스레드란?

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

그림23-1. 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.

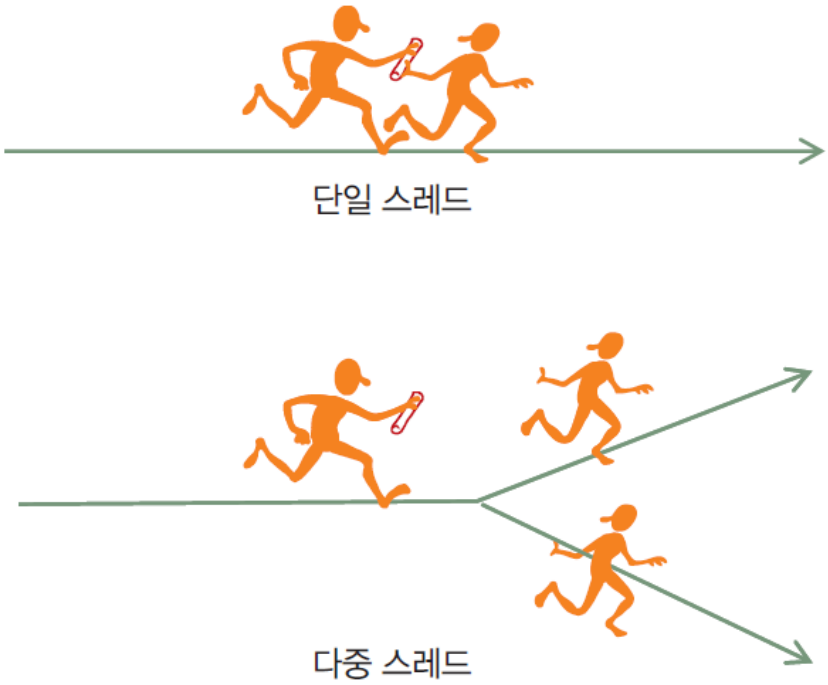


그림23-2. 다중 스레드의 개념

프로세스와 스레드

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

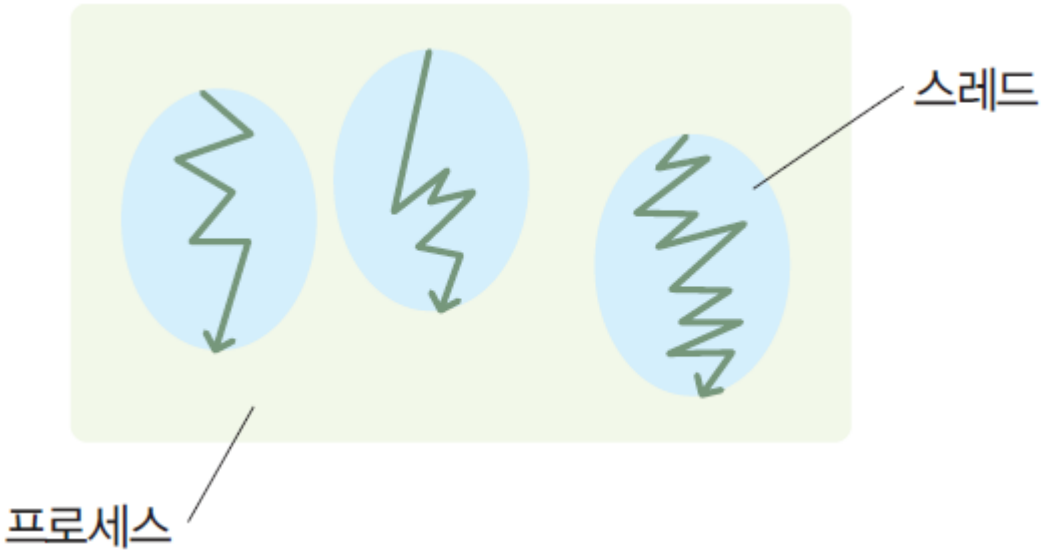


그림23-3. 스레드는 하나의 프로세스 안에 존재한다.

