자바의 장점

1. 운영체제에 독립적이다.
이것은 일종의 에뮬레이터인 자바가상머신(JVM)을 통해서 가능하다.
자바로 작성된 프로그램은 운영 체제에 독립적 이지만 JVM은 운영체제 에 종속적이어서 여러 운영체제에 설치할 수 있는 서로 다른 버전의 JVM을 제공하고 있다.
그래서 자바로 작성된 프로그램은 운영체제와 하드웨어에 관계없이 실행 가능하며 이것을 '한번 작성하면, 어디서나 실행된다.(Write once, run anywhere)'고 표현하기도 한다.

2. 객체지향언어이다.
자바는 프로그래밍의 대세로 자리 잡은 객체지향 프로그래밍 언어 (obiect-oriented pro- gramming language) 중의 하나로 객체지향개념의 특정인 상속, 캡슐화, 다형성이 잘 적용된 순수한 객체지향언어라는 평가를 받고 있다.

3. 배우기 쉽다.

4. 자동 메모리 관리 (Garbage Collection)
자바로 작성된 프로그램 이 실행되면, 가비지컬렉터 (garbage collector)가 자동적으로 메모리를 관리해주기 때문에 프로그래머는 메모리를 따로 관리 하지 않아도 된다.
가비지컬렉터가 없다면 프로그래머가 사용하지 않는 메모리를 체크하고 반환하는 일을 수동적으로 처리해야할 것이다. 자동으로 메모리를 관리한다는 것이 다소 비효율적인 면도 있지만, 프로그래머가 보다 프로그래밍에 집중할 수 있도록 도와준다.

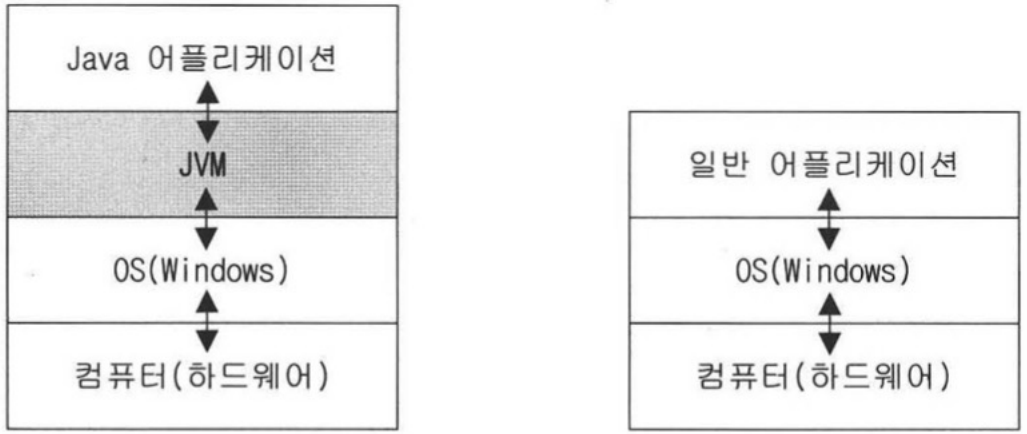
5. 네트워크와 분산처리를 지원한다.
인터넷과 대규모 분산환경을 염두에 둔 까닭인지 풍부하고 다양한 네트워크 프로그래밍 라이브러리(Java API)를 통해 비교적 짧은 시간에 네트워크 관련 프로그램을 쉽게 개발할 수 있도록 지원한다.

6. 멀티쓰레드를 지원한다.
자바스크립트는 싱글스레드를 지원함

7. 동적 로딩 (Dynamic Loading)을 지원한다.
보통 자바로 작성된 어플리케이션은 여러 개의 클래스로 구성되어 있다. 자바는 동적 로 딩을 지원하기 때문에 실행 시에 모든 클래스가 로딩되지 않고 필요한 시점에 클래스를 로딩하여 사용할 수 있다는 장점이 있다. 그 외에도 일부 클래스가 변경되어도 전체 어플 리케이션 을 다시 컴파일하지 않아도 되며, 어플리케이션의 변경사항이 발생해도 비교 적 적은 작업만으로도 처리할 수 있는 유연한 어플리케이션을 작성할 수 있다.

