19장 다중 스레드

• 학습 목표

- 스레드의 개념과 스레드 생명주기에 관해 학습한다.
- 스레드 생성 방법에 관해 학습한다.
- 스레드의 우선순위에 관해 학습한다.
- 다중 스레드의 시작과 종료에 관해 학습한다.
- 다중 스레드의 사용 방법에 관해 학습한다.
- 스레드 사이의 통신 방법에 관해 학습한다.

1-1 프로그램과 프로세스

- 프로세스 : 실행중인 프로그램
- 다중 프로세스 : 다수 개의 프로세스를 다수 개의 CPU가 동시에 처리

그림 16-1 실행중인 프로세스

1-2 프로세스와 스레드

- 스레드는 하나의 프로세스 내에서 동작하는 작은(lightweight) 프로세스
- 다중 스레드(multi thread)는 하나의 프로세스(프로그램) 내에서 다수 개의 스레드가 동시에 동작하는 개념

1-2 프로세스와 스레드

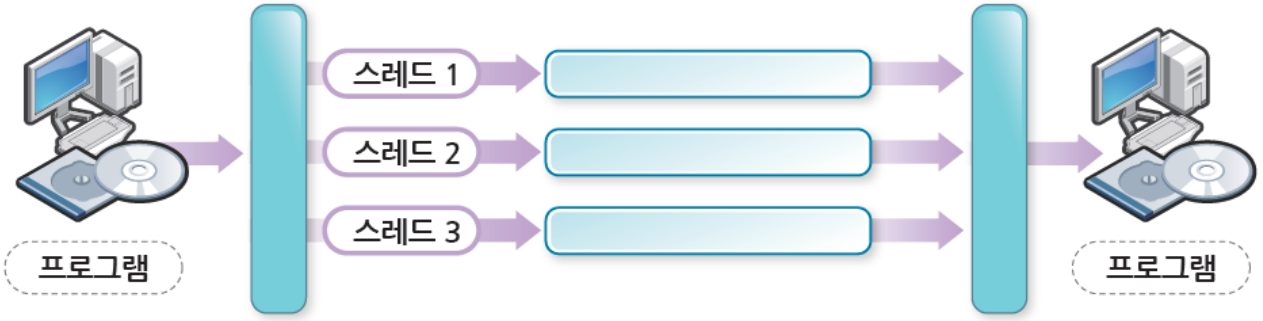

그림 16-2 다수 개의 CPU를 가진 시스템에서 다중 스레드의 실행

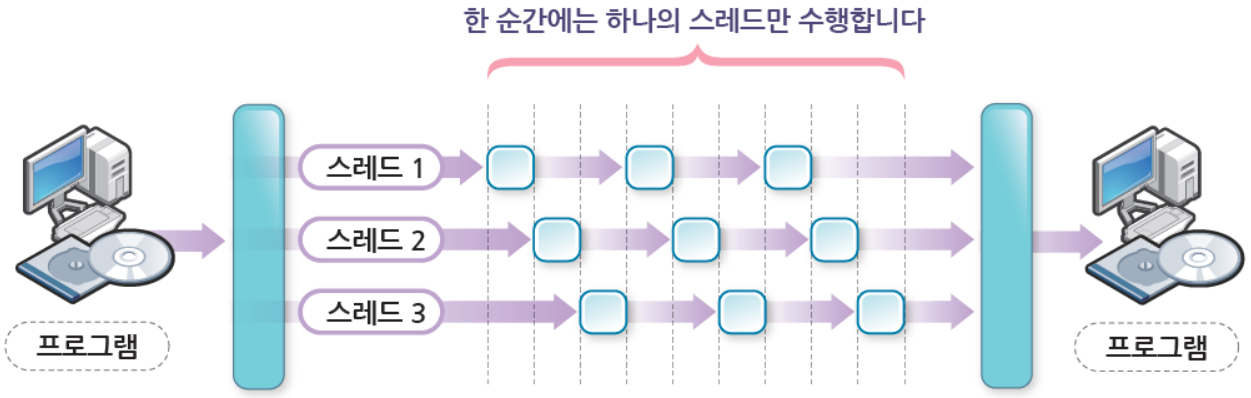

그림 16-3 하나의 CPU를 가진 시스템에서 다중 스레드의 실행

Thread 클래스

- java.lang 패키지에 라이브러리 클래스로 제공

[형식] Thread 클래스의 생성자

```
Thread()
Thread(String s)
Thread(Runnable r)
Thread(Runnable r, String s)
```