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.

동시작업의 문제점

- 같은 데이터를 공유함으로 동기화의 문제해결해야.

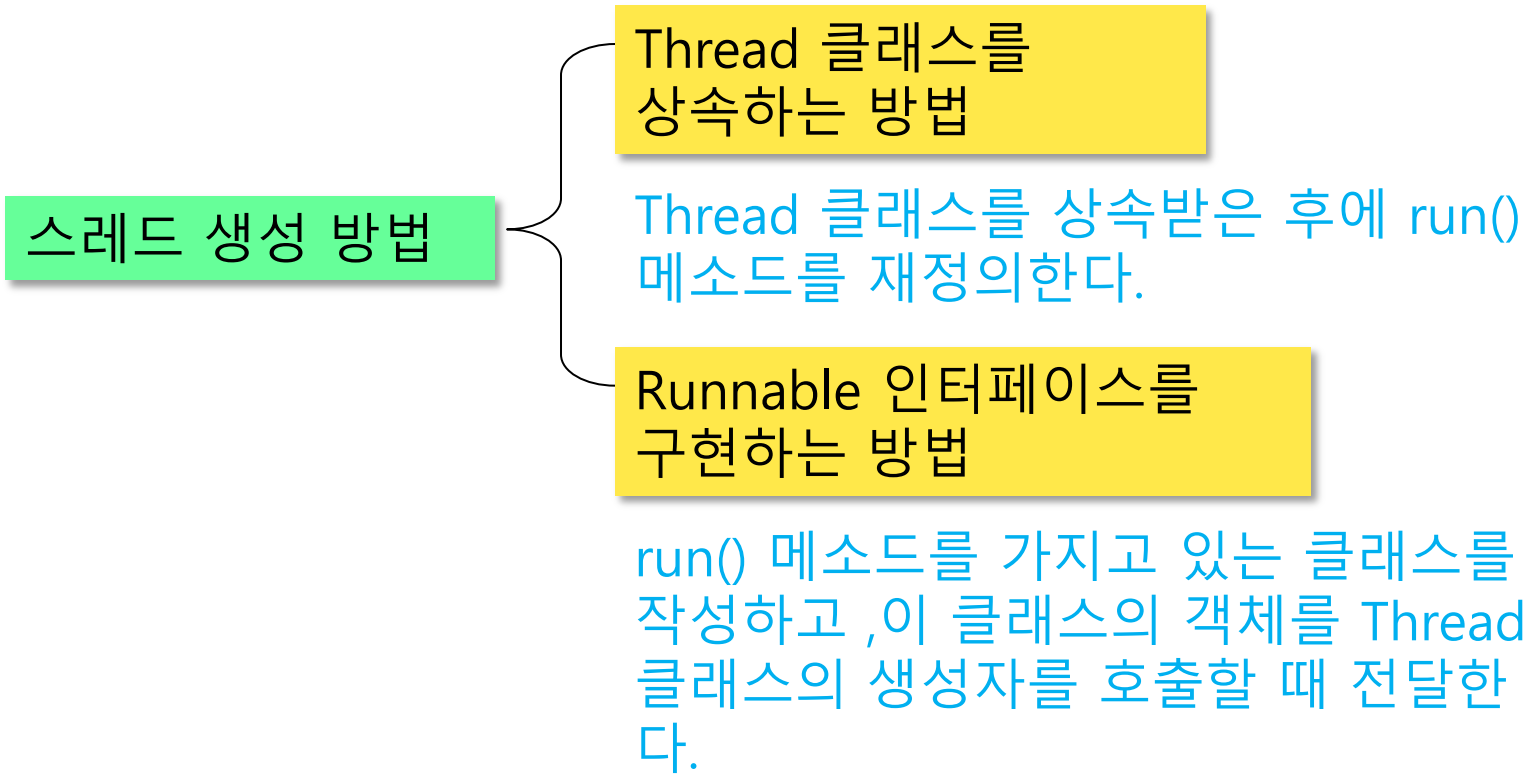
스레드 생성과 실행

- 스레드는 Thread 클래스가 담당한다.

