어서와 Java는 처음이지!

제16장 스레드

멀티태스킹

- 멀티 태스킹(muli-tasking)는 여러 개의 애플리케이션을 동시에 실행하여서 컴퓨터 시스템의 성능을 높이기 위한 기법이다

그림 16-1 • 병렬 처리의 예

스레드란?

- 다중 스레딩(multi-threading)은 하나의 프로그램이 동시에 여러 가지 작업을 할 수 있도록 하는 것
- 각각의 작업은 스레드(thread)라고 불린다.

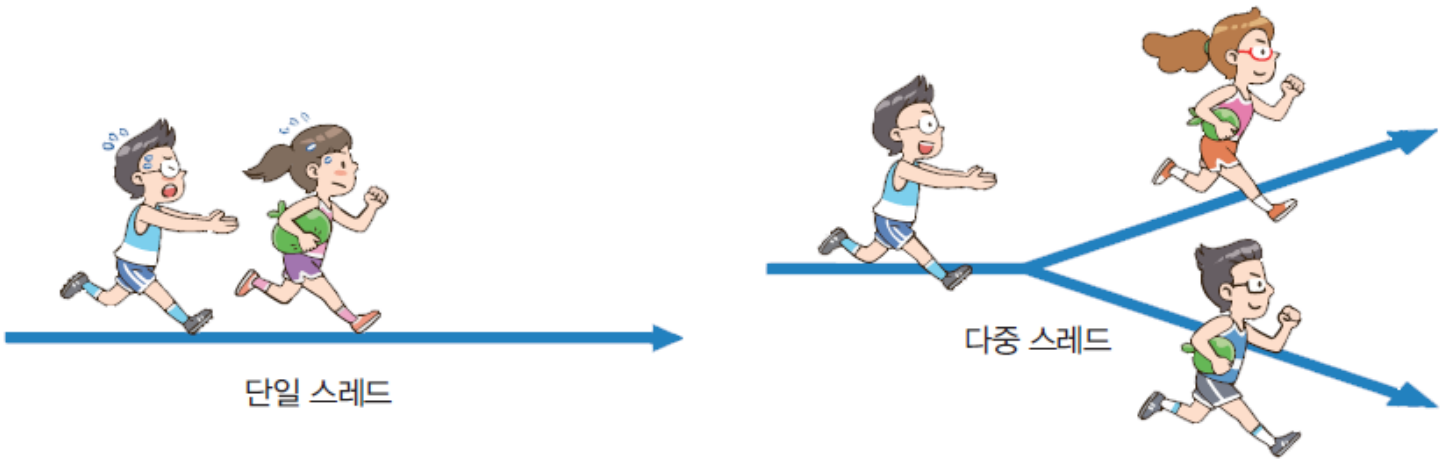


그림 16-2 • 다중 스레드의 개념

프로세스와 스레드

- 프로세스(process): 자신만의 데이터를 가진다.
- 스레드(thread): 동일한 데이터를 공유한다.

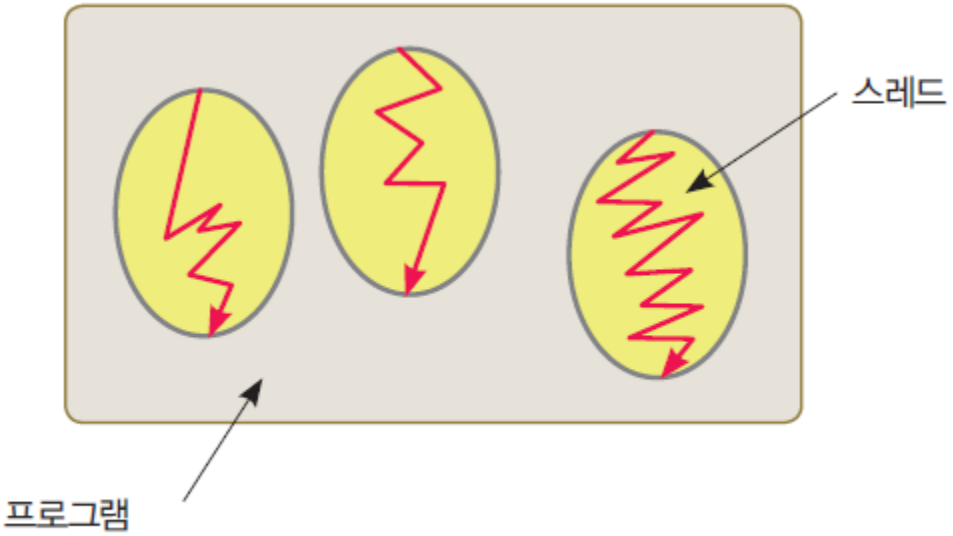


그림 16-3 • 스레드는 하나의 프로세스 안에 존재한다.

스레드를 사용하는 이유

- 웹 브라우저에서 웹 페이지를 보면서 동시에 파일을 다운로드할 수 있도록 한다.
- 워드 프로세서에서 문서를 편집하면서 동시에 인쇄한다.
- 게임 프로그램에서는 응답성을 높이기 위하여 많은 스레드를 사용한다.
- GUI에서는 마우스와 키보드 입력을 다른 스레드를 생성하여 처리한다.