r : Runnable 인터페이스 객체
s : 스레드를 구별할 수 있는 이름

2 Thread 클래스와 스레드 생명주기

표 16-1 스레드 클래스의 메소드

메소드	이름 설명
static void sleep(long msec) throws InterruptedException	msec에 지정된 밀리초(milliseconds) 동안 대기
static void sleep(long msec, int nsec) throws InterruptedException	msec에 지정된 밀리 초 +nsec에 지정된 나노초(nanosec onds) 동안 대기
String getName()	스레드의 이름을 반환
void setName(String s)	스레드의 이름을 s로 설정
void start()	스레드를 시작시킨다. run() 메소드를 호출
final int getPriority()	스레드의 우선순위를 반환
final void setPriority(int p)	스레드의 우선순위를 p 값으로 설정
boolean isAlive()	스레드가 실행 가능 상태, 실행상태, 대기상태에 있으면 true를 그렇지 않으면 false를 반환
void join() throws InterruptedException	스레드가 끝날 때까지 대기
void run()	스레드가 실행할 부분을 기술하는 메소드. 하위 클래스에서 오버라이딩되어야 합니다.
void suspend()	스레드가 일시 정지됩니다. resume()에 의해 다시 시작될 수 있습니다.
void resume()	일시 정지된 스레드를 다시 시작시킨다.

2 Thread 클래스와 스레드 생명주기

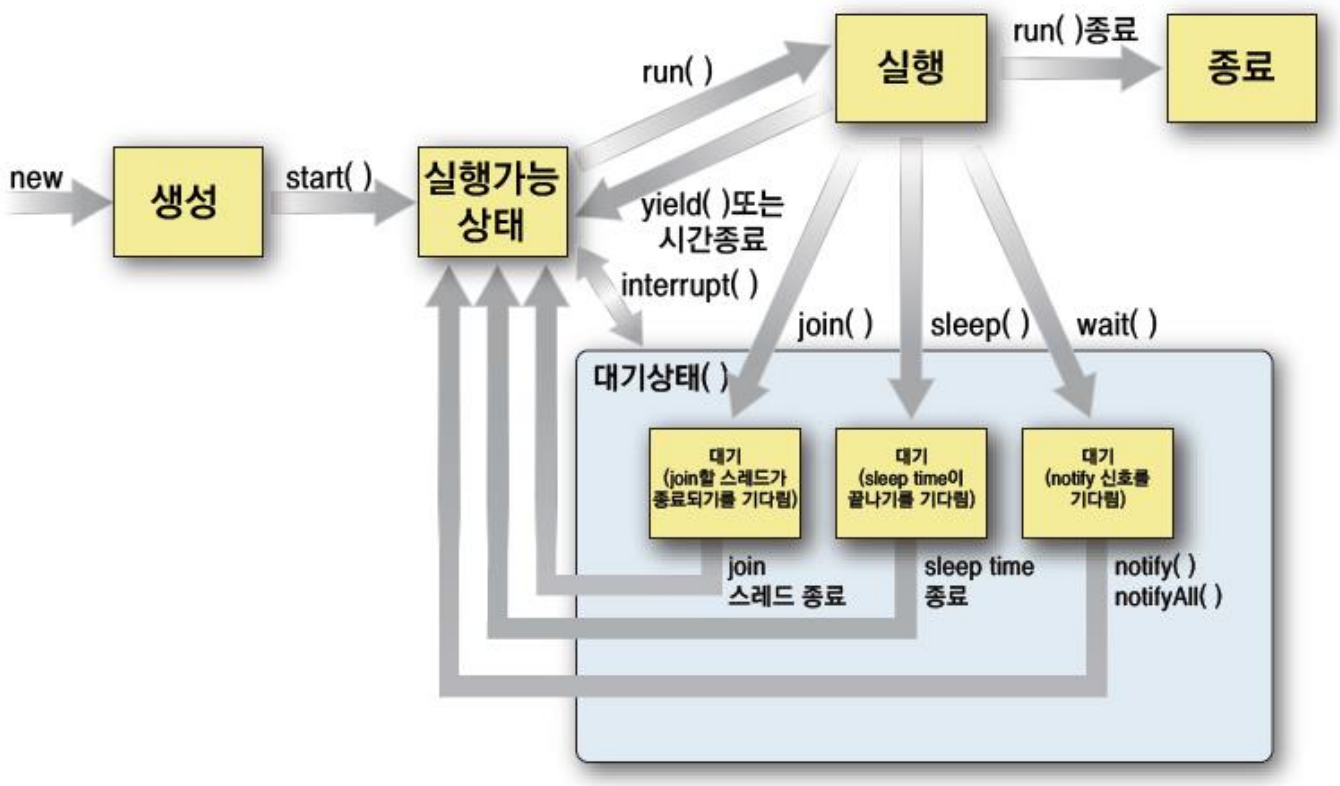

그림 16-4 스레드의 생명주기

- 스레드는 생성되어 종료될 때까지 5가지의 상태
  - 생성 상태
  - 실행 가능 상태
  - 실행 상태
  - 대기상태
  - 종료

3-1 Thread 클래스 이용

- **Thread(java.lang.Thread) 클래스로부터 직접 상속받아 스레드 특징을 가지는 클래스를 생성하여 스레드를 사용하는 방법**

```
class ThreadA extends Thread {
    .........
    public void run() {
        ....
    }
    .......
}
```

