Java 수업교안

1. 자바의 개요 및 개발환경 구축

▶ 첫 번째 예제

- com.javastudy.ch01

```
/* 첫 번째 자바 클래스
 * 자바의 모든 실행 코드는 클래스 안(클래스의 { } 블록 안에)에 작성해야 한다.
 **/
public class FirstJavaClass {

    public static void main(String[] args) {

        /* 자바에서 변수를 만들기 위해서는 그 변수에 담을 데이터 타입을 먼저
         * 변수 앞에 지정하고 변수를 만들어야 하는데 이를 변수의 선언이라고 한다.
         * 정수형 변수 x, y를 선언하고 30과 50의 정수를 대입(저장)하고 있다.
         **/
        int x = 30;
        int y = 50;

        // 문자형 변수 ch를 선언하고 A를 대입하고 있다.
        char ch = 'A';

        // 문자열 형 변수 hello를 선언하고 Hello Java를 대입하고 있다.
        String hello = "Hello Java";

        /* ch 변수에 저장된 데이터를 콘솔(화면)에 출력
         * System.out.println() 메서드는 인수로 지정한 데이터를 콘솔에 출력하는 메서드 이다.
         **/
        System.out.println(ch);

        // hello 변수의 내용을 콘솔로 출력하기
        System.out.println(hello);

        // 정수형 변수에 저장된 x의 값과 y의 값을 더해서 콘솔에 출력하고 있다.
        System.out.println("30 + 50 = " + (x + y));
    }
}
```

2. 자바 기본 프로그래밍(데이터 타입과 연산자)

2.1 자료형

자료형(Data Type)은 프로그래밍에서 사용되는 숫자 자료, 문자 자료 등과 같은 자료의 형태를 의미하며 프로그램 언어에서 가장 기본이 되는 단위가 자료형 이다. 그러므로 프로그래밍 언어를 공부할 때는 해당 언어에 대한 자료형을 충분히 이해하고 있어야 된다.
자바에서 사용되는 자료형은 숫자와 같이 단일 값으로 되어 있는 기본(원시, Primitive) 자료형과 복합 데이터를 클래스(Class)로 정의해 하나의 자료형으로 만들어 사용하는 객체(Object) 자료형이 있다. 참고로 객체 자료형은 참조(Reference) 자료형이라고도 한다.
자바의 기본 자료형은 숫자, 문자(char), 불(boolean) 등의 기본 자료형이 있으며 숫자 자료형은 정수와 실수 자료형으로 구분해 사용된다. 정수 자료형에는 표현할 수 있는 숫자의 범위에 따라서 byte, short, int, long과 같은 4가지 정수 자료형이 있으며 실수 자료형에는 float, double과 같은 2가지 자료형이 있다. 그러므로 자바의 기본 자료형에는 숫자 자료형 6가지와 한 문자를 표현할 수 있는 문자(char) 자료형, 그리고 참과 거짓의 정보를 표현할 수 있는 논리(불, boolean) 자료형으로 구성되어 있어 모두 8가지 존재한다.
객체 자료형은 이미 정의되어 있는 기본 자료형과 다르게 개발자가 클래스를 정의해 자료형을 만들 수 있으므로 필요해 의해서 다양한 클래스가 수많은 개발자에 의해서 정의될 수 있기 때문에 그 수가 몇 개인지 셀 수가 없다.

▶ 변수 선언과 문장의 종료

- com.javastudy.ch02.datatype

```
public class SecondJavaClass {

    // 자바에서 main() 메서드가 있는 클래스가 어플리케이션의 실행 진입점이 된다.
    public static void main(String[] args) {

        /* 변수는 프로그램 수행 중에 데이터를 담아두는 상자라 할 수 있다. 어떤
         * 프로그램이 실행되면 그 프로그램 안의 변수는 컴퓨터의 메인 메모리에
         * 만들어지는데 이렇게 만들어진 메모리 공간의 이름이 바로 변수인 것이다.
         * 프로그램에서는 변수를 통해 메모리에 접근하여 데이터를 저장하거나
         * 가져올 수 있고 변수는 하나의 값만 저장할 수 있으며 프로그램 수행 중에
         * 값이 변경될 수 있기 때문에 변수라고 부른다.
         *
         * 자바에서 변수를 만들기 위해서는 그 변수에 담을 데이터 타입을 먼저
         * 변수 앞에 지정하고 변수를 사용해야 하는데 이를 변수의 선언이라고 한다.
         * 변수의 데이터 타입을 변수명 앞에 지정하지 않으면 컴파일시 에러가 발생한다.
         * 이렇게 컴파일시 변수의 타입을 체크하는 언어를 강 타입의 언어라고도 한다.
         *
         * 변수와 같이 데이터를 구분하기 위해 사용하는 이름을 식별자(Identifier)라
         * 하며 자바에서 식별자를 지정할 때 꼭 지켜야 하는 규칙이 있다.
         * 식별자는 문자, 숫자, 2개의 특수문자($, _)를 사용해 명명할 수 있으며
         * 식별자의 첫 문자는 숫자를 사용할 수 없고 반드시 문자나 2개의 특수문자로
```

```
     * 시작해야 한다. 변수도 식별자이므로 이 규칙을 따라 이름을 지어야 한다.
     * 또한 자바에서 사용하는 예약어는 식별자로 사용할 수 없다.
     *
     * 자바는 유니코드를 지원하는 언어로 여러 나라의 다양한 문자를 변수명으로
     * 사용할 수 있기 때문에 한글을 사용해 변수명을 지정할 수 있지만 주로 영문을
     * 사용한다. 또한 대문자와 소문자를 구분하므로 num, Num, NUM은 모두
     * 다른 식별자로 구분되기 때문에 대소문자 사용에 주의를 기울여야 한다.
     *
     * 적합한 변수명 : $harp, _7up, $ession, seven11
     * 부적합한 변수명 : s#arp, #harp, 7up, &percent, 11, ?uestion
     * */

    /* 자바에서 식별자를 명명하는 규칙(문법)으로 지정된 것은 아니지만
     * 주로 아래와 같은 표기법을 사용해 식별자를 지정하는 것이 관례이다.
     *
     * ◆ 클래스, 인터페이스, 생성자
     *      파스칼 표기법(Pascal Casing) 사용 -> MemberService
     *
     * ◆   메서드, 멤버 필드(변수)
     *     카멜 표기법(Camel Casing) 사용 -> memberInfo
     *
     * ◆ 상수
     *    전체 대문자로 표기하고 단어와 단어는 _로 구분 -> DEFAULT_STATE
     *
     * ◆ 윈도우 GUI 프로그램 - Swing 등과 같은 화면 컨트롤
     *    헝가리언 표기법(Hungarian Casing) -> txtName, lblAge
     *
     * ◆ 패키지
     *    전체 소문자로 표기 -> java.lang, com.javastudy.ch02
     **/

    /* int형 변수를 선언하고 동시에 값을 할당하고 있다.
     * 자바에서는 하나의 문장이 끝나면 세미콜론(;)으로 문장의 종료를 알린다.
     *
     * 아래와 같이 변수의 우측에 "=" 연산자를 사용해 값을 지정하는 것을 변수에
     * 값을 "할당 한다." 또는 "대입 한다."라고 말하며 변수를 선언하고 그 변수에
     * 값을 최초로 할당하는 것을 변수의 초기화라고 말한다.
     **/
    int x = 35;
    int y = 11;

    // x와 y의 값을 뺄셈과 나눗셈 연산을 수행하고 콘솔에 출력
    System.out.println(x + " - " + y + " = " + (x - y));
    System.out.println(x + " / " + y + " = " + (x / y));
}
```

```
}
```

▶ 변수 선언과 주석문 사용하기

- com.javastudy.ch02.datatype

```
public class VariableDeclaration {
    public static void main(String[] args) {

        // 여기는 자바의 한 줄 주석으로 주석은 프로그램 코드로 해석되지 않는다.
        // 변수만 선언하는 경우
        int a, b;
        float f;

        /* 여기는 자바의 여러 줄 주석
         * 아래는 변수 선언과 초기화를 동시에 하는 경우이다.
         * char 형은 자바 기본형 8개 중 하나로 한 문자를 저장할 수 있는 데이터 형이다.
         **/
        char ch = '한';

        // 변수에 처음 데이터를 할당(저장)하는 것을 변수의 초기화라 한다.
        a = 10;
        b = 100;
        f = 10.532F;

        // 각 변수에 저장된 데이터를 콘솔로 출력하고 있다.
        System.out.println(a + " : " + b);
        System.out.println(f);
        System.out.println(ch);

        /* 위에서 a에 10을 할당한 후 아래와 같이 변수 b를 할당하면 기존 a에
         * 있던 값 10은 지워지고 최종적으로 할당한 b의 값 100이 저장된다.
         * 이렇게 변수에는 최종적으로 할당된 값만 저장된다.
         **/
        a = b;
        System.out.println("a : " + a);

        // 변수 b가 보유한 값 100에 10을 더하여 110이 최종적으로 b에 저장된다.
        b = b + 10;
        System.out.println("b : " + b);
    }
}
```

▶ 정수형 데이터 사용하기 1

- com.javastudy.ch02.datatype

```
public class IntNum01 {

    /* 정수 자료형은 음수, 0, 양수를 나타내는 수치 자료형으로 자바에서는 byte, short,
     * int, long형 등의 4가지 자료형을 제공하고 있다. 각각의 자료형은 메모리에 저장되는
     * 크기가 다르며 부호가 있는 정수를 다루기 때문에 표현할 수 있는 수의 범위가 다르다
     * 컴퓨터 내부에서 수를 표현할 때는 2진수 0과 1로 표현되며 부호가 있는 수를 2진
     * 비트로 표현하기 위해서 맨 앞의 첫 번째 비트를 부호 비트로 사용하며 이 부호 비트가
     * 0이면 양수를 나타내고 1이면 음수를 나타낸다.
     * 자바의 4가지 정수 자료형이 표현할 수 있는 수의 범위는 아래와 같다.
     *
     * byte     : 1Byte(-2의 7승 ~ 2의 7승 - 1)
     * short    : 2Byte(-2의 15승 ~ 2의 15승 - 1)
     * int      : 4Byte(-2의 31승 ~ 2의 31승 - 1)
     * long     : 8Byte(-2의 63승 ~ 2의 63승 - 1)
     * */
    public static void main(String[] args) {

        // int형 변수를 먼저 선언하고 각 변수에 정수 값을 할당하고 있다.
        int a, b;
        a = 300;
        b = 200;

        // 변수에 저장된 값을 연산자를 이용해 계산하여 콘솔로 출력하고 있다.
        System.out.println("a + b = " + (a + b));
        System.out.println("a * b = " + (a * b));
    }
}
```

▶ 정수형 데이터 사용하기 2

- com.javastudy.ch02.datatype

```
public class IntNum02 {

    public static void main(String[] args) {

        // short형과 int형 변수를 선언하고 동시에 값을 할당
        short s = 65;
        int i = 137;

        /* 정수형의 기본형은 int 형이므로 long형 데이터를 사용할 때는
         * 숫자 뒤에 L 또는 l을 붙여서 long형 데이터임을 표현한다.
         **/
        long l = 574L;
```

```
        /* int형 변수에 저장된 데이터를 short형 변수에 저장된 데이터로
         * 뺄셈하여 그 결과를 콘솔로 출력하기
         **/
        System.out.println("137 - 65 = " + (i - s));

        /* long형 변수에 저장된 데이터를 int형 변수에 저장된 데이터로
         * 나누어 그 결과를 콘솔로 출력하기
         **/
        System.out.println("574 / 137 = " + l / i);
    }
}
```

▶ 정수형 변수 간의 할당

- com.javastudy.ch02.datatype

```
public class IntNum03 {
    public static void main(String[] args) {

        int intNum01 = 30;
        long longNum01 = 100L;

        /* 큰 데이터 형으로 정의된 변수를 작은 데이터 형의 변수에 할당하는 것은 큰 수를
         * 작은 그릇에 담는 것이므로 명시적으로 형 변환이 필요하다. 이렇게 작은 그릇에
         * 큰 수를 담기 위해서 수의 크기를 작게 조절할 필요가 있기 때문에 작은 쪽으로
         * 형 변환하라고 명시적으로 알려줘야 한다. 이렇게 작은 쪽으로 데이터 형을 변환하는
         * 것을 명시적 형 변환 또는 강제 형 변환이라고 한다. 강제로 형 변환을 하게 되면
         * 본래의 수가 손실될 수도 있다.
         **/
        // int intNum02 = longNum01; // 오류 발생
        int intNum02 = (int)longNum01;

        /* 작은 데이터 형으로 정의된 변수를 큰 데이터 형의 변수에 할당할 때는 작은 수를
         * 큰 그릇에 담는 것이므로 문제되지 않으며 자동으로 형 변환 된다. 이렇게 자동으로
         * 형 변환되는 것을 자동 형 변환 또는 묵시적 형 변환이라고 한다.
         **/
        long longNum02 = intNum01;

        /* long형 데이터와 int형 데이터의 곱셈 연산
         * long형 데이터와 int형 데이터를 연산하면 먼저 두 변수의 형을 동일하게 맞추고
         * 연산을 하게 되는데 이 때 int형 변수를 long형으로 형 변환 한 후 두 수를 곱셈한다.
         * 즉 큰 데이터 형과 작은 데이터 형을 연산하게 되면 두 수의 형을 맞추기 위해 작은
         * 데이터 형을 큰 데이터 형으로 자동 형 변환 한 후 연산을 하게 된다.
         **/
```

```
        long multipleNum = longNum01 * intNum01;

        // long형과 int형을 곱셈한 결과를 콘솔에 출력
        System.out.println("longNum01 * intNum01 = " + multipleNum);

        // long형 데이터와 int형 데이터를 나눗셈하여 결과를 콘솔 출력
        System.out.println("longNum01 / intNum01 = " + longNum01 / intNum01);
    }
}
```

▶ 실수형 데이터 사용하기

- com.javastudy.ch02.datatype

```
public class RealNum01 {

    /* 실수 자료형은 소수점이 있는 수치 데이터를 표현하는 자료형으로
     * 컴퓨터에서 실수를 표현하기 위해서 일반적으로 부동소수점 방식을 사용한다.
     * 부동소수점 방식은 가수 부분과 지수 부분을 나누어 표현하는 방식으로 예를 들어
     * 0.5를 부동소수점 방식으로 표현하면 5.0 x 10의 -1승으로 표현할 수 있으며
     * 여기서 가수부 5.0과 지수부 10의 -1을 나눠서 실수를 표현하는 방식이다.
     * 부동소수점 방식을 사용하면 같은 메모리 크기에서 보다 많은 실수를 표현할 수 있다.
     * 실수도 부호가 있는 수를 표현해야 하므로 첫 번째 비트를 부호 비트로 사용하며
     * 정수 표현과 마찬가지로 부호 비트가 0이면 양수를 1이면 음수를 나타낸다.
     *
     * 자바에서 실수를 표현하기 위해 float과 double 두 가지 자료형을 제공하고 있다.
     * 두 가지 자료형은 메모리에 저장되는 크기가 다르며 표현할 수 있는 수의 범위와
     * 정밀도가 다르다. 실수 자료형이 표현할 수 있는 수의 범위는 다음과 같다.
     *
     * float     : 4Byte(부호 1비트, 지수부 8비트, 가수부 23비트)
     * double    : 8Byte(부호 1비트, 지수부 11비트, 가수부 52비트)
     **/
    public static void main(String[] args) {

        /* 실수형 변수를 선언하고 동시에 데이터를 저장
         * 실수형의 기본 데이터 형은 double 형 이므로 float 형을 표현할 때는
         * 숫자 뒤에 F 또는 f를 붙여서 float 형임을 명시적으로 표현해야 한다.
         **/
        float f = 10.0F;
        float f2 = 10.35F;

        /* double 형도 숫자 뒤에 d 또는 D를 붙여 double 형임을 명시적으로 표현해도
         * 되지만 실수형의 기본이 double 형이므로 생략해도 double 형 데이터가 된다.
         **/
        double d = 20.5;
```

```
        double d2 = 5.0e3D;

        // 실수형 데이터를 더하거나 곱셈한 후 콘솔로 출력하기
        System.out.println("f + d = " + f + d);
        System.out.println("d2 * d = " + d2 * d);
    }
}
```

▶ float 형과 double 형의 정밀도 비교

- com.javastudy.ch02.datatype

```
public class RealNum02 {

    public static void main(String[] args) {

        float f = 0.763F;
        float f1 = 0.763f;
        double d = 0.763d;

        /* double형과 float형을 연산하게 되면 먼저 두 수의 형을
         * 동일하게 맞춘 후에 실제 연산이 이루어지는데 이 때 float형의
         * 변수가 double형으로 자동 형 변환 된 후 연산을 하게 된다.
         *
         * float과 double은 수의 표현 범위가 다르므로 소수 자리를 표현하는 범위도 다르다.
         **/
        System.out.println(f + " / " + f1 + "(f / f1) = " + (f / f1));
        System.out.println(f + " / " + d + "(f / d) = " + (f / d));
        System.out.println();
        System.out.println((float)(9.9 / 3));
        System.out.println(9.9 / 3);
    }
}
```

▶ 문자형(char) 데이터 사용하기

- com.javastudy.ch02.datatype

```
public class CharLiteral01 {

    public static void main(String[] args) {

        /* 많은 프로그램 언어에서 리터럴(Literal)이라는 단어가 많이 사용되는데
         * 리터럴이란 코드에서 사용되는 고정된 값 자체를 의미하는 것으로 변수에 저장되는
         * 정수, 실수, 논리 값, 문자, 문자열 데이터 자체를 의미 한다. 다시 말해 리터럴은
         * 아래와 같이 변수 또는 상수에 직접 할당되는 데이터 자체를 말하는 것으로 변수에
```

```
         * 정수가 할당되면 정수 리터럴, 문자열이 할당 되면 문자열 리터럴 등으로 부른다.
         * 리터럴은 데이터를 표현하는 값 자체이므로 변하지 않기 때문에 상수의 의미를 가진다.
         **/
        int i = 100;
        String str = "안녕 자바";

        /* 전 세계 문자를 컴퓨터에서 표현하기 위해서 유니코드라는 표준 문자 체계가 만들어
         * 졌는데 이 유니코드는 문자 하나하나에 코드 값을 매겨 수치 데이터로 저장된다.
         * 자바는 유니코드 문자 하나를 저장할 수 있는 2byte의 char 형을 제공하고 있다.
         * 유니코드와 같이 표준화된 문자 체계를 문자 셋이라고도 부른다.
         **/
        char ch1 = '자';
        char ch2 = '바';

        /* int형 변수를 char형으로 강제 형 변환하면 유니코드 문자를 얻을 수 있다.
         * 영문 대문자 A는 유니코드 값 65이며 소문자 a는 유니코드 값 97 이다.
         **/
        char chA = (char) 65;
        char cha = (char) 97;
        System.out.println(chA + " - " + cha);

        // 유니코드의 첫 번째 문자는 공백 문자이며 char형의 기본 값이다.
        char hexaCh1 = '\u0000';

        /* 한글은 초성 19개, 중성 21개, 종성 28개를 조합해
         * 19 x 21 x 28 = 11,172 자의 완성형 한글이 유니코드에 등록되어 있다.
         * 유니코드에서 한글은 AC00(44032) ~ D7A3(55203)의 범위에 지정되어 있다.
         **/
        // 0x로 시작하는 정수 데이터는 16진수로 표현하는 정수를 의미한다.
        int hanStart = 0xAC00;

        /* 자바에서 char 형은 데이터가 저장될 때 내부적으로 정수로
         * 변환되어 저장되기 때문에 사칙 연산자를 이용해 산술연산이 가능하다.
         **/
        System.out.println(ch1 + ch2);
        System.out.println("유니코드 첫 문자 - hexaCh1 : " + hexaCh1);

        /* 다음과 같이 유니코드 범위에 있는 정수(0 ~ 65535)를 char 형으로
         * 형 변환 하면 그 문자 코드 값에 해당하는 유니코드 문자를 확인할 수 있다.
         **/
        System.out.println("유니코드 한글 첫 문자 : " + (char) hanStart);
        System.out.println("유니코드 한글 마지막 문자 : " + (char) 55203);
    }
}
```

▶ 상수 사용하기

- com.javastudy.ch02.datatype

```
public class JavaConstant01 {

    /* 상수도 변수와 마찬가지로 데이터가 저장되는 메모리 공간의 이름으로
     * 상수에 저장할 데이터 타입과 함께 선언해야 하며 아래 코드와 같이
     * 데이터 타입 앞에 final이라는 예약어를 사용해 상수를 선언할 수 있다.
     * 변수는 프로그램 중에 값이 변경될 수 있지만 상수는 한번 값이 지정되면
     * 프로그램이 실행되는 동안 그 값을 변경할 수 없다.
     **/

    // 상수는 초기화 이후 값을 변경할 수 없다.
    private static final int DEFAULT_NUM = 10;
    private static final String DEFAULT_NAME = "홍길동";

    public static void main(String[] args) {

        // 상수에 새로운 값을 할당 하면 에러 발생
        // DEFAULT_COUNT = 10;

        System.out.println(DEFAULT_NUM);
        System.out.println(DEFAULT_NAME);
    }
}
```

▶ 기본 타입의 자동 형 변환과 강제 형 변환

- com.javastudy.ch02.datatype

```
public class PrimitiveCasting01 {

    public static void main(String[] args) {

        byte b = 6;
        int i = 100;
        float f = 6.0f;
        double d = 6.0;

        /* byte 타입과 int 타입의 연산은 연산하기 전에 두 변수의 타입을 맞추기 위해
         * byte 타입의 변수 b의 값을 큰 쪽인 int 타입으로 자동으로 형 변환 한 후
         * 두 변수 b와 i의 값을 연산하게 되며 그 결과 값 또한 int 형이 된다.
         **/
        System.out.println("i x b = " + i * b);
        System.out.println();
```

```
        /* 정수형과 실수형 연산은 정수형 변수를 자동으로 실수형 타입으로 형 변환
         * 한 후 두 변수 i와 f의 값을 연산하게 되며 그 결과 값 또한 실수형이 된다.
         **/
        System.out.println("i / d = " + i / d);
        System.out.println();

        /* long 형의 크기는 8Byte 이고 float 형의 크기는 4Byte 이지만
         * float 형이 수의 표현 범위가 넓기 때문에 float 형을 long 형의
         * 변수에 담기 위해서는 강제 형 변환이 필요하다. 그 반대의 경우는 float 형이
         * long 형보다 수의 표현 범위가 넓기 때문에 자동 형 변환이 이루어진다.
         * 기본형의 수의 표현 범위를 작은 것부터 큰 순으로 나열하면 아래와 같다.
         * byte -> short/char -> int -> long -> float -> double
         **/
        long l = (long) f;
        f = l;
    }
}
```

▶ Scanner 클래스를 이용해 키보드 입력 처리하기

- com.javastudy.ch02.datatype

```
public class Scanner01 {

    public static void main(String[] args) {

        /* Scanner 클래스의 next() 메서드는 사용자가 입력한 키의 값을 공백(' ', \t, \f,
         * \r, \n)을 기준으로 구분하여 한 단위로 분리하고 분리된 한 단위씩 데이터를
         * 읽어 들인다. 또한 데이터가 입력되지 않은 상태(아무것도 입력되지 않은 상태,
         * 공백 문자만 입력되었을 때 모두 해당됨)에서 Enter 키가 눌려져도
         * 다음 단계로 넘어가지 않고 데이터가 입력 될 때까지 대기하는 특징이 있다.
         * 참고로 Scanner 클래스에는 nextLine() 메서드가 있는데 이 메서드는
         * 한 줄 단위로 데이터를 읽어오기 때문에 대기하지 않고 다음 단계로 넘어간다.
         **/
        Scanner scanner = new Scanner(System.in);

        /* next()는 공백으로 분리된 단위로 데이터를 읽기 때문에 "홍길동 11"을
         * 한 줄에 입력하면 "홍길동"은 이름을 입력받는 첫 번째 next()가 읽고
         * 나이를 입력받는 두 번째 next()는 두 번째 데이터가 이미 입력되었기
         * 때문에 대기하지 않고 바로 11을 읽어 변수에 저장한다.
         * */
        System.out.print("이름을 입력해 주세요 : ");
        String name = scanner.next();
```

```
        System.out.print("나이를 입력해 주세요 : ");
        String age = scanner.next();

        System.out.println("안녕하세요~ " + name + "님!");
        System.out.println("당신의 나이는 " + age + "세 군요 ^_^");

        // Scanner 사용이 끝나면 자원을 해제한다.
        scanner.close();
    }
}
```

2.2 연산자

▶ 나머지 연산자 사용하기

- com.javastudy.ch02.operator

```
public class Operator01 {

    public static void main(String[] args) {

        // 정수형 나눗셈 연산과 나머지 연산의 결과는 정수형이 된다.
        int x = 10;
        int y = 3;
        System.out.println("10 / 3의 몫 : " + x / y);
        System.out.println("10 / 3의 나머지 : " + x % y);
        System.out.println();


        // 실수형 나눗셈 연산과 나머지 연산의 결과는 실수형이 된다.
        float f1 = 10.0f;
        float f2 = 3.0f;
        System.out.println("10.0 / 3.0의 몫 : " + f1 / f2);
        System.out.println("10.0 / 3.0의 나머지 : " + f1 % f2);
        System.out.println();

        // char형 나눗셈 연산과 나머지 연산의 결과는 int 형이 된다.
        char chA = 'A';
        char cha = 'a';
        System.out.println("a / A의 몫 : " + cha / chA);
        System.out.println("a / A의 나머지 : " + cha % chA);
    }
}
```

▶ 비교 연산자와 논리 연산자 사용하기

- com.javastudy.ch02.operator

```
public class Operator02 {

    public static void main(String[] args) {

        int x = 10;
        int y = 30;
        int z = 10;

        // 피연산자에 저장된 값의 크기를 비교하는 연산자를 대소 비교 연산자라고 한다.
        System.out.println("10 > 30 : " + (x > y));
        System.out.println("10 <= 30 : " + (x <= y));
        System.out.println();

        // 피연산자에 저장된 값이 같은지 다른지를 비교하는 연산자를 등가 비교 연산자라 한다.
        System.out.println("10 == 10 : " + (x == z));
        System.out.println("10 != 10 : " + (x != z));
        System.out.println();

        /* 논리 연산자는 양쪽의 피연산자 모두가 boolean(true, flase) 값이어야 한다.
         * &&(AND) 연산자는 양쪽의 값이 모두 true일 때 true를 반환
         **/
        System.out.println("10 > 30 && 30 > 10 : " + ((x > y) && (y > z)));

        // ||(OR) 연산자는 한 쪽의 값이 true 이거나 양쪽 모두 true일 때 true를 반환
        System.out.println("10 < 30 || 30 > 10 : " + ((x < y) || (y > z)));

        // ^(Exclusive OR) 연산자는 양쪽의 값이 서로 다르면 true를 반환
        System.out.println("10 > 30 ^ 30 > 10 : " + ((x > y) ^ (y > z)));
        System.out.println("10 < 30 ^ 30 > 10 : " + (x < y ^ y > z));
        System.out.println();

        // 논리 부정 연산자는 논리 값을 반전시킴 - boolean 형에만 사용할 수 있는 연산자
        boolean isChecked = false;
        System.out.println("!isChecked : " + !isChecked);
    }
}
```

▶ 문자열 데이터 비교하기

- com.javastudy.ch02.operator

```
public class Operator03 {
```

```
public static void main(String[] args) {

    /* 자바에서 문자열은 객체로 다뤄지며 문자열 처리를 위해서 String 클래스를
     * 제공하고 있다. 문자열은 프로그램에서 숫자만큼이나 많이 사용되는 데이터로
     * 아래와 같이 문자열 리터럴을 통해 String 객체를 생성할 수 있도록 지원한다.
     *
     * 아래는 "Hello Java"라는 문자열 리터럴로 String 객체를 생성하여
     * 객체가 생성된 메모리 주소의 참조 값을 String 타입의 str1 변수에 저장한다.
     * str1에 객체가 생성된 메모리 주소의 참조 값이 저장되므로 변수 str1을
     * 참조형 변수라고 부른다.
     *
     * 자바에서 변수는 8개의 기본 형 데이터를 저장하는 기본형 변수와 객체가 저장된
     * 메모리 주소의 참조 값이 저장되는 참조형 변수로 나눌 수 있으며 기본형 변수의
     * 크기는 기본형 데이터의 크기와 동일하고 모든 참조형 변수는 객체가 생성된
     * 메모리 주소의 참조 값을 4Byte 크기로 저장한다.
     **/
    String str1 = "Hello Java";

    /* 프로그램에서 null 값은 자주 사용되는데 null이 의미하는 것은 현재 값이 정해져
     * 있지 않은 미정의 값 또는 값이 없음을 의미하기도 한다. null 값은 참조형 변수에만
     * 사용할 있고 기본형 변수에는 null 값을 사용할 수 없다. 참조형 변수에 null 값을
     * 할당 하면 그 변수가 어떠한 객체도 참조하고 있지 않음을 의미하며 아래는 String
     * 타입의 참조형 변수 str2를 선언하고 null로 초기화 하는 예이다.
     **/
    String str2 = null;

    /* 문자열 리터럴 비교하기
     * 앞에서도 언급했듯이 리터럴은 상수의 개념으로 문자열 리터럴 또한 상수이며
     * JVM에서 상수를 저장하고 관리하는 Constant Pool이라는 메모리 공간에
     * 저장된다. String 타입의 변수에 문자열 리터럴을 할당하게 되면 Constant Pool에
     * 동일한 문자열이 존재하는지를 먼저 검사한 후 동일한 문자열이 존재하면
     * 그 문자열을 참조하게 되고 동일한 문자열이 존재하지 않으면 지정한 문자열을
     * Constant Pool에 새로 등록하고 그 주소의 참조 값을 변수에 저장 한다.
     **/
    str2 = "Hello Java";
    System.out.println("str1 == str2 : " + (str1 == str2));

    /* String은 참조타입으로 데이터가 저장된 주소의 참조 값을 변수에 저장 한다.
     * 참조타입 변수의 "==" 연산은 변수에 저장된 참조 값이 같은지를 비교한다.
     **/
    String str3 = new String("Hello Java");
    System.out.println("str1 == str3 : " + (str1 == str3));

    /* 두 String 객체의 내용이 같은지를 비교하려면 String 클래스가
```

```
        * 제공하는 equals() 메서드를 사용해 아래와 같이 비교하면 된다.
        **/
        System.out.println("str.equals(str3) : " + str1.equals(str3));
    }
}
```

[연습문제 2-1]

Scanner 클래스를 이용해 태어난 년도를 입력받아 나이를 계산해 출력해 주는 다음과 같이 실행되는 프로그램을 작성하시오.

먼저 "JavaStudyCh02Exercise" 프로젝트를 만들고 "com.javastudy.ch02.operator" 패키지를 만들어 이 패키지에 클래스를 생성하여 프로그램을 작성하시오.

[실행결과] - 나이는 현재 년도에 따라 달라질 수 있음

태어난 년도를 4자리로 입력해주세요 :

1991

당신은 1991년 생으로 현재 30세 입니다.

[연습문제 2-2]

다음 요구사항에 따라 변수를 선언하고 그에 맞는 프로그램을 작성하시오

1. 정수형 변수 x, y를 선언과 동시에 각각 15와 7로 초기화 하시오.
2. 실수형(float) 변수 f1, f2를 먼저 선언하고 그 아래에서 각각 15.0과 7.0으로 초기화 하시오.
3. x, y를 사칙 연산한 결과와 나머지 연산한 결과를 출력하시오.
4. f1, f2를 나눗셈, 곱셈한 결과와 나머지 연산한 결과를 출력하시오.
5. **[연습문제 2-1]**과 같은 패키지에 새로운 클래스를 생성하여 작성하시오.

[실행결과]

x, y를 사칙 연산한 결과와 나머지 연산한 결과

15 + 7 = 22
15 - 7 = 8
15 / 7 = 2
15 * 7 = 105
15 % 7 = 1

f1, f2를 나눗셈, 곱셈, 나머지 연산한 결과

15.0 / 7.0 = 2.142857
15.0 * 7.0 = 105.0

15.0 % 7.0 = 1.0

▶ 복합 대입 연산자와 증감 연산자 사용하기

- com.javastudy.ch02.operator

```
public class Operator04 {

    public static void main(String[] args) {

        int num1 = 10;
        int num2 = 10;
        int num3 = 10;

        /* 복합 대입연산자
         * 아래에서 사용한 "+=", "*=", "%=" 연산자 들은 모두 먼저 더하기(+), 곱하기(*),
         * 나머지(%) 연산을 수행한 후에 좌측의 변수에 대입하는 두 가지 동작(연산)을
         * 수행하는 연산자로 이런 유형의 대입 연산자를 복합 대입연산자라고 한다.
         * 아래에서 num1 += 10; 코드의 동작은 먼저 num1 + 10의 연산을 수행한 후에
         * 그 결과를 다시 num1에 대입하게 된다. 즉 num1 = num1 + 10과 동일하다.
         **/
        System.out.println("num1 += 10 = " + (num1 += 10));
        System.out.println("num2 *= 10 = " + (num2 *= 10));
        System.out.println("num3 %= 3 = " + (num3 %= 3));
        System.out.println();

        int i = 5;
        int j = 10;

        /* 증감 연산자
         * 증감 연산자는 연산 대상이 되는 피연산자가 하나인 단항 연산자로 피연산자의 값을
         * 1 증가 또는 1 감소시키는 연산자이다. 증감 연산자는 아래와 같이 피연산자의 앞에
         * 지정할 수도 있고 뒤에 지정할 수도 있다. 앞에 지정하는 경우를 선 증감 연산자라고
         * 하며 뒤에 지정하는 경우를 후 증감 연산자라고 한다. 선 증감 연산자는 다음 연산을
         * 수행하기 전에 피연산자의 값을 먼저 1 증감 시키고 다음 연산을 수행하며 후 증감
         * 연산자는 다음 연산을 수행한 후에 피연산자의 값을 먼저 1 증감 시킨다.
         *
         * 아래 코드에서 i++은 앞에 있는 문자열 데이터 "i++ = "과 연결(+) 연산을
         * 수행한 후에 a의 값을 1 증가시키고 --j는 먼저 j의 값을 1 감소 시킨 후에 앞에
         * 있는 문자열 데이터 "--j = "과 연결(+) 연산을 수행한다.
         **/
        System.out.println("i++ = " + i++ + " : " + i);
        System.out.println("--j = " + --j + " : " + j);
```

```
        int x = 10;
        int y = 0;

        // 먼저 x에 저장된 10을 y에 대입(할당)한 후 x의 값을 10에서 1증가 시킨다.
        y = x++;
        System.out.println("x : " + x + ", y : " + y);

        x = 10;
        y = 0;

        // 먼저 x에 저장된 10을 1증가 시킨 후 증가된 x의 값 11을 y에 대입 시킨다.
        y = ++x;
        System.out.println("x : " + x + ", y : " + y);
    }
}
```

▶ 조건 연산자(삼 항 연산자) 사용하기

- com.javastudy.ch02.operator

```
public class Operator05 {

    public static void main(String[] args) {

        /* 조건 연산자는 연상 대상이 되는 피연산자가 3개라 삼 항 연산자라고도 한다.
         * 조건식 ? 조건식이 true일 때 실행 : 조건식이 false일 때 실행
         **/
        int x = 3;
        System.out.println("x는 : " + (x % 2 == 1 ? "홀수" : "짝수"));
        System.out.println();

        /* 삼 항 연산자는 아래와 같이 중첩하여 연속적으로 사용할 수 있다.
         * x의 값에 따라서 조건식이 참이면 "양수", 거짓이면 다시 한 번 조건식을
         * 사용해 x의 값을 비교하여 x가 음수면 "음수"가 거짓이면 "0"이 출력된다.
         **/
        x = -3;
        System.out.println("x는 : " + (x > 0 ?  "양수" : x < 0 ? "음수" : "0"));
    }
}
```

▶ 입력된 첫 번째 문자가 한글인지 아닌지 판단하기(삼 항 연산자 활용)

- com.javastudy.ch02.operator

```
public class Operator06 {
```

```
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.println("한글 한 자만 입력해주세요");

        // 키보드에 입력된 데이터를 문자열로 읽어온다.
        String input = sc.next();

        /* 키보드에서 읽어온 데이터는 문자열 데이터이므로 자바에서 기본 제공하는
         * String 클래스의 charAt() 메서드를 이용하면 첫 번째 문자를 읽어 올 수 있다.
         * 자바는 String 클래스로 문자열을 처리할 수 있는 다양한 메서드를 제공하고 있다.
         **/
        char ch = input.charAt(0);
        System.out.print("첫 번째 문자 : " + ch + "는 ");
        System.out.println(ch >= '가' && ch <= '힣' ? "한글 입니다." : "한글이 아닙니다.");

        // Scanner 사용이 끝나면 자원을 해제한다.
        sc.close();
    }
}
```

▶ 정수와 실수를 0으로 나눌 때의 결과

- com.javastudy.ch02.operator

```
public class Operator07 {

    public static void main(String[] args) {

        int x = 376;
        float y = 123.456f;

        // 정수를 0으로 나누면 컴파일은 가능하나 실행시 ArithmeticException 발생
        try {
            System.out.println("x / 0 = " + x / 0 + ", " + "x % 0 = " + x % 0);

        } catch(ArithmeticException e) {
            System.out.println("정수를 0으로 나눌 수 없습니다.");
        }

        // 실수를 0.0으로 나누면 Infinity(무한수)의 결과를 얻는다.
        System.out.println("y / 0 = " + y / 0 + ", " + "y % 0 = " + y % 0);

        // 0을 0.0으로 나누면 NaN(Not a Number - 숫자 아님)의 결과를 얻는다.
        System.out.println("0 / 0.0 = " + 0 / 0.0 + "0 % 0.0 = " + 0 % 0.0);
```

```
    }
}
```

[연습문제 2-3]

삼 항 연산자를 활용해 사용자가 입력한 숫자가 홀수인지 짝수인지를 판별하는 다음과 같이 실행되는 프로그램을 작성하시오.

[연습문제 2-1]과 같은 패키지에 새로운 클래스를 생성해 작성하시오.

[실행결과]

숫자를 입력해주세요 :

127

입력된 숫자는 : 127(으)로 홀수입니다.

숫자를 입력해주세요 :

6

입력된 숫자는 : 6(으)로 짝수 입니다.

[연습문제 2-4]

삼 항 연산자를 활용해 사용자가 입력한 숫자가 9의 배수인지를 판별하는 다음과 같이 실행되는 프로그램을 작성하시오.

[연습문제 2-1]과 같은 패키지에 새로운 클래스를 생성해 작성하시오.

[실행결과]

9의 배수를 입력해주세요 :

276

입력된 숫자는 : 276(으)로 9의 배수가 아닙니다.

9의 배수를 입력해주세요 :

117

입력된 숫자는 : 117(으)로 9의 배수 입니다.

[연습문제 2-5]

삼 항 연산자를 활용해 사용자가 입력한 첫 번째 문자가 영문 대문자인지 소문자인지를 판별하는 다음과 같이 실행되는 프로그램을 작성하시오.

[연습문제 2-1]과 같은 패키지에 새로운 클래스를 생성해 작성하시오.

[실행결과]
알파벳 한 자를 입력해주세요 :
F
첫 번째 입력된 문자 F는 영문 대문자 입니다.

알파벳 한 자를 입력해주세요 :
z
첫 번째 입력된 문자 z는 영문 소문자입니다.

3. 제어문과 배열

3.1 조건문

▶ if 문과 나머지 연산자를 이용한 배수와 홀, 짝수 인지 판단하기

- com.javastudy.ch03.conditional

```
public class If01 {

    public static void main(String[] args) {

        int x = 21;
        int y = 7;
        int score = 79;

        /* if 문은 조건식의 결과가 boolean(true 또는 false) 값만 가능하다.
         * 어떤 수를 특정수로 나누어 나머지가 0이면 어떤 수는 특정수의 배수이다.
         **/
        System.out.println("배수 구하기");
        if(x % y == 0) {
            System.out.println(x + "은(는) " + y + "의 배수 입니다.");
        }

        // 특정수를 2로 나누어 나머지가 0이면 짝수 그렇지 않으면 홀수이다.
        System.out.println("홀/짝수 구하기");
        if(x % 2 != 0) {
            System.out.println(x + "은(는) 홀수 입니다.");
        } else {
            System.out.println(x + "은(는) 짝수 입니다.");
        }

        // 점수가 80점 이상과 미만 일 때 각각 다른 메시지를 출력하기
        if(score > 80) {
            System.out.println("축하 합니다. 합격 했습니다.");
        } else {
            System.out.println("아쉽네요 이번에는 불합격 입니다.");
        }
    }
}
```

▶ 키보드로 입력받은 값이 5의 배수인지 판단하기

- com.javastudy.ch03.conditional

```
public class If02 {
```

```
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.print("5의 배수를 입력해 주세요 : ");

        int input = sc.nextInt();

        // 입력된 값이 0이 아니고 5로 나누어 나머지가 0 이면 5의 배수
        if(input != 0 && input % 5 == 0) {
            System.out.println("입력한 수 " + input + "는(은) 5배수 입니다.");

        } else {
            System.out.println("입력한 수 " + input + "는(은) 5의 배수가 아닙니다.");
        }
    }
}
```

▶ if 문을 이용한 학점 출력하기

- com.javastudy.ch03.conditional

```
public class If03 {

    public static void main(String[] args) {

        /* 1 ~ 100까지 랜덤한 수를 얻어 변수에 저장
         * Math.random()은 0 ~ 1사이의 실수를 생성하여 반환해 준다.
         * 생성되는 랜덤 수는 아래와 같이 0은 포함되고 1은 포함되지 않는다.
         * 예 : 0.00000, 0.00012, 0.34567, 0.78912, 0.99999
         **/
        int score = (int)(Math.random() * 100) + 1;

        char grade = ' ';

        if(score >= 90) { // 90점 이상이면
            grade = 'A';
        } else if(score >= 80) { // 80점 이상이면
            grade = 'B';
        } else if(score >= 70) { // 70점 이상이면
            grade = 'C';
        } else {
            grade = 'F';
        }
```

```
        System.out.print("당신의 점수는 " + score + "점으로 ");
        System.out.println(grade + "학점 입니다.");
    }
}
```

▶ 중첩 if 문을 이용해 점수에 해당하는 학점 출력하기

- com.javastudy.ch03.conditional

```
public class If04 {

    public static void main(String[] args) {

        int score = 59;
        String grade = null;

        if(score >= 90) { // 90점 이상이면
            grade = "A";

            if(score >= 95) { // 95점 이상이면
                grade += "+";

            } else {          // 95점 미만이면
                grade += "-";
            }
        } else if(score >= 80) { // 80점 이상이면
            grade = "B";

            if(score >= 85) { // 85점 이상이면
                grade += "+";

            } else {          // 85점 미만이면
                grade += "-";
            }
        } else if(score >= 70) { // 70점 이상이면
            grade = "C";

            if(score >= 75) { // 75점 이상미면
                grade += "+";

            } else {          // 75점 미만이면
                grade += "-";
            }
        } else if(score >= 60) { // 60점 이상이면
            grade = "D";
```

```
        } else {
            grade = "F";
        }

        System.out.print("당신의 점수는 " + score + "점으로 ");
        System.out.println("학점은 " + grade + "입니다.");
    }
}
```

▶ switch 문을 이용한 경품 추첨기 만들기

- com.javastudy.ch03.conditional

```
public class Switch01 {

    public static void main(String[] args) {

        int score = (int) (Math.random() * 10) + 1;

        /* switch 문은 식의 결과가 int 형으로 자동 형 변환 가능하거나
         * byte, char, short, int 형 데이터만 가능하였으나
         * jdk 1.7부터는 String과 enum 타입도 가능해졌다.
         *
         * 아래에서는 switch 문의 괄호 안의 식의 결과가 100, 200, 300, 400
         * 일때 해당 각각의 case 절의 코드가 실행되고 break 명령을 만나서 switch
         * 문을 빠져 나가며 이에 해당하지 않을 때는 default 절의 코드가 실행된다.
         **/
        switch(score * 100) {
            case 100  :
                System.out.println("축 당첨~ 경품은 라면 1박스 입니다.");
                break;
            case 200  :
                System.out.println("축 당첨~ 경품은 5만원 상품권 입니다.");
                break;
            case 300  :
                System.out.println("축 당첨~ 경품은 자전거 입니다.");
                break;
            case 400  :
                System.out.println("측 당첨~ 경품은 자동차 입니다.");
                break;
            default :
                System.out.println("아쉽네요~ 당첨 되지 못했습니다.");
        }
    }
```

```
}
```

▶ switch 문을 이용한 경품 추첨기 2

- com.javastudy.ch03.conditional

```
public class Switch02 {

    public static void main(String[] args) {

        int num = (int) (Math.random() * 10) + 1;
        int score = num * 100;
        String str = "";

        /* 다음과 같이 case 절에 break 명령을 사용하지 않으면 switch 문의
         * 조건식과 일치하는 case 절의 코드부터 그 아래로 나오는 모든 case
         * 절과 default 절의 코드가 실행된다. 다시 말해 score가 800 이라면
         * case 800:의 코드가 실행되고 그 아래의 case 700:의 코드, default
         * 절의 코드가 모두 실행되어 경품은 5만원 상품권, 라면 1박스, 고무장갑이 된다.
         **/
        switch(score) {
            case 1000 :
                str += "자동차, ";
            case 900 :
                str += "자전거, ";
            case 800 :
                str += "5만원 상품권, ";
            case 700 :
                str += "라면 1박스, ";
            default :
                str += "고무장갑";
        }

        System.out.print("당신의 점수는 " + score + "점으로 ");
        System.out.println("경품은 " + str + "입니다.");
    }
}
```

▶ switch 문을 이용해 영문 문자열에 따라서 메달 종류 출력하기

- com.javastudy.ch03.conditional

```
public class Switch03 {

    public static void main(String[] args) {
```

```
        String medal = "silver";

        /* switch 문은 식의 결과가 int 형으로 자동 형 변환 가능하거나
         * byte, char, short, int 형 데이터만 가능하였으나
         * jdk 1.7부터는 String과 enum 타입도 가능해졌다.
         *
         * switch 문의 조건식에 해당하는 case 절이 없다면 default 절만 실행된다.
         **/
        switch(medal) {
        case "Gold": case "gold":
            System.out.println("금메달 입니다.");
            break;
        case "Silver": case "silver":
            System.out.println("은메달 입니다.");
            break;
        case "Bronze": case "bronze":
            System.out.println("동메달 입니다.");
            break;
        default:
            System.out.println("아쉽네요 no 메달 입니다.");
            break;
        }
    }
}
```

[연습문제 3-1]

아래와 같이 main() 메서드 안에 변수가 선언되어 있다. 이 변수에 지정된 나이에 따라서 아래와 같이 출력하는 프로그램을 작성하시오. 단, 나이의 판단은 if문을 이용하시오.

```
    public static void main(String[] args) {

        /* 6세 이하(미취학생), 7세 이하(유치원생), 13세 이하(초등학생),
         * 16세 이하(중학생), 19세 이하(고등학생), 그 외(성인)
         **/
        int age = 17;
        String msg = "";

    }
```

먼저 “JavaStudyCh03Exercise” 프로젝트를 만들고 “com.javastudy.ch03.conditional” 패키지를 만들어 이 패키지에 클래스를 생성하여 프로그램을 작성하시오.

[실행결과]
나이가 17세로 고등학생 입니다.

[연습문제 3-2]

아래와 같이 main() 메서드 안에 변수가 선언되어 있다. 이 변수에 지정된 나이에 따라서 아래 실행 결과와 같이 입장료를 출력하는 프로그램을 작성하시오. 단, 나이의 판단은 if문을 이용하시오.

```
public static void main(String[] args) {

    /* 7세 이하(미취학생) : 0원, 13세 이하(초등학생) : 500원
     * 19세 이하(중 고등학생) : 1000원, 그 외(일반인) : 2000원
     **/
    int age = 9;
    int charge = 0;
    String msg = "";

}
```

[연습문제 3-1]과 같은 패키지에 새로운 클래스를 생성해 작성하시오

[실행결과]
9살로 초등학생 이며, 입장료는 500원 입니다.

[연습문제 3-3]

앞에서 “if문을 이용한 학점 출력하기” If03 클래스의 내용을 switch 문을 사용해 다음과 같이 동작하는 프로그램으로 변경하시오.