스레드 생성과 실행

- 스레드는 Thread 클래스가 담당한다.

```
Thread t = new Thread();        // 스레드 객체를 생성한다.
t.start();                      // 스레드를 시작한다.
```

스레드를 생성하는 방법

- Thread 클래스를 상속하는 방법:
  - Thread 클래스를 상속받은 후에 run() 메소드를 재정의한다.
  - run() 메소드 안에 작업을 기술한다.

- Runnable 인터페이스를 구현하는 방법:
  - run() 메소드를 가지고 있는 클래스를 작성하고 ,
  - 이 클래스의 객체를 Thread 클래스의 생성자를 호출할 때 전달한다.

Thread 클래스를 상속하는 방법

```
class MyThread extends Thread {
        public void run() {
                for (int i = 10; i >= 0; i--)
                        System.out.print(i + " ");
        }
}
public class MyThreadTest {
        public static void main(String args[]) {
                Thread t = new MyThread();
                t.start();
        }
}
```


```
10 9 8 7 6 5 4 3 2 1 0
```

Runnable 인터페이스를 구현하는 방법

① Runnable 인터페이스를 구현한 클래스를 작성한다.

② run() 메소드를 작성한다.

③ Thread 객체를 생성하고 이때 MyRunnable 객체를 인수로 전달한다.

④ start()를 호출하여서 스레드를 시작한다.

Thread 객체 = 일꾼

Runnable 객체 = 작업의 내용

Runnable 인터페이스를 구현하는 방법

```
class MyRunnable implements Runnable {
        public void run() {
                for (int i = 10; i >= 0; i--)
                        System.out.print(i + " ");
        }
}
public class MyRunnableTest {
        public static void main(String args[]) {
                Thread t = new Thread(new MyRunnable());
                t.start();
        }
}
```


```
10 9 8 7 6 5 4 3 2 1 0
```

어떤 방법이 좋은가?

- 자바에서 다중 상속이 불가능한 것을 감안한다면 Runnable 인터페이스를 사용하는 것이 좋다.
- Runnable 인터페이스를 사용하면 고수준의 스레드 관리 API도 사용할 수 있다.

Runnable 구현

Thread 상속

예제

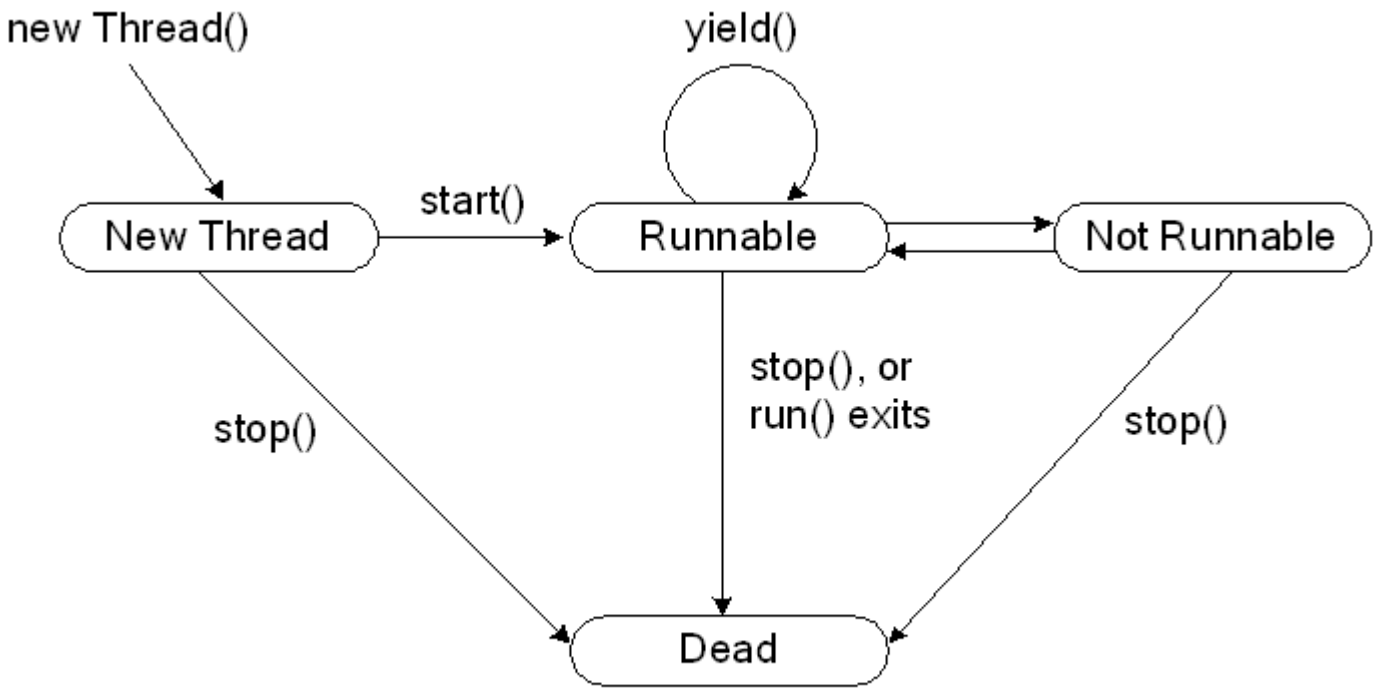
```
class MyRunnable implements Runnable {
        String myName;
        public MyRunnable(String name) {                    myName = name;      }
        public void run() {
                for (int i = 10; i >= 0; i--)
                        System.out.print(myName + i + " ");
        }
}
public class TestThread {
        public static void main(String[] args) {
                Thread t1 = new Thread(new MyRunnable("A"));
                Thread t2 = new Thread(new MyRunnable("B"));
                t1.start();
                t2.start();
        }
}
```


```
A10 B10 A9 B9 B8 A8 B7 B6 A7 B5 A6 B4 A5 B3 A4 A3 A2 B2 A1 B1 A0 B0
```