Thread 클래스로부터 상속

상위 클래스인 Thread 클래스의 run() 메소드를 오버라이딩하여 스레드가 수행하여야 하는 문장들을 기술한다.

```
ThreadA ta = new ThreadA();
ta.start();
```

3-2 Runnable 인터페이스 이용

- 스레드의 특성을 가져야 하는 클래스가 이미 다른 클래스로부터 상속을 받고 있다면 Runnable 인터페이스를 이용

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

```
class RunnableB extends Applet implements Runnable {
    .........
    public void run() {
        ........
    }
    ........
}
```

Runnable 인터페이스에 정의된 run() 메소드를 오버라이딩하여 스레드가 수행할 문장들을 기술한다.

3-2 Runnable 인터페이스 이용

- 인터페이스를 이용하여 생성된 클래스로부터 스레드 객체를 생성하는 예

```
RunnableB rb = new RunnableB();   ——— 객체 rb 생성
Thread tb = new Thread(rb);       ——— rb를 매개변수로 하여 스레드 객체 tb를 생성
tb.start();                       ——— 스레드 시작
```

```
RunnableB rb = new RunnableB();   ——— 스레드 객체를 생성하여 바로 시작
new Thread(rb).start();
```

3-2 Runnable 인터페이스 이용

예제 16.1 ThreadFromThread.java

```
01: class ThreadTest extends Thread {
02:     public void run() {
03:         for (int i=1 ; i<=10 ; i++) {
04:             System.out.println("재미있는 자바 :" + i);
05:         }
06:     }
07: }
08: public class ThreadFromThread {
09:     public static void main(String args[]) {
10:         ThreadTest t = new ThreadTest();
11:         t.start();
12:     }
13: }
```

Thread 클래스로부터 상속받아 클래스 작성

Thread 클래스로부터 상속받아 클래스 작성

스레드 특성을 가진 객체 생성

스레드 시작(run() 메소드 호출)

실행 결과

```
재미있는 자바 :1
재미있는 자바 :2
재미있는 자바 :3
재미있는 자바 :4
재미있는 자바 :5
재미있는 자바 :6
재미있는 자바 :7
재미있는 자바 :8
재미있는 자바 :9
재미있는 자바 :10
```

3-2 Runnable 인터페이스 이용

예제 16.2 ThreadFromRunnable.java

```
01: class RunnableTest implements Runnable {
02:     public void run() {
03:         for (int i=1 ; i<=10 ; i++) {
04:             System.out.println("재미있는 자바 :" + i);
05:         }
06:     }
07: }
08: public class ThreadFromRunnable {
09:     public static void main(String args[]) {
10:         RunnableTest r = new RunnableTest();
11:         Thread t = new Thread(r);
12:         t.start();
13:     }
14: }
```

Runnable 인터페이스를 포함하는 클래스 작성

run() 메소드 오버라이딩

객체 생성

생성된 객체를 이용하여 스레드 객체 생성

스레드 시작(run() 메소드 호출)

실행 결과

앞 프로그램의 결과와 같습니다.

3-2 Runnable 인터페이스 이용

예제 16.3 DoubleThread.java

```
01: class ThreadTest1 extends Thread {          ← Thread 클래스로부터 상속받아 클래스 작성
02:    public ThreadTest1(String str) {
03:       setName(str);                         ← 스레드의 이름을 설정
04:    }
05:    public void run() {
06:       for (int i=1 ; i<=10 ; i++) {
07:          System.out.println(i + getName());  ← 번호와 스레드 이름을 출력
08:       }
09:       System.out.println("끝" + getName());
10:    }
11: }
12: public class DoubleThread {
13:    public static void main(String args[]) {
14:       ThreadTest1 t1 = new ThreadTest1       ←
15:          (" : 배우기 쉬운 자바");              |  이름이 다른 두 개의 스레드 객체 생성
16:       ThreadTest1 t2 = new ThreadTest1       ←
17:         (" : 배우기 어려운 자바");
18:       t1.start();                           ← 스레드 동시 실행
19:       t2.start();                           ← 스레드 동시 실행
20:    }
21: }
```

실행 결과

이 프로그램은 실행시킬 때마다 서로 다른 결과를 출력합니다. 실행될 때마다 CPU의 실행 환경이 다르기 때문입니다.

```
1 : 배우기 어려운 자바
1 : 배우기 쉬운 자바
2 : 배우기 어려운 자바
2 : 배우기 쉬운 자바
3 : 배우기 어려운 자바
3 : 배우기 쉬운 자바
4 : 배우기 어려운 자바
4 : 배우기 쉬운 자바
5 : 배우기 어려운 자바
5 : 배우기 쉬운 자바
6 : 배우기 어려운 자바
6 : 배우기 쉬운 자바
7 : 배우기 어려운 자바
7 : 배우기 쉬운 자바
8 : 배우기 어려운 자바
8 : 배우기 쉬운 자바
9 : 배우기 어려운 자바
9 : 배우기 쉬운 자바
10 : 배우기 어려운 자바
10 : 배우기 쉬운 자바
끝 : 배우기 어려운 자바
끝 : 배우기 쉬운 자바
```