JVM은 통역가이다

일반 어플리케이션의 코드는 os만 거치고 하드웨어로 전달되는데 비해 Java어플리케이션 은JVM을 한번더거치기때문에 그리고하드웨어에맞게완전히컴파일된상태가아니고 실행 시에 해석(interpret)되기 때문에 속도가 느리 다는 단점을 가지고 있다. 그러나 요즘엔 바이트코드(컴파일된 자바코드)를 하드웨어의 기계어로 바로 변환해주는 JIT컴파일러와 향상된 최적화 기술이 적용되어서 속도의 격차를 많이 줄였다.



단, JVM은 OS에 종속적이기 때문에 해당 OS에서 실행가능한 JVM이 필요하다.

[그림1-2] 다양한 OS용 JVM

자바를 실행하기 위해서는

자바로 프로그래밍 을 하기 위해서 는 먼저 JDK(Java Development Kit)를 설치해야 한다.
JDK를 설치하면, 자바가상머신(Java Virtual Machine, JVM)과 자바클래스 라이브러리 (Java API)외에 자바를 개발하는데 필요한 프로그램들이 설치된다.

```
class 클래스명 {
   /*
        주석을 제외한 모든 코드는 클래스의 블럭{} 내에 작성해야한다.
   */
}
```

[참고] 나중에 배우게 될 package문과 import문은 예외적으로 클래스의 밖에 작성한다.

자바의 모든 코드는 반드시 000 안에 존재해야 한다.

자바에서 모든 코드는 반드시 클래스 안에 존재해야 하며 서로 관련된 코드들을 그룹으로 나누어 별도의 클래스를 구성하게 된다.
그리고 이 클래스들이 모여 하나의 Java 어플리케이션을 이룬다.
클래스를 작성하는 방법은 간단하다. 키워드 'class' 다음에 클래스의 이름을 적고, 클래스의 시작과 끝을 의미하는 괄호{}안에 원하는 코드를 넣으면 된다.

```
class 클래스명 {
    public static void main(String[] args)  // main메서드의 선언부
    {
        // 실행될 문장들을 적는다.
    }
}
```

자바 파일 작성 원칙

Java 어플리케이션은 main메서드의 호출로 시작해서 main메서드의 첫 문장부터 마지막 문장까지 수행을 마치면 종료된다.
모든 클래스가 mam메서드를 가지고 있어야 하는 것은 아니지만
하나의 Java 어플리케 이션에는 main메서드를 포함한 클래스가 반드시 하나는 있어야 한다.

main메서드를 포함한 클래스가 반드시 하나는 있어야 하는 이유?

main메서드는 Java 어플리케이션의 시작점이므로 main메서드 없이는 Java 어플리케이션은 실행될 수 없기 때문이다.

하나의 소스파일에 하나의 클래스만을 정의하는 것이 보통이지만, 하나의 소스파일에 둘 이상의 클래스를 정의하는 것도 가능하다.
이 때 주의해야할 점은 **'소스파일의 이름은 public class의 이름과 일치해야 한다.'**는 것이다.
만일 소스파일 내에 public class가 없다면, 소스파일의 이름은 소스파일 내의 어떤 클래스의 이름으로 해도 상관없다.

올바른 작성 예	설 명

Hello2.java
`public class Hello2 {}`
`class Hello3 {}`
	public class가 있는 경우, 소스파일의 이름은 반드시 public class의 이름과 일치해야한다.

Hello2.java
`class Hello2 {}`
`class Hello3 {}`
	public class가 하나도 없는 경우, 소스파일의 이름은 'Hello2.java', 'Hello3.java' 둘 다 가능하다.

잘못된 작성 예	설 명

Hello2.java
`public class Hello2 {}`
`public class Hello3 {}`
	하나의 소스파일에 둘 이상의 public class가 존재하면 안된다. 각 클래스를 별도의 소스파일에 나눠서 저장하던가 아니면 둘 중의 한 클래스에 public을 붙이지 않아야 한다.

Hello3.java
`public class Hello2 {}`
`class Hello3 {}`
	소스파일의 이름이 public class의 이름과 일치하지 않는다. 소스파일의 이름을 'Hello2.java'로 변경해야 맞다.

hello2.java
`public class Hello2 {}`
`class Hello3 {}`
	소스파일의 이름과 public class의 이름이 일치하지 않는다. 대소문자를 구분하므로 대소문자까지 일치해야한다.
	그래서, 소스파일의 이름에서 'h'를 'H'로 바꿔야 한다.

