

자바 프로그래밍

제16장 멀티 스레딩

학습 내용

학습목차

- 01 스레드의 개요
- 02 스레드 생성과 실행
- 03 람다식을 이용한 스레드 작성
 - LAB 자동차 경주 게임 작성
- 04 스레드 활용
- 05 동기화
- 06 스레드간의 조정
 - LAB 슈팅 게임 작성하기
 - LAB 공 움직이기

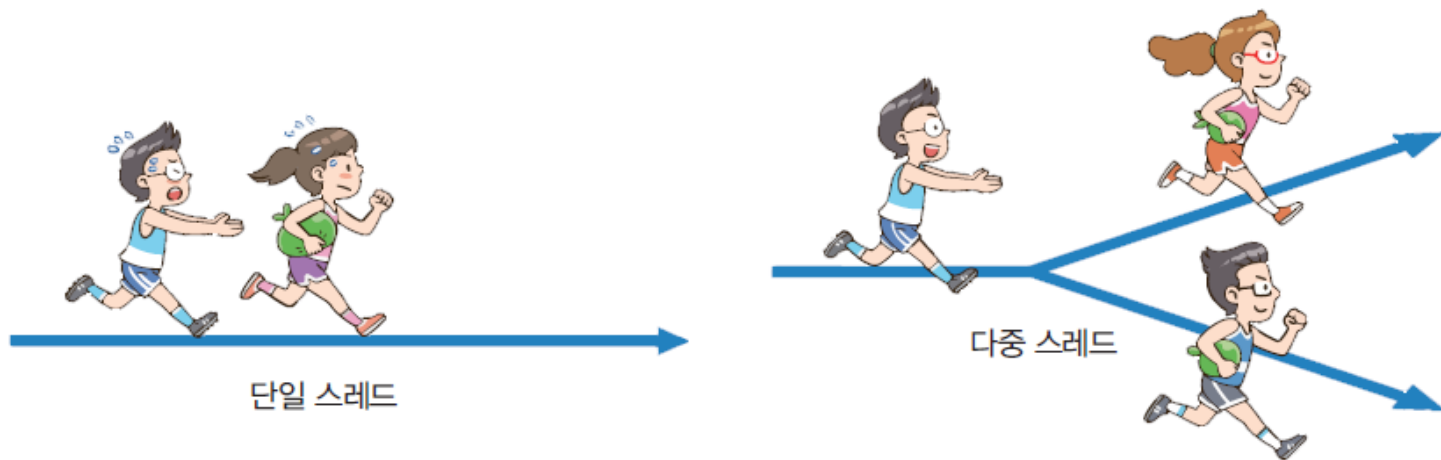
스레드는 “실”이잖아요? 자바
하고 어떤 관계가 있나요?

스레드는 CPU가 독립적으로
처리하는 하나의 작업 단위를
일컫는 용어입니다. 스레드를
사용하면 CPU를 효과적으로
사용할 수 있지요!



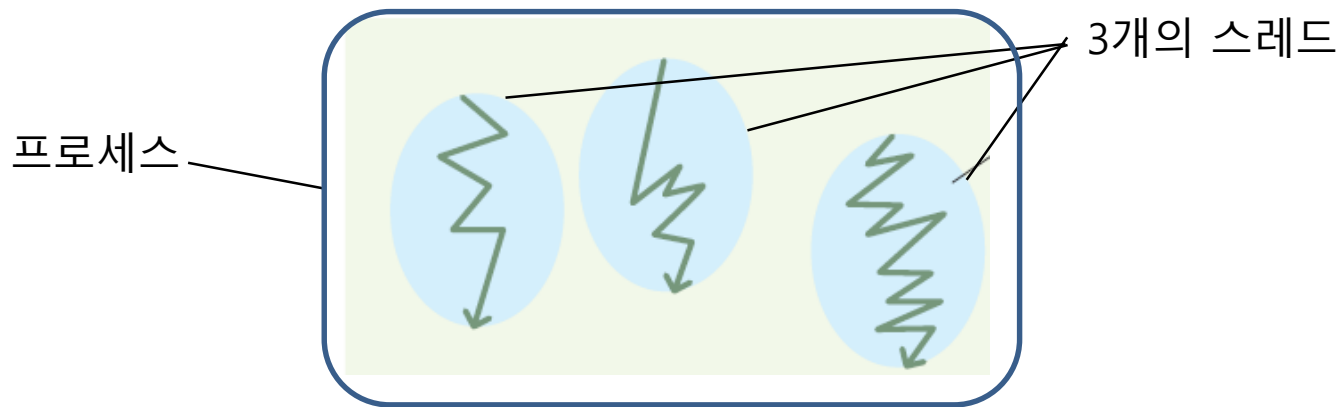
스레드란?