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

- 스레드의 작업은 Thread 클래스의 run() 메소드 안에 기술한다.

스레드 생성과 실행


스레드 생성 방법
Thread 클래스를 상속하는 방법
Thread 클래스를 상속받은 후에 run() 메소드를 재정의한다.
Runnable 인터페이스를 구현하는 방법
run() 메소드를 가지고 있는 클래스를 작성하고 ,이 클래스의 객체를 Thread 클래스의 생성자를 호출할 때 전달한다.

Thread 클래스를 상속하기

- Thread를 상속받아서 클래스를 작성한다.
- run() 메소드를 재정의한다.

```
class MyThread extends Thread {
    public void run() {
        ...                  <---------- 여기에 수행하여야 하는 작업을 적어준다.
    }
}
```

- Thread 객체를 생성한다.

```
Thread t = new MyThread();
```

- start()를 호출하여서 스레드를 시작한다.

```
t.start();
```

- 단점
  - 단일 상속만이 가능하므로 다른 클래스를 상속받은 클래스는 스레드를 만들수 없다.
    ➔ Runnable 인터페이스로 구현

Thread 클래스를 상속하기

MyThreadTest.java

```
01  class MyThread extends Thread {
02      public void run() {
03          for (int i = 10; i >= 0; i--)
04              System.out.print(i + " ");
05      }
06  }
07
08  public class MyThreadTest {
09      public static void main(String args[]) {
10          Thread t = new MyThread();
11          t.start();
12      }
13  }
```

MyThread 클래스는 Thread를 상속받는다. Thread 클래스는 java.lang 패키지에 들어 있어서 따로 import할 필요가 없다. MyThread 클래스는 하나의 메소드 run()만을 가지고 있는데 run()은 이 스레드가 시작되면 자바 런타임 시스템에 의하여 호출된다. 스레드가 실행하는 모든 작업은 이 run() 메소드 안에 있어야 한다. 현재는 단순히 10부터 0까지를 화면에 출력한다.

스레드를 실행시키려면 Thread에서 파생된 클래스 MyThread의 인스턴스를 생성한 후 start()를 호출한다. Thread 타입의 변수 t가 선언되고 MyThread의 객체가 생성하였다. 객체가 생성되었다고 스레드가 바로 시작되는 것은 아니다. start() 메소드를 호출해야만 스레드가 실행된다.

실행결과

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

Runnable 인터페이스를 구현하는 방법

- Runnable 인터페이스를 구현한 클래스를 작성한다.
- run() 메소드를 재정의한다.

```
class MyRunnable implements Runnable {
    public void run() {
        ...
    }
}
```

- Thread 객체를 생성하고 이때 MyRunnable 객체를 인수로 전달한다.

```
Thread t = new Thread(new MyRunnable());
```

- start()를 호출하여서 스레드를 시작한다.

```
t.start();
```

MyRunnable객체 생성하여 스레드에 담아 사용

Runnable 인터페이스를 구현하는 방법

MyRunnableTest.java

```
01  class MyRunnable implements Runnable {
02      public void run() {
03          for (int i = 10; i >= 0; i--)
04              System.out.print(i + " ");
05      }
06  }
07
08  public class MyRunnableTest {
09      public static void main(String args[]) {
10          Thread t = new Thread(new MyRunnable());
11          t.start();
12      }
13  }
```

Runnable을 구현하는 클래스를 작성한다. run() 메소드를 재정의하여 작업에 필요한 코드를 넣는다.

Thread 클래스의 인스턴스를 생성하고, Runnable 객체를 Thread 생성자의 매개변수로 넘긴다. Thread 객체의 start() 메소드를 호출하여야 한다.