Thread 클래스

메소드	설명
Thread()	매개 변수가 없는 기본 생성자
Thread(String *name*)	이름이 name인 Thread 객체를 생성한다
Thread(Runnable *target*, String *name*)	Runnable을 구현하는 객체로부터 스레드를 생성한다.
static int activeCount()	현재 활동중인 스레드의 개수를 반환한다.
String getName()	스레드의 이름을 반환
int getPriority()	스레드의 우선 순위를 반환
void interrupt()	현재의 스레드를 중단한다.
boolean isInterrupted()	현재의 스레드가 중단될 수 있는지를 검사
void setPriority(int *priority*)	스레드의 우선 순위를 지정한다.
void setName(String *name*)	스레드의 이름을 지정한다.
static void sleep(int *milliseconds*)	현재의 스레드를 지정된 시간만큼 재운다.
void run()	스레드가 시작될 때 이 메소드가 호출된다. 스레드가 하여야 하는 작업을 이 메소드 안에 위치시킨다.
void start()	스레드를 시작한다.
static void yield()	현재 스레드를 다른 스레드에 양보하게 만든다.

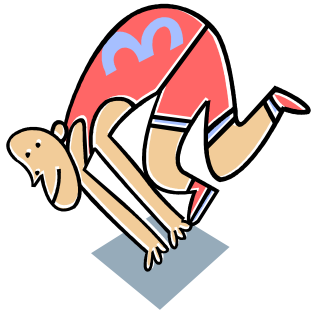
스레드 상태


new Thread()
yield()
start()
New Thread
Runnable
Not Runnable
stop()
stop(), or
run() exits
stop()
Dead

생성 상태와 실행 가능 상태

- 생성 상태
  - Thread 클래스를 이용하여 새로운 스레드를 생성
  - start()는 생성된 스레드를 시작
  - stop()은 생성된 스레드를 멈추게 한다.

- 실행 가능 상태
  - 스레드가 스케줄링 큐에 넣어지고 스케줄러에 의해 우선순위에 따라 실행

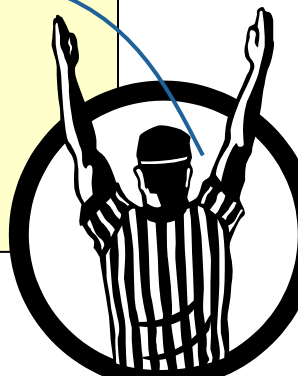


실행 중지 상태

- 실행 가능한 상태에서 다음의 이벤트가 발생하면 실행 중지 상태로 된다.
  - 스레드나 다른 스레드가 suspend()를 호출하는 경우
  - 스레드가 wait()를 호출하는 경우
  - 스레드가 sleep()을 호출하는 경우
  - 스레드가 입출력 작업을 하기 위해 대기하는 경우