- 다중 스레딩(multi-threading)은 하나의 프로그램이 동시에 여러 가지 작업을 할 수 있도록 하는 것
- 각각의 작업은 스레드(thread)라고 부른다.
- 프로그래머가 여러 개의 스레드를 갖는 프로그램을 작성하여 실행하면 자바 런타임 시스템이 이 스레드들을 동시에 실행한다.



프로세스와 스레드

- 프로세스(process)
 - 실행 중인 프로그램
- 스레드(thread)
 - 프로세스 안에 존재하며, 하나의 프로세스에 속한 스레드들은 자원을 공유함
 - 이제까지 다룬 프로그램은 하나의 메인 스레드(main thread)를 갖는 프로그램이며, 메인 스레드(또는 다른 스레드)에서 추가 스레드를 생성할 수 있음



다중 스레드 프로그램

- 다중 스레딩의 예
 - 음악 재생 프로그램 : 인터넷을 통해 파일을 다운로드하면서 동시에 압축을 풀어 음악을 재생
 - 웹 브라우저 : 웹 페이지를 보면서 동시에 파일을 다운로드
 - 워드 프로세서 : 문서를 편집하면서 동시에 인쇄
 - 게임 프로그램 : 응답성을 높이기 위하여 동시에 여러 스레드를 사용
 - GUI : 마우스와 키보드 입력을 다른 스레드를 생성하여 처리
- 동시에 여러 스레드들이 실행하는 경우 잘못된 결과를 얻을 수 있음
 - 스레드들 간의 동기화(synchronization) 필요
 - 자바는 이 문제를 해결하는 도구들을 포함

스레드 생성과 실행

- 스레드는 Thread 클래스가 담당한다.

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

- 위의 코드를 실행하면 스레드가 생성되어 시작되지만, 이 스레드가 수행할 작업을 지시하지 않았으므로 할 일이 없다. 따라서 스레드는 시작 후 바로 종료한다.
- 스레드가 수행할 작업은 Thread 클래스의 run() 메소드 안에 기술한다. ➔ run() 메소드 오버라이드

Thread 클래스는 Runnable을 구현함

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

스레드 생성과 실행

```
public class ThreadTest {  
    public static void main(String[] args) {  
        Thread t = new Thread();  
        t.start();  
    }  
}
```

Thread 클래스는
java.lang 패키지에 속함

스레드를 생성하는 방법

스레드 생성 방법

1. Thread 클래스를 상속하는 방법

Thread를 상속받아 새로운 클래스를 정의 하되 run() 메소드를 오버라이드한다. 이 클래스 객체를 생성하여 실행한다.

2. Runnable 인터페이스를 구현하는 방법

Runnable을 구현하는(즉, run() 메소드를 정의) 새로운 클래스를 정의한다. 이 클래스 객체를 매개변수로 하여 Thread 객체를 생성하여 실행한다.

Thread 클래스를 상속하는 방법

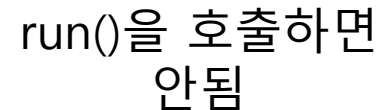
(1) Thread를 상속하는 클래스를 작성한다.

이 때 run() 메소드를 오버라이드한다.

이 클래스를 MyThread라고 하자.

(2) MyThread 객체를 생성한다.