자바프로그램의 실행과정

만일 클래스에 main 메서드가 존재하지 않는다면 다음과 같이 에러 메시지가 출력된다.
Exception in thread “main” java.lang.NoSuchMethodError: main

1. 프로그램의 실행에 필요한 클래스(*.class파일)를 로드한다.
2. 클래스파일을 검사한다.(파일형식, 악성코드 체크)
3. 지정된 클래스(Hello)에서 main(String[] args)를 호출한다.

main메서드의 첫 줄부터 코드가 실행되기 시작하여 마지막 코드까지 모두 실행되면 프로그램이 종료되고, 프로그램에서 사용했던 자원들은 모두 반환된다.

주석 달기

1️⃣ 한줄 주석 - 앞에 // 넣을 경우
ex) // String hello = “Hello”;

2️⃣ 범위 주석 - /* 이 부분이 범위 주석처리됨 */
ex)
/*
여기는 주석처리가 됩니다.
*/

👀 변수

변수의 의미와 변수의 선언

컴퓨터 언어에서 변수 (variable)란 값을 저장할 수 있는 메모리상의 공간을 의미한다.

변수를 사용하기 위해서는 먼저 변수를 선언해야한다. 변수가 선언되면 메모리에 변수의 타입에 알맞은 크기의 저장공간이 확보되어, 값을 저장할 준비가 되는 것이다.
변수를 선언하는 방법은 다음과 같다.

```
변수타입 변수이름;

int number;            // 정수형 변수 number를 선언한다.
```

변수를 선언할 때는 변수의 타입과 이름을 함께 써주어야 한다. 위의 예는 number라는 이름의 정수형 변수를 선언한 것이다. 변수타입은 변수에 담을 값의 종류와 범위를 충분히 고려하여 결정해야한다.

변수를 선언한 후부터는 변수를 사용할 수 있으며, 변수를 사용하기에 앞서 적절한 값을 저장해주는 것이 필요하다. 이것을 변수의 초기화라고 하는데, 보통 아래와 같이 변수의 선언과 함께 한다.

```
// 정수형 변수 number를 선언하고 변수의 값을 10으로 초기화한다.
int number = 10;

// 위 문장은 아래의 두 문장과 동일하다.
int number;    // 정수형 변수 number를 선언하고
number = 10;   // 변수 number에 10을 저장한다.
```

양쪽의 코드는 서로 같은 의미의 다른 코드이다. 변수의 타입이 같은 경우 오른쪽과 같이 콤마','를 이용해서 코드를 간결하게 할 수 있다는 것을 보여준다.

```
int a;
int b;

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

⟷