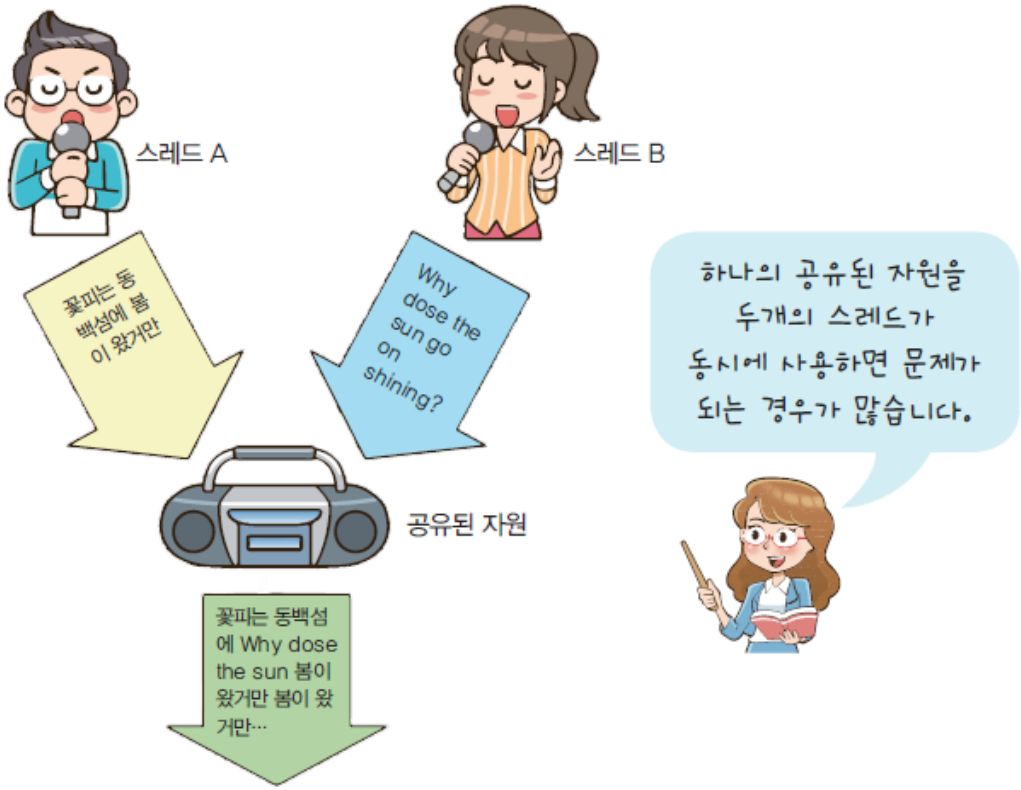
강제적인 종료

```
public static void main(String args[]) throws InterruptedException {
                int tries = 0;
                print("추가적인 스레드를 시작합니다.");
                Thread t = new Thread(new MessageLoop());
                t.start();
                print("추가적인 스레드가 끝나기를 기다립니다.");
                while (t.isAlive()) {
                        print("아직 기다립니다.");
                        t.join(1000);
                        tries++;
                        if (tries > 2) {
                                print("참을 수 없네요!");
                                t.interrupt();
                                t.join();
                        }
                }
                print("메인 스레드 종료!");
        }
}
```