실행결과

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

예제#1

TestThread.java

```
01  class MyRunnable implements Runnable {
02      String myName;
03      public MyRunnable(String name) {     myName = name;     }
04      public void run() {
05          for (int i = 10; i >= 0; i--)
06              System.out.print(myName + i + " ");
07      }
08  }
09  public class TestThread {
10      public static void main(String[] args) {
11          Thread t1 = new Thread(new MyRunnable("A"));
12          Thread t2 = new Thread(new MyRunnable("B"));
13          t1.start();
14          t2.start();
15      }
16  }
```

스레드를 구분하기 위하여 이름을 설정한다.

이름이 "A"와 "B"인 스레드 2개를 생성하고 시작한다.

실행결과

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

2개의 스레드가 실행되면서 스레드의 출력이 섞이는 것을 알 수 있다.

Thread 클래스

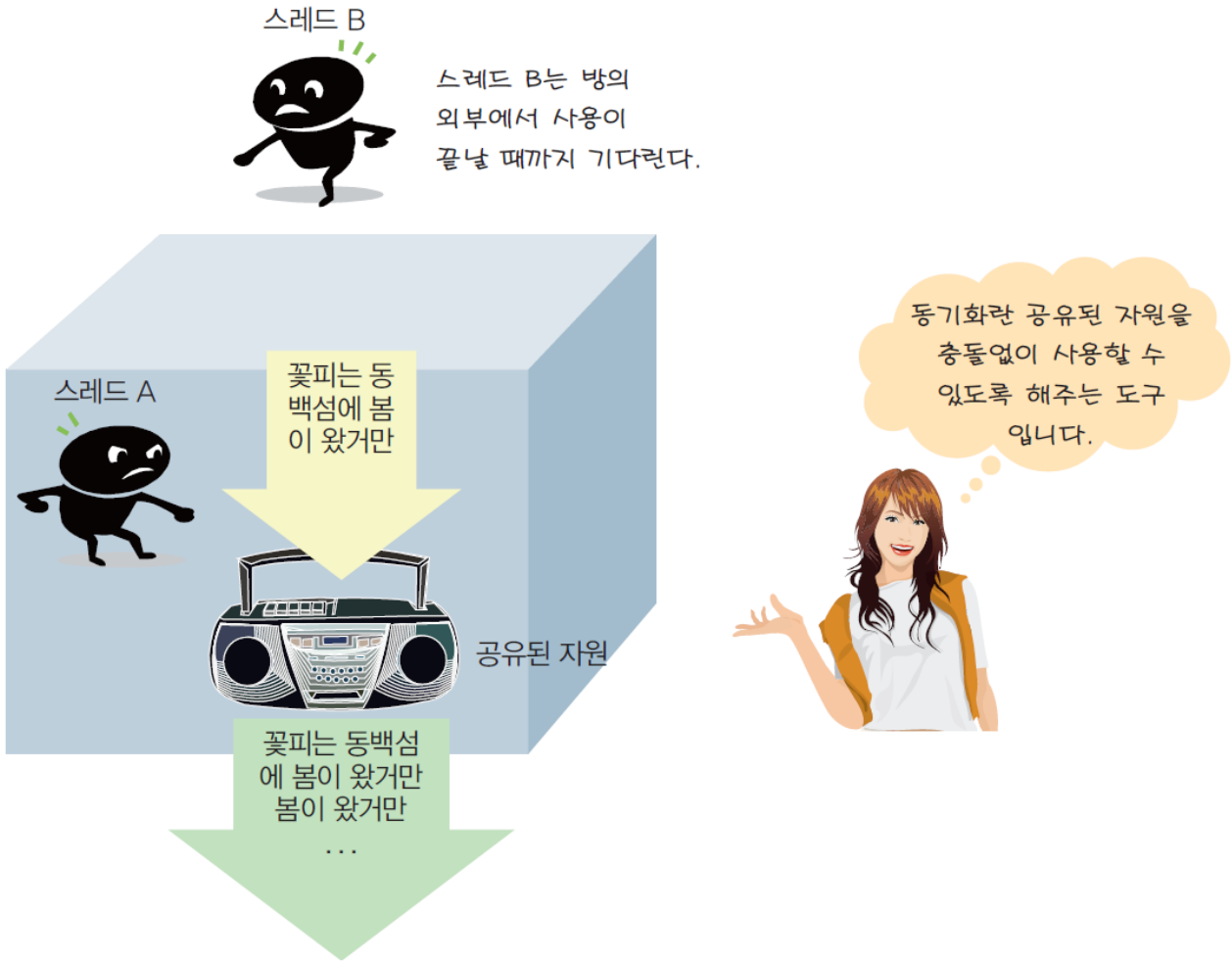
메소드	설명
Thread()	매개 변수가 없는 기본 생성자
Thread(String name)	이름이 name인 Thread 객체를 생성한다.
Thread(Runnable target, String name)	Runnable을 구현하는 객체로부터 스레드를 생성한다.
static int activeCount()	현재 활동 중인 스레드의 개수를 반환한다.
String getName()	스레드의 이름을 반환
int getPriority()	스레드의 우선순위를 반환
void interrupt()	현재의 스레드를 중단한다.
boolean isInterrupted()	현재의 스레드가 중단될 수 있는지를 검사
void setPriority(int priority)	스레드의 우선순위를 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 하여야하는 작업을 이 메소드 안에 위치시킨다.
void start()	스레드를 시작한다.
static void yield()	현재 스레드를 다른 스레드에 양보하게 만든다.

