CHAPTER 02

데이터를 담아두는 상자 변수(Variables)

프로그램에서 데이터를 표현할 때 사용하는 변수라는 개념을 정확히 배워봅시다.

프로그램을 구성하는 요소는 크게 두 가지라 할 수 있습니다. 하나는 처리해야 하는 데이터이고 다른 하나는 그 데이터를 핸들링하는 논리(이하 로직(Logic))입니다. 여러분이 어떤 언어를 배우는 과정 역시 이 두 가지를 어떻게 사용하는 것인지를 배운다고 생각하시면 됩니다. 그중에서도 가장 우선이 되는 것이 데이터를 프로그램으로 선언하는 문법적인 부분인데 이를 변수 선언이라고 합니다.

1 변수란 무엇인가?

변수(Variables)는 어떤 데이터를 나타내는 일종의 기호 혹은 지시 대명사와 같습니다. 비슷한 의미로 특정한 기호를 이용해서 의미를 전달하는 방식이라고 생각할 수도 있습니다. 영어에서 this와 that이 가리키는 대상이 상황마다 다르듯이 변수도 현재 코드 내에서 어떤 데이터를 의미하려고 사용합니다.

변수를 어렵게 생각하실 것 없습니다. 가장 쉬운 예제를 하나 보도록 합니다.

예제 | 숫자 변수를 하나 선언해본 VarTest.java

```
public class VarTest1 {

        public static void main(String[] args) {

                int v1 = 10;
```

```
                System.out.println(v1);
                System.out.println(v1);
                System.out.println(v1);
                System.out.println(v1);
                System.out.println(v1);
        }
}
```

```
10
10
10
10
10
```

위에서 'int v1'이라는 것이 바로 변수 선언입니다. 이 변수를 해석할 때는 다음과 같은 방식으로 해석합니다.

변수를 해석할 때에는 다음과 같이 해석합니다.

- int v1 = 10; → v1라는 이름의 변수에는 숫자 10이라는 값을 담는다.

사실 정확한 표현으로 하자면 메모리상에 있는 어떤 데이터를 가리킨다고 표현하는 것이 정확합니다. 다만, 이 책에서 변수를 상자에 비유하고 데이터를 담는다고 표현하는 것은 좀 더 직관적인 이해를 돕기 위해서입니다.

변수가 만들어지면 메모리상에 어떤 공간을 차지하게 되는데 이것을 하나의 상자처럼 생각하시면 됩니다. 그리고 상자 안에 담기는 수는 변수가 의미하는 데이터입니다.

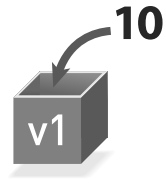


그림 1 변수라는 상자

변수를 가장 쉽게 이해하는 것은 그저 변수가 하나의 상자라고 생각하는 겁니다. 따라서 위는 v1이라고 이름이 붙은 상자가 하나 있고, 그 상자 안의 내용물이 10이라는 데이터라고 생각하시면 됩니다. 우리가 10이라는 데이터를 보통은 '리터럴(Literals, 상수)'이라고 합니다. 리터럴은 간단히 말하자면 누가 봐도 동일한 의미의 기호입니다. 즉 '10'이라 것은 숫자 10을 의미하며 'A'라는 글자가 알파벳 A라는 것은 모든 사람이 공통으로 인식하는 하나의 의미입니다.

앞의 코드를 보면 변수를 써서 뭐가 좋아지는지도 단번에 보입니다. 코드의 실행 결과를 보면 v1이라는 상자에 들어 있는 내용물이 출력되는 것을 보실 수 있습니다. 하지만, 제가 바보가 아닌 이상 같은 코드를 왜 여러 번 사용했을까요? 눈치가 빠르신 분들은 아시겠지만, 위의 코드에서 v1 상자의 내용물을 20으로 변경하면 나머지 코드는 손대지 않아도 수정한 후에는 결과가 20으로 변경된다는 겁니다.

> 데이터를 변수를 이용해서 선언하는 이유는 필요한 데이터를 나중에 한 번만 수정하기 위해서입니다.

이번에는 조금 다른 예제를 봅니다.

예제 | 두 개의 변수로 다양한 연산을 하는 코드

```
public class VarTest2 {

	public static void main(String[] args) {

		int a = 100;
		int b = 20;

		System.out.println( a + b);
		System.out.println( a - b);
		System.out.println( a * b);
		System.out.println( a / b);
	}
}
```

```
120
80
2000
5
```

이번에는 변수를 두 개를 선언해 보았습니다. 그리고 몇 가지의 연산 작업을 해 보았습니다. 연산자라는 것은 뒤에서 다시 배우겠지만, 우선은 변수를 이용했더니 여러 번 사용할 때 편리하더라는 것을 봐 주시면 됩니다. 위의 코드에서 a라는 변수나 b라는 변수의 값을 수정해 주기만 하면 나머지 코드는 수정하지 않아도 됩니다.