- **스레드에 우선순위를 부여하여 우선순위가 높은 스레드에 실행의 우선권을 주어 실행**

- **setPriority()**
  - Thread 클래스에는 스레드에 우선순위를 부여하는 메소드가 제공

```
static final int MAX_PRIORITY     ← 우선순위 값으로 10을 가진다.
static final int MIN_PRIORITY     ← 우선순위 값으로 1을 가진다.
static final int NORM_PRIORITY    ← 우선순위 값으로 5를 가진다.
```

4 스레드 우선순위

예제 16.4 ThreadPriority.java

```
04: class PriorityTest extends Thread{
05:    public PriorityTest(String str){
06:       setName(str);
07:    }
08:    public void run(){
09:       for(int i=1; i<=5; i++){
10:          System.out.println( i + getName() + " 우선순위 : "
11:                              + getPriority() );
12:       }
13:    }
14: }
15: class ThreadPriority {
16:    public static void main(String args[]){
17:       PriorityTest t1 = new PriorityTest(" : 첫번째 스레드");
18:       PriorityTest t2 = new PriorityTest(" : 두번째 스레드");
19:       PriorityTest t3 = new PriorityTest(" : 세번째 스레드");
20:       int priority_t1 = Integer.parseInt(args[0]);
21:       int priority_t2 = Integer.parseInt(args[1]);
22:       t1.setPriority(priority_t1);
23:       t2.setPriority(priority_t2);
24:       t3.setPriority(Thread.MIN_PRIORITY);
25:       t1.start();
26:       t2.start();
27:       t3.start();
28:    }
29: }
```

getPriority() 메소드로 우선순위 출력

우선순위 설정

실행 결과

[인자값 2개 있어야 함. 실행시 5, 10값 넣고 실행한 결과]

```
1 : 세번째 스레드  우선순위 : 1
1 : 첫번째 스레드  우선순위 : 5
1 : 두번째 스레드  우선순위 : 10
2 : 두번째 스레드  우선순위 : 10
3 : 두번째 스레드  우선순위 : 10
4 : 두번째 스레드  우선순위 : 10
2 : 첫번째 스레드  우선순위 : 5
3 : 첫번째 스레드  우선순위 : 5
4 : 첫번째 스레드  우선순위 : 5
5 : 첫번째 스레드  우선순위 : 5
2 : 세번째 스레드  우선순위 : 1
5 : 두번째 스레드  우선순위 : 10
3 : 세번째 스레드  우선순위 : 1
4 : 세번째 스레드  우선순위 : 1
5 : 세번째 스레드  우선순위 : 1
```

● 다중 스레드를 가진 프로그램의 실행 흐름

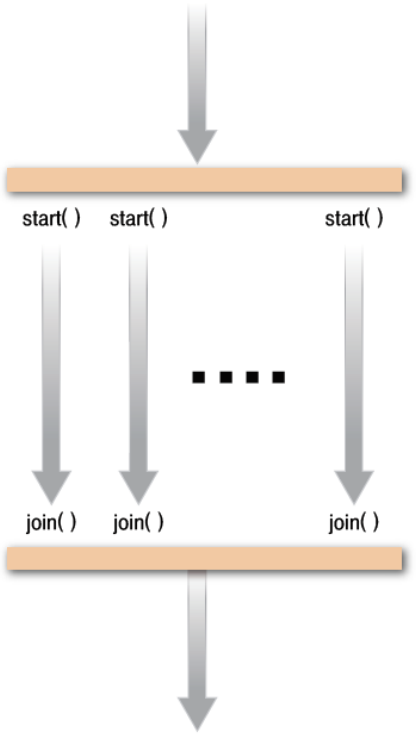

그림 16-5 다중 스레드를 가진 프로그램의 실행 흐름

5 스레드의 시작과 종료

예제 16.5 DoubleThread1.java

```
01: class DoubleThreadTest1 extends Thread {
02:    public DoubleThreadTest1(String str) {
03:       setName(str);
04:    }
05:    public void run() {
06:       for (int i=1 ; i<=3 ; i++) {
07:          System.out.println(i + getName());
08:       }
09:       System.out.println("끝" + getName());
10:    }
11: }
12: public class DoubleThread1 {
13:    public static void main(String args[]) {
14:       DoubleThreadTest1 t1 =
15:          new DoubleThreadTest1(" : 배우기 쉬운 자바");
16:       DoubleThreadTest1 t2 =
17:          new DoubleThreadTest1(" : 배우기 어려운 자바");
18:       System.out.println("***** 스레드 시작 전 *****");
19:       t1.start();
20:       t2.start();
21:       System.out.println("***** 스레드 종료 후 *****"); <---------- 스레드 시작 전에 출력
22:    }
23: }
```