```
int a, b;
int x=0, y=0;
```

변수의 종류에 따라 변수의 초기화를 생략할 수 있는 경우도 있지만, 변수는 사용되기 전에 적절한 값으로 초기화 하는 것이 좋다.

[참고] 지역변수는 사용되기 전에 초기화를 반드시 해야 하지만 클래스변수와 인스턴스변수는 초기화를 생략할 수 있다. 변수의 초기화에 대해서는 후에 자세히 학습하게 될 것이다.

1.3 명명규칙(naming convention)

변수의 이름, 메서드의 이름, 클래스의 이름 등 모든 이름을 짓는 데는 반드시 지켜야 할 공통적인 규칙이 있으며 다음과 같다.

1. 대소문자가 구분되며 길이에 제한이 없다.
   - True와 true는 서로 다른 것으로 간주된다.
2. 예약어를 사용해서는 안 된다.
   - true는 예약어라서 사용할 수 없지만, True는 가능하다.
3. 숫자로 시작해서는 안 된다.
   - top10은 허용하지만, 7up은 허용되지 않는다.
4. 특수문자는 '_'와 '$'만을 허용한다.
   - $harp은 허용되지만, S#arp은 허용되지 않는다.

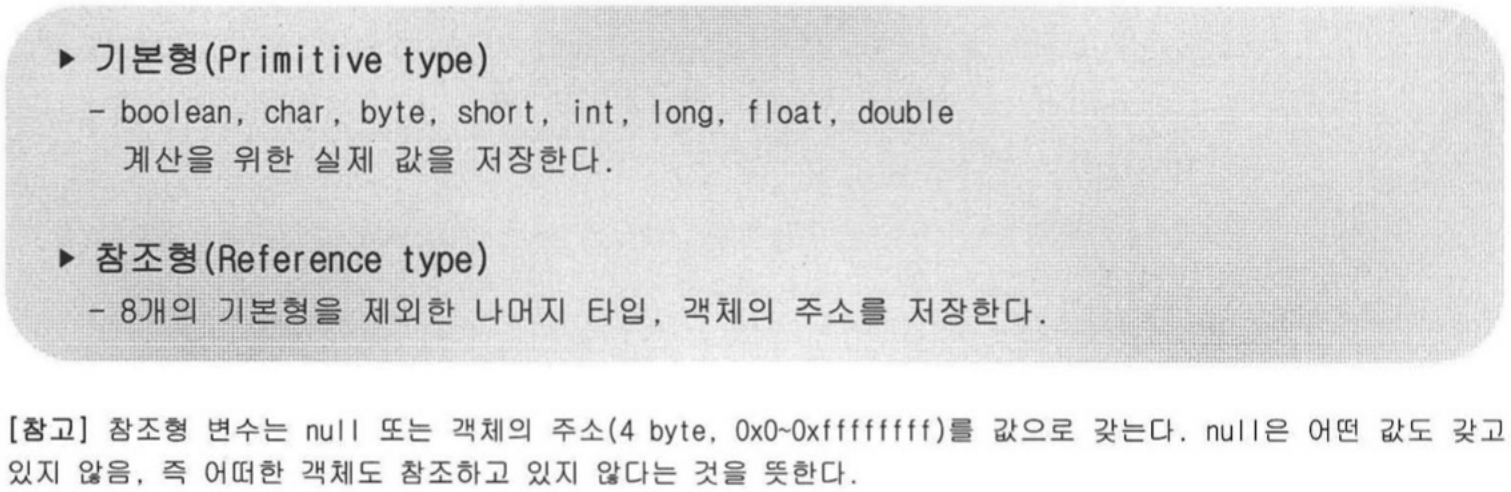
[참고] 예약어는 키워드(keyword) 또는 리져브드 워드(reserved word)라고 하는데, 프로그래밍언어의 구문에 사용되는 단어를 뜻한다. 그래서 예약어는 클래스나 변수, 메서드의 이름(identifier)으로 사용할 수 없다.

1. 클래스 이름의 첫 글자는 항상 대문자로 한다.
   - 변수와 메서드의 이름의 첫 글자는 항상 소문자로 한다.
2. 여러 단어로 이루어진 이름은 단어의 첫 글자를 대문자로 한다.
   - lastIndexOf, StringBuffer
3. 상수의 이름은 모두 대문자로 한다. 여러 단어로 이루어진 경우 '_'로 구분한다.
   - PI, MAX_NUMBER

위의 규칙들은 반드시 지켜야 하는 것은 아니지만, 코드를 보다 이해하기 쉽게 하기 위한 자바 개발자들 사이의 암묵적인 약속이다. 이 규칙을 따르지 않는다고 해서 문제가 되는 것은 아니지만 가능하면 지키도록 노력하자.

변수의 타입은 크게 2가지로 나뉜다.

변수의 타입은 크게 기본형과 참조형 2가지로 나눌 수 있는데
기본형 변수는 **실제 값 (data)**을 저장하는 반면
참조형 변수는 **어떤 값이 저장되어 있는 주소를** 값으로 갖는다.
자바는 C언어와 달리 참조형 변수 간의 연산을 할 수 없으므로 실제 연산에 사용되는 것은 모두 기본형 변수이다.



기본형의 개수는 모두 8개이고, 참조형은 프로그래머가 직접 만들어 추가할 수 있으므로 그 수가 정해져 있지 않다.

종 류 \ 크 기	1 byte	2 byte	4 byte	8 byte
논리형	boolean			
문자형		char		
정수형	byte	short	**int**	long
실수형			float	**double**

[표2-2] 기본형의 종류와 크기

자료형	저장 가능한 값의 범위	크 기	
		bit	byte
boolean	false, true	8	1
char	₩u0000~₩uffff (0~2^{16}-1, 0~65535)	16	2