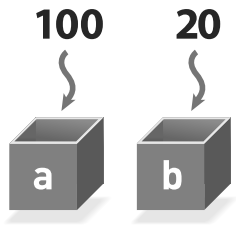


그림 2 변수와 상자

결국, 위의 내용을 정리해 보면 다음과 같은 두 가지 가정이 생깁니다.

- 변수를 선언하고 변수를 이용한 코드가 많으면 나중에 수정할 때 편리하다.
- 변수는 어떤 데이터를 나타내기 위해서 사용한다.

이제 각 내용에 대해서 좀 더 자세히 알아보도록 하겠습니다.

2 변수는 왜 사용하는가?

변수는 필요한 데이터를 한 번 선언해 두고, 나중에 계속해서 사용하기 위한 하나의 문법적인 장치입니다.

2.1 변수를 이용하면 나중에 한 곳만 고쳐도 됩니다.

여러분이 변수라는 말을 처음 들어본 곳은 아마도 수학을 배울 때였을 겁니다. 주로 다음과 같은 방식으로 사용합니다.

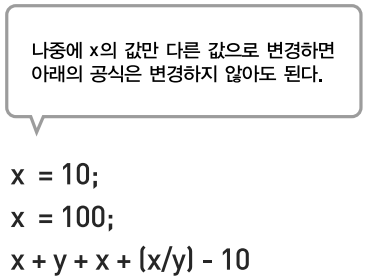


그림 3 변수의 사용

수학에서는 왜 변수를 사용할까요? 변수라는 것을 이용하게 되면 나중에 변수에 할당된 값만 수정하면 다른 부분을 수정하지 않아도 되기 때문입니다. 변수를 이용하면 변수를 사용하는 곳이 10곳이든 100곳이든 상관이 없습니다. 최소의 변경으로 최대의 효과를 얻기 위해서는 매번 숫자를 활용하는 것이 아니라 나타내려고 하는 숫자에 대해서 일종의 지시 대명사를 이용하는 것이 바로 변수를 이용하는 이유입니다.

2.2 앞으로 그 내용이 변할지도 모르는 수이므로 변수라 합니다.

수학에서는 변수에 x나 y라는 값을 이용합니다. 혹은 다른 수많은 기호를 이용하는 것이 일반적입니다. 기호라는 것이 그렇지만 기호는 일종의 상징입니다. 만일 여러분이 코드 중간에 x라는 값이나 y라는 값을 만나셨다면 여러분은 이 값이 어떤 외부에 의해서 변경되는 값이라는 점을 생각하게 될 겁니다. 즉, 변수는 말 그대로 변하는 수를 의미합니다. 프로그래밍에서는 변수를 많이 사용하기 때문에 변수의 이름을 조금 더 구체적으로 사용하는 경우가 더 많습니다.

- 숫자 userInput　　ex) int userInput
- 문자 name　　ex) String name

위의 예는 문법에 맞추기보다는 여러분이 뜻을 쉽게 이해하실 수 있도록만 변수를 선언해본 것입니다. userInput이나 name이라는 것을 프로그래밍에서는 변수라고 하는데 여러분이 프로그래밍을 전혀 모른다고 하더라도 userInput이나 name이라는 변수를 보면 "이것은 사용자가 입력한 데이터를 의미하는 것이군." 혹은 "이건 이름을 표시하기 위해서 쓴 것이군."처럼 짐작할 수 있습니다. 가끔은 기초적인 프로그래밍 책에 간단하게 변수의 이름을 a, b, c, d와 같이 사용하는 것

을 보곤 하는데 별로 추천할 만한 습관은 아닙니다. 앞으로도 누누이 강조하겠지만, 프로그램은 나만을 위해서 작성하는 것이 아니기 때문입니다. 프로그램을 만드는 것은 여러분이지만, 다른 사람들이 항상 여러분이 작성하는 코드를 볼 것이라는 것을 기억하시기 바랍니다.

2.2.1 수학과 프로그래밍에서의 변수의 의미

- 수학에서는 변수라는 것이 주로 숫자만을 의미합니다.
- 프로그래밍에서의 변수라는 것은 문자, 숫자 등 여러 가지를 의미할 수 있습니다.