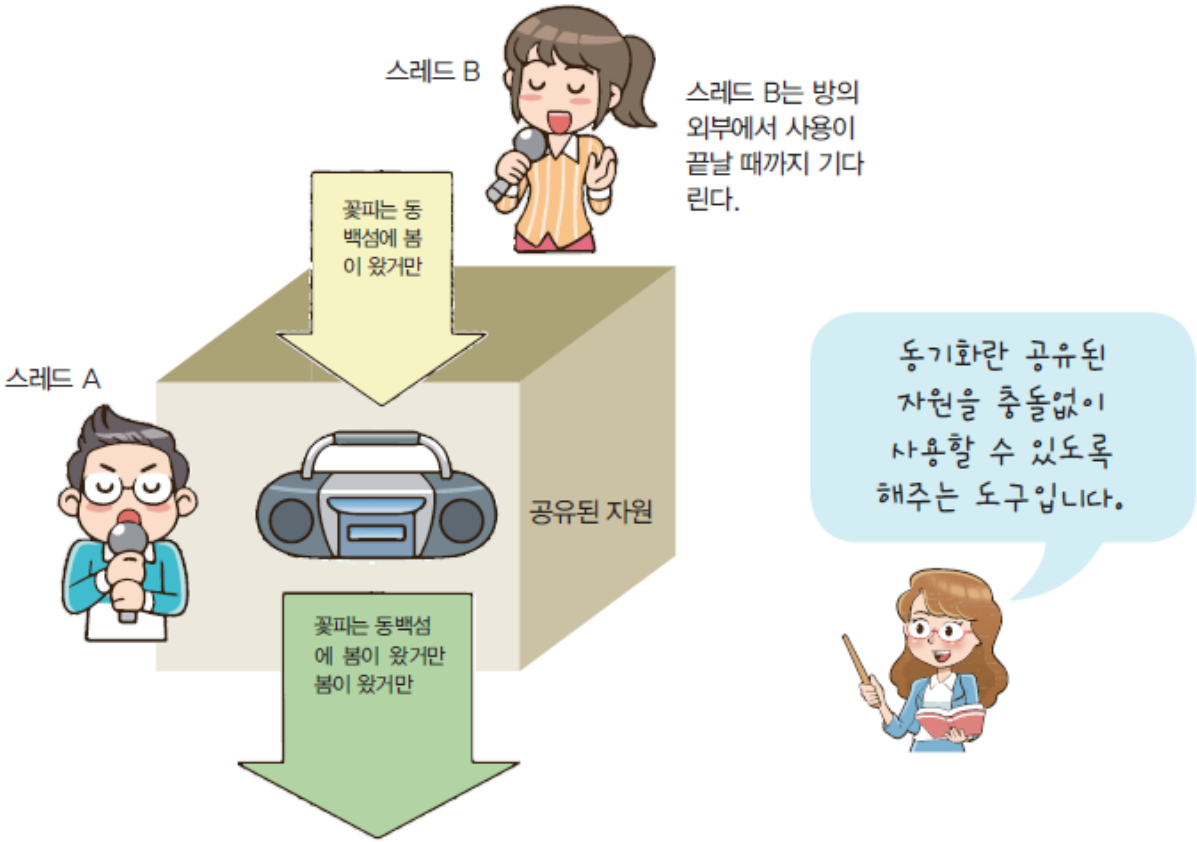
동기화

- 동기화(synchronization): 한 번에 하나의 스레드 만이 공유 데이터를 접근할 수 있도록 제어하는 것이 필요



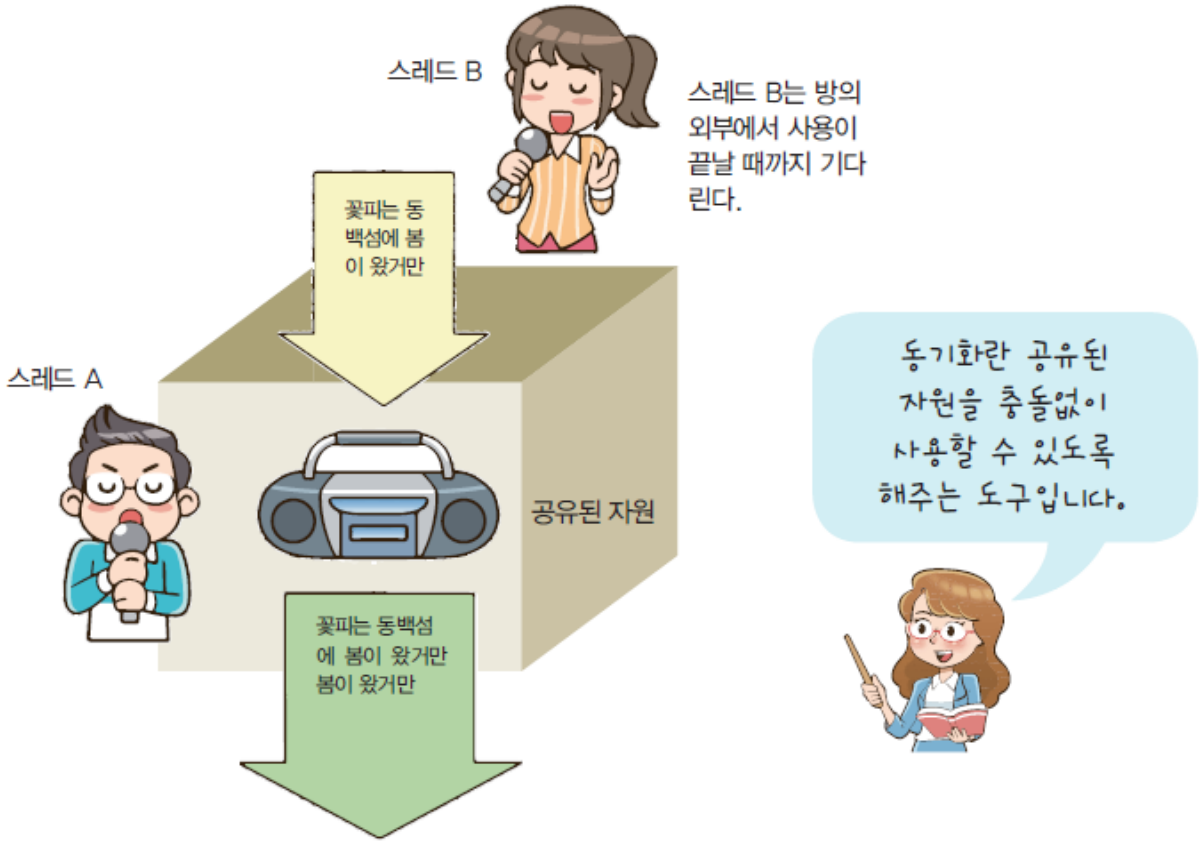
동기화의 기본 해법

- 밀폐된 방 안에 자원을 놓고 한 번에 하나의 스레드만 방문을 열고 사용할 수 있게 한다.



동기화의 기본 해법

- 하나의 스레드의 작업이 끝나면다음 스레드가 사용할 수 있도록 한다.