[연습문제 3-1]과 같은 패키지에 새로운 클래스를 생성해 작성하시오

[실행결과]
당신의 점수는 74점으로 C학점입니다.

당신의 점수는 53점으로 F학점입니다.

당신의 점수는 1점으로 F학점 입니다.

3.2 반복문

▶ for문을 이용해 1부터 100까지 합 출력하기

- com.javastudy.ch03.forwhile

```
public class For01 {

    public static void main(String[] args) {

        int sum = 0;
        for(int i = 1; i <= 100; i++) {
            sum += i;
        }

        System.out.println("1 ~ 100까지 합 : " + sum);
    }
}
```

▶ for문을 이용해 구구단 7단 출력하기

- com.javastudy.ch03.forwhile

```
public class For02 {

    public static void main(String[] args) {

        int x = 7;

        for(int i = 1; i < 10; i++) {
            System.out.println(x + " x " + i + "= " + x * i);
        }
    }
}
```

▶ for 문을 이용한 1 ~ 100까지 짝수와 홀수의 합 구하기

- com.javastudy.ch03.forwhile

```
public class For03 {
    public static void main(String[] args) {

        int oddSum = 0;
        int evenSum = 0;

        // 반복문을 돌면서 짝수의 합과 홀수의 합을 구한다.
        for(int i = 1; i <= 100; i++) {
```

```
            // 현재 i를 2로 나누어 나머지가 0이면 짝수
            if(i % 2 == 0) {
                evenSum += i;
            } else {
                oddSum += i;
            }
        }

        System.out.println("1 ~ 100까지 짝수의 합 : " + evenSum);
        System.out.println("1 ~ 100까지 홀수의 합 : " + oddSum);
    }
}
```

▶ for문을 이용한 1부터 ~ 100까지 3의 배수와 개수 그리고 합계 구하기

- com.javastudy.ch03.forwhile

```
public class For04 {

    public static void main(String[] args) {

        final int num = 3;
        int count = 0;
        int sum = 0;
        String nums = "";

        for(int i = 1; i <= 100; i++) {

            // 어떤 수를 3으로 나눠서 나머지가 0이번 그 수는 3의 배수이다.
            if(i % num == 0) {
                // 3의 배수를 콤마(,)로 구분해 문자열로 저장
                nums += (100 - num >= i) ?  i + ", " : i;

                // 3의 배수의 합계와 개수를 구한다.
                sum += i;
                count = count + 1; // count++;
            }
        }

        System.out.println("1부터 ~ 100까지 3의 배수는\n " + nums);
        System.out.println("개수는 " + count + "개이며 합계는 " + sum + "입니다.");
    }
}
```

▶ 중첩 for 문을 이용해 구구단 출력하기

- com.javastudy.ch03.forwhile

```
public class For05 {

    public static void main(String[] args) {

        for(int i = 1; i <= 9; i++) {

            for(int j = 2; j <= 9; j++) {
                System.out.print(j + " x " + i + " = " + i * j + "\t");
            }
            System.out.println();
        }
    }
}
```

[연습문제 3-4]

정수 33부터 ~ 235까지 홀수와 짝수의 개수를 구하여 아래와 같이 출력하는 프로그램을 작성하시오.

[연습문제 3-1]과 같은 프로젝트에 "com.javastudy.ch03.forwhile" 패키지를 만들고 이 패키지에 새로운 클래스를 생성하여 작성하시오

[실행결과]

33부터 ~ 235까지 홀수의 개수 : 102
33부터 ~ 235까지 짝수의 개수 : 101

[연습문제 3-5]

100부터 ~ 150까지 정수에서 7의 배수와 7의 배수의 개수를 아래와 같이 출력하는 프로그램을 작성해 보자. 단, 7의 배수를 출력할 때는 삼 항 연산자를 활용해 아래와 같이 쉼표(,)로 구분하여 출력하고 맨 마지막 배수만 쉼표(,)를 생략해 출력해 보자.

[연습문제 3-4]와 같은 패키지에 새로운 클래스를 생성해 작성하시오

[실행결과]

100부터 ~ 150까지 7의 배수 : 105, 112, 119, 126, 133, 140, 147
100부터 ~ 150까지 7의 배수의 개수 : 7개

[연습문제 3-6]

아래와 같이 50부터 ~ 100까지 3의 배수이면서 6의 배수가 아닌 정수와 그 배수의 개수 그리고 그 배수의 합계를 출력하는 프로그램을 작성하시오.

[연습문제 3-4]와 같은 패키지에 새로운 클래스를 생성해 작성하시오

[실행결과]

```
### 50부터 ~ 100까지 3의 배수이면서 6의 배수 구하기 ###
3의 배수이면서 6의 배수가 아닌 수 : 51, 57, 63, 69, 75, 81, 87, 93, 99
3의 배수이면서 6의 배수가 아닌 정수의 개수 : 9
3의 배수이면서 6의 배수가 아닌 정수의 합 : 675
```

▶ while문을 이용해 1 ~ 100까지 합 구하기

- com.javastudy.ch03.forwhile

```
public class While01 {

    public static void main(String[] args) {

        int i = 1;
        int sum = 0;

        while(true) {

            /* while문은 for문과는 다르게 초기문과 반복 후의 명령문을 지정할 수 없어
             * 조건이 true로 주어지는 경우 아래와 같이 while문을 종료하기 위해 필요한
             * 증감식과 특정 조건에서 while문을 빠져나가는 구문이 기술되어야 한다.
             * 그렇지 않으면 while문은 무한 반복문이 된다.
             *
             *  후 증감 연산자를 이용해 i의 값을 sum에 더한 후 1증가시킴
             **/
            sum += i++;

            // i가 100보다 크면 while문을 빠져나간다.
            if(i > 100) {
                break;
            }
        }
        System.out.println("1 ~ 100까지의 합 : " + sum);
```

```
    }
}
```

▶ while 문을 이용한 1 ~ 100까지 7의 배수 출력하기

- com.javastudy.ch03.forwhile

```
public class While02 {

    public static void main(String[] args) {

        final int num = 7;
        int i = 1;
        String nums = "";

        while(i <= 100) {

            // 상수 num의 값이 7이므로 num으로 나누어 나머지가 0이 되면 7의 배수임
            if(i % num == 0) {

                /* 7의 배수를 콤마(,)로 구분하여 nums의 문자열에 추가
                 * 비교 연산자 보다 사칙 연산자가 연산자 우선순위가 높으므로
                 * 100 - 7의 연산을 먼저 수행한 후 그 결과를 현재의 i에 저장된
                 * 값과 비교하여 i에 저장된 값이 작거나 같으면 콤마(,)를 추가하고
                 * 그렇지 않으면 i의 값만 문자열로 추가하기 위해 삼항 연산자를 사용함
                 * 즉 1 ~ 100까지 7의 배수 중 맨 마지막에 위치한 7의 배수를 제외하고
                 * 나머지 7의 배수를 콤마(,)로 구분하여 출력하기 위해 삼항 연산자를 사용함
                 **/
                nums += (i <= 100 - num) ? i + ", " : i;
            }

            /* while문의 조건식에 사용된 i를 1씩 증가시켜 101이 되면
             * while문의 조건식 결과가 false가 되므로 while문을 빠져 나가게 된다.
             **/
            i++;
        }
        System.out.println(nums);
    }
}
```

▶ 한번은 무조건 실행되는 do~while문 사용하기

- com.javastudy.ch03.forwhile

```
public class DoWhile01 {
```

```
    public static void main(String[] args) throws IOException {

        String input = null;
        Scanner sc = new Scanner(System.in);

        /* do~while문 안의 코드가 한 번은 무조건 실행된 후
         * while문의 조건식을 체크하게 된다.
         **/
        do {
            System.out.println("좋아하는 숫자를 입력해 주세요(do~while)\n"
                    + "종료하려면 x 또는 X를 입력하세요");
            System.out.print(">> ");

            input = sc.next();
            System.out.println("입력된 값 : " + input);

        } while(! (input.equals("x") || input.equals("X")));
    }
}
```

▶ 반복문의 처음으로 이동하는 continue 문 사용하기

- com.javastudy.ch03.forwhile

```
public class LoopContinue01 {

    public static void main(String[] args) {

        int sum = 0;

        for(int i = 1; i <= 100; i++) {

            if(i % 2 == 0) {

                /* continue 명령을 만나면 현재 반복문의 처음으로 돌아간다.
                 * for 문일 경우 continue 명령을 만나면 증감식으로 이동하여
                 * 값을 증가시킨 후 조건식으로 이동하여 true면 for 문 안의 코드를
                 * 수행하고 조건식이 false면 for 문을 빠져 나가게 된다.
                 * while 문일 경우 조건식으로 이동하고 do~while 문일 경우
                 * while 문의 조건식으로 이동하여 true면 반복문 안의 코드를
                 * 수행하고 false면 반복문을 빠져 나가게 된다.
                 * while 문이나 do~while 문의 조건식에 사용되는 변수의 값이
                 * 변경되는 위치에 따라 무한 반복문이 될 수 있으므로 주의가 필요하다.
                 **/
                continue;
```

```
            } else {
                sum += i;
            }
            // 이 아래 다른 추가 코드가 있다고 가정
        }
        System.out.println("1 ~ 100까지 홀수의 합 : " + sum);
    }
}
```

▶ 반복문을 빠져 나가는 break 문 사용하기

- com.javastudy.ch03.forwhile

```
public class LoopBreak01 {

    public static void main(String[] args) {

        for(int i = 1; i <= 10; i++) {
            if(i >= 5) {
                // break 명령을 만나면 현재 반복문을 빠져나간다.
                System.out.println("break x : " + i);
                break;
            }
        }
        System.out.println();

        for(int i = 1; i < 5; i++) {
            for(int j = 1; j < 5; j++) {

                if(j >= 3) {
                    /* break 명령이 기술된 안쪽 for 문만 빠져나간다.
                     * 현재 for문을 빠져 나가 밖에 있는 for 문의 "바깥쪽 for 문..." 부분이
                     * 콘솔에 출력되고 증감식으로 이동하여 i를 1증가시키고 조건식을 비교하여
                     * true면 다시 바깥쪽 for 문으로 들어와 안쪽 for 문이 실행된다.
                     **/
                    System.out.println("안쪽 for 문 break : j = " + j);
                    break;
                }
                System.out.println("안쪽 for 문 : j = " + j);
            }
            System.out.println("바깥쪽 for 문  : i = " + i);
        }
    }
}
```

[연습문제 3-7]

반복문을 사용해 아래와 같이 1부터 ~ 60까지의 수를 한 줄에 10단위씩 출력하는 프로그램을 작성하시오.

[연습문제 3-4]와 같은 패키지에 새로운 클래스를 생성해 작성하시오

[실행결과]

1, 2, 3, 4, 5, 6, 7, 8, 9, 10
11, 12, 13, 14, 15, 16, 17, 18, 19, 20
21, 22, 23, 24, 25, 26, 27, 28, 29, 30
31, 32, 33, 34, 35, 36, 37, 38, 39, 40
41, 42, 43, 44, 45, 46, 47, 48, 49, 50
51, 52, 53, 54, 55, 56, 57, 58, 59, 60

3.3 배열

▶ 동일한 타입의 데이터를 하나로 묶어 다루는 배열(Array)

- com.javastudy.ch03.array

```
public class JavaArray01 {

    public static void main(String[] args) {

        // 5개의 int 형 자료를 저장할 수 있는 배열의 선언과 초기화
        int numbers[] = new int[5];
        int[] nums = {1, 3, 5, 7, 9};

        // 배열의 원소(데이터)는 index로 접근 - index는 0부터 시작 한다.
        int num1 = nums[0];
        System.out.println("배열 nums의 첫 번째 데이터 : " + num1);

        // 배열의 길이는 index와 달리 배열 안에 저장된 데이터 개수를 의미 한다.
        System.out.println("배열의 길이 : " + nums.length);

        /* 배열의 크기를 이용해 for문의 조건식을 작성
         * 배열의 원소는 0부터 시작하는 index로 접근하기 때문에 배열의
         * 마지막 원소가 위치한 index는 1부터 시작하는 배열의 길이보다 1작다.
         * 배열의 index 범위를 벗어난 index를 지정하여 배열에 접근하면
         * IndexOutOfBoundsException이 발생한다.
         **/
        System.out.print("배열 nums의 데이터 : ");
        for(int i = 0; i < nums.length; i++) {
            // i가 배열의 마지막 데이터가 아니면 콤마(,)로 구분하여 출력
            System.out.print(i != nums.length - 1 ? nums[i] + ", " : nums[i]);
        }

        // for문을 이용해 배열 mumbers에 데이터 저장하기
        for(int i = 0; i < numbers.length; i++) {
            numbers[i] = i;
        }
        System.out.println();

        System.out.print("배열 numbers의 데이터 : ");
        for(int i = 0; i < numbers.length; i++) {
            System.out.print(i < numbers.length - 1 ? i + ", " : i);
        }
    }
}
```

▶ 배열을 이용한 학생의 성적(총점, 평균) 출력하기

- com.javastudy.ch03.array

```
public class JavaArray02 {

    public static void main(String[] args) {

        // int 데이터를 저장하는 배열의 선언 및 초기화
        int[] score = new int[3];
        score[0] = 88;
        score[1] = 81;
        score[2] = 86;
        int total = 0;

        // for문을 이용해 배열에 저장된 점수의 총점을 구한다.
        for(int i = 0; i < score.length; i++) {
            total += score[i];
        }

        System.out.println("총점 : " + total + ", 평균 : " + total / score.length);

        total = 0;
        /* 향상된 for 문은 배열이나 열거형(Enumeration) 또는 컬렉션 프레임 워크의
         * List 계열 객체에 순차적으로 접근할 때 유용하게 사용할 수 있는 반복문이다.
         * for문 안에서 사용할 변수를 선언하고 배열과 열거형 또는 컬렉션을 지정하면
         * 반복문이 반복될 때 마다 변수를 이용해 현재 위치의 데이터에 접근할 수 있다.
         **/
        for(int s : score) {

            /* 변수 s는 index의 역할을 하는 것이 아니라 반복문이 반복될 때 마다
             * score 배열의 현재 index에 해당하는 데이터가 s에 저장 된다.
             * 즉 s = score[0], score[1], score[2]... 와 같이 동작하게 된다.
             **/
            total += s;
        }

        System.out.println("총점 : " + total + ", 평균 : " + total / score.length);
    }
}
```

▶ 배열을 이용한 로또번호 생성기 만들기

- com.javastudy.ch03.array

```
public class JavaArray03 {
```

```
    public static void main(String[] args) {

        // 로또번호 6개를 저장할 배열 변수를 선언하고 초기화 한다.
        int[] lotto = new int[6];
        for(int i = 0; i < lotto.length; i++) {
            // for문이 반복될 때 마다 난수를 발생해 로또 번호를 생성하고 변수 lotto에 저장
            int num = (int) (Math.random() * 45) + 1;
            lotto[i] = num;
        }

        // for문을 이용해 생성된 로또 번호를 화면에 출력한다.
        for(int i = 0; i < lotto.length; i++) {
            System.out.print(i < (lotto.length - 1) ? lotto[i] + ", " : lotto[i]);
        }
    }
}
```

▶ 버블 정렬을 이용한 배열 데이터 오름차순 정렬하기

- com.javastudy.ch03.array

```
public class JavaArray04 {

    public static void main(String[] args) {

        /* 자바에서 배열은 C나 C++과 달리 객체로 다루고 있다.
         * 자바에서는 변수의 종류를 크게 기본형 변수와 참조형 변수로 나뉜다.
         * 기본형 변수는 자바의 8개 기본형을 저장할 수 있는 변수를 말하며 그 크기 또한
         * 각 기본형 타입의 크기와 동일하다. 하지만 참조형 변수는 객체가 생성된 메모리
         * 주소의 참조 값을 저장하는 변수로 참조형 변수는 모두 4byte의 크기를 가진다.
         * 다시 말해 기본형 변수에는 할당 하는 값 자체가 저장 되지만 참조형 변수에는
         * 객체가 생성된 메모리 주소를 참조할 수 있는 4byte의 참조 값이 저장된다.
         * 아래 1번과 같이 배열 변수를 선언하는 것은 단지 배열 데이터를 다루기 위해
         * 메모리의 참조 값이 저장되는 참조형 변수를 위한 공간이 만들어 질 뿐 실제
         * 배열이 생성되는 것은 아니다.
         * 배열에 데이터를 저장하기 위해서는 아래 2번과 같이 new 연산자를 이용해
         * 배열을 생성해야 비로소 데이터를 저장할 수 있는 배열이 메모리 공간에 만들어 진다.
         * 아래 3번과 4번 코드는 배열 변수 chars를 선언하고 배열에 데이터를 지정하여
         * 배열 변수를 초기화 하는 코드로 변수의 선언과 동시에 메모리 공간에 배열이
         * 생성되고 그 배열에 지정한 실제 데이터가 저장되는 코드이다.
         *
         *  1. int[] nums;
         *  2. String[] strs = new String[5];
         *  3. char[] chars = { '가', '나', '다' };
```

```
         *  4. int[] ints = new int[]{ 1, 2, 3, 4, 5 };
         * */

        /* int형 배열 변수 numbers를 선언하고 int형 데이터 10개를 저장할 수 있는
         * 배열을 생성하여 배열이 생성된 주소의 참조 값으로 numbers 변수를 초기화 한다.
         * int형 배열 변수 numbers는 기본 타입 변수가 아니라 참조 타입 변수로
         * 위에서 언급한 것과 같이 4byte 크기를 가지며 저장 되는 값은 10개의 배열
         * 데이터가 아니라 실제 배열이 생성된 메모리 주소의 참조 값이 저장된다.
         **/
        int[] numbers = {8, 3, 9, 1, 5, 0, 7, 6, 2, 4};
        //int numbers[] = {8, 3, 9, 1, 5, 0, 7, 6, 2, 4};

        /* 배열에 저장된 데이터를 오름차순으로 정렬하기 위해 버블정렬 알고리즘을 이용해
         * 크기순으로 정렬하는 코드로 버블정렬 알고리즘은 배열의 크기가 n개일 때 배열의
         * 첫 번째 요소부터 n-1까지의 요소를 다음에 오는 요소와 크기를 비교하여 자리
         * 바꿈을 반복하는 것이다.
         **/
        for(int i = 0; i < numbers.length; i++) {
            for(int j = 0; j < numbers.length -1 - i; j++) {
                /* 현재 배열의 요소와 바로 다음에 위치한 배열의 요소를 비고하여 현재의
                 * 요소가 크다면 즉 j == 0 일때 배열의 첫 번째 요소와 두 번째 요소를
                 * 비교하여 첫 번째 요소가 크다면 첫 번째 요소를 뒤 쪽으로 옮기고
                 * 두 번째 요소를 앞으로 옮기는 작업을 한다.
                 **/
                if(numbers[j] > numbers[j + 1]) {
                    /* 임시 저장소로 사용할 temp 변수를 선언하고 크기가 큰 현재
                     * 위치의 요소를 temp 변수에 저장한 후 크기가 작은 다음 위치의
                     * 요소를 현재 위치에 저장한다. 그리고 temp 변수에 저장된
                     * 값을 다음 위치에 저장하여 현재 위치의 요소를 뒤 쪽으로 옮긴다.
                     **/
                    int temp = numbers[j];
                    numbers[j] = numbers[j + 1];
                    numbers[j + 1] = temp;
                }
            }

            // 향상된 for 문을 이용해 numbers 배열의 정렬된 데이터를 출력한다.
            for(int k : numbers) {
                System.out.print(k);
            }
            System.out.println();
        }
    }
}
```

▶ 2차원 배열 사용하기

- com.javastudy.ch03.array

```
public class JavaArray05 {

    public static void main(String[] args) {

        /* 2차원 배열을 선언하고 각 요소를 1차원 배열로 초기화
         * 2차원 배열은 배열안의 배열로 2차원 배열의 각 요소는 1차원 배열로 구성된다.
         **/
        int [][] nums = {
                { 1, 2, 3}, { 4, 5, 6 }, { 7, 8, 9 }
        };

        /* 2 x 3 크기의 2차원 배열을 생성하고
         * 배열이 생성된 메모리 주소의 참조 값을 변수 str에 저장한다.
         **/
        String strs[][] = new String[2][3];

        /* 2차원 배열 strs의 첫 번째 요소인 1차원 배열의 index 0의 위치에
         * 문자열 데이터 "자바"를 저장하고 strs의 2번째 요소인 1차원 배열의
         * index 2의 위치에 문자열 데이터 "오라클"을 저장한다. 나머지 배열
         * 요소는 String이 참조 타입이므로 참조 타입 변수의 기본 값인 null로 채워진다.
         **/
        strs[0][0] = "자바";
        strs[1][2] = "오라클";

        // 2차원 배열은 중첩 for문을 이용해 배열의 각 요소에 순차적으로 접근할 수 있다.
        for(int i = 0; i < strs.length; i++) {
            for(int j = 0; j < strs[i].length; j++) {
                System.out.println("strs[" + i + "][" + j + "] - " + strs[i][j] );
            }
        }

        // 2차원 배열도 index를 이용해 배열의 각 요소에 접근할 수 있다.
        System.out.println();
        System.out.println(nums[1][1] + ", " + nums[2][0]);

        // 2차원 배열은 중첩 for문을 이용해 배열의 각 요소에 순차적으로 접근할 수 있다.
        for(int i = 0; i < nums.length; i++) {

            for(int j = 0; j < nums[i].length; j++) {

                System.out.print(nums[i][j] + ", ");
```

```
            }
        }
    }
}
```

[연습문제 3-8]

아래 표는 국내 지역별 국립공원을 정리한 표이다. 이 표의 내용을 참고해 지역을 1차원 배열에 저장하고 각 지역의 국립공원을 2차원 배열에 저장해 실행결과와 같이 출력하는 프로그램을 작성하시오.

[연습문제 3-1]과 같은 프로젝트에 "com.javastudy.ch03.array" 패키지를 만들어 이 패키지에 새로운 클래스를 생성하여 작성하시오.

지역	**국립공원**		
서울	관악산	도봉산	북한산
중부	계룡산	월악산	속리산
남부	내장산	지리산	가야산
태백	설악산	오대산	태백산

[실행결과]

```
  ###  지역별 국립공원  ###
서울지역 : 관악산, 도봉산, 북한산
중부지역 : 계룡산, 월악산, 속리산
남부지역 : 내장산, 지리산, 가야산
태백지역 : 설악산, 오대산, 태백산
```

4. 클래스와 객체

객체(Object)란 실세계에 물리적으로 존재하거나 추상적인 것들 중에 속성을 가지고 있고 다른 것과 식별이 가능한 것을 의미한다. 실제 물리적으로 존재하는 것은 사람, 컴퓨터, 책, 휴대폰, 자동차, 동물 등을 예로 들 수 있고 추상적인 것들은 실제로는 눈에 보이지 않지만 현실 세계에서 행해지고 있는 것을 의미하고 주문, 강의, 수강 등이 있다.
객체는 모두 속성(데이터)과 동작(기능)을 가지고 있는데 예를 들면 사람의 속성은 이름, 나이, 성별 등이 있고 동작은 말하다, 먹다, 걷다 등이 있다. 또한 휴대폰은 제조사, 사양, 가격 등의 속성을 가지고 있고 화면을 켜다, 전화를 걸다, 메일을 보내다 등의 동작이 있다.
객체의 속성(Property)을 필드(Field)라고도 부르며 동작을 메서드(Method)라고 부른다.
객체는 개별적으로 사용될 수도 있고 다른 객체와 관계를 맺고 더 크고 많은 속성과 기능을 가질 수도 있다. 객체간의 관계에는 포함 관계와 상속 관계가 있으며 포함 관계는 현재 객체 내부에서 또 다른 객체의 속성과 동작을 사용하는 것을 의미하고 상속 관계는 우리 인간 세상의 상속과 같은 개념으로 부모 객체가 가지고 있는 속성과 기능을 자식 객체가 물려받아 부모 객체로부터 물려받은 속성과 기능을 자식 객체가 사용할 수 있는 것을 의미한다.
클래스는 객체의 속성과 동작을 정의해 놓은 설계도로 설계도를 보고 여러 개의 상품을 대량 생산할 수 있는 것처럼 하나의 클래스를 정의해 놓고 여러 개의 객체를 생성할 수 있다. 이렇게 여러 개의 객체를 찍어 내듯이 생성할 수 있어서 클래스를 객체의 틀이라도 한다.
앞에서와 같이 프로그램 구현에 필요한 객체를 파악하여 각각의 객체의 역할을 정의하고 객체들 간의 상호작용을 통해 프로그램을 작성하는 패러다임을 객체지향 프로그래밍(OOP, Object Oriented Programming)이라고 한다.
객체지향 프로그래밍에는 앞에서 언급한 객체, 클래스, 상속의 개념 이외에도 인스턴스, 추상화, 캡슐화, 다형성과 같은 다양한 개념들이 존재하며 추상화(Abstraction), 캡슐화(Encapsulation), 상속(Inheritance), 다형성(Polymorphism)은 객체지향 프로그래밍의 대표적인 특징이라고 할 수 있다.

4.1 클래스 정의와 객체 생성

▶ 클래스 없이 시간 데이터 다루기

com.javastudy.ch04.classdefinition

```
public class NoTimeClass {

    public static void main(String[] args) {

        /* 여러 개의 시간 데이터를 다루기 위해 여러 개의 변수가 필요하다.
         * 아래는 3개의 시간 데이터를 다루기 위해 변수 9개를 선언하였다.
         * 만약 100개의 시간 데이터를 다뤄야 한다면 그때 마다 매번
         * 3개의 변수를 추가로 선언해야 하는 불편함이 따른다.
         **/
        int hour1, hour2, hour3;
        int minute1, minute2, minute3;
        int second1, second2, second3;
```

```
        hour1 = 12;
        minute1 = 35;
        second1 = 57;
        System.out.println("현재 시간 : "
                + hour1 + "시 " + minute1 + "분 " + second1 + "초");

        /* 여러 개의 시간 데이터를 다루기 위해 배열을 사용하면 변수만 사용할 때
         * 보다 조금 간편해 지긴 했지만 시, 분, 초 데이터가 따로 분리되어 있어
         * 프로그래밍 중에 다른 시간, 분, 초 데이터와 서로 뒤섞일 가능성이 있다.
         **/
        int[] hour = new int[3];
        int[] minute = new int[3];
        int[] second = new int[3];
        hour[0] = 9;
        hour[1] = 11;
        hour[2] = 10;
        minute[0] = 24;
        minute[1] = 37;
        minute[2] = 57;
        second[0] = 3;
        second[1] = 56;
        second[2] = 27;

        for(int i = 0; i < hour.length; i++) {
            System.out.println((i + 1) + "번째 시간 데이터 : "
                    + hour[i] + "시 " + minute[i] + "분 " + second[i] + "초");
        }
        System.out.println();
    }
}
```

▶ 시간 정보를 저장하는 클래스 정의하고 사용하기

com.javastudy.ch04.classdefinition 패키지에 각각 별도의 클래스 파일로 작성

```
public class Time {
    // 시간을 하나의 데이터 타입으로 다루기 위해 시, 분, 초 3개의 필드를 정의
    int hour;
    int minute;
    int second;
}


public class TimeUseExam {
```

```
    public static void main(String[] args) {

        /* Time 클래스를 정의해 사용하면 시, 분, 초에 대한 시간 데이터를 하나로 묶어서
         * 데이터 타입으로 관리할 수 있으므로 시간 데이터가 뒤섞일 염려가 없어진다. 또한
         * 새로운 시간 데이터가 더 필요하면 Time 클래스의 객체를 생성해 얼마든지 새로운
         * 시간 데이터로 사용할 수 있기 때문에 시간 데이터를 위해서 매번 새로운 변수를
         * 여러 개 만들어야 하는 불필요한 작업이 필요 없으며 이미 만들어 놓은 클래스의
         * 코드를 재사용하므로 코드의 중복을 줄이고 재사용성을 높일 수 있다는 장점이 있다.
         **/
        Time time = new Time();
        time.hour = 12;
        time.minute = 27;
        time.second = 58;
        System.out.println("Time 클래스 현재 시간 : "
                + time.hour + "시 " + time.minute + "분 " + time.second + "초");

        /* Time 클래스 타입의 배열을 사용하면
         * 여러 시간 데이터를 하나로 묶어서 관리할 수 있어서 편리하다.
         **/
        Time[] times = new Time[2];
        times[0] = new Time();
        times[0].hour = 14;
        times[0].minute = 38;
        times[0].second = 29;

        times[1] = new Time();
        times[1].hour = 11;
        times[1].minute = 58;
        times[1].second = 59;

        for(int i = 0; i < times.length; i++) {
            System.out.println("Time 클래스 현재 시간 : " + times[i].hour + "시 "
                    + times[i].minute + "분 " + times[i].second + "초");
        }
    }
}
```

▶ 게임에 참가하는 선수 정보를 클래스로 정의하고 사용하기

com.javastudy.ch04.classdefinition 패키지에 별도의 클래스 파일로 작성

```
/* 객체(Object)란 실세계에 물리적으로 존재하거나 추상적인 것들 중에 속성을 가지고 있고
 * 다른 것과 식별이 가능한 것을 의미한다. 실제 물리적으로 존재하는 것은 사람, 컴퓨터,
 * 책, 휴대폰, 자동차, 동물 등을 예로 들 수 있고 추상적인 것들은 실제로는 눈에 보이지
 * 않지만 현실 세계에서 행해지고 있는 것을 의미하고 주문, 강의, 수강 등이 있다.
```

```
 * 객체는 모두 속성(데이터)과 동작(기능)을 가지고 있는데 예를 들면 사람의 속성은 이름, 나이,
 * 성별 등이 있고 동작은 말하다, 먹다, 걷다 등이 있다. 또한 휴대폰은 제조사, 사양, 가격 등의
 * 속성을 가지고 있고 화면을 켜다, 전화를 걸다, 메일을 보내다 등의 동작이 있다.
 * 객체의 속성(Property)을 필드(Field)라고도 부르며 동작을 메서드(Method)라고 부른다.
 * 객체는 개별적으로 사용될 수도 있고 다른 객체와 관계를 맺고 더 크고 많은 속성과 기능을
 * 가질 수도 있다. 객체간의 관계에는 포함 관계와 상속 관계가 있으며 포함 관계는 현재 객체
 * 내부에서 또 다른 객체의 속성과 동작을 사용하는 것을 의미하고 상속 관계는 우리 인간
 * 세상의 상속과 같은 개념으로 부모 객체가 가지고 있는 속성과 기능을 자식 객체가 물려받아
 * 부모 객체로부터 물려받은 속성과 기능을 자식 객체가 사용할 수 있는 것을 의미한다.
 *
 * 클래스는 객체의 속성과 동작을 정의해 놓은 설계도로 설계도를 보고 여러 개의 상품을
 * 대량 생산할 수 있는 것처럼 하나의 클래스를 정의해 놓고 여러 개의 객체를 생성할 수 있다.
 * 이렇게 여러 개의 객체를 찍어 내듯이 생성할 수 있어서 클래스를 객체의 틀이라도 한다.
 *
 * 앞에서와 같이 프로그램 구현에 필요한 객체를 파악하여 각각의 객체의 역할을 정의하고
 * 객체들 간의 상호작용을 통해 프로그램을 작성하는 패러다임을 객체지향 프로그래밍
 * (OOP, Object Oriented Programming)이라고 한다.
 * 객체지향 프로그래밍에는 앞에서 언급한 객체, 클래스, 상속의 개념 이외에도 인스턴스,
 * 추상화, 캡슐화, 다형성과 같은 다양한 개념들이 존재하며 추상화(Abstraction),
 * 캡슐화(Encapsulation), 상속(Inheritance), 다형성(Polymorphism)은
 * 객체지향 프로그래밍의 대표적인 특징이라고 할 수 있다.
 **/

/* 실세계에 존재하는 객체를 컴퓨터 세계의 객체로 다루기 위해서 실세계의 대상 객체가
 * 가지는 공통적인 특징(속성과 기능)을 도출해 아래와 같이 클래스로 정의하는 것을
 * 객체지향 프로그래밍에서는 추상화라고 한다.
 **/
public class Player {

    // 객체가 가지는 데이터를 속성(Property) 또는 필드(Field)라고 부른다.
    public String name;
    public int age;
    public String gender;
    public String nationality;

    /* 프로그래밍에서 특정 동작을 코드로 묶어 놓은 것을 함수(Function)라고 부르지만
     * 객체지향 프로그래밍에서는 클래스 내부에 정의한 함수를 메서드(Method)라고 부른다.
     * 함수와 메서드가 각각 존재하는 객체지향 프로그래밍 언어도 있지만 자바에서는 모든
     * 속성과 동작이 클래스 내부에 작성되므로 객체가 가지는 동작을 메서드(Method)라고
     * 부른다. 객체가 제공하는 기능이 없다면 클래스 안에 Method는 정의하지 않을 수 있다.
     **/

}
```

```
public class PlayerPrint {

    public static void main(String[] args) {

        /* 클래스(Class), 객체(Object), 인스턴스(Instance)
         * 클래스는 객체의 속성과 동작을 정의해 놓은 설계도라고 하며 이 설계도를
         * 통해 여러 개의 상품을 대량 생산할 수 있는 것처럼 하나의 클래스를 정의해 놓고
         * 아래와 같이 여러 개의 객체를 생성할 수 있다. 또한 클래스에 정의된 정보를 이용해
         * 여러 개의 객체를 찍어 내듯이 생성할 수 있어서 클래스를 객체의 틀이라고도 한다.
         *
         * 클래스를 통해 메모리에 실제 만들어지는 하나하나의 객체를 클래스의 인스턴스라고
         * 부르며 인스턴스는 클래스를 통해 실제 메모리에 실체화된 객체를 가리키는 용어이다.
         *
         * 아래에서 Player cheolSu = new Player(); 코드로 만들어진 것은 객체이며
         * cheolSu라는 객체는 Player 클래스를 통해 메모리에 실제 만들어진 인스턴스이다.
         * 인스턴스라는 말은 특정 객체가 어떤 클래스를 통해 메모리에 만들어진 객체인지를
         * 표현할 때 사용하는 용어이다. 아래에서 cheolSu를 객체라고 부르며 이와 더불어
         * "cheolSu는 Player 클래스의 인스턴스다."라고 부른다. cheolSu는 인스턴스라고
         * 부르기 보다는 객체라는 표현 더 잘 어울리며 또한 cheolSu는 Player 클래스의
         * 객체라고 부르는 것 보다는 cheolSu는 Player 클래스의 인스턴스라고 부르는 것이
         * 더 잘 어울리는 표현이다.
         **/

        /* Player 클래스의 인스턴스를 생성하고
         * cheolSu라는 변수에 인스턴스가 생성된 메모리 주소의 참조 값을 할당
         * 변수 cheolSu는 인스턴스의 참조 값을 저장하는 변수로 참조형 변수라고 부른다.
         *
         * name=null, age=0, gender=null, nationality=null로 초기화됨
         **/
        Player cheolSu = new Player();

        // 참조형 변수 cheolSu를 이용해 인스턴스의 속성을 설정한다.
        cheolSu.name = "철수";
        cheolSu.age = 25;
        cheolSu.gender="남성";
        cheolSu.nationality = "대한민국";

        /* Player 클래스의 인스턴스를 생성하고
         * christie라는 변수에 인스턴스가 생성된 메모리 주소의 참조 값을 할당
         **/
        Player christie = new Player();

        // 참조형 변수 christie를 이용해 인스턴스의 속성을 설정한다.
        christie.name = "크리스티";
        christie.age = 21;
```

```
        christie.gender="여성";
        christie.nationality="영국";

        System.out.println("\t\t\t## 출전 선수 ##");
        System.out.println("\t국가명\t\t이름\t\t나이\t\t성별");
        System.out.println("\t" + cheolSu.nationality + "\t" + cheolSu.name
                + "\t\t" + cheolSu.age + "\t\t" + cheolSu.gender);
        System.out.println("\t" + christie.nationality + "\t\t" + christie.name
                + "\t" + christie.age + "\t\t" + christie.gender);
    }
}
```

[연습문제 4-1]
학생 정보를 저장하는 클래스를 정의하고 출력하기

우리는 앞에서 시간 데이터를 다루기 위해 여러 개의 변수를 사용하는 방식, 배열을 사용하는 방식, 그리고 Time 클래스를 정의하여 사용하는 방식을 알아보았다. 앞에서 Time 클래스를 만들어 학습한 방식으로 아래 요구사항에 따라 클래스를 작성하고 학생 정보를 출력하는 프로그램을 작성해 보자.

먼저 "JavaStudyCh04Exercise" 프로젝트를 만들고 "com.javastudy.ch04.classdefinition" 패키지를 만들어 이 패키지에 필요한 클래스를 생성하여 프로그램을 작성하시오.

1. 학생 정보 이름, 나이, 성별, 주소를 저장할 수 있는 Student 클래스를 정의 한다.
2. 학생 정보를 출력하는 StudentInfoPrint 클래스를 별도의 소스 파일로 만들고 이 클래스에 프로그램 진입점인 main() 메서드를 만들어 Student 클래스의 인스턴스 3개를 생성한 후 아래와 같이 출력되도록 프로그램을 작성하시오.

[실행결과]
학생 정보 출력(클래스 정의)
이름 : 강바람, 나이 : 20, 성별 : 남성, 주소 : 인천시 부평구 부개동
이름 : 오빛나, 나이 : 21, 성별 : 여성, 주소 : 서울시 영등포구 당산동
이름 : 어머나, 나이 : 25, 성별 : 여성, 주소 : 부산시 해운대구 반여동

▶ 속성과 메서드를 가지는 클래스 정의하고 사용하기
- com.javastudy.ch04.classdefinition

```
public class Animal01 {

    // 객체가 가지는 데이터를 속성(Property) 또는 필드(Field)라고 부른다.
```

```
String name;
int age;
String sound;
String kind;

/* 객체지향 프로그래밍에서는 객체가 가지는 동작(기능)을 메서드(Method)라고 부른다.
 * 메서드는 객체가 가지는 특정 동작을 구현하기 위해서 작성한 코드의 묶음이다.
 * 여러 곳에서 반복적으로 사용될 수 있는 코드를 묶어서 메서드로 만들어 놓으면
 * 그 기능이 필요한 곳에 코드를 중복해서 사용하지 않고 메서드를 사용하면 되므로
 * 코드의 중복을 최소화 하고 코드의 재사용성을 높을 수 있다는 장점이 있다.
 *
 * 메서드를 정의할 때는 그 메서드가 제공하는 기능을 정의하고 그 기능을 구현하기
 * 위해서 외부에서 데이터가 입력되어야 하는지와 메서드가 실행을 완료한 후에
 * 자신을 호출한 곳으로 돌아갈 때 어떤 데이터를 외부에 돌려줘야 하는지를 결정해서
 * 메서드의 선언부를 작성해야 한다.
 *
 * 아래의 eat() 메서드는 동물이 먹는 모습을 텍스트로 출력하는 메서드로 이 메서드가
 * 실행되기 위해서 외부로부터 받아야할 데이터는 필요 없기 때문에 메서드의 () 안을
 * 비어 놓았다. 또한 메서드 이름 앞에 지정해야 할 반환 타입은 이 메서드가 실행을
 * 완료한 후에 외부에 돌려줘야할 데이터가 없다는 의미로 void를 지정하고 있다.
 **/
void eat() {
    System.out.println(name + "이(가) 먹습니다.");
}

/* 아래의 sleep() 메서드는 어떤 이름을 가진 동물이 잠을 자는지를 출력하는
 * 메서드로 이름을 외부에서 받기 위해서 메서드의 () 안에 이름을 받을 수 있도록
 * String 타입의 매개변수(Parameter)를 정의하고 있다. 그리고 메서드 내부에서
 * 출력만 하면 되므로 외부에 돌려줄 데이터가 없기 때문에 메서드 이름 앞에 지정해야
 * 할 반환 타입은 void를 지정하고 있다.
 **/
void sleep(String name) {
    System.out.println(name + "이(가) 잠을 잡니다.");
}

/* 아래 play() 메서드는 어떤 이름을 가진 동물이 즐겁게 노는지를 문자열로 반환하는
 * 메서드로 클래스 내부 데이터를 사용해 외부에 반환되는 데이터를 만들기 때문에
 * 메서드 () 안을 비어 놓았다. 그리고 문자열을 반환하기 위해서 메서드 이름 앞에
 * 지정해야 할 반환타입은 String 타입을 지정하고 있다.
 **/
String play() {
    return name + "이(가) 즐겁게 장난을 칩니다.";
}

/* 아래의 cry() 메서드는 어떤 이름을 가진 어떤 종류의 동물이 어떤 소리로 우는지를
```

```
     * 문자열로 반환하는 메서드로 동물의 이름은 클래스 내부 데이터를 사용하지만 동물의
     * 종류와 울음소리는 외부로부터 받기 위해서 메서드의 () 안에 String 타입의 매개변수
     * (Parameter)를 정의하고 있다. 그리고 최종 결과로 문자열을 반환하기 위해서
     * 메서드 이름 앞에 지정해야 할 반환타입은 String 타입을 지정하고 있다.
     **/
    String cry(String kind, String sound) {
        return name + "은(는) " + kind + "(으)로" + sound;
    }

    /* 이 클래스는 별도로 상속 받는 클래스를 지정하지 않았기 때문에 컴파일러에 의해
     * 자바의 최상위 조상인 Object 클래스를 자동으로 상속 받게 되며 아래와 같이
     * 콘솔에 출력해 주는 println() 메서드에 객체를 참조하는 참조 변수만 기술해도
     * Object 클래스로부터 상속받은 toString()이 자동으로 호출 된다. 참고로
     * toString() 메서드는 현재 객체의 상태를 문자열로 반환하는 메서드 이다.
     *
     * 아래는 Object 클래스로부터 상속받은 toString() 메서드를 이 클래스에 맞게
     * 다시 정의(수정) 했기 때문에  Object로부터 상속 받은 toString() 메서드가
     * 호출되는 것이 아니라 이 클래스에서 다시 정의한 아래의 toString()이 호출 된다.
     *
     * Animal01 animal = new Animal01();
     * System.out.println(animal);
     *
     * 상속 받은 부모의 메서드와 동일한 이름과 특징(반환타입, 매개변수)을 지닌 메서드를
     * 선언하게 되면 이 클래스로부터 생성된 객체는 부모의 메서드가 호출되는 것이 아니라
     * 자신의 메서드가 호출되는데 이러한 기법을 오버라이딩(Overriding) 이라 한다.
     * 오버라이딩은 부모로부터 물려받은 메서드를 자식 대에서 필요한 기능을 추가 하거나
     * 수정하는 것을 의미하며 이는 부모로부터 상속 받은 메서드를 자식이 필요에 의해서
     * 메서드의 기능을 다시 정의하는 것이므로 메서드 재정의라고도 한다.
     *
     * 아래와 같이 메서드 이름 앞에 이 메서드가 실행을 완료한 후 호출한 곳으로
     * 다시 돌아갈 때 가지고 갈 데이터 타입을 지정하는데 이 메서드는 현재 인스턴스의
     * 상태를 문자열로 반환하는 기능을 제공하는 메서드이므로 String을 지정했다.
     **/
    @Override
    public String toString() {
        return name + "은(는) " + kind + "(으)로 " + age + "살 입니다.";
    }
}
```

- com.javastudy.ch04.classdefinition

```
public class AnimalUseExam01 {
    public static void main(String[] args) {
```

```
        /* new 연산자를 사용해 Animal01 클래스의 인스턴스를 생성하고
         * dog라는 변수에 인스턴스가 생성된 메모리 주소의 참조 값을 할당한다.
         * 변수 dog는 인스턴스의 참조 값을 저장하는 변수로 참조형 변수가 된다.
         *
         * name=null, age=0, sound=null, kind=null로 초기화됨
         **/
        Animal01 dog = new Animal01();

        // 참조형 변수 cheolSu를 이용해 인스턴스의 속성을 설정
        dog.name = "희망이";
        dog.age = 5;
        dog.sound = "멍멍";
        dog.kind = "강아지";
        dog.eat();

        /* sleep() 메서드를 호출하면서 메서드의 인수로 "희망이2"를 지정하고 있다.
         * 메서드를 정의하거나 사용하면서 매개변수(Parameter)와 인수(Argument)라는
         * 용어를 사용하게 되는데 이 용어는 메서드 사용에서 혼용되어 사용되는 경우가 많다.
         * 매개변수(Parameter)는 외부에서 메서드 안으로 전달되는 값을 저장하는 변수를
         * 의미하며 인수(Argument)는 메서드를 호출할 때 메서드에 입력되는 값을 의미 한다.
         **/
        dog.sleep("희망이2");
        dog.cry("강아지", "왈왈");
        System.out.println(dog);
        System.out.println();

        Animal01 cat = new Animal01();
        cat.name = "야옹이";
        cat.age = 4;
        cat.sound ="야옹야옹";
        cat.kind = "고양이";
        cat.eat();
        cat.sleep("나비");
        cat.cry("고양이", "야옹");
        System.out.println(cat);
        System.out.println();
    }
}
```

4.2 생성자(Constructor)

▶ 학생 정보를 다루는 클래스 정의하고 생성자를 이용해 인스턴스 맴버 초기화

- com.javastudy.ch04.classdefinition

```
public class Student {

    /* 속성(Property)을 private으로 선언해 외부로 부터 직접 접근을 차단(정보은닉)
     *
     * 캡슐화(Encapsulation)는 객체가 독립적인 역할을 수행할 수 있도록 속성과 기능을
     * 하나의 클래스로 묶어 관리하는 것을 말한다. 캡슐화를 통해서 객체가 가진 중요한
     * 속성과 기능은 외부에 노출되지 않도록 하고 꼭 필요한 속성과 기능만 외부에 노출할
     * 수도 있다. 이렇게 실제 구현된 객체의 중요한 부분은 외부에 노출되지 않게 하고 필요한
     * 부분만 외부에 노출 시키는 것을 정보은닉(Information Hiding)이라고 한다.
     * 정보은닉은 접근 지정자(접근 제어자, Access Modifier)를 통해 적절한 제어 권한이
     * 있는 사용자에게만 객체 내부의 정보에 접근할 수 있도록 구현하는데 아래와 같이
     * 객체의 속성이나 기능에 private을 적용하면 클래스 안에 구현된 내용을 외부에서
     * 볼 수 없게 되며 객체가 필요에 의해서 외부로 노출하는 속성과 메서드를 통해서만
     * 접근할 수 있도록 할 수 있다.
     **/
    private String name;
    private int age;

    /* 생성자의 이름은 클래스 이름과 같아야 하며 반환 타입은 지정하지 않는다.
     * 아래와 같이 () 부분에 파라미터가 없는 것을 기본 생성자라고 한다.
     **/
    public Student() {}

    /* 인스턴스가 생성되면서 name 필드를 초기화 하는 생성자
     * 생성자를 생성자 메서드라고도 부르며 생성자는 객체 생성 후에 그 객체의 초기화를
     * 위해 사용된다. 외부로부터 입력된 값을 사용해 객체를 초기화 하려면 아래와 같이
     * 생성자의 () 부분에 필요한 타입의 변수를 선언하여 외부로부터 데이터를 받을 수
     * 있도록 생성자를 정의하면 된다. 이는 생성자 뿐만이 아니라 메서드도 외부로 부터
     * 입력된 값이 필요하다면 메서드의 () 부분에 필요한 데이터 타입의 변수를 선언해
     * 외부로부터 필요한 데이터를 입력 받을 수 있도록 메서드를 정의하면 된다.
     **/
    public Student(String name) {
        this.name = name;
    }

    /* 인스턴스가 생성되면서 name과 age 필드를 초기화 하는 생성자
     * 여러 생성자를 정의하는 것을 생성자 오버로딩(Overloading)이라고 한다.
     **/
    public Student(String name, int age) {
        this.name = name;
```

```
        this.age = age;
    }

    /* 정보 보호를 위해 외부에서 직접 접근하는 것을 차단한 속성(Property)에
     * 접근할 수 있도록 다음과 같이 public의 getter와 setter 메서드를 제공 한다.
     **/
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public int getAge() {
        return age;
    }
}
```

- com.javastudy.ch04.classdefinition

```
public class StudentUseExam {

    public static void main(String[] args) {

        /* 클래스를 통해서 아래와 같이 여러 개의 객체를 생성할 수 있어서 클래스를 객체의
         * 설계도 또는 객체를 찍어 내듯이 여러 개 생성할 수 있어서 객체의 틀이라고 한다.
         *
         * 클래스를 통해 메모리에 실제 만들어지는 하나하나의 객체를 클래스의 인스턴스라고
         * 부르며 인스턴스는 클래스를 통해 실제 메모리에 실체화된 객체를 의미하는 용어이다.
         *
         * 기본 생성자를 이용해 Student 클래스의 인스턴스 생성
         * new 연산자로 Student 클래스의 인스턴스가 생성되고 기본 생성자에 의해 초기화
         * 된다. 초기 값을 지정하지 않았기 때문에 각 필드의 데이터 타입에 따라 기본 값으로
         * 초기화 된다.
         * name = null, age = 0으로 초기화 된다.
         **/
        Student s1 = new Student();

        /* 이름을 설정할 수 있는 생성자를 이용해 Student 클래스의 인스턴스 생성
         * new 연산자로 Student 클래스의 인스턴스가 생성되고 name은 생성자의 인수에
         * 지정한 값으로 초기화 되고 age는 기본 값으로 초기화 된다.
         * name = "이순신", age = 0으로 초기화 된다.
```

```
         **/
        Student s2 = new Student("이순신");

        /* 이름과 나이를 설정할 수 있는 생성자를 이용해 Student 클래스의 인스턴스 생성
         * new 연사자로 Student 클래스의 인스턴스가 생성되고 name과 age는 생성자의
         * 인수에 지정한 값으로 초기화 된다.
         * name="홍길동", age = 17로 초기화 된다.
         **/
        Student s3 = new Student("홍길동", 17);

        /* 위에서 생성한 Student 클래스의 각 인스턴스가 가지고 있는 속성(Property) 값을
         * getter 메소드를 사용해 출력할 수 있다. Student 클래스의 속성은 모두 private으로
         * 선언되어 있으므로 참조 변수를 통해 직접 접근할 수 없고 setter와 getter 메서드를
         * 통해서 데이터를 설정하거나 현재 객체가 가지고 있는 데이터를 읽어올 수 있다.
         **/
        System.out.println(
                "첫 번째 학생 - 이름 : " + s1.getName() + ", 나이 : " + s1.getAge());
        System.out.println(
                "두 번째 학생 - 이름 : " + s2.getName() + ", 나이 : " + s2.getAge());
        System.out.println(
                "세 번째 학생 - 이름 : " + s3.getName() + ", 나이 : " + s3.getAge());
    }
}
```

[연습문제 4-2]
상품정보를 저장하는 Product 클래스 정의하고 사용하기

상품 정보를 저장할 수 있는 아래와 같은 Product 클래스를 정의하고 Product 클래스의 인스턴스를 생성하여 상품 정보를 출력하는 프로그램을 작성해 보자.