'변수'라는 용어 자체가 주로 사용되는 것은 프로그래밍과 수학입니다. 그럼 수학에서 쓰이는 변수와 프로그래밍에서 쓰이는 변수라는 말은 어떻게 다른 걸까요? 프로그래밍에서 사용하는 변수와 수학에서 사용하는 변수의 가장 큰 차이는 바로 프로그래밍에서 의미하는 변수는 단순히 숫자를 의미하지 않는다는 점에 있습니다. 프로그래밍에서의 변수는 훨씬 더 많은 것을 의미할 수 있습니다. 숫자나 문자 혹은 자료구조 등을 의미할 수 있다는 겁니다. 이러한 이유 때문에 프로그래밍에서 사용되는 변수라는 것은 조금 다른 방식의 접근이 필요합니다.

2.2.2 프로그래밍에서 변수: 데이터를 의미하는 상자

그럼 이제 좀 더 본격적으로 프로그래밍에서 사용하는 변수에 대해서 알아보도록 합니다. 프로그래밍에서의 변수는 다음과 같이 몇 가지 점을 알아 두어야만 합니다.

- 프로그래밍에서 변수는 상자에 어떤 데이터(문자인지, 숫자인지)를 의미하는지를 미리 언급하는데 이것을 변수의 자료형(Type)이라 합니다.
- 프로그래밍에서의 모든 변수는 이름(태그)을 붙여서 구분합니다.
- 변수를 선언한다는 것은 메모리에 일정 공간을 차지하는 상자를 만든다는 겁니다.

이제 본격적으로 Java에서 변수를 선언하는 방법을 배워 보도록 합니다.

3 Java로 변수를 선언한다는 것

Java에서 변수를 선언할 때는 다른 프로그래밍 언어보다도 조금 엄격한 문법을 사용합니다. 변수가 어떤 종류의 데이터를 담고 있는지와, 변수가 어떤 수치의 데이터들을 표현할 수 있는지를 아주 엄격하게 적용합니다. 예를 들어 웹에서 많이 쓰이는 Javascript(Java와는 아무런 관계가 없습니다.)와 같은 언어는 데이터가 문자인지 숫자인지 상관하지 않고 그냥 'var a = 10;' 혹은 'var a = "홍길동";'과 같은 방식으로 표현할 수 있지만 Java의 경우는 숫자는 숫자, 문자는 문자로 정확히 기술해 주어야만 합니다.

Java는 반드시 변수의 타입에 맞는 데이터만 표현할 수 있습니다. Java, C, C++ 등 역시 비슷한데 이런 언어들을 엄격한 언어라고 합니다. 엄격한 언어는 인간이 작성할 때에는 조금 번거롭지만, 컴퓨터가 기계어를 해석할 때 명확하기 때문에 수행 시에 이득이 있습니다. 또한, 다른 개발자들이 코드를 볼 때 변수가 어떤 데이터인지 짐작할 수 있다는 장점이 있습니다.

3.1 Java에서 변수의 종류: 기본 자료형 vs 객체 자료형

기본 자료형(Primitive Type)이란 프로그래밍을 하는데 가장 많이 쓰이는 데이터를 쉽게 사용하기 위해서 프로그래밍언어에서 미리 만들어 둔 타입을 의미합니다. 이에 비해 객체 자료형은 단순한 데이터가 아닌 복합적인 데이터를 의미하고, C 언어에서는 포인터 변수라고 합니다. Java에서는 레퍼런스라는 용어를 사용합니다. 쉽게 말하면 기본 자료형은 단순한 데이터를 담는 상자이고 객체 자료형은 덩치가 큰 데이터를 담을 때 사용하는 것으로 생각하면 됩니다.