byte	-128~127 (-2^7~2^7-1)	8	1
short	-32,768~32,767 (-2^{15}~2^{15}-1)	16	2
int	-2147483648~2147483647 (-2^{31}~2^{31}-1)	32	4
long	-9223372036854775808~9223372036854775807(-2^{63}~2^{63}-1)	64	8
float	1.4E-45~3.4028235E38	32	4
double	4.9E-324~1.7976931348623157E308	64	8

* int형은 대략 9자리 수(약 20억 , 2 , 000 , 000 , 000)의 값을 저장할 수 있다.

논리형(boolean)
논리형에는 'boolean' 한가지 밖에 없다. boolean형 변수에는 true와 false 중 하나를 저 장할 수 있으며 기본값 (default)은 false이다
boolean형 변 수는 대답 (yes/no), 스위 치 (on/off) 등 의 논리구현 에 주로 사용된다. 그리고 boolean형은 true와 false, 두 가지의 값만을 표현하면 되므로 기본형 중에서 가장 크기가 작은 1 byte이다.

문자형(char)
유니코드는 세계 각 국의 언어를 통일된 방법으로 표현할 수 있게 제안된 국제적인 코드 규약이다. 미국에서 개발되어진 컴퓨터는 그 구조가 영어를 바탕으로 정의되어 있기에 26 자의 영문 알파뱃과 몇 가지 특수 문자를 표현하기에는 1 byte 만으로 충분하였기 때문에 모든 정보(문자)가 1 byte를 단위로 표현되고 있었으나 동양 3국의 한글 , 한자 또는 일어 등과 같은 문자는 1 byte로는 표현이 불가능하기에 2바이트로 문자를 표현하는 유 니코드가 만들어 졌다.

ASCII코드는 1287ß의 문자집합을 제공하는 7비트 (bit) 부호로 , 처음 327ß의 부호는 인쇄 와 전송 제어용으로 사용된다. 보통은 ASCII코드를 확장해서 8비트 (1바이트 , 2567ß의 문 자표현가능)로 된 문자집합(character set)을 사용하는데 1287ß의 ASCII코드와 특수문자 들로 이루어져 있다.
char형의 크기는 2 byte이므로 16진수로 0000부터 ffff까지 , 문자를 표현하는데 65536 개(216)의 코드를 사용할 수 있으며, char형 변수는 이 범위 내의 코드 중 하나를 저장할 수 있다

char형 변수에 저장되는 값은 부호없는 정수의 형태로 저장된다. 이 값은 해당문자의 유니코드 인데 char c h 'A';가 수행되면 char형 변수 ch에 는 문자 'A'의 유니코드인 657} 저장된다. 변수에 문자를 저장하는 것 같지만 실제로는 정수값(유니코드)이 저장되는 것이 다. 즉, **모든 데이터는 숫자로 저장**된다. 변수의 값을 출 력할 때 변수의 타입이 정수형이면 정수값을 그대로 출력하고 char형이면 정수값에 해당하는 문자를 찾아서 출력한다.

char형 변수는 단 하나의 문자 밖에 저장할 수 없기 때문에 여러 문자를 저장하기 위해서는 String클래스를 사용해야 한다.

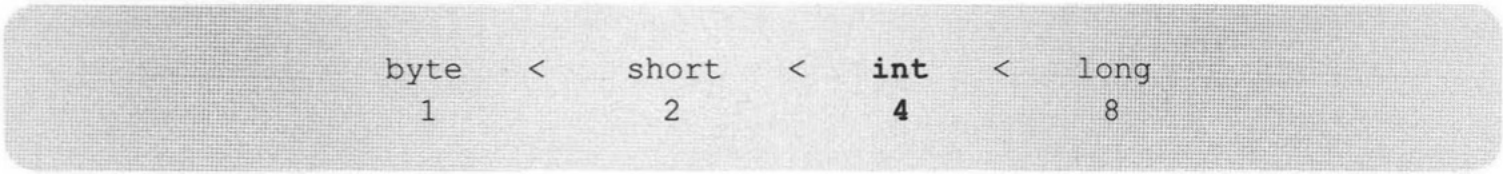
[예제2-4]/ch2/StringTest.java