[연습문제 4-1]과 같은 프로젝트의 “com.javastudy.ch04.classdefinition” 패키지에 아래 요구사항에 따라서 새로운 클래스를 생성하여 프로그램을 작성하시오.

1. Product 클래스는 상품명, 가격, 제조사, 상품설명을 저장할 수 있는 속성(Property)을 정의 하고 있다.
2. Product 클래스의 각 속성(Property)은 접근지정자가 private으로 선언되어 있으며 각 속성에 접근할 수 있는 public 접근지정자를 가진 setter와 getter가 정의되어 있다.
3. Product 클래스는 속성에 저장된 상품 정보를 문자열로 반환하는 toString 메서드를 오버라이딩 하고 있다.
4. ProductPrint 클래스를 작성하여 Product 클래스의 인스턴스를 3개 생성하고 각각의 속성(Property) 값을 설정한 후 이를 출력하는 프로그램을 작성해 보자.

[실행결과]

상품 리스트

아메리카노 1+1(2990), 제조사 : 스타벅스, 상품설명 : 스타벅스 오리지날 아메리카노 커피 1+1

뉴그랜저(32500000), 제조사 : 현대자동차, 상품설명 : 3000cc 동급 최강 승용차

데스크탑5(799000), 제조사 : 삼성전자, 상품설명 : CUP 6코어 i5 10400F 4.3GH

4.3 맴버 변수와 맴버 메서드

▶ 클래스 맴버와 인스턴스 맴버

- com.javastudy.ch04.member

```
public class Employee {

    /* 사번의 기준이 되는 코드 - 회사의 이니셜을 사용하도록 만듦
     * 회사의 이니셜은 모든 사원의 사번에 공통으로 적용해야 되므로 static으로 선언함
     * 예 : 삼성(SAM_), 엘지(LG_) 등
     *
     * Employee 클래스가 처음 사용될 때 이 클래스의 정보는 JVM의 메모리 공간 중
     * Method Area라는 메모리 공간으로 로딩되는데 이 때 static이 붙은 변수와
     * 메서드도 클래스를 따라서 같이 로딩된다. 그래서 static이 붙은 변수와 메서드를
     * 클래스 맴버라고 부른다. 아래의 baseCode는 static이 붙은 맴버 변수이므로
     * 클래스 맴버 변수라고 부르며 static 메서드를 클래스 맴버 메서드라고 부른다.
     *
     * 클래스 정보는 그 클래스가 사용될 때 단 한 번만 메모리에 로딩되며 static 변수는
     * 클래스를 따라 생성되기 때문에 클래스가 로딩될 때 단 하나만 생성된다. static
     * 변수는 클래스를 따라서 하나만 생성되기 때문에 여러 인스턴스에서 하나의 static
     * 변수를 참조하게 된다. 그러므로 static 변수는 여러 인스턴스에서 공용으로 사용하는
     * 공유 변수라고 할 수 있다.
     **/
    public static String baseCode = "SM_";

    /* static 예약어(키워드)가 붙지 않은 맴버 변수를 인스턴스 맴버 변수라고 부른다.
     * 이는 새로운 인스턴스가 JVM의 Heap 이라는 영역에 생성될 때 마다 인스턴스를 따라서
     * 생성되기 때문에 인스턴스 맴버라고 부른다. 인스턴스 맴버 변수는 각각의 인스턴스 별로
     * 생성되기 때문에 각각의 인스턴스가 현재 가지고 있는 데이터를 저장하는 속성이다.
     *
     * static이 붙지 않은 맴버 메서드를 인스턴스 맴버 메서드라고 부른다.
     * 인스턴스 맴버 메서드는 인스턴스 맴버 변수와 다르게 인스턴스를 따라서 Heap 영역에
     * 생성되는 것이 아니라 Method Area에 생성된다. 메서드는 특정 명령을 실행하는
     * 명령의 집합으로 인스턴스가 달라도 동일한 기능을 수행하기 때문에 인스턴스 마다
     * 새롭게 생성할 필요가 없고 클래스 별로 구분해 저장하면 인스턴스에서 사용할 수 있다.
     **/
    public String sabun;
    public String name;
    public String gender;
    public int age;

    public Employee(String serialNo, String name, String gender, int age) {
        // 사번의 기준이 되는 번호인 년도와 매개변수로 넘겨받은 값을 조합해 사번을 만듦
        this.sabun = baseCode + serialNo;
        this.name = name;
```

```
        this.gender = gender;
        this.age = age;
    }
}
```

- com.javastudy.ch04.member

```
public class EmployeeUseExam {

    public static void main(String[] args) {

        // baseCode는 클래스 맴버이므로 인스턴스 생성 없이 클래스 명으로 바로 접근이 가능함
        System.out.println("사번의 기준 번호 : " + Employee.baseCode);

        /* new 연산자를 사용해 Employee 클래스의 인스턴스를 생성하고 생성자를
         * 이용해 인스턴스를 초기화 한 후 변수 emp01에 참조 값을 할당하고 있다.
         **/
        Employee emp01 = new Employee("2020005", "홍길동", "남성", 28);

        // 인스턴스 맴버는 반드시 인스턴스를 생성한 후에 접근이 가능
        System.out.println("emp01의 사번 : " + emp01.sabun);
        System.out.println("emp01의 이름 : " + emp01.name);

        // 인스턴스 변수로 클래스 멤버에 접근은 가능하나 클래스 명을 사용하는 것이 정석이다.
        System.out.println("emp01 사번의 기준 코드 : " + emp01.baseCode);
        System.out.println();

        /* new 연산자를 사용해 Employee 클래스의 인스턴스를 생성하고 생성자를
         * 이용해 인스턴스를 초기화 한 후 변수 emp02에 참조 값을 할당하고 있다.
         **/
        Employee emp02 = new Employee("5032150", "어머나", "여성", 27);

        // 인스턴스 맴버는 반드시 인스턴스를 생성한 후에 접근이 가능하다.
        System.out.println("emp02의 성별 : " + emp01.gender);
        System.out.println("emp02의 나이 : " + emp01.age);

        // 인스턴스 변수로 클래스 멤버에 접근은 가능하나 클래스 명을 사용하는 것이 정석이다.
        System.out.println("emp02 사번의 기준 코드 : " + Employee.baseCode);
        System.out.println();

        /* 클래스 정보는 그 클래스가 사용될 때 단 한 번만 메모리에 로딩되며 static 변수는
         * 클래스를 따라 생성되기 때문에 클래스가 로딩될 때 단 하나만 생성된다. static
         * 변수는 클래스를 따라서 하나만 생성되기 때문에 여러 인스턴스에서 하나의 static
         * 변수를 참조하게 된다. 그러므로 static 변수는 여러 인스턴스에서 공용으로 사용하는
```

```
         * 공유 변수라고 할 수 있다.
         **/
        emp02.baseCode = "TM_";

        /* 위에서 baseCode 값을 변경 했으므로 emp01 인스턴스가 참조하는
         * baseCode 값도 "SM_"에서 "TM_"로 변경된 것을 확인 할 수 있다.
         **/
        System.out.println("emp01 사번의 기준 코드 : " + emp01.baseCode);
        System.out.println("emp02 사번의 기준 코드 : " + emp02.baseCode);
    }
}
```

[연습문제 4-3]
4칙 연산을 수행하는 Calculator 클래스 정의하기

4칙 연산 기능을 제공하는 아래와 같은 Calculator 클래스를 정의하고 Calculator 클래스의 메소드를 사용하여 4칙 연산을 수행하는 프로그램을 작성해 보자.

[연습문제 4-1]과 같은 프로젝트에 “com.javastudy.ch04.member” 패키지를 만들고 이 패키지에 새로운 클래스를 생성하여 프로그램을 작성하시오.

1. Calculator 클래스는 두 수를 입력 받아 각각 덧셈, 뺄셈, 나눗셈, 곱셈을 수행한 후 그 결과를 리턴 하는 4개의 메소드를 가지고 있다.
2. Calculator 클래스 사용하는 CalculatorTest 클래스를 만들어 main() 메서드에서 4칙 연산의 결과를 출력하는 프로그램을 작성하시오.

[실행결과]

```
3 + 5 = 8
100 - 78 = 22
101 x 23 = 2323
400 / 25 = 16
```

4.4 오버로딩

▶ 덧셈 계산기의 생성자 오버로딩과 메서드 오버로딩

- com.javastudy.ch04.overloading

```
public class Calculator {

    int x;
    int y;

    /* 오버로딩(Overloading)
     * 오버로딩은 하나의 클래스 안에서 같은 이름을 가진 메서드를 여러 개 정의할
     * 수 있는 것을 말한다. 오버로딩은 자바의 객체지향에서 다형성(Polymorphism)을
     * 구현하는 방법 중 하나이며 생성자 오버로딩(Constructor Overloading)과
     * 메서드 오버로딩(Method Overloading)이 있다.
     * 다형성이란 하나의 객체가 여러 자료형을 가질 수 있는 성질로 클래스나 메서드가
     * 다양한 방식으로 동작이 가능한 것을 말한다.
     * 오버로딩을 통해서 다형성이 구현된다는 것은 같은 이름을 가진 메서드를 여러 개
     * 정의해서 이름은 하나이지만 다양한 기능을 구현할 수 있다는 것을 의미한다.
     * 다시 말해 오버로딩을 통해서 하나의 이름으로 다양한 기능을 제공하는 것을 의미한다.
     **/

    /* 생성자 오버로딩(Constructor Overloading)
     * 아래 생성자와 같이 하나의 클래스에 여러 개의 생성자를 정의하는 것을 생성자
     * 오버로딩이라고 한다. 생성자에서 매개변수의 개수가 다르거나 매개변수의
     * 타입이 다르면 생성자 오버로딩이 성립된다. 또한 매개변수의 개수가 같다
     * 하더라도 매개변수의 순서에 따라 타입이 다르면 생성자 오버로딩이 성립된다.
     **/
    public Calculator() {
        x = 10;
        y = 20;
    }

    public Calculator(int x, int y) {
        this.x = x;
        this.y = y;
    }

    /* 메서드 오버로딩(Method Overloading)
     * 아래 add() 메서드와 같이 하나의 클래스에 같은 이름을 가진 메서드를 여러 개
     * 정의할 수 있는 것을 메서드 오버로딩이라 한다. 메서드의 반환 타입은 중요하지
     * 않고 메서드에서 오직 매개변수의 개수가 다르거나 매개변수의 타입이 다르면
     * 메서드 오버로딩이 성립된다. 또한 매개변수의 개수가 같다 하더라도 매개변수의
     * 순서에 따라 타입이 다르면 메서드 오버로딩이 성립된다.
     **/
```

```
    public int add() {
        return x + y;
    }

    /* int 형 숫자 두 개를 입력 받아 두 수를 더하여 반환 하는 메서드
     **/
    public int add(int x, int y) {
        return x + y;
    }

    // int 형 숫자 세 개를 입력 받아 모두 더하여 반환 하는 메서드
    public int add(int x, int y, int z) {
        return x + y + z;
    }

    // 실수 형 숫자 두 개를 입력 받아 두 수를 더하여 반환 하는 메서드
    public double add(double x, double y) {
        return x + y;
    }

    // 실수 형 숫자 세 개를 입력 받아 모두 더하여 반환 하는 메서드
    public double add(double x, double y, double z) {
        return x + y + z;
    }

    /* 가변인수 메서드(Variable Argument)
     * 가변인수로 정의된 메서드를 호출할 때 1개 이상의 인수를 지정할 수 있는
     * 동적 인수 지정 방식으로 이 클래스에는 add() 메서드의 매개변수가 2개와
     * 3개짜리가 이미 선언되어 있기 때문에 add() 메서드를 호출할 때
     * 1개의 인수를 지정하거나, 4개 이상 인수를 지정하여 호출할 때 이 가변인수
     * 메서드가 호출된다. 메서드 인수의 개수에는 제한이 없다.
     **/
    public int add(int... nums) {
        int sum = 0;

        // 가변인수로 선언된 매개변수는 메서드 내부에서 배열처럼 접근할 수 있다.
        for(int i = 0; i < nums.length; i++) {
            sum += nums[i];
        }
        return sum;
    }
}


public class CalculatorTest {
```

```
    public static void main(String[] args) {

        // 생성자 오버로딩이 되어 있기 때문에 두 가지 방법으로 객체를 생성할 수 있다.
        Calculator cal1 = new Calculator();
        Calculator cal2 = new Calculator(100, 200);

        System.out.println("cal2.add() 결과 : " + cal2.add());
        System.out.println("add() 결과 : " + cal1.add());
        System.out.println("add(5, 10) 결과 : " + cal1.add(5, 10));
        System.out.println("add(5, 10, 15) 결과 : " + cal1.add(5, 10, 15));
        System.out.println("add(10.4, 1.7) 결과 : " + cal1.add(10.4, 1.7));
        System.out.println("add(5.7, 1.3, 3.5) 결과 : " + cal1.add(5.7, 1.3, 3.5));

        /* Calculator 클래스에 add() 메서드의 매개변수가 2개와 3개짜리가 이미 선언되어
         * 있기 때문에 가변 인수를 가진 add(int... nums) 메서드가 호출되게 하려면
         * 이 메서드를 호출할 때 1개 또는 4개 이상의 인수를 지정하여 호출하면 된다.
         **/
        System.out.println("add(1, 2, 3...) 결과 : "
                + cal1.add(1, 2, 3, 4, 5, 6, 7, 8, 9, 10));
    }
}
```

▶ 자신의 다른 생성자를 호출하는 this()와 생성자 오버로딩

- com.javastudy.ch04.overloading

```
public class Player {

    // Player 클래스는 인스턴스 변수(Property - 속성) 4개를 가지고 있다.
    private String name;
    private int age;
    private String gender;
    private String nationality;

    /* 생성자를 정의할 때 생성자의 이름은 클래스 이름과 동일해야 하며
     * 생성자도 메서드에 일종으로 여러 개 정의할 수 있는 생성자 오버로딩이 가능하다.
     * 생성자는 객체가 생성된 후 그 객체의 속성을 초기화 할 목적으로 주로 사용된다.
     **/
    public Player(String name, int age, String gender) {

        /* this() 생성자는 같은 클래스 내의 다른 생성자를 호출 한다.
         * this() 생성자는 항상 생성자의 맨 첫줄에 기술해야 한다.
         * this() 생성자를 이용해 매개변수가 4개인 생성자를 호출하고 있다.
         **/
```

```
        this(name, age, gender, "대한민국");
    }

    public Player(String name, String gender, String nationality) {

        // this() 생성자를 이용해 매개변수가 4개인 생성자를 호출하고 있다.
        this(name, 20, gender, nationality);
    }

    public Player(String name, int age, String gender, String nationality) {

        // 매개변수로 받은 데이터를 인스턴스 변수에 대입하고 있다.
        this.name = name;
        this.age = age;
        this.gender = gender;
        this.nationality = nationality;
    }

    public static void main(String[] args) {

        Player p1 = new Player("크리스", "여성", "미국");
        Player p2 = new Player("마이클", 25, "남성", "호주");

        System.out.println("\t\t\t## 출전 선수 ##");
        System.out.println("\t국가명\t\t이름\t\t나이\t\t성별");
        System.out.println("\t" + p1.nationality + "\t\t" + p1.name
                + "\t\t" + p1.age + "세\t\t" + p1.gender);
        System.out.println("\t" + p2.nationality + "\t\t" + p2.name
                + "\t\t" + p2.age + "세\t\t" + p2.gender);
    }
}
```

[연습문제 4-4]
Member 클래스의 생성자 오버로딩 구현하기

다음과 같이 회원 한 명의 정보를 저장할 수 있는 Member 클래스를 정의하고 사용하는 프로그램을 작성해 보자.

[연습문제 4-2]와 같은 프로젝트에 “com.javastudy.ch04.overloading” 패키지를 만들고 이 패키지에 Member 클래스를 정의하고 Member 클래스를 사용하는 MemberPrint 클래스를 작성하시오.

1. Member 클래스는 회원의 이름, 나이, 아이디, 비밀번호, 성별, 주소를 저장할 수 있는 속성(Property)을 가지고 있다.
2. Member 클래스는 이름과 나이를 매개변수로 가지는 생성자, 이름, 나이, 아이디, 비밀번호를 매개변수로 가지는 생성자, 전체 속성을 매개변수로 가지는 3개의 생성자를 가지고 있다.
3. Member 클래스의 속성은 모두 private으로 정의되어 있으며 각각의 속성에 접근할 수 있는 getter메서드와 setter 메서드를 가지고 있다.
4. Member 클래스는 속성에 저장된 회원 정보를 문자열로 반환하는 toString() 메서드를 오버라이딩 하고 있다.
5. MemberPrint 클래스를 별도의 소스 파일로 만들고 이 클래스에 프로그램 진입점인 main() 메서드를 만들어 Member 클래스에 정의되어 있는 생성자 3개를 모두 사용하여 Member 클래스의 인스턴스를 생성하고 아래 실행결과와 같이 출력되도록 프로그램을 작성하시오. 또한 생성자의 매개변수가 2개와 4개로 생성한 인스턴스는 모든 속성이 설정되지 않았으므로 setter 메서드를 사용해 생성자에서 초기화 하지 못한 속성을 설정하여 회원의 모든 정보가 출력될 수 있도록 작성하시오.

[실행결과]

```
이름 : 홍길동, 나이 : 25, 아이디 : hong, 비밀번호 : 1234, 성별 : 남성, 주소 : 서울 구로구 구로동 1번지
이름 : 이순신, 나이 : 30, 아이디 : leesunshin, 비밀번호 : 4321, 성별 : 남성, 주소 : 서울 강서구 화곡동 3번지
이름 : 유관순, 나이 : 35, 아이디 : midas, 비밀번호 : 1111, 성별 : 여성, 주소 : 경기도 부천시 오정구 고강동
```

5. 상속과 다형성

5.1 상속

▶ 클래스 상속하기

- com.javastudy.ch05.inheritance

```
public class Tv {

    protected String name;
    protected boolean power;
    protected int channel;

    public Tv(String name) {
        this.name = name;
    }

    // Tv 전원을 On/Off 하는 메서드
    public void power() {
        power = !power;
    }

    // Tv 채널을 1씩 증가시키는 메서드
    public int channelUp() {
        return ++channel;
    }

    // Tv 채널을 1씩 감소시키는 메서드
    public int channelDown() {
        return --channel;
    }
}
```

- com.javastudy.ch05.inheritance

```
/* 자바에서 다른 클래스를 상속받기 위해서는 아래와 같이 클래스 선언부에
 * extends 키워드를 사용해 상속받을 클래스 이름을 지정하면 된다.
 *
 *      public class 클래스 이름 extends 상속받을 클래스 이름
 *
 * 상속하는 클래스를 부모 클래스, 기반 클래스, 상위 클래스 등으로 부르며
 * 상속받는 클래스를 자식 클래스, 서브 클래스, 하위 클래스 등으로 부른다.
 * 아래 SmartTv 클래스와 같이 extends 키워드를 사용해 Tv 클래스를 상속 받으면
 * 부모 클래스인 Tv 클래스가 가지고 있는 모든 속성(필드)과 모든 기능(메서드)을
```

```
 * 자식 클래스인 SmartTv 클래스에서 자신의 것과 마찬가지로 사용할 수 있게 된다.
 *
 * 부모 클래스인 Tv 클래스와 자식 클래스인 SmartTv 클래스를 상속 관계가
 * 아닌 각각 개별적인 클래스로 작성하면 SmartTv 클래스에서는 Tv 기능을
 * 구현하기 위해서 Tv 클래스에 있는 동일한 코드를 다시 한 번 작성해야 한다.
 * 이렇게 각각 Tv 클래스와 SmartTv 클래스에서 필요한 Tv 기능을 구현하기
 * 위해서 동일한 코드의 중복이 발생하게 된다. 코드의 중복은 에러율을 높이고
 * 유지보수 등의 작업을 어렵게 만들게 되므로 결국에는 소프트웨어 개발과
 * 유지보수 비용을 증가시키는 원인이 된다. 하지만 상속을 활용하면 코드의
 * 중복을 최소화 하고 재사용성을 높일 수 있는 효율적인 프로그래밍을 할 수 있다.
 **/
// Tv 클래스를 상속 받아서 SmartTv의 속성과 동작을 정의한 클래스
public class SmartTv extends Tv {

    boolean isInternet;

    /* 아래는 부모 클래스인 Tv 클래스로부터 상속 받은 power() 메서드를
     * 자식인 SmartTv 클래스에서 다시 정의 하고 있는데 이렇게 상속 받은
     * 메서드를 자식이 자신에 맞게 다시 정의하는 것을 메서드 재 정의라고
     * 하며 이런 객체지향 기법을 메서드 오버라이딩 이라고 부른다.
     **/
    @Override
    public void power() {
        power = !power;
        if(power) {
            isInternet = true;
        }
    }

    public SmartTv() {

        /* Object 클래스를 제외하고 자바의 모든 클래스의 생성자는 첫 줄에
         * 자신의 다른 생성자 또는 부모의 생성자를 호출해야 한다. 그렇지 않으면
         * 컴파일러에 의해서 모든 생성자의 첫 줄에 부모 클래스의 기본 생성자를
         * 호출하는 super() 코드가 자동으로 추가된다. super()는 부모 클래스의
         * 기본 생성자를 호출하는 코드로써 자식 클래스의 생성자 안에서만 호출
         * 할 수 있으며 항상 생성자의 첫 줄에 작성되어야 한다.
         **/
        this("기본 생성자 TV");
        //super("기본생성자 TV");
    }

    public SmartTv(String name) {
        super(name);
    }
```

```
    public void internet() {
        if(isInternet) {
            System.out.println("요청 사이트로 이동 중...");
        } else {
            System.out.println("인터넷 연결을 확인해주세요");
        }
    }

    public void movieInfo() {
        System.out.println("요청한 영화정보 검색 중...");
    }
}
```

- com.javastudy.ch05.inheritance

```
public class SmartTvTest {

    public static void main(String[] args) {

        SmartTv tv = new SmartTv("우리집 TV");
        tv.power();
        tv.channel = 11;
        System.out.println("현재 채널 : " + tv.channelDown());
        tv.internet();
    }
}
```

▶ 접근 제어자와 super() 생성자

- com.javastudy.ch05.inheritance

```
/* 자바에서 다른 클래스를 상속받기 위해서는 클래스 이름 뒤에 extends 예약어를
 * 사용해 상속받을 클래스를 지정하면 된다. 아래 Computer 클래스는 별도로 상속받는
 * 클래스를 지정하지 않았기 때문에 컴파일러가 클래스 이름 뒤에 extends Object를
 * 붙여서 자바에서 최상위 조상인 Object 클래스를 자동으로 상속받게 만들어 준다.
 **/
public class Computer {

    /* 접근 제어자(Access Modifier, 접근 지정자, 한정자라고도 한다.)는 클래스 또는
     * 생성자와 메소드 그리고 멤버 변수에 사용할 수 있으며 4가지 종류의 제어자가 있다.
     *
     * private : 같은 클래스 내에서만 접근이 가능하다.
     * default : 같은 패키지 내에서만 접근이 가능하다.
     * protected : 같은 패키지와 다른 패키지의 상속관계에서 접근이 가능하다.
```

```
 * public : 접근 제한이 없어 어디서든 접근이 가능하다.
 *
 * default 접근 제어자는 따로 접근 제어자를 붙이지 않는다. 다시 말해 접근 제어자가
 * 붙지 않은 변수, 메서드, 클래스는 default 접근 제어자가 지정된 것이다.
 * 클래스는 default와 public 접근 제어자만 지정할 수 있다.
 **/
protected String name;
protected int price;
protected String spec;

public Computer(String name, int price) {

    /* Object 클래스를 제외하고 자바의 모든 클래스의 생성자는 첫 줄에 자신의
     * 다른 생성자 또는 부모의 생성자를 명시적으로 호출해야 한다. 그렇지 않으면
     * 컴파일러에 의해서 생성자의 첫 줄에 부모 클래스의 기본 생성자를 호출하는
     * super() 코드가 자동으로 추가된다. 이렇게 묵시적인 부모 클래스의 기본
     * 생성자 호출은 자바의 최상위 조상인 Object 클래스까지 거슬러 올라간다.
     * super()는 자식의 생성자 안에서만 호출 할 수 있으며 항상 생성자의 첫 줄에
     * 작성되어야 한다. 이 클래스는 컴파일러에 의해서 자동으로 Object 클래스를
     * 상속 받았기 때문에 super()는 Object 클래스의 기본 생성자를 호출한다.
     * 만약 부모 클래스에 기본 생성자가 존재하지 않으며 컴파일 오류가 발생하므로
     * 부모 클래스에 존재하는 생성자가 호출될 수 있도록 코드를 작성해야 한다.
     **/
    this.name = name;
    this.price = price;
    System.out.println("Computer(name, price) 생성자 호출됨");
}

public Computer(String name, int price, String spec) {
    this.name = name;
    this.price = price;
    this.spec = spec;
    System.out.println("Computer(name, price, spec) 생성자 호출됨");
}

public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public int getPrice() {
    return price;
}
public void setPrice(int price) {
```

```
        this.price = price;
    }
    public String getSpec() {
        return spec;
    }
    public void setSpec(String spec) {
        this.spec = spec;
    }

    public String toString() {
        System.out.println("Computer.toString 메소드 호출됨");
        return name + "\t" + spec + "\t" + price;
    }
}
```

- com.javastudy.ch05.inheritance2

```
/* 자바에서 다른 클래스를 상속받기 위해서는 아래와 같이 클래스 이름 뒤에 extends
 * 예약어를 이용해 상속받을 클래스를 지정하면 된다. 아래와 같이 Computer 클래스를
 * 상속받게 되면 Computer 클래스가 정의한 모든 속성(필드)과 모든 기능(메서드)을
 * 상속받아 자기 자신의 것과 마찬가지로 자유롭게 사용할 수 있게 된다.
 **/
public class NoteBookComputer extends Computer {

    /* 접근 제어자(Access Modifier, 접근 지정자, 한정자라고도 한다.)는 클래스 또는
     * 생성자와 메소드 그리고 멤버 변수에 사용할 수 있으며 4가지 종류의 제어자가 있다.
     *
     * private : 같은 클래스 내에서만 접근이 가능하다.
     * default : 같은 패키지 내에서만 접근이 가능하다.
     * protected : 같은 패키지와 다른 패키지의 상속관계에서 접근이 가능하다.
     * public : 접근 제한이 없어 어디서든 접근이 가능하다.
     *
     * default 접근 제어자는 따로 접근 제어자를 붙이지 않는다. 다시 말해 접근 제어자가
     * 붙지 않은 변수, 메서드, 클래스는 default 접근 제어자가 지정된 것이다.
     * 클래스는 default와 public 접근 제어자만 지정할 수 있다.
     **/

    /* 자식 클래스에서 부모 클래스의 멤버 변수 이름과 동일한 이름의 변수를 선언할 수 있다.
     * 변수의 타입이 달라도 하나의 클래스에는 동일한 이름을 가진 멤버 변수를
     * 선언할 수 없다. 하지만 자식 클래스에서 부모의 멤버 변수와 동일한 이름을
     * 가진 멤버 변수를 중복해서 선언할 수 있다.
     **/
    protected String spec;
```

```
    public NoteBookComputer(String name, int price, String spec) {

        /* Object 클래스를 제외하고 자바의 모든 클래스의 생성자는 첫 줄에 자신의
         * 다른 생성자 또는 부모의 생성자를 명시적으로 호출해야 한다. 그렇지 않으면
         * 컴파일러에 의해서 생성자의 첫 줄에 부모 클래스의 기본 생성자를 호출하는
         * super() 코드가 자동으로 추가된다. 이렇게 묵시적인 부모 클래스의 기본
         * 생성자 호출은 자바의 최상위 조상인 Object 클래스까지 거슬러 올라간다.
         * super()는 자식의 생성자 안에서만 호출 할 수 있으며 항상 생성자의 첫 줄에
         * 작성되어야 한다. 이 클래스는 Computer 클래스를 상속 받았으므로 아래와
         * 같이 super(name, price) 생성자를 호출하면 부모인 Computer 클래스의
         * super(name, price) 생성자가 호출된다.
         * 만약 아래에서 명시적으로 super(name, price) 생성자를 호출하지 않는다면
         * 컴파일러에 의해서 super() 코드가 자동으로 추가되기 때문에 컴파일 오류가
         * 발생한다. 그 이유는 Computer 클래스에는 기본 생성자가 없기 때문이다.
         * 그러므로 아래와 같이 Computer 클래스에 존재하는 생성자를 호출해야 한다.
         **/
        super(name, price);
        this.spec = spec;
        System.out.println("NoteBookComputer(name, price, spec) 생성자 호출됨");
    }

    public String getSpec() {
        return spec;
    }
    public void setSpec(String spec) {
        this.spec = spec;
    }

    public String toString() {
        System.out.println("NoteBookComputer.toString 메소드 호출됨");
        return name + "\t" + spec + "\t" + price;
    }

    public void info() {

        /* 자식 클래스에서 자신의 인스턴스 멤버 변수와 동일한 이름을 가진 부모 클래스의
         * 인스턴스 멤버 변수에 접근하기 위해서 부모의 인스턴스를 참조하는 부모 참조 변수인
         * super를 사용해 부모의 인스턴스 변수에 접근할 수 있다. 모든 인스턴스 메서드에서
         * 자기 자신의 인스턴스를 참조하는 자기 참조 변수 this와 부모의 인스턴스를 참조하는
         * 부모 참조 변수 super를 사용할 수 있다. this와 super는 인스턴스와 관련된 변수
         * 이므로 static 메소드에서는 사용할 수 없다.
         **/
        System.out.println("Super Info : " + super.spec + ", Sub Info : " + spec);
    }
}
```

- com.javastudy.ch05.inheritance

```
public class ComputerTest {

    public static void main(String[] args) {

        Computer com = new Computer("My Desktop", 769000, "I5 4.2GHz 8GB-Ram");
        System.out.println();

        NoteBookComputer noteBook = new NoteBookComputer(
                "My NoteBook", 976000, "I5 4.2GHz 8GB-Ram 15inch");
        System.out.println();

        noteBook.info();
        System.out.println();

        System.out.println(com);
        System.out.println();

        System.out.println(noteBook);
    }
}
```

▶ 메서드 오버라이딩

- com.javastudy.ch05.inheritance

```
public class Phone {

    protected String name;

    public Phone(String name) {

        /* 현재 이 클래스는 다른 클래스를 명시적으로 상속받지 않기 때문에 컴파일러에
         * 의해서 extends Object가 클래스 선언부에 추가되어 Object 클래스를 직접
         * 상속받게 된다. 또한 이 생성자의 첫 줄에 부모 클래스의 기본 생성자를 호출하는
         * super() 코드가 자동으로 추가되어 Object 클래스의 기본 생성자가 호출된다.
         * 이렇게 부모 클래스의 생성자 호출은 현재 클래스와 상속 관계에 있는 모든 부모
         * 클래스의 생성자가 호출될 수 있도록 자바의 최상위 조상인 Object 클래스의
         * 생성자가 호출될 때 까지 거슬러 올라간다.
         **/
        this.name = name;
        System.out.println("Phone(name) 생성자 호출됨");
    }
```

```
    public void sendPhone() {
        System.out.println("Phone.SendPhone() - 전화를 건다.");
    }

    public void receivePhone() {
        System.out.println("Phone.receivePhone() - 전화를 받는다.");
    }
}
```

- com.javastudy.ch05.inheritance

```
public class HandPhone extends Phone {

    protected boolean isEnabled;

    public HandPhone(String name, boolean isEnabled) {

        /* HandPhone 클래스가 상속받은 Phone 클래스에는 기본 생성자가 없으므로
         * Phone 클래스에 정의된 생성자를 아래와 같이 명시적으로 호출해야 한다.
         * 그렇지 않으면 컴파일러에 의해서 부모의 기본 생성자를 호출하는 super()
         * 코드가 자동으로 추가되기 때문에 부모 클래스인 Phone 클래스에 존재하는
         * 생성자를 아래와 같이 명시적으로 호출해야 컴파일 오류가 발생하지 않는다.
         * 이렇게 부모 클래스의 생성자 호출은 현재 클래스와 상속 관계에 있는 모든
         * 부모 클래스의 생성자가 호출될 수 있도록 자바의 최상위 조상인 Object
         * 클래스의 생성자가 호출될 때 까지 거슬러 올라간다.
         **/
        super(name);
        this.isEnabled = isEnabled;
        System.out.println("HandPhone(name, isEnabled) 생성자 호출됨");
    }

    /* Phone 클래스로부터 상속받은 sendPhone() 메서드를 자식인 HandPhone 클래스에서
     * 수정하고 있다. 이렇게 부모로부터 상속 받은 부모의 인스턴스 메서드를 자식 클래스에서
     * 수정하여 다시 정의하는 것을 메서드 오버라이딩(Overriding) 이라고 한다.
     *
     * 부모로부터 상속받은 메서드 이름과 동일하고 메서드의 반환타입과 매개변수가 동일해야
     * 메서드 오버라이딩이 성립된다. 오버라이딩은 부모로부터 물려받은 메서드를 자식 대에서
     * 필요한 기능을 추가 하거나 수정하는 것을 의미하며 이는 부모로부터 상속 받은 메서드를
     * 자식이 필요에 의해서 메서드의 기능을 다시 정의하는 것이므로 메서드 재정의라고도 한다.
     **/
    @Override
    public void sendPhone() {
        System.out.println("HandPhone.SendPhone() 호출됨");
```

```
        if(! isEnabled) {
            System.out.println("기지국 연결을 확인 하세요");
            return;
        }

        /* 부모 참조 변수인 super를 사용해 부모 인스턴스의 sendPhone()
         * 메서드를 호출하고 있다. 이렇게 자식 클래스에서 자신의 인스턴스 멤버
         * (인스턴스 멤버 변수, 인스턴스 멤버 메서드)와 동일한 이름을 가진 부모
         * 클래스의 인스턴스 멤버에 접근하기 위해서 부모의 인스턴스를 참조하는
         * 부모 참조 변수인 super를 사용해 부모의 인스턴스 멤버에 접근할 수 있다.
         * 모든 인스턴스 메서드에서 자기 자신의 인스턴스를 참조하는 자기 참조
         * 변수 this와 부모의 인스턴스를 참조하는 부모 참조 변수 super를
         * 사용할 수 있다. this와 super는 인스턴스와 관련된 변수 이므로
         * static 메소드 안에서는 사용할 수 없다.
         **/
        super.sendPhone();
    }

    @Override
    public void receivePhone() {
        System.out.println("HandPhone.receivePhone() 호출됨");

        if(! isEnabled) {
            System.out.println("기지국 연결을 확인 하세요");
            return;
        }
        super.receivePhone();
    }

    public void game() {
        System.out.println("게임을 시작 합니다.");
    }

    public void setEnabled(boolean isEnabled) {
        this.isEnabled = isEnabled;
    }
    public boolean isEnabled() {
        return isEnabled;
    }
}
```

- com.javastudy.ch05.inheritance2

```
public class PhoneTest {

    public static void main(String[] args) {

        HandPhone hp = new HandPhone("핸드폰", true);
        System.out.println();

        hp.sendPhone();
    }
}
```

▶ 업 캐스팅과 다운 캐스팅

- com.javastudy.ch05.polymorphism

```
public class PhoneTest02 {

    public static void main(String[] args) {

        HandPhone hp = new HandPhone("HandPhone1", true);

        /* 부모 클래스타입 참조 변수에 자식타입의 인스턴스를 생성하여 아래와 같이
         * 할당하면 자동으로 형 변환되어 자식타입의 레퍼런스가 저장된다. 이 때 실제
         * 인스턴스에는 아무런 영향을 주지 않고 참조 변수만 부모타입으로 바뀌는 것 이다.
         **/
        Phone p1 = new HandPhone("HandPhone2", true);

        /* 부모 클래스타입의 참조 변수에 자식타입의 레퍼런스를 대입하고 있다.
         * 부모타입으로 형 변환되는 것을 업 캐스팅이라 하며 자동으로 형 변환 된다.
         **/
        Phone p2 = hp;

        p1.receivePhone();
        System.out.println(p1);
        System.out.println(p2);

        /* 오버라이딩된 메소드가 아니면 부모타입 참조 변수로는 접근이 불가능 하다.
         * 그 이유는 참조변수의 형 변환은 인스턴스에 아무런 영향을 미치지 않고 단지 참조
         * 변수만 부모타입의 변수로 바뀌며 이 때 부모가 정의한 멤버에만 접근할 수 있다.
         **/
        // p2.game();
        p2.receivePhone();
        System.out.println(p2);

        /* 업 캐스팅된 참조 변수를 본래의 자신타입 참조변수에 대입하고 있다.
```

```
         * 부모타입으로 형 변환 되었던 참조변수의 타입이 본래의 자기 타입으로
         * 돌아오는 것을 다운 캐스팅이라 하며 명시적(강제)으로 형 변환해야 한다.
         *
         * 참조변수의 형 변환은 인스턴스에 아무런 영향을 미치지 않고 단지 참조변수의
         * 타입만 변환되는 것으로 그 참조변수가 접근할 수 있는 멤버의 개수를 제한하게 된다.
         * 부모의 멤버는 항상 자식의 멤버 개수와 같거나 적으므로 부모타입으로 형 변환된
         * 참조변수가 접근할 수 있는 멤버는 부모 클래스에 정의된 멤버에 국한된다. 상속받은
         * 멤버가 아닌 자신의 멤버에 접근하려면 업캐스팅 상태에서는 접근이 불가능하고
         * 본래의 자기타입으로 다시 되돌려야 접근할 수 있다. 이렇게 부모타입으로 업캐스팅
         * 되었다가 다시 본래의 자기타입으로 되돌아오는 것을 다운 캐스팅이라고 한다.
         **/
        HandPhone hp2 = (HandPhone) p2;
        hp2.game();

        /* 업 캐스팅 되지 않은 참조 변수를 자식 타입으로 형 변환 시 컴파일은
         * 제대로 되지만 실행 타임에서 ClassCastException이 발생한다.
         * 아래 코드를 주석을 풀고 실행하면 ClassCastException이 발생한다.
         **/
        // Phone p3 = new Phone("전화기");
        // p3 = (HandPhone) p;
    }
}
```

▶ instanceof 연산자를 이용한 객체의 타입 체크(같은 소스 파일에 작성할 것)

- com.javastudy.ch05.polymorphism

```
class Car { }
class Bus extends Car { }
class Bongo extends Bus { }
class Ambulance extends Bus { }

public class InstanceCheck {

    public static void main(String[] args) {

        Car car = new Car();
        Car bus = new Bus();
        Car bongo = new Bongo();
        Car ambulance = new Ambulance();

        /* 자식의 인스턴스를 참조하고 있는 자식 타입의 참조 변수와 부모 클래스 타입을
         * instanceof 연산을 수행하면 true가 반환 된다. 이 처럼 instanceof 연산의
         * 결과가 true이면 참조 변수를 비교한 클래스 타입으로 형 변환이 가능하다.
         * 부모 클래스를 상속 받으면 부모의 모든 멤버를 자식이 포함하고 있기 때문이다.
```

```
         **/
        System.out.println("bus instanceof Car : " + (bus instanceof Car));
        System.out.println("bus instanceof Bus : " + (bus instanceof Bus));
        System.out.println("bus instanceof Object : " + (bus instanceof Object));
        System.out.println();

        /* 부모의 인스턴스를 참조하고 있는 부모 타입의 참조 변수와 자식 클래스 타입을
         * instanceof 연산을 수행하면 false가 반환된다. 그 이유는 상속 관계라
         * 하더라도 부모 클래스의 멤버가 자식 클래스의 멤버 보다 적거나 같기 때문이다.
         **/
        System.out.println("bus instanceof Bongo : " + (bus instanceof Bongo));
        System.out.println();

        /* 자바의 클래스 상속 관계에서 형제 관계는 존재하지 않고 오로지 직계 상속 관계만
         * 가능하다. Bongo 클래스가 Bus 클래스를 상속 하였으므로 bongo instanceof Bus
         * 연산을 수행하면 true가 반환 된다. Ambulance 클래스 또한 Bus 클래스를 상속하여
         * Bongo 클래스와 Ambulance 클래스는 형제 관계지만 bongo instanceof Ambulance
         * 연산을 수행하면 false가 반환 된다.
         **/
        System.out.println("bongo instanceof Bus : " + (bongo instanceof Bus));
        System.out.println("bongo instanceof ambulance : "
                + (bongo instanceof Ambulance));
        System.out.println();

        /* 아무리 상속 관계의 계층이 많아도 자식 타입의 참조 변수와 조상 클래스 타입을
         * instanceof 연산을 수행하면 자식 클래스가 상위의 모든 부모 클래스가 가진
         * 멤버를 포함하고 있기 때문에 true가 반환 된다.
         **/
        System.out.println("ambulance instanceof Object : "
                + (ambulance instanceof Object));
    }
}
```

[연습문제 5-1]
기준 클래스를 정의하고 상속과 오버라이딩 구현하기

다음과 같이 동물의 기본 정보와 동작을 정의한 Animal 클래스를 정의하고 이 클래스를 상속받는 Human, Dog, Cat 클래스를 정의하고 사용하는 프로그램을 작성하시오.

먼저 “JavaStudyCh05Exercise” 프로젝트를 만들고 “com.javastudy.ch05.inheritance” 패키지를 만들어 이 패키지에 필요한 클래스를 생성하여 프로그램을 작성하시오.

1. Animal 클래스는 이름, 종, 나이, 소리를 저장할 수 있는 4개의 속성을 가지고 있다.
2. Animal 클래스의 속성은 모두 private으로 정의되어 있으며 각각의 속성에 접근할 수 있는 setter 메서드와 getter 메서드를 제공하고 있다.
3. Animal 클래스는 먹는다, 울다 등의 기능을 표현한 메서드를 가지고 있다.
4. Animal 클래스는 기본 생성자와 4개의 속성을 한 번에 초기화할 수 있는 생성자를 가지고 있다.
5. Animal 클래스는 자신의 현재 상태를 문자열로 반환하는 toString() 메서드를 오버라이딩 하고 있다.
6. Human 클래스는 취미를 저장할 수 있는 속성을 가지고 있으며 private으로 정의되어 있다.
7. Human 클래스는 5개의 속성을 한 번에 초기활 할 수 있는 생성자를 가지고 있다.
8. Human, Dog, Cat 클래스는 Animal 클래스를 상속하여 Animal 클래스로부터 상속받은 먹는다, 울다 등의 기능을 표현한 메서드와 toString() 메서드를 각각 자신에 맞게 재정의 하고 있다.
9. AnimalTest 클래스를 별도의 소스 파일로 만들고 이 클래스에 프로그램 진입점인 main() 메서드를 만들어 Animal 타입의 참조 변수로 Human, Dog, Cat 클래스의 인스턴스를 생성한 후 아래와 같이 출력되도록 프로그램을 작성하시오.

[실행결과]

희망이는 강아지로 사료를 먹습니다.
희망이는 강아지로 멍멍 짖습니다.
희망이은(는) 강아지로 5살 입니다.

야옹이는 고양이로 사료를 먹습니다.
야옹이는 고양이로 야옹 거립니다.
야옹이은(는) 고양이로 3살 입니다.

철수는 사람으로 아침을 먹습니다.
철수는 사람으로 15살 이며 취미는 독서입니다.
철수는 사람로 엉엉 웁니다.

5.2 추상 클래스

우리는 일상적으로 사물에 대해 이름을 부여하거나 사물들끼리 분류하여 사용하는 경우가 많다. 이렇게 사물을 분류할 때 사물이 가지는 공통된 특징을 바탕으로 분류하는 경우가 많은데 이 때 사물이 가지는 공통된 특징을 뽑아내는 작업을 추상화라 할 수 있다.
각각의 사물들은 저마다 자기만의 특징을 지니고 있으면서 동시에 공통된 특징들로 구성되어 있다. 연관되어 있는 여러 사물들이 가지는 특징을 추출하는 것은 각각의 사물이 가지는 세부적인 특징이 아니라 공통적 요소를 뽑아내기 위해서 특정 기준을 만들고 그 기준에 따라서 대표적이고 개념적인 요소를 추출하기 때문에 추상적이라 할 수 있는 반면 각각의 사물이 가지는 세부적인 특징(속성과 기능)을 구체적으로 나열하는 것을 구체화라 할 수 있다.
객체지향 프로그램에서도 이 추상화와 구체화 개념이 도입되어 있는데 프로그램 설계 단계에서 공통된 특징(속성과 기능)에 대한 이름이나 종류 정도만 추상적으로 정의하고 실제 세부적인 속성과 동작은 프로그래밍 구현 단계에서 공통된 특징을 정의한 주체를 상속 기법을 활용해 구체적으로 구현하게 된다.
자바의 객체지향에서 객체의 특징을 추출해 클래스로 정의하는 것도 추상화 작업이라 할 수 있지만 이보다 더 추상적인 객체의 특징을 정의할 경우가 있는데 이럴 경우 추상 클래스(Abstract Class)와 인터페이스(Interface)라는 특별한 클래스를 통해 구현하게 된다. 먼저 추상 클래스를 정의하는 방법에 대해서 알아보자.