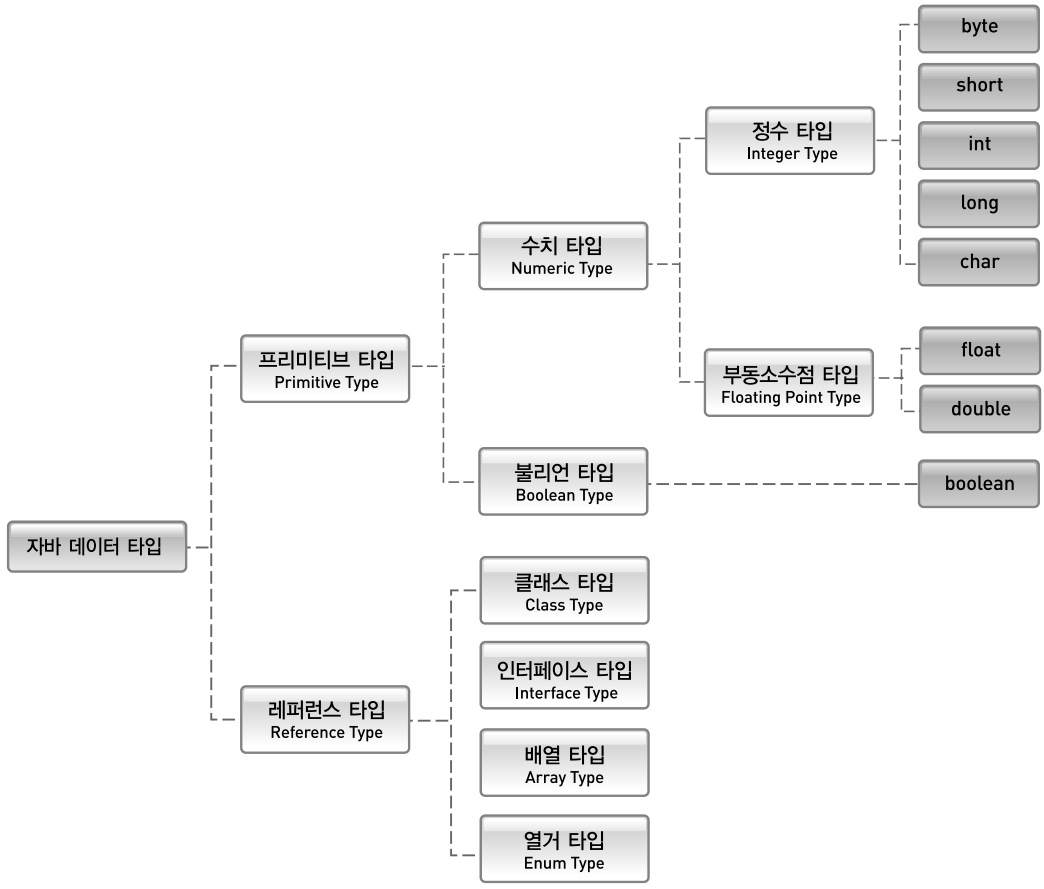


그림 4 Java에서의 변수 분류

예를 들어 여러분이 정수 x라는 변수를 선언하고 싶다면 int x와 같은 방식으로 선언하게 됩니다. 무언가 새로운 변수를 선언하려면 앞에 지금부터 선언할 변수가 어떤 종류의 데이터인지를 알려주고, 뒤에 어떤 이름으로 사용할 것인지를 알려주는 방식을 사용합니다.

3.2 기본 자료형에서 가장 많이 쓰이는 타입들

우선 가장 많이 사용되는 자료형을 간단히 선언해보도록 합니다. 여러분이 특별히 신경을 써서 코드를 작성하지 않는 이상 사실 아래의 4가지만 정확히 알고 있으면 됩니다.

- 정수를 표현할 때는 int를 사용한다.
- 소수를 표현할 때는 double을 쓴다.
- 참, 거짓(불 타입)인 경우에는 boolean으로 쓴다.
- 글자 하나를 의미할 때는 char를 쓴다.

예제 | 가장 많이 쓰이는 기본 자료형들

```
public class VarEx {
        public static void main(String[] args) {
                //정수를 표현할 때에는 int라는 타입을 쓴다.
                int i = 100;
                //소수를 표현할 때에는 double이라는 타입을 쓴다.
                double d = 10.5555;
                //참 거짓을 표현할 때에는 boolean을 쓴다.
                boolean b = true;
                //문자 하나를 의미할 때에는 char를 쓴다.
                char c = '가';

                System.out.println(i);
                System.out.println(d);
                System.out.println(b);
                System.out.println(c);
        }
}
```

```
100
10.5555
true
가
```

기본적으로 가장 연산이 많은 자료형은 정수, 소수, 문자 그리고 참/거짓입니다.

■ Java에서의 변수 선언 규칙(Naming Convention)

Java 언어는 꽤 엄격한 언어이기 때문에 변수를 선언하고 사용하는 데 있어서도 규칙이 있습니다.

- Java는 대, 소문자를 구분합니다.
- 문자나 '_' 혹은 '$'으로 변수 이름을 시작할 수는 있지만, 숫자로 변수 이름을 시작할 수는 없습니다.
- 지정된 단어(Keyword)는 사용할 수 없습니다.

이 중에서 가장 여러분이 신경 써야 하는 것은 대소문자를 명확하게 구분한다는 점입니다. 예를 들어 소문자로 작성된 name이라는 이름의 변수와 대문자로 시작하는 Name이라는 변수는 Java에서는 전혀 다른 변수를 의미합니다. 아울러 변수는 그 자체가 어떤 데이터를 담기 때문에 개발자들은 남들이 변수명을 보고 짐작이 가능하도록 신경을 써 주는 것이 일반적입니다. 따라서 변수를 작성할 때에는 가능하면 다음과 같은 가이드를 지켜주는 것이 좋습니다.