동기화

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

동기화 기법

- 동기화란 쉽게 설명하면 공유된 자원 중에서 동시에 사용하면 안 되는 자원을 보호하는 도구이다.



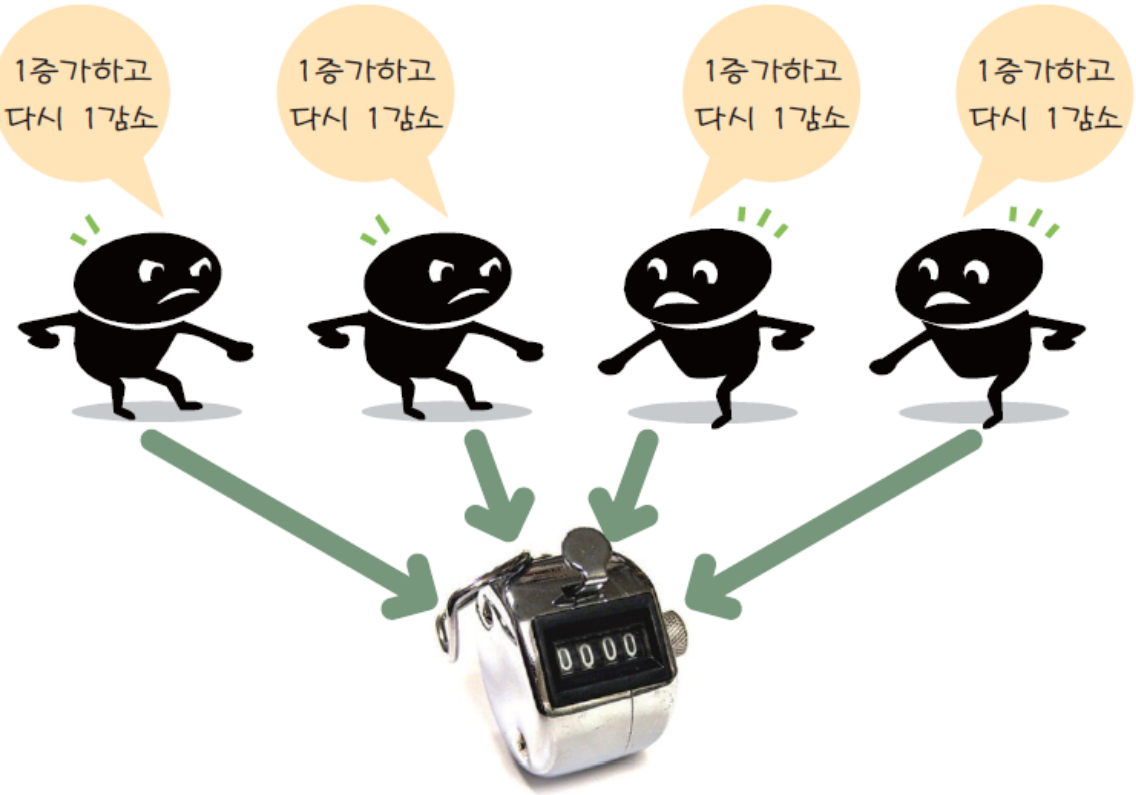
스레드 간섭

- **스레드 간섭(thread interference)**이란 서로 다른 스레드에서 실행되는 두 개의 연산이 동일한 데이터에 적용되면서 서로 겹치는 것을 의미
- (예) 카운터

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

문제 발생

- 다음과 같은 상황을 가정하자.
- 4개의 스레드가 하나의 카운터를 증가했다가 다시 감소한다.



예제

CounterTest.java

```
01  class Counter {
02     private int value = 0;
03     public void increment() {   value++;   }
04     public void decrement() {   value--;   }
05     public void printCounter() {   System.out.println(value);   }
06  }
07
08  class MyThread extends Thread {
09     Counter sharedCounter;
10
11     public MyThread(Counter c) {
12        this.sharedCounter = c;
13     }
14
15     public void run() {
16        int i = 0;
17        while (i < 20000) {
18           sharedCounter.increment();
19           sharedCounter.decrement();
20           if (i % 40 == 0)
21              sharedCounter.printCounter();
22           try {
```

Counter를 정의한다.

공유된 Counter 객체의 참조값을 저장한다.

증가했다가 감소시키기 때문에 카운터의 값은 변화가 없어야 한다.

가끔 카운터의 값을 출력하여 본다.

예제

```
23                sleep((int) (Math.random() * 2));   <-- 난수 시간만큼 스레드를 중지한다.
24             } catch (InterruptedException e) { }