▶ 추상 클래스와 오버라이딩

- com.javastudy.ch05.abstractclass

```
/* 추상 클래스(Abstract Class)란 미완성 클래스를 의미하며 추상 클래스 내부에 미완성
 * 메서드(추상 메서드)를 가지고 있다. 추상 메서드는 메서드 선언부만 있고 구현부가 없는
 * 메서드를 말한다. 추상 클래스는 아래와 같이 클래스 선언부에 abstract 예약어를 사용해
 * 정의할 수 있으며 추상 클래스로 정의되면 미완성 클래스이므로 자체적으로 객체를 생성할
 * 수 없고 상속을 통해서 완성될 수 있다. 또한 추상 클래스는 미완성 메서드인 추상 메서드를
 * 가지고 있지 않아도 abstract 예약어를 사용해 추상 클래스로 선언할 수 있으며 이때도
 * 미완성 클래스가 되므로 자체적으로 객체를 생성할 수 없고 상속을 통해서 완성될 수 있다.
 **/
public abstract class Calculator {

    // 추상 클래스는 인스턴스 맴버 변수를 가질 수 있다.
    int x;
    int y;

    // 추상 클래스는 생성자를 가질 수 있다.
    public Calculator() {
        this(10, 20);
    }

    public Calculator(int x, int y) {
        this.x = x;
        this.y = y;
    }
```

```
    // 추상 클래스는 완성 메서드를 가질 수 있다.
    public int add(int x, int y) {
        return x + y;
    }

    /* 추상 메서드 정의 - 추상 메서드는 선언부만 있고 구현부(몸통)는 없다.
     * 추상 메서드는 아래와 같이 메서드의 특징을 정의하는 선언부만 있고
     * 메서드의 기능을 구체적으로 작성하는 구현부(몸통)가 없는 메서드 이다.
     **/
    public abstract double add(double x, double y);
}

/* 마우스 이벤트를 처리하는 MouseAdapter 클래스는 추상 클래스로 선어되어 있다.
 * 이 클래스는 추상 메서드를 가지고 있지 않지만 추상 클래스로 선언되어 있다.
 * 이 클래스는 MouseListener, MouseWheelListener, MouseMotionListener 3개의
 * 인터페이스에 정의된 추상 메서드를 모두 빈 구현으로 구현하였으므로 이 추상 클래스를
 * 상속받아 필요한 메서드만 오버라이딩 하여 사용할 수 있도록 설계되어 있다.
 **/
class MyMouseAdapter extends MouseAdapter { }
```

- com.javastudy.ch05.abstractclass

```
// 추상 클래스를 상속받을 때도 일반 클래스 상속과 마찬가지로 extends 예약어를 사용한다.
public class EngineeringCalculator extends Calculator {

    public double getX() {
        return x;
    }
    public double getY() {
        return y;
    }

    /* 추상 클래스로부터 상속 받은 추상 메소드 구현
     * EngineeringCalculator 클래스는 Calculator 추상 클래스를 상속받아
     * 상속받은 추상 메서드를 자신에 맞게 모두 구현해야 완성 클래스가 된다.
     *
     * 추상 클래스를 상속받아 부모 클래스의 추상 메서드를 모두 구현하지 않으면
     * 추상 클래스를 상속받은 클래스 또한 미완성 클래스인 추상 클래스가 된다.
     * 자식 클래스에서 상속받은 추상 클래스의 메서드를 모두 구현해야 비로소
     * 인스턴스를 생성할 수 있기 때문에 추상 클래스는 상속을 통해서 완성될 수 있다.
     **/
    @Override
    public double add(double x, double y) {
        return x + y;
```

```
    }
}
```

- com.javastudy.ch05.abstractclass

```
public class CalculatorTest {

    public static void main(String[] args) {

        // 추상 클래스는 직접 인스턴스를 생성 할 수 없다. - 주석을 풀면 에러가 발생한다.
        //Calculator cal = new Calculator();

        Calculator engCal1 = new EngineeringCalculator();
        EngineeringCalculator engCal2 = new EngineeringCalculator();

        // Calculator 클래스로부터 상속받은 add() 메소드가 호출된다.
        System.out.println("engCal1.add(int, int) : " + engCal1.add(30, 50));
        System.out.println("engCal2.add(int, int) : " + engCal2.add(50, 70));
        System.out.println();

        /* 아래의 engCal1.add(30.0, 50.0) 메서드 호출은 참조 변수의 타입이 부모인
         * Calculator 타입이지만 실제 인스턴스가 EngineeringCalculator 타입이므로
         * EngineeringCalculator 클래스에서 Override한 메서드가 실행된다.
         **/
        System.out.println("engCal1.add(double, double) : " + engCal1.add(30.0, 50.0));
        System.out.println("engCal2.add(double, double) : " + engCal2.add(50.0, 70.0));
    }
}
```

5.3 인터페이스

추상클래스에서 언급한 것처럼 자바 객체지향에서 추상화를 구현하는 기법 중 하나가 바로 인터페이스를 이용하는 것이다. 인터페이스는 상수와 추상 메서드만 가질 수 있는 특수한 형태의 클래스로 자체적으로 객체를 생성할 수 없는 껍데기만 존재하는 클래스이지만 기준이 되는 클래스로써 자바의 객체지향에서 아주 중요한 구성요소 이다.
자바는 원칙적으로 클래스간의 단일 상속만을 지원하고 있는데 이런 한계를 극복하기 위해서 인터페이스를 통해 다중 상속을 지원하고 있다. 또한 인터페이스를 상속받는 클래스는 구현부가 없는 미완성 메서드인 추상 메서드를 꼭 구현(재정의)해야 하는 규칙을 지켜야하기 때문에 특정 작업에 대한 표준적인 프로그래밍 기법을 제공하기 위해서 인터페이스를 활용하게 된다. 표준적인 프로그래밍 기법의 좋은 예로 자바와 관계형 데이터베이스 연동을 위해서 제공하는 API(Application Programing Interface)인 JDBC(Java Database Connectivity)를 예로 둘 수 있다. 이 JDBC는 자바에서 관계형 데이터베이스 연동에 필요한 작업을 인터페이스로 정의해 놓고 있으며 관계형 데이터베이스를 제공하는 업체(Oracle, MySQL, MSSQL 등을 제공하는 업체)는 이 인터페이스를 기준으로 데이터베이스 작업에 필요한 클래스를 구현해 접속드라이버로 제공하고 있다. 이 접속 드라이버만 있으면 실제 데이터베이스와 연동하는 프로그래밍을 작성하는 프로그래머는 어떤 관계형 데이터베이스를 사용하던지 간에 관계없이 기준이 되는 인터페이스와 이 인터페이스에 정의된 메서드를 사용해 표준적인 데이터베이스 프로그램을 작성할 수 있게 된다.

▶ 인터페이스를 이용한 다중상속 구현

- com.javastudy.ch05.interfaceclass

```
/* 인터페이스는 추상 메서드(미완성 메서드)와 상수로만 이루어진 특수한 형태의 클래스 이다.
 * 인터페이스는 미완성이므로 자체적으로 인스턴스를 생성할 수 없고 상속을 통해 완성된다.
 **/
public interface Phone {

    /* 인터페이스는 추상 메서드와 상수로만 구성되기 때문에 예약어 없이 아래와 같이
     * 변수로 선언해도 컴파일러에 의해 public static final이 적용되어 상수가 된다.
     * 인터페이스의 모든 상수는 public static final이어야 하므로 이 부분은 생략할 수 있다.
     **/
    String PHONE_SERIAL_NUMBER = "PHONE_001";

    // 전화를 거는 기능을 추상 메소드로 정의
    public abstract void sendPhone();

    /* 전화를 받는 기능을 추상 메소드로 정의
     * 인터페이스는 추상 메서드와 상수로만 구성되기 때문에 예약어 없이 아래와 같이
     * 메서드를 선언해도 컴파일러에 의해 public abstract가 적용되어 추상 메서드가 된다.
     * 인터페이스의 모든 메서드는 public abstract이어야 하므로 이 부분은 생략할 수 있다.
     **/
    void receivePhone();
}
```

- com.javastudy.ch05.interfaceclass

```
/* 인터페이스는 추상 메서드(미완성 메서드)와 상수로만 이루어진 특수한 형태의 클래스 이다.
 * 인터페이스는 미완성이므로 자체적으로 인스턴스를 생성할 수 없고 상속을 통해 완성된다.
 *
 * 자바는 원칙적으로 클래스간의 다중 상속은 지원하지 않지만 인터페이스를 통해서
 * 다중 상속을 지원한다. 인터페이스와 인터페이스 간의 상속은 extends 키워드를 사용해
 * "인터페이스 extends 인터페이스1, 인터페이스2 ..." 와 같이 하나 이상의 인터페이스를
 * 지정할 수 있도록 하여 인터페이스 간의 다중 상속을 정의 할 수 있도록 지원하고 있다.
 **/
public interface MobilePhone extends Phone {

    // 문자 전송 기능을 추상 메소드로 정의
    public abstract void sendSMS();

    /* 문자 받는 기능을 추상 메소드로 정의
     * 인터페이스는 추상 메서드와 상수로만 구성되기 때문에 예약어 없이 아래와 같이
     * 메서드를 선언해도 컴파일러에 의해 public abstract가 적용되어 추상 메서드가 된다.
     * 인터페이스의 모든 메서드는 public abstract이어야 하므로 이 부분은 생략할 수 있다.
     **/
    void receiveSMS();
}
```

- com.javastudy.ch05.interfaceclass

```
/* 클래스에서 인터페이스를 상속받기 위해서는 implements 키워드를 사용한다.
 *
 * 자바는 원칙적으로 클래스간의 다중 상속은 지원하지 않지만 인터페이스를 통해서
 * 다중 상속을 지원한다. 클래스와 인터페이스 간의 상속은 implements 키워드를 사용해
 * "클래스 extends 클래스1 implements 인터페이스1, 인터페이스2 ..." 와 같이 상속받을
 * 클래스와 하나 이상의 인터페이스를 지정하여 다중 상속을 정의 할 수 있도록 지원하고 있다.
 **/
public class MyHomePhone implements Phone {

    protected String name;

    public MyHomePhone(String name) {
        this.name = name;
    }

    /* 인터페이스로부터 상속 받은 추상 메소드 구현
     * MyHomePhone 클래스는 Phone 인터페이스에게 상속받은
     * 추상 메서드를 자신에 맞게 모두 구현해야 비로소 완성 클래스가 된다.
     *
     * 인터페이스를 상속받은 클래스가 인터페이스가 가지고 있는 추상 메서드를
```

```
     * 모두 구현하지 않으면 이 클래스는 미완성 클래스로 추상 클래스가 된다.
     * 인터페이스로부터 상속받은 추상 메서드를 자식 클래스에서 모두 구현해야 비로소
     * 인스턴스를 생성할 수 있기 때문에 인터페이스는 상속을 통해서 완성될 수 있다.
     **/
    @Override
    public void sendPhone() {
        System.out.println(this.name + " 전화 걸기...");
    }

    @Override
    public void receivePhone() {
        System.out.println(this.name + " 전화 받기...");
    }
}
```

- com.javastudy.ch05.interfaceclass

```
/* 클래스에서 아래와 같이 클래스와 인터페이스를 동시에 상속해 다중 상속을 정의할 수 있다.
*
* 자바는 원칙적으로 클래스간의 다중 상속은 지원하지 않지만 인터페이스를 통해서
* 다중 상속을 지원한다. 추상 클래스와 클래스, 클래스와 클래스 그리고 인터페이스와
* 인터페이스 간의 상속은 extends 예약어를 사용하고 인터페이스와 추상 클래스,
* 인터페이스와 클래스 간의 상속은 implements 예약어를 사용해 상속을 정의한다.
*
* 어떤 클래스에서 다중 상속을 받으려면 다음과 같이 클래스와 하나 이상의 인터페이스를
* 지정하여 다중 상속을 정의 할 수 있도록 지원하고 있다.
*
* "클래스 extends 클래스1 implements 인터페이스1, 인터페이스2 ..."
*
* SmartPhone 클래스는 MyHomePhone 클래스를 상속받고 MobilePhone 인터페이스를
* 구현하여 다중 상속을 하고 있다. 이렇게 상속이 이루어지면 SmartPhone 클래스는
* MyHomePhone 타입인 동시에 MobilePhone 타입도 되고 MobilePhone 인터페이스가
* 상속한 Phone 타입도 되므로 이 3가지 클래스나 인터페이스 타입으로 업 캐스팅 할 수 있다.
**/
public class SmartPhone extends MyHomePhone implements MobilePhone {

    protected boolean isEnabled;

    public SmartPhone(String name, boolean isEnabled) {

        /* SmartPhone 클래스가 상속 받은 MyHomePhone 클래스에는 기본 생성자가
         * 정의되어 있지 않으므로 MyHomePhone 클래스에 정의된 생성자를 아래와 같이
         * 명시적으로 호출해야 한다. 그렇지 않으면 컴파일러에 의해서 기본 생성자를 호출하는
         * this() 코드가 첫 줄에 추가되기 때문에 컴파일 오류가 발생한다.
```

```
     **/
    super(name);
    this.isEnabled = isEnabled;
}

@Override
public void sendPhone() {
    if(! isEnabled) {
        System.out.println(this.name + " 기지국 연결을 확인 하세요");
        return;
    }
    super.sendPhone();
}

@Override
public void receivePhone() {
    if(! isEnabled) {
        System.out.println(this.name + " 기지국 연결을 확인 하세요");
        return;
    }
    super.receivePhone();
}

/* 인터페이스로부터 상속 받은 추상 메소드 구현
 * SmartPhone 클래스는 MobilePhone 인터페이스에게 상속받은
 * 추상 메서드를 자신에 맞게 모두 구현해야 비로소 완성 클래스가 된다.
 *
 * 인터페이스를 상속받은 클래스가 인터페이스가 가지고 있는 추상 메서드를
 * 모두 구현하지 않으면 이 클래스는 미완성 클래스로 추상 클래스가 된다.
 * 인터페이스로부터 상속받은 추상 메서드를 자식 클래스에서 모두 구현해야 비로소
 * 인스턴스를 생성할 수 있기 때문에 인터페이스는 상속을 통해서 완성될 수 있다.
 **/
@Override
public void sendSMS() {
    if(! isEnabled) {
        System.out.println(this.name + " 기지국 연결을 확인 하세요");
        return;
    }
    System.out.println(this.name + " 문자 메시지 보내기...");
}

@Override
public void receiveSMS() {
    if(! isEnabled) {
        System.out.println(this.name + " 기지국 연결을 확인 하세요");
```

```
        }
        System.out.println(this.name + " 문자 메시지 받기...");
        return;
    }

    public void webSurfing() {
        System.out.println(this.name + " 브라우저를 화면에 띄운다.");
    }
}
```

- com.javastudy.ch05.interfaceclass

```
public class SmartPhoneTest {

    public static void main(String[] args) {

        SmartPhone sp = new SmartPhone("스마트 폰", true);

        sp.sendPhone();
        sp.sendSMS();
        sp.webSurfing();

        /* SmartPhone 클래스는 MyHomePhone 클래스를 상속하고, MobilePhone
         * 인터페이스를 구현하여 다중 상속을 구현하였기 때문에 SmartPhone 클래스는
         * MyHomePhone 타입인 동시에 MobilePhone 타입도 되고 MobilePhone
         * 인터페이스가 상속한 Phone 타입도 되므로 아래와 같이 위의 3가지 타입으로
         * SmartPhone 클래스의 인스턴스를 참조할 수 있다. 이렇게 하나의 객체가
         * 여러 개의 타입을 가질 수 있는 성질을 다형성(Polymorphism)이라고 한다.
        **/
        Phone p1 = new SmartPhone("스마트 폰1", true);
        MobilePhone p2 = new SmartPhone("스마트 폰2", true);
        MyHomePhone p3 = new SmartPhone("스마트 폰3", true);
    }
}
```

6. 예외처리(Exception Handling)

응용프로그램이 실행 중에 뜻하지 않은 원인으로 프로그램이 오동작하거나 비정상적으로 종료되는 경우가 종종 발생하는데 이러한 경우를 프로그램 에러 또는 프로그램 오류라고 하며 오류가 발생하는 시점에 따라서 컴파일 에러와 런타임 에러로 나뉜다.

컴파일 에러는 소스 코드를 컴파일할 때 발생하는 에러로 컴파일러에 의해서 문법적인 요소를 체크할 때 발생하는 에러가 대부분이며 주로 오타나 문법에 맞지 않는 구문 때문에 발생한다. 런타임 에러는 컴파일이 완료되고 프로그램 실행되는 도중에 발생하는 에러를 말한다.

자바에서 런타임 에러는 에러(Error)와 예외(Exception) 두 가지로 분류하고 있는데 에러는 프로그램 실행 중 메모리가 부족(OutOfMemoryError)하거나 스택 오버플로우(Stack Overflow)와 같이 한 번 발생하면 복구 불가능한 치명적인 오류를 말하며, 예외는 에러 보다 조금은 덜 심각한 오류로 예외가 발생하더라도 그 예외 상황에 대비해 프로그램머가 적절한 예외 대처 코드를 작성하면 프로그램이 비정상적으로 종료되는 것을 막을 수 있고 정상적인 실행 흐름을 유지 시킬 수 있다. 이렇게 예외 상황에 대비해 코드를 작성하는 것을 예외처리(Exception Handling)라고 한다.

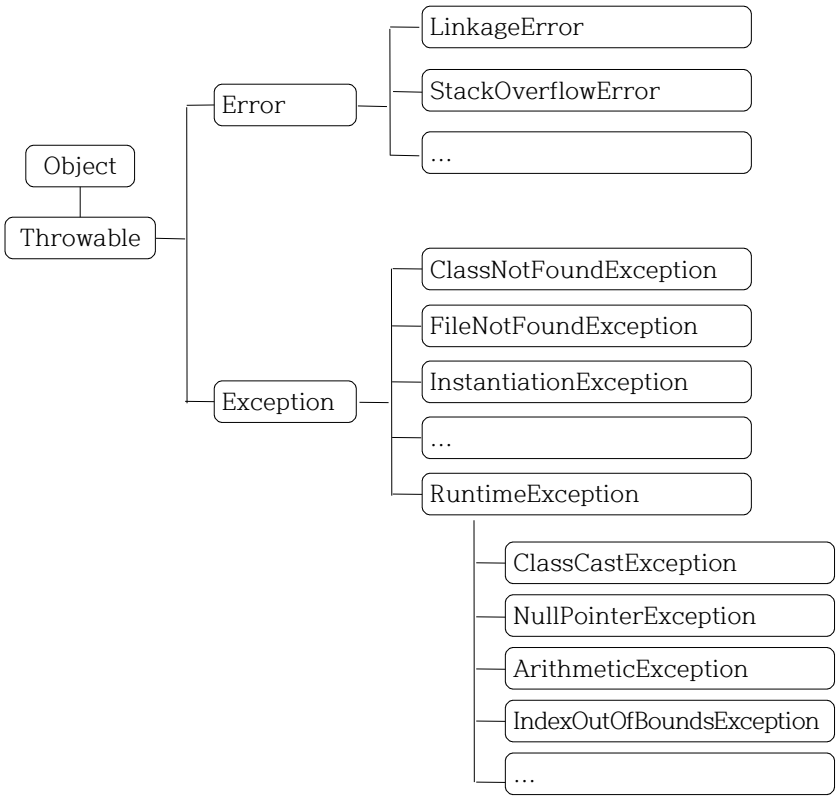

그림 6-1 예외 클래스 계층구조

자바에서는 그림 6-1 예외 클래스 계층구조와 같이 발생할 수 있는 오류(Error와 Exception)를 클래스로 정의하고 있다. 자바의 최상위 클래스는 Object 클래스이므로 Error와 Exception 클래스 또한 Object 클래스의 하위 클래스가 된다.

그림 6-1 예외 클래스 계층 구조와 같이 Exception 클래스는 크게 RuntimeException과 그 외의 Exception 클래스들로 나눌 수 있는데 RuntimeException 계열 클래스는 주로 프로그래머의 실수에 의해서 발생될 수 있는 예외들이다. 예를 들면 어떤 객체도 참조하지 않는 참조 변수를 사용해 클래스 멤버에 접근하려고 했다던가(NullPointerException), 배열의 index 범위를 벗어나 접근했다

던가(IndexOutOfBoundsException) 또는 정수를 0으로 나누는 코드를 작성 했다던가(Arithmetic Exception) 하면 발생되는 예외가 RuntimeException 계열 예외들이다.
그 외의 Exception 클래스들은 주로 외부의 영향에 의해 발생할 수 있는 예외들로 프로그램 사용자에 의해 존재하지 않는 클래스 이름을 사용했다던가(ClassNotFoundException), 존재하지 않는 파일 이름을 사용했다던가(FileNotFoundException) 하면 발생할 수 있는 예외들이다.
RuntimeException 계열은 try-catch 문을 이용해 Exception을 처리하는 경우도 있겠지만 정수를 0으로 나누지 않도록 코드를 작성하거나, index 범위를 벗어나 배열에 접근하지 않도록 코드를 작성하는 것이 올바른 처리방법이다.
RuntimeException 계열에 속하는 예외는 try-catch 문으로 예외처리를 해주지 않아도 컴파일을 할 수 있지만 그 외의 Exception에 속하는 예외는 반드시 try-catch 문으로 예외를 처리해야 컴파일을 할 수 있다. 즉 그 외의 Exception에 속하는 예외는 컴파일시 예외처리가 되었는지 컴파일러가 체크해주는 CheckedException이고 RuntimeExceptin에 속하는 예외는 컴파일시 컴파일러에 의해 체크되지 않는 UncheckedException으로 구분하기도 한다.

▶ 정상적인 프로그램 흐름에서 try-catch 문의 실행 흐름

- com.javastudy.ch06.exception

```
public class TryCathFlow01 {

    public static void main(String[] args) {

        System.out.println("1번");

        try {
            System.out.println("2번");
            System.out.println("3번");

        } catch(Exception e) {
            // try 블럭에서 Exception이 발생하지 않아 4번이 출력되지 않음
            System.out.println("4번");

        } finally { // finally 블록은 필요하지 않으면 생략할 수 있다.
            // 예외가 발생되거나 정상 흐름 모두 항상 finally 블럭이 실행된다.
            System.out.println("5번");
        }
        System.out.println("6번");
    }
}
```

▶ 예외 발생시 try-catch 문의 실행 흐름

- com.javastudy.ch06.exception

```
public class TryCathFlow02 {
```

```
    public static void main(String[] args) {

        System.out.println("1번");

        try {
            System.out.println("2번");

            // 강제로 ArithmeticException 발생
            System.out.println(0 / 0);

            /* Exception이 발생하면 바로 catch 구문으로 이동
             * 아래 3번은 출력되지 않는다.
             **/
            System.out.println("3번");

        } catch(Exception e) {
            // try 블럭에서 Exception이 발생하여 4번이 출력됨
            System.out.println("4번");

        } finally { // finally 블록은 필요하지 않으면 생략할 수 있다.
            // 예외가 발생되거나 정상 흐름 모두 항상 finally 블럭이 실행된다.
            System.out.println("5번");
        }
        System.out.println("6번");
    }
}
```

▶ 배열을 사용할 때 자주 발생하는 ArrayIndexOutOfBoundsException 예외 처리하기

- com.javastudy.ch06.exception

```
public class JavaIndexOutOfBoundException01 {

    public static void main(String[] args) {

        int[] nums = { 1, 3, 5, 6, 9 };

        try {
            /* for문 맨 마지막에 배열의 index 범위를 넘어 접근했기 때문에
             * ArrayIndexOutOfBoundsException이 발생한다.
             * Exception이 발생 했지만 try{} catch{}문을 사용해 예외를 처리했기
             * 때문에 프로그램의 비정상 종료를 막고 다음 코드로 정상적인 실행 흐름을
             * 유지 시킬 수 있다.
             **/
            for(int i = 0; i <= nums.length; i++) {
```

```
                System.out.println(nums[i]);
            }

        } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("잘못된 index 접근 : " + e.getMessage());
        }

        try {
            // 배열 index를 -1로 지정하여 배열의 index 범위를 벗어난다.
            System.out.println(nums[-1]);
            //System.out.println(0 / 0);

        /* 아래와 같이 catch 블럭을 여러 개 지정해 다양한 예외에 대해 처리할 수 있다.
         * 이 때 주의할 것은 예외 타입도 업 캐스팅이 되므로 하위 예외 타입의 catch 블럭을
         * 위쪽에 배치하고 상위 예외 타입을 아래쪽에 배치해야 세부적인 예외처리가 가능하다.
         * 만약 아래에서 Exception 타입의 catch 블럭을 위쪽에 배치한다면
         * ArrayIndexOutOfBoundsException을 비롯한 Exception의 하위 타입의
         * 예외가 발생하더라도 모두 Exception 타입의 catch 블럭에서 예외가 처리되므로
         * 세부적인 예외 처리가 필요하다면 상위 타입의 catch 블럭을 아래쪽에 배치해야 한다.
         **/
        } catch(ArrayIndexOutOfBoundsException aoe) {
            System.out.println("잘못된 index 접근 : " + aoe.getMessage());

        } catch(Exception e) {
            /* ArrayIndexOutOfBoundsException 발생으로
             * 이 catch 블럭은 실행되지 않고 finally 블럭으로 넘어간다.
             **/
            System.out.println(e.getMessage());

        } finally {
            /* finally 블록은 예외가 발생할 때나 발생하지 않을 때 모두 실행된다.
             * finally 블록에는 주로 프로그램에서 사용한 자원(DB 연결 해제 등)을
             * 해제하거나 예외가 발생하더라도 꼭 실행해야 하는 코드를 기술한다.
             **/
            System.out.println("finally는 항상 실행됨");
        }
    }
}
```

▶ 참조 값이 null인 변수를 사용하면 발생하는 NullPointerException 예외 처리하기

- com.javastudy.ch06.exception

```
public class JavaNullPointerException01 {
```

```
    static Member member;

    public static void main(String[] args) {

        Member m1 = new Member("이순신", 35);
        System.out.println("m1 : " + m1);

        try {
            /* member 변수는 클래스 맴버로 클래스 로딩시 null로 초기화 된다.
             * member 변수가 가리키는 인스턴스가 없는 상태에서 인스턴스 메소드를
             * 호출하게 되면 NullPointerException이 발생하게 된다.
             * member 변수의 참조 값이 null 이므로 member 변수는 어떠한 객체도
             * 가리키고 있지 않은 상태에서 메소드를 호출하기 때문에 발생하는 예외이다.
             * 즉 member 변수만 존재하고 그 변수가 참조하고 있는 객체가 없기 때문에
             * member 변수를 사용해 메소드를 호출하는 것은 메모리에 존재하지도 않는
             * 객체를 이용해 메소드를 호출하는 격이 된다.
             **/
            member.setName("을지문덕");

        } catch(NullPointerException e) {
            // Stack을 추적하여 Stack의 내용을 출력하는 메소드
            e.printStackTrace();
            System.out.println("존재하지 않는 객체를 참조 : " + e.getMessage());
        }
        System.out.println(member);
    }
}
```

- com.javastudy.ch06.exception

```
public class Member {

    String name;
    int age;

    public Member(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return name + "(" + age + ")";
    }
```

```
    public void setName(String name) {
        this.name = name;
    }
}
```

▶ 문자열을 숫자로 변경할 때 발생할 수 있는 NumberFormatException 예외 처리하기

- com.javastudy.ch06.exception

```
public class JavaNumberFormatException01 {

    public static void main(String[] args) {

        String num1 = "110";
        String num2 = "110a";

        try {
            /* Integer.parseInt() 메서드는 지정한 문자열을 숫자로 변환해 주는 메서드로
             * 숫자 형식이 아닌 문자열을 지정하면 NumberFormatException이 발생한다.
             **/
            //int i = Integer.parseInt(num1);
            int i = Integer.parseInt(num2);
            System.out.println(i + 10);

        } catch(NumberFormatException e) {
            System.out.println("유효한 숫자가 아닙니다 : " + e.getMessage());
        }
    }
}
```

▶ 다운 캐스팅할 때 발생할 수 있는 ClassCastException 예외 처리하기

- com.javastudy.ch06.exception

```
public class JavaClassCastException01 {

    public static void main(String[] args) {

        Parent c1 = new Child01();
        Parent c2 = new Child02();
        Child01 c3 = null;

        try {
            /* Child01, Child02 클래스는 모두 Parent 클래스를 상속 받은 형제
             * 클래스이지만 서로 형 변환은 불가능 하기 때문에 ClassCastException이
             * 발생하게 된다. 다운 캐스팅시 instanceof 연산자를 사용해 형 변환 가능
```

```
            * 여부를 체크한 후 캐스팅을 시도하는 것이 바람직한 코딩 기법이다.
            * if(c2 instanceof Child01) 문은 false 이므로 예외가 발생하지 않고
            * 다음 코드로 실행 흐름이 이동된다.
            **/
            if(c2 instanceof Child01) {
                    c3 = (Child01) c2;
            }
            System.out.println("c2 instanceof Child01 실행됨");

            /* Child01 타입의 참조 변수를 Child02 타입으로 형 변환을 시도하면
             * ClassCastException이 발생하고 실행 흐름이 catch 블럭으로 이동하여
             * "c3 = (Child01) c2 실행됨"은 콘솔에 출력되지 못하게 된다.
             **/
            c3 = (Child01) c2;
            System.out.println("c3 = (Child01) c2 실행됨");

        } catch(ClassCastException e) {
            System.out.println(e.getMessage());
        }
        System.out.println(c3 == null ? "참조하는 객체가 없는 변수" : c3);
    }
}

class Parent { }
class Child01 extends Parent { }
class Child02 extends Parent { }
```

▶ 컴파일러가 체크해 주는 예외와 체크하지 않는 예외

- com.javastudy.ch06.exception

```
public class JavaCheckedException01 {

    public static void main(String[] args) {

        try {
            /* 모든 예외 클래스는 Exception 클래스를 상속해 구현되었지만 크게
             * RuntimeException과 그 외의 Exception 들로 나눌 수 있다.
             * RuntimeException 클래스 또한 Exception을 상속해 구현되었지만
             * 이 계열의 예외는 컴파일러가 체크해 주지 않는 UncheckedException으로
             * try-catch 문을 사용해 예외를 처리하지 않아도 컴파일은 가능하다.
             * IOException은 Exception을 상속한 클래스로 컴파일러가 체크해
             * 주는 CheckedException에 속하기 때문에 try-catch 문을 사용해
             * 반드시 예외를 처리해야 컴파일이 가능하다.
             * CheckedException은 주로 프로그램 실행시 외부의 영향에 의해 발생할 수
```

```
            * 있는 예외로 FileNotFoundException, ClassNotFoundException,
            * InstantiationException 등이 여기에 속한다.
            * 아래에서 IOException을 생성하여 던졌으므로 IOException이 발생한다.
            **/
            throw new IOException("컴파일시 체크되는 익셉션");

        } catch(IOException e) {
            System.out.println("IOException : " + e.getMessage());

        } catch(Exception e) {
            /* 위쪽의 catch 블록에 정의한 IOException의 catch 블록에
             * 캐치되어 Exception이 처리되고 이 catch 블럭은 실행되지 않는다.
             **/
            System.out.println("Exception : " + e.getMessage());
        }

        /* RuntimeException도 Exception을 상속한 클래스이지만
         * UncheckedException으로 컴파일시 체크되지 않아 try-catch
         * 문을 사용하지 않아도 컴파일은 가능하나 실행 타임에 에러가 발생한다.
         * RuntimeException은 주로 프로그래머의 실수로 인해 발생하는 예외이다.
         * IndexOutOfBoundException, NullPointerException,
         * ClassCastException, ArithmeticException 등이
         * RuntimeException을 상속해 구현된 예외 클래스 들이다.
         * 아래에서 RuntimeException을 생성하여 던졌으므로 예외가 발생한다.
         **/
        throw new RuntimeException("컴파일시 체크되지 않는 익셉션");
    }
}
```

▶ 예외 던지기와 예외를 선언하여 호출자에게 예외처리를 미루기

- com.javastudy.ch06.exception

```
public class JavaExceptionThrows01 {

    public static void main(String[] args) {

        /* 메소드 선언부에 throws IllegalArgumentException을 기술해
         * add(byte, byte) 메소드를 사용하는 호출자에게 예외처리를 미루고
         * 있지만 IllegalArgumentException은 UncheckedException으로
         * try-catch 문을 사용해 예외 처리를 하지 않아도 컴파일은 가능하나
         * 아래 코드가 실행되면 IllegalArgumentException이 발생한다.
         **/
        byte b = add((byte) 100, (byte) 120);
        System.out.println("연산결과 : " + b);
```

```
        /* 메소드 선언부에 throws Exception을 기술해 add(short, short)
         * 메소드를 사용하는 호출자에게 예외처리를 미루고 있기 때문에
         * CheckedException에 대한 예외 처리가 되지 않아 컴파일을 할 수 없다.
         * try-catch 문으로 반드시 예외 처리를 해야 컴파일이 가능하다.
         **/
        //short s = add((short) 32765, (short) 200);

    }

    /* UncheckedException - RuntimeException을 상속한 Exception
     * UncheckedException은 컴파일시 체크되지 않는 Exception으로
     * 프로그램이 실행되는 과정에서 예외가 발생하게 된다.
     * 아래 메소드는 파라미터로 넘어오는 값 a와 b를 더해서 byte형의 범위 보다
     * 크면 if문 안에서 throw를 사용해 IllegalArgumentException을 던지고
     * throws를 이용해 예외를 선언하여 호출자에게 예외처리를 미루고 있다.
     * 이 메소드에서 던지는 예외가 UncheckedException 이므로 throws로 예외를
     * 미루지 않고 try-catch 문을 사용해 예외를 처리하지 않아도 컴파일이 가능하다.
     **/
    static byte add(byte a, byte b) throws IllegalArgumentException {
        if((a + b) > 127) {
            throw new IllegalArgumentException(
                    "숫자 범위를 초과한 수가 입력됨 : " + (a + b));
        }
        return (byte) (a + b);
    }

    /* CheckedException - 그외 Exception 들
     * CheckedException은 컴파일시 체크되기 때문에 Exception 처리를 반드시 해야 한다.
     * 아래 메소드는 파라미터로 넘어오는 값 a와 b를 더해서 short형의 범위 보다
     * 크면 if문 안에서 throw를 사용해 Exception을 던지고 throws를 이용해
     * 예외를 선언하여 호출자에게 예외처리를 미루고 있다.
     * 이 메소드에서 던지는 예외가 CheckedException 이므로 throws를 사용해 예외처리를
     * 미루지 않는다면 try-catch 문을 사용해 반드시 예외를 처리해야 컴파일 할 수 있다.
     **/
    static short add(short a, short b) throws Exception {
        if((a + b) > 32767) {
            throw new Exception("숫자 범위를 초과한 수가 입력됨 : " + (a + b));
        }
        return (short) (a + b);
    }
}
```

▶ 사용자 정의 예외 클래스 만들기

- com.javastudy.ch06.exception

```
/* CheckedException 계열의 사용자 정의 예외 클래스를 정의하려면 Exception
 * 클래스를 상속받고 UncheckedException 계열의 사용자 정의 예외 클래스를
 * 정의하려면  RuntimeException 클래스를 상속받아 사용자 정의 예외 클래스를
 * 작성하면 된다.
 *
 * 아래는 mp3 파일이 존재하지 않으면 발생하는 예외를 Exception 클래스를
 * 상속받아 CheckedException 계열의 사용자 정의 예외 클래스를 정의하고 있다.
 **/
class MP3NotFoundException extends Exception {

    public MP3NotFoundException(String message) {
        super(message);
    }
}

/* 유효하지 않은 mp3 파일일 경우 발생하는 예외를 Exception 클래스를
 * 상속받아 CheckedException 계열의 사용자 정의 예외 클래스를 정의하고 있다.
 **/
class MP3FileInvalidException extends Exception {

    public MP3FileInvalidException(String message) {
        super(message);
    }
}


public class UserDefinitionExceptionTest {

    public static void main(String[] args) {

        /* 프로젝트의 src 폴더에 있는 test.txt 파일과 test.mp3 파일을
         * D:\에 복사한 후 아래의 File() 생성자를 수정해 보면서 테스트 해 보자.
         * mp3 파일은 test.txt 파일을 확장자만 변경해 놓은 것이다.
         **/
        File file = new File("D:\\test.mp3");

        try {
            mp3Open(file);
            play(file);

        } catch(MP3NotFoundException e) {
            System.out.println(e.getMessage());
```

```
        } catch(MP3FileInvalidException e) {
            System.out.println(e.getMessage());

        } finally {
            // File을 읽을 때 사용한 Stream을 닫는 코드를 여기에 기술한다.
        }

        // 자바 7 부터는 여러 예외를 하나의 catch 문에 선언할 수 있다.
        try {
            mp3Open(file);
            play(file);

        } catch(MP3NotFoundException | MP3FileInvalidException e) {
            System.out.println(e.getMessage());

        } catch(Exception e) {
            System.out.println(e.getMessage() + " : " + e.getCause());

        } finally {
            // File을 읽을 때 사용한 Stream을 닫는 코드를 여기에 기술한다.
        }
    }

    /* mp3 파일을 불러오는 메소드
     * throws를 사용해 사용자가 정의한 MP3NotFoundException과
     * MP3FileInvalidException을 선언하여 호출자에게 예외 처리를 미루고 있다.
     **/
    static void mp3Open(File file) throws
        MP3NotFoundException, MP3FileInvalidException {

        /* 참조 변수 file이 null 이거나 File 객체를 생성하면서 생성자에
         * 지정한 파일 또는 디렉터리가 존재하지 않으면 if문이 실행되어
         * MP3NotFoundException이 발생하게 된다.
         **/
        if(file == null || ! file.exists()) {
            throw new MP3NotFoundException("MP3 파일이 존재하지 않습니다.");
        }
        System.out.println(file.getName() + " 을 읽는 중...");
    }

    /* File 객체를 매개 변수로 받아 mp3 파일을 재생 하는 메소드
     * throws를 사용해 사용자가 정의한 MP3FileInvalidException을
     * 선언하여 호출자에게 예외 처리를 미루고 있다.
     **/
```

```
    static void play(File file) throws MP3FileInvalidException {

        /* 확장자 정보를 이용해 mp3 파일인지 체크한다.
         * 확장자가 mp3 파일이 아니면 MP3FileInvalidException이 발생한다.
         **/
        if(! file.getName().endsWith("mp3")) {
            throw new MP3FileInvalidException("선택한 파일은 MP3파일이 아닙니다.");
        }
        System.out.println(file.getName() + " 을 재생하는 중...");
    }
}
```

7. 자바 기본 패키지의 클래스

7.1 Object 클래스

java.lang 패키지의 클래스들은 별도의 import 문을 기술하지 않아도 자바의 모든 클래스에서 사용이 가능한 기본 패키지이다. 이 패키지는 자바 프로그래밍에 기본적으로 필요한 클래스와 인터페이스를 제공하며 자바 프로그래밍에서 가장 많이 사용되는 클래스들로 구성된 패키지이다. 이 패키지에는 자바의 모든 클래스의 최상위 조상인 Object 클래스를 비롯한 문자열을 다루기 위해 제공되는 String, StringBuilder, StringBuffer 클래스, 표준 입출력과 같이 시스템(OS)의 일부 기능을 사용할 수 있도록 지원하는 System 클래스, 자바의 8개 기본타입 데이터를 객체로 다룰 수 있도록 지원하는 8개의 Wrapper 클래스, 클래스와 인터페이스의 메타 데이터(클래스 이름, 생성자 정보, 속성 정보, 메서드 정보 등)를 다룰 수 있도록 지원하는 Class 클래스, 배열 조작(복사, 정렬 등)을 위해 제공되는 Arrays 클래스, 수학 계산을 위해 제공되는 Math 클래스, 예외처리를 위해 제공되는 각종 예외 클래스 등이 있으며 이외에도 자바프로그래밍에서 가장 기본적으로 필요한 기능을 제공하기 위해 다양한 클래스가 정의되어 있다.
우리는 제일 먼저 모든 자바 클래스의 조상인 Object 클래스에 대해 알아 볼 것이다. Object 클래스는 모든 자바 클래스들이 직접 혹은 간접적으로 상속받는 최상위 부모 클래스로 속성(Property)은 정의되어 있지 않지만 자바 프로그래밍에서 꼭 필요한 기능을 제공하기 위한 한 개의 정적(static) 메서드와 11개의 인스턴스 메서드가 정의되어 있다.

▶ Object 클래스의 메서드

- com.javastudy.ch07.objectclass

```
public class ObjectMethods01 {

    public static void main(String[] args) {

        Person p1 = new Person("이순신");

        // 인스턴스의 클래스 이름 출력하기
        System.out.println("p1의 완전한 클래스명 : " + p1.getClass().getName());
        System.out.println("p1의 클래스명 : " + p1.getClass().getSimpleName());

        /* hashCode() 메소드는 인스턴스의 메모리 주소를 사용해
         * 해시코드를 생성하기 때문에 중복되지 않는 고유한 정수를 반환 한다.
         **/
        System.out.println("p1의 해쉬코드 : " + p1.hashCode());

        // 인스턴스의 클래스 명@16진수 해시코드 값 형태로 출력 한다.
        System.out.println("p1을 이용한 정보출력하기 : " + p1.getClass().getName()
                + "@" + Integer.toHexString(p1.hashCode()));

        /* Person 클래스는 toString() 메소드를 오버라이드 하지 않았기 때문에
```

```
        /* Person 클래스는 toString() 메소드를 오버라이드 하지 않았기 때문에
         * 부모 클래스인 Object 클래스로부터 상속받은 toString() 메소드가 호출된다.
         * Object 클래스의 toString() 메소드는 인스턴스에 대한 정보를 문자열로 반환하는
         * 메서드로 "인스턴스의 클래스 명@16진수 해시코드 값" 형식의 문자열을 반환한다.
         **/
        System.out.println("Object toString이 호출됨 : " + p1);
    }
}

class Person {

    String name;

    Person(String name) {
        this.name = name;
    }
}
```

▶ Object equals() 메서드로 같은 인스턴스인지 체크하기

- com.javastudy.ch07.objectclass

```
public class ObjectEquals01 {

    public static void main(String[] args) {

        Person p1 = new Person("홍길동");
        Person p2 = new Person("홍길동");

        /* 참조 변수 p1과 p2가 같은 인스턴스를 가리키고 있으면 true를 반환한다.
         * 다시 말해 참조 변수 p1과 p2에 저장된 인스턴스의 참조 값을 비교하여
         * 같으면(동일한 객체) true를 반환하고 다르면(다른 객체) false를 반환한다.
         * Object 클래스의 equals() 대신 동등연산자(==)를 사용하여 참조 변수에
         * 저장된 참조 값을 직접 비교하여 동일한 객체인지 판단할 수 있다.
         **/
        if(p1.equals(p2)) {
            System.out.println("p1과 p2는 같은 사람입니다.");

        } else {
            System.out.println("p1과 p2는 다른 사람입니다.");
        }

        /* p2의 참조값을 p1에 대입하여 같은 인스턴스를 가리키게 하고 있다.
         * 이전에 p1이 참조하고 있던 인스턴스는 가비지가 된다.
         **/
```

```
        p1 = p2;

        if(p1.equals(p2)) {
            System.out.println("p1과 p2는 같은 사람입니다.");

        } else {
            System.out.println("p1과 p2는 다른 사람입니다.");
        }
    }
}
```

▶ equals() 메서드를 재정의해 같은 정보를 가진 객체를 동일 객체로 체크하기

- com.javastudy.ch07.objectclass

```
public class ObjectEquals02 {

    public static void main(String[] args) {

        /* 일반적으로 회원 아이디와 이름이 같으면 같은 사람이라 할 수 있다.
         * 아래는 동일한 정보를 가지고 있는 Person01 클래스의 인스턴스이지만
         * 실제로 인스턴스가 생성되는 주소가 다르기 때문에 서로 다른 객체가 된다.
         * 실생활에서는 동일한 속성을 가지면 같은 사람이라고 판단해야 하는 경우가 있다.
         * 예를 들면 주민번호와 이름이 같은 사람이면 동일한 사람이라고 보아야 한다.
         * 아래와 같이 동일한 정보를 가지는 두 개의 인스턴스를 생성해서 동등 연산자로
         * 비교하면 서로 다른 인스턴스이므로 false 값을 얻게되는데 이런 경우에는
         * 동일한 클래스로부터 생성된 서로 다른 인스턴스를 비교해서 특정 속성이 같을 경우
         * true를 반환할 수 있도록 Object 클래스의 equals() 메서드를 오버라이딩 하면 된다.
         **/
        Person01 p1 = new Person01("midas", "홍길동");
        Person01 p2 = new Person01("midas", "홍길동");

        /* 참조 변수의 동등연산(==)은 참조 변수에 저장된 참조 값을 비교한다.
         * 참조 변수의 참조 값을 비교하여 같으면 if문을 실행한다.
         **/
        if(p1 == p2) {
            System.out.println(p1 + " : " + p2 + "는 같은 객체입니다.");

        } else {
            System.out.println(p1 + " : " + p2 + "는 다른 객체입니다.");
        }

        /* Persion01 클래스에서 오버라이드된 equals() 메소드를 이용해 서로 다른
         * 인스턴스라 할지라도 id와 name이 동일하면 if문 안의 코드가 실행된다.
         **/
```

```
        if(p1.equals(p2)) {

            /* Object hashCode() 메소드는 인스턴스의 주소 값을 이용해
             * 해시코드를 생성하므로 인스턴스마다 고유한 정수 값을 반환한다.
             * 즉 서로 다른 인스턴스라면 해시코드 값은 다르게 반환된다.
             **/
            System.out.println("p1과 p2의 해시코드값 비교 : "
                               + p1.hashCode() + " : " + p2.hashCode());
            System.out.println(p1 + " : " + p2 + "은 같은 사람입니다.");

        } else {
            System.out.println(p1.hashCode() + " : " + p2.hashCode());
            System.out.println(p1 + " : " + p2 + "은 다른 사람입니다.");
        }
    }
}

// 상속받는 클래스가 없는 경우 컴파일러에 의해 Object 클래스를 상속받는다.
class Person01 {

    String id;
    String name;

    public Person01(String id, String name) {
        this.id = id;
        this.name = name;
    }

    // 아이디와 이름이 같으면 true를 반환하도록 Object 클래스의 equals()를 재정의 한다.
    @Override
    public boolean equals(Object obj) {

        /* 매개변수로 넘어온 인스턴스의 참조 값이 null이 아니고 Person01 타입으로
         * 형 변환이 가능하면 Persion01로 형 변환한 후 현재 인스턴스의 id와 name을
         * 매개변수로 넘어온 인스턴스의 id와 name과 비교하여 같으면 true를 반환한다.
         * Persion01 클래스의 id와 name 필드는 String 객체 이므로 String 클래스에서
         * 오버라이드된 equals()를 이용해 비교하게 된다. String 클래스의 equals()는
         * 두 문자열을 문자 단위로 비교하여 같으면 true를 반환하도록 정의되어 있다.
         **/
        if(obj != null && obj instanceof Person01) {
            Person01 p = (Person01) obj;
            return this.id.equals(p.id) && name.equals(p.name);
        }

        return false;
```

```
    }

    @Override
    public String toString() {
        return name + "(" + id + ")";
    }
}
```

▶ clone() 메서드로 객체 복제하기

- com.javastudy.ch07.objectclass

```
public class ObjectClone01 {

    public static void main(String[] args) {

        Student originStudent = new Student("홍길동", 25);

        /* clone() 메소드는 자기 자신을 복제하여 새로운 인스턴스를 생성하여 반환한다.
         * Object 타입을 반환 하므로 다운 캐스팅이 필요하다.
         **/
        Student copyStudent = (Student) originStudent.clone();

        /* 원본의 name을 변경해도 같은 인스턴스가 아니므로 copy 본의
         * name은 변경되지 않는다. 기본형과 String 타입의 멤버 변수는
         * 제대로 복제가 이루어지지만 참조 타입 멤버 변수는 참조 값이
         * 복사되므로 완전한 복제가 이루어지지 않는다.
         **/
        originStudent.setName("이순신");
        originStudent.setAge(28);

        System.out.println("originStudent : " + originStudent);
        System.out.println("copyStudent : " + copyStudent);

        // originStudent와 copyStudent가 같은 인스턴스인지 비교하고 있다.
        System.out.println("originStudent.equals(copyStudent) : "
                + originStudent.equals(copyStudent));
    }
}

/* clone()를 사용하기 위해서는 Cloneable 인터페이스를 반드시 구현해야 한다.
 * Cloneable을 구현하지 않으면 CloneNotSupportedException이 발생한다.
 * Cloneable 인터페이스는 구현해야 할 추상 메소드가 없는 마커 인터페이스 이다.
 * 마커 인터페이스란 추상 메소드는 갖고 있지 않지만 이 마커 인터페이스를 구현하게
 * 되면 어떤 기능을 제공할 지의 여부를 판단하기 위해 정의된 인터페이스를 말한다.
```

```
 **/
class Student implements Cloneable {

    private String name;
    private int age;

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }

    public String toString() {
        return name + "(" + age + ")";
    }

    /* Object 클래스의 clone() 메소드는 protected 접근 제어자로 선언되었다.
     * 어디에서든 접근할 수 있는 public 접근 제어자로 사용하기 위해 재정의했다.
     **/
    @Override
    public Object clone() {

        Object obj = null;

        try {
            /* 오버라이드 하는 자식의 clone() 메소드 안에서 부모 클래스인
             * Object 클래스의 clone() 메소드를 이용해 자신을 복제한다.
             **/
            obj = super.clone();

        } catch(CloneNotSupportedException e) {
            System.out.println("Cloneable이 구현되지 않은 인스턴스");
            e.printStackTrace();
```

```
        }

        // 복제한 새로운 인스턴스를 반환한다.
        return obj;
    }
}
```

[연습문제 7-1]
Object 클래스의 메서드 활용하기

앞에서 Object 클래스가 가지고 있는 메서드에 대해서 학습하였다. 앞에서 알아본 Object 클래스의 메서드를 이용하는 다음과 같은 프로그램을 작성해 보자.