- 변수의 이름은 가능하면 의미를 알아볼 수 있는 이름으로 합니다.
  ex) userId, userPw, grade
- 변수의 이름은 소문자로 시작하고, 단어와 단어를 이어서 의미를 주고자 할 때에는 '_'를 이용하거나 연결되는 단어의 첫 글자를 대문자로 표시해서 구분합니다.
  ex) totalCount, sum_result, xPostion, salePrice

3.3 기본 자료형으로 변수 선언하기

변수를 선언하는 것을 간단히 표현하면 아래와 같습니다. 여러분이 반드시 기억해야 하는 사실은 변수를 선언하면 실제로 메모리상에 상자처럼 공간을 차지한다는 사실입니다.

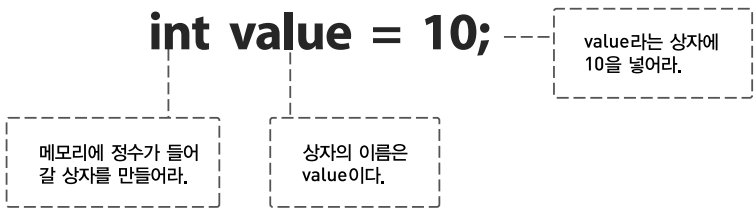


그림 5

3.3.1 변수의 맨 앞에는 상자의 종류가 들어갑니다.

여러분이 어떤 변수를 선언하게 되면 가장 먼저 JVM에게 지금부터 데이터를 보관할 공간을 만들라고 얘기하는 일이 필요합니다. 따라서 변수 선언의 맨 앞은 메모리에 만드는 상자의 종류를 의미합니다.

변수의 타입(Type)이라는 것은 택배 상자와 같습니다.
타입에 따라 담을 수 있는 데이터를 다르게 합니다.

그림 6

변수의 담을 데이터의 종류에 따라서 메모리상에 적당한 상자를 만들려면 반드시 앞에 상자의 종류를 명시합니다.

3.3.2 =을 기준으로 왼쪽에는 대상 상자, 오른쪽에는 상자의 내용물

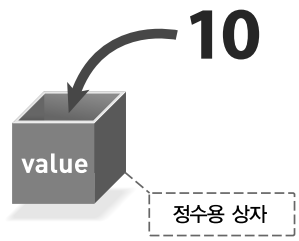


그림 7

변수 선언에 = 표시가 있다는 것은 메모리의 상자를 만들자마자 그 상자 안에 값을 넣어 주겠다는 표현입니다. =는 할당(Assign) 연산자입니다. 수학과는 다르게 단순한 지시 대명사가 아닙니다.

3.3.3 = 뒤에는 원하는 데이터를 줄 수도 있고 안 줄 수도 있습니다.

'int value = 10;'의 선언에서 = 뒤는 변수에 들어가는 데이터를 의미합니다. 변수를 선언할 때에는 1) 그냥 메모리상에 상자만 만들어 두거나, 2) 상자를 만들면서 데이터도 같이 넣어주거나 중에서 선택하게 됩니다. 2)의 경우를 "선언과 동시에 변수를 초기화한다."라고 표현합니다.

- int i; → i라는 이름의 변수를 메모리에 만들어라(상자만 만들어 둡니다).
- int i = 10; → i라는 이름의 변수를 메모리에 만들고, 10이라는 데이터를 넣어줍니다.

3.4 = 표시의 정확한 의미

프로그래밍에서 =은 엄청나게 중요한 의미가 있기 때문에 반드시 정확하게 의미를 이해해야 합니다.

- =은 할당 연산자(Assign Operator)입니다. 수학과는 달리 어떤 연산 작업을 한다는 표시입니다.
- =의 왼쪽은 데이터를 넣으려고 하는 대상을 의미합니다.
- =의 오른쪽은 꺼낼 데이터를 의미합니다.

예제 | = 오른쪽은 상자의 내용물이다.

```
public class VarTest {
	public static void main(String[] args) {
		int i = 10; // 상자 i에 데이터 10을 담는다.
		int j = i;  // j 상자에 i의 내용물을 꺼낸다.
		System.out.println(i);
		System.out.println(j);
	}
}
```

```
10
10
```

여기서 변수가 만들어지고 초기화되는 과정을 좀 더 자세히 보도록 합시다.