25             i++;
26          }
27       }
28    }
29
30    public class CounterTest {
31       public static void main(String[] args) {
32          Counter c = new Counter();   <-- 공유 카운터 객체를 생성한다.
33          new MyThread(c).start();
34          new MyThread(c).start();
35          new MyThread(c).start();     <-- 확실하게 잘못된 결과를 내기 위하여 스레드를 4개나 생성하여 실행한다.
36          new MyThread(c).start();
37       }
38    }
```

실행결과

실행결과

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

실행결과는 컴퓨터와 상황에 따라서 상당히 달라진다. 스레드 간섭이 없다면 모두 0이 출력되어야 한다.

Week14_3.java | assign14_1a.java | Assign14_1.java | P583.java | CounterTest.java

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

class MyThread extends Thread {
    Counter sharedCounter;
    public MyThread(Counter c) {
        this.sharedCounter = c;
    }

    public void run() {
        int i = 0;
        while (i < 20) {
            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();
    }
}
```

- **synchronized**
  - -하나의 스레드가 공유 메소드를 실행하는 동안에 다른 스레드는 공유 메소드를 실행할 수 없다.
  - -하나의 스레드가 임계영역에 진입하면 다른 스레드들은 락이 풀릴때까지 기다리게 함.

Problems

<terminated> C

```
0
0
0
0
0
0
0
0
0
0
0
0
0
1
1
0
0
0
0
1
0
```

가끔 1이 나오지만 다시 0이 출력된다. 1이 가끔 나오는 이유는 하나의 스레드가 출력하기 직전에 다른 스레드가 1로 증가시켰기 때문이다. 즉 증가된 값이 출력되는 것이다. 다시 0으로 감소된다.