먼저 “JavaStudyCh07Exercise” 프로젝트를 만들고 “com.javastudy.ch07.objectmethods” 패키지를 만들어 이 패키지에 필요한 클래스를 생성하여 프로그램을 작성하시오.

1. Member 클래스는 회원의 이름, 나이, 성별을 저장하는 인스턴스 멤버를 가지고 있으며 이들을 모두 private으로 선언하고 이 인스턴스 멤버에 접근할 수 있는 getter 메서드와 setter 메서드를 제공 하시오.

2. Member 클래스에는 기본 생성자와 3개의 인스턴스 멤버를 한 번에 초기화 할 수 있는 생성자를 정의하고 인스턴스의 현재 상태를 아래와 같은 문자열로 반환하는 toString() 메서드를 오버라이드 하시오.

    홍길동(25 - 남성)

3. 이름, 나이, 성별이 같으면 동일한 회원으로 인식할 수 있도록 Object 클래스의 equals() 메서드를 오버라이드 하시오.
4. Member 클래스와 같은 패키지에 Member 클래스를 사용하는 MemberUserExam 클래스를 만들고 main() 메서드 안에서 Member 클래스의 인스터스 2개를 아래와 같이 생성하시오.

```
Member m1 = new Member("홍길동", 25, "남성");
Member m2 = new Member("홍길동", 25, "남성");
```

5. 위에서 m1이 참조하고 있는 인스턴스의 정보를 아래와 같이 “인스턴스의 클래스명@16진수 해쉬코드” 형태로 출력하시오.

m1을 이용한 정보출력하기 :
com.javastudy.ch07.objectmethods.Member@15db9742

6. if문을 이용해 m1과 m2가 같은 회원인지 비교하여 다음과 같이 각 인스턴스의 해쉬코드와 같이 출력되도록 하시오.

p1과 p2의 해시코드 비교 : 366712642 - 1829164700
홍길동(25 - 남성) : 홍길동(25 - 남성)은 같은 사람입니다.

7.2 Wrapper 클래스

▶ 오토박싱과 언박싱

- com.javastudy.ch07.wrapperclass

```
public class WrapperClass01 {

    public static void main(String[] args) {

        /* Integer 클래스는 기본형인 int형 데이터를 객체로 다루어야 할 때 사용하는
         * 클래스 이다. 자바에서 기본형 데이터를 객체로 저장하거나 메소드를 호출 할 때
         * 메소드의 인수로 객체 타입을 지정해야 할 경우가 있는데 이때 기본형 데이터를
         * 감싸서 객체로 사용할 수 있도록 지원하는 클래스를 Wrapper 클래스라 한다.
         *
         * 자바는 기본형 8개에 대해 Wrapper 클래스를 제공하고 있다.
         * char형과 int형을 제외한 나머지는 기본 자료형 이름의 첫 글자를 대문자로
         * 정의한 클래스로 예를 들어 byte -> Byte, boolean -> Boolean으로
         * 정의되어 있다. char 형은 Character, int 형은 Integer로 정의되어 있다.
         *
         * 자바 1.5 이상에서는 오토박싱/언박싱을 지원하므로 add(x, y)가 호출되면
         * 자동으로 Integer 타입으로 변환 된 후 Object 타입으로 업캐스팅 된다.
         * 오토박싱(Autoboxing)은 기본형 데이터를 Wrapper 타입으로 자동
         * 변환해 주는 것을 말하며 언박싱(Unboxing)은 그 반대의 개념이다.
         *
         * Wrapper 타입의 변수에 기본형 데이터를 할당하게 되면 오토박싱에
         * 의해서 new 연산자와 대입하는 값을 인수로 하는 생성자가 호출된다.
         * 아래는 Integer i = new Integer(100)으로 변경되어 실행된다.
         **/
        Integer i = 100;

        /* 아래와 같이 Wrapper 타입의 데이터를 기본형 변수에 대입하면
         * 자동으로 언박싱 되어 기본형 데이터로 변경된다.
         * 아래는 int x = i.intValue()와 같이 변경되어 실행된다.
         **/
        int x = i;

        /* Integer 타입 변수 i에 int형 데이터 10을 감싸서
         * Integer 클래스의 인스턴스를 생성하고 그 참조 값을 i에 대입한다.
         **/
        Integer j = new Integer(x);
        System.out.println(i + " : " + j);

        /* Wrapper 클래스 타입의 참조 변수 i와 j를 등가 비교하면
         * i와 j는 서로 다른 인스턴스이므로 아래는 false가 출력된다.
         **/
        System.out.println("i == j : " + (i == j));
```

```
        System.out.println();

        /* 모든 Wrapper 클래스는 그 인스턴스의 내부에서 감싸고 있는
         * 실제 값을 비교하는 equals() 메소드를 오버라이딩 하고 있다.
         * 약간의 예외 사항이 있지만 Wrapper 클래스의 인스턴스가 감싸고
         * 있는 내부의 값이 같은지를 비교할 때도 각 Wrapper 클래스에서
         * 오버라이딩한 equals() 메소드를 사용해 내부의 값을 비교할 수 있다.
         **/
        System.out.println("i.equals(j) : " + i.equals(j));

        // 당연한 얘기겠지만 기본형은 equals() 메소드를 사용할 수 없다.
        //System.out.println(x.equals(y));
    }
}
```

▶ Wrapper 클래스의 메서드 사용하기

- com.javastudy.ch07.wrapperclass

```
public class WrapperClass02 {

    public static void main(String[] args) {

        // 숫자 형태의 문자열을 Wrapper 클래스인 Integer 타입으로 변환
        System.out.println(Integer.valueOf("100"));

        /* 숫자 형태의 문자열을 int 형으로 변환
         * 자바 1.5 이상에서는 오토박싱/언박싱이 적용되므로
         * 반환 값이 기본형 일 때와  Wrapper 타입 일 때의 차이가 없어졌다.
         **/
        System.out.println(Integer.parseInt("200"));

        // Integer 타입의 숫자 5를 int 형으로 변환하여 반환한다.
        System.out.println(new Integer(5).intValue());
        System.out.println();

        // 16진수 FF를 10진수로 변환
        System.out.println("16진수 FF : " + Integer.parseInt("FF", 16));

        /* Wrapper 클래스도 new 연산자를 사용해 각각의 인스턴스를 생성할 수 있다.
         * 모든 Wrapper 클래스는 내부의 값을 String으로 반환해 주는 toString()
         * 메소드가 오버라이딩 되어 각 Wrapper 클래스의 toString() 메소드가 호출된다.
         **/
        System.out.println("new Boolean(String) : " + new Boolean("true"));
        System.out.println("new Double(String) : " + new Double("3.141592"));
```

```
        System.out.println("new Character(char) : " + new Character('c'));
        System.out.println();

        /* 모든 Wrapper 클래스에는 인수로 지정하는 기본형 데이터를 String으로
         * 변환해 반환하는 static 메서드인 toString() 메서드가 정의되어 있다.
         **/
        System.out.println("Character.toString(char) : " + Character.toString('한'));
        System.out.println("Integer.toString(int) : " + Integer.toString(38));
        System.out.println("Double.toString(double) :" + Double.toString(3.141592));

        /* String 클래스에는 int, long, float, double, char, boolean 등의
         * 데이터를 문자열로 변환해 주는 valueOf() 메서드가 오버로드 되어 있다.
         **/
        System.out.println("String.valueOf(double) : " + String.valueOf(3.14));
    }
}
```

7.3 String 클래스

▶ String 리터럴과 new 연산자를 이용해 생성한 String 인스턴스의 차이

- com.javastudy.ch07.stringclass

```
public class StringLiteral01 {

    public static void main(String[] args) {

        /* String Literal은 상수 풀에 저장되며 같은 문자열이 존재하면
         * 새로 생성하지 않고 기존에 존재하는 문자열을 참조하게 된다.
         * 자바에서 문자열은 한번 생성되면 변하지 않는 특성을 가지고 있다.
         **/
        String str1 = "Java Study";
        String str2 = "Java Study";

        // str1과 str2의 문자열을 연결하여 새로운 문자열을 생성한다.
        System.out.println(str1 + str2);

        /* new 연산자를 이용해 String 객체를 생성하면 heap 영역에 각각의
         * 인스턴스가 생성되고 각각의 인스턴스에 문자열이 저장된다.
         **/
        String str3 = new String("Java Study");
        String str4 = new String("Java Study");

        /* 상수 풀에 존재하는 하나의 문자열을 2개의 변수가 참조하므로
         * str1과 str2의 참조 값은 같다. 그러므로 true가 반환된다.
         **/
        System.out.println("str1 == str2 : " + (str1 == str2));

        /* String 클래스에서 오버라이딩한 equals() 메소드로 문자열의 내용을 한다.
         * 그러므로 str1과 str2의 문자열 내용이 같기 때문에 true가 반환된다.
         **/
        System.out.println("str1.equals(str2) : " + str1.equals(str2));

        /* 상수 풀에 존재하는 문자열과 heap 영역의 인스턴스를 비교 - false
         * 참조변수 str1에 저장된 값은 상수 풀에 생성된 String 인스턴스의 참조 값이며
         * str3에 저장된 값은 heap 영역에 생성된 String 인스턴스의 참조 값이므로
         * 두 참조변수의 값을 비교하는 동등연산(==)의 결과는 false가 된다.
         **/
        System.out.println("str1 == str3 : " + (str1 == str3));

        /* 오버라이딩된 String 클래스의 equals() 메소드를 사용해 비교 - true
         * str1은 상수 풀에 생성된 인스턴스의 참조 값을 str3은 heap 영역에 생성된
         * 인스턴스를 참조하므로 동등연산자(==)를 이용하면 참조 값을 비교하게 되어
         * false가 반환되지만 String 클래스에서 오버라이딩한 equals() 메서드를
```

```
         * 이용하면 참조 값이 다른 String 인스턴스의 내용이 동일한지 비교할 수 있다.
         **/
        System.out.println("str1.equals(str3) : " + str1.equals(str3));

        // new 연산자로 생성된 다른 영역에 존재하는 String 인스턴스를 비교 - false
        System.out.println("str3 == str4 : " + (str3 == str4));

        // String 클래스의 equals() 메소드를 사용해 문자열 내용을 비교 - true
        System.out.println("str3.equals(str4) : " + str3.equals(str4));
    }
}
```

▶ valueOf() 메서드로 기본형 데이터를 문자열 데이터로 변환하기

- com.javastudy.ch07.stringclass

```
public class StringValueOf01 {

    public static void main(String[] args) {

        int i = 30;
        boolean b = true;
        char c = 's';

        /* 기본형 데이터를 문자열로 변환하는 메서드
         * String 클래스의 valueOf() 메서드는 static으로 정의되어 있어 객체를 생성할
         * 필요 없이 클래스 이름을 사용해 기본형 데이터를 문자열로 변환할 수 있다.
         * String 클래스에 valueOf() 메서드는 byte와 short 형을 제외한 기본 형 데이터
         * 6개를 문자열로 변환할 수 있도록 9개의 메소드로 오버로딩 되어있다.
         **/
        String iStr = String.valueOf(i);
        String bStr = String.valueOf(b);
        String cStr = String.valueOf(c);

        /* 문자열을 기본형 데이터로 변환하기
         * 자바 기본형 8개에 대응하는 Wrapper 클래스가 존재하며 각 Wrapper
         * 클래스에는 각각 대응되는 기본형으로 변환하는 메소드를 가지고 있다.
         **/
        i = Integer.parseInt(iStr);
        b = Boolean.getBoolean(bStr);
        c = cStr.charAt(0);
    }
}
```

▶ 문자열 안에서 특정 문자열을 찾아 필요한 문자열만 추출하기

- com.javastudy.ch07.stringclass

```
// trim(), length(), indexOf(), lastIndexOf(), substring()
public class StringMethods01 {

    public static void main(String[] args) {

        // 아래와 같이 양쪽에 공백이 있도록 문자열을 만든다.
        String filePath = "          D:\\javaStudy.zip          ";

        // trim() 메서드는 문자열의 양쪽 끝에서 공백 문자열을 제거하는 메소드
        filePath = filePath.trim();

        /* 매개변수로 넘겨준 문자열을 filePath 문자열에서 찾아 index를 반환한다.
         * 매개변수로 지정한 첫 번째 만나는 문자열의 index를 반환하고 검색을 종료 한다.
         * filePath 문자열에 매개변수로 넘겨준 문자열이 존재하지 않으면 -1을 반환한다.
         **/
        int index = filePath.indexOf("\\");

        /* 매개변수로 넘겨준 index의 위치부터 끝까지 문자열을 추출하여 반환한다.
         * 아래는 파일의 경로에서 확장자를 포함한 파일명을 추출한다.
         **/
        String fileName = filePath.substring(index + 1);

        /* 매개변수로 넘겨준 시작 index와 끝 index 바로 이전까지의 문자열 반환한다.
         * 문자열의 처음부터 점(.)이 위치한 바로 이전의 문자열까지 추출하여 반환한다.
         * 아래는 파일명에서 확장자를 제외한 파일 이름만 추출한다.
         **/
        index = fileName.indexOf(".");
        String name = fileName.substring(0, index);

        /* 문자열 중에서 점(.)이 있는 문자열 다음부터 마지막까지 추출한다.
         * 파일명에서 확장자만 추출한다.
         **/
        String extension = fileName.substring(index + 1);

        System.out.println("filePath : " + filePath);
        System.out.println("fileName : " + fileName);
        System.out.println("파일명의 길이 : " + fileName.length() + "자");
        System.out.println("확장자를 제외한 파일 이름 : " + name);
        System.out.println("파일의 확장자 : " + extension);
        System.out.println();

        filePath = "D:\\javastudy\\example\\string.study.txt";
```

```
        /* 매개변수로 넘겨준 문자열을 filePath 문자열의 끝에서 부터 찾아 index를
         * 반환한다. 매개변수로 넘겨준 문자열의 뒤에서 부터 찾아 첫 번째 만나는
         * index를 반환하고 검색을 종료 한다.
         * 아래는 전체 경로에서 확장자를 포함한 파일명을 추출한다.
         **/
        index = filePath.lastIndexOf("\\");
        fileName = filePath.substring(index + 1);

        // 아래는 파일명에서 확장자를 제외한 파일 이름을 추출한다.
        index = fileName.lastIndexOf(".");
        name = fileName.substring(0, index);

        // 아래는 파일명에서 확장자만 추출한다.
        extension = fileName.substring(index + 1);

        System.out.println("filePath : " + filePath);
        System.out.println("filePath의 길이 : " + filePath.length() + "자");
        System.out.println("fileName : " + fileName);
        System.out.println("확장자를 제외한 파일명 : " + name);
        System.out.println("파일의 확장자 : " + extension);
    }
}
```

▶ 문자열 안에서 지정한 문자열이 포함되어 있으면 대문자로 변환하기

- com.javastudy.ch07.stringclass

```
// contains(), toUpperCase(), toLowerCase(), replace(), split()
public class StringMethods02 {

    public static void main(String[] args) {

        String[] files = {
                "javastudy.zip", "EnglishStudy.txt", "java수업.hwp",
                "JSPStudy.txt", "자바 웹프로그래밍", "Java과제.txt" };

        String strFiles = "";
        int count = 0;
        String keyword = "java";
        System.out.println("초기 files 배열의 내용 - " + Arrays.toString(files));

        // 반복문 안에서 keyword를 포함하고 있으면 대문자 그렇지 않으면 소문자로 변환
        for(int i = 0; i < files.length; i++) {
```

```
            // 매개변수로 넘어온 keyword의 문자열을 포함하고 있으면 true를 반환한다.
            if(files[i].contains(keyword)) {
                count++;

                // 문자열을 대문자로 변환하는 메소드
                files[i] = files[i].toUpperCase();

            } else {

                // 문자열을 소문자로 변환하는 메소드
                files[i] = files[i].toLowerCase();
            }
        }

        System.out.println(keyword + "에 대한 검색 결과 " + count + "개 파일을 찾음");
        System.out.println("검색 후 files 배열의 내용 - "
                        + (strFiles = Arrays.toString(files)));

        // 첫 번째 매개변수에 지정한 문자열을 찾아 두 번째 매개변수에 지정한 문자열로 치환
        strFiles = strFiles.replace("[", "");
        strFiles = strFiles.replace("]", "");
        System.out.println("[와 ]를 공백문자열로 치환 후 : " + strFiles);

        // 첫 번째 매개변수에 지정한 문자열을 두 번째 매개변수에 지정한 문자열로 치환
        strFiles = strFiles.replace(',', '/');
        System.out.println(strFiles);

        // 매개변수로 지정한 문자열을 기준으로 문자열을 분할하여 String 배열로 반환한다.
        String[] strs = strFiles.split("/");
        System.out.println("분할한 문자열의 개수 : " + strs.length);

        for(int i = 0; i < strs.length; i++) {
            System.out.println(strs[i]);
        }
    }
}
```

▶ 특정 문자열로 시작하거나 끝나는 문자열 찾기

- com.javastudy.ch07.stringclass

```
// startsWith(), endsWith(), equalsIgnoreCase()
public class StringMethods03 {

    public static void main(String[] args) {
```

```
        String[] files = {
                "javastudy.zip", "EnglishStudy.hwp", "java수업.hwp",
                "JSPStudy.txt", "자바 웹프로그래밍", "Java과제.hwp" };

        int count1 = 0;
        int count2 = 0;
        String keyword = "hwp";
        System.out.println("초기 files 배열의 내용 - " + Arrays.toString(files));

        // 반복문 안에서 지정한 문자열로 시작하거나 끝나는 문자열의 개수를 count
        for(int i = 0; i < files.length; i++) {

            // 매개변수로 넘어온 keyword의 문자열로 끝나면 true를 반환한다.
            if(files[i].endsWith(keyword)) {
                count1++;
            }

            // 매개변수로 넘어온 문자열이 java로 시작하면 true를 반환한다.
            if(files[i].startsWith("java")) {
                count2++;

            } else if(files[i].substring(0, 4).equalsIgnoreCase("java")){
                // 대소문자 구분 없이 java 라는 단어로 시작하면 카운트 증가
                count2++;
            }
        }

        System.out.println("files 배열의 hwp 파일의 수 : " + count1 + "개");
        System.out.println("files 배열의 java로 시작하는 파일의 수 : " + count2 + "개");
    }
}
```

[연습문제 7-2]
String 클래스를 활용해 URL에서 요청 명령을 구분해 출력하기

웹 서버로 들어오는 요청이 아래에서 제시하는 String 배열에 저장된 URL 정보와 같을 때 이 배열의 데이터를 이용해 요청 명령을 분석하여 각각의 명령에 따라서 처리해야할 업무를 문자열로 출력하는 프로그램을 작성해 보자.

[연습문제 7-1]과 같은 프로젝트에 “com.javastudy.ch07.stringclass” 패키지를 만들어 이 패키지에 필요한 클래스를 생성하여 프로그램을 작성하시오.

1. 클래스 이름은 RequestCommandProcess로 지정하고 이 클래스는 프로그램 진입점인 main() 메서드를 가지고 있으며 main() 메서드 안에는 아래와 같이 URL 정보를 저장하고 있는 String 배열이 정의되어 있다.

```
String[] urls = {
        "http://www.localhost:8080/WebApp/joinProcess",
        "http://www.localhost:8080/WebApp/orderProcess",
        "http://www.localhost:8080/WebApp/writeMemo"
};
```

2. 일반적으로 서버로 들어오는 요청에 대해서 URL 전체를 명령으로 채택해 처리하는 경우는 거의 없으며 URL 중에 일부를 추출해 현재 요청이 어떤 분석하여 각각의 요청을 처리하게 된다.
서버로 들어오는 요청 URL을 자세히 살펴보면 항상 동일한 부분과 요청이 올 때마다 매번 서로 다른 부분으로 나뉘는데 요청 명령을 분석할 때는 전체 URL을 명령으로 사용하기에는 너무 길기 때문에 전체 URL에서 항상 동일한 부분을 제외한 나머지를 추출해 요청 명령으로 사용하는 것이 일반적이다.
우리도 일반적인 방법을 사용해 요청 명령을 구분하기 위해서 RequestCommandProcess 클래스에 위의 urls 배열에 저장된 URL 정보 하나를 매개변수로 받아서 요청 명령을 추출해 다음과 같이 반환하는 getCommand() 메서드를 정의해 사용해 보자.

입력 값 : "http://www.localhost:8080/WebApp/joinProcess"
결과 값 : "/joinProcess"

3. main() 메서드 안에서 향상된 for 문을 활용해 배열의 URL을 하나씩 꺼내와 2번에서 정의한 getCommand() 메서드를 이용해 요청 명령을 추출하고 조건문을 활용해 추출한 명령에 대해서 처리해야할 업무를 아래 실행 결과와 같이 문자열로 출력되도록 프로그램을 작성하시오.

[실행결과]
요청 명령 : /joinProcess - 회원 가입 완료
요청 명령 : /orderProcess - 상품 주문 완료
요청 명령 : /writeMemo - 메모 작성 완료

7.4 StringBuilder

▶ StringBuilder의 생성자와 특징

- com.javastudy.ch07.stringbuilder

```
public class StringBuilder01 {

    public static void main(String[] args) {

        /* StringBuilder 인스턴스생성
         * String 데이터는 내부의 문자열이 한 번 생성되면 그 데이터를 변경할 수 없다.
         * 자바에서 String 데이터끼리의 + 연산은 문자열을 연결하여 새로운
         * String 객체를 생성하는 방식으로 문자열을 관리한다.
         * StringBuffer와 StringBuilder는 내부에 버퍼를 사용해 문자열을 저장해 두고
         * 그 버퍼에서 문자열을 추가, 수정, 삭제하는 방식으로 문자열 데이터를 관리한다.
         **/
        StringBuilder sb = new StringBuilder();
        StringBuffer sf = new StringBuffer();

        /* StringBuilder의 append() 메서드는 자바의 기본형 데이터와 문자열
         * 데이터 등을 문자열로 추가할 수 있도록 13개 메서드를 오버로딩 하고 있다.
         **/
        sb.append("String Builder");
        sb.append(true);
        sb.append(123.5f);

        // StringBuilder와 StringBuffer의 메소드는 거의 같다.
        sf.append("String Builder");
        sf.append(true);
        sf.append(123.5f);

        // StringBuilder와 StringBuffer의 실제 데이터의 길이, 내용을 출력한다.
        System.out.println("sb의 길이 : " + sb.length() + ", sb의 내용 : " + sb.toString());
        System.out.println("sf의 길이 : " + sf.length() + ", sb의 내용 : " + sf.toString());

        /* StringBuilder와 StringBuffer은 equals()가 오버라이딩 되어 있지 않고
         * toString()만 오버라이딩 되어 있어 내용을 비교할 때는 toString()을 이용하여
         * 문자열 데이터로 변환 후 String 클래스의 equals()를 이용해 비교하면 된다.
         **/
        System.out.println("sb.equals(sf) : " + sb.equals(sf));
        System.out.println("sb.toString().equals(sf.toString()) : "
                + sb.toString().equals(sf.toString()));
    }
}
```

▶ StringBuilder의 메서드 사용하기

- com.javastudy.ch07.stringbuilder

```
public class StringBuilder02 {

    public static void main(String[] args) {

        StringBuilder sb = new StringBuilder("1234567890");
        StringBuffer sf = new StringBuffer("어려운 Java");

        // StringBuilder에 저장된 문자열을 거꾸로 나열한다.
        sb.reverse();
        System.out.println("sb를 역순으로 출력 : " + sb);

        /* StringBuilder에 저장된 문자열에서 index 범위의 문자를 제거 한다.
         * 두 번째 매개변수의 index 바로 이전까지 제거 한다.
         **/
        sb.delete(0, 1);
        System.out.println("sb의 첫 번째 문자 제거 : " + sb);

        /* StringBuilder에 저장된 문자열에 첫 번째 매개변수로 지정한
         * index 위치에 두 번째 매개변수의 데이터를 추가 한다.
         **/
        sb.insert(2, ".");
        System.out.println(sb);

        /* StringBuffer에 저장된 문자열에서 index 범위의 문자열을 지정한
         * 문자열로 치환 한다. 두 번째 매개변수의 index 바로 이전까지 적용 된다.
         **/
        sf.replace(0, 3, "쉬운");
        System.out.println(sf);

        // 지정한 위치부터 문자열 추출
        System.out.println(sb.substring(sb.indexOf(".") + 1));
        System.out.println(sf.substring(3));
    }
}
```

▶ String과 StringBuilder, StringBuffer의 속도 비교

- com.javastudy.ch07.stringbuilder

```
public class StringBuilder03 {

    public static void main(String[] args) {
```

```
        String str = "";
        StringBuilder sb = new StringBuilder();
        StringBuffer sf = new StringBuffer();

        long start = 0;
        long end = 0;

        // String의 작업 속도 계산
        start = System.currentTimeMillis();
        for(int i = 0; i < 10000; i++) {
            str += "java";
        }
        end = System.currentTimeMillis();
        System.out.println("String : " + (end - start));

        // StringBuilder의 작업 속도 계산
        start = System.currentTimeMillis();
        for(int i = 0; i < 500000; i++) {
            sb.append("java");
        }
        end = System.currentTimeMillis();
        System.out.println("StringBuilder : " + (end - start));

        // StringBuffer의 작업 속도 계산
        start = System.currentTimeMillis();
        for(int i = 0; i < 500000; i++) {
            sf.append("java");
        }
        end = System.currentTimeMillis();
        System.out.println("StringBuffer : " + (end - start));
    }
}
```

7.5 Math 클래스

▶ Math 클래스의 메서드 1

- com.javastudy.ch07.mathclass

```
// abs(), sqrt(), pow(), log(), random()
public class MathClass01 {

    public static void main(String[] args) {

        /* Math 클래스는 수학과 관련된 기능을 제공하는 클래스로 절대값, 난수 발생,
         * 삼각함수, 제곱근, 거듭제곱, 로그, 오일러 상수 등을 구해주는 다양한 메서드를
         * 제공하고 있다. Math 클래스는 외부에서 인스턴스를 생성할 수 없도록 기본
         * 생성자를 private으로 정의하고 있기 때문에 클래스 이름으로 바로 접근할 수
         * 있도록 모든 메서드를 static으로 정의하고 있다.
         **/
        // 절대값 구하기
        System.out.println("-7의 절댓값 : " + Math.abs(-7));

        // 제곱근과 거듭제곱 구하기
        System.out.println("2의 제곱근 : " + Math.sqrt(2));
        System.out.println("2의 제곱 : " + Math.pow(2, 2));

        // 로그 구하기
        System.out.println("log(10) : " + Math.log(10));

        /* 1 ~ 10까지 난수 구하기
         * random() 메서드는 0 ~ 1사이의 난수를 double 형으로
         * 구해주는 메서드로 0은 포함되고 1은 포함되지 않는다.
         **/
        int num = (int) (Math.random() * 10) + 1;
        System.out.println(num);
    }
}
```

▶ Math 클래스의 메서드 2

- com.javastudy.ch07.mathclass

```
// round(), ceil(), floor(), min(), max()
public class MathClass02 {

    public static void main(String[] args) {

        // 소수 점 반올림 하기
        System.out.println("10.24567 소수 첫째자리 반올림: " + Math.round(10.24567));
        System.out.println("10.24567 소수 셋째자리 반올림 : "
```

```
                + Math.round(10.24567 * 1000) / 1000.0f);

        // 지정한 인수보다 크거나 같은 가장 작은 정수 구하기
        System.out.println("10.2 보다 크거나 같은 가장 작은 정수 : " + Math.ceil(10.2));
        System.out.println("-10.2 보다 크거나 같은 가장 작은 정수 : " + Math.ceil(-10.2));

        // 지정한 인수보다 작거나 같은 가장 큰 정수 구하기
        System.out.println("10.2 보다 작거나 같은 가장 큰 정수 : " + Math.floor(10.2));
        System.out.println("-10.2 보다 작거나 같은 가장 큰 정수 : " + Math.floor(-10.2));

        // 인수로 지정한 두 수 중에서 큰 수와 작은 수 구하기
        System.out.println("10과 20중 큰 수 : " + Math.max(10, 20));
        System.out.println("10.7과 10.3중 작은 수 : " + Math.min(10.7, 10.3));
    }
}
```

7.6 유용한 클래스

▶ Date 클래스를 이용해 날짜와 시간 데이터 다루기

- com.javastudy.ch07.usefulclass

```
public class DateClass01 {

    public static void main(String[] args) {

        /* Date 클래스는 날짜와 시간 정보를 GMT(Greenwich Mean Time) 시간을
         * 기준으로 제공하는 클래스 이다. 이 클래스는 1970년 01월 01일 0시를 기준으로
         * 1000 분의 1초 단위로 시간 정보를 관리하는 유용한 클래스 이지만 일부 메서드는
         * 1900년(getYear() 등)을 기준으로하기 때문에 사용하는데 주의가 필요하다. 실제로
         * Date 클래스의 메서드 대부분은 Deprecated(사용을 권장 하지 않음) 되어 있다.
         * 그러므로 다음 예제에서 다룰 Calendar 클래스를 사용하는 것이 좋다.
         *
         * 아래는 시스템의 오늘 날짜를 기준으로 Date 클래스의 인스턴스를 생성한다.
         **/
        Date d = new Date();
        int year = d.getYear();
        int month = d.getMonth();
        int day = d.getDate();
        int week = d.getDay();

        /* Date 클래스의 getYear() 메서드는 1900년 이후의 년도를 반환하기
         * 때문에 정확한 년도를 구하기 위해서는 추가적인 보정 작업이 필요하다.
         **/
        System.out.println("오늘은 " + year + "년 " + (month + 1) + "월 "
                + day + "일(" + week + ")" + d.getHours() + ":" + d.getMinutes()
                + ":" + d.getSeconds() + " 입니다.");

        /* Date 클래스는 이미 많이 사용되어져 있으므로 다른 데이터와 호환될 수 있도록
         * 프로그램을 작성하는 것이 중요하며 Date 또는 시간을 다루는 클래스의 시간 데이터를
         * 또 다른 클래스의 시간 데이터로 변경할 때에는 생성자 또는 메서드의 인수로 밀리 초
         * 형식으로 시간 데이터를 입력 받아 사용하는 것이 일반적인 방식이므로 각각의 시간
         * 데이터에서 밀리 초를 반환해 주는 메서드를 기억하면 편리하다.
         **/
        System.out.println("1970년 1월 1일 0시 부터 오늘까지의 밀리 초 : " + d.getTime());
    }
}
```

▶ Calendar 클래스를 이용해 날짜와 시간 데이터 다루기 1

- com.javastudy.ch07.usefulclass

```
public class CalendarClass01 {
```

```
public static void main(String[] args) {

    /* Calendar 클래스는 날짜와 시간 정보를 GMT(Greenwich Mean Time) 시간을
     * 기준으로 제공하는 클래스 이다. 이 클래스는 1970년 01월 01일 0시를 기준으로
     * 1000 분의 1초 단위로 시간 정보를 처리할 수 있는 기능을 제공하는 클래스 이다.
     * Date 클래스와는 다르게 오로지 1970년 01월 01일 0시를 기준으로 한다.
     *
     * Calendar 클래스는 추상 클래스로 new 연산자를 이용해 객체를 생성할 수 없으며
     * 아래와 같이 getInstance() 메서드를 사용해 Calendar 클래스의 인스턴스를
     * 생성할 수 있다. 이 메서드는 외부에서 new 연산자를 이용해 인스턴스를 생성할 수
     * 없으므로 클래스 이름으로 바로 접근할 수 있도록 static 메서드로 정의되어 있다.
     *
     * 아래는 시스템의 오늘 날짜를 기준으로 Calendar 클래스의 인스턴스를 생성한다.
     **/
    Calendar cal = Calendar.getInstance();

    /* 아래와 같이 get() 메서드를 이용해 각각의 날짜와 시간 데이터를 구할 수 있다.
     * 월 데이터는 0 베이스이므로 실제 월을 표현할 때는 1을 더하면 된다.
     **/
    int year = cal.get(Calendar.YEAR);
    int month = cal.get(Calendar.MONTH) + 1;
    int day = cal.get(Calendar.DAY_OF_MONTH);
    int hour = cal.get(Calendar.HOUR);
    int minute = cal.get(Calendar.MINUTE);
    int second = cal.get(Calendar.SECOND);

    // 아래는 Calendar 인스턴스의 요일 SUNDAY(1) ~ SATURDAY(7)을 반환한다.
    int week = cal.get(Calendar.DAY_OF_WEEK);
    String strWeek = "";
    switch(week) {
    case 1 :
        strWeek = "일요일";
        break;
    case 2 :
        strWeek = "월요일";
        break;
    case 3 :
        strWeek = "화요일";
        break;
    case 4 :
        strWeek = "수요일";
        break;
    case 5 :
        strWeek = "목요일";
```

```
            break;
        case 6 :
            strWeek = "금요일";
            break;
        case 7 :
            strWeek = "토요일";
            break;
        }

        System.out.println("오늘은 " + year + "년 " + month + "월 "
                + day + "일(" + strWeek + ")이며 현재 시간은 " + hour + "시 "
                + minute + "분 " + second + "초 입니다.");
    }
}
```

▶ Calendar 클래스를 이용해 날짜와 시간 데이터 다루기 2

- com.javastudy.ch07.usefulclass

```
public class CalendarClass02 {

    public static void main(String[] args) {

        // 시스템의 오늘 날짜를 기준으로 Calendar 클래스의 인스턴스를 생성한다.
        Calendar cal = Calendar.getInstance();

        /* getTime() 메서드는 Calendar 객체를 Date 객체로 변환해 반환한다.
         **/
        Date d1 = cal.getTime();

        // Calendar 객체의 밀리 초 정보를 이용해 Date 객체를 생성
        Date d2 = new Date(cal.getTimeInMillis());

        /* Date 클래스의 getYear() 메서드는 1900년 이후의 년도를
         * 반환하기 때문에 정확한 년도를 표현하려면 추가적인 보정이 필요하다.
         **/
        System.out.println("올해는 : " + d1.getYear() + "년");
        System.out.println("올해는 : " + (d2.getYear() + 1900) + "년");

        // Date 객체의 밀리 초 정보를 이용해 Calendar 객체의 시간을 설정할 수 있다.
        cal.setTimeInMillis(d1.getTime());

        // Date 객체를 아래와 같이 Calendar 객체의 시간 정보로 설정할 수 있다.
        cal.setTime(d2);
```

```
        /* Calendar 객체에 set() 메서드를 이용해 새로운 시간 데이터를 설정할 수 있다.
         * 다양한 시간 설정을 위해서 set() 메서드는 4개로 오버로딩 되어 있다.
         **/
        cal.set(2021, 11, 23, 14, 35, 55);
        System.out.println(cal.get(Calendar.YEAR) + "년 "
                + (cal.get(Calendar.MONTH) + 1) + "월 "
                + cal.get(Calendar.DAY_OF_MONTH) + "일 "
                + cal.get(Calendar.HOUR_OF_DAY) + "시 "
                + cal.get(Calendar.MINUTE) + "분 "
                + cal.get(Calendar.SECOND) + "초 입니다.");
    }
}
```

▶ 출력 포맷을 지정해 문자열을 생성하고 출력하기

- com.javastudy.ch07.usefulclass

```
public class StringFormatMethod01 {
    public static void main(String[] args) {

        /* String.format() 메서드의 첫 번째 인수로 문자열과 함께 format 형식을
         * 지정하고 format 형식의 순서에 맞춰 데이터를 지정하면 String 객체로 반환된다.
         *
         * %s : 문자열,  %b : 논리값, %c : char, %t : 날짜 및 시간 출력 형식을 지정
         * %d : 10진 정수, %o : 8진 정수, %x : 16진 정수
         * %f : 실수, %e : 지수와 가수부 표기, %g : 반올림이 적용된 십진수 또는 지수
         *
         * 자세한 내용 format() 메서드 API 문서에서 format A format string 부분 참고
         * 이외에도 java.util.Formatter 클래스의 API 문서를 참고
         **/
        String name = "홍길동";
        int age = 33;
        float weight = 83.53f;
        int money = 10000000;

        String memberInfo1 = String.format("%s의 나이 %d세", name, age);
        String memberInfo2 = String.format("몸무게 %.2fKg, 자산 %,d원", weight, money);

        System.out.println(memberInfo1);
        System.out.println(memberInfo2);

        /* printf() 메서드의 첫 번째 인수로 문자열과 함께 format 형식을 지정하고
         * 순서에 맞춰 데이터를 지정하면 지정한 포맷에 맞는 문자열을 출력할 수 있다.
         **/
        System.out.printf("%s의 나이 %d세", name, age);
```

```
        System.out.println();
        System.out.printf("몸무게 %.2fKg, 자산 %,d원", weight, money);
        System.out.println();

        /* 날짜 포맷 형식은 날짜 및 시간 데이터를 다양한 문자열로 표현할 수 있도록 많은
         * 포맷 형식을 가지고 있으며 자세한 내용은 printf() 메서드의 API 문서에서
         * format A format string as described in Format string syntax 부분 참고
         * 이외에도 java.util.Formatter 클래스의 API 문서를 참고
         **/
        Calendar cal = Calendar.getInstance();
        System.out.printf("오늘은 %tY년 %tm월 %te일 입니다.", cal, cal, cal);
        System.out.println();

        // 아래와 같이 인수의 위치를 1$로 지정하여 첫 번째 cal 변수 하나만 참조할 수도 있다.
        System.out.printf("현재 시간은 %1$tH시 %1$tM분 %1$tS초 입니다.", cal);
        System.out.println();
    }
}
```

▶ StringTokenizer 클래스를 활용해 문자열 분리하기

- com.javastudy.ch07.usefulclass

```
public class StringTokenizer01 {

    public static void main(String[] args) {

        String str = "id=midas&name=강감찬&age=27";
        int count = 0;
        StringTokenizer st = new StringTokenizer(str, "&");

        // 생성자의 두 번째 문자열을 기준으로 분리할 수 있는 token의 수를 얻는다.
        System.out.println("token의 수 : " + (count = st.countTokens()));

        for(int i = 0; i < count; i++) {

            /* "&"를 기준으로 분리한 토큰 중 i 번째 위치한 토큰을 가져와 다시
             * String 클래스의 split()를 이용해 "="로 분리하여 배열로 반환 받는다.
             **/
            String temp[] = st.nextToken().split("=");

            if(temp[0].equals("id")) {
                System.out.println("아이디 : " + temp[1]);

            } else if(temp[0].equals("name")) {
```

```
                System.out.println("이 름 : " + temp[1]);

            } else if(temp[0].equals("age")) {
                System.out.println("나 이 : " + temp[1]);
            }
        }
    }
}
```

8. 컬렉션 프레임워크와 제네릭

자바에서는 다수의 데이터를 쉽게 관리할 수 있는 방법을 제공하기 위해 가변배열 기능/자료구조 등을 클래스로 구현하여 JDK 1.2부터 컬렉션으로 제공하고 있는데 이를 컬렉션 프레임워크(Collection Framework)라 한다. 컬렉션 프레임워크란 데이터의 집합을 저장/관리하는 클래스들을 표준화하여 다수의 데이터를 쉽게 처리할 수 있도록 표준화된 프로그래밍 방식을 제공하는 클래스들의 묶음이라 할 수 있다.

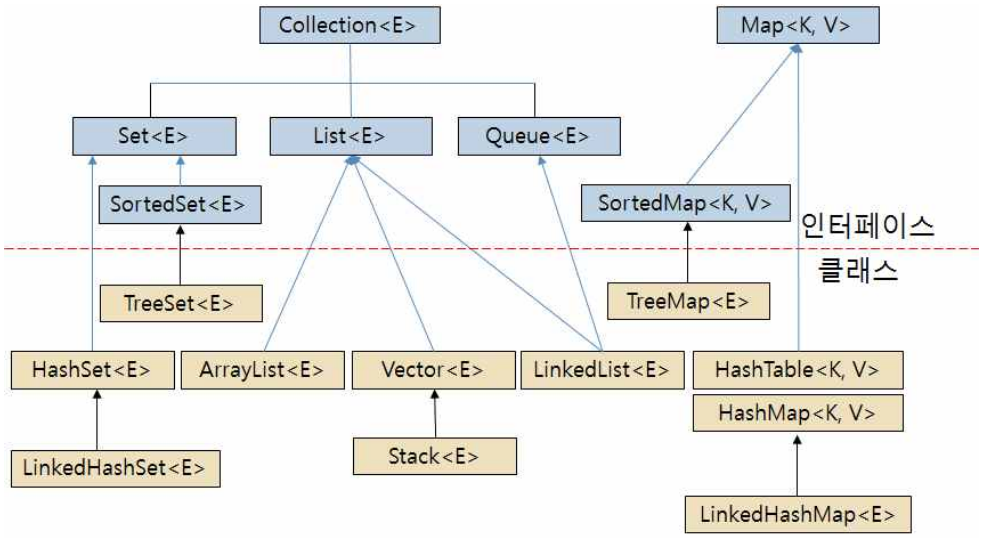

그림 8-1 컬렉션 프레임워크의 상속 계층도

인터페이스	특징 및 구현 클래스
List	중복 데이터를 허용하고 저장 순서를 유지하므로 순서가 있는 데이터의 집합을 표현할 때 사용하며 가변길이 배열/자료구조가 구현되어 있음 구현 클래스 : ArrayList, Vector, LinkedList, Stack 등
Set	중복 데이터를 허용하지 않으며 저장 순서를 유지하지 않으므로 순서가 없는 데이터의 집합을 표현할 때 사용하며 수학의 집합을 표현할 때 주로 사용 구현 클래스 : HashSet, TreeSet
Map	키의 중복을 허용하지 않으며 저장 순서를 유지하지 않는 데이터의 집합 키(key)와 값(Value) 쌍으로 이루어진 데이터를 저장하고 관리하기 위해 사용 구현 클래스 : HashMap, HashTable, TreeMap, Properties

표 8-1 컬렉션 프레임워크의 핵심 인터페이스와 클래스

8.1 List 계열 클래스

▶ ArrayList의 메서드

- com.javastudy.ch08.list

```
public class ArrayListMethods01 {

    public static void main(String[] args) {

        /* 제네릭(Generic)은 Java 5 부터 도입된 기법으로 컬렉션, 스트림, NIO,
         * 람다식 등에서 많이 사용되기 때문에 사용법은 확실히 익혀두는 것이 좋다.
         *
         * 제네릭 기법은 아래와 같이 꺽쇠(<>) 안에 사용할 데이터 타입을 지정하며
         * 이렇게 제네릭으로 타입을 지정하면 하나의 타입만 저장될 수 있도록 고정된다.
         * 제네릭을 사용하면 컴파일 시점에 데이터 타입을 체크할 수 있어 잘못된 타입이
         * 사용되는 것을 미연에 방지할 수 있기 때문에 타입 안전성을 높일 수 있다.
         *
         * ArrayList<Member> mList = new ArrayList<Member>();
         **/

        /* List 인터페이스를 구현한 ArrayList는 데이터가 저장되는 순서를 유지하고
         * 중복된 데이터를 저장할 수 있는 가변배열 기능을 제공하는 클래스 이다.
         *
         * ArrayList와 Vector는 가변 배열 기능을 제공하기 위한 클래스로 대부분 동일한
         * 이름을 가진 메서드를 가지고 있다. Vector는 쓰레드의 동기화 문제를 내부적으로
         * 구현하고 있고 ArrayList는 그렇지 않기 때문에 Vector 보다 효율성(처리속도)이
         * 좋아 일반적(쓰레드가 아닌 부분에서)으로 ArrayList를 더 많이 사용한다.
         *
         * ArrayList 내부에 저장할 수 있는 데이터를 제네릭 기법을 이용해 String으로
         * 지정하고 ArrayList 객체를 생성 한다. 아래와 같이 제네릭 기법을 이용해 타입을
         * 지정하면 지정한 타입만 ArrayList에 저장할 수 있다. 제네릭을 사용하지 않으면
         * ArrayList에는 Object까지 저장할 수 있어 자바의 모든 타입을 저장할 수 있다.
         *
         * add() 메서드를 이용해 데이터를 추가하면 index 0부터 순차적으로 저장된다.
         **/
        ArrayList<String> strList = new ArrayList<String>();
        strList.add("ArrayList");
        strList.add("HashSet");
        strList.add("HashMap");

        System.out.println("strList의 크기 : " + strList.size());
        System.out.println("strList의 2번째 데이터 : " + strList.get(1));
        System.out.println(strList);
        System.out.println();
```

```
        // strList의 2번째(index 1) 위치에 있는 데이터 삭제하기
        strList.remove(1);
        System.out.println("2번째 요소 삭제 후 strList의 크기 : " + strList.size());
        System.out.println("strList의 내용 출력 : " + strList);
        System.out.println();

        /* strList 순차접근
         * for 문에서 ArrayList의 size() 메서드를 이용해 ArrayList 객체의 길이를 알 수
         * 있고 get() 메서드를 이용해 index 번째 위치한 데이터를 읽어 올 수 있다.
         **/
        for(int i = 0; i < strList.size(); i++) {
            System.out.println(i + " : " + strList.get(i));
        }
        System.out.println();

        /* clear() 메서드는 ArrayList의 모든 요소를 삭제하고 isEmpty() 메서드는
         * ArrayList가 비어 있으면 true를 비어 있지 않으면 false를 반환한다.
         **/
        strList.clear();
        System.out.println("clear 이후 : strList 크기 : " + strList.size());
        System.out.println("clear 이후 strList이 비어 있는지 여부 : " + strList.isEmpty());
        System.out.println();
    }
}
```

▶ 여러 명의 회원 정보를 ArrayList에 저장하고 출력하기

- com.javastudy.ch08.list

```
// 회원 한 명의 정보를 저장하는 클래스
public class Member {

    private String id;
    private String name;
    private int age;

    public Member(String id, String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    public String getId() {
        return id;
    }
```

```
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
}
```

- com.javastudy.ch08.list

```
public class ArrayListMethods02 {

    public static void main(String[] args) {

        // 3명의 회원 정보를 저장하기 위해 3개의 Member 클래스의 인스턴스를 생성
        Member m1 = new Member("midas1", "이순신", 25);
        Member m2 = new Member("komans", "홍길동", 39);
        Member m3 = new Member("eclipse", "어머나", 33);

        // 여러 명의 회원을 저장하기 위해 제네릭 타입을 Member로 지정해 ArrayList 생성
        ArrayList<Member> memberList = new ArrayList<Member>();
        memberList.add(m1);
        memberList.add(m2);
        memberList.add(m3);

        System.out.println("\t\t### 회원 리스트 - for ###");
        System.out.println("\t아이디\t\t\t이름\t\t\t나이");
        System.out.println("\t=========================");

        // ArrayList의 크기만큼 반복하여 ArrayList에 저장된 회원 정보를 출력 한다.
        for(int i = 0; i < memberList.size(); i++) {
            System.out.println("\t" + memberList.get(i).getId() + "\t\t"
                    + memberList.get(i).getName() + "\t\t\t"
                    + memberList.get(i).getAge());
        }
        System.out.println();

        System.out.println("\t\t### 회원 리스트 - 향상된 for ###");
        System.out.println("\t아이디\t\t\t이름\t\t\t나이");
        System.out.println("\t=========================");

        /* 아래와 같이 향상된 for 문을 이용하면 한 번 반복할 때 마다 memberList의
         * 다음 index에 저장된 Member 클래스의 인스턴스를 m으로 참조할 수 있다.
         **/
```

```
        for(Member m : memberList) {
            System.out.println("\t" + m.getId() + "\t\t"
                    + m.getName() + "\t\t\t" + m.getAge());
        }
        System.out.println();
        System.out.println("\t 입력된 회원수 : " + memberList.size() + "명");
    }
}
```