실행 결과

```
***** 스레드 시작 전 *****
***** 스레드 종료 후 *****
1 : 배우기 어려운 자바
1 : 배우기 쉬운 자바
2 : 배우기 어려운 자바
2 : 배우기 쉬운 자바
3 : 배우기 어려운 자바
3 : 배우기 쉬운 자바
끝 : 배우기 어려운 자바
끝 : 배우기 쉬운 자바
```

5 스레드의 시작과 종료

예제 16.6 DoubleThread2.java

```
01: class DoubleThreadTest2 extends Thread {
02:     public DoubleThreadTest2(String str) {
03:         setName(str);
04:     }
05:     public void run() {
06:       for (int i=1 ; i<=3 ; i++) {
07:           System.out.println(i + getName());
08:       }
09:       System.out.println("끝" + getName());
10:     }
11: }
12: public class DoubleThread2 {
13:     public static void main(String args[])throws Exception {   <-- join() 메소드 사용을 위해 예외 처리
14:         DoubleThreadTest2 t1 = new DoubleThreadTest2
15:             (" : 배우기 쉬운 자바");
16:         DoubleThreadTest2 t2 = new DoubleThreadTest2
17:             (" : 배우기 어려운 자바");
18:         System.out.println("***** 스레드 시작 전 *****");
19:         t1.start();
20:         t2.start();
21:         t1.join();   <-- join() 메소드 수행
22:         t2.join();   <--
23:         System.out.println("***** 스레드 종료 후 *****");   <-- 스레드 종료 후에 실행
24:     }
25: }
```

실행 결과

```
***** 스레드 시작 전 *****
1 : 배우기 쉬운 자바
1 : 배우기 어려운 자바
2 : 배우기 쉬운 자바
2 : 배우기 어려운 자바
3 : 배우기 쉬운 자바
3 : 배우기 어려운 자바
끝 : 배우기 쉬운 자바
끝 : 배우기 어려운 자바
***** 스레드 종료 후 *****
```

- **임계영역(critical section)**
  - 다수 개의 스레드가 접근 가능한 영역
  - 한순간에는 하나의 스레드만 사용할 수 있는 영역

