CHAPTER 19

다중 스레드

19 CHAPTER 다중 스레드

학습목표

- 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.

구성

SECTION 1

스레드 개요

1.1 프로그램과 프로세스

컴퓨터에서 하나의 프로그램이 실행될 때, 실행중인 프로그램을 프로세스process라고 합니다. 윈도우 운영체제의 작업 관리자를 살펴보면 프로세스의 개념을 알 수 있습니다. 예를 들어 윈도우에서 한글 프로그램일 실행할 때의 작업관리자를 살펴보면 프로세스에 나타나게 되고 자바 프로그램을 실행할 때도 프로세스에 나타나게 됩니다.

윈도우 시스템은 **다수 개의 프로세스를 다수 개의 CPU가 동시에 처리합니다. 이러한 개념을 다중 프로세스**multi process라고 합니다. 작업관리자의 프로세스 창을 살펴보면 동시에 수행되고 있는 프로세스들을 볼 수 있습니다.

그림 19-1 실행중인 프로세스

1.2 프로세스와 스레드

스레드는 하나의 프로세스 내에서 동작하는 작은lightweight**(또는 경량) 프로세스**라고 볼 수 있습니다. **다중 스레드**multi thread**는 하나의 프로세스(프로그램) 내에서 다수 개의 스레드가 동시에 동작하는 개념**입니다.

자바 언어의 중요한 특징 중에 하나가 다중 스레드를 지원한다는 점입니다. **자바 언어는 다중 스레드의 기능을 실행 시스템과는 상관없이 JVM에 내장된**builtin **형태로 구현**하고 있습니다.

다중 스레드는 다수 개의 CPU를 가진 컴퓨터에서는 동시에 수행될 수 있습니다.

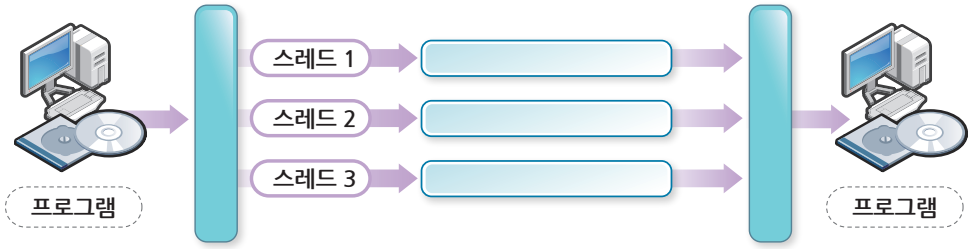

그림 19-2 다수 개의 CPU를 가진 시스템에서 다중 스레드의 실행

일반적으로 다수 개의 CPU를 가진 컴퓨터에서 다수 개의 스레드를 처리하는 방법은 운영체제에 따라 다르며, 한 개의 CPU를 가진 컴퓨터에서는 다중 스레드를 처리하기 위해서 CPU를 공유하면서 실행됩니다.

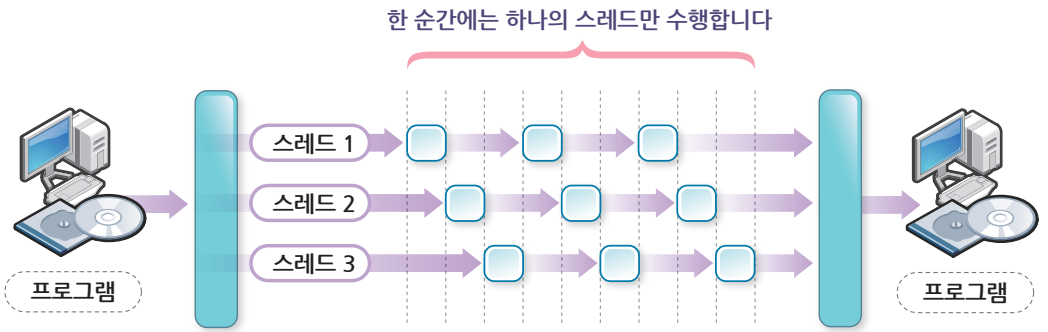

그림 19-3 하나의 CPU를 가진 시스템에서 다중 스레드의 실행

지금까지 앞에서 작성한 자바 프로그램들은 하나의 스레드만을 가진 프로그램입니다. 즉 앞에서 작성한 모든 자바 프로그램들은 main() 스레드 하나만 실행되는 형태입니다. 자바 프로그램은 실행이 요청되면 해당 클래스의 클래스 메소드인 main() 메소드를 호출하여 순차적으로 실행한 다음 종료됩니다.

SECTION 2

Thread 클래스와 스레드 생명주기

자바 프로그램에서 **스레드를 생성하기 위해서는 두 가지 방법을 사용**할 수 있습니다. **하나는 Thread 클래스로부터 직접 상속받아 원하는 스레드를 생성하는 방법**이고, **다른 하나는 스레드 기능을 제공하는 Runnable 인터페이스를 사용하는 방법**입니다. 두 가지 방법은 조금 다른 형식으로 사용됩니다.

스레드 생성과 사용은 다음 절에서 기술하고 이 절에서는 Thread 클래스에 대하여 기술합니다. **Thread 클래스는 java.lang 패키지에 라이브러리 클래스로 제공**되고 있으며, 사용자가 스레드를 사용하기 위해서는 이 클래스를 이용해야 합니다. Thread 클래스는 다음과 같은 생성자를 가지며 사용자는 적합한 생성자를 이용하여 스레드를 생성하여야 합니다.

[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

스레드 클래스는 4개의 생성자를 가집니다(생성자 오버로딩). 세 번째와 네 번째 생성자는 Runnable 인터페이스를 사용한 클래스로부터 스레드를 생성하기 위해 사용합니다.

표 19-1 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.

자바 프로그램에서 사용되는 스레드는 탄생하고 소멸될 때까지의 생명주기life cycle를 가집니다. 스레드는 이러한 생명주기에 따라 작동합니다.

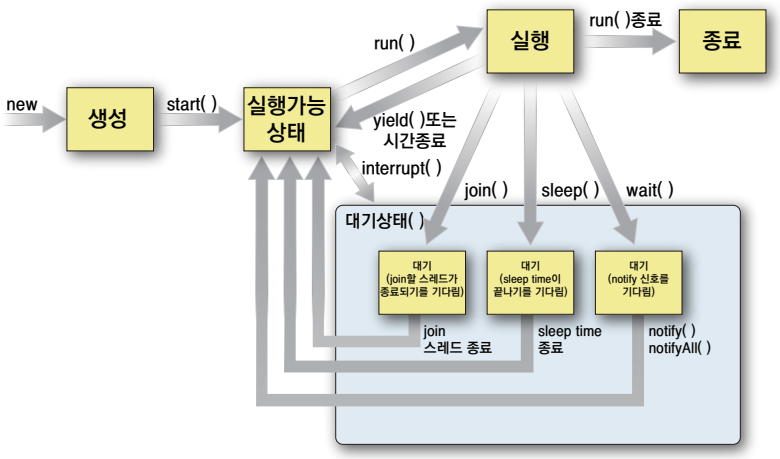

그림 19-4 스레드의 생명주기

스레드는 생성되어 종료될 때까지 5가지의 상태를 가집니다. 스레드가 생성되면 그 스레드는 메모리를 할당받고 생성 상태에 있게 됩니다.

생성된 스레드는 start() 메소드에 의해 실행 가능 상태로 들어가게 됩니다. 실행 가능 상태는 실행 상태가 아니며 운영체제로부터 CPU가 할당되기를 기다리는 상태를 의미합니다.

CPU가 할당되면 스레드는 run() 메소드가 수행되어 실행 상태가 됩니다. 실행 상태의 스레드는 CPU로부터 할당된 시간이 종료되거나 yield() 메소드가 수행되면 다시 실행 가능 상태로 바뀌게 됩니다.

다중 스레드란 여러 개의 스레드가 동시에 수행됨을 의미합니다. JVM은 다중 스레드를 지원하기 위해 스레드의 상태를 실행 상태와 실행 가능 상태로 변환해 가면서 여러 개의 스레드를 동시에 수행하는 것과 같은 효과를 내고 있습니다. **JVM은 실행 가능 상태의 스레드 중에서 하나를 선택하기 위해 우선순위 스케줄링 기법을 사용**합니다.

자바에서의 모든 스레드는 우선순위를 가지며, 사용자에 의해 스레드의 우선순위가 지정될 수 있습니다. **JVM은 모든 스레드가 같은 우선순위를 가질 경우에는 일정 시간**timeslice **동안 스레드를 번갈아 가며 수행**합니다.

스레드가 wait(), sleep(), join() 메소드를 수행하거나 입출력(interrupt())등을 요구하게 되면, 스레드는 대기상태로 전환되며 수행이 중지됩니다. **대기상태의 스레드가 해제되어 다시 실행 준비 상태로 전환되기 위해서는 join 할 스레드가 종료되거나, sleep time이 종료되거나, notify(), notifyAll() 메소드가 수행되거나, 입출력 등이 종료되는 행위가 발생**해야 합니다.

스레드는 위와 같이 실행 가능 상태, 실행상태, 대기상태를 거치면서 실행이 완료되면 종료 상태로 전환되어 더 이상 스레드의 의미를 가지지 못하게 됩니다.

SECTION 3 스레드의 생성과 사용

3.1 Thread 클래스 이용

이 방법은 **Thread(java.lang.Thread) 클래스로부터 직접 상속받아 스레드 특징을 가지는 클래스를 생성하여 스레드를 사용하는 방법**입니다. Thread 클래스는 스레드의 실행 부분을 지정하는 run() 메소드를 제공하고 있습니다([표 16-1] 참조). Thread 클래스로부터 상속되어 생성된 클래스에서는 **상위 클래스인 Thread 클래스에 정의된 run() 메소드를 오버라이딩하여 스레드의 작동을 기술**하여야 합니다.

다음은 Thread 클래스로부터 상속받아 클래스를 생성하는 예입니다.

```
01: class ThreadA extends Thread {
02:     .........
03:     public void run() {
04:         ....
05:     }
06:     .......
07: }
```

Thread 클래스로부터 상속

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

위의 예는 Thread 클래스로부터 직접 상속받아 스레드 클래스를 생성하였습니다. run() 메소드는 스레드를 시작시키는 메소드인 start() 메소드에 의해 자동적으로 수행되는 메소드입니다. 즉 run() 메소드는 프로그램에서 명시적으로 호출되지 않고, start() 메소드에 의해 자동적으로호출됩니다.

다음은 위의 스레드 클래스로부터 스레드 객체를 생성하여 수행시키는 예입니다.

```
01: ThreadA ta = new ThreadA();
02: ta.start();
```

ThreadA 클래스로부터 스레드 객체 ta를 생성하여 ta 객체의 start() 메소드를 호출합니다. **start() 메소드는 Thread 클래스로부터 상속된 메소드로써 스레드를 실행 가능 상태로 만들게 되며, JVM은 CPU 할당이 가능할 때 run() 메소드를 호출하여 실행**하게 됩니다.

