이터 : " + data);
        notify();
    }
}
```

data 필드가 null이 아니면 생산자 스레드를 일시 정지 상태로 만든다.

data 필드에 값을 저장하고 소비자 스레드를 실행 대기 상태로 만든다.

실습 – 스레드 상태 제어

❖ **ProducerThread 클래스**

```
public class ProducerThread extends Thread {
    private DataBox dataBox;

    public ProducerThread(DataBox dataBox) {
        this.dataBox = dataBox;
    }

    @Override
    public void run() {
        for (int i=1; i<=3; i++) {
            String data = "Data-" + i;
            dataBox.setData(data);
        }
    }
}
```

실습 – 스레드 상태 제어

❖ **ProducerThread 클래스**

```
public class ProducerThread extends Thread {
    private DataBox dataBox;

    public ProducerThread(DataBox dataBox) {
        this.dataBox = dataBox;
    }

    @Override
    public void run() {
        for (int i=1; i<=3; i++) {
            String data = "Data-" + i;
            dataBox.setData(data);
        }
    }
}
```

공유 객체를 필드에 저장

새로운 데이터를 저장

실습 – 스레드 상태 제어

❖ ConsumerThread 클래스

```
public class ConsumerThread extends Thread {
    private DataBox dataBox;

    public ConsumerThread(DataBox dataBox) {
        this.dataBox = dataBox;
    }

    @Override
    public void run() {
        for (int i=1; i<=3; i++) {
            String data = dataBox.getData();
        }
    }
}
```

실습 – 스레드 상태 제어

❖ ConsumerThread 클래스

```
public class ConsumerThread extends Thread {
    private DataBox dataBox;

    public ConsumerThread(DataBox dataBox) {
        this.dataBox = dataBox;
    }

    @Override
    public void run() {
        for (int i=1; i<=3; i++) {
            String data = dataBox.getData();
        }
    }
}
```

공유 객체를 필드에 저장

새로운 데이터를 읽음

실습 – 스레드 상태 제어

❖ **WaitNotifyExample 클래스**

```
public class WaitNotifyExample {
    public static void main(String[] args) {
        DataBox dataBox = new DataBox();

        ProducerThread prodThread = new ProducerThread(dataBox);
        ConsumerThread consThread = new ConsumerThread(dataBox);

        prodThread.start();
        consThread.start();
    }
}
```

실습 – 스레드 상태 제어

❖ **WaitNotifyExample 클래스**

```
public class WaitNotifyExample {
    public static void main(String[] args) {
        DataBox dataBox = new DataBox();

        ProducerThread prodThread = new ProducerThread(dataBox);
        ConsumerThread consThread = new ConsumerThread(dataBox);

        prodThread.start();
        consThread.start();
    }
}
```

```
Producer Thread가 생성한 데이터 : Data-1
Consumer Thread가 읽은 데이터 : Data-1
Producer Thread가 생성한 데이터 : Data-2
Consumer Thread가 읽은 데이터 : Data-2
Producer Thread가 생성한 데이터 : Data-3
Consumer Thread가 읽은 데이터 : Data-3
```