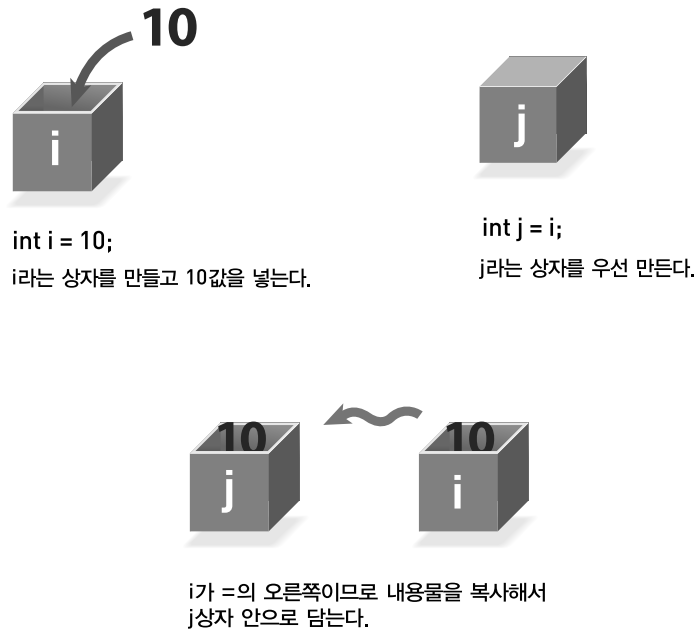


그림 8

보시는 바와 같이 화면에는 i 값도 10으로, j 값도 10으로 출력됩니다. 그럼 코드를 다음과 같이 수정하면 어떻게 될까요?

예제

```
int i = 10;
int j = i;
i = 100;
// j의 값은 10일까요? 100일까요?
```

마지막을 보면 i라는 변수를 다시 사용하고 있습니다. i라는 상자에는 100이라는 값을 넣어주었습니다. 우선 i라는 변수는 이미 위해서 한번 상자가 만들어졌기 때문에 다시 타입을 붙이지 않는다는 점을 주의 깊게 봐 주시면 좋겠습니다.

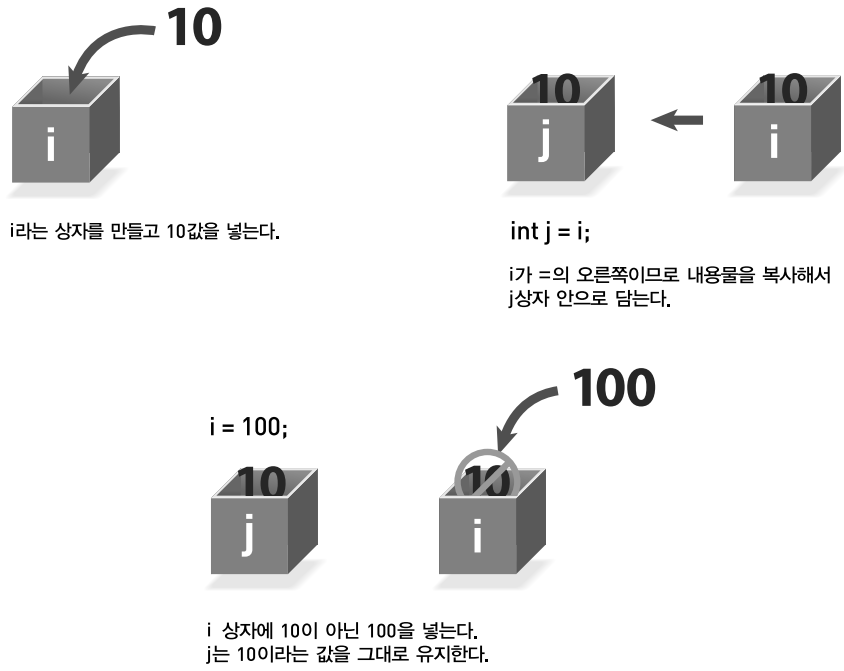


그림 9

기억하세요! Java에서 'a = b'는 b 상자의 내용물을 a 상자로 복사한다는 뜻입니다.

위의 그림과 같은 이유로 실제 프로그램을 작성해서 테스트한 결과는 다음과 같습니다.

```
100     ← 변수 i의 값
10      ← 변수 j의 값
```

일반적인 수학에서와는 결과가 다르게 나오는 것에 대한 이유를 정확하게 아실 필요가 있습니다.

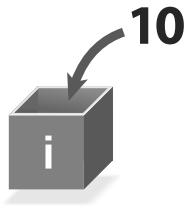
■ **=의 오른쪽과 왼쪽의 의미**

=이라는 할당 연산자의 오른쪽인지 왼쪽인지에 따라서 변수가 어떻게 사용되고 있는지를 한 가지 다른 경우를 들어서 한 번 더 이해해보도록 합시다.