3.2 Runnable 인터페이스 이용

스레드를 생성시키는 또 다른 방법은 Runnable 인터페이스를 이용하는 것입니다. **스레드의 특성을 가져야 하는 클래스가 이미 다른 클래스로부터 상속을 받고 있다면 Runnable 인터페이스를 이용**해야 합니다.

Runnable 인터페이스에는 다음과 같이 run() 메소드 하나만 선언되어 있습니다.

```
01: public interface Runnable {
02:     public void run();
03: }
```

다음은 Runnable 인터페이스를 이용하여 스레드 특징을 가지는 클래스를 생성하는 예입니다. Thread 클래스를 사용하는 것과 같이 **Runnable 인터페이스로 구현된 클래스도 Runnable 인터페이스에서 제공하고 있는 run() 메소드를 오버라이딩하여 스레드가 수행할 문장들을 정의**해야 합니다.

```
01: class RunnableB extends Applet implements Runnable {
02:     .........
03:     public void run() {  <-- Runnable 인터페이스에 정의된 run() 메소드를 오버라이딩하여 스레드가 수행할 문장들을 기술
04:         ........
05:     }
06:     ........
07: }
```

앞의 예에서 기술한 클래스는 이미 Applet 클래스로부터 상속을 받고 있으므로 인터페이스를 이용하여 스레드를 사용하였습니다. 다음은 인터페이스를 이용하여 생성된 클래스로부터 스레드 객체를 생성하는 예입니다.

```
01: RunnableB rb = new RunnableB();  ← 객체 rb 생성
02: Thread tb = new Thread(rb);  ← rb를 매개변수로 하여 스레드 객체 tb를 생성
03: tb.start();  ← 스레드 시작
```

또는

```
01: RunnableB rb = new RunnableB();  ← 스레드 객체를 생성하여 바로 시작
02: new Thread(rb).start();
```

위의 두 방법은 같은 의미입니다.

RunnableB 클래스로부터 rb 객체를 생성합니다. 생성된 객체 rb는 스레드의 특성 중 run() 메소드만을 가지고 있습니다. Runnable 인터페이스에는 run( ) 메소드만이 선언되어 있기 때문입니다. 스레드의 특성을 가지게 하기 위해 객체 rb를 매개변수로 하여 Thread 클래스로부터 객체를 생성하여야 합니다. 객체 rb를 매개변수로 Thread 객체 tb를 생성한 다음 스레드를 시작시킵니다.

실습 예제

다음 프로그램은 Thread 클래스로부터 상속받아 스레드 클래스를 생성하는 예를 보이고 있습니다. 이 프로그램에서의 스레드 클래스의 run( ) 메소드에서 반복문을 이용하여 10개의 문장을 출력하고 있습니다.

예제 19.1 ThreadFromThread.java

```
01: class ThreadTest extends Thread {  ← Thread 클래스로부터 상속받아 클래스 작성
02:     public void run() {
03:         for (int i=1 ; i<=10 ; i++) {
04:             System.out.println("재미있는 자바 :" + i);
05:         }
```

Thread 클래스로부터 상속받아 클래스 작성

```
06:    }
07: }
08: public class ThreadFromThread {
09:    public static void main(String args[]) {
10:        ThreadTest t = new ThreadTest();  ← 스레드 특성을 가진 객체 생성
11:        t.start();  ← 스레드 시작(run( ) 메소드 호출)
12:     }
13: }
```

설명

01 스레드의 특성을 가진 클래스를 작성하기 위해 Thread 클래스로부터 상속을 받아 클래스를 작성하였다. Thread 클래스로부터 상속받았기 때문에 이 클래스의 객체에서는 Thread 클래스의 모든 메소드를 사용할 수 있다.

02~05 run( ) 메소드는 Thread 클래스가 가지고 있는 메소드이다. run( ) 메소드를 오버라이딩하여 스레드가 해야할 일을 기술한다. run( ) 메소드는 start( ) 메소드에 의해 자동 호출되는 메소드이다.

10 스레드 객체를 생성한다.

11 객체의 start( ) 메소드를 호출하여 스레드를 시작시킨다. start( ) 메소드는 Thread 클래스에서 제공되는 메소드로서 run( ) 메소드를 호출한다. 이때 실행되는 run( ) 메소드는 Thread 클래스에서 제공된 run( ) 메소드가 아니라, 하위 클래스에 오버라이딩된 run( ) 메소드가 수행된다.

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
```

실습 예제

다음 프로그램은 앞 프로그램을 Runnable 인터페이스를 이용하여 작성한 프로그램입니다. 이 프로그램은 앞 프로그램과 같은 결과를 출력합니다.

예제 19.2 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)

설명

01~05 이미 클래스가 다른 클래스로부터 상속을 받고 있다고 가정하여, Runnable 인터페이스를 포함하는 클래스를 작성하였고, Runnable 인터페이스에 선언된 run( ) 메소드를 오버라이딩하여 스레드가 해야 할 기능을 정의하였다. 그러나 이 클래스는 아직 완벽한 스레드의 특성이 아닌 Runnable 인터페이스의 특성만(run( )메소드만 가짐) 가지고 있는 상태이다.

10 인터페이스를 포함한 클래스로부터 객체를 생성한다. 이 객체는 Runnable 인터페이스의 특성만 가지게 된다.

11 Runnable 특성을 가진 객체를 이용하여 스레드 객체를 생성한다. 스레드 클래스는 Runnable 형의 객체를 이용하여 객체를 생성할 수 있는 생성자를 제공하고 있다.

12 스레드를 시작시킨다.

실행 결과

앞 프로그램의 결과와 같습니다.

실습 예제

다음 프로그램은 다중 스레드를 사용하고 있습니다. 하나의 스레드 클래스로부터 두 개의 스레드 객체를 생성하여 시작시켰습니다. 시작된 두 개의 스레드는 아무런 관계없이 동시에 실행됩니다. 이 프로그램은 실행시킬 때마다 다른 결과를 출력합니다. 스레드를 실행하는 JVM의 상태에 따라 각각의 스레드들이 실행되기 때문입니다.

예제 19.3 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: }
```

설명

01 스레드의 특성을 가진 클래스를 작성하기 위해 Thread 클래스로부터 상속을 받아 클래스를 작성하였다.

03 생성자에서 스레드의 이름을 설정하였다. setName( ) 메소드는 Thread 클래스에서 제공되는 메소드로 스레드의 이름을 설정하는 메소드이다.

07 getName( ) 메소드를 사용하여 스레드의 이름을 출력한다.

14~17 하나의 스레드 클래스로부터 두 개의 스레드 객체를 생성한다.

18, 19 두 개의 스레드를 시작시킨다. 두 개의 시작 문장이 순차적으로 수행되는 개념이 아니고 스레드의 특성을 가진 객체이기 때문에 동시에 수행된다.

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
```

실행 결과

[또 다른 실행결과]

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

SECTION 4

스레드 우선순위

앞 절에서 두 개의 스레드가 수행되는 예를 살펴보았습니다. 예에서와 같이 두 개의 스레드는 어떤 순서 없이 수행시킬 때마다 다른 결과를 나타내었습니다. 자바에서는 **스레드에 우선순위를 부여하여 우선순위가 높은 스레드에 실행의 우선권을 주어 실행**시킬 수 있습니다.

Thread 클래스에는 스레드에 우선순위를 부여하는 setPriority() 메소드가 제공되고 있습니다. 또한 Thread 클래스에는 우선순위를 지정하기 위한 다음과 같은 상수가 제공되고 있습니다.

```
01: static final int MAX_PRIORITY ◄····· 우선순위 값으로 10을 가집니다.
02: static final int MIN_PRIORITY ◄····· 우선순위 값으로 1을 가집니다.
03: static final int NORM_PRIORITY ◄····· 우선순위 값으로 5를 가집니다.
```

앞에서 배운 스레드의 생명주기 그림에서도 살펴보았듯이 스레드는 실행 상태와 실행 가능 상태를 가집니다. JVM은 현재의 스레드를 수행한 다음에는 실행 가능 상태의 스레드 중에서 하나를 선택하여 실행하게 되는데, 우선순위가 같을 때에는 스레드 스케줄러의 기준에 의하여 실행될 스레드가 결정됩니다.

스레드의 우선순위는 스레드 스케줄러가 다음에 실행할 스레드를 선정하는데 영향을 미치게 됩니다. 즉 우선순위가 높은 스레드가 우선적으로 선정되어 실행되게 됩니다.

실습 예제

다음 프로그램은 세 개의 스레드 객체에 각기 다른 우선순위를 부여하여 실행시키는 프로그램입니다. 프로그램 실행 시 입력받은 두 개의 매개변숫값으로 우선순위를 설정하였습니다. 이 프로그램은 실행시킬 때마다 서로 다른 결과를 출력하는데, 대체적으로 우선순위가 높은 스레드가 우선 수행됨을 볼 수 있습니다.

이 프로그램은 프로그램을 실행할 때 main() 메소드의 매개변수로 우선순위를 입력하였습니다. 이클립스에서 main() 메소드의 매개변수를 입력하는 방법은 9장 4절의 4.5 부분을 참고하시기 바랍니다.

예제 19.4 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

설명

07, 08 getPriority( ) 메소드를 이용하여 스레드가 가진 우선순위를 출력하였다.

19~21 매개변수로 받은 두 개의 값과 Thread 클래스에서 제공되는 상숫값(MIN_PRIORITY)을 이용하여 스레드의 우선순위를 설정(setPriority( )) 하였다.

이 프로그램을 여러 번 실행시켜보면 대체적으로 우선순위가 높은 스레드가 먼저 수행됨을 알 수 있습니다.

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
```

SECTION 5

스레드의 시작과 종료

다중 스레드는 하나의 프로그램 내에서 여러 개의 스레드가 실행되는 것을 의미합니다. **일반적인 프로그램은 하나의 실행 흐름을 가지는 반면, 다중 스레드가 있는 프로그램은 다수 개의 실행 흐름을 가질 수 있습니다.**

다중 스레드를 가진 프로그램에서의 실행은 하나의 흐름을 가지다가 다수 개의 스레드가 시작되면 다중 흐름으로 프로그램이 실행됩니다. **스레드를 시작시키는 start() 메소드에 의해 다중 흐름이 시작**됩니다.

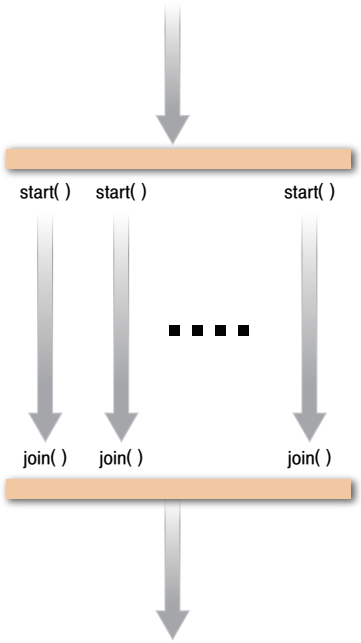

그림 19-5 다중 스레드를 가진 프로그램의 실행 흐름

프로그램에서 다수 개의 실행 흐름을 가지는 경우, 스레드가 종료되는 시점에서는 이러한 다수 개의 실행 흐름이 다시 하나의 흐름으로 통합되어야 합니다.