(3) start()를 호출하여 스레드를 시작한다.



run()을 호출하면
안됨

예: Thread 클래스를 상속하는 방법

```
class MyThread extends Thread {                                // (1)
    @Override
    public void run() {
        for (int i = 10; i >= 1; i--)
            System.out.print(i + " ");
    }
}

public class MyThreadTest {
    public static void main(String[] args) {
        Thread t = new MyThread();                            // (2)
        t.start();                                              // (3)
    }
}
```

10 9 8 7 6 5 4 3 2 1

```

class MyThread extends Thread {                                // (1)
    @Override
    public void run() {
        for (int i = 10; i >= 1; i--)
            System.out.print(i + " ");
    }
}

public class MyThreadTest2 {
    public static void main(String[] args) {
        Thread t = new MyThread();                               // (2)
        t.start();                                                // (3)
        for (char ch = 'a'; ch <= 'j'; ch++)
            System.out.print(ch + " ");
    }
}

```

a 10 b 9 c 8 d 7 e f 6 g 5 h 4 i 3 j 2 1 // 순서는 실행시마다 다름

main 스레드 출력: a b c d e f g h i j
 스레드 t 출력: 10 9 8 7 6 5 4 3 2 1

```

class MyThread extends Thread {
    @Override
    public void run() {
        for (int i = 10; i >= 1; i--)
            System.out.print(i + " ");
    }
}

class MyClass {
    public void f() {
        for (char ch = 'A'; ch <= 'F'; ch++)
            System.out.print(ch + " ");
    }
}

public class MyThreadTest3 {
    public static void main(String[] args) {
        MyClass m = new MyClass();
        Thread t = new MyThread();
        t.start();
        m.f();
        for (char ch = 'G'; ch <= 'L'; ch++)
            System.out.print(ch + " ");
    }
}

```

A B C 10 D E F 9 G H I 8 7 6 5 4 3 2 1 J K L // 순서는 실행시마다 다름

main 스레드 출력: A B C D E F G H I J K L
 스레드 t 출력: 10 9 8 7 6 5 4 3 2 1

Runnable 인터페이스 구현하는 방법

(1) Runnable 인터페이스를 구현한 클래스를 작성한다.

이 때 run() 메소드를 정의한다.

이 클래스를 MyRunnable이라고 하자.

(2) Thread 객체를 생성한다.

이 때 MyRunnable 객체를 매개변수로 전달한다.

(3) start()를 호출하여 스레드를 시작한다.

run()을 호출하면
안됨

예: Runnable 인터페이스를 구현하는 방법

```
class MyRunnable implements Runnable {           // (1)
    @Override
    public void run() {
        for (int i = 10; i >= 1; i--)
            System.out.print(i + " ");
    }
}

public class MyRunnableTest {
    public static void main(String[] args) {
        Thread t = new Thread(new MyRunnable()); // (2)
        t.start();                                // (3)
    }
}
```

10 9 8 7 6 5 4 3 2 1

```

class MyRunnable implements Runnable { // (1)
    @Override
    public void run() {
        for (int i = 10; i >= 1; i--)
            System.out.print(i + " ");
    }
}

public class MyRunnableTest2 {
    public static void main(String[] args) {
        Thread t = new Thread(new MyRunnable()); // (2)
        t.start(); // (3)
        for (char ch = 'a'; ch <= 'j'; ch++)
            System.out.print(ch + " ");
    }
}

```

a 10 b 9 c 8 d 7 e f 6 g h i j 5 4 3 2 1 // 순서는 실행시마다 다름

main 스레드 출력: a b c d e f g h i j
 스레드 t 출력: 10 9 8 7 6 5 4 3 2 1

어떤 방법이 좋은가?

- Runnable 인터페이스를 사용하는 것이 더 일반적이다.
 - Thread 클래스를 상속받으면 다른 클래스를 상속받을 수 없지만, Runnable을 구현하는 클래스는 다른 클래스를 상속받을 수 있다.
 - Thread로부터 작업(Runnable)을 분리하는 설계가 가능하다.



Thread 객체 = 일꾼



Runnable 객체 = 작업의 내용

예: Runnable 구현하는 방법

```
class MyClass implements Runnable {  
    private String name;  
    public MyClass(String name) {  
        this.name = name;  
    }  
    public void run() {  
        for (int i = 5; i >= 1; i--)  
            System.out.print(name + i + " ");  
    }  
}  
  
public class ThreadTest {  
    public static void main(String[] args) {  
        Thread t1 = new Thread(new MyClass("A"));  
        Thread t2 = new Thread(new MyClass("B"));  
        t1.start();  
        t2.start();  
    }  
}
```

A5 B5 A4 A3 A2 A1 B4 B3 B2 B1

// 순서는 실행시마다 다름

람다식을 이용한 스레드 작성

```
public class ThreadTest {  
    public static void main(String[] args) {  
        Runnable task = () -> {  
            for (int i = 10; i >= 1; i--)  
                System.out.print(i + " ");  
        };  
        new Thread(task).start();  
    }  
}
```