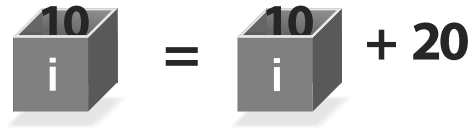
예제

```
int i = 10;
i = i + 20;
```

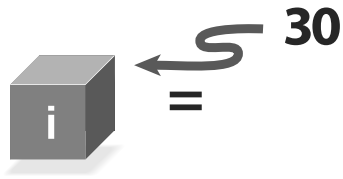
이번에는 i라는 이름의 변수가 =의 양쪽에 사용되었습니다. 이 경우 연산은 어떻게 될까요? 우선 왼쪽을 먼저 생각해 봅니다. 왼쪽은 데이터를 넣을 대상 상자를 의미하기 때문에 i라는 이름이 붙은 상자를 사용하겠다는 것입니다. 오른쪽은 어떻게 생각해야 할까요? =보다 오른쪽에 나오는 경우에는 상자 안에 들어 있는 데이터를 의미하기 때문에 i라는 상자 안에 있는 내용물을 꺼냅니다. 내용물은 당연히 위에 선언된 것처럼 10이라는 값이 들어가 있습니다. 그럼 + 연산을 통해서 30이라는 값이 만들어지게 됩니다.



int i = 10;
i라는 상자를 만들고 10이라는 값을 넣는다.



i = i + 20;
=의 오른쪽은 값을 꺼내는 상자



연산된 결과를 다시 상자 i에 담는다.

그림 10

마지막으로 양측을 다 종합해 보면 i라는 상자에 기존에 i라는 변수가 가진 값 10에 추가된 데이터 20을 더한 연산의 결과물이 다시 i라는 변수에 저장되는 것을 의미합니다.

예제 | 변수를 재사용하는 연산

```
public class VarTest2 {
        public static void main(String[] args) {
                int i = 10;
                i = i +20;
                System.out.println(i);
        }
}
```

```
30
```

프로그래밍에서의 변수의 개념을 잘 못 이해하게 되면 모든 프로그래밍을 제대로 할 수 없으므로 기본적인 내용이지만 확실히 공부하셔야만 합니다. 이제 변수라는 상자 안에 들어가는 내용물에 대해 공부를 하도록 합니다.

4 기본 자료형: byte, short, int, long

우선은 기본 자료형으로 가장 많이 선언하게 될 정수에 대한 이야기부터 시작하는 것이 좋을 듯 합니다. Java에서 정수를 처리하기 위해서 제공되는 것은 4가지입니다만, 여러분이 주로 많이 사용하게 될 것은 int라는 타입입니다. 우선은 정수 타입의 변수가 필요하다 생각되면 int 타입으로 선언하면 된다고 생각하실 만큼 숫자 사용 시에 많이 사용됩니다. 네 가지 타입이 의미하는 데이터는 아래와 같습니다.

자료형	크기	표현 범위
byte	1바이트	−128 ~ 127
short	2바이트	−32,768 ~ 32,767
int	4바이트	−2,147,483,648 ~ 2,147,483,647
long	8바이트	−9,223,372,036,854,775,808 ~ +9,223,372,036,854,775,807

표 기본 자료형

4.1 적절한 메모리 공간의 할당을 위한 다양한 타입

Java는 왜 정수를 표현하는 데 이렇게 4가지나 사용해야 할까요? 정수만 해도 벌써 4가지는 되기 때문에 개발자들은 더 힘들고 복잡해 집니다. 이 문제는 프로그램의 소스를 만드는 개발자가

아닌 컴퓨터의 입장에서 한번 생각해보면 이해하실 수 있을 겁니다. 앞서 프로그램이 실행되면서 변수 선언을 만나면 메모리에 상자가 하나 만들어진다고 설명했습니다. 정수를 표현하는 4가지의 타입에서는 이 경우 담을 수 있는 상자의 크기가 각각 달라집니다. 이 말이 의미하는 바는 변수의 타입에 따라서 메모리를 차지하는 공간의 크기가 다르다는 겁니다.

byte/short/int/long: 변수의 타입에 따라서 메모리 안의 상자 크기가 결정됩니다.

예제 적절한 크기의 변수 선언

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

                short s = 1000;
                System.out.println("변수 s는 " +s);

                int i = 10000;
                System.out.println("변수 i는 " + i);

                long l = 123456789123456789L;
                System.out.println("변수 l은 " + l);
        }
}
```