▶ Collections 클래스를 이용한 컬렉션 정렬하기

- com.javastudy.ch08.list

```
public class CollectionsSort {

    public static void main(String[] args) {

        ArrayList<String> list = new ArrayList<String>();
        list.add("자바스크립트 완벽 가이드");
        list.add("HTML5&CSS3");
        list.add("안드로이드 프로그래밍 정복");
        list.add("프로스프링 3");
        list.add("스프링 인 액션");

        /* 향상된 for 문은 배열이나 열거형(Enumeration) 또는 컬렉션 프레임 워크의
         * List 계열 객체에 순차적으로 접근할 때 유용하게 사용할 수 있는 반복문이다.
         * for문 안에서 사용할 변수를 선언하고 배열과 열거형 또는 컬렉션을 지정하면
         * 반복문이 반복될 때 마다 변수를 이용해 현재 위치의 데이터에 접근할 수 있다.
         **/
        for(String str:list) {
            System.out.println(str);
        }

        /* Collections 클래스의 reverse() 메소드를 이용해
         * list의 요소를 역순으로 재구성 한다.
         **/
        Collections.reverse(list);
        System.out.println();
        System.out.println("reverse : " + list);
        System.out.println();

        /* Collections 클래스의 sort()를 이용해 list의 요소를 오름차순 정렬 한다.
         * List에 저장되는 데이터 타입(제네릭으로 지정된 클래스)이 Compareable을
         * 구현하지 않았으면 컴파일은 잘 되지만 실행시 에러가 발생한다.
         * String은 Compareable 인터페이스를 구현한 클래스이다.
```

```
         **/
        Collections.sort(list);
        System.out.println("sort : " + list);
        System.out.println();

        /* Collections 클래스의 max(), min()을 이용해
         * List 계열 요소의 최대 값과 최소 값을 얻을 수 있다.
         **/
        System.out.println("ArrayList의 내용 중 최댓값 : " + Collections.max(list));
        System.out.println("ArrayList의 내용 중 최솟값 : " + Collections.min(list));
    }
}
```

8.2 Set 계열 클래스

▶ 중복 데이터를 허용하지 않고 저장 순서를 유지하지 않는 HashSet

```
public class HashSetClass01 {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<String>();
        list.add("자바 프로그래밍");
        list.add("오라클 데이터 베이스");
        list.add("자바 프로그래밍");

        System.out.println("ArrayList : " + list);
        System.out.println();

        /* List 계열 데이터를 이용해 Set 객체의 데이터를 초기화할 수 있다.
         * HashSet은 중복을 허용하지 않고 저장 순서를 유지하지 않기 때문에
         * ArrayList에서는 중복해 저장되던 "자바 프로그래밍"은 저장되지 않는다.
         **/
        HashSet<String> set = new HashSet<String>(list);

        // 아래에서도 "자바 프로그래밍"은 저장되지 않는다.
        set.add("자바 프로그래밍");
        set.add("스프링 MVC 프로그래밍");
        System.out.println("HashSet : " + set);
        System.out.println();

        /* Set 데이터는 저장 순서를 유지하지 않기 때문에 ArrayList에서와 같이
         * index를 통해서 순차적으로 데이터에 접근할 수 있는 get() 메서드와 같은
         * 메서드를 지원하지 않는다. 그래서 처음부터 끝까지 순차적으로 접근하려면
         * 향상된 for 문을 사용하거나 Iterator 객체를 이용해야 한다.
         **/
        System.out.println("### 향상된 for - Set 데이터 출력 ###");
        for(String obj : set) {
            System.out.println(obj);
        }
        System.out.println();

        System.out.println("### Iterator - Set 데이터 출력 ###");
        Iterator<String> iter = set.iterator();

        /* Iterator에 다음 데이터가 존재하면 while을 반복하고 그렇지 않으면
         * while문을 빠져 나간다. 처음 Iterator 객체를 얻으면 커서는 첫 번째
         * 데이터 바로 이전을 가리키고 있다. Iterator 클래스의 hasNex()
         * 메서드는 다음에 읽을 데이터가 있으면 true 없으면 false를 반환한다.
         * */
        while(iter.hasNext()) {
```

```
            /* Iterator 클래스의 next() 메서드는 다음 데이터를 읽어와 반환한다.
             * next() 메서드가 한 번 호출될 때 마다 뒤로 이동하기 때문에 데이터가 없는
             * 상태에서 next() 메서드를 호출하면 NoSuchElementException이 발생한다.
             **/
            System.out.println(iter.next());
        }
    }
}
```

▶ 중복 데이터를 허용하지 않는 HashSet을 이용해 로또번호 생성하기

- com.javastudy.ch08.set

```
public class LottoNumCreator {

    public static void main(String[] args) {

        // 난수를 생성하기 위해 Random 클래스의 인스턴스를 생성 한다.
        Random rnd = new Random();

        // 중복 데이터를 저장하지 않는 HashSet 인스턴스 생성
        HashSet<Integer> set = new HashSet<Integer>();

        for(int i = 0; i < 100; i++) {

            // Random 클래스의 nextInt()를 이용해 난수를 생성 한다.
            int num = rnd.nextInt(45) + 1;

            /* HashSet에 로또번호를 추가
             * HashSet은 중복 데이터를 허용하지 않기 때문에
             * 기존에 중복된 데이터가 있으면 저장되지 않는다.
             **/
            set.add(num);

            // set에 데이터가 6일 될 때 for문을 빠져 나간다.
            if(set.size() == 6) {
                break;
            }
        }

        System.out.println("정렬전 금주의 로또 : " + set);

        /* HashSet의 데이터를 정렬하기 위해 List 계열 객체를 이용
         * List 계열의 객체를 생성할 때 List, Set 데이터로 초기화 할 수 있다.
         **/
```

```
        ArrayList<Integer> list = new ArrayList<Integer>(set);

        // Collections 클래스의 sort()를 이용해 로또번호를 오름차순 정렬 한다.
        Collections.sort(list);
        System.out.println("정렬된 금주의 로또 : " + list);
    }
}
```

▶ 데이터 추가시 오름차순 정렬되는 TreeSet을 이용한 로또번호 생성기

- com.javastudy.ch08.set

```
public class TreeSetLottoNumCreator {

    public static void main(String[] args) {

        /* TreeSet은 이진검색트리 알고리즘을 구현한 Set 계열의 클래스로
         * 노드의 추가, 삭제에 시간이 많이 걸리나 검색과 정렬이 매우 빠른 자료구조이다.
         * TreeSet 또한 Set 계열의 클래스이므로 데이터의 중복은 허용하지 않는다.
         **/
        TreeSet<Integer> set = new TreeSet<Integer>();
        for(int i = 0; i < 50; i++) {

            int num = (int) (Math.random() * 45) + 1;
            set.add(num);

            // 로또번호가 6개가 될 때 for문을 빠져 나간다.
            if(set.size() == 6) {
                break;
            }
        }

        /* TreeSet은 저장하면서 내부적으로 정렬이 되므로 따로 정렬할 필요가 없다.
         * Integer 클래스가 Comparable을 구현했기 때문에 TreeSet에 저장이 가능하다.
         * Comparable을 구현하지 않은 클래스를 사용하게 되면 실행시 에러가 발생 한다.
         **/
        System.out.println("TreeSet 로또번호 : " + set);
    }
}
```

8.3 Map 계열 클래스

▶ 데이터를 key와 value 한 쌍으로 관리하는 HashMap 사용하기

- com.javastudy.ch08.map

```
public class HashMapMethods01 {

    public static void main(String[] args) {

        /* Map 계열의 클래스는 데이터를 key와 value 한 쌍으로 관리하며 저장 순서를
         * 유지하지 않는다. key의 값은 중복을 허용하지 않으며 value는 중복이 가능하다.
         * 중복된 key가 저장되면 기존의 같은 key의 value 값을 덮어 쓴다.
         *
         * put() 메서드를 이용해 key와 value를 지정해 Map에 데이터를 추가한다.
         * */
        HashMap<String, String> map1 = new HashMap<String, String>();
        map1.put("010-1234-5678", "홍길동");
        map1.put("010-2345-2356", "이순신");
        map1.put("010-1234-5678", "이순신");

        // get() 메서드에 키(key)를 지정해 데이터를 바로 읽어올 수 있다.
        System.out.println("010-2345-2356 : "
                        + map1.get("010-2345-2356") + "\n");

        // put() 메서드에 key와 value를 지정해 데이터를 바로 변경할 수 있다.
        map1.put("010-2345-2356", "강감찬");
        System.out.println("put() 메서드로 변경 : "
                        + map1.get("010-2345-2356") + "\n");;

        // Map 객체 안에 지정한 키(key) 데이터가 존재하는지 체크
        System.out.println("010-1234-5678 key 존재 여부 : "
                        + map1.containsKey("010-1234-5678"));

        // Map 객체 안에 지정한 값(value) 데이터가 존재하는지 체크
        System.out.println("홍길동 value 존재 여부 : "
                        + map1.containsValue("홍길동") + "\n");

        /* key 리스트를 Set 데이터로 구해서 향상된 for문으로 데이터 출력하기
         *
         * keySet() 메서드는 Map 데이터의 key 리스트를 Set 데이터로 반환하는 메서드
         * keySet() 메서드를 이용해 Map 객체에 저장된 key 리스트를 읽어와 향상된 for 문을
         * 사용해 Set 데이터에 접근하고 있다. 아래에서 keySet에 들어있는 데이터가 key의
         * 리스트이므로 반복문 안에서 key를 이용해 value 데이터를 읽어올 수 있다.
         **/
        Set<String> keySet = map1.keySet();
```

```
        // Set 데이터도 집합 데이터이므로 아래와 같이 향상된 for문으로 접근할 수 있다.
        System.out.println("keySet()과 향상된 for문을 이용한 데이터 출력");
        for(String key: keySet) {
            System.out.println(key + " : " + map1.get(key));
        }
        System.out.println();

        /* Iterator 객체를 이용해 데이터 출력하기
         *
         * Iterator를 구하고 while문의 조건식에서 hasNext()를 이용해 다음 key가
         * 존재하는지 체크하여 key가 존재하면 while문 안에서 next()를 이용해 key를
         * 읽어 Map 클래스의 get()의 인자로 key를 지정하면 value 값을 읽어 올 수 있다.
         **/
        Iterator<String> keyIter = keySet.iterator();

        System.out.println("keySet() Iterator를 이용한 데이터 출력");
        while(keyIter.hasNext()) {
            /* next()로 Iterator에서 다음에 위치한 key 값을 읽어와
             * 그 key 값을 이용해 Map에 저장된 value 값을 읽어 온다.
             **/
            String key = keyIter.next();
            System.out.println(map1.get(key) + " : " + key);
        }
    }
}
```

▶ HashMap의 중복데이터 처리

- com.javastudy.ch08.map

```
public class HashMapOverlapData01 {

    public static void main(String[] args) {

        // key를 String으로 value를 Member로 저장하는 HashMap 객체 생성
        HashMap<String, Member> map = new HashMap<String, Member>();
        map.put("midas1", new Member("midas1", "이순신", 25));
        map.put("komans", new Member("komans", "홍길동", 39));
        map.put("eclipse", new Member("eclipse", "어머나", 33));

        /* key 리스트를 Set 데이터로 구해서 향상된 for문으로 데이터 출력하기
         *
         * keySet() 메서드는 Map 데이터의 key 리스트를 Set 데이터로 반환하는 메서드
         * keySet() 메서드를 이용해 Map 객체에 저장된 key 리스트를 읽어와 향상된 for 문을
         * 사용해 Set 데이터에 접근하고 있다. 아래에서 keySet에 들어있는 데이터가 key의
```

```
         * 리스트이므로 반복문 안에서 key를 이용해 value 데이터를 읽어올 수 있다.
         **/
        Set<String> keySet = map.keySet();

        System.out.println("keySet()과 향상된 for문");
        for(String key : keySet) {
            System.out.printf("%s - %s\n", key, map.get(key).getName());
        }
        System.out.println();

        /* HashMap은 key로 데이터를 구분하므로 key의 중복은 허용되지 않고
         * value의 중복은 체크하지 않기 때문에 value는 중복되어 저장될 수 있으며
         * 데이터의 저장 순서를 유지하지 않는 특징을 가지고 있다. 만약 중복된 key의
         * 데이터가 저장되면 기존의 key에 해당하는 value 값을 덮어 쓴다.
         *
         * key가 midas1인 value는 새로운 Member 클래스의 인스턴스로 변경된다.
         **/
        map.put("midas1", new Member("kingjjang", "왕호감", 22));

        /* Iterator 객체를 이용해 데이터 출력하기
         *
         * Iterator를 구하고 while문의 조건식에서 hasNext()를 이용해 다음 key가
         * 존재하는지 체크하여 key가 존재하면 while문 안에서 next()를 이용해 key를
         * 읽어 Map 클래스의 get()의 인자로 key를 지정하면 value 값을 읽어 올 수 있다.
         **/
        Iterator<String> keyIter = keySet.iterator();

        System.out.println("keySet()의 Iterator");
        while(keyIter.hasNext()) {
            /* next()로 Iterator에서 다음에 위치한 key 값을 읽어와
             * 그 key 값을 이용해 Map에 저장된 value 값을 읽어 온다.
             **/
            String key = keyIter.next();
            System.out.println(key + " : " + map.get(key).getName());
        }
    }
}
```

▶ Properties 클래스를 이용해 어플리케이션의 환경정보 관리하기

- com.javastudy.ch08.map

```
public class PropertiesMethods01 {

    public static void main(String[] args) {
```

```
        /* Hashtable을 상속한 Properties 클래스는 key와 value를
         * 모두 String 타입으로 관리할 수 있도록 구현된 클래스 이다.
         * Properties는 주로 응용프로그램의 환경정보를 관리하는데 사용한다.
         * Properties 클래스도 Map 계열의 클래스로 key의 중복을 허용하지
         * 않고 데이터의 저장 순서를 유지하지 않는 특징을 가지고 있다.
         * 중복된 key의 데이터가 입력되면 기존의 value 데이터를 덮어 쓴다.
         **/
        Properties prop = new Properties();
        prop.setProperty("dirver", "oracle.jdbc.driver.OracleDriver");
        prop.setProperty("url", "jdbc:oracle:thin:@localhost:1521:xe");
        prop.setProperty("username", "HR");
        prop.setProperty("password", "12345678");

        System.out.println("데이터베이스 연결정보");

        // Properties에 저장된 key 값을 Enumeration(열거형) 타입으로 반환한다.
        Enumeration<?> e = prop.propertyNames();

        /* Enumeration(열거형)도 Iterator와 같이 처음 Enumeration 타입으로
         * 데이터를 받았을 때 커서가 첫 번째 데이터 바로 이전을 가리키고 있어 while문의
         * 조건식에서 hasMoreElements()를 사용해 열거형에서 다음 위치에 데이터가
         * 존재하는지 체크하고 데이터가 존재 한다면 while문 안에서 nextElement()를
         * 사용해 다음에 존재하는 데이터를 읽어 올 수 있다.
         **/
        while(e.hasMoreElements()) {
            /* nextElement()로 열거형에서 다음에 위치한 key 값을 읽어와
             * 그 key 값을 이용해 Properties에 저장된 value 값을 읽어 온다.
             **/
            String key = (String) e.nextElement();
            System.out.println(key + " : " + prop.getProperty(key));
        }
    }
}
```

[연습문제 8-1]
Student 클래스의 인스턴스를 ArrayList와 HashMap에 저장하고 출력하기

아래와 같은 Student 클래스를 정의하고 이 클래스의 인스턴스를 ArrayList와 HashMap에 저장하고 출력하는 다음과 같은 프로그램을 작성해 보자.

먼저 “JavaStudyCh08Exercise” 프로젝트를 만들고 “com.javastudy.ch08.collection”

패키지를 만들어 이 패키지에 필요한 클래스를 생성하여 프로그램을 작성하시오.

1. Student 클래스는 학생의 학번, 이름, 학년, 성별을 저장하는 인스턴스 멤버를 가지고 있으며 이들을 모두 private으로 선언하고 이 인스턴스 멤버에 접근할 수 있는 getter 메서드와 setter 메서드를 가지고 있다.

2. Student 클래스는 기본 생성자와 3가지 속성을 한 번에 초기화 할 수 있는 생성자를 가지고 있다.

3. 객체의 현재 상태를 문자열로 반환하는 toString() 메서드를 오버라이딩 하고 있다.

4. StudentTest 클래스를 별도의 소스 파일로 만들고 이 클래스에 프로그램 진입점인 main() 메서드를 정의하여 Student 클래스의 인스턴스 5개를 생성해 ArrayList에 저장하고 반복문을 활용해 다음과 같이 콘솔에 출력하시오.

```
===== ArrayList =====
홍길동(202001003 - 1학년 남성)
어머나(201709103 - 3학년 여성)
왕빛나(201903001 - 2학년 여성)
김유신(201905023 - 2학년 남성)
전해영(202007275 - 1학년 여성)
```

5. ArrayList에 저장된 Student 데이터를 반복문을 이용해 HashMap 객체에 저장한 후 다시 반복문을 활용해 HashMap에 저장된 Student 객체의 정보를 다음과 같이 출력하시오

```
===== HashMap =====
어머나(201709103 - 3학년 여성)
홍길동(202001003 - 1학년 남성)
전해영(202007275 - 1학년 여성)
김유신(201905023 - 2학년 남성)
왕빛나(201903001 - 2학년 여성)
```

6. 소스코드를 보고서에 추가하고 출력 결과를 갈무리해 보고서 같이 추가하시오. 또한 보고서에 사용한 모든 이미지는 가로 700px 이하의 jpg 포맷으로 압축해 보고서와 같이 제출하시오.

7. 프로젝트 소스는 export 해서 zip 파일로 제출하시오.

9. 자바 I/O와 스트림

컴퓨터 외부로부터 데이터를 입력 받거나 외부로 데이터를 출력하는 것을 입출력이라 한다. 입출력을 얘기할 때 주로 I/O란 말을 사용하는데 이 I/O란 용어는 Input과 Output의 약자이다.
말 그대로 Input은 컴퓨터의 내부(메모리)로 데이터를 읽어 오는 것이고 Output은 컴퓨터 내부(메모리)에서 외부(하드 디스크, 파일 등등)로 데이터를 보내는 것을 말한다. 이때 빼놓을 수 없는 개념이 바로 스트림(Stream)인데 이 스트림은 개울, 줄기 등의 의미를 가진 말로 물이 흐르는 길 또는 통로를 의미하는 것이다. 자바 프로그래밍에서도 스트림은 이와 비슷한 의미를 가지고 있는데 자바에서 스트림은 물이 흐르는 통로가 아니라 바로 데이터가 이동하는 통로를 의미한다.
스트림은 크게 입력스트림(Input Stream)과 출력스트림(Output Stream)으로 나눌 수 있다.
입력스트림은 키보드와 같은 입력장치로부터 입력된 데이터를 컴퓨터 내부로 읽어 들이는 역할을 하는 스트림이며 출력스트림은 컴퓨터 내부의 데이터를 모니터와 같은 출력 장치로 내보내는 역할을 하는 스트림이다. 여기서 입력장치라고 하면 단순히 키보드나 마우스를 떠올릴 수 있는데 기본적인 입력장치를 포함한 화면 입력, 파일, DB, 네트워크를 통해 입력되는 것을 의미하며 출력 장치는 모니터와 같은 일반적인 출력장치를 포함한 파일, DB, 네트워크로 출력하는 것을 의미 한다.
스트림은 한 쪽에서 다른 쪽으로 차례대로 데이터를 흘려보내는 역할을 한다. 그렇기 때문에 입력스트림은 입력, 출력스트림은 출력에 대한 한 쪽 방향으로만 데이터를 전달할 수 있다.
자바 프로그래밍을 예를 들자면 입력스트림은 입력장치에 입력된 데이터를 자바프로그램으로 읽어오는 역할을 하고 출력스트림은 자바프로그램에서 출력장치로 내보내는 역할을 한다.
위에서 스트림은 데이터가 이동하는 통로라 했다. 즉 외부로부터 데이터를 읽어 오려면 외부의 입력장치와 자바 프로그램 간에 데이터를 보내고 받는 통로가 있어야 데이터를 읽어 올 수 있을 것이고 다시 자바프로그램에서 외부로 데이터를 출력하려면 외부로 연결된 통로가 있어야 외부로 데이터를 전달할 수 있을 것이다.
자바에서 스트림을 통해 흐르는 데이터의 종류에 따라 입력스트림과 출력스트림은 각각 바이트스트림과 문자 스트림으로 나누어진다.
스트림은 기본적으로 바이트(Byte) 단위로 데이터를 흘려보낸다. 이 기본 단위인 1Byte씩 차례대로 데이터를 흐르게 할 수 있는 통로를 바이트 스트림이라 부른다. 또한 자바는 유니코드 문자를 지원하는 프로그래밍 언어로 유니코드 한 문자를 다루기 위해 2Byte char 형을 지원하고 있다. 자바에서 문자 데이터를 보다 쉽게 입출력 할 수 있도록 1Byte가 아닌 문자 단위로 데이터가 흐를 수 있는 통로를 지원하는데 이를 문자 스트림이라 한다.
문자 스트림은 우리가 일상적으로 사용하는 문자 데이터를 기반으로 한 문자씩 차례대로 전달할 수 있는 스트림이고 바이트 스트림은 동영상, 이미지, 음악파일 등의 이진데이터(0과 1, 바이너리 데이터라고도 함)로 이루어진 데이터를 1Byte씩 차례대로 전달할 수 있는 스트림이다.

9.1 바이트 스트림

바이트 스트림은 앞에서도 언급했듯이 이진데이터(바이너리 데이터라고도 함)를 1Byte 단위로 전달할 수 있는 통로이다. 바이트 스트림은 바이너리 데이터를 있는 그대로 전달하기 때문에 압축, 동영상, 음악 파일은 물론 문자로 이루어진 텍스트 파일도 전달할 수 있는 스트림이다.
바이트 스트림에는 데이터가 입출력되는 통로를 자체적으로 만들어 실제 입출력을 수행하는 바이트기반 스트림과 이 바이트기반 스트림의 입출력 효율을 높이기 위해 제공되는 바이트기반의 보조스트림이 있다. 보조스트림은 기반스트림을 보조하는 역할을 하는 스트림으로 데이터가 입출력되는 통로를 자체적으로 만들 수 없어 항상 기반스트림과 같이 사용해야 한다.

바이트기반 스트림의 슈퍼 클래스는 추상클래스인 InputStream과 OutputStream으로 이 클래스들은 바이트기반 입출력을 위한 공통기능을 정의한 클래스이다. 그리고 모든 바이트기반 보조스트림의 슈퍼 클래스는 InputStream을 상속한 FilterInputStream과 OutputStream을 상속한 FilterOutputStream 이다.

▶ FileInputStream을 이용해 파일에서 데이터 읽어오기

- 테스트용 파일 : src/fileInput_en.txt

http://www.naver.com
http://www.google.com

- com.javastudy.ch09.bytestream

```
public class FileInputStreamTest {

    public static void main(String[] args) {

        String filePath = "src\\fileInput_en.txt";

        /* FileInputStream 클래스는 파일로부터 바이너리 데이터를 읽어오기 위한
         * 스트림 클래스로 추상 클래스인 InputStream을 상속하여 구현된 클래스이다.
         **/
        FileInputStream fis = null;

        try {
            fis = new FileInputStream(filePath);
            int i = 0;

            /* 파일에서 1Byte씩 읽어 들인다.
             * 더 이상 읽어올 데이터가 없으면 -1을 반환 한다.
             **/
            while((i = fis.read()) != -1) {

                // 1Byte씩 읽어 출력하기 때문에 한글은 깨진다.
                System.out.print((char) i);
            }
        } catch(FileNotFoundException e) {
            e.printStackTrace();

        } catch(IOException e) {
            e.printStackTrace();
        } finally {
            // 사용이 완료된 스트림을 닫는다.
            try {
                if(fis != null) fis.close();
```

```
            } catch(IOException e) { }
        }
    }
}
```

▶ FileOutputStream을 이용해 파일로 데이터 쓰기

- com.javastudy.ch09.bytestream

```
public class FileOutputStreamTest {

    public static void main(String[] args) {

        String filePath = "src\\fileOutput.txt";
        byte[] b = {'s', 't', 'r', 'e', 'a', 'm'};
        int[] nums = {65, 66, 67, 68, 69, 70};

        /* FileOutputStream 클래스는 파일에 바이너리 데이터를 출력하기 위한
         * 스트림 클래스로 추상 클래스인 OutputStream을 상속하여 구현된 클래스 이다.
         **/
        FileOutputStream fos = null;

        try {
            fos = new FileOutputStream(filePath);

            // 파일에 b 배열의 내용을 바이너리 데이터로 출력한다.
            fos.write(b);
            fos.write('\n');
            for(int i = 0; i < nums.length; i++) {

                // 파일에 nums 배열의 정수를 바이너리 데이터로 출력한다.
                fos.write(nums[i]);
            }
        } catch(IOException e) {
            e.printStackTrace();
        } finally {
            // 사용이 완료된 스트림은 닫는다.
            try {
                if(fos != null) fos.close();
            } catch(IOException e) {}
        }
    }
}
```

▶ FileInputStream과 FileOuputStream을 이용한 파일 복사하기

- 테스트용 파일 : src/fileInput.txt

파일에서 바이트 단위로 읽어 들이는 FileInputStream

FileInputStream은 byte 기반 스트림 이다.

- com.javastudy.ch09.bytestream

```
public class FileInputOutputStream {

    public static void main(String[] args) {

        String path = "src/fileInput.txt";
        String destinationPath = "src/fileOutput.txt";
        File file = new File(path);
        //String realPath = file.getAbsolutePath();

        /* FileInputStream 클래스와 FileOutputStream 클래스는 파일 입출력을
         * 위한 클래스로 추상 클래스인 InputStream과 OutputStream을 상속하여
         * 구현된 클래스이다. FileInputStream 클래스는 파일로부터 바이너리 데이터를
         * 읽어오는 클래스고 FileOutputStream 클래스는 파일로 바이너리 데이터를
         * 출력하는 클래스 이다.
         **/
        FileInputStream fis = null;
        FileOutputStream os = null;

        try {
            fis = new FileInputStream(path);

            // 생성자 안에서 destinationPath에 지정한 파일을 생성한다.
            os = new FileOutputStream(destinationPath);

            // 아래는 기존의 파일이 있으면 기존의 내용 뒤에 새로운 내용을 추가한다.
            //os = new FileOutputStream(destinationPath, true);

            int input = 0;

            /* 파일에서 1Byte씩 읽어 들인다.
             * 더 이상 읽어올 데이터가 없으면 -1을 반환 한다.
             **/
            while((input = fis.read()) != -1) {

                /* 읽어온 1Byte를 파일에 쓴다. 1Byte씩 연속적으로
                 * 쓰기 때문에 한글과 같은 유니코드도 깨지지 않는다.
                 **/
                os.write(input);
```

```
                /* 자바에서 한글 등의 유니코드 문자는 2Byte 이므로
                 * 1Byte 씩 콘솔에 출력하면 깨져서 출력된다.
                 **/
                System.out.print(input + ", ");
            }

            // 마지막에 줄 바꿈하여 쓰기를 종료한다.
            os.write('\n');

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {

            try {
                // OutputStream을 먼저 닫고 InputStream을 닫는다.
                os.close();
                fis.close();
            } catch (IOException e) { e.printStackTrace(); }
        }
    }
}
```

▶ BufferedInputStream과 BufferedOutputStream으로 입출력 효율 높이기

- com.javastudy.ch09.bytestream

```
public class BufferedStreamTest {

    public static void main(String[] args) {

        String sourcePath = "src\\JAVA_PRINT.pdf";
        String destinationPath = "src\\java_print_copy.pdf";

        /* FileInputStream과 FileOutputStream은 자체적으로 입출력 통로를 만들어
         * 실제 입출력을 수행하는 바이트기반 스트림이다.
         * BufferedInputStream과 BufferedOutputStream은 기반스트림의 입출력
         * 효율을 높이기 위해 사용하는 스트림으로 자체적으로 입출력 통로를 만들 수 없기
         * 때문에 기반스트림과 같이 사용해야 한다.
         **/
        FileInputStream fis = null;
        BufferedInputStream bis = null;
        FileOutputStream fos = null;
```

```
BufferedOutputStream bos = null;

try {
    // 기반스트림을 먼저 생성하고 보조스트림의 생성자를 통해 인수로 넣어준다.
    fis = new FileInputStream(sourcePath);

    /* BufferedInputStream은 입력을 위한 보조스트림으로 내부적으로
     * 버퍼를 가지고 있는 객체이다. 이 클래스는 두 개의 생성자가 오버로딩 되어
     * 있으며 파라미터 1개인 생성자는 버퍼의 크기가 8192 byte로 설정된다.
     * 이 객체는 입력소스로 부터 데이터를 읽기 위해 최초로 read() 메서드를
     * 호출하면 입력소스로 부터 버퍼 크기만큼의 데이터를 읽어와 버퍼에
     * 저장하고 버퍼에 저장된 데이터를 읽기오기 때문에 매번 입력소스로 부터
     * 직접 읽는 것 보다 처리 속도가 훨씬 빠르다.
     **/
    bis = new BufferedInputStream(fis);
    //bis = new BufferedInputStream(fis, 8192);
    fos = new FileOutputStream(destinationPath);

    /* BufferedOutputStream은 출력을 위한 보조스트림으로 내부적으로
     * 버퍼를 가지고 있는 객체이다. 이 클래스는 두 개의 생성자가 오버로딩 되어
     * 있으며 파라미터 1개인 생성자는 버퍼의 크기가 8192 byte로 설정된다.
     * 이 객체는 외부로 데이터를 출력할 때 버퍼에 먼저 저장하고 버퍼가 다 차면
     * 버퍼에 있는 모든 데이터를 한 번에 출력소스로 흘려보낸다. 버퍼와 먼저
     * 작업하고 버퍼가 차면 출력소스로 출력하기 때문에 매번 출력소스로 출력
     * 하는 것 보다 처리 속도가 훨씬 빠르다.
     **/
    bos = new BufferedOutputStream(fos);
    //bos = new BufferedOutputStream(fos, 8192);
    int input = 0;

    /* 소스 파일에서 1Byte씩 읽어 들인다.
     * 더 이상 읽어올 데이터가 없으면 -1을 반환 한다.
     **/
    while((input = bis.read()) != -1) {

        // 소스 파일에서 읽은 1Byte를 복사 파일에 출력한다.
        bos.write(input);
    }
    System.out.println(sourcePath + "를 " + destinationPath + "로 복사함");
} catch(IOException e) {
    e.printStackTrace();
} finally {
    try {
        /* 출력 스트림을 먼저 닫고 입력 스트림을 닫는다.
         * 보조스트림을 먼저 닫고 기반스트림을 닫는다.
```

```
                * 보조스트림만 닫으면 보조스트림의 close() 안에서
                * 기반스트림의 close()가 호출된다.
                *
                * BufferedOutputStream은 버퍼에 데이터가 가득찼을 때 목적지에
                * 데이터를 흘려보내기 때문에 버퍼가 가득차지 않은 상태에서
                * 프로그램이 종료되면 데이터가 목적지에 완전히 출력되지 못 할
                * 수도 있다. 그래서 마지막에 반드시 flush() 메서드나 close()
                * 메소드를 호출해 버퍼에 남아 있는 데이터가 목적지에 완전히
                * 출력되도록 해야 한다. flush() 메서드는 버퍼의 데이터를
                * 목적지로 흘려보내는 메서드로 close() 메소드 안에서 flush()
                * 메서드가 호출 되도록 구현되어 있다.
                **/
                if(bos != null) bos.close();
                //if(fos != null) fos.close();
                if(bis != null) bis.close();
                //if(fis != null) fis.close();
            } catch(IOException e) { }
        }
    }
}
```

9.2 문자 스트림

문자 스트림은 앞에서도 언급했듯이 문자 데이터를 기반으로 한 문자씩 차례대로 전달할 수 있는 통로이다. 자바는 2Byte 유니코드를 지원하는 프로그래밍 언어로 한 문자를 다루는 데이터 형인 char 형이 2Byte를 지원하기 때문에 바이트기반 스트림으로 문자를 다루는데 어려움이 있어 이를 보완하기 위해 문자기반 스트림을 지원한다.
바이트 스트림과 마찬가지로 자체적으로 데이터가 입출력되는 통로를 만들어 실제 입출력을 수행하는 문자기반 스트림과 문자기반 스트림의 입출력 효율을 높이기 위해 제공되는 문자기반의 보조 스트림이 있다. 보조스트림은 기반스트림을 보조하는 역할을 하는 스트림으로 자체적으로 데이터를 입출력하는 통로를 만들 수 없기 때문에 항상 기반스트림과 같이 사용해야 한다.
문자기반 스트림의 슈퍼 클래스는 추상클래스인 Reader와 Writer로 이 클래스들은 문자기반 입출력을 위한 공통기능을 정의한 클래스이다. 또한 Reader와 Writer를 포함해 이 클래스의 자식 클래스들은 여러 종류의 문자 인코딩(Encoding)을 자바에서 사용하는 유니코드(UTF-16) 문자로 자동변환 기능을 제공한다. 그리고 모든 문자기반 보조스트림의 슈퍼 클래스 또한 Reader와 Writer 이다.

▶ FileReader와 FileWriter로 텍스트 파일 복사하기

- com.javastudy.ch09.charstream

```
public class FileReaderWriter {

    public static void main(String[] args) {

        String path = "src/fileReader.txt";
        String destinationPath = "src/fileWriter.txt";
        File file = new File(path);

        /* FileReader 클래스는 파일에서 문자 단위로 데이터를 읽어오기 위한 스트림
         * 클래스로 추상 클래스인 Reader를 상속하여 구현된 클래스 이다.
         * FileWriter 클래스는 문자 단위로 파일에 테이터를 출력하기 위한 스트림
         * 클래스로 추상 클래스인 Writer를 상속하여 구현된 클래스 이다.
         **/
        FileReader reader = null;
        FileWriter writer = null;
        int input = 0;

        try {

            reader = new FileReader(path);

            /* 생성자의 두 번째 파라미터에 true를 지정하면 파일에 내용이 존재할 경우
             * 기존 내용은 그대로 두고 그 다음에 데이터를 추가하기 시작한다.
             **/
            //writer = new FileWriter(destinationPath);
            writer = new FileWriter(destinationPath, true);
```

```
            /* 파일에서 한 문자씩 읽어 들인다.
             * 더 이상 읽어올 데이터가 없으면 -1을 반환 한다.
             **/
            while((input = reader.read()) != -1) {

                // 한 문자씩 읽어오기 때문에 한글도 깨지지 않는다.
                System.out.print((char) input);

                // 읽어온 데이터를 파일에 출력한다.
                writer.write(input);
            }
            writer.write('\n');

        } catch (IOException e) {
            e.printStackTrace();

        } finally {

            try {
                // 작업이 완료되면 출력 스트림부터 닫고 입력스트림을 닫는다.
                writer.close();
                reader.close();
            } catch(IOException e) { }
        }
    }
}
```

▶ BufferedReader와 BufferedWriter로 입출력 효율 높이기

- com.javastudy.ch09.charstream

```
public class BufferedReaderWriter {

    public static void main(String[] args) {

        String path = "src/fileReader.txt";
        String destinationPath = "src/fileWriter.txt";
        File file = new File(path);

        /* FileReader와 FileWriter는 자체적으로 입출력 통로를 만들어 실제 입출력을
         * 수행하는 문자기반 스트림이다. BufferedReader와 BufferedWriter는
         * 기반스트림의 입출력 효율을 높이기 위해 사용하는 스트림으로 자체적으로 입출력
         * 통로를 만들 수 없기 때문에 기반스트림과 같이 사용해야 한다.
         **/
```

```
FileReader reader = null;
BufferedReader br = null;
FileWriter writer = null;
BufferedWriter bw = null;

try {
    // 기반스트림을 먼저 생성하고 보조스트림의  생성자를 통해 인수로 넣어준다.
    reader = new FileReader(path);

    /* BufferedReader는 입력을 위한 문자기반의 보조스트림으로 내부적으로
     * 버퍼를 가지고 있는 객체이다. 이 클래스는 두 개의 생성자가 오버로딩 되어
     * 있으며 파라미터 1개인 생성자는 버퍼의 크기가 8192 byte로 설정된다.
     * 이 객체는 입력소스로부터 데이터를 읽기 위해 최초로 read() 메서드나
     * readLine() 메서드를 호출하면 입력소스로 부터 버퍼 크기 만큼의 데이터를
     * 읽어와 버퍼에 저장하고 버퍼에 저장된 데이터를 읽기오기 때문에 매번
     * 입력소스로 부터 직접 읽는 것 보다 처리 속도가 훨씬 빠르다.
     **/
    br = new BufferedReader(reader);

    /* 생성자의 두 번째 파라미터에 true를 지정하면 파일에 내용이 존재할 경우
     * 기존 내용은 그대로 두고 그 다음에 데이터를 추가하기 시작한다.
     **/
    //writer = new FileWriter(destinationPath);
    writer = new FileWriter(destinationPath, true);

    /* BufferedWriter는 출력을 위한 문자기반의 보조스트림으로 내부적으로
     * 버퍼를 가지고 있는 객체이다. 이 클래스는 두 개의 생성자가 오버로딩 되어
     * 있으며 파라미터 1개인 생성자는 버퍼의 크기가 8192 byte로 설정된다.
     * 이 객체는 외부로 데이터를 출력할 때 버퍼에 먼저 저장하고 버퍼가 다 차면
     * 버퍼에 있는 모든 데이터를 한 번에 출력소스로 흘려보낸다. 버퍼와 먼저
     * 작업하고 버퍼가 차면 출력소스로 출력하기 때문에 매번 출력소스로 출력
     * 하는 것 보다 처리 속도가 훨씬 빠르다.
     **/
    bw = new BufferedWriter(writer);
    //bw = new BufferedWriter(writer, 8192);

    String input = "";

    /* BufferedReader 클래스의 readLine() 메서드를 이용하면 파일에서
     * 라인 단위로 데이터를 읽어 올 수 있어 편리하다.
     * 이 메서드는 읽을 데이터가 존재하지 않으면 null을 반환한다.
     **/
    while((input = br.readLine()) != null) {
        // 한 라인을 읽어 그대로 파일에 출력한다.
        bw.write(input);
```

```
                // newLine() 메서드는 줄바꿈 해주는 메서드 이다.
                bw.newLine();

                // 문자 단위로 읽어오기 때문에 한글도 깨지지 않는다.
                System.out.println(input);
            }

        } catch (IOException e) {
            e.printStackTrace();

        } finally {

            try {
                /* 출력 스트림을 먼저 닫고 입력 스트림을 닫는다.
                 * 보조스트림을 먼저 닫고 기반스트림을 닫는다.
                 * 보조스트림만 닫으면 보조스트림의 close() 안에서
                 * 기반스트림의 close()가 호출된다.
                 *
                 * BufferedWriter는 버퍼에 데이터가 가득찼을 때 목적지에
                 * 데이터를 흘려보내기 때문에 버퍼가 가득차지 않은 상태에서
                 * 프로그램이 종료되면 데이터가 목적지에 완전히 출력되지 못 할
                 * 수도 있다. 그래서 마지막에 반드시 flush() 메서드나 close()
                 * 메소드를 호출해 버퍼에 남아 있는 데이터가 목적지에 완전히
                 * 출력되도록 해야 한다. flush() 메서드는 버퍼의 데이터를
                 * 목적지로 흘려보내는 메서드로 close() 메소드 안에서 flush()
                 * 메서드가 호출 되도록 구현되어 있다.
                 **/
                bw.close();
                //writer.close();
                br.close();
                //reader.close();
            } catch(IOException e) { }
        }
    }
}
```

▶ InputStreamReader와 OutputStreamWriter

- 테스트용 파일 : src/inputStreamReader.txt

InputStreamReader와 OutputStreamWriter는 바이트기반 스트림을
문자기반 스트림으로 변환해 주는 역할을 하는 스트림 이다.

- com.javastudy.ch09.charstream

```java
public class InputStreamReaderWriter {

    public static void main(String[] args) {

        String path = "src/inputStreamReader.txt";
        String destinationPath = "src/outputStreamWriter.txt";

        /* InputStreamReader와 OutputStreamWriter는 바이트기반 스트림을
         * 문자기반 스트림으로 변환해 주는 스트림이다. 입력소스가 바이트기반 스트림인
         * 경우 문자 단위로 작업하기 위해 이 클래스를 사용하면 아주 편리하다.
         **/
        InputStreamReader isr = null;
        OutputStreamWriter osw = null;
        BufferedReader br = null;
        BufferedWriter bw = null;

        try {

            // 생성자로 바이트기반 스트림을 생성해 전달한다.
            isr = new InputStreamReader(new FileInputStream(path));
            br = new BufferedReader(isr);
            osw = new OutputStreamWriter(new FileOutputStream(destinationPath));
            bw = new BufferedWriter(osw);
            System.out.println("인코딩 : " + isr.getEncoding());

            String input = ""     ;
            while((input = br.readLine()) != null) {
                // 한 라인을 읽어 그대로 파일에 출력한다.
                bw.write(input);

                // newLine() 메서드는 줄 바꿈 해주는 메서드 이다.
                bw.newLine();
                System.out.println(input);
            }
            System.out.println(path + "를 " + destinationPath + "로 복사함.");

        } catch(IOException e) {
            e.printStackTrace();

        } finally {

            try {
                /* 출력스트림을 먼저 닫고 입력스트림을 닫는다.
                 * BufferedReader와 BufferedWriter를 닫으면 close() 메서드 안에서
                 * InputStreamReader, OutputStreamWriter, 기반스트림을 닫는다.
```

```
                **/
                if(bw != null) bw.close();
                if(br != null) br.close();
            } catch(IOException e) {}
        }
    }
}
```

9.3 File 클래스

파일은 아마도 컴퓨터에서 가장 많이 사용되는 입출력 소스 일 것이다.
자바는 File 클래스를 통해 파일과 디렉터리를 다룰 수 있도록 다양한 메서드를 지원하고 있다.
예제를 통해 간단히 File 클래스가 지원하는 메서드에 대해 알아보자.

▶ File 클래스의 메서드

- com.javastudy.ch09.file

```
public class FileClass {

    public static void main(String[] args) {

        String path = "src/test";
        File file1 = new File(path);

        /* Java는 플랫폼(OS) 독립적인 프로그래밍 언어로 OS마다 파일의 경로나
         * 파일의 이름 또는 디렉터리를 구분하는 구분자가 다를 수 있기 때문에 File
         * 클래스의 static 멤버 변수를 이용해 프로그램이 실행되는 현재 플랫폼(OS)에서
         * 사용하는 경로 구분자나 이름 구분자를 적용하여 파일의 경로를 지정하게 되면
         * Java의 플랫폼 독립적인 특징을 최대한 활용할 수 있고 또한 프로그램 실행 중에
         * 플랫폼 마다 다른 구분자로 인해 발생할 수 있는 에러를 미연에 방지할 수 있다.
         **/
        /* src/test/subTest 디렉터리를 다루기 위한 File 객체를 생성한다.
         * 이 객체는 디렉터리를 다루기 위한 객체로 디렉터리가 생성되는 것은 아니다.
         * mkdir() 메서드가 호출되어야 비로소 디렉터리가 생성된다.
         **/
        File file2 = new File(path + File.separator + "subTest");

        /* mkdir() 메서드는 디렉터리를 생성해 주는 메서드 이다.
         * 기준이 되는 상위의 경로가 존재하지 않아도 Exception은 발생하지 않고
         * 단지 디렉터리가 생성되지 않을 뿐 이다.
         **/
        file1.mkdir();
        file2.mkdir();

        /* src/test/subTest/test.txt 파일을 다루기 위한 File 객체를 생성한다.
         * 이 객체는 파일을 다루기 위한 객체로 파일이 생성되는 것은 아니다.
         * 아래에서 createNewFile() 메서드를 호출해 파일을 생성하거나
         * FileWriter 객체를 생성하면 그 때 비로소 파일이 생성된다.
         **/
        File file3 = new File(file2.getPath() + File.separator + "test.txt");
        FileWriter writer = null;

        if(file1.isDirectory()) {
```

```
            System.out.println(file1.getName() + "는 디렉터리 입니다.");
        } else if(file1.isFile()){
            System.out.println(file1.getName() + "는 파일입니다.");
        }
        System.out.println("file2 path : " + file2.getPath());
        System.out.println("file3 path : " + file3.getPath());
        System.out.println("file3 절대경로 : " + file3.getAbsolutePath());

        try {
            /* createNewFile() 메서드는 파일을 생성해 주는 메서드 이다.
             * 이 메서드는 지정한 경로가 존재하지 않으면 IOException이 발생한다.
             * 그렇기 때문에 src/test 디렉터리가 반드시 존재해야 한다.
             * 아니면 exists() 메서드를 호출해 존재 여부를 판단하고 작업해야 한다.
             **/
            file3.createNewFile();

            /* FileWriter 클래스의 생성자를 호출하면 지정한 경로에 파일을 생성해 준다.
             * 지정한 경로가 존재하지 않으면 FileNotFoundException이 발생한다.
             * 그렇기 때문에 src/test/subTest 디렉터리가 반드시 존재해야 한다.
             **/
            writer = new FileWriter(file3);
            String message = "FileWriter는 문자 기반 스트림이다.";
            writer.write(message);
            writer.flush();

        } catch (IOException e) {
            e.printStackTrace();
        }

        System.out.println("file3의 크기 : " + file3.length() + "Byte");
        System.out.println("file3가 있는 드라이브의 크기 : "
                + file2.getTotalSpace() / 1024 / 1024 / 1024 + "GB");
    }
}
```

▶ 파일 복사하기

- com.javastudy.ch09.file

```
public class FileCopy {

    public static void main(String[] args) {

        String inPath = "src/copy/";
        String outPath = "src/destination/";
```

```
File inFile = new File(inPath + "zipcode.txt");
File outFile = new File(outPath + inFile.getName());
System.out.println(inFile.getPath());
FileInputStream fis = null;
BufferedInputStream bis = null;
FileOutputStream fos = null;
BufferedOutputStream bos = null;

try {

    fis = new FileInputStream(inFile);
    fos = new FileOutputStream(outFile);
    bis = new BufferedInputStream(fis);
    bos = new BufferedOutputStream(fos);

    int input = 0;
    byte[] buf = new byte[16384];

    if(inFile.exists()) {

        // 버퍼를 사용하지 않으면 15초 이상 소요된다.
        /*
        long start = System.currentTimeMillis();
        while((input = fis.read()) != -1) {
            fos.write(input);
        }
        long end = System.currentTimeMillis();
        System.out.println("복사시간(버퍼x) : " + (end - start));
        */

        // 버퍼의 크기에 따라서 파일 복사 시간이 약간씩 차이가 난다.
        long start = System.currentTimeMillis();
        while((input = bis.read(buf)) != -1) {
            bos.write(buf, 0, input);
        }
        long end = System.currentTimeMillis();
        System.out.println("복사시간(buffered) : " + (end - start));
    }

} catch(IOException e) {
    e.printStackTrace();

} finally {

    try {
```

```
                if(fos != null) fos.close();
                if(fis != null) fis.close();
            } catch(IOException e) { }
        }
    }
}
```

10. 스레드(Thread)

사용자가 어떤 프로그램을 실행하게 되면 그 프로그램은 OS(Operating System)로부터 프로그램 실행에 필요한 자원을 할당받아 실행하게 되는데 이렇게 실행된 하나의 프로그램을 프로세스(Process)라 한다. 하나의 애플리케이션은 여러 개의 프로세스를 만들어 실행하면서 동시에 여러 작업을 처리할 수도 있고 하나의 프로세스 안에서 동시에 여러 작업을 처리할 수도 있는데 이렇게 동시에 여러 작업을 처리하는 것을 멀티태스킹(Multi Tasking)이라 한다.
하나의 애플리케이션에서 여러 개의 프로세스를 실행해 동시에 여러 작업을 처리하는 것을 멀티 프로세스 방식이라고 하며 하나의 프로세스 안에서 여러 개의 스레드를 실행해 동시에 여러 작업을 처리하는 것을 멀티 스레드 방식이라고 한다.
멀티 프로세스 방식으로 동시에 여러 작업을 처리하는 것은 애플리케이션 단위의 멀티태스킹이라고 할 수 있으며 멀티 스레드 방식으로 동시에 여러 작업을 처리하는 것은 하나의 애플리케이션 안에서의 이루어지는 멀티태스킹이라고 할 수 있다.
멀티 프로세스 방식은 각각의 프로세스 마다 OS로부터 자원을 할당받아 실행되기 때문에 다른 프로세스에 영향을 주지 않고 서로 독립적으로 동작하지만 멀티 스레드 방식은 하나의 프로세스 안에서 여러 개의 스레드로 실행되기 때문에 하나의 스레드에서 오류가 발생하게 되면 전체 프로그램이 비정상 종료 될 수 있다는 단점이 있다. 하지만 멀티 프로세스는 각각의 프로세스 마다 애플리케이션 실행에 필요한 자원을 할당 받아 프로세스를 새로 시작하기 때문에 프로그램 실행에 많은 시간이 걸리고 자원의 낭비가 생길 수 있는 단점이 있는 반면 멀티 스레드는 하나의 프로세스에 할당된 자원을 공유하기 때문에 자원을 효율적으로 사용하면서 동시에 여러 작업을 할 수 있다는 장점이 있다. 또한 멀티 스레드는 멀티 프로세스에 비해 적은 시스템 자원으로 실행되기 때문에 경량 프로세스(LightWeightProcess)라고도 한다. 하지만 멀티 스레드는 같은 프로세스 내에서 공유된 자원을 활용해 실행되기 때문에 동기화(Synchronization), 교착상태(Deadlock)와 같은 문제점이 발생할 수 있다. 동기화란 하나의 스레드가 프로세스 자원에 접근해 작업을 할 때 다른 스레드가 동일한 자원에 접근할 수 없도록 하는 것을 말하여 교착상태는 하나의 스레드가 프로세스 자원에 접근해 작업하는 동안 다른 스레드가 그 자원을 사용하기 위해 대기하는 상태를 말한다.
하나의 프로세스 안에서 사용할 수 있는 스레드의 개수는 제한되어 있지 않고 스레드 또한 프로세스 안에서 개별적인 메모리 공간이 필요하기 때문에 프로세스의 메모리 한계에 따라서 사용할 수 있는 스레드의 개수가 결정된다.
자바 애플리케이션은 main() 메서드가 실행되면서 프로그램이 시작되는데 이 main() 메서드도 스레드로 동작하기 때문에 메인 스레드라고 부르기도 한다.
자바에서 싱글 스레드 방식으로 동작하는 애플리케이션은 main() 메서드가 종료되면 프로그램이 종료되지만 멀티 스레드 방식으로 동작하는 애플리케이션은 main() 메서드가 종료되었다 하더라도 현재 실행되는 스레드가 존재하면 프로그램은 종료되지 않는다.