10 9 8 7 6 5 4 3 2 1

예: 람다식을 이용한 스레드 작성

```
import java.util.concurrent.Executor;
import java.util.concurrent.Executors;

public class ThreadTest2 {
    public static void main(String[] args) {
        Runnable r1 = () -> {
            for (int i = 10; i >= 1; i--) System.out.print(i + " ");
        };
        Runnable r2 = () -> {
            for (int i = 20; i >= 11; i--) System.out.print(i + " ");
        };
        Executor executor = Executors.newCachedThreadPool();
        executor.execute(r1);
        executor.execute(r2);
    }
}
```

개발자가 스레드를 직접 생성하여 실행하지 않고,
시스템에 스레드 관리를 맡김.
스레드풀을 이용하여 스레드를 실행한다.

20 10 19 9 18 8 17 16 15 14 7 13 6 12 11 5 4 3 2 1 // 실행시마다 다름

Thread 클래스

메소드	설명
Thread()	매개 변수가 없는 기본 생성자
Thread(String <i>name</i>)	이름이 <i>name</i> 인 Thread 객체를 생성한다
Thread(Runnable <i>target</i> , String <i>name</i>)	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 <i>priority</i>)	스레드의 우선 순위를 지정한다.
void setName(String <i>name</i>)	스레드의 이름을 지정한다.
static void sleep(int <i>milliseconds</i>)	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.

sleep() 메소드

```
public class Main {  
    public static void main(String[] args) throws InterruptedException {  
        for (int i = 10; i >= 1; i--) {  
            Thread.sleep(1000); // 1000 밀리초 동안 멈춤  
            System.out.print(i + " ");  
        }  
    }  
}
```

public static void sleep(long millis) throws [InterruptedException](#)

10 9 8 7 6 5 4 3 2 1

인터럽트(interrupt)

- 스레드 실행을 중지하도록 하는 메커니즘
 - 스레드 t의 실행을 중지하려면 t.interrupt(); 를 실행
- 인터럽트를 받은 스레드는 어떤 반응을 보일 것인가

```
for (int i = 10; i >= 1; i--) {  
    try {  
        Thread.sleep(1000);  
    } catch (InterruptedException e) {  
        return;  
    }  
    System.out.print(i + " ");  
}
```

인터럽트 처리는
여기서 해준다.

- sleep()을 호출하지 않으면 InterruptedException을 받지 못함
 - 무한루프 중간에 다음과 같은 인터럽트 검사해주는 것이 좋다.

```
if (Thread.interrupted()) { // 인터럽트를 받으면  
    return;                // 단순히 리턴  
}
```

조인(join)

- join() 메소드는 하나의 스레드가 다른 스레드의 종료를 기다리는 메소드이다.
 - 스레드 t가 종료될 때까지 기다리려면 다음 문장을 실행
t.join();
 - 스레드 t가 종료될 때까지 1000밀리초 동안 기다리려면 다음 문장을 실행
t.join(1000);