Thread 클래스는 다수 개의 스레드 **실행 흐름을 통합하기 위해 join() 메소드를 제공**하고 있습니다. join() 메소드를 사용하기 위해서는 관련된 예외(InterruptedException)를 처리해 주어야 합니다.

실습 예제

다음 프로그램은 앞에서 사용한 두 개의 스레드를 가진 프로그램을 변환하여 사용하였습니다. 스레드를 시작시키는 문장 다음의 문장이 스레드가 끝나기 전에 실행됨을 볼 수 있습니다.

예제 19.5 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: }
```

설명

18~21 스레드를 시작하기 전에 "***** 스레드 시작 전 *****"을 출력하고 스레드 시작 문장 다음에 "***** 스레드 종료 후 *****"를 출력하였다. 프로그램의 실행결과를 보면 두 개의 문장이 처음에 출력됨을 알 수 있다. 이 결과는 스레드를 시작시키는 문장이 동시에 실행된다는 의미이다. 스레드 시작 문장이 먼저 출력된 것은 스레드가 종료되기를 기다리지 않고 프로그램은 계속 수행이 된다는 의미이다.

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
```

실습 예제

다음 프로그램을 변환하여, 스레드가 종료된 다음 출력문의 내용을 출력하도록 프로그램을 작성하였습니다. Thread 클래스의 join( ) 메소드를 이용하여 모든 스레드가 종료되기를 기다려 다음을 실행하는 프로그램입니다. join( ) 메소드를 사용하기 위해서는 관련된 예외를 처리하여야 합니다.

예제 19.6 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 {
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();
22:         t2.join();
23:         System.out.println("***** 스레드 종료 후 *****");
24:     }
25: }
```

join( ) 메소드 사용을 위해 예외 처리

join( ) 메소드 수행

스레드 종료 후에 실행

설명

13 main( ) 메소드 내에서 join( ) 메소드를 사용하기 위해 예외를 처리하였다. 예외 처리에서 지정된 Exception 클래스는 대부분의 예외 관련 클래스를 하위 클래스로 가지고 있다. Exception을 지정함으로써 하위 클래스와 관련된 모든 예외를 처리할 수 있다. throws 절을 이용하여 발생된 예외를 JVM(main( ) 메소드를 호출하였기 때문에)에 넘기는 예외 처리를 하고 있다.

21, 22 각 스레드가 끝나기를 기다렸다가, 모든 스레드가 종료되기를 기다린다. 스레드가 종료되면 다음 문장으로 제어가 이동된다.

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 *****
```

SECTION 6

스레드 동기화 Synchronization

앞 절의 예제에서 두 개의 스레드가 동시에 수행되는 프로그램을 보았습니다. 두 개의 스레드는 아무런 상관없이 독립적으로 수행되는 형태입니다.

그러나 **우리가 작성하려는 응용 프로그램들의 대부분은 동시 수행되는 다수 개의 스레드가 어떤 연관 관계를 가지면서 작동하는 형태**입니다. 예를 들어 한 대의 프린터를 여러 대의 컴퓨터가 네트워크로 공유하는 환경에서, 하루에 프린터에서 소요되는 페이지 수를 알아내는 프로그램을 생각해 보자. 각각의 컴퓨터에서 동시에 또는 경쟁적으로 요구되는 프린터 사용 요청을 스레드로 표현하고, 하루에 소요되는 페이지 수를 구하기 위해 count라는 변수를 사용한다고 가정하자.

각각의 스레드는 count 변수를 상호 배타적으로 사용해야 할 것입니다. 즉 하나의 스레드가 프린터를 사용하고 count=count+1을 수행하는 동안에는 다른 스레드가 이 변수에 접근할 수 없음을 의미합니다. 이러한 영역을 임계영역 critical section이라 합니다. 즉 **임계영역이란 다수 개의 스레드가 접근 가능한 영역이면서, 한순간에는 하나의 스레드만 사용할 수 있는 영역을 의미**합니다.

자바에서는 임계영역 지정을 위한 synchronized 메소드를 제공합니다. 스레드는 임계영역인 synchronized 메소드에 들어가면서 lock을 얻고 이 메소드를 벗어나면서 lock을 양보합니다. 하나의 스레드가 synchronized 메소드를 수행 중이면 다른 스레드는 lock이 양보될 때까지 기다려야 합니다.

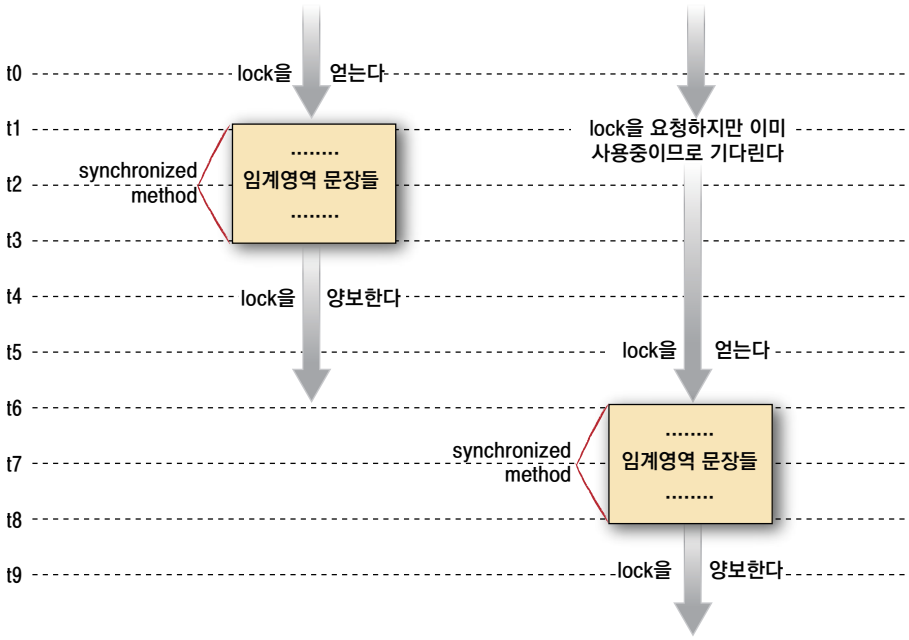

그림 19-6 다중 스레드 환경에서의 synchronized 메소드 실행

위 그림에서는 각각의 스레드가 서로 다른 시간에 lock을 요청하는 형태입니다. 만일 두 개 이상의 스레드가 같은 시간에 lock을 요청하는(synchronized 메소드의 실행을 요청하면) 경우에는 어떻게 처리될까요?

같은 시간에 각각의 스레드가 동시에 하나의 메소드(synchronized 메소드)의 수행을 요청하면 JVM은 그들 중 하나에 실행을 허용합니다. 스레드에 들어가는 lock을 얻고 양보하는 일은 JVM에 의해 자동적으로 수행되는 원자적인 atomically : 분리되어 수행될 수 없는 동작입니다.

실습 예제

다수 개의 스레드가 경쟁적으로 동기화 메소드를 요청하는 프로그램을 작성해 봅시다. 다음 프로그램은 TV에서 자주 볼 수 있는 불우 이웃 돕기 ARS 시스템을 부분적으로 구현한 예입니다. 5명의 성금자가 성금 총액이 50만 원이 될 때까지 1,000원씩을 경쟁적으로 낸다고 가정합니다. 이 프로그램에서 성금자를 스레드로 표현하였습니다. 5명의 성금자 스레드가 경쟁적으로 총액에 1,000원을 더하려고 시도합니다.

현재의 총액에 1,000원씩을 더하는 부분은 임계영역(synchronized 메소드)으로서 한순간에 하나의 스레드만 수행할 수 있어야 합니다. 만일 동시에 접근이 허용된다면(다수 개의 스레드가 같은 금액을 가져와 각각 1,000원씩을 더한다), 성금의 총액이 정확하게 집계되지 않게 됩니다.

다음 그림은 프로그램의 작동 구조를 나타낸 것입니다. 중요한 부분은 5개의 스레드가 공유하는 부분인 account 객체입니다. 이 객체의 deposit( ) 메소드를 통하여 각각의 스레드들은 경쟁적으로 1,000원씩 성금을 내려고 하는 구조입니다.

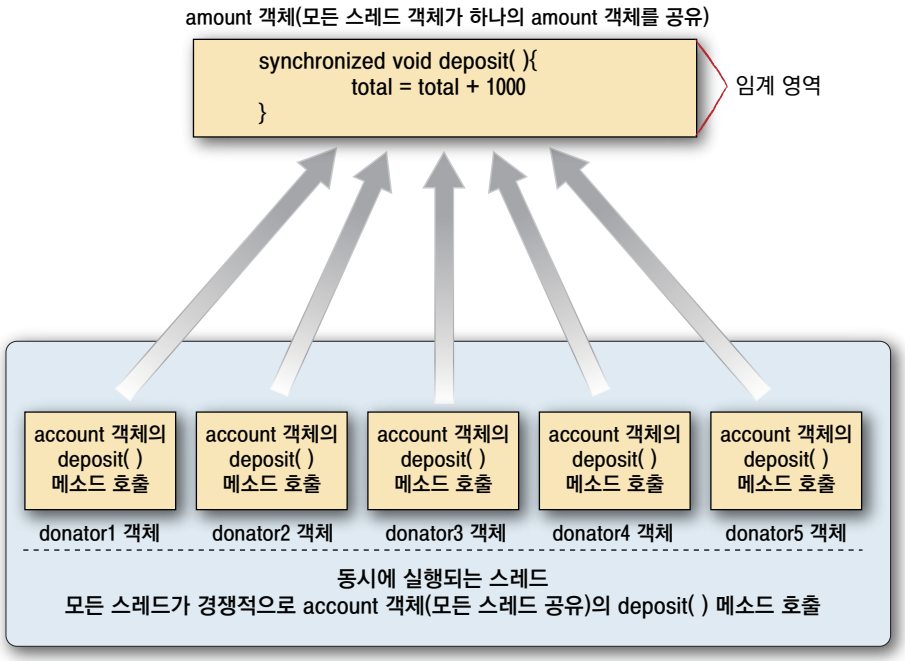

그림 19-7 프로그램의 작동 구조

예제 19.7 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;
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();
20:          if (same_a.gettotal() >= 500000) break;
21:       }
22:    }
23: }
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();
```

스레드 이름과 Account 객체를 설정

반복문을 돌며 deposit() 메소드 호출

전체금액이 50만원 보다 크면 반복문을 벗어남

Account 클래스로부터 객체 생성

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