```
class StringTest {
    public static void main(String[] args) {
        String a = 7 + " ";
        String b = " " + 7;
        String c = 7 + "";
        String d = "" + 7;
        String e = "" + "";
        String f = 7 + 7 + "";
        String g = "" + 7 + 7;

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(d);
        System.out.println(e);
        System.out.println(f);
        System.out.println(g);
    }
}
```

[실행결과]

```
7
 7
7
7

14
77
```

정수형 (byte, short, int, long)



byte부터 long까지 1 byte부터 시작해서 2배씩 크기가 증가한다는 것을 알 수 있다.

변수에 저장하려는 정수값의 범위에 따라 4개의 정수형 중에서 하나를 선택하면 되겠지만,
byte, short보다는 int를 사용하도록 하자.

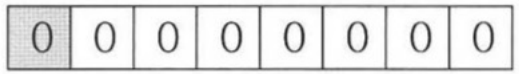
byte와 short이 int보다 크기가 작아서 메모리를 조금 더 절약할 수는 있지만, 저장할 수 있는 값의 범위가 작은 편이라서 연산 시에 범위를 넘어서 잘못된 결과를 얻기가 쉽다.

그리고 JVM의 피연산자 스택 (operand stack)이 피연산자를 4 byte단위로 저장하기 때문에 크기가 4 byte보다 작은 자료형 (byte, short)의 값을 계산할 때는 4byte로 변환하여 연산이 수행된다. 그래서 오히려 int를 사용하는 것이 더 효율적이다.

```
byte  b = 1;
short s = 2;
int   finger = 10;
long  big = 100000000000L;  // long타입의 리터럴에는 접미사 L을 붙여야 한다.
```

[참고] long타입의 리터럴에는 접미사 'L' 또는 'l'(소문자)을 반드시 붙여야한다. 리터럴에 접미사가 붙어있지 않으면 int타입으로 간주하기 때문이다. 'l'은 숫자 '1'과 혼동할 수 있으므로 'L'을 사용하자.

[참고] 그 자체로 데이터인 것을 '리터럴(literal)'이라고 한다. 예를 들어 'A', "AB", 123은 리터럴이다. 사실 리터럴은 상수(constant)와 의미가 같지만 프로그래밍에서는 상수를 '값을 한번 저장하면 변경할 수 없는 저장공간'으로 정의하기 때문에 이와 구분하기 위해 '리터럴'이라는 용어를 사용한다.



[그림2-1] byte의 2진 표현

1 byte는 위의 그림에서 알 수 있듯이 8자리로 구성되어 있으며, 각 자리에는 O또는 1 만이 허용된다. 그래서 1 byte로 28 즉 256가지의 값을 표현할 수 있다.
모든 정수형은 0을 포함한 양수와 음수를 저장하는 값의 범위로 하기 때문에, 8자리 중 에서 왼쪽에서 첫 번째 자리를 부호자리 (sign bit)로 사용한다.

그래서 실제로 값을 표현 할 수 있는 자리수는 모두 7개,
즉 표현할 수 있는 값의 범위는 土27개가 되는 것이다.
그런 이유로 byte형 변수가 가질 수 있는 값의 범위가 -128-127(-27_27-1)인 것이다

[예제2-5]/ch2/ByteOverflow.java

```
class ByteOverflow
{
    public static void main(String[] args)
    {
        byte b = 0;                 // byte형 변수 b를 선언하고 0으로 초기화.
        int  i = 0;

        // 반복문을 이용해서 b의 값을 1씩, 0부터 270까지 증가시킨다.
        for(int x=0; x <= 270; x++) {
            System.out.print(b++);   // print는 출력 후  줄바꿈을 하지 않는다.
            System.out.print('\t');  // tab을 출력한다.
            System.out.println(i++);
        }
    }
}
```

[실행결과]

```
0       0
1       1
2       2
3       3
...          ← 실행결과의 일부를 생략
125     125
126     126
127     127
-128    128   ← byte의 최대값인 127을 넘어서 오버플로우 발생
-127    129
-126    130
...
-3      253
-2      254
-1      255   ← b의 값은 0~-1, i의 값은 0~255(256개)
0       256   ← 다시 0부터 시작해서 0~-1을 반복
1       257
2       258
...
12      268
13      269
14      270   ← 왼 쪽값(변수 b의 값)이 270이 아닌 14라는 것을 확인하자.
```

전에 언급한 것과 같이 byte형으로 표현할 수 있는 값의 범위는 -128~127(-2^7~2^7-1)이다. 위의 예제를 보면, byte형 변수 b의 값이 0부터 시작해서 1씩 계속 증가하여 270이 되어야 한다. 하지만, b가 표현할 수 있는 양의 정수값의 범위는 127까지 인데, 270까지 값이 증가되면 어떤 일이 일어날까? byte가 저장할 수 있는 범위를 넘어섰기 때문에 오버플로우(overflow)가 발생하게 된다. 그래도 에러없이 잘 실행된다. 다만 예상했던 결과를 얻지 못할 뿐이다.

그래서 원하는 결과를 얻으려면 byte보다 큰 정수형 변수, int형 변수를 사용해야한다.

그러나 변수의 범위를 넘는 값으로 초기화하는 것은 허용하지 않는다. 컴파일하면 에러가 발생한다.