10.1 스레드 구현하기

스레드는 Runnable 인터페이스를 구현하거나 Thread 클래스를 상속 받아 구현할 수 있다.
자바는 단일 상속만을 허용하기 때문에 다른 클래스를 상속 받아야 되는 경우에는 Thread 클래스를 상속 받을 수 없으므로 Runnable 인터페이스를 구현(implements)하고 상속(extends)이 필요한 클래스를 상속(extends) 하는 형식으로 스레드를 구현하면 된다.
Runnable 인터페이스는 run() 추상 메서드 하나만 정의하고 있기 때문에 아래와 같은 방식으로 Runnable 인터페이스를 상속받아 구현하면 된다.

```
class RunnableImpl implements Runnable {
    @Override
    public void run() {
        // 멀티 스레드에서 처리할 코드
    }
}
```

Thread 클래스를 상속받는 방법은 Thread 클래스로부터 상속받은 run() 메서드를 아래와 같은 방식으로 오버라이딩 하여 구현할 수 있다.

```
class MultiThread extends Thread {
    // Thread 클래스의 run() 메서드를 오버라이딩 한다.
    public void run() {
        // 멀티 스레드에서 처리할 코드
    }
}
```

▶ Runnable 인터페이스를 구현해 스레드 만들기

- com.javastudy.ch10.thread

```
public class RunnableImplTest {
    static long mainNum;

    public static void main(String[] args) {

        /* Runnable 인터페이스를 구현(implements)해 스레드 만들기
         *
         * 1. Runnable 인터페이스를 구현(implements)한 클래스를 작성한다.
         *
         * 2. Runnable 인터페이스를 구현한 클래스의 인스턴스를 생성한다.
         **/
        RunnableImpl r1 = new RunnableImpl(20);
        RunnableImpl r2 = new RunnableImpl(20);

        /* 3. Thread 클래스의 인스턴스를 생성하면서 위에서 생성한 Runnable
         * 인터페이스를 구현한 클래스의 인스턴스를 생성자를 통해서 전달한다.
         **/
        Thread t1 = new Thread(r1);
        Thread t2 = new Thread(r2);
```

```
        /* 4. Thread 클래스의 start() 메서드를 호출해 스레드를 실행한다.
         * start() 메서드가 호출되면 이 메서드 안에서 Thread 클래스의 인스턴스
         * 메서드인 run() 메서드가 호출되고 이 run() 메서드 안에서 생성자를 통해
         * 전달받은 Runnable 인터페이스를 구현한 클래스의 인스턴스 메서드인 run()
         * 메서드가 호출된다. 이 run() 메서드가 종료되면 현재 스레드도 종료된다.
         *
         * 하나의 스레드에 대해서 start() 메서드는 단 한 번만 호출 할 수 있다.
         * start() 메서드가 호출되면 스레드가 작업을 실행할 호출스택(Call Stack)이
         * 생성되고 run() 메서드가 호출되어 그 호출스택에 run() 메서드가 첫 번째에
         * 자리하게 된다. 모든 스레드는 자신만의 독립된 작업을 위해서 호출스택이 필요하다.
         **/
        t1.start();
        t2.start();

        System.out.println("main 메서드 종료");
    }
}

// Runnable 인터페이스를 구현하는 클래스
class RunnableImpl implements Runnable {
    int num;

    public RunnableImpl(int num) {
        this.num = num;
    }

    /* 스레드에서 동작할 코드는 아래와 같이 run() 메서드를 오버라이딩해 구현한다.
     * 스레드는 run() 메서드가 호출 되면서 시작되고 이 메서드가 종료되면 종료된다.
     **/
    @Override
    public void run() {

        for(int i = 1; i <= num; i++) {
            RunnableImplTest.mainNum += i;
            System.out.println(Thread.currentThread()
                    + " - num 1 : " + RunnableImplTest.mainNum);

            try {
                /* 현재 실행중인 스레드를 1초 동안 잠을 재운다.
                 * 스레드가 1초 동안 일시정지 되고 실행 대기 상태로 간다.
                 **/
                Thread.sleep(1000);
            } catch(InterruptedException e) {
                return;
            }
```

```
        }
    }
}
```

▶ Thread 클래스를 상속해 스레드 만들기

- com.javastudy.ch10.thread

```
public class ThreadExtends {
    static int mainNum;

    public static void main(String[] args) {

        /* Thread 클래스를 상속(extends)해 스레드 만들기
         *
         * 1. Thread 클래스를 상속(extends)한 클래스를 작성한다.
         *
         * 2. Thread 클래스를 상속한 클래스의 인스턴스를 생성한다.
         **/
        MultiThread t1 = new MultiThread(20);
        MultiThread t2 = new MultiThread(20);

        /* 3. Thread 클래스로부터 상속받은 start() 메서드를 호출해 스레드를 실행한다.
         * Runnable 인터페이스를 구현한 클래스의 run() 메서드는 Thread 클래스의
         * 인스턴스를 통행 실행되지만 Thread 클래스를 직접 상속한 클래스는 부모인
         * Thread 클래스의 모든 멤버를 상속 받았기 때문에 자신의 인스턴스를 생성하고
         * 부모인 Thread 클래스로부터 상속받은 start() 메서드를 호출 할 수 있다.
         * Thread 클래스 또한 Runnable 인터페이스를 구현한 클래스 이다.
         *
         * 하나의 스레드에 대해서 start() 메서드는 단 한 번만 호출 할 수 있다.
         * start() 메서드가 호출되면 스레드가 작업을 실행할 호출스택(Call Stack)이
         * 생성되고 run() 메서드가 호출되어 그 호출스택에 run() 메서드가 첫 번째에
         * 자리하게 된다. 모든 스레드는 자신만의 독립된 작업을 위해서 호출스택이 필요하다.
         **/
        t1.start();
        t2.start();

        System.out.println("main 메서드 종료");
    }
}

// Thread 클래스를 상속하는 클래스
class MultiThread extends Thread {
    int num;
```

```
    public MultiThread(int num) {
        this.num = num;
    }

    /* Thread 클래스를 상속 받는 클래스는 아래와 같이 run() 메서드를 오버라이딩
     * 해야 한다. 스레드는 run() 메서드가 호출 되면서 시작되고 이 메서드가 종료되면
     * 스레드 또한 종료된다. 만약 run() 메서드를 오버라이딩 하지 않으면 부모 클래스인
     * Thread 클래스의 run() 메서드가 호출되는데 Thread 클래스의 run() 메서드는
     * 특별한 작업을 하는 코드가 없기 때문에 스레드가 바로 종료된다.
     **/
    @Override
    public void run() {
        for(int i = 1; i <= num; i++) {
            ThreadExtends.mainNum += i;
            System.out.println(currentThread()
                    + " - num 1 : " + ThreadExtends.mainNum);

            try {
                /* 현재 실행중인 스레드를 1초 동안 잠을 재운다.
                 * 스레드가 1초 동안 일시정지 되고 실행 대기 상태로 간다.
                 **/
                Thread.sleep(1000);
            } catch(InterruptedException e) {
                return;
            }
        }
    }
}
```

▶ start() 메서드와 run() 메서드

- com.javastudy.ch10.thread

```
public class ThreadCallStack {

    public static void main(String[] args) {

        Thread t = new TestThread01(10);

        /* 스레드 인스턴스를 생성하고 start() 메서드를 호출하면 새로운 스레드가
         * 시작된다. 이때 작업에 필요한 호출스택(Call Stack)을 생성한 후 start()
         * 메서드 안에서  run() 메서드를 호출한다. 모든 스레드는 자신만의 독립적인
         * 작업을 위해서 호출스택을 필요로 한다. 그래서 새로운 스레드를 생성하고
         * 실행할 때 마다 항상 새로운 호출스택이 만들어 진다.
         *
```

```
         * t.start()가 호출되면 새로운 스레드가 실행되면서 새로운 호출스택을
         * 생성하기 때문에 start() 메서드 안에서 호출된 TestThread01 클래스의
         * run() 메서드는 새로운 호출스택에서 첫 번째 메서드로 자리를 잡고 이 run()
         * 메서드에서 호출한 add() 메서드가 그 위에 자리를 잡는다.
         **/
        t.start();

        /* 스레드의 인스턴스를 생성하고 run() 메서드를 호출하면 단지 참조 변수가
         * 가리키는 인스턴스의 run() 메서드를 호출하는 것이 되므로 아래와 같이
         * t.run() 메서드를 호출하면 새로운 스레드가 실행되는 것이 아니기 때문에
         * main() 메서드가 실행되는 main 스레드의 호출스택에 자리하게 된다.
         **/
        //t.run();

         /* main() 메서드가 종료되어도
          * 현재 실행중인 스레드가 존재하면 프로그램은 종료되지 않는다.
          **/
        System.out.println("main() 메서드 종료");
    }
}

class TestThread01 extends Thread {
    int num;
    int sum;

    public TestThread01(int num) {
        this.num = num;
    }

    @Override
    public void run() {
        add(num);
    }

    public void add(int num) {
        try {
            throw new Exception();

        } catch(Exception e) {
            e.printStackTrace();
        }

        for(int i = 1; i <= num; i++) {
            sum += i;
```

```
            // 현재 실행중인 스레드 정보를 콘솔에 출력한다.
            System.out.println(Thread.currentThread() + " - num 1 : " + sum);

            try {
                /* 현재 실행중인 스레드를 1초 동안 잠을 재운다.
                 * 스레드가 1초 동안 일시정지 되고 실행 대기 상태로 간다.
                 **/
                Thread.sleep(1000);
            } catch(InterruptedException e) {
                return;
            }
        }
    }
}
```

▶ 스레드 우선순위

- com.javastudy.ch10.thread

```
public class ThreadPriority01 {

    /* 스레드는 실행에 대한 우선순위(Priority) 속성을 가지고 있다.
     * 이 우선순위에 따라서 스레드가 실행되는 시간이 달라진다.
     * 중요도가 높은 스레드에 우선순위 값을 높게 지정해 그 스레드가 프로세스로 부터
     * 실행 시간을 더 많이 할당 받을 수 있도록 할 수 있다.
     * 스레드의 우선순위는 1 ~ 10까지 지정할 수 있으며 숫자가 클수록 우선순위가
     * 높다. 또한 스레드 우선순위 값은 스레드를 생성한 스레드의 우선순위 값을 상속
     * 받는다. 참고로 main() 메서드를 수행하는 메인 스레드의 우선순위는 5이다.
     * 그래서 main() 메서드에서 생성된 스레드는 우선순위 값이 5로 자동 지정된다.
     * 스레드의 우선순위가 같으면 각각의 스레드에 같은 실행 시간이 주어진다.
     * 하지만 이 우선순위가 스레드의 실행시간을 반드시 보장해 주는 것은 아니다.
     * 그 이유는 OS의 프로세스 스케줄러에 따라 다르고 JVM 마다 다를 수 있기
     * 때문이다. 하지만 JVM은 스레드 스케줄링 시 기본적으로 우선순위가 높은 스레드를
     * 우선적으로 선택해 실행해 준다.
     **/
    public static void main(String[] args) {

        Thread t1 = new PriorityThread("/");
        Thread t2 = new PriorityThread("*");

        System.out.println(t1.getName() + " - [/]: " + t1.getPriority());
        System.out.println(t2.getName() + " - [*]: " + t2.getPriority());

        System.out.println("스레드의 최소 우선순위 값 : " + Thread.MIN_PRIORITY);
        System.out.println("스레드의 기본 우선순위 값 : " + Thread.NORM_PRIORITY);
```

```
        System.out.println("스레드의 최대 우선순위 값 : " + Thread.MAX_PRIORITY);

        /* 스레드의 최소 우선순위 값부터 최대 우선순위 값 사이의 정수를
         * 지정해도 되고 바로 위의 상수를 사용해 우선순위를 지정할 수도 있다.
         **/
        // t2.setPriority(10);
        t2.setPriority(Thread.MAX_PRIORITY);

        t1.start();
        t2.start();
    }
}

// Thread 클래스를 상속한 스레드 클래스
class PriorityThread extends Thread {
    private String pattern;

    public PriorityThread(String pattern) {
        this.pattern = pattern;
    }

    @Override
    public void run() {
        for(int i = 0; i <= 500; i++) {
            System.out.print(pattern);
            for(int j = 0; j <=1000000; j++);
        }
    }
}
```

10.2 스레드 상태

스레드의 인스턴스를 생성하고 start() 메서드를 호출하면 곧 바로 스레드가 실행되는 것처럼 보이지만 그 스레드는 실행되는 것이 아니라 실행대기(Runnable) 상태에 놓이게 된다. 실행대기 상태란 스케줄링(Scheduling)에 의해 선택되지 못해 자원을 할당 받기 위해 대기하는 상태를 의미한다. 스레드 우선순위에서도 잠깐 언급했지만 스레드는 OS의 프로세스 스케줄러(Scheduler)에 따라서 다르기 때문에 스레드 스케줄링에 의해 선택된 스레드만이 비로소 자원을 할당 받아 실행될 수 있는 것이다. 스레드가 스케줄링에 의해 선택되어 실행되면 실행(Running) 상태에 놓이게 되고 이 상태의 스레드는 run() 메서드가 종료되기 전에도 스케줄링에 의해 다시 실행대기 상태로 갈수 있다. 이렇게 run() 메서드가 종료되기 전까지 실행대기 상태와 실행 상태를 여러 번씩 오갈 수 있으며 run() 메서드가 종료되면 스레드도 종료(Terminated) 된다. 경우에 따라서 실행대기 상태에서 실행상태로 가지 못하고 일시정지(Wating) 상태로 갈 수 있으며 이 상태는 스레드가 실행할 수 없는 상태를 의미한다. 일시정지 상태에 놓인 스레드는 바로 실행상태로 가지 못하며 다시 실행대기 상태로 가야 실행상태가 될 수 있다.
스레드의 현재 상태는 getState() 메서드를 호출하면 알 수 있는데 이 메서드는 아래 표10-1과 같이 Thread 클래스에 정의된 State 열거 상수에 정의된 값을 반환한다.

스레드 상태	열거 상수	설 명
객체생성	NEW	스레드 객체는 생성되었지만 아직 start() 메서드가 호출되지 않은 상태로 start() 메서드가 호출되면 RUNNABLE 상태가 된다.
실행대기	RUNNABLE	스케줄러로부터 자원을 할당 받기 위해 대기하는 상태로 언제든 실행될 수 있는 상태이다.
일시정지	WATING	다른 스레드가 통지할 때까지 기다리는 상태로 다른 스레드가 notify(), notifyAll() 메서드를 호출해 불러주기를 기다리는 상태이다.
	TIMED_WAITING	지정한 시간만큼 기다리는 상태로 스레드 안에서 sleep(m) 메서드를 호출해 m 밀리 초만큼 일시 정지된 상태이다.
	BLOCKED	사용하고자 하는 객체의 락(Lock)이 풀릴 때까지 기다리는 상태이다.
종료	TERMINATED	스레드가 실행을 마치고 종료된 상태이다.

표 10-1

▶ 스레드 상태 체크하기

- com.javastudy.ch10.threadstate

```
public class ThreadStateControl01 {

    public static void main(String[] args) {

        TestThread t = new TestThread("TestThread");
```

```
        ThreadStatePrint ts = new ThreadStatePrint(t);
        ts.setName("ThreadStatePrint");
        ts.start();
        System.out.println("main 종료");
    }
}

// Thread 클래스를 상속해 스레드 상태를 출력하는 클래스
class ThreadStatePrint extends Thread {
    TestThread t;

    public ThreadStatePrint(TestThread t) {
        this.t = t;
    }

    @Override
    public void run() {
        while(true) {

            // TestThread 스레드의 현재 상태를 구한다.
            Thread.State state = t.getState();
            System.out.println(this.getName() + " 상태 : " + this.getState());
            System.out.println(t.getName() + " 상태 : " + state);

            if(state == Thread.State.NEW) {
                /* start() 메서드를 호출하면 곧 바로 스레드가 실행되는 것이 아니라
                 * RUNNABLE 상태가 된다. t 스레드는 현재 실행대기 상태에 놓인다.
                 **/
                t.start();
                System.out.println("t 시작");
            }

            if(state == Thread.State.TERMINATED) {
                // TestThread가 종료 상태면 반복문을 빠져나가 종료한다.
                System.out.println("ThreadStatePrint 종료...");
                break;
            }

            try {
                /* ThreadStatePrint 클래스의 스레드가 지정한 시간동안 일시정지
                 * 되는 것이 아니라 현재 실행중인 스레드를 0.5초 정지 시킨다.
                 **/
                Thread.sleep(500);
            } catch(InterruptedException e) {
                e.printStackTrace();
```

```
            }
        }
    }
}

class TestThread extends Thread {

    public TestThread(String name) {
        super(name);
    }

    @Override
    public void run() {

        int cnt = 0;
        String str = " ";
        for(int i = 0; i <= 30000; i++) {
            str += i;
        }

        while(true) {
            try {
                /* TestThread 클래스의 스레드가 지정한 시간동안 일시정지
                 * 되는 것이 아니라 현재 실행중인 스레드를 0.5초 정지 시킨다.
                 **/
                Thread.sleep(500);
                cnt++;
                if(cnt > 3) {
                    break;
                }
            } catch(InterruptedException e) {
                e.printStackTrace();
            }
        }

        for(int i = 0; i <= 30000; i++) {
            str += i;
        }
    }
}
```

▶ 스레드 상태 제어 메서드

멀티 스레드 프로그램을 사용하다 보면 가끔씩 프로그램이 다운되거나 먹통이 되는 경우가 가끔 발

생하게 되는데 이는 스레드 상태를 정교하게 제어하지 못해서 발생하는 경우 대부분 이다. 스레드 상태를 정교하게 관리해서 스레드 스케줄링이 제대로 이루어지도록 프로그래밍 하려면 스레드 상태와 관련된 메서드를 제대로 이해하고 있어야 한다.
스레드를 적절히 사용하게 되면 시스템 자원을 효율적으로 사용해 프로그램 성능을 높일 수 있지만 그렇지 못한 경우 프로그램의 치명적인 버그가 되기 때문에 스레드를 정확하게 제어하기 위해서 반드시 스레드 상태를 제어하는 메서드를 잘 알고 있어야 한다.
다음 표 10-2는 스레드 상태와 관련된 메서드를 설명한 것이다.

메서드	설 명
sleep(long millis) sleep(long millis, int nanos)	매개변수로 지정한 시간동안 스레드를 일시정지 상태로 만들고 지정한 시간이 지나면 실행대기 상태로 간다.
join() join(long millis) join(long millis, int nanos)	join() 메서드를 호출한 스레드는 일시정지 상태가 되고 실행대기 중인 스레드가 매개변수로 주어진 시간동안 실행되도록 한다. 작업이 종료되면 join() 메서드를 호출한 스레드로 돌아와 작업을 계속한다.
interrupt()	sleep(), join() 메서드에 의해서 일시정지 상태에 있는 스레드를 실행대기 상태로 만든다. 해당 스레드에서 InterruptedException을 발생시켜 예외처리(catch 블록)에서 일시정지 상태를 벗어나 실행대기 상태로 가게하거나 종료상태로 가게 구현하면 된다.
wait() wait(long millis) wait(long millis, int nanos)	동기화(synchronized) 블록에서 스레드를 일시정지 상태로 만든다. 매개변수로 지정한 시간이 지나면 다시 실행대기 상태가 된다. 매개변수를 지정하지 않으면 notify(), notifyAll() 메서드에 의해 실행대기 상태로 갈 수 있다.
notify() notifyAll()	동기화(synchronized) 블록에서 일시정지 상태에 있는 스레드를 실행대기 상태로 만든다.
yield()	스레드 자신에게 주어진 시간을 실행대기 중인 스레드에게 실행을 양보하고 실행대기 상태로 간다.
sespend()	스레드를 일시정지 상태로 만든다. resume() 메서드를 호출하면 실행대기 상태로 간다. Deprecated 되어 이 메서드 대신에 wait() 메서드 사용을 권장
resume()	suspend() 메서드에 의해 일시정지 상태에 있는 스레드를 실행대기 상태로 만든다. Deprecated 되어 이 메서드 대신에 notify(), notifyAll() 메서드 사용을 권장
stop()	스레드를 즉시 종료시킨다. 실행중인 스레드를 stop() 메서드를 사용해 즉시 종료하게 되면 그 스레드가 사용 중이던 시스템 자원이 불안전한 상태로 남아있기 때문에 Deprecated 되었다. 플래그 변수나 interrupt() 메서드를 이용해 자원을 정리한 후 스레드가 종료되도록 구현

표 10-2

▶ 다른 스레드가 종료될 때 까지 기다리는 join() 메서드

스레드는 독립적으로 실행되는 것이 기본이지만 현재 실행중인 스레드 안에서 다른 스레드를 실행시켜 그 스레드가 작업을 마칠 때 까지 기다렸다가 다음 작업을 수행해야 하는 경우도 있는데 이때는 현재 실행되는 스레드 안에서 실행이 완료되기를 기다려야 하는 스레드의 join() 메서드를 호출하면 현재 실행되는 스레드가 종료되지 않고 실행이 완료되기를 기다려야 하는 스레드가 종료될 때까지 기다리게 된다.

- com.javastudy.ch10.threadstate

```
public class ThreadJoinMethod01 {

    public static void main(String[] args) {

        SumThread t = new SumThread(1, 100);
        t.setName("SumThread");
        t.start();

        try {
            // SumThread가 종료될 때 까지 main() 스레드는 종료되지 않고 기다린다.
            t.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        /* 위에서 t.join() 메서드가 호출되지 않았다면 SumThread 스레드가
         * 작업을 완료하기 전에 main 스레드는 종료되었을 것이다.
         *
         * 위의 try { } catch() { } 블록을 주석 처리하고 테스트 하면
         * SumThread가 실행을 완료할 때까지 main() 메서드가
         * 기다리지 않고 아래 코드가 실행되기 때문에 아래의 결과는 0이 된다.
         **/
        System.out.println("1 ~ 100까지의 합 : " + t.getTotal());
        System.out.println("main 스레드 종료");
    }
}

class SumThread extends Thread {

    private int total;
    private int start;
    private int end;

    public SumThread(int start, int end) {
```

```
        this.start = start;
        this.end = end;
    }

    @Override
    public void run() {

        for(int i = start; i <= end; i++) {
            total += i;
        }

        System.out.println("SumThread 종료");
    }

    public int getTotal() {
        return total;
    }

    public void setTotal(int total) {
        this.total = total;
    }

    public int getStart() {
        return start;
    }

    public void setStart(int start) {
        this.start = start;
    }

    public int getEnd() {
        return end;
    }

    public void setEnd(int end) {
        this.end = end;
    }
}
```

10.3 스레드 동기화

main 스레드만 사용할 경우 싱글 스레드로 프로세스 자원을 사용하기 때문에 공유자원 사용에 대한 문제가 발생하지 않지만 멀티 스레드일 경우 프로세스 자원을 서로 공유하여 사용하기 때문에 현재 실행중인 스레드의 작업이 다른 스레드에 의해서 영향을 받을 수 있다. 예를 들면 스레드 A가 메모리에 접근해 작업을 하는 도중에 스레드 B가 그 메모리의 내용을 변경했다면 원래 의도한 결과를 얻을 수 없게 된다. 이처럼 멀티 스레드 작업에서 여러 스레드가 자원을 공유하는 경우에 발생하는 문제를 방지하기 위해 스레드 동기화(Synchronization)라는 개념이 도입되었다.
스레드 동기화란 현재 실행중인 스레드가 공유 자원에 접근해 작업하는 동안 다른 스레드가 같은 공유자원에 접근하지 못하도록 영역을 지정해 잠금(Lock)을 하는 것을 의미한다.
스레드의 동기화를 위해 지정된 영역을 임계영역(Critical Section)이라고 하며 이 영역으로 지정되면 현재 실행중인 스레드가 작업을 마칠 때 까지 다른 스레드가 접근할 수 없도록 잠금(Lock)을 하게 된다.
스레드 동기화는 여러 가지 방법으로 구현할 수 있겠지만 우리는 synchronized 키워드를 사용해 동기화를 설정하는 방법과 wait(), notify(), notifyAll() 메서드를 이용해 동기화를 제어하는 방법에 대해서 알아볼 것이다.
synchronized 키워드를 이용한 스레드 동기화는 아래와 같이 메서드 전체 또는 메서드 안에서 특정 코드만 블록으로 지정해 임계영역을 설정하는 방법이 있다.

▶ 메서드 전체를 임계영역으로 설정

```
public synchronized void multiCalculator() {

    /* synchronized 키워드로 메서드 전체가 임계영역으로 설정된다.
     * 스레드는 이 메서드가 호출된 시점부터 이 메서드가 포함된 객체의
     * Lock을 얻어 작업을 수행하고 작업이 끝나면 Lock을 반환한다.
     **/

}
```

▶ 메서드의 일부 코드만 임계영역으로 설정

```
public void multiply() {

    /* 아래 동기화 블록(synchronized(this) 블록)에서 파라미터에 this를 지정했는데
     * 항상 this가 되는 것이 아니라 동기화 대상 즉 락(Lock)을 걸 객체의 참조를
     * 파라미터로 지정해야 한다.
     **/

    synchronized(this) {

        /* 이 synchronized(this) { } 블록만 임계영역으로 설정된다.
         * () 안에는 잠금(Lock)을 할 객체의 참조변수를 지정하면 된다.
         * 이 블록 안으로 들어오면서 부터 스레드는 지정된 객체의 Lock을 얻게 되고
         * 블록을 벗어나면 Lock을 반납하게 된다. Lock의 획득과 반납은 JVM에 의해
```

```
            * 자동으로 이루어지므로 적절히 임계영역만 설정해 주면 된다.
            **/

        }
    }
```

▶ SumClass의 add() 메서드에 동기화 설정을 못해서 공유자원 문제가 발생하는 경우

- com.javastudy.ch10.threadnosync

```
public class SumClassNoSynchronized {

    public static void main(String[] args) {

        Calculator ca = new Calculator();
        Thread t1 = new Thread(ca, "t1");
        Thread t2 = new Thread(ca, "t2");

        /* 두 스레드가 시작되면 SumClass의 add() 메서드를 통해 sum에 접근하게 되는데
         * 이때 t1이 add() 메서드를 호출하고 다음 호출 전에 t2가 add() 메서드를 호출하는
         * 문제가 발생할 수 있다. 즉 하나의 스레드에서 작업하는 자원에 다른 스레드가
         * 접근해 그 자원의 값을 변경할 수 있는 문제가 발생하게 된다. 그래서 하나의
         * 스레드가 공유 자원에 접근할 때 다른 스레드가 동시에 접근할 수 없도록 Lock을
         * 걸어야 하는데 이를 스레드 동기화라고 한다.
         **/
        t1.start();
        t2.start();
    }
}

class SumClass {
    private int sum;

    public int add(int num) {
        if(sum <= 300) {
            sum += num;
            try {
                // 현재 실행 중인 스레드를 1초 잠재운다.
                Thread.sleep(1000);
            } catch(InterruptedException e) { }
        }
        return sum;
    }
```

```
    public int getSum() {
        return sum;
    }
}

class Calculator implements Runnable {
    SumClass sc = new SumClass();

    @Override
    public void run() {

        /* SumClass 클래스의 add() 메서드에 synchronized로 임계영역을 설정하지 않으면
         * t1이 add() 메서드에 접근해 값을 더 하고 동시에 t2가 접근해 값을 더 하여
         * t1이 add() 메서드를 호출해 값을 더 하고 출력할 당시 합계는 t1이 더한 값과
         * t2가 더한 값이 출력된다. 또한 t2가 add() 메서드를 호출해 값을 더 하고
         * 출력할 당시 합계는 t1과 t2가 한 번씩 더한 값이 되어야 하지만 t1이 추가로
         * 한 번 더 더한 값이 출력되는 현상이 발생한다. 이렇게 멀티 스레드가 하나의 자원에
         * 접근해 작업하는 경우 의도하지 못한 동기화 문제가 발생한다.
         **/
        while(sc.getSum() <= 300) {

            // 10 ~ 100 사이의 난수를 발생해 add() 메서드 인수로 지정
            int num = (int) (Math.random() * 10 + 1) *10;
            System.out.println(Thread.currentThread().getName()
                    + " num : " + num + ",  sum : " + sc.add(num));
        }
    }
}
```

▶ SumClass의 add() 메서드에 synchronized 키워드로 동기화 설정하기

- com.javastudy.ch10.threadsync

```
public class SumClassSynchronized {

    public static void main(String[] args) {

        Calculator ca = new Calculator();
        Thread t1 = new Thread(ca, "t1");
        Thread t2 = new Thread(ca, "t2");

        /* 두 스레드가 시작되면 SumClass의 add() 메서드를 통해 sum에 접근하게 되는데
         * 이때 t1이 add() 메서드를 호출하고 다음 호출 전에 t2가 add() 메서드를 호출하는
         * 문제가 발생할 수 있다. 즉 하나의 스레드에서 작업하는 자원에 다른 스레드가
         * 접근해 그 자원의 값을 변경할 수 있는 문제가 발생하게 된다. 그래서 하나의
```

```
         * 스레드가 공유 자원에 접근할 때 다른 스레드가 동시에 접근할 수 없도록 Lock을
         * 걸어야 하는데 이를 스레드 동기화라고 한다.
         **/
        t1.start();
        t2.start();
    }
}

class SumClass {
    private int sum;

    /* synchronized 키워드로 아래 메서드 전체가 임계영역으로 설정된다.
     * 스레드는 이 메서드가 호출된 시점부터 이 메서드가 포함된 객체의
     * Lock을 얻어 작업을 수행하고 작업이 끝나면 Lock을 반환한다.
     **/
    public synchronized int add(int num) {
        if(sum <= 300) {
            sum += num;
            try {
                // 현재 실행 중인 스레드를 1초 잠재운다.
                Thread.sleep(1000);
            } catch(InterruptedException e) { }
        }
        return sum;
    }

    public int getSum() {
        return sum;
    }
}

class Calculator implements Runnable {
    SumClass sc = new SumClass();

    @Override
    public void run() {

        /* SumClass 클래스의 add() 메서드에 synchronized로 임계영역을 설정하지 않으면
         * t1이 add() 메서드에 접근해 값을 더 하고 동시에 t2가 접근해 값을 더 하여
         * t1이 add() 메서드를 호출해 값을 더 하고 출력할 당시 합계는 t1이 더한 값과
         * t2가 더한 값이 출력된다. 또한 t2가 add() 메서드를 호출해 값을 더 하고
         * 출력할 당시 합계는 t1과 t2가 한 번씩 더한 값이 되어야 하지만 t1이 추가로
         * 한 번 더 더한 값이 출력되는 현상이 발생한다. 이렇게 멀티 스레드가 하나의 자원에
         * 접근해 작업하는 경우 의도하지 못한 동기화 문제가 발생한다.
         *
```

```
            * 그러므로 하나의 스레드가 SumClass의 add() 메서드에 접근해 작업을 할 때
            * 다른 스레드가 SumClass의 add() 메서드에 접근하지 못하도록 해야 한다.
            * 다시 말해 t1이 add() 메서드를 호출하고 실행이 끝나기 전에 t2가 add() 메서드를
            * 호출하면 t1이 호출한 add() 메서드가 완전히 종료될 때 까지 두 번째 스레드인
            * t2가 add() 메서드에 접근하지 못하도록 대기시켜야 한다. 그래서 synchronized
            * 키워드를 add() 메서드에 적용해 동기화 설정을 했다.
            **/
            while(sc.getSum() <= 300) {

                // 10 ~ 100 사이의 난수를 발생해 add() 메서드 인수로 지정
                int num = (int) (Math.random() * 10 + 1) * 10;
                System.out.println(Thread.currentThread().getName()
                        + " num : " + num + ",  sum : " + sc.add(num));
            }
        }
    }
```

▶ Account 클래스에 동기화 설정을 못해서 공유자원 문제가 발생하는 경우

- com.javastudy.ch10.threadnosync

```
public class AccountClassNoSynchronized {

    public static void main(String[] args) {

        // 계좌 클래스의 인스턴스를 생성하면서 계좌 잔액을 5000원으로 설정
        Runnable r = new Withdraw(new Account(5000));
        Thread t1 = new Thread(r, "t1");
        Thread t2 = new Thread(r, "t2");

        /* 두 스레드가 시작되면 Account의 withdraw() 메서드를 통해 balance에 접근하게
         * 되는데 이때 t1이 withdraw() 메서드를 호출하고 다음 호출 전에 t2가 withdraw()
         * 메서드를 호출하는 문제가 발생할 수 있다. 즉 하나의 스레드에서 작업하는 자원에
         * 다른 스레드가 접근해 그 자원의 값을 변경할 수 있는 문제가 발생하게 된다.
         * 그래서 하나의 스레드가 공유 자원에 접근할 때 다른 스레드가 동시에 접근할 수
         * 없도록 Lock을 걸어야 하는데 이를 스레드 동기화라고 한다.
         **/
        t1.start();
        t2.start();
    }
}

// 계좌 클래스
class Account {
```

```
    // 계좌 잔액
    private int balance;

    public Account(int balance) {
        this.balance = balance;
    }

    public int getBalance() {
        return balance;
    }

    // 계좌에서 지정한 금액 만큼 인출하는 메서드
    public int withdraw(int amount) {

        if(balance >= amount) {
            try {
                // 현재 실행 중인 스레드를 1초 잠재운다.
                Thread.sleep(1000);
            }catch(InterruptedException e) {
                e.printStackTrace();
            }
            balance -= amount;
        }
        return balance;
    }
}

// 계좌에 있는 돈을 인출하는 클래스
class Withdraw implements Runnable {
    Account account;

    public Withdraw(Account account) {
        this.account = account;
    }

    @Override
    public void run() {

        /* Account 클래스의 withdraw() 메서드에 synchronized로 임계영역을 설정하지
         * 않으면 t1이 출금하는 동시에 t2가 출금하는 문제가 발생해 잔액이 마이너스가
         * 되거나 잔액이 정확하게 계산되지 못하는 문제가 발생한다. 그러므로 withdraw()
         * 메서드에 synchronized 키워드로 임계영역을 설정해 t1이 출금하는 동안
         * 또 다른 스레드인 t2는 withdraw() 메서드에 접근하지 못하도록 대기시켜야 한다.
         **/
        while(account.getBalance() > 0){
```

```
            // 한 번에 1000씩 차감
            int amount = 1000;
            System.out.println(Thread.currentThread().getName() +
                    " : " + amount + ", 잔액 : " + account.withdraw(amount));
        }
    }
}
```

▶ Account 클래스에 synchronized 키워드로 동기화 블록 설정하기

- com.javastudy.ch10.threadsync

```
public class AccountClassSynchronized {

    public static void main(String[] args) {

        // 계좌 클래스의 인스턴스를 생성하면서 계좌 잔액을 5000원으로 설정
        Runnable r = new Withdraw(new Account(5000));
        Thread t1 = new Thread(r, "t1");
        Thread t2 = new Thread(r, "t2");

        /* 두 스레드가 시작되면 Account의 withdraw() 메서드를 통해 balance에 접근하게
         * 되는데 이때 t1이 withdraw() 메서드를 호출하고 다음 호출 전에 t2가 withdraw()
         * 메서드를 호출하는 문제가 발생할 수 있다. 즉 하나의 스레드에서 작업하는 자원에
         * 다른 스레드가 접근해 그 자원의 값을 변경할 수 있는 문제가 발생하게 된다.
         * 그래서 하나의 스레드가 공유 자원에 접근할 때 다른 스레드가 동시에 접근할 수
         * 없도록 Lock을 걸어야 하는데 이를 스레드 동기화라고 한다.
         **/
        t1.start();
        t2.start();
    }
}

//계좌 클래스
class Account {

    // 계좌 잔액
    private int balance;

    public Account(int balance) {
        this.balance = balance;
    }

    public int getBalance() {
```

```
        return balance;
    }

    // 계좌에서 지정한 금액 만큼 인출하는 메서드
    public int withdraw(int amount) {

        /* 아래와 같이 synchronized(this) { } 블록을 사용해 메서드
         * 전체가 아닌 메서드 안의 특정 코드만 임계영역으로 설정할 수 있다.
         * () 안에는 잠금(Lock)을 할 객체의 참조변수를 지정하면 된다.
         * 이 블록 안으로 들어오면서 부터 스레드는 지정된 객체의 Lock을 얻게 되고
         * 블록을 벗어나면 Lock을 반납하게 된다. Lock의 획득과 반납은 JVM에 의해
         * 자동으로 이루어지므로 적절히 임계영역만 설정해 주면 된다.
         **/
        synchronized(this) {
            if(balance >= amount) {
                try {
                    // 현재 실행 중인 스레드를 1초 잠 재운다.
                    Thread.sleep(1000);
                }catch(InterruptedException e) {
                    e.printStackTrace();
                }
                balance -= amount;
            }
        }
        return balance;
    }
}

//계좌에 있는 돈을 인출하는 클래스
class Withdraw implements Runnable {
    Account account;

    public Withdraw(Account account) {
        this.account = account;
    }

    @Override
    public void run() {

        /* Account 클래스의 withdraw() 메서드에 synchronized로 임계영역을 설정하지
         * 않으면 t1이 출금하는 동시에 t2가 출금하는 문제가 발생해 잔액이 마이너스가
         * 되거나 잔액이 정확하게 계산되지 못하는 문제가 발생한다. 그러므로 withdraw()
         * 메서드에 synchronized 키워드로 임계영역을 설정해 t1이 출금하는 동안
         * 또 다른 스레드인 t2는 withdraw() 메서드에 접근하지 못하도록 대기시켜야 한다.
         *
```

```
         * 이번 예제는 앞의 예제와 다르게 메서드 전체를 동기화 한 것이 아니라
         * Account 클래스의 withdraw() 메서드 안에서 필요한 부분만 동기화를
         * 하기 위해서 synchronized(this) {} 블럭을 사용해 동기화 하였다.
         **/
        while(account.getBalance() > 0){

            // 한 번에 1000씩 차감
            int amount = 1000;
            System.out.println(Thread.currentThread().getName() +
                    " : " + amount + ", 잔액 : " + account.withdraw(amount));
        }
    }
}
```

▶ wait()와 notify() 메서드

앞에서 동기화를 통해 공유자원을 보호하는 부분에 대해 알아보았다.
멀티 스레드를 제어할 때 한 가지 중요한 것은 특정 스레드가 하나의 공유자원을 너무 오랫동안 점유하지 못하도록 해야 한다는 것이다. 하나의 스레드가 동기화 상태에서 공유자원을 너무 오랫동안 점유하고 있으면 그 자원을 사용하기 위해 대기하고 있는 다른 스레드가 교착상태(Deadlock)에 빠지게 된다. 그래서 하나의 스레드를 일시정지 시키고 다시 실행할 수 있도록 지원하는 메서드가 있는데 이 메서드가 바로 wait()와 notify() 메서드 이다.
임계영역의 코드를 수행하다 계속해서 작업을 진행할 상황이 아니면 wait()를 호출해 Lock을 반납하고 Wating Pool에서 대기 한다. Wating Pool에서 대기하는 동안 다른 스레드가 Lock을 받아 해당 객체에 대한 작업을 수행한다. 작업이 끝나면 notify() 메서드를 호출해 Wating Pool에서 대기중인 임의의 스레드를 깨워 실행 할 수 있도록 한다. 또한 notifyAll() 메서드를 호출해 대기 중인 모든 스레드에 통지할 수 있지만 오직 스레드 하나만 Lock을 얻어 실행될 수 있다. Wating Pool은 객체마다 각각 존재하기 때문에 notifyAll() 메서드가 호출 된다고 해서 프로세스 내의 모든 스레드에게 통지되는 것이 아니라 notifyAll() 메서드가 호출된 객체의 Wating Pool에 대기 중인 스레드에게만 통지된다.
wait(), notify(), notifyAll() 메서드는 Object 클래스에 정의되어 있으며 동기화(synchronized) 블록에서만 사용이 가능하다.