```
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: }
```

>> 설명

03~05 Account 클래스에 deposit( ) 동기화 메소드를 선언하였다. 이 클래스로부터 생성된 객체는 5개의 스레드에 의해 공유된다. 즉 5개의 스레드가 하나의 Account 클래스 객체를 공유하면서 동기화 메소드인 deposit( )을 경쟁적으로 호출하게 된다. 동기화 메소드는 동시에 호출되어도 하나의 한순간에는 스레드만 실행할 수 있고, 나머지 스레드는 대기하게 된다.

12~15 스레드 클래스이다. 생성자에서 스레드의 이름과 Account 객체를 설정하고 있다. 이미 5개의 스레드가 같은 Account 객체를 사용하여(27–31라인 참조) 스레드 객체를 생성하였기 때문에 5개의 스레드가 Account 객체를 공유하게 된다.

19~20 스레드 클래스의 run( ) 메소드 내에 200번을 반복하며 deposit( ) 메소드를 호출하도록 하였다. 총 금액이 50만 원이 넘으면 반복을 중단하도록 하였다.

26 Account 클래스로부터 하나의 객체 same_account를 생성하였다.

27~36 하나의 same_account 객체를 이용하여 5개의 스레드 객체를 생성하였다. 결국 5개의 스레드 객체는 하나의 same_account 객체를 공유하는 형태이다.

프로그램을 실행시킬 때마다 다른 결과를 출력합니다. 지면상 출력의 중간 부분은 생략하였습니다. 이 프로그램의 결과를 보면 네 번째 성금자는 125번째 성금을 납부하였습니다. 즉 모든 스레드가 경쟁적으로 성금을 납부하게 되고 시스템의 상태에 따라 스레드마다 납부하는 성금 횟수가 다르게 됩니다.

또한 이 프로그램은 성금 총액을 504,000을 출력하였습니다. 각각의 스레드는 run( ) 메소드의 if(account1.gettotoal( ) >= 500000) 문장이 true가 되면 종료됩니다.

성금 총액이 504,000원이 나온 이유는 하나의 스레드가 if문을 수행하는 순간에 이미 다른 스레드는 임계영역(1,000원을 더하기 위해)의 수행을 위해 기다리고 있기 때문입니다.

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
```

SECTION 7

스레드 사이의 통신

앞 절에서 다수 개의 스레드가 순서 없이 경쟁적으로 임계영역에 접근하는 방법을 보았습니다. 이제 **스레드가 서로 통신하면서 수행되는 프로그램**의 형태를 살펴보겠습니다.

스레드 사이의 통신 문제를 살펴보기 위해 쌀을 생산하는 생산자와 쌀로 밥을 짓는 소비자의 관계를 살펴보겠습니다. 생산자와 소비자는 각자의 스레드로서 동시에 작동합니다. 생산자와 소비자 사이에는 항상 한 개의 그릇(버퍼)만 있다고 가정하자. 생산자가 쌀을 생산하여 그릇에 가져다 놓으면 소비자는 가져가서 밥을 짓는 과정입니다.

만일 생산자가 생산을 해서 쌀을 가져다 놓으려 했는데, 그릇에 있는 쌀을 소비자가 가져가지 않았다고 가정해보자. 생산자는 소비자가 가져가기를 무한정 대기해야 합니다. 소비자는 쌀을 가져간 다음에는 무한정 대기하고 있는 생산자에게 연락을 해주어야 할 것입니다. 반대의 경우에도 마찬가지일 것입니다.

위와 같은 문제를 해결하기 위해서는 스레드가 무한정 대기로 들어가는 방법과 대기하고 있는 스레드를 깨워주는 방법이 있어야 합니다.

자바에서는 **스레드 사이의 통신을 위한 메소드를 java.lang.Object 클래스(11장 참조)에 제공**하고 있습니다. **Object 클래스에서 제공되는 메소드 중 wait(), notify(), notifyAll() 메소드가 스레드 사이의 통신에 이용**됩니다.

스레드가 메소드를 수행 중에 **wait() 메소드**를 만나면, 가지고 있는 lock을 양보하고 대기 상태에 들어간다([그림 16-3] 참조). wait() 메소드는 3가지 형태를 가집니다.

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

wait()는 무한정 대기 상태에 들어가는 메소드이며 다른 메소드에 의해 깨워져야 합니다. **wait(long msec)**는 msec 밀리초 동안 대기하다 스스로 깨어나는 메소드입니다. **wait(long msec, int nsec)**는 msec 밀리초와 nsec 나노초 동안 대기하다 스스로 깨어나는 메소드입니다. 이 wait() 메소드는 예외를 처리해야 하는 메소드입니다.

notify()와 notifyAll() 메소드는 synchronized 메소드를 수행하기 위해 기다리는 대기 상태의 스레드를 깨워줍니다. notify()는 대기 중인 스레드가 여러 개일 경우 그중에 하나의 스레드만을 깨운다. 여러 개의 스레드 중 하나를 선정하는 일은 JVM에 의해 이루어집니다. **notifyAll()** 메소드는 대기 중인 모든 스레드를 깨우는 것을 제외하고는 notify()와 같습니다.

두 메소드의 형식은 다음과 같습니다.

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

wait()와 notify(), notifyAll() 메소드는 다음 그림과 같이 작동합니다.

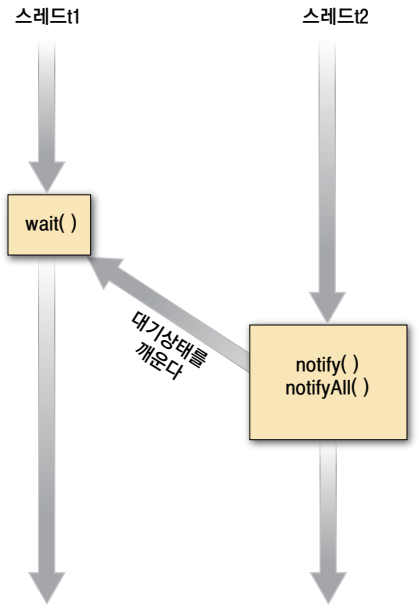

그림 19-8 wait() 메소드와 notify(), notifyAll() 메소드의 작동

일반적으로 notify()와 notifyAll() 메소드는 synchronized 메소드 내에서 사용될 수 있습니다. 왜냐하면 대기 상태에 있는 스레드를 깨우는 역할을 하므로, 다수 개의 스레드에서 동시에 호출되면 안 되기 때문입니다.

이상의 3가지의 스레드 관련 메소드는 모두 java.lang.Object 클래스(표 16-2 참조)에 정의되어 있으므로 사용자가 작성하는 모든 클래스에서 사용할 수 있습니다.

예제 19.8 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 {
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();
55:       Producer p1 = new Producer(buff);
56:       Consumer c1 = new Consumer(buff);
57:       p1.start() ;
58:       c1.start() ;
59:    }
60: }
```

소비자 스레드 클래스

하나의 Buffer 객체 생성

동일한 Buffer 객체를 사용하여 스레드 객체 생성

설명

02 생산자와 소비자가 공유하는 데이터(쌀)를 넣는 곳으로 private 변수를 선언하였다. 이 변수를 이용하여 생산자는 숫자를 저장하고, 소비자는 저장된 숫자를 가져간다.

03 생산자와 소비자가 번갈아가며 수행하기 위한 이진 변수를 사용한다. 이 문제에서는 먼저 생산자가 생산을 해야 하므로 초깃값을 false로 지정한다.

04 생산자가 생산된 데이터를 가져다 놓는 동기화 메소드이다.

05~08 생산자가 데이터를 가져다 놓을 때 현재의 공유 변수의 값을 소비자가 소비했는지를 검사하는 부분이다. 이진 변수의 값에 따라 데이터를 출력할 것인지, 아니면 소비자가 가져갈 때까지 대기할 것인지가 결정된다.

14 이 부분은 데이터를 생산(출력)하고 소비자 스레드를 깨우는 부분이다. 소비자 스레드가 대기 상태에 있다면, 이 메소드의 수행으로 소비자 스레드가 다시 실행된다.

17 데이터를 소비하는 동기화 메소드를 선언한다.

18~21 소비자가 데이터를 가져갈 때 현재의 공유 변수의 값을 생산했는지를 검사하는 부분이다. 이진 변수의 값에 따라 데이터를 가져갈 것인지, 아니면 생산자가 생산할 때까지 대기할 것인지가 결정된다.

26 이 부분은 데이터를 소비하고 생산자 스레드를 깨우는 부분이다. 생산자 스레드가 대기 상태에 있다면, 이 메소드의 수행으로 생산자 스레드가 다시 실행된다.

31,41 생산자와 소비자 스레드 클래스이다. 각 클래스의 run() 메소드 내에서 반복적으로 put() 메소드와 get() 메소드를 호출하고 있다.

54 하나의 Buffer 객체를 생성한다.

55, 56 생산된 하나의 동일한 Buffer 객체를 사용하여 스레드 객체를 생산함으로써, 두 개의 스레드가 Buffer 객체를 공유하며 실행된다.

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
```

학/습/정/리

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.

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.

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.

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.

CHAPTER 20

네트워크

20 CHAPTER 네트워크

학습목표

- 네트워킹의 개념에 대해 학습합니다.
- 인터넷의 주소와 URL을 네트워크를 통해 사용하는 방법을 학습합니다.
- 연결성 통신 방법인 TCP 소켓에 대해 학습합니다. 두 개의 프로그램이 연결성 통신 방법을 사용하여 통신하는 프로그램을 작성합니다.
- 비연결성 통신 방법인 UDP 소켓에 대해 학습합니다. 두 개의 프로그램이 비연결성 통신 방법을 사용하여 통신하는 프로그램을 작성합니다.

구성

SECTION 1

네트워킹의 개요와 java.net 패키지

자바는 편리하고 다양한 네트워크 관련 기능을 제공합니다. 네트워크(인터넷)로 연동된 컴퓨터와 관련된 정보를 알아내고, WWW 문서 정보를 알아내고, 다른 컴퓨터와 연동되어 정보를 주고받는 등의 다양한 기능들을 제공하고 있습니다.

이러한 기능들을 제공하기 위해 자바는 네트워크의 복잡한 구조를 잘 이해하지 못하고도, **네트워크 관련 프로그램을 작성할 수 있는 편리한 클래스들을 java.net 패키지로 제공**하고 있습니다. 사용자는 이 패키지에서 제공되는 편리한 클래스들을 익혀 간단하게 네트워크 관련 프로그램들을 작성할 수 있습니다.

이 절에서는 자바 네트워킹 프로그램을 작성하기 위한 네트워크의 기본 개념과 java.net 패키지, 소켓을 이용한 클라이언트-서버 통신, 데이터그램을 이용한 통신 등에 관해 기술합니다.