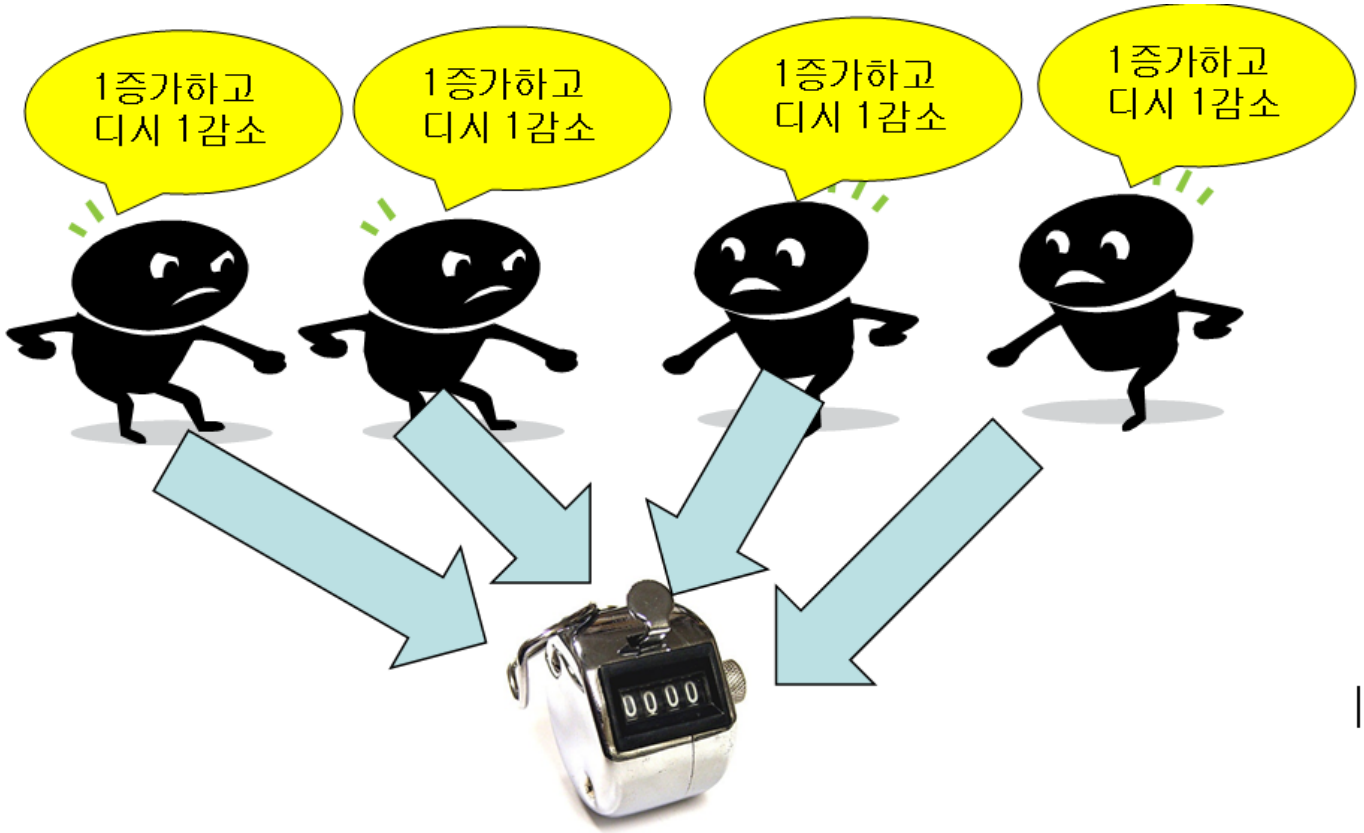
스레드 간섭

- 스레드 간섭(thread interference)이란
  - 서로 다른 스레드에서 실행되는 두 개의 연산이 동일한 데이터에 적용되면서 서로 겹치는 것을 의미한다.

```
class Counter {
        private int value = 0;
        public void increment() { value++;          }
        public void decrement() { value--; }
        public void printCounter() {        System.out.println(value);  }
}
```

아주 발견하기 힘든 버그

- 4개의 스레드가 하나의 카운터를 공유한다고 가정하자.



스레드 간섭

```
class Counter {
        private int value = 0;
        public void increment() { value++;          }
        public void decrement() { value--;          }
        public void printCounter() {                System.out.println(value);          }
}
class MyThread extends Thread {
        Counter sharedCounter;
        public MyThread(Counter c) {
                this.sharedCounter = c;
        }
```

스레드 간섭

```
        public void run() {
                int i = 0;
                while (i < 20000) {
                        sharedCounter.increment();
                        sharedCounter.decrement();
                        if (i % 40 == 0)
                                sharedCounter.printCounter();
                        try {
                                sleep((int) (Math.random() * 2));
                        } catch (InterruptedException e) {              }
                        i++;
                }
        }
}
public class CounterTest {
        public static void main(String[] args) {
                Counter c = new Counter();
                new MyThread(c).start();
                new MyThread(c).start();
                new MyThread(c).start();
                new MyThread(c).start();
        }
}
```

실행 결과


```
...
-7
-7
-7
-8
-7
-7
...
```

해결 방법

- 동기화된 메소드(synchronized methods)

```
class Counter {
        private int value = 0;
        public synchronized void increment() { value++;       }
        public synchronized void deccrement() { value--;       }
        public synchronized void printCounter() {  System.out.println(value);          }
}
```


```
0
0
0
0
0
1
0
0
...
```