- com.javastudy.ch10.threadsync

```
public class SynchronizedFridge {
    public static void main(String[] args) {

        Fridge fridge = new Fridge();
        Mom mom = new Mom(fridge);
        Child cheolSu = new Child(fridge, "철수");
        Child younghee = new Child(fridge, "영희");

        new Thread(mom, "엄마").start();
```

```
        new Thread(cheolSu, cheolSu.getName()).start();
        new Thread(younghee, younghee.getName()).start();
    }
}

class Fridge {
    private final int MAX_COUNT= 7;
    private String[] iceCreams = {
            "파인트", "와츄원", "초코", "싱글레귤러", "와츄원",  "초코",  "와츄원" };
    private ArrayList<String> iceCreamList = new ArrayList<String>();

    public synchronized void addIceCream(String iceCreamName) {

        // 냉장고에 아이스크림이 꽉 찼으면
        if(iceCreamList.size() >= MAX_COUNT) {
            System.out.println("냉장고에 아이스크림이 꽉 찼음...");
            notifyAll();
            return;
        }

        /* 냉장고에 아이스크림이 꽉 차지 않았으면
         * 엄마 스레드를 통해 냉장고에 아이스크림을 채운다.
         **/
        iceCreamList.add(iceCreamName);
        System.out.println(Thread.currentThread().getName() + "가 "
                + iceCreamName + "을 냉장고에 넣음...");

        // Waiting Pool에 대기 중인 Child 스레드 중 하나를 깨운다.
        //notify();
        System.out.println(iceCreamList.toString());
    }

    public void  removeIceCream(String iceCreamName) {
        synchronized(this) {

            /* 냉장고에 아이스크림이 없으면
             **/
            if(iceCreamList.size() == 0) {
                try {
                    System.out.println("아이스크림 없음 : "
                            + Thread.currentThread().getName()
                            + " 스레드를 대기시킨다");

                    /* Fridge 클래스의 인스턴스는 Mom 스레드와 Child 스레드가
                     * 공유하는 객체이다. 동기화 블록에서 wait() 메서드가 호출되면
```

```
                * 동기화 블록에서 현재 작업 중인 스레드는 락(Lock)을 반납하고
                * 해당 객체의 대기실(Waiting Pool)에서 대기하게 된다.
                * notify() 메서드가 호출되어 통지할 때 까지 기다리지만 nofify()
                * 메서드가 호출되었다 하더라도 이 스레드가 락(Lock)을 얻어 다시
                * 실행될지는 전적으로 JVM에 달려있기 때문에 알 수 없다.
                *
                * Fridge 클래스는 Object로 부터 wait() 메소드를 상속 받았다.
                * 사실 자바의 모든 클래스는 Object를 직/간접적으로 상속 받기
                * 때문에 스레드 동기화 관련 메소드인 wait(), notify(),
                * notifyAll() 메서드를 상속 받게 된다.
                *
                * Child 스레드를 Waiting Pool에 대기시킨다.
                **/
                wait();
            } catch(InterruptedException e) {
                e.printStackTrace();
            }
        } // end if

        for(int i = 0; i < iceCreamList.size(); i++) {

            // 냉장고에 먹고 싶은 아이스크림이 존재하면
            if(iceCreamName.equals(iceCreamList.get(i))) {

                // 현재 Child 스레드가 꺼내먹고 아이스크림을 지운다.
                iceCreamList.remove(i);
                System.out.println(Thread.currentThread().getName()
                        + "이(가) " + iceCreamName + "을 먹음...");

                /* notify() 메서드가 호출되면 해당 객체의 대기실에 임의의
                 * 스레드 1개만 통지를 받아 실행된다. 그리고 notifyAll()
                 * 메서드가 호출되면 대기실의 모든 스레드에게 통지되지만
                 * 그 중에서 하나의 스레드만 선택되어 실행 될 수 있다.
                 * 이 또한 전적으로 JVM에 의해 결정된다.
                 *
                 * Waiting Pool에서 대기 중인 엄마 스레드를 깨운다.
                 **/
                notify();
                return;
            }
        } // end for

        try {
            System.out.println(Thread.currentThread().getName()
                    + " 스레드 대기중...");
```

```
                /* 냉장고에 원하는 아이스크림이 없는
                 * Child 스레드를 Waiting Pool에 대기시킨다.
                 **/
                wait();

            } catch(InterruptedException e) {
                e.printStackTrace();
            }

        } // end synchronized
    } // end removeIceCream

    public int getIceCreamNameCount() {
        return iceCreams.length;
    }

    public String getIceCreamName(int index) {
        return iceCreams[index];
    }
}

class Child implements Runnable {
    private Fridge fridge;
    private String name;

    public Child(Fridge fridge, String name) {
        this.fridge = fridge;
        this.name = name;
    }

    @Override
    public void run() {
        while(true) {

            int index = (int) (Math.random() * fridge.getIceCreamNameCount());
            String iceCreamName = fridge.getIceCreamName(index);

            try {
                System.out.println("Child : " + Thread.currentThread().getName()
                                   + " - " + iceCreamName);

                // 아이들은 아이스크림을 냉장고에서 꺼내 먹는다.
                fridge.removeIceCream(iceCreamName);
                Thread.sleep(1000);
```

```
            } catch(InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    public String getName() {
        return name;
    }
}

class Mom implements Runnable {
    private Fridge fridge;

    public Mom(Fridge fridge) {
        this.fridge = fridge;
    }

    @Override
    public void run() {
        while(true) {

            int index = (int) (Math.random() * fridge.getIceCreamNameCount());
            String iceCreamName = fridge.getIceCreamName(index);

            try {
                System.out.println("Mom : " + Thread.currentThread().getName()
                                   + " - " + iceCreamName);

                // 엄마는 아이스크림을 냉장고에 넣는다.
                fridge.addIceCream(iceCreamName);
                Thread.sleep(1000);
            } catch(InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

11. 네트워크 프로그래밍

우리 생활에서 다른 사람들과 관계를 형성하는 인맥이 중요하듯이 컴퓨터 프로그래밍에서도 다른 컴퓨터와 관계를 형성해 데이터를 주고받는 것이 매우 중요한 시대가 되었다. 컴퓨터 프로그래밍으로 다른 컴퓨터와 데이터를 주고받는 기능을 구현하는 것은 네트워크 프로그래밍이라 한다.
사람이 사람들과 관계를 맺고 인적 네트워크(인맥)를 이루어 살아가듯이 컴퓨터 세계에서도 네트워크를 구성해 다른 컴퓨터와 데이터를 공유하고 있다. 네트워크란 컴퓨터와 컴퓨터가 연결되어 있는 구조를 일컫는 말이며 컴퓨터 네트워크가 시작되는 초기에는 몇 안 되는 컴퓨터가 연결된 네트워크였으나 오늘날에는 전 세계 컴퓨터가 연결된 거대한 네트워크인 인터넷으로 발전하였다. 인터넷은 각 지역 또는 각 국가의 네트워크가 서로 연결되어 하나로 묶어진 거대한 네트워크 이다.
우리 생활에서 친구 또는 다른 누군가에게 전화를 걸기위해서는 전화번호를 알아야 하고 편지를 보내려면 받는 사람의 주소를 알아야 보낼 수 있듯이 인터넷에 연결된 컴퓨터끼리 데이터를 주고받기 위해서는 각 컴퓨터가 가지는 고유한 주소가 필요하다. 이 고유한 주소를 IP 주소(IP Address)라고 한다. 현재 사용되고 있는 IP 주소 체계는 IPv4이며 이 주소 체계는 아래와 같이 32bit로 구성된다. 각 자리는 0 ~ 255까지 표현되는 8bit로 구성되어 있기 때문에 32bit로 구성된다.

192.168.123.10

IP 주소는 네트워크에서 컴퓨터를 유일하게 식별할 수 있는 식별자 역할을 하며 중복해서 사용할 수 없다. 오늘날 컴퓨터 사용이 급격히 늘어나면서 IP 주소의 부족으로 128bit 체계의 IPv6가 고안되어 테스트 사용 중에 있다.
하나의 컴퓨터에는 많은 응용프로그램이 실행되고 이중에서 여러 응용프로그램이 네트워크를 사용하고 있다. 그렇기 때문에 네트워크를 통해 다른 컴퓨터의 응용프로그램과 통신하기 위해서 IP 주소만 가지고는 불가능하다. IP 주소는 네트워크에서 컴퓨터를 유일하게 구분할 수 있는 식별자 역할을 하지만 IP 주소만 가지고는 컴퓨터 안에서 실행되는 응용프로그램을 식별할 수 없기 때문에 하나의 컴퓨터 안에서 특정 응용프로그램과 연결할 수 있는 문이 필요한데 이 문을 포트(Port)라 부른다.
우리 생활에서 외부로 연결되는 관문을 포트(Port, Airport 등)라 부르듯이 컴퓨터에서도 응용프로그램이 다른 컴퓨터의 응용프로그램과 통신할 수 있는 관문을 포트라 부르고 있다.
응용프로그램이 다른 컴퓨터의 응용프로그램과 통신하기 위해서는 그 컴퓨터의 IP 주소와 응용프로그램이 사용하는 포트 번호를 알아야 비로소 통신을 할 수 있는 것이다.
포트 번호는 0 ~ 65535까지 개발자가 임의로 지정할 수 있지만 0 ~ 1023까지는 운영체제가 사용하고 있기 때문에 그 이상의 포트 번호를 사용해야 한다.
자바에서는 네트워크 프로그래밍을 지원하기 위해 java.net 패키지를 제공하고 있다. 이 패키지의 클래스를 이용하면 원격지 컴퓨터와 통신할 수 있는 응용프로그램을 보다 쉽게 작성할 수 있다. 자바 프로그래밍에서 IP 주소를 다루려면 이 패키지에서 제공하는 InetAddress 클래스를 사용하면 된다. 이 클래스를 이용하면 원격지 컴퓨터와 로컬 컴퓨터의 호스트 명이나 IP 주소를 구할 수 있다.

▶ InetAddress 클래스로 호스트 명과 IP 주소 다루기

- com.javastudy.ch11.network

```
public class InetAddressClass {
```

```
public static void main(String[] args) {

    try {

        /* 지정한 도메인 명(host 명)을 이용해 도메인에 해당하는
         * IP 주소 정보를 갖는 InetAddress 객체를 구한다.
         * InetAddress 클래스는 IP 주소를 다루기 위해 제공되는 클래스 이다.
         **/
        InetAddress inetAddr = InetAddress.getByName("www.naver.com");
        String hostName = inetAddr.getHostName();
        String ipAddr = inetAddr.getHostAddress();
        System.out.println("naver.com의 호스트 명 : " + hostName
                + ", IP 주소 : " + ipAddr);

        /* 호스트에 해당하는 IP 주소를 byte 배열로 구한다.
         * ip는 1byte의 부호 없는 정수로 0 ~ 255 사이를 표현하고 byte 형은
         * -128 ~ 127까지 표현하기 때문에 이 범위를 넘어서는 수는 음수로 표시된다.
         **/
        byte[] ips = inetAddr.getAddress();
        System.out.println("ip : " + Arrays.toString(ips));
        for(int i = 0; i < ips.length; i++) {
            System.out.print(ips[i] < 0 ? ips[i] + 256 : ips[i]);
            System.out.print(i < ips.length - 1 ? "." : "\n");
        }

        /* 지정한 도메인 명(host 명)을 이용해 도메인에 해당하는
         * 모든 IP 주소 정보를 갖는 InetAddress 객체 배열을 구한다.
         **/
        InetAddress[] inetAddrs = InetAddress.getAllByName("www.naver.com");
        System.out.println("naver.com의 모든 호스트 명과 IP 주소");
        for(InetAddress inet : inetAddrs) {
            System.out.println("호스트 명 : " + inet.getHostName()
                    + ", IP 주소 : " + inet.getHostAddress());
        }

        /* 프로그램이 실행되는 현재 컴퓨터의
         * IP 주소 정보를 갖는 InetAddress 객체를 구한다.
         **/
        InetAddress localInet  = InetAddress.getLocalHost();
        System.out.println(localInet.getHostName() + ", "
                    + localInet.getHostAddress());

        /* 컴퓨터는 자신의 호스트 명과 IP 주소를 지칭할 때 localhost와
         * 127.0.0.1을 부여하게 되는데 이를 Loopback Address라고 한다.
         **/
```

```
            System.out.println(InetAddress.getLoopbackAddress());

            /* UnknownHostException 클래스는
             * IOException 클래스를 상속해 구현한 예외처리 클래스이다.
             **/
        } catch(UnknownHostException e) {
            e.printStackTrace();
        }
    }
}
```

11.1 URL(Uniform Resource Locator)

오늘날 많은 사람들이 매일 같이 웹 서핑(Web Surfing)을 즐기고 있다. 우리가 웹 서핑을 할 때 브라우저의 주소 창에 입력하는 서버의 주소를 URL이라고 부르는데 이 URL은 인터넷에서 서비스 되는 자원을 구분하기 위해 사용되는 주소로 아래와 같은 형식으로 구성되어 있다.

프로토콜://서비스명.호스트명:포트번호/경로명/파일명?쿼리스트링
http://www.naver.com:80/search.naver?where=post&sm=tab_jum&ie=utf8

컴퓨터끼리 통신하기 위해서 프로토콜(Protocol)이라는 것을 사용하는데 이 프로토콜은 컴퓨터끼리 통신하기 위해 꼭 지켜야 하는 규칙을 정의해 놓은 것으로 통신 규약이라고도 한다.
우리는 네이버에 접속하기 위해 웹 브라우저의 주소 입력란에 “http://www.naver.com”이라고 입력한다. 이렇게 URL의 맨 앞에 “http” 라고 입력하는데 이것이 바로 HTTP(Hypertext Transfer Protocol)라는 프로토콜을 사용한 것이다.
자바에서는 URL을 다루기 위해 URL 클래스를 제공하고 있다. URL 클래스는 웹상의 자원에 대한 정보를 지정하는 클래스로 이 클래스의 객체를 사용해 인터넷으로 서비스 되는 URL에서 데이터를 읽어 올 수 있다.

▶ URL 클래스로 웹 페이지 읽어오기

- com.javastudy.ch11.network

```
public class URLClass {

    public static void main(String[] args) {

        BufferedReader reader = null;

        try {
            /* 네이버 검색 페이지는 https 프로토콜을 사용하는
             * 웹페이지로 https는 http 보다 보안이 강화된 프로토콜 이다.
```

```
 **/
URL naver = new URL("https://search.naver.com:443/");
String keyword = "코로나19";

/* 한글 등의 유니코드 문자는 웹 페이지의 주소에 사용할 수 없기 때문에
 * 주소에 사용할 수 있는 영문자, 숫자, 일부 특수 문자로 변환해야 하는데
 * 이를 URLEncoding 이라고 한다.
 **/
String encKeyword = URLEncoder.encode(keyword, "utf-8");
System.out.println("encKeyword : " + encKeyword);

// 아래와 같이 상대경로 방식으로 URL 객체를 생성할 수 있다.
URL search = new URL(naver, "search.naver?query=" + encKeyword);

System.out.println("호스트 명과 포트 : " + search.getAuthority());
System.out.println("프로토콜 : " + search.getProtocol());
System.out.println("호스트 명 : " + search.getHost());
System.out.println("포트 : " + search.getPort()
    + ", 기본포트 : " + search.getDefaultPort());
System.out.println("경로명 : " + search.getPath());
System.out.println("파일명 : " + search.getFile());
System.out.println("쿼리스트링 : " + search.getQuery());

/* URL 객체를 생성했다고 해서 원격지 컴퓨터와 연결된 것이 아니다.
 * 원격지 컴퓨터와 연결하고 실제 데이터를 읽어오려면 아래와 같이
 * openStream()을 호출해 원격지 컴퓨터와 연결하고 데이터를
 * 읽어올 스트림(통로)을 개설해야 한다.
 *
 * URL 클래스의 openStream() 메서드를 사용하는 방법은 간단하긴 하지만
 * 컨넥션 문제 등에 대한 여러가지 설정을 할 수 없어 잘 사용되지 않는다.
 **/
reader = new BufferedReader(
        new InputStreamReader(search.openStream()));

// httpbin.org는 여러 유형의 요청을 테스트 할 수 있는 사이트 이다.
//URL httpBin = new URL("https://httpbin.org/get");
//reader = new BufferedReader(
//        new InputStreamReader(httpBin.openStream()));
String line = "";
while((line = reader.readLine()) != null) {
    System.out.println(line);
}

/* MalformedURLException 클래스는
 * IOException 클래스를 상속해 구현한 예외처리 클래스이다.
```

```
             **/
        } catch(MalformedURLException e) {
            e.printStackTrace();
            System.out.println("잘못된 URL 정보 입니다.");

        } catch(IOException e) {
            e.printStackTrace();

        } finally {
            try {
                if(reader != null) reader.close();
            } catch(IOException e) { }
        }
    }
}
```

11.2 URLConnection

URLConnection 클래스는 인터넷에서 서비스 되는 URL에 접속해 데이터를 읽어오거나 원격지 URL로 데이터를 보내기 위한 클래스이다. URL 클래스와 비슷한 기능을 제공하지만 URLConnection 클래스가 더 많은 기능을 제공하기 때문에 URL 클래스 보다 효과적으로 사용할 수 있다. 웹 서버에서 데이터를 읽어 오는 방법을 GET 방식 요청이라 하고 웹 서버에 데이터 처리를 의뢰하는 방법을 POST 방식 요청이라고 한다. GET 방식 요청은 우리가 웹 브라우저를 사용해 웹 서핑을 할 때 주소 입력란에 주소를 입력해 이동하거나 웹 페이지에서 클릭하여 다른 페이지로 이동하는 동작이 GET 방식 요청이다. 이 GET 방식 요청은 서버로 어떤 요청을 보낼 때 추가적인 데이터를 URL 주소 뒤에 붙여서 보내는 방식이다. 이 때 URL 뒤에 덧붙여 보내는 데이터를 쿼리 스트링이라 부른다. 반면 로그인 처리와 같은 폼(Form) 데이터 전송, 파일 업로드를 통한 파일 전송 등은 POST 방식 요청으로 구현된다. POST 방식 요청을 할 때 서버로 보내지는 데이터는 요청 본문을 통해 전달된다.
URL 객체와 URLConnection 객체는 모두 GET 방식 요청을 할 수 있는 객체이다. 하지만 POST 방식 요청은 URL 객체에서는 지원하지 않고 URLConnnection 객체만 지원하고 있다.

▶ URLConnection 클래스를 이용해 네이버 검색 요청하기

- com.javastudy.ch11.network

```
public class URLConnection01 {

    public static void main(String[] args) {

        BufferedReader reader = null;

        try {
            /* 네이버 검색 페이지는 https 프로토콜을 사용하는
```

```
         * 웹페이지로 https는 http 보다 보안이 강화된 프로토콜 이다.
         **/
        URL naver = new URL("https://search.naver.com:443/");
        String keyword = "코로나19";

        /* 한글 등의 유니코드 문자는 웹 페이지의 주소에 사용할 수 없기 때문에
         * 주소에 사용할 수 있는 영문자, 숫자, 일부 특수 문자로 변환해야 하는데
         * 이를 URLEncoding 이라고 한다.
         **/
        String encKeyword = URLEncoder.encode(keyword, "utf-8");
        System.out.println("encKeyword : " + encKeyword);

        // 아래와 같이 상대경로 방식으로 URL 객체를 생성할 수 있다.
        URL url = new URL(naver, "search.naver?query=" + encKeyword);

        /* URL 클래스의 openConnection() 메서드를 이용해 URLConnection
         * 클래스의 자식 클래스인 HttpURLConnection 클래스 타입의 객체를 구한다.
         *
         * HttpURLConnection 클래스를 사용하면 다양한 옵션을 설정할 수 있다.
         * 아래와 같이 요청 방식을 GET 또눈 POST로 설정할 수도 있고, Timeout
         * 설정과 UserAgent 등을 지정해 보다 세밀하게 구현할 수 있다는 장점이 있다.
         **/
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();

        conn.setRequestMethod("GET");
        conn.setReadTimeout(1000);
        conn.connect();

        /* 서버의 응답을 읽어오기 위해 스트림을 생성하고 있다.
         * 서버와 연결된 커넥션 객체로부터 InputStream 객체를 구해 바이트
         * 스트림을 문자 스트림으로 변환해 주는 InputStreamReader의
         * 생성자 인수로 지정해 InputStreamReader 객체를 생성한 후
         * 이 객체를 BufferedReader의 생성자 인수로 지정하고 있다.
         **/
        reader = new BufferedReader(
                new InputStreamReader(conn.getInputStream(), "UTF-8"));

        String line = "";
        while((line = reader.readLine()) != null) {
            System.out.println(line);
        }
    } catch(IOException e) {
        e.printStackTrace();

    } finally {
```

```
            try {
                if(reader != null) reader.close();
            } catch(IOException e) { }
        }
    }
}
```

▶ URLConnection 클래스를 이용해 POST 방식 로그인 요청하기

- com.javastudy.ch11.network

```
public class URLConnection02 {

    public static void main(String[] args) {

        OutputStreamWriter writer = null;
        BufferedReader reader = null;

        try {

            /* 접속할 URL 객체를 생성하고 openConnection() 메서드를
             * 이용해 URLConnection 객체를 구한다.
             **/
            //URL url = new URL("http://localhost:8080/JavaNetworkTest/postTestJson.jsp");
            URL url = new URL("http://localhost:8080/JavaNetworkTest/postTestHtml.jsp");
            URLConnection conn = url.openConnection();

            /* 서버로 데이터를 보내기 위해 setDoOutput(true) 메서드를 호출했다.
             * 이 메서드에 true를 지정하면 요청이 GET에서 POST 방식으로 변경된다.
             **/
            conn.setDoOutput(true);

            /* 서버로 데이터를 보내기 위해 서버와 연결된 커넥션 객체로부터
             * OutputStream 객체를 구해 바이트 스트림을 문자 스트림으로 변환해
             * 주는 OutputStreamWriter의 생성자 인수로 지정하고 있다.
             **/
            writer = new OutputStreamWriter(conn.getOutputStream());

            // 서버로 보낼 데이터를 스트림에 쓴다.
            writer.write("id=java&pass=1234");

            /* 스트림을 통해 데이터를 서버로 흘려보낸다.
             * 명시적으로 flush() 메서드나 close() 메서드를 호출해야 스트림에 연결된
             * 서버로 데이터가 전달된다. 그렇지 않으면 서버에서 데이터를 받지 못한다.
             **/
```

```
            writer.flush();

            /* 서버의 응답을 읽어오기 위해 스트림을 생성하고 있다.
             * 서버와 연결된 커넥션 객체로부터 InputStream 객체를 구해 바이트
             * 스트림을 문자 스트림으로 변환해 주는 InputStreamReader의
             * 생성자 인수로 지정해 InputStreamReader 객체를 생성한 후
             * 이 객체를 BufferedReader의 생성자 인수로 지정하고 있다.
             **/
            reader = new BufferedReader(
                    new InputStreamReader(conn.getInputStream()));
            String line = "";
            while((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch(IOException e) {
            e.printStackTrace();

        } finally {
            try {
                if(writer != null) writer.close();
                if(reader != null) reader.close();
            } catch(IOException e) { }
        }
    }
}
```

아래는 위의 URLConnection 클래스를 이용해 POST 방식 요청을 하고 응답을 받기 위한 JSP 코드이다. 먼저 Eclipse Java EE를 설치하고 JavaNetworkClass.war 프로젝트를 생성하고 아래 코드를 작성하여 테스트 해 보자.

- WebContent/loginForm.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <form action="postTestHtml.jsp" method="post">
        아이디 : <input type="text" name="id" /><br/>
        비밀번호 : <input type="password" name="pass" /><br/>
        <input type="submit" value="로그인" />
    </form>
```

```
</body>
</html>
```

- WebContent/postTestHtml.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
    request.setCharacterEncoding("utf-8");
    String id = request.getParameter("id");
    String pass = request.getParameter("pass");
    String contentType = request.getContentType();
    String queryString = request.getQueryString();
    String message = "";

    System.out.println("method : " + request.getMethod());
    System.out.println("id : " + id + ", pass : " + pass);

    if(id.equals("java") && pass.equals("1234")) {
        message =  "로그인 성공";
    } else {
        if(! id.equals("java")) {
            message = "로그인 실패 - 존재하지 않는 아이디";

        } else if(! pass.equals("1234")) {
            message = "로그인 실패 - 비밀번호가 틀림";
        }
    }
%>
<html>
<head>
    <title>post 방식의 요청 처리</title>
</head>
<body>
    <h3>post 방식의 요청 처리</h3>
    contentType : <%= contentType %><br/>
    queryString : <%= queryString %><br/>
    id : <%= id %><br/>
    pass : <%= pass %><br/>
    message : <%= message %>
</body>
</html>
```

11.3 소켓(Socket) 프로그래밍

인터넷을 통해 전송되는 데이터를 데이터그램(Datagram)이라 부른다. 데이터그램은 일정한 크기의 패킷(패킷이란 네트워크를 통해 전송하기 쉽도록 자른 데이터의 전송단위)으로 나누어 전송된다. 데이터그램에는 헤더(Header)와 페이로드(Payload)를 포함하고 있는데 헤더는 데이터를 수신할 목적지의 주소와 포트, 패킷을 보낸 발신지의 주소와 포트, 데이터 손상을 탐지하기 위한 체크섬과 신뢰할 수 있는 데이터 전송을 위해 필요한 정보들을 가지고 있으며 페이로드에는 데이터 자체가 들어있다. 데이터그램은 길이가 제한되어 있기 때문에 데이터를 보내는 송신측에서 데이터를 일정 크기(패킷 단위)로 잘라서 송신하게 되고 이를 받는 수신측에서 데이터를 받아 재조립해야 한다. 또한 전송 중에 패킷의 손상이 생기면 재전송이 필요하거나 패킷이 송신측에서 보낸 순서와 다르게 도착하게 되면 수신측에서 데이터의 재 정렬이 필요하다. 이러한 모든 경우를 처리하기 위해 프로그래머는 많은 노력과 복잡한 코드를 구현해야 한다. 하지만 소켓을 사용하게 되면 이런 일련의 복잡한 작업을 하지 않아도 네트워크 프로그래밍을 할 수 있다. 소켓은 바이트 단위로 데이터를 읽고 쓰는 다른 스트림과 같이 네트워크 통신을 처리할 수 있도록 해 준다. 소켓은 패킷 크기와 분할, 에러 탐지, 패킷 재전송 등과 같은 세부적인 네트워크 작업을 내부적으로 감싸고 있다.

소켓(Socket)이란 네트워크를 통해 통신하는 컴퓨터 간의 양방향 통신 연결의 한쪽 끝단(Endpoint)을 의미하며, 소켓은 기본적으로 TCP/IP 프로토콜 기반에서 동작하도록 설계되어 있다.

TCP/IP 프로토콜은 인터넷의 핵심 프로토콜로 인터넷에 컴퓨터를 연결하고 데이터를 주고받을 수 이도록 하는데 필요한 통신 프로토콜(TCP, IP, UDP, ICMP, ARP, RARP 등)들의 집합이다. TCP 프로토콜은 Transmission Control Protocol의 약자로 다른 두 시스템 간에 신뢰성 있는 데이터 전송을 다루기 위한 통신 프로토콜로 IP(Internet Protocol) 프로토콜 위에서 동작한다. IP 프로토콜은 패킷 교환 네트워크에서 송신 호스트와 수신 호스트가 데이터를 주고받는 것을 다루기 위한 프로토콜이다. TCP/IP 프로토콜 기반에서 동작하는 대표적인 응용프로토콜로는 HTTP, FTP 등이 있다.

자바에서는 소켓 프로그래밍을 위해 java.net 패키지를 제공하고 있으며 이 패키지에는 여러 가지 프로토콜을 이용해 소켓 통신을 할 수 있는 클래스가 구현되어 있다. 이들 중에서 우리는 TCP와 UDP 소켓 통신을 지원하는 클래스에 대해서 알아볼 것이다.

TCP와 UDP는 TCP/IP 프로토콜의 4 계층(네트워크 계층, 인터넷 계층, 전송 계층, 응용 계층) 중에서 전송 계층에 속하는 프로토콜이다.

TCP(Transmisson Control Protocol, 전송 제어 프로토콜)는 데이터를 전송하기 전에 상대방과 먼저 연결 한 후 데이터를 전송하기 때문에 연결지향 프로토콜이라고 한다. 또한 데이터 전송 중에 손실되거나 손상된 데이터를 재전송 하고 데이터의 순서를 보장하기 때문에 신뢰성 있는 프로토콜이라고도 부른다.

UDP(User Datagram Protoco, 사용자 데이터그램 프로토콜)는 상대방과 연결을 맺지 않은 상태에서 데이터를 전송하기 때문에 비 연결지향 프로토콜이라고 하며 수신 측에서 패킷의 손상을 체크할 수 있지만 송신 측에서는 데이터가 올 바르게 전송되었는지 체크하지 않으며 수신되는 패킷의 순서를 보장하지 않기 때문에 신뢰성 없는 프로토콜이라고도 부른다.

TCP와 UDP 통신은 위에서 설명한 것과 같이 전송 방식이 서로 다르기 때문에 각 방식에 따라서 서로 다른 장단점을 가지고 있다. 그렇기 때문에 응용프로그램을 개발할 때 응용프로그램의 특징에 따라서 적절한 통신 방식을 선택해 구현해야 한다.

11.3.1 TCP 소켓 프로그래밍

앞에서도 언급한 것과 같이 TCP 소켓 통신은 클라이언트와 서버가 연결된 후 일대일로 통신이 이루어지며 패킷의 손상이 발생하면 재전송 하는 방식으로 통신하기 때문에 신뢰성 있는 데이터의 전송에 사용된다.

일반적으로 네트워크를 통해 요청을 받아 정보를 제공하는 쪽을 서버라 하며 정보를 요청해 받아가는 쪽을 클라이언트라 부른다. 이들은 클라이언트에서 서버에 접속해 정보를 요청하고 서버는 요청을 처리한 결과를 클라이언트에게 응답하는 방식으로 동작한다.
자바에서는 TCP 소켓 프로그래밍을 지원하기 위해 클라이언트 소켓을 구현한 Socket 클래스와 서버 소켓을 구현한 ServerSocket 클래스를 제공하고 있다.
클라이언트는 Socket 클래스를 이용해 서버의 IP와 포트를 지정한 후 소켓 객체를 생성하고 서버와 접속을 시도한다. 서버는 ServerSocket 클래스를 이용해 클라이언트가 접속할 포트를 지정해 소켓 객체를 생성하고 이 포트로 들어오는 클라이언트 요청을 기다린다. 서버는 클라이언트가 접속되면 accept() 메서드에서 클라이언트와 통신할 소켓을 따로 생성해 접속된 클라이언트와 통신하게 된다. ServerSocket은 클라이언트와 통신하는 목적으로 사용되는 것이 아니라 클라이언트의 접속을 받아들이는 목적으로만 사용된다. 클라이언트가 접속할 때 마다 accept() 메서드에서 클라이언트와 통신할 소켓을 별도로 생성해 통신하게 된다.

▶ Socket 클래스로 TCP 클라이언트 구현하기

- com.javastudy.ch11.tcp

```
public class TCPClient {

    public static void main(String[] args) {

        Socket client = null;

        try {
            // 1. 접속할 서버의 주소와 포트 번호를 지정해 소켓 객체를 생성한다.
            /* 아래와 같이 소켓을 생성하면 지정한 서버로 바로 접속을 시도한다.
             * 이 때 서버가 대기 중이 아니면 ConnectException이 발생한다.
             **/
            client = new Socket("localhost", 9999);
            System.out.println("서버로 접속 요청함...");

            // 2. 서버와 네트워크 입출력을 위한 스트림을 생성한다.
            /* 소켓이 생성되고 서버와 접속이 이루어지면 소켓 객체의 getInputStream()
             * 메서드를 이용해 서버로부터 전송되는 데이터를 받을 입력 스트림을 개설하고
             * getOutputStream() 메서드를 이용해 서버로 데이터를 보낼 출력 스트림을
             * 개설할 수 있다. 이 처럼 Socket 클래스를 이용하면 복잡한 네트워크
             * 프로그래밍을 스트림을 사용하는 것과 동일하게 처리할 수 있다.
             **/
            BufferedReader reader = new BufferedReader(
                    new InputStreamReader(client.getInputStream()));
            BufferedWriter writer = new BufferedWriter(
                    new OutputStreamWriter(client.getOutputStream()));

            // 3. 서버로 보낼 요청 데이터를 스트림에 쓰고 서버로 전송한다.
            /* flush() 메서드는 스트림을 통해 데이터를 서버로 흘려보낸다.
```

```
             * 명시적으로 flush() 메서드나 close() 메서드를 호출해야 서버로
             * 데이터가 전달된다. 그렇지 않으면 서버는 데이터를 받지 못한다.
             **/
            writer.write("Hello Server\r\n");
            writer.flush();

            // 4. 서버로부터 수신된 데이터를 읽어 콘솔에 출력한다.
            String line = reader.readLine();
            System.out.println("서버의 응답 데이터 : " + line);

        } catch(UnknownHostException e) {
            e.printStackTrace();
            System.out.println("알수 없는 호스트 입니다.");

        } catch(IOException e) {
            e.printStackTrace();

        } finally {
            try {
                // 5. 소켓 통신이 완료되면 소켓을 닫는다.
                // 소켓을 닫으면 소켓의 입출력 스트림도 같이 닫힌다.
                if(client != null) client.close();
            } catch(IOException e) {}
        }
    }
}
```

▶ ServerSocket 클래스로 TCP 서버 구현하기

- com.javastudy.ch11.tcp

```
public class TCPServer {

    public static void main(String[] args) {

        ServerSocket server = null;
        Socket client = null;
        try {
            // 1. 클라이언트가 연결될 포트를 지정해 서버 소켓을 생성한다.
            server = new ServerSocket(9999);
            System.out.println("클라이언트 요청 대기중...");

            // 2. 클라이언트 연결 요청을 기다린다.
            /* accept() 메서드는 클라이언트의 연결이 올 때까지 계속 기다린다.
             * 클라이언트 요청이 들어오면 연결을 수락하고 클라이언트와 통신할
```

```
            * 새로운 Socket 객체를 생성해 반환한다. 이때 새로 생성되는
            * Socket 객체의 포트 번호는 7777번이 아니고 자동으로 할당된다.
            **/
            client = server.accept();
            System.out.println("클라이언트 접속 수락...");

            // 3. 클라이언트와 네트워크 입출력을 위한 스트림을 생성한다.
            /* 클라이언트와 연결된 소켓이 생성되면 소켓 객체의 getInputStream()
            * 메서드를 이용해 클라이언트의 요청 데이터를 받을 입력 스트림을 개설하고
            * getOutputStream() 메서드를 이용해 클라이언트로 응답할 출력 스트림을
            * 개설할 수 있다. 이 처럼 Socket 클래스를 이용하면 복잡한 네트워크
            * 프로그래밍을 스트림을 사용하는 것과 동일하게 처리할 수 있다.
            **/
            BufferedReader request = new BufferedReader(
                    new InputStreamReader(client.getInputStream()));
            BufferedWriter response = new BufferedWriter(
                    new OutputStreamWriter(client.getOutputStream()));

            // 4. 클라이언트로부터 들어온 요청 데이터를 읽어온다.
            String line = request.readLine();
            System.out.println("클라이언트의 요청 데이터 : " + line);

            // 5. 클라이언트로 보낼 응답 데이터를 스트림에 쓰고 클라이언트로 전송한다.

            /* flush() 메서드는 스트림을 통해 데이터를 서버로 흘려보낸다.
            * 명시적으로 flush() 메서드나 close() 메서드를 호출해야 클라이언트로
            * 데이터가 전달된다. 그렇지 않으면 클라이언트는 데이터를 받지 못한다.
            **/
            response.write("Hi Client\r\n");
            response.flush();

        } catch(IOException e) {
            e.printStackTrace();

        } finally {
            try {
                // 5. 소켓 통신이 완료되면 소켓을 닫는다.
                /* 소켓을 닫으면 소켓의 입출력 스트림도 같이 닫힌다.
                * 클라이언트와 통신한 통신 소켓을 먼저 닫고 서버 소켓을 닫는다.
                **/
                if(client != null) client.close();
                if(server != null) server.close();
            } catch(IOException e) {}
        }
    }
```

}

11.3.2 UDP 소켓 프로그래밍

UDP 통신은 상대방과 연결을 맺지 않은 상태에서 데이터를 전송하는 비 연결지향 통신이기 때문에 데이터를 전송할 때 정확히 도착했는지 확인할 방법이 없으며 보낸 순서대로 데이터가 제대로 도착했는지 보장되지 않는다. 이렇게 신뢰성은 보장되지 않지만 TCP 통신에 비해 속도가 훨씬 빠르다. TCP 통신과 UDP 통신을 비교할 때 전화와 우편으로 많이 비유하고 있다. 사실 우편은 전화보다 빠르지는 않지만 전화보다 신뢰성이 떨어지고 UDP와 비슷한 측면이 있다.
자바에서는 UDP 소켓 프로그래밍을 지원하기 위해 UDP 통신에서 사용하는 소켓을 구현한 DatagramSocket 클래스와 데이터를 담아 보낼 패킷을 구현한 DatagramPacket 클래스를 제공하고 있다. DatagramSocket은 UDP 데이터그램을 송수신 하는 기능을 제공하고 Datagram Packet은 송수신 하는 데이터를 담는 그릇 역할을 하는 클래스이다. 데이터를 담는 Datagram Packet은 헤더와 데이터로 구성되며 헤더에는 데이터를 받을 호스트의 정보가 저장된다.
UDP 클라이언트와 서버는 모두 DatagramSocket 클래스와 DatagramPacket 클래스를 사용해 구현하면 된다. UDP 송신 측에서는 데이터를 전송하기 위해서 DatagramPacket 객체에 데이터를 담아 DatagramSocket 객체를 사용해 패킷을 전송하고 UDP 수신 측에서는 Datagram Socket 객체에서 DatagramPacket 객체를 가져와 패킷의 내용을 읽어 들인다.

▶ DatagramSocket과 DatagramPacket 클래스로 UDP 클라이언트 구현하기

- com.javastudy.ch11.udp

```
public class UDPClient {

    public static void main(String[] args) {

        DatagramSocket socket = null;

        try {
            /* 1. UDP 통신을 통해 데이터를 송수신할 DatagramSocket 객체를
             * 생성한다. 기본 생성자를 호출하거나 생성자의 인수로 0을 전달하면
             * 자바가 사용가능한 임의의 포트를 선택해 객체를 생성한다.
             **/
            socket = new DatagramSocket();
            //socket = new DatagramSocket(8888);

            // 2. 호스트명을 이용해 서버의 IP 주소를 구한다.
            InetAddress serverAddr = InetAddress.getByName("localhost");
            System.out.println("server IP : " + serverAddr.getHostAddress());

            // 3. 송수신 데이터가 저장될 byte 배열을 생성한다.
            byte[] inBuf = new byte[1024];
            byte[] outBuf = "안녕 UDP 서버".getBytes();
```

```
            // 4. 서버로 전송할 데이터를 담을 DatagramPacket 객체를 생성한다.
            /* DatagramPacket 클래스의 생성자 중에서 InetAddress나 SocketAddress
             * 타입의 파라미터를 가진 생성자는 네트워크를 통해 데이터를 보내기
             * 위한 객체를 생성할 때 사용하는 생성자 이다.
             * UDP 통신을 사용하는 대부분의 소프트웨어는 데이터그램당 8,192Byte를
             * 초과하지 않는다. 이론적으로 IPv4의 데이터그램은 65,507Byte로 제안하고
             * 있으며 DatagramPacket 객체는 이 크기까지 데이터를 손실없이 받을 수
             * 있다. 하지만 많은 운영체제에서 8KByte 이상의 UDP 데이터그램을 지원하지
             * 않으며 이 보다 클 경우 패킷을 잘라 버린다. 이런 경우 패킷이 버려진 것을
             * 통보 받을 수 없기 때문에 UDP 통신 프로그램을 작성할 경우에는 8KByte
             * 보다 큰 DatagramPacket 객체를 생성하지 않도록 주의해야 한다.
             **/
            DatagramPacket outPacket =
                    new DatagramPacket(outBuf, outBuf.length, serverAddr, 8888);

            System.out.println("서버 IP : " + outPacket.getAddress().getHostAddress()
                    + ", 서버 포트 : " + outPacket.getPort());

            // 5. 서버의 응답 데이터를 담을 DatagramPacket 객체를 생성한다.
            /* DatagramPacket 클래스의 생성자 중에서 byte 배열과 int 형으로
             * 구성된 파라미터를 가진 생성자는 네트워크로부터 데이터를 받기 위한
             * 객체를 생성할 때 사용하는 생성자 이다.
             **/
            DatagramPacket inPacket = new DatagramPacket(inBuf, inBuf.length);

            // 6. 송수신할 준비가 끝나면 소켓을 통해 데이터를 보내고 응답을 받는다.
            /* 소켓의 연결 타임아웃을 설정한다. 지정한 시간동안 연결이
             * 되지 않으면 SocketTimeoutException이 발생한다.
             **/
            socket.setSoTimeout(10000);
            socket.send(outPacket);
            socket.receive(inPacket);

            // 7. 서버로부터 응답 받은 데이터를 DatagramPacket에서 읽는다.
            String data = new String(inPacket.getData());
            System.out.println("서버에서 받은 응답 : " + data);

        } catch(SocketException e) {
            e.printStackTrace();

        } catch(UnknownHostException e) {
            e.printStackTrace();

        } catch(IOException e) {
            e.printStackTrace();
```

```
        } finally {
            // 8. 모든 작업이 끝나면 소켓을 닫는다.
            if(socket != null) socket.close();
        }
    }
}
```

▶ DatagramSocket과 DatagramPacket 클래스로 UDP 서버 구현하기

- com.javastudy.ch11.udp

```
public class UDPServer {

    public static void main(String[] args) {

        DatagramSocket socket = null;

        try {
            /* 1. UDP 통신을 통해 데이터를 송수신할 DatagramSocket 객체를
             * 생성한다. 기본 생성자를 호출하거나 생성자의 인수로 0을 전달하면
             * 자바가 사용가능한 임의의 포트를 선택해 객체를 생성한다.
             **/
            socket = new DatagramSocket(8888);
            //socket = new DatagramSocket(8888);

            // 2. 송수신 데이터가 저장될 byte 배열을 생성한다.
            byte[] requestBuf = new byte[1024];
            byte[] responseBuf = "안녕 UDP 클라이언트".getBytes();

            // 3. 클라이언트이 요청 데이터를 담을 DatagramPacket 객체를 생성한다.
            /* DatagramPacket 클래스의 생성자 중에서 byte 배열과 int 형으로
             * 구성된 파라미터를 가진 생성자는 네트워크로부터 데이터를 받기 위한
             * 객체를 생성할 때 사용하는 생성자 이다.
             **/
            DatagramPacket request =
                    new DatagramPacket(requestBuf, requestBuf.length);

            // 4. 클라이언트의 요청 데이터를 받는다.
            /* UDP 클라이언트와 다르게 먼저 클라이언트의 요청 데이터를 수신하고
             * 응답 데이터를 작성해 데이터를 클라이언트로 전송하면 된다.
             **/
            socket.receive(request);

            // 5. 클라이언트로부터 요청 받은 데이터를 DatagramPacket에서 읽는다.
```

```
            String data = new String(request.getData());
            System.out.println("클라이언트에서 받은 요청 : " + data);

            /* 위에서 출력 결과를 보면 클라이언트 요청으로 받은 실제 데이터 보다
             * 수신 DatagramPacket 객체의 버퍼가 크기 때문에 데이터 뒤 쪽에
             * 모두 0으로 채워져 받게 된다. 0을 다시 String으로 변화하는 과정에서
             * 유니코드 첫 번째 문자('\u0000')로 변환되어 출력된다.
             * 실제 클라이언트의 요청 데이터만 출력하기 위해 아래와 같이 처리했다.
             **/
            int len = request.getLength();
            byte[] result = new byte[len];
            System.arraycopy(request.getData(), 0, result, 0, len);
            System.out.println(new String(result));

            /* 6. 응답 데이터를 보내기 위해 클라이언트의 요청 데이터를 담고 있는
             * DatagramPacket 객체로 부터 클라이언트의 IP 주소와 포트를 구한다.
             **/
            InetAddress clientAddr = request.getAddress();
            int port = request.getPort();

            // 7. 클라이언트로 전송할 데이터를 담을 DatagramPacket 객체를 생성한다.
            /* DatagramPacket 클래스의 생성자 중에서 InetAddress나 SocketAddress
             * 타입의 파라미터를 가진 생성자는 네트워크를 통해 데이터를 보내기
             * 위한 객체를 생성할 때 사용하는 생성자 이다.
             * UDP 통신을 사용하는 대부분의 소프트웨어는 데이터그램당 8,192Byte를
             * 초과하지 않는다. 이론적으로 IPv4의 데이터그램은 65,507Byte로 제안하고
             * 있으며 DatagramPacket 객체는 이 크기까지 데이터를 손실없이 받을 수
             * 있다. 하지만 많은 운영체제에서 8KByte 이상의 UDP 데이터그램을 지원하지
             * 않으며 이 보다 클 경우 패킷을 잘라 버린다. 이런 경우 패킷이 버려진 것을
             * 통보 받을 수 없기 때문에 UDP 통신 프로그램을 작성할 경우에는 8KByte
             * 보다 큰 DatagramPacket 객체를 생성하지 않도록 주의해야 한다.
             **/
            DatagramPacket response =
                    new DatagramPacket(responseBuf,
                            responseBuf.length, clientAddr, port);

            // 8. 클라이언트로 응답 데이터를 전송한다.
            socket.send(response);

        } catch(SocketException e) {
            e.printStackTrace();

        } catch(UnknownHostException e) {
            e.printStackTrace();
```

```
        } catch(IOException e) {
            e.printStackTrace();

        } finally {
            // 9. 모든 작업이 끝나면 소켓을 닫는다.
            if(socket != null) socket.close();
        }
    }
}
```

11.3.3 스레드를 활용한 TCP 채팅 프로그램 구현하기

▶ TCP 메시지를 전송하는 스레드

- com.javastudy.ch11.chat

```
public class SendMessage extends Thread {

    Socket socket;
    String name;

    public SendMessage(Socket socket, String name) {
        this.socket = socket;
        this.name = name;
    }

    @Override
    public void run() {

        try {
            // 키보드 입력을 받기 위해 System 클래스의 입력 스트림을 사용했다.
            BufferedReader reader = new BufferedReader(
                    new InputStreamReader(System.in));

            /* 소켓 객체의 getOutputStream() 메서드를 이용해
             * 소켓에 연결된 출력 스트림을 구할 수 있다.
             **/
            BufferedWriter writer = new BufferedWriter(
                    new OutputStreamWriter(socket.getOutputStream()));

            while(true) {

                // 키보드로부터 입력된 데이터를 읽어 들인다.
                String msg = reader.readLine();
                Thread.sleep(10);
```

```
                // 키보드 입력이 "bye"라면 스레드를 종료한다.
                if(msg.equals("bye")) {
                    System.out.println("Send - 채팅을 종료합니다.");
                    break;
                }

                /* 키보드로 입력된 데이터를 스트림으로 출력하고
                 * 원격지 컴퓨터로 흘려보낸다.
                 **/
                writer.write("[" + name + "] : " + msg + "\r\n");
                writer.flush();
            }
        } catch(IOException e) {
            e.printStackTrace();

        } catch(InterruptedException e) {
            System.out.println("Interrupted - 채팅을 종료합니다.");
            return;

        }catch(Exception e) {
            System.out.println("excepton 발생");
            e.printStackTrace();

        } finally {
            try {
                // 소켓을 닫으면 스트림도 닫힌다.
                if(socket != null) socket.close();
            } catch(IOException e) { }
        }
    }
}
```

▶ TCP 메시지를 수신하는 스레드

- com.javastudy.ch11.chat

```
public class ReceiveMessage extends Thread {

    Socket socket;
    Thread t;

    public ReceiveMessage(Socket socket, Thread t) {
        this.socket = socket;
        this.t = t;
```

```
}

@Override
public void run() {

    try {
        /* 소켓 객체의 getInputStream() 메서드를 이용해
         * 소켓에 연결된 입력 스트림을 구할 수 있다.
         **/
        BufferedReader reader = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));

        while(true) {

            /* 원격지 컴퓨터로부터 수신된 데이터를 읽어 들인다.
             * 현재 소켓에 연결된 원격지 컴퓨터의 소켓이 강제 종료되면
             * SocketException이 발생한다. 프로그램을 강제로 종료하면
             * 소켓이 강제로 닫히기 때문에 SocketException이 발생한다.
             * 원격지 컴퓨터에서 소켓을 정상적으로 닫으면 null을 받는다.
             **/
            String msg = reader.readLine();

            /* 현재 소켓에 연결된 원격지 컴퓨터의 소켓이 정상적으로 닫히면
             * readLine() 메서드로 읽어온 데이터는 null이 된다.
             * 이 때 메시지 수신 스레드만 종료하게 되면 메시지 송신 스레드는
             * 종료되지 않기 때문에 프로그램은 종료되지 않는다.
             * 그래서 메시지 송신 스레드에 강제로 인터럽트를 발생시켜
             * InterruptedException을 처리하려 했으나 송신 스레드는
             * reader.readLine() 메서드를 호출해 I/O 작업 상태이기
             * 때문에 JVM에 의해 BLOCKED된 상태이다. I/O 작업이
             * 완료될 때까지 대기 상태로 있다가 I/O 작업이 완료되면 다시
             * RUNNABLE 상태가 된다. BLOCKED된 스레드는 interrupt()
             * 메서드를 호출해도 InterruptedException이 발생하지
             * 않기 때문에 프로그램을 종료하는 System.exit(0)를 호출해
             * 응용프로그램을 종료했다.
             **/
            if(msg == null) {
                System.out.println("Receive - 채팅을 종료합니다.");
                t.interrupt();
                System.exit(0);
            }
            System.out.println(msg);
        }
    } catch(SocketException e) {
        e.printStackTrace();
```

```
            System.out.println("SecketException - 채팅을 종료 합니다.");
            System.exit(0);

        } catch(IOException e) {
            e.printStackTrace();
            System.out.println(e.getMessage());

        } catch(Exception e) {
            e.printStackTrace();

        } finally {
            try {
                // 소켓을 닫으면 스트림도 닫힌다.
                if(socket != null) socket.close();
            } catch(IOException e) { }
        }
    }
}
```

▶ 채팅 클라이언트

- com.javastudy.ch11.chat

```
public class ChatClient {

    @SuppressWarnings("resource")
    public static void main(String[] args) {

        Socket socket = null;
        Scanner scan = null;

        try {
            // 사용자 입력을 받을 Scanner 클래스의 인스턴스 생성
            scan = new Scanner(System.in);
            System.out.print("이름을 입력해 주세요 : ");
            String name =  scan.nextLine();

            /* 접속할 서버의 주소와 포트 번호를 지정해 소켓 객체를 생성한다.
             * 아래와 같이 소켓을 생성하면 지정한 서버로 바로 접속을 시도한다.
             * 이 때 서버가 대기 중이 아니면 ConnectException이 발생한다.
             **/
            socket = new Socket("127.0.0.1", 9999);

            // 메시지 송수신 스레드 객체를 생성한다.
            SendMessage send = new SendMessage(socket, name);
```

```
            ReceiveMessage receive = new ReceiveMessage(socket, send);

            // 메시지 송수신 스레드를 시작한다.
            send.start();
            receive.start();

            /*
            // 메시지 수신 스레드가 종료될 때까지 기다린다.
            receive.join();
            if(socket.isClosed()) {
                System.exit(0);
            }
            */

        } catch(IOException e) {
            e.printStackTrace();

        /*
        } catch(InterruptedException e) {
            e.printStackTrace();
        */
        } finally {
            // Scanner 자원을 해제 한다.
            //if(scan != null) scan.close();

            try {
                /* 소켓 통신이 완료되면 클라이언트 소켓을 닫는다.
                 * 소켓을 닫으면 소켓에서 사용한 입출력 스트림도 같이 닫힌다.
                 **/
                if(socket != null) socket.close();
            } catch(IOException e) { }
        }
        System.out.println("ChatClient main() 종료");
    }
}
```

▶ 채팅 서버

- com.javastudy.ch11.chat

```
public class ChatServer {

    public static void main(String[] args) {

        ServerSocket server = null;
```

```
        Socket client = null;

        try {
            // 클라이언트가 연결될 포트를 지정해 서버 소켓을 생성한다.
            server = new ServerSocket(9999);
            System.out.println("채팅 서버를 시작합니다.");

            /* 클라이언트 연결 요청을 기다린다.
             * accept() 메서드는 클라이언트의 연결이 올 때까지 계속 기다린다.
             * 클라이언트 요청이 들어오면 연결을 수락하고 클라이언트와 통신할
             * 새로운 Socket 객체를 생성해 반환한다. 이때 새로 생성되는
             * Socket 객체의 포트 번호는 9999번이 아니고 자동으로 할당된다.
             **/
            client = server.accept();
            System.out.println("클라이언트가 채팅방에 입장했습니다.");

            // 메시지 송수신 스레드 객체를 생성한다.
            SendMessage send = new SendMessage(client, "서버");
            ReceiveMessage receive = new ReceiveMessage(client, send);

            // 메시지 송수신 스레드를 시작한다.
            send.start();
            receive.start();

        } catch(IOException e) {
            e.printStackTrace();

        } finally {
            try {
                /* 소켓 통신이 완료되면 서버 소켓을 닫는다.
                 * 소켓을 닫으면 소켓에서 사용한 입출력 스트림도 같이 닫힌다.
                 **/
                if(server != null) server.close();
            } catch(IOException e) { }
        }
        System.out.println("ChatServer main() 종료");
    }
}
```