1.1 TCP/IP Transmission Control Protocol/Internet Protocol

인터넷이나 네트워크를 통하여 컴퓨터들이 연결되기 위해서는 상호 연결 방법을 정의한 프로토콜protocol**을 사용**해야 합니다. **프로토콜은 컴퓨터 상호 간에 통신을 위한 규약으로 정의**할 수 있습니다. 현재 인터넷에서는 TCP/IP 프로토콜을 표준 프로토콜로 사용하고 있으며 인터넷에서 사용되는 모든 서비스들은 이러한 TCP/IP 프로토콜을 기반으로 작동합니다.

TCP/IP 프로토콜은 다음 그림과 같이 4개의 기능 계층들로 구성되어 있습니다. 우리가 작성하여 사용하는 대부분의 통신 응용 프로그램들은 대부분 응용 계층에서 이루어집니다.

사용자가 작성한 응용 프로그램이 인터넷으로 연결된 컴퓨터의 다른 응용 프로그램과 통신을 한다고 가정하자. **사용자는 단순히 응용 계층에서 연결하는 기능을 이용하지만, 실제 통신은 매우 복잡한 과정을 거쳐 상대 컴퓨터와 연결되고 데이터가 전송**됩니다. 즉 사용자는 단순히 응용 계층만을 이용하여 통신을 이용하지만 실제는 하부 계층의 관련 프로토콜을 통하여 전송이 이루어집니다.

그림 20-1 TCP/IP 계층 구조

자바에서 네트워크 관련 클래스를 이용하여 프로그래밍하는 것은 응용 계층에서의 사용을 의미합니다. 자바의 네트워크 관련 클래스들은 사용자가 하위 계층을 잘 모르더라도 쉽게 통신 프로그램을 작성할 수 있는 기능을 제공합니다.

12 TCP와 UDP

TCP/IP의 전달 계층은 크게 두 가지로 구분됩니다. **연결형인 TCP**Transmission Control Protocol**와 비연결형인 UDP**User Datagram Protocol**로 나뉜다.** TCP나 UDP 모두 패킷packet 단위로 전송이 이루어집니다. 패킷은 주고받는 정보의 단위입니다. **TCP와 UDP는 전화와 편지에 비유해 볼 수 있습니다.**

TCP는 연결성 통신 방식으로 전화처럼 먼저 수신자와 연결을 설정한 다음 정보를 주고받는 방식을 의미하고, 전화가 끊어지면 상호 통신은 끝나게 됩니다. 연결을 설정한다는 것은 이후에 전송될 모든 패킷이 유일한 경로를 따르게 됨을 의미하므로, 보내어지는 정보 즉 패킷은 보낸 순서대로 도착하게 되어 신뢰성이 보장됩니다.

TCP는 상호 연결 상태에서 신뢰성 있는 통신을 보장하기 위한 응용에 많이 사용됩니다. 인터넷에서 상대 컴퓨터를 연결하는 TELNET, WWW 서비스를 제공하는 HTTP, 파일을 전송하기 위한 FTP 등은 TCP 기반의 응용입니다.

UDP는 비연결성 통신 방식으로 편지처럼 보내는 사람에 의해 보내지면 받는 사람은 그 편지를 받아야 편지가 도착한 사실을 알게 되는 경우입니다. 이 방식은 TCP 방식에 비해 신뢰성이 떨어집니다.

UDP 방식은 전송 전에 연결을 미리 설정하지 않으므로 전송되는 모든 패킷이 목적지 주소를 반드시 가져야 합니다. 네트워크를 통하여 전달되는 정보는 순서에 의해 도착되어야 하고 훼손되거나 중복되지 않아야 합니다. 그러나 UDP 방식에서는 전송된 패킷은 데이터가 중복되거나, 도착하지 않거나, 도착하는 순서가 일정하지 않을 수 있습니다.

UDP는 TCP처럼 연결 상태가 아닌 일방적으로 상대방에게 정보를 보내는 응용에서 많이 사용됩니다. UDP는 신뢰성이 떨어지는 반면, TCP보다는 네트워크에 부담을 주지 않습니다. 홍보 자료와 같은 대량의 데이터를 많은 사람들에게 보낼 때 유용합니다. 상대 컴퓨터의 상태를 점검하기 위해 사용하는 ping이 UDP를 사용하는 예입니다.

13 소켓 socket

네트워크의 소켓socket**은 전기선의 소켓과 유사**합니다. 즉 컴퓨터가 연결된 통신의 끝점을 의미합니다. 소켓은 상호 연결된 응용 프로그램들 사이의 안정된 정보 교환을 제공하고 있는 매체라 할 수 있습니다. 즉 소켓에 쓰는 일은 상대 프로그램에 데이터를 전달하는 것이고, 소켓으로부터 읽어들이는 것은 상대 프로그램이 전송한 데이터를 수신하는 것입니다. **TCP/IP소켓에는 TCP** Transmission Control Protocol **소켓과 UDP** User Datagram Protocol **소켓이 있고, 자바는 이들 두 소켓과 관련된 클래스들을 지원**하고 있습니다.

14 포트 port

일반적으로 컴퓨터들은 하나의 통신선에 의해 네트워크와 연동합니다. 하나의 컴퓨터에는 여러 개의 포트(논리적인 개념)가 있습니다. 한 대의 컴퓨터에서 여러 개의 통신 프로그램들이 동시에 수행되는 환경을 가정해 봅시다. 통신 프로그램들이 사용하는 컴퓨터의 물리적인 통신선은 하나지만 각 프로그램들은 서로 다른 포트를 사용하여 동시에 통신을 수행할 수 있습니다.

즉 포트는 통신선을 통해 수신되는 데이터가 컴퓨터 내의 여러 통신 프로그램 중에서 하나의 프로그램에 전달되도록 하기 위한 번호입니다. 인터넷을 통하여 전달되는 정보들은 목적지 컴퓨터의 주소(32비트)와 16비트의 포트번호로 구성된다(예 203.233.51.1:8088).

15 java.net 패키지

앞에서도 언급하였듯이 자바는 플랫폼과 독립적인 언어입니다. **java.net 패키지에서 제공되는 클래스들도 시스템 독립적인 통신 기능을 제공**하고 있습니다. 다음은 java.net 패키지의 클래스 구성도입니다.

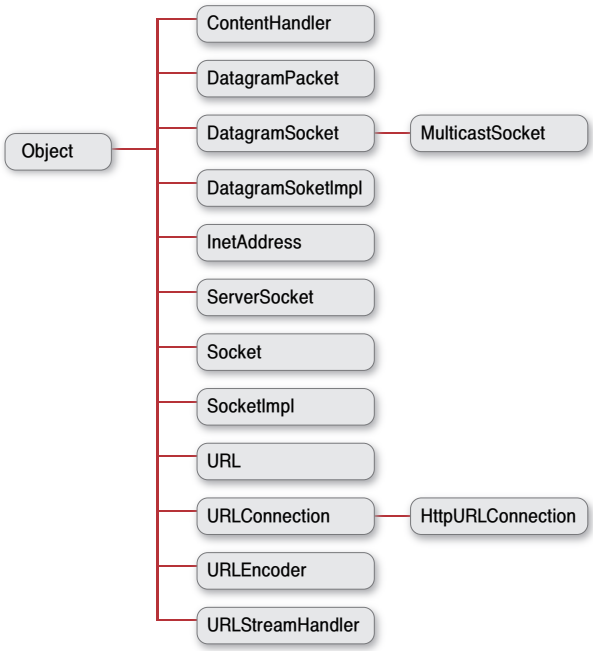

그림 20-2 java.net 패키지의 클래스

SECTION 2

인터넷 주소와 URL

전화를 걸려면 상대방의 전화번호를 알아야 하듯이 인터넷이나 네트워크와 연동된 컴퓨터와 연결하기 위해서는 상대 컴퓨터의 주소를 알아야 합니다. 인터넷에서는 현재 32비트 주소 체계를 사용하고 있으나, 주소의 포화 상태로 인하여 128비트 주소 체계로 확장될 예정입니다.

인터넷의 주소에는 컴퓨터 상호 간에 사용하는 숫자로 구성된 IP Internet Protocol **주소가 있고 (예 : 134.23.33.200), 우리가 많이 사용하는 문자 중심의 도메인**Domain **이름(예 www.ehan.co.kr)이 있습니다.**

사용의 편리함으로 인하여 대부분의 사용자는 도메인 이름을 사용하고 있습니다. 그러나 인터넷에서 사용하는 도메인 이름은 DNS Domain Name System**에 의해 IP 주소로 바뀌어 사용**되게 됩니다. 즉 컴퓨터와 컴퓨터 사이의 통신은 IP 주소에 의해 이루어집니다.

URL Uniform Resource Locator 은 웹 World Wide Web 에서 사용하는 주소입니다.

[형식] URL의 형식

```
protocol://host:port/filename(경로포함)
```

protocol은 인터넷상에서 자원을 검색할 때 사용할 프로토콜을 의미합니다. 웹 문서를 검색하기 위해서 http, 파일을 전송하기 위해서는 ftp, 전자우편을 보내기 위해서는 mailto 프로토콜을 사용합니다. host는 연결을 원하는 컴퓨터의 주소를 의미하며, port는 연결할 컴퓨터의 포트번호를 의미합니다. 포트번호가 생략되면 묵시적인 default 포트에 연결합니다.

filename은 연결된 컴퓨터에서 검색을 원하는 파일의 이름(경로 포함)입니다.

자바에서는 인터넷의 주소와 URL과 관련된 클래스들을 제공하고 있습니다.

21 InetAddress 클래스

InetAddress 클래스는 인터넷 주소에 관한 정보를 가집니다. 이 클래스는 주소와 관련된 다양한 메소드를 제공하고 있습니다.

표 20-1 InetAddress 클래스의 주요 메소드

메소드	설명
boolean equals(InetAddress other)	현 객체가 other 객체와 같은 주소를 가지면 true, 아니면 false를 반환
byte[ ] getAddress( )	주소를 나타내는 4개의 요소를 가진 바이트 배열을 반환
String getHostAddress( )	주소 정보를 나타내는 문자열을 반환
String getHostName( )	컴퓨터 이름을 나타내는 문자열을 반환
static InetAddress getLocalHost( ) throws UnknownHostException	현재 컴퓨터를 나타내는 InetAddress 객체를 반환
static InetAddress getByName (String hostName) throws UnknownHostException	hostName으로 지정된 컴퓨터를 나타내는 InetAddress 객체를 반환
static InetAddress[ ] getAllByName (String hostName) throws UnknownHostException	hostName으로 지정된 모든 컴퓨터(하나의 도메인 이름으로 여러 대의 컴퓨터를 사용하는 경우)를 나타내는 InetAddress 객체들의 배열을 반환

실습 예제

이 프로그램은 InetAddress 객체를 생성하여 생성된 객체의 속성을 인쇄하는 프로그램입니다.