- **자바에서는 임계영역 지정을 위한 synchronized 메소드를 제공**

● 다중 스레드 환경에서의 synchronized 메소드 실행

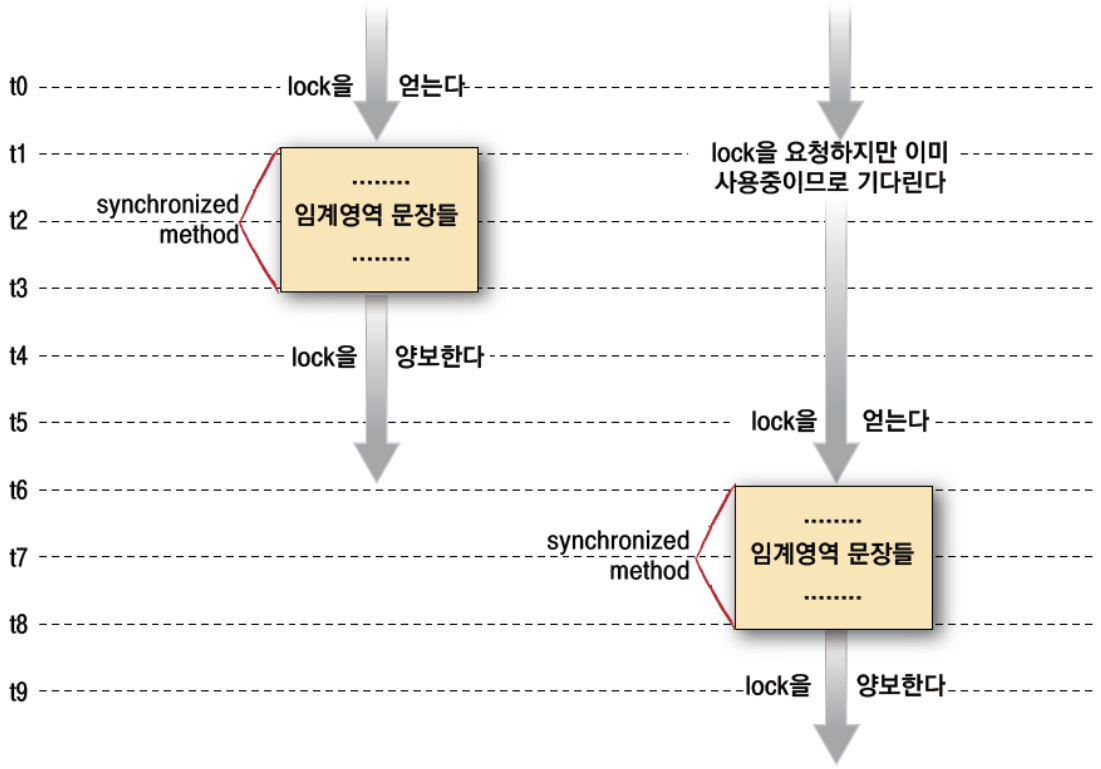

그림 16-6 다중 스레드 환경에서의 synchronized 메소드 실행

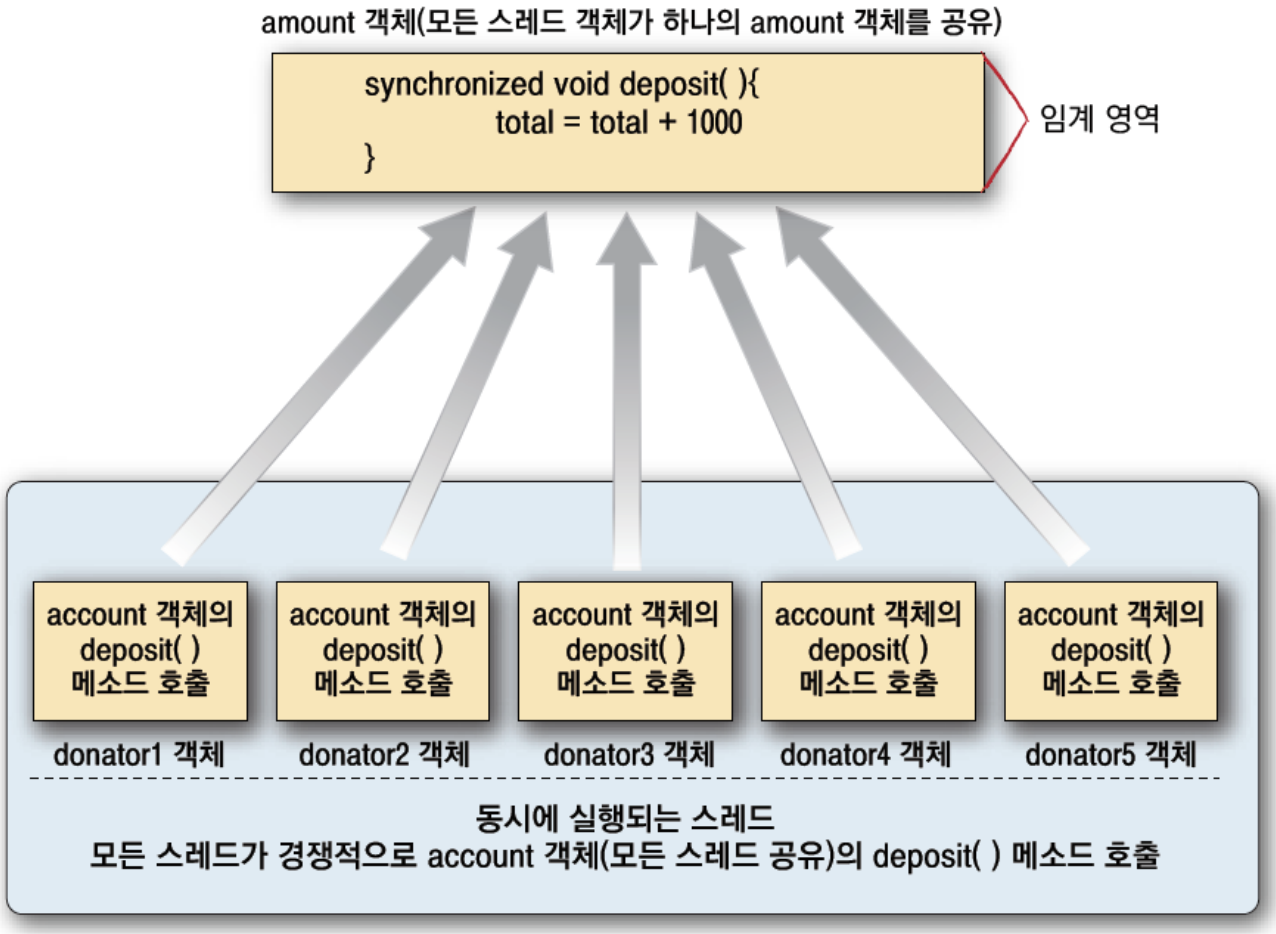

그림 16–7 프로그램의 작동 구조

6 스레드 동기화(Synchronization)