강제적인 종료

```
public class ThreadControl {
    private static void print(String message) { // 스레드 이름과 함께 메시지를 출력
        String threadName = Thread.currentThread().getName();
        System.out.format("%s: %s%n", threadName, message);
    }

    private static class MessageLoop implements Runnable {
        public void run() {
            String[] messages = {"apple", "basket", "candy", "dog"};
            try {
                for (int i = 0; i < messages.length; i++) {
                    print(messages[i]);
                    Thread.sleep(2000);
                }
            } catch (InterruptedException e) {
                print("아직 끝나지 않았어요!"); // 인터럽트되면 메시지를 출력
            }
        }
    }

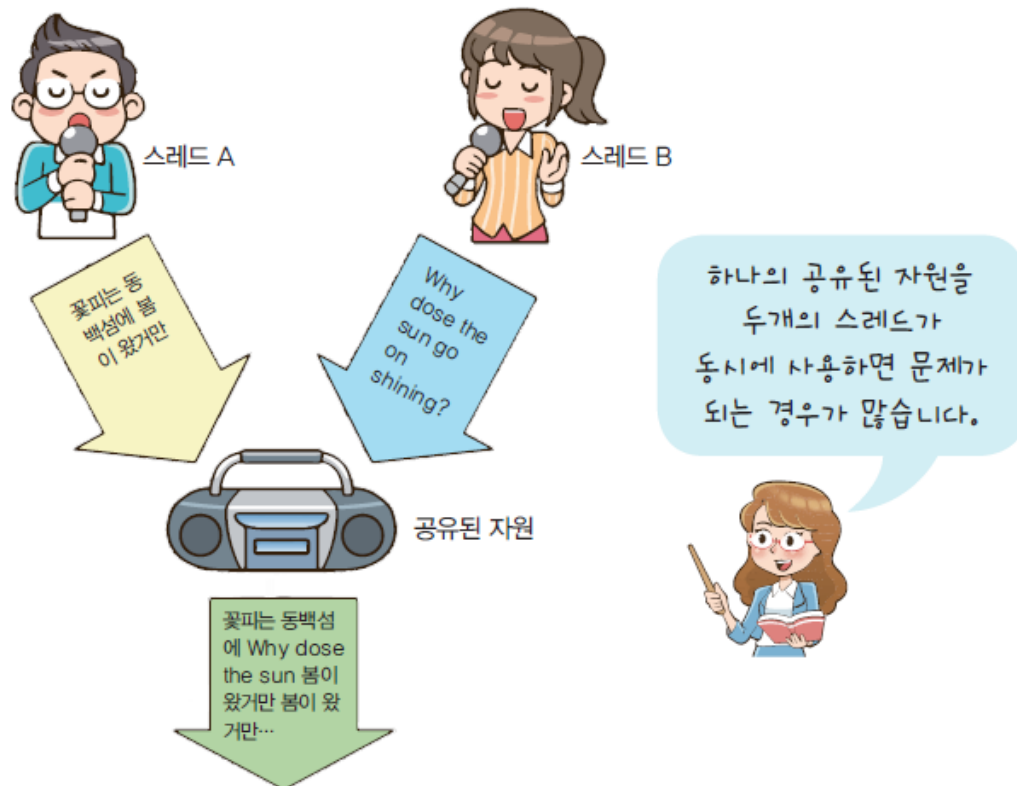
    public static void main(String[] args) throws InterruptedException {
        print("시작합니다");
        int tries = 0;
        print("추가 스레드를 시작합니다.");
        Thread t = new Thread(new MessageLoop());
        t.start();
        while (t.isAlive()) {
            print("추가 스레드가 끝나기를 기다립니다.");
            t.join(1000); // 스레드 t가 종료하기를 1초동안 기다림
            if (++tries > 2) {
                print("참을 수 없네요!");
                t.interrupt(); // 스레드 t를 인터럽트
                t.join(); // 스레드 t가 종료하기를 기다림
            }
        }
        print("메인 스레드 종료!");
    }
}
```

실행결과

```
main: 시작합니다
main: 추가 스레드를 시작합니다.
main: 추가 스레드가 끝나기를 기다립니다.
Thread-0: apple
main: 추가 스레드가 끝나기를 기다립니다.
Thread-0: basket
main: 추가 스레드가 끝나기를 기다립니다.
main: 참을 수 없네요!
Thread-0: 아직 끝나지 않았어요!
main: 메인 스레드 종료!
```


동기화(synchronization)