스레드간의 조정

- 만약 두개의 스레드가 데이터를 주고 받는 경우에 발생

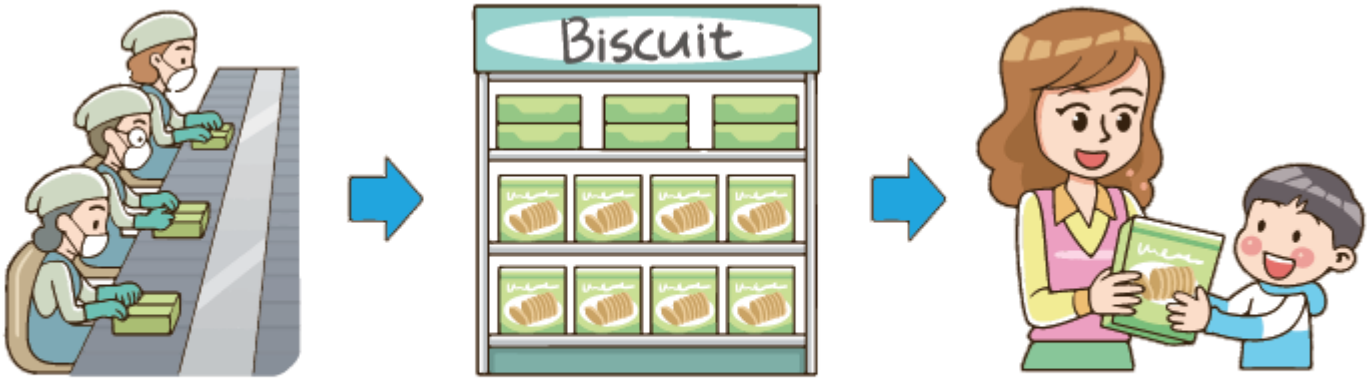
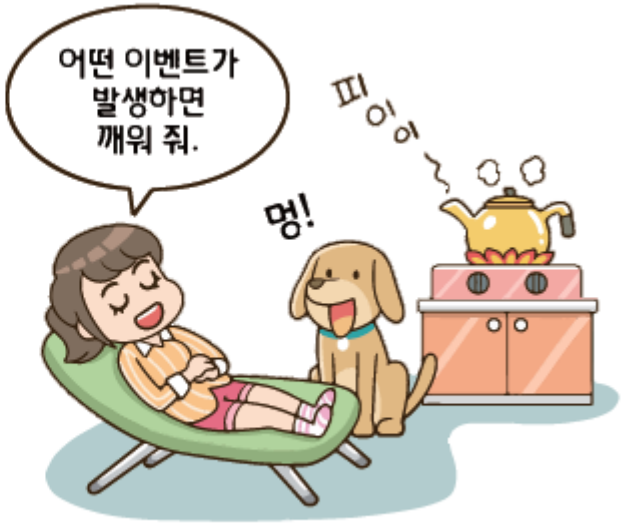


그림 16-5 • 생산자와 소비자 문제

2가지의 방법

좋지 못한 방법(polling)



좋은 방법(wait & notify)

wait()와 notify()