예제 20.1 AddressTest.java

```
01: import java.net.*;
02: public class AddressTest {
03:    public static void main(String args[]) throws UnknownHostException {
04:       InetAddress Address = InetAddress.getLocalHost();
05:       System.out.println
06:          ("로컬 컴퓨터의 이름 : " + Address.getHostName());
```

현재 컴퓨터의 InetAddress 객체 생성

컴퓨터의 이름 추출

```
07:        System.out.println                                  컴퓨터의 IP 주소 추출
08:           ("로컬 컴퓨터의 IP 주소 : " + Address.getHostAddress());
09:        Address = InetAddress.getByName("java.sun.com");
10:        System.out.println                                  도메인 이름으로 InetAddress 객체 생성
11:           ("java.sun.com 컴퓨터의 이름과 IP 주소 : " + Address);
12:        InetAddress all[] =                                 www.daum.net 컴퓨터의 주소를 배열로 생성
13:           InetAddress.getAllByName("www.daum.net");
14:        for (int i=0; i < all.length; i++)
15:           System.out.println(all[i]);
16:     }
17: }
```

설명

04 현재 프로그램이 실행 중인 컴퓨터의 InetAddress 객체를 생성한다.

05, 06 현재 컴퓨터의 이름을 getHostName() 메소드를 이용하여 추출한다.

07, 08 현재 컴퓨터의 IP 주소를 getHostAddress() 메소드를 사용하여 추출한다.

09 도메인 이름으로 InetAddress 객체를 생성한다. 이렇게 생성된 객체를 출력문에서 출력하면 도메인 이름과 IP 주소가 같이 출력된다.

12, 13 클래스 메소드인 getAllByName()은 지정된 주소를 사용하는 모든 컴퓨터에 대한InetAddress 객체의 배열을 반환하는 메소드이다. 프로그램에서 사용된 "www.daum.net"은 모두 10개의 IP 주소를 가지고 있다.

실행 결과

```
로컬 컴퓨터의 이름 : cskim-PC
로컬 컴퓨터의 IP 주소 : 177.17.6.89
java.sun.com 컴퓨터의 이름과 IP 주소 : java.sun.com/192.9.162.55
www.daum.net/211.115.77.212
www.daum.net/211.115.77.213
www.daum.net/211.32.117.30
www.daum.net/211.115.77.211
www.daum.net/222.231.51.77
www.daum.net/222.231.51.78
www.daum.net/211.115.115.212
www.daum.net/222.231.51.40
www.daum.net/211.115.77.214
www.daum.net/211.115.115.211
```

22 URL 클래스

URL 클래스는 Web에서 사용하는 URL에 관한 정보를 가집니다.

[형식] URL 클래스의 생성자

```
URL(String protocol, String host, int port, String file) throws MalformedURLException
URL(String protocol, String host, String file) throws MalformedURLException
URL(String urlString) throws MalformedURLException

protocol, host, port, file : URL의 구성요소
urlString : 모든 요소를 표현한 문자열
```

표 20-2 URL 클래스의 주요 메소드

메소드	설명
String getFile( )	URL의 파일 이름을 반환
String getHost( )	URL의 호스트 이름을 반환
String getPort( )	URL의 포트번호를 반환, 묵시적인(default) 포트일 경우 –1을 반환
String getProtocol( )	URL의 프로토콜 이름을 반환
String toExternalForm( )	전체 URL의 문자열 객체를 반환
URLConnection openConnection( ) throws IOException	지정된 URL과 연결 후 URLConnection 객체를 반환
InputStream openStream( ) throws IOException	지정된 URL로부터 정보를 읽어들이기 위한 객체를 반환

실습 예제

이 프로그램은 하나의 URL 객체를 생성하여 다양한 정보를 출력하고 있습니다.

예제 20.2 URLTest.java

```
01: import java.net.*;
02: public class URLTest {
03:     public static void main(String args[]) throws
04:     MalformedURLException {
05:         URL kbs = new URL
06:             ("http://www.kbs.co.kr/aboutkbs/history.html");
07:         System.out.println("프로토콜: " + kbs.getProtocol());
08:         System.out.println("포트: " + kbs.getPort());
09:         System.out.println("호스트: " + kbs.getHost());
10:         System.out.println("파일(경로포함): " + kbs.getFile());
11:         System.out.println("전체 URL: " + kbs.toExternalForm());
12:     }
13: }
```

특정 URL을 지정하여 객체 생성

프로토콜을 출력

포트를 출력

설명

05, 06 특정 주소를 가진 URL 객체를 생성한다.

07~11 URL 클래스에서 제공되는 다양한 메소드를 이용하여 정보를 출력한다.

실행 결과

```
프로토콜: http
포트: -1
호스트: www.kbs.co.kr
파일(경로포함): /aboutkbs/history.html
전체 URL: http://www.kbs.co.kr/aboutkbs/history.html
```

20.3 URLConnection 클래스

URLConnection 클래스는 원격지 자원의 속성attribute**을 알아내기 위한 목적으로 많이 사용**합니다. 이 클래스는 상대 컴퓨터와 연결된 상태에서 원격지 자원의 속성을 파악하고, 원격지 파일을 읽어오는 다양한 메소드를 제공합니다.

표 20-3 URLConnection 클래스의 주요 메소드

메소드	설명
int getContentLength( )	해당 문서의 길이를 바이트 수로 반환
String getContentType( )	해당 문서의 타입을 반환
long getDate( )	해당 문서의 생성 날짜를 반환
long getExpiration( )	해당 문서의 파기 날짜를 반환
long getLastModified( )	해당 문서의 마지막 수정 날짜를 반환
InputStream getInputStream( ) throws IOException	원격지로부터 정보를 읽어들이기 위한 InputStream 객체를 생성하여 반환

URL과 URLConnection 클래스를 이용하여 원격지 자원의 속성 정보를 알아내고 원격지 자원을 가져오는 순서는 다음과 같다.

① 우선 URL 클래스를 이용하여 연결하고자 하는 컴퓨터의 정보를 가진 URL 객체를 생성합니다.
② URL 클래스의 OpenConnection( ) 메소드를 이용하여 URLConnection 객체를 생성합니다.
③ URLConnection 객체를 이용하여 속성을 알아냅니다.
④ URLConnection 클래스의 getInputStream( ) 메소드나 URL 클래스의 openStream( ) 메소드를 이용하여 원격지로부터 정보를 읽기 위한 InputStream 객체를 생성합니다. URL 클래스의 openStream( ) 메소드를 사용하는 경우 이 메소드는 자동으로 URL 클래스의 OpenConnection( ) 메소드를 호출한 다음 openStream( ) 메소드를 수행합니다.
⑤ InputStream 객체를 이용하여 원격지의 정보를 읽어들입니다.

위 표에 URLConnection 클래스에서 제공하는 주요 메소드를 나타내었습니다.

실습 예제

이 프로그램은 지정된 URL을 통하여 특정 파일의 속성정보와 파일 내용을 읽어오는 프로그램입니다. 우선 "http://www.kbs.co.kr/aboutkbs/audience.html"을 매개변수로 하여 URL 객체 kbs를 생성하고, kbs 객체를 매개변수로 하여 URLConnection 객체 kbsCon을 생성합니다. kbsCon 객체를 이용하여 파일의 속성정보를 읽습니다. 500바이트의 파일 내용을 읽기 위해 URLConnection 클래스에서 제공하는 getInputStream() 메소드를 이용하여 InputStream 객체 input을 생성합니다. 생성된 input 객체를 이용하여 연결된 파일의 내용을 200바이트 읽어들여 출력합니다. 이 프로그램은 지정된 URL이 실행 시 인터넷에서 유효하지 않을 경우 결과를 나타내지 않을 수도 있습니다.

예제 20.3 URLConnectionTest.java

```
01: import java.net.*;
02: import java.io.*;
03: public class URLConnectionTest {
04:     public static void main(String args[]) throws Exception {
05:         URL kbs = new URL
06:             ("http://www.kbs.co.kr/aboutkbs/history.html");
07:         URLConnection kbsCon = kbs.openConnection();   <-- URL 객체를 이용하여 URLConnection 객체 생성
08:         System.out.println
09:             ("문서의 타입: " + kbsCon.getContentType());
10:         System.out.println("=== 문서의 내용 ===");
11:         InputStream input = kbsCon.getInputStream();   <-- 입력을 위한 객체 생성
12:         int i = 500;
13:         int c;
14:         while (((c = input.read()) != -1) && (--i > 0)) {   <-- 입력된 내용을 출력
15:             System.out.print((char) c);
16:         }
17:         input.close();
18:     }
19: }
```

설명

05~07 URL 클래스와 URLConnection 클래스를 이용하여 객체를 생성한다.

11 URLConnection 클래스의 getInputStream( ) 메소드를 이용하여 InputStream 객체(문자 스트림)를 생성한다.

14 파일의 끝 또는 500바이트가 될 때까지 파일로부터 문자를 읽어들여 출력한다.

실행 결과

```
문서의 타입: text/html
=== 문서의 내용 ===
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=euc-kr">
<title>KBS¿ ¡ ´???¿?</title>
<link  href="http://img.kbs.co.kr/cms/aboutkbs/image/style.css"  rel="stylesheet"  type="text/
css">
<script language="JavaScript" type="text/JavaScript">
<!--
function MM_reloadPage(init) {  //reloads the window if Nav4 resized
  if (init==true) with (navigator) {if ((appName=="Netscape")&&(parseInt(appVersion)==4)) {
    document.MM_pgW=innerWidth; document.MM_pgH=innerHeight; onresi
```

SECTION 3

TCP 소켓

자바는 클라이언트-서버 응용 프로그램 작성을 위한 ServerSocket 클래스와 Socket 클래스를 제공하고 있습니다. 이 **소켓들은 상호 연결된 상태에서 신뢰성 있는 통신을 제공**합니다.

클라이언트-서버 환경에서의 서버 응용 프로그램은 항상 먼저 실행되어 클라이언트의 요청을 기다린다. 클라이언트에 의해 요청이 발생되면, 서버는 그 요청을 처리하고 그 결과를 클라이언트에 보내준다.

ServerSocket 클래스는 서버 측에서 실행되는 응용 프로그램 작성을 위해 사용됩니다.

[형식] ServerSocket 클래스의 생성자

```
ServerSocket(int port) throws IOException)
```

port : 요청을 받아들일 포트 번호

표 20-4 ServerSocket 클래스의 주요 메소드

메소드	설명
Socket accept( ) throws IOException	클라이언트의 요청을 받아들인 다음, Socket 클래스 객체를 반환합니다.
void close( ) throws IOException	서버 소켓을 닫습니다.

accept( ) 메소드는 클라이언트로부터의 연결 요청을 받아들인 다음 Socket 클래스의 객체를 생성하여 반환합니다. 즉 실제 클라이언트와 서버와의 통신은 클라이언트의 Socket 객체와 서버의 accept( ) 메소드에 의해 생성된 Socket 객체 사이에 이루어집니다.