예제 16.7 TVContribution.java

```
01: class Account {
02:    private int total = 0;
03:    synchronized void deposit() {          <-- 1000원을 더하는 동기화 메소드 선언
04:       total = total + 1000;
05:    }
06:    int gettotal() {
07:       return total;
08:    }
09: }
10: class Customer extends Thread {
11:    Account same_a;
12:    Customer(Account a, String s) {
13:       same_a = a;                          <-- 스레드 이름과 Account 객체를 설정
14:       setName(s);
15:    }
16:    public void run() {
17:       for(int i = 1; i <= 200 ; i++) {
18:          System.out.println(getName() + " : " + i +"번째");
19:          same_a.deposit();                 <-- 반복문을 돌며 deposit() 메소드 호출
20:          if (same_a.gettotal() >= 500000) break;   <-- 전체금액이 50만원 보다 크면 반복문을 벗어남
21:       }
22:    }
23: }
```

6 스레드 동기화(Synchronization)

```
24: public class TVContribution {
25:    public static void main(String args[])throws Exception {
26:       Account same_account = new Account();
27:       Customer donator1 =
28:          new Customer(same_account,"1번째 성금자");
29:       Customer donator2 =
30:          new Customer(same_account,"2번째 성금자");
31:       Customer donator3 =
32:          new Customer(same_account,"3번째 성금자");
33:       Customer donator4 =
34:          new Customer(same_account,"4번째 성금자");
35:       Customer donator5 =
36:          new Customer(same_account,"5번째 성금자");
37:       donator1.start();
38:       donator2.start();
39:       donator3.start();
40:       donator4.start();
41:       donator5.start();
42:       donator1.join();
43:       donator2.join();
44:       donator3.join();
45:       donator4.join();
46:       donator5.join();
47:       System.out.println("성금 총액은 : " +
48:          same_account.gettotal());
49:    }
50: }
```

Account 클래스로부터 객체 생성

생성자에서 동일한 객체를 지정함 (5개의 스레드가 same_account 객체 공유)

실행 결과

[첫 번째 실행결과]

3번째 성금자 : 1번째
3번째 성금자 : 1번째
1번째 성금자 : 1번째
4번째 성금자 : 1번째
.....중간생략
5번째 성금자 : 62번째
3번째 성금자 : 124번째
1번째 성금자 : 126번째
4번째 성금자 : 125번째
2번째 성금자 : 62번째
성금 총액은 : 504000

[두 번째 실행결과]

1번째 성금자 : 1번째
1번째 성금자 : 2번째
1번째 성금자 : 3번째
3번째 성금자 : 1번째
.....중간생략
2번째 성금자 : 42번째
3번째 성금자 : 102번째
5번째 성금자 : 92번째
1번째 성금자 : 62번째
2번째 성금자 : 43번째
성금 총액은 : 503000

- 스레드 사이의 통신을 위한 메소드를 java.lang.Object 클래스에 제공

- Object 클래스에서 제공되는 메소드 중 wait(), notify(), notifyAll() 메소드가 스레드 사이의 통신에 이용

- wait() 메소드의 3가지 형태

```
void wait() throws InterruptedException
void wait(long msec) throws InterruptedException
void wait(long msec, int nsec) throws InterruptedException
```

- wait() : 무한정 대기 상태에 들어가는 메소드
- wait(long msec) : msec 밀리초 동안 대기하다 스스로 깨어나는 메소드
- wait(long msec, int nsec) : msec 밀리초와 nsec 나노초 동안 대기하다 스스로 깨어나는 메소드

- notify() : 대기 중인 스레드가 여러 개일 경우 그중에 하나의 스레드만을 깨운
- notifyAll() 메소드 : 대기 중인 모든 스레드를 깨우는 것을 제외하고는 notify()와 같음

```
void notify()
void notifyAll()
```