```
byte b = 128;  // 컴파일 에러 발생. byte의 범위 밖의 값으로 초기화 할 수 없다.
```

실수형 (float, double)

실수는 정수와 달리 부동소수점(floating-point) 방식으로 저장된다. 부동소수점 방식은 실수를 $\pm a \times 10^n$의 형태로 표현하는 것인데, a는 가수(mentissa)이고 n은 지수(exponent)이다. 단, 가수 a는 $0 \leq a < 1$ 범위의 실수이어야 한다.

예를 들어 실수 3.1415를 부동소수점 방식으로 표현하면 0.31415×10^1이며 가수는 0.31415이고 지수는 1이 된다.

```
float pi = 3.14f;              // f 대신 F를 사용해도 된다.
double velocity = 3.0e5d;      //  d 대신 D를, e 대신 E를 사용해도 된다.
double rate = 1.618;           //  접미사 d를 생략할 수 있다.
```

정수형에서는 int가 기본 자료형인 것처럼 실수형에서는 double이 기본 자료형이다. 그래서 접미사를 생략하면 double형 리터럴로 간주된다.

```
float pi = 3.14;  // 에러. float형 변수에 double형 리터럴을 저장할 수 없음.
```

3.14는 접미사가 붙지 않았으므로 float형 리터럴이 아닌 double형 리터럴로 간주된다.

그래서 위의 문장은 컴파일 시에 에러가 발생한다. 컴파일 에러를 피하려면 3.14f와 같이 접미사를 붙여서 float형 리터럴로 변경해주어야 한다.

[참고] 리터럴에 접미사가 붙는 자료형은 long, float, double뿐인데, double은 생략가능하므로 long과 float의 리터럴에 접미사를 붙이는 것만 잊지 않으면 된다.

형변환

기본형과 참조형 사이에는 원래는 형변환이 성립되지 않는다.
기본형은 기본형으로만 참조형은 참조형으로만 형변환 이 가능하다

그러나, JDK1.5부터 오토박싱 (autoboxing)기능이 추가되어 기본형을 Object클래스나 wrapper클래스 같은 참조형으로 형변환할 수 있게 되었다.
즉, 기본형과 참조형 간의 형연환도 가능해진 것이다

(타입)피연산자

```
int score = (int)85.4;  // double형 값을 int형으로 변환하여 score에 85가 저장된다.
byte b = (byte)score;   // score에 저장된 값을 byte형으로 변환하여 byte에 저장.
```

8개의 기본형 중에서 boolean을 제외한 나머지 7개의 기본형 간에는 서로 형변환이 가능하다.

변 환	수 식	결 과
int → char	(char)65	'A'
char → int	(int)'A'	65
float → int	(int)1.6f	1
int → float	(float)10	10.0f

[표2-8] 기본형간의 형변환

각 자료형 마다 표현할 수 있는 값의 범위가 다르기 때문에 범위가 큰 자료형에서 범위가 작은 자료형으로의 변환은 값 손실이 발생할 수 있다.

예를 들어 실수형을 정수형으로 변환하는 경우 소수점 이하의 값은 버려지게 된다.
표 2-8에서도 알 수 있듯이 floatè.1터럴인 1.6f를 mt로 변환하면 I이 된다.

반대로 범위가 작은 자료형에서 큰 자료형으로 변환하는 경우에는 절대로 값 손실이 발 생하지 않으므로 변환에 아무런 문제가 없다

[예제2-8]/ch2/CastingEx2.java

```
class CastingEx2
{
    public static void main(String[] args)
    {
        byte b = 10;
        int i = (int)b;
        System.out.println("i=" + i);
        System.out.println("b=" + b);

        int i2 = 300;
        byte b2 = (byte)i2;
        System.out.println("i2=" + i2);
        System.out.println("b2=" + b2);
    }
}
```

[실행결과]

```
i=10
b=10
i2=300
b2=44
```

int i (int)b;를 int b;와 같이 캐스트 연산자를 생략할 수 있으나
byte b2 = (byte)i2;에서는 캐스트 연산자를 생략해서 byte b2 = i2;와 같이 할 수 없다.

값의 표현범위가 큰 자료형에서 작은 자료형으로의 형변환에 캐스트 연산자를 사용하지 않으면 컴파일시 에러가 발생한다.