```
변수 b는 10
변수 s는 1000
변수 i는 10000
변수 l은 123456789123456789
```

컴퓨터의 입장에서는 실행할 때 상자를 만들 때 큰 상자를 만들어야 하는지 작은 상자를 만들어야 하는지를 타입을 보고 결정할 수 있습니다. 예를 들어 누군가 byte 타입의 변수를 선언했다면 위에서 보듯이 작은 정숫값을 담으려고 선언했다는 것이고, 메모리에 큰 공간을 할당하는 것이 아니라 작은 공간을 할당하게 됩니다.

4.2 'byte b = 10'의 해부: 2진수, bit, byte

기본 자료형을 이해하기 위해서는 bit와 byte의 관계를 알아둘 필요가 있습니다. 이 개념은 나중에 입출력 프로그래밍을 할 때에도 필요하므로 정확히 알아두어야 합니다.

4.2.1 byte b: 메모리에 1byte(8bit)의 공간을 만들어 냅니다.

예를 들어 byte b와 같은 선언을 만나면 메모리에 자동으로 8개의 bit가 모인 공간이 생성됩니다. 만들어진 8개의 bit에는 0으로 데이터가 채워집니다. 사실 이런 이유 때문에 기본 자료형들의 경우에는 변수에 값을 주지 않아도 기본값을 가지게 됩니다.

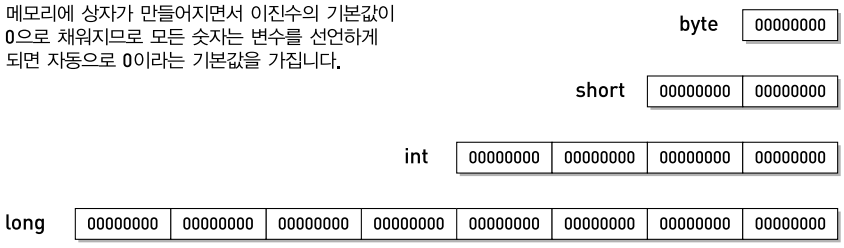


그림 11

4.2.2 byte b = 10: 10이라는 값을 0과 1로 표현한다: 먼저 2진법을 알아야 합니다.

이제 변수라는 상자에 값을 담는 과정을 봅니다. 'byte b = 10;'이라고 하게 되면 b라는 상자에 10이라는 데이터를 넣는다는 겁니다. 그럼 10이라는 숫자는 어떻게 처리될까요? 수학 시간에 2진법이라는 것을 배우신 적이 있을 겁니다. 2진법은 숫자의 단위가 10이 아니라 2라는 기준을 가지는 방식입니다. 따라서 10진법이 기준인 인간의 계산 방식과는 좀 다릅니다.

2진수	10진수
0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8

표

보시면 10진수에서 2의 값이 2진수에서는 10으로 표현되는 것을 볼 수 있습니다. 따라서 여러분이 byte에 해당하는 데이터의 값을 정확하게 표현하기 위해서는 10진수를 2진수로 변환하는 방법을 알아 두시면 좋을 듯합니다.

4.2.3 2진법은 2의 몇 제곱인지만 알면 됩니다.

2진수를 10진수로 변환할 때에는 아래처럼 표를 그려 놓으면 간단하게 알아낼 수 있습니다. 2진법의 핵심은 각 숫자가 2의 몇 승인지를 나타낸다는 겁니다.

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
10진수값	128	64	32	16	8	4	2	1
2진수값	0	0	0	0	0	1	1	1
						4 +	2 +	1

결과 7

그림 12 2진수 111의 계산

4.2.4 byte라는 자료형이 가질 수 있는 최댓값과 최솟값

byte 데이터 타입에는 8개의 bit가 들어갈 수 있습니다. 그럼 byte 타입이 가질 수 있는 최대의 값은 얼마일까요?

> (2진수)11111111

이 경우는 모든 데이터가 2의 x승 ×1의 값을 가지므로 가장 큰 값을 가질 것으로 생각됩니다. 하지만, 실제로는 예상과는 조금 다릅니다. byte는 8개 bit에 다음과 같이 값을 가질 수 있습니다. 문제는 맨 앞에 있습니다. 맨 앞에 들어가는 숫자가 1일 경우에는 음수가 됩니다. 즉, 위 맨 앞의 값은 2의 7승이긴 하지만 음수인 값입니다. 그림으로 표현하면 다음과 같습니다.

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
10진수값	128	64	32	16	8	4	2	1
2진수값	1	1	1	1	1	1	1	1
	-128 +	64 +	32 +	16 +	8 +	4 +	2 +	1

결과 -1

맨 앞에 1이면 음수가 되어서 -128이 됩니다.
(이름 부호 비트라고 합니다.)

그림 13 2진수 11111111의 계산

이러한 이유 때문에 byte에서 최대한으로 표현할 수 있는 데이터는 다음과 같습니다.

> (2진수)01111111
> 0(맨 앞이 1이면 음수이므로) + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 127

반대로 가장 작은 값은 맨 앞의 수만 1이고 나머지 양수들이 0인 경우가 됩니다.

> (2진수)10000000
> −128 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 = −128