- wait( ) 메소드와 notify( ), notifyAll( ) 메소드의 작동

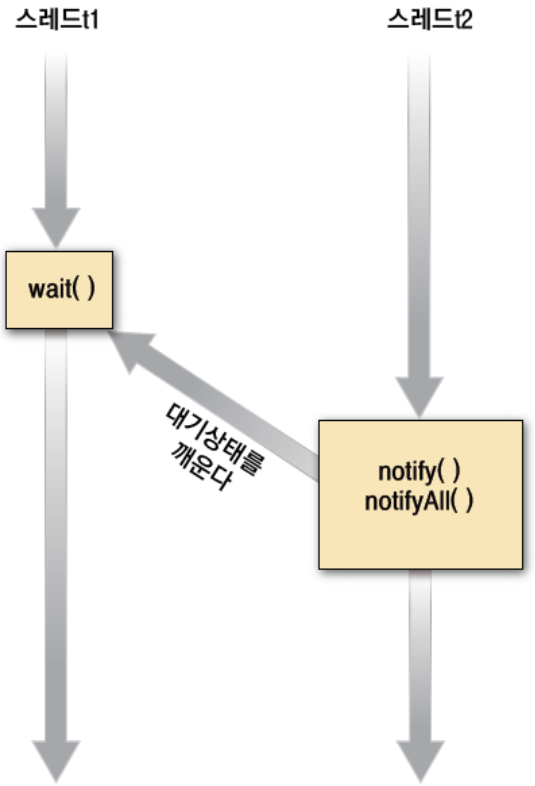

그림 16-8 wait( ) 메소드와 notify( ), notifyAll( ) 메소드의 작동

예제 16.8 ProducerConsumer.java

```
01: class Buffer {
02:    private int contents;
03:    private boolean available = false;
04:    public synchronized void put(int value) {
05:       while (available == true ) {
06:          try{
07:             wait();
08:          }
09:          catch (InterruptedException e) {}
10:       }
11:       contents = value;
12:       System.out.println
13:          ("생산자########## : 생산 " + contents);
14:       notify();
15:       available = true;
16:    }
17:    public synchronized int get()  {
18:       while (available == false ) {
19:          try {
20:             wait();
21:          }
22:          catch (InterruptedException e) {}
23:       }
24:       System.out.println
25:          ("소비자########## : 소비 " + contents);
26:       notify();
27:       available = false;
28:       return contents;
29:    }
30: }
31: class Producer extends Thread {
32:    private Buffer b;
33:    public Producer(Buffer blank) {
34:       b = blank ;
35:    }
36:    public void run() {
37:    for (int i = 1; i <= 10;i++)
38:       b.put(i);
39:    }
40: }
41: class Consumer extends Thread {        <-- 소비자 스레드 클래스
42:    private Buffer b;
43:    public Consumer(Buffer blank) {
44:       b = blank;
45:    }
46:    public void run() {
47:       int value = 0;
48:       for (int i = 1 ; i <= 10 ; i++ )
49:          value = b.get();
50:    }
51: }
52: public class ProducerConsumer {
53:    public static void main(String args[]) {
54:       Buffer buff = new Buffer();        <-- 하나의 Buffer 객체 생성
55:       Producer p1 = new Producer(buff);  <-- 동일한 Buffer 객체를 사용하여 스레드 객체 생성
56:       Consumer c1 = new Consumer(buff);  <--
57:       p1.start() ;
58:       c1.start() ;
59:    }
60: }
```

실행 결과

```
생산자########## : 생산 1
소비자########## : 소비 1
생산자########## : 생산 2
.....중간생략
소비자########## : 소비 7
생산자########## : 생산 8
소비자########## : 소비 8
생산자########## : 생산 9
소비자########## : 소비 9
```

학습 정리

● 스레드 개요

① 스레드는 실행 중인 프로세스라 할 수 있으며, 하나의 프로그램에 다수 개의 스레드를 실행시킬 수 있습니다.

② 한 개의 CPU를 가진 컴퓨터에서 스레드는 CPU에 의해 돌아가며 수행되게 됩니다.

● Thread 클래스와 스레드 생명주기

① 자바에서는 스레드를 지원하기 위해 Thread 클래스를 제공하고 있습니다.

② 스레드 클래스는 스레드를 지원하는 다양한 메소드를 가지고 있습니다.

③ 스레드는 생성, 실행 가능 상태, 실행 상태, 대기 상태 등의 생명주기를 가지고 있습니다.

학습 정리

● 스레드의 생성과 사용

① 자바에서 스레드는 두 가지 방법으로 사용이 가능하다.

② Thread 클래스로부터 상속받아 스레드를 사용하는 방법과 Runnable 인터페이스를 포함하여 스레드 클래스를 사용하는 방법이 있으며, 현재의 클래스가 다른 클래스로부터 이미 상속을 받고 있는 경우에 Runnable 인터페이스를 이용하여 스레드를 작성합니다.

● 스레드의 우선순위

① 각각의 스레드는 1~10 사이의 우선순위를 가질 수 있습니다.

② CPU는 실행 가능 상태의 스레드 중에서 우선순위가 높은 스레드를 먼저 수행합니다.

학습 정리

● 스레드의 시작과 종료

① 다중 스레드를 가진 프로그램에서 start() 메소드에 의해 스레드가 시작되면, 프로그램의 흐름이 단일 흐름에서 다중 흐름으로 전환됩니다.

② 다중 스레드는 동시 수행되는 개념이므로 스레드가 수행을 마치고 다시 단일 흐름으로 프로그램을 실행시키기 위해서는 join() 메소드를 이용하여 흐름을 하나로 만들어야 합니다.

● 스레드 동기화

① 다수 개의 스레드가 임계영역을 수행하기 위해서는 synchronized 메소드를 사용해야 합니다.

② synchronized 메소드는 한순간에 하나의 스레드만 실행할 수 있습니다.

③ 한 스레드가 동기화 메소드를 수행 중이면 다른 스레드는 대기해야 합니다.

학습 정리

● 스레드 사이의 통신

① 스레드 사이의 통신을 위해 wait(), notify(), notifyAll() 메소드가 사용되며, 이러한 메소드는 Object 클래스에 제공하고 있습니다.

② notify() 메소드는 대기 상태에 있는 스레드를 깨우는 역할을 합니다.