ServerSocket 객체는 연결만 확립하고 실제 통신은 수행하지 않습니다. 서버는 데몬daemon 형태로 수행되며, 만일 클라이언트의 연결 요청이 없으면 ServerSocket 객체의 accept( ) 메소드는 연결 요청이 발생할 때까지 대기합니다.

Socket 클래스는 클라이언트와 서버 사이에 실질적인 정보 교환을 위해 사용합니다.

[형식] Socket 클래스의 생성자

```
Socket(String hostName, int port) throws UnknownHostException, IOException
```

hostName, port : 연결을 요청할 컴퓨터의 주소와 포트 번호

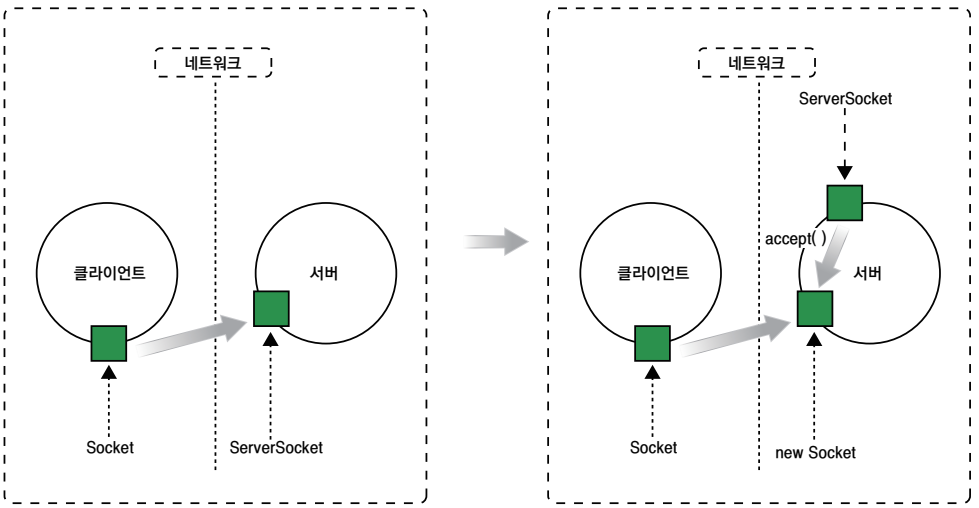

그림 20-3 TCP 소켓의 작동

소켓이 생성되고 나면 정보교환을 위한 입력 또는 출력 스트림을 생성하여야 합니다. Socket 클래스는 현재의 소켓을 통하여 입출력을 하기 위한 스트림 생성 메소드를 제공하고 있습니다.

표 20-5 Socket 클래스의 주요 메소드

메소드	설명
OutputStream getOutputStream( ) throws IOException	현재의 소켓과 관련된 OutputStream 객체를 반환
void close( ) throws IOException	소켓을 닫습니다.
InetAddress getInetAddress( )	현재 소켓에 연결된 컴퓨터의 주소를 반환
InetAddress getLocalAddress( )	현재 소켓을 사용하고 있는 컴퓨터의 주소를 반환
int getPort( )	현재 소켓에 연결된 컴퓨터의 포트번호를 반환

메소드	설명
int getLocalPort( )	현재 소켓이 사용하고 있는 포트번호를 반환
InputStream getInputStream( ) throws IOException	현재의 소켓과 관련된 InputStream 객체를 반환

getInputStream( ) 메소드와 getOutputStream( ) 메소드는 각각 InputStream, OutputStream 객체를 반환합니다. 이 객체들은 DataInputStream과 DataOutputStream 객체들을 생성하기 위해 사용하는 객체입니다(13장 입출력 참조).

SeverSockek과 Socket 클래스를 이용한 클라이언트–서버 간의 통신은 다음과 같은 순서로 이루어집니다.

서버 측

① 서버 소켓 객체를 ServerSocket 클래스로부터 생성한 다음 클라이언트의 요청을 기다립니(accept( ) 메소드).
② 클라이언트의 요청이 발생되면 요청을 받아들여 Socket 객체를 생성합니다.
③ Socket 객체를 이용하여 입출력을 위한 스트림을 생성합니다.
④ 통신을 수행합니다.
⑤ 소켓을 닫습니다.

클라이언트 측

① 연결을 원하는 서버의 주소와 포트번호로 Socket 객체를 생성합니다.
② Socket 객체를 이용하여 입출력을 위한 스트림을 생성합니다.
③ 통신을 수행합니다.
④ 소켓을 닫습니다.

실습 예제

이 프로그램은 서버와 클라이언트 사이의 통신 예를 보이고 있습니다. 서버는 클라이언트의 요청이 발생하면 1부터 10까지의 정수를 클라이언트 측에 보내고, 클라이언트는 이 데이터를 받아서 화면에 출력시켜주는 프로그램입니다. 이 프로그램은 실행 시 두 개의 매개 변숫값을 받아들인다. 첫 번째 매개변수는 통신이 이루어질 포트번호이고 두 번째 매개변수는 서버가 클라

이언트의 요청을 처리할 횟수입니다. 두 번째 매개변수의 값을 5로 지정하였다고 가정하면, 서버 프로그램은 5번의 클라이언트 요청을 처리하고 종료됩니다.

예제 20.4 ServerSide.java

```
01: import java.io.*;
02: import java.net.*;
03: class ServerSide {
04:    public static void main(String args[]) throws Exception {
05:       int port = Integer.parseInt(args[0]);
06:       int times = Integer.parseInt(args[1]);
07:       ServerSocket ss = new ServerSocket(port);
08:       int i = 1;
09:       while( i <= times) {
10:          Socket s = ss.accept();
11:          OutputStream os = s.getOutputStream();
12:          DataOutputStream dos = new DataOutputStream(os);
13:          for(int j = 1 ; j <= 10 ; j++)
14:             dos.writeInt(j);
15:          s.close();
16:          ++i;
17:       }
18:       ss.close();
19:    }
20: }
```

포트번호로 ServerSocket 객체 생성

지정된 횟수만큼 클라이언트 요청 처리

클라이언트 요청을 기다려 Socket 객체를 생성

기본 자료형 데이터 출력을 위한 객체 생성

정수를 출력

바이트 스트림 출력을 위한 객체를 Socket 객체로 생성

설명

07 프로그램 실행 시 지정된 포트번호로 ServerSocket 객체를 생성한다.

09 반복문에서는 두 번째 매개변수로 지정된 횟수만큼을 반복하면서 클라이언트 요청을 처리한다.

10 accept() 메소드는 클라이언트 요청이 발생할 때까지 대기하는 메소드이다. 지정된 포트를 통하여 클라이언트의 요청이 발생되면 이 메소드는 관련된 Socket 객체를 생성한다. accept() 메소드에서 요청이 발생하여 Socket 객체가 생성되었다는 의미는 포트를 통하여 클라이언트 컴퓨터와 연결이 확립되었다는 의미이다.

11 Socket 클래스의 getOutputStream() 메소드를 이용하여 OutputStream 객체를 생성한다. OutputStream 클래스는 바이트 단위의 입출력을 수행하는 클래스이다.

12 기본 자료형 데이터를 출력하기 위해 OutputStream 객체를 이용하여 DataOutputStream 객체를 생성한다.

14 1~10까지의 정수를 출력한다.

실행 결과

이 프로그램을 실행하면 서버 프로그램은 대기 상태가 됩니다. 지정된 횟수만큼의 클라이언트 요청을 처리하면 프로그램이 종료됩니다.

실습 예제

이 프로그램은 서버 프로그램에 요청을 수행하는 클라이언트 프로그램입니다. 클라이언트 프로그램이 실행되기 위해서는 서버 컴퓨터의 주소와 포트번호를 알아야 합니다. 클라이언트 프로그램은 실행 시 두 개의 값을 받아들인다. 클라이언트 프로그램은 서버의 주소와 포트를 가지고 연결을 확립하고, 서버에서 출력되는 10개의 정수 데이터를 받아들여 출력하는 프로그램입니다.

예제 20.5 ClientSide.java

```
01: import java.io.*;
02: import java.net.*;
03: public class ClientSide {
04:    public static void main(String args[]) throws Exception {
05:       String server = args[0];
06:       int port = Integer.parseInt(args[1]);
07:       Socket c = new Socket(server, port);
08:       InputStream is = c.getInputStream();
09:       DataInputStream dis = new DataInputStream(is);
10:       for(int i=1 ; i <= 10 ; i++) {
11:          int j = dis.readInt();
12:          System.out.println
13:             ("서버로부터 받은 데이터 " + j + " 출력");
14:       }
15:       c.close();
16:    }
17: }
```

서버의 주소와 포트번호로 Socket 객체 생성

정수를 입력받기 위한 스트림 객체 생성

정수를 읽는다

설명

07 프로그램 실행 시 입력받은 주소와 포트번호를 가지고 Socket 클래스의 객체를 생성한다.

08~09 서버 프로그램에서 출력되는 정보를 입력받기 위한 입력 스트림 객체를 생성한다.

11 입력 스트림으로부터 10개의 정수를 읽어들인다.

실행 결과

상대방 컴퓨터의 주소와 포트번호를 지정하여 프로그램을 실행하여 다음과 같은 결과를 출력하였습니다. 한 대의 컴퓨터에서 서버와 클라이언트 프로그램을 모두 실행시키는 경우, 컴퓨터의 주소로 127.0.0.1(자신의 컴퓨터를 의미) 또는 localhost로 지정하면 됩니다. 포트번호는 4자릿수의 포트번호를 자유롭게 사용하면 됩니다(예 1111).

```
서버로부터 받은 데이터 1 출력
서버로부터 받은 데이터 2 출력
서버로부터 받은 데이터 3 출력
서버로부터 받은 데이터 4 출력
서버로부터 받은 데이터 5 출력
서버로부터 받은 데이터 6 출력
서버로부터 받은 데이터 7 출력
서버로부터 받은 데이터 8 출력
서버로부터 받은 데이터 9 출력
서버로부터 받은 데이터 10 출력
```

SECTION 4

UDP 소켓

앞 절에서 살펴본 TCP 소켓은 상호 연결된 상태에서의 통신 방법입니다. 이 방법은 높은 신뢰성이 요구되는 응용 프로그램에 적합합니다. 그러나 높은 신뢰성이 요구되지 않거나 적은 양의 데이터 전송에는 네트워크에 많은 부담을 주는 TCP 소켓이 적합하지 않습니다.

UDPUser Datagram Protocol**는 TCP 소켓에 비해 신뢰성과 안정성 측면에서 뒤지지만, 연결을 설정하지 않으므로 네트워크에 부담을 주지 않는다는 장점이 있습니다.** 사용자는 응용 프로그램의 특성에 따라 적합한 소켓을 사용하여야 합니다.