- 한 번에 하나의 스레드 만이 공유 데이터를 접근할 수 있도록 제어하는 것이 필요



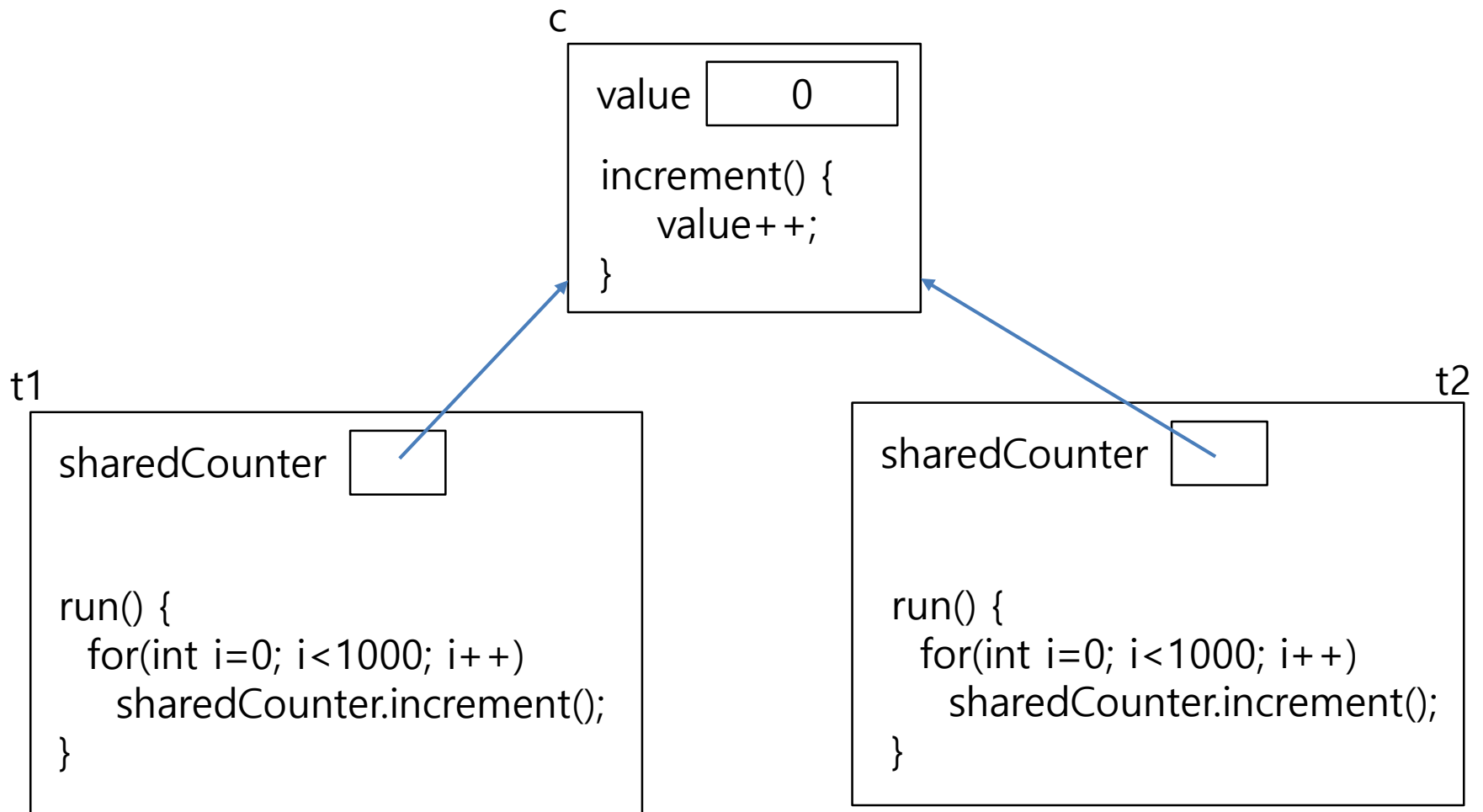
스레드 간섭

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

class MyThread extends Thread {
    private Counter sharedCounter;
    public MyThread(Counter c) {
        this.sharedCounter = c;
    }
    public void run() {
        for(int i = 0; i < 1000; i++) // 공유 카운터를 1000번 증가시킴
            sharedCounter.increment();
    }
}

public class Main {
    public static void main(String[] args) throws InterruptedException {
        Counter c = new Counter();
        Thread t1 = new MyThread(c);
        Thread t2 = new MyThread(c);
        t1.start();
        t2.start();
        t1.join();
        t2.join();
        c.printCounter(); // 2000이 출력될 것으로 기대함
    }
}
```

1608



동기화된 메소드

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

class MyThread extends Thread {
    private Counter sharedCounter;
    public MyThread(Counter c) {
        this.sharedCounter = c;
    }
    public void run() {
        for(int i = 0; i < 1000; i++) // 공유 카운터를 1000번 증가시킴
            sharedCounter.increment();
    }
}

public class Main {
    public static void main(String[] args) throws InterruptedException {
        Counter c = new Counter();
        Thread t1 = new MyThread(c);
        Thread t2 = new MyThread(c);
        t1.start();
        t2.start();
        t1.join();
        t2.join();
        c.printCounter(); // 2000이 출력됨
    }
}
```

2000