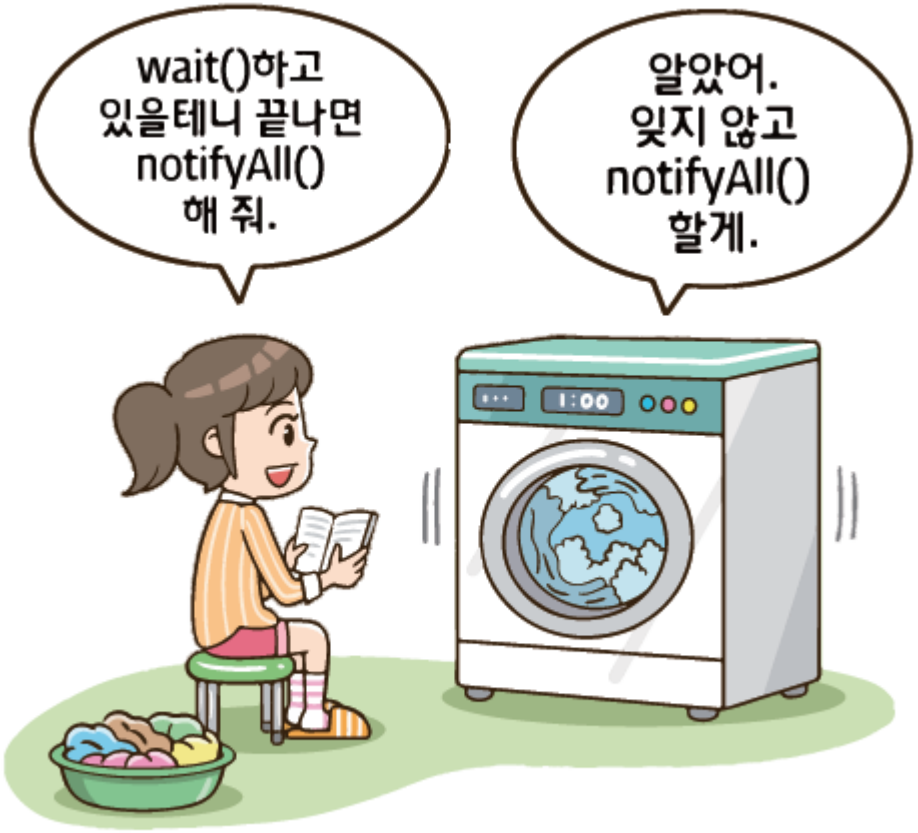
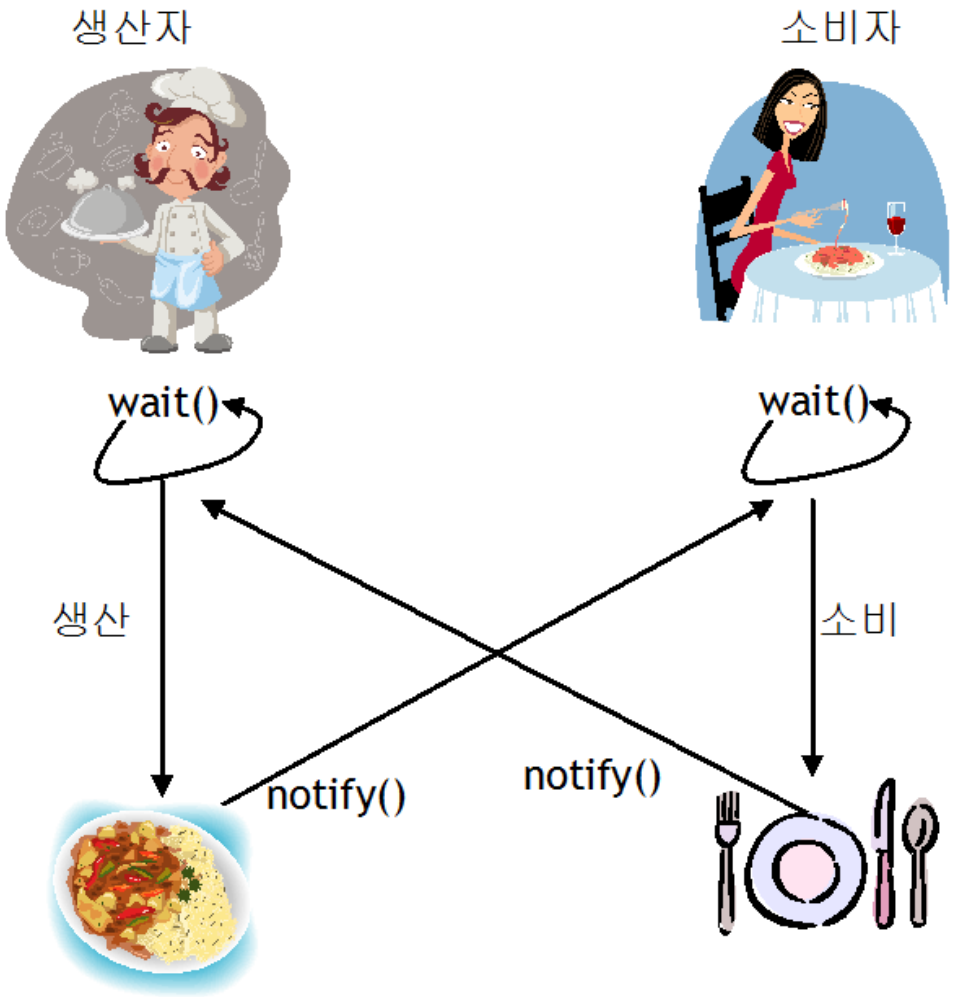


그림 16-6 • wait() notify()

생산자/소비자 문제에 적용



Buffer 클래스

```
class Buffer {
        private int data;
        private boolean empty = true;
        public synchronized int get() {
                while (empty) {
                        try {
                                wait();
                        } catch (InterruptedException e) {
                        }
                }
                empty = true;
                notifyAll();
                return data;
        }
```

Buffer 클래스

```
public synchronized void put(int data) {
                        while (!empty) {
                                try {
                                        wait();
                                } catch (InterruptedException e) {
                                }
                        }
                        empty = false;
                        this.data = data;
                        notifyAll();
                }
}
```

생산자

```
class Producer implements Runnable {
        private Buffer buffer;
        public Producer(Buffer buffer) {
                this.buffer= buffer;
        }
        public void run() {
                for (int i = 0; i < 10; i++) {
                        buffer.put(i);
                        System.out.println("생산자: " + i + "번 케익을
생산하였습니다.");
                        try {
                                Thread.sleep((int) (Math.random() * 100));
                        } catch (InterruptedException e) {
                        }
                }
        }
}
```

소비자

```
class Consumer implements Runnable {
        private Buffer buffer;
        public Consumer(Buffer drop) {
                this.buffer= drop;
        }
        public void run() {
                for (int i = 0; i < 10; i++) {
                        int data = buffer.get();
                        System.out.println("소비자: " + data + "번 케익을
소비하였습니다.");
                        try {
                                Thread.sleep((int) (Math.random() * 100));
                        } catch (InterruptedException e) {
                        }
                }
        }
}
```

실행 결과


```
생산자: 0번 케익을 생산하였습니다.
소비자: 0번 케익을 소비하였습니다.
생산자: 1번 케익을 생산하였습니다.
소비자: 1번 케익을 소비하였습니다.
...
생산자: 9번 케익을 생산하였습니다.
소비자: 9번 케익을 소비하였습니다.
```

중간 점검 문제

1. wait()와 notify() 메소드는 왜 필요한가?
2. wait()는 어떤 역할을 하는가?
3. notify()는 어떤 역할을 하는가?