자바에서는 UDP를 지원하는 DatagramPacket, DatagramSocket 클래스를 제공하고 있습니다. **DatagramPacket 클래스는 응용 프로그램들이 주고받을 데이터와 관련된 클래스**입니다. 실제 데이터의 전송은 DatagramSocket 클래스에 의해 이루어집니다.

[형식] DatagramPacket 클래스의 생성자

```
DatagramPacket(byte[ ] buffer, int size)
DatagramPacket(byte[ ] buffer, int size, InetAddress ia, int port)
```

buffer : 송수신될 데이터가 저장되어 있는 배열
size : 배열의 크기
ia : 상대방 컴퓨터의 주소(InetAddress 객체)
port : 상대방 컴퓨터의 포트 번호

첫 번째 생성자는 데이터를 수신하는 응용 프로그램에서 사용하는 생성자입니다. 두 번째 생성자는 데이터를 송신하는 응용 프로그램에서 사용하는 생성자입니다.

표 20-6 DatagramPacket 클래스의 주요 메소드

메소드	설명
InetAddress getAddress( )	수신 응용 프로그램에서 사용합니다. 정보를 보낸 컴퓨터의 주소를 반환
byte[ ] getData( )	패킷으로부터 데이터를 읽어들여 바이트 배열로 반환
int getLength( )	패킷의 바이트 수를 반환
int getPort( )	포트번호를 반환
void setAddress(InetAddress ia)	ia를 주소로 설정
void setData(byte buffer[ ])	buffer의 내용을 패킷의 데이터로 설정
void setLength(int size)	패킷의 크기를 size로 설정
void setPort(int pt)	포트번호를 pt 값으로 설정

DatagramSocket 클래스는 실제 정보를 주고받기 위한 기능을 제공하는 클래스입니다. 두 개의 생성자를 제공합니다.

[형식] DatagramPacket 클래스의 생성자

```
DatagramSocket() throws SocketException
DatagramSocket(int port) throws SocketException
```

port : 소켓이 사용할 포트번호.

표 20-7 DatagramSocket 클래스의 주요 메소드

메소드	설명
void receive(DatagramPacket dgram) throws IOException	현재의 소켓으로부터 정보를 읽어 dgram 패킷에 저장
void send(DatagramPacket dgram) throws IOException	현재의 소켓을 통하여 dgram 패킷을 전송
void close( ) throws IOException	소켓을 닫습니다.

TCP는 연결성 통신방법이기 때문에 전송 전에 반드시 연결이 설립되어야 합니다. 클라이언트는 소켓을 생성하여 연결을 요청하고, 서버는 accept() 메소드로 연결 요청을 받아들인다. 만일 클라이언트의 연결 요청에 서버가 응답하지 못한다면 접속거부connection refused 오류가 발생합니다.

UDP는 비연결성 통신 방법입니다. 통신을 위해 소켓을 사용하기는 하지만, 이 소켓(데이터 그램 소켓)은 상대 컴퓨터에 연결 요청을 하지 않습니다. 단순히 데이터를 전송하기 위한 소켓입니다. **UDP에서는 패킷에 데이터를 받을 컴퓨터의 주소와 포트번호를 기록한 다음 소켓을 통하여 전송**하면 됩니다. 패킷은 지정된 컴퓨터의 포트에 도착하여 데이터를 전달합니다. 패킷을 전송할 때마다, 패킷 내부에 목적지 주소와 포트번호를 명시하여야 합니다. UDP는 상대 컴퓨터에 연결 요청을 하지 않고 데이터를 보냄으로써 접속거부와 같은 오류가 발생하지 않습니다.

실습 예제

이 프로그램은 편지를 받는 프로그램입니다. TCP와는 달리 UDP에서는 연결을 확립하지 않습니다. 편지를 받을 포트와 연관된 DatagramSocket 객체를 생성하고, receive() 메소드를 수행하여 편지가 도착하기를 기다리면 됩니다. 다음 프로그램은 실행 시 두 개의 값을 지정받습니다. 첫 번째 값은 사용할 포트번호이고, 두 번째는 클라이언트로부터 받을 편지의 횟수입니다.

예제 20.6 UDPReceiver.java

```
01: import java.net.*;
02: public class UDPReceiver {
03:    public static void main(String args[]) throws Exception {
04:       int port = Integer.parseInt(args[0]);
05:       int times = Integer.parseInt(args[1]);
06:       DatagramSocket ds = new DatagramSocket(port);   ← 특정 포트를 지정하여 객체 생성
07:       int i=1 ;
08:       while( i <= times) {
09:          byte buffer[] = new byte[30];   ← 바이트 배열(30바이트 크기) 생성
10:          DatagramPacket dp =
11:             new DatagramPacket(buffer, buffer.length);   ← 패킷을 생성(30바이트 크기)
12:          ds.receive(dp);   ← 대기하다가 편지가 오면 패킷에 저장
```

```
13:             String str = new String(dp.getData()); <-------- 패킷에 저장된 데이터를 추출
14:             System.out.println("수신된 데이터 : " + str);
15:             ++i;
16:         }
17:     }
18: }
```

설명

06 컴퓨터에서 편지를 받을 특정 포트를 지정하여 DatagramSocket 객체를 생성한다. DatagramSocket 객체는 연결을 확립하지 않는다.

09 30바이트 크기의 바이트 배열을 생성한다. 이 프로그램에서는 편지의 길이가 30바이트 이내라고 가정하고 있다.

10, 11 30바이트 크기의 편지를 저장할 수 있는 DatagramPacket 객체를 생성한다. 패킷의 크기만을 가지고 DatagramPacket 객체를 생성하는 경우는 편지를 받는 측에서 사용하는 형태이다.

12 receive( ) 메소드를 수행하여 지정된 포트에 메시지가 도착하기를 기다린다. 메시지가 도착하면 receive( ) 메소드는 도착된 메시지를 매개변수로 받은 패킷(dp)에 저장한다.

13 getData( ) 메소드를 이용하여 패킷으로부터 내용을 추출한다.

실행 결과

```
UDPReceiver 프로그램에서 보낸 내용이 출력되었습니다.

수신된 데이터 : 자바는쉽다
수신된 데이터 : 아니다어렵다
수신된 데이터 : 자바는정말쉽다
수신된 데이터 : 열심히하면쉽다
```

실습 예제

이 프로그램은 편지를 보내는 프로그램입니다. TCP와는 달리 UDP에서는 연결을 확립하지 않습니다. 각각의 데이터를 보낼 때 주소와 포트를 패킷에 담아 전송하면 됩니다. 다음 프로그램은 실행 시 3개의 값을 지정받습니다. 첫 번째 값은 상대방 컴퓨터의 주소이고, 두 번째 값은 포트번호, 세 번째 값은 보낼 내용을 의미합니다.

예제 20.7 UDPSender.java

```
01: import java.net.*;
02: public class UDPSender {
03:    public static void main(String args[]) throws Exception {
04:       DatagramSocket ds = new DatagramSocket();
05:       InetAddress ia = InetAddress.getByName(args[0]);
06:       int port = Integer.parseInt(args[1]);
07:       byte buffer[] = args[2].getBytes();
08:       DatagramPacket dp =
09:          new DatagramPacket
10:            (buffer, buffer.length, ia, port);
11:       ds.send(dp);
12:    }
13: }
```

DatagramSocket 객체를 생성(주소 지정 안 함)

InetAddress 객체 생성 (주소를 가진 객체)

세 번째 매개변수의 값(편지내용)을 바이트 배열로 변환

패킷 객체 생성 (주소, 포트, 내용)

패킷을 보낸다

설명

04 DatagramSocket 객체를 생성한다. 객체 생성 시에 주소를 지정하지 않았다. send( ) 메시지를 이용하기 위한 객체 생성이다.

05~07 3개의 매개변수를 이용하여 주소 객체와 포트번호, 편지의 내용을 변수에 저장하였다. 주소는 InetAddress 객체로 포트번호는 정수로, 편지의 내용에 해당하는 내용은 바이트 배열로 작성하였다.

08, 09 DatagramPacket 객체를 생성하였다. 앞에서 마련된 주소와 포트, 편지내용을 가지고 패킷 객체를 생성하였다.

11 DatagramSocket 클래스의 send( ) 메소드를 이용하여 패킷 객체를 전송하였다. 패킷 객체에 있는 주소와 포트로 네트워크를 통하여 찾아가게 된다.

실행 결과

위 프로그램을 한 대의 컴퓨터에서 실행시키기 위해 다음과 같이 3개의 매개변수를 지정하였습니다. 4번을 실행하였습니다.

```
127.0.0.1 1111 자바는쉽다
127.0.0.1 1111 아니다어렵다
127.0.0.1 1111 자바는정말쉽다
127.0.0.1 1111 열심히하면쉽다
```

학/습/정/리

네트워킹의 개요와 java.net 패키지

① 자바는 네트워킹 관련 클래스들을 java.net 패키지로 제공하고 있습니다.

② 인터넷에서는 TCP/IP 프로토콜을 표준 프로토콜로 사용하고 있습니다.

③ 통신 방법에는 연결성 통신 방법인 TCP 방법과 비연결성 통신 방법인 UDP 방법이 있습니다. 소켓은 컴퓨터가 네트워크에 연결된 끝점을 의미합니다.

④ 포트는 하나의 컴퓨터에 여러 개 존재하는 논리적인 개념입니다. 서로 다른 포트를 이용하여 컴퓨터에 서로 다른 일을 시킬 수 있습니다.

인터넷 주소와 URL

① 인터넷의 주소에는 컴퓨터 상호 간에 사용하는 숫자로 구성된 IP(Internet Protocol) 주소가 있고(예 134.23.33.200), 우리가 많이 사용하는 문자 중심의 도메인(Domain) 이름(예 www.hollywood.com)이 있습니다.

② InetAddress 클래스는 인터넷의 주소 정보를 가진 클래스입니다.

③ URL 클래스와 URLConnection 클래스를 이용하여 인터넷으로부터 정보를 얻어낼 수 있습니다.

TCP 소켓

① 자바는 클라이언트-서버 응용 프로그램 작성을 위한 ServerSocket 클래스와 Socket 클래스를 제공하고 있습니다.

② TCP 소켓은 상호 연결된 상태에서 신뢰성 있는 통신을 제공합니다.

③ ServerSocket 클래스는 서버 측에서 사용되며 실제 통신을 수행하지 않습니다. 실제 통신은 Socket 객체에 의해 이루어집니다.

UDP 소켓

① UDP(User Datagram Protocol)는 TCP 소켓에 비해 신뢰성과 안정성 측면에서 뒤지지만, 연결을 설정하지 않으므로 네트워크에 부담을 주지 않는다는 장점이 있습니다.

② 자바는 UDP를 지원하기 위해 DatagramPacket, DatagramSocket 클래스를 제공하고 있습니다.

③ DatagramPacket 클래스는 응용 프로그램들이 주고받을 데이터와 관련된 클래스이고, 실제 데이터의 전송은 DatagramSocket 클래스에 의해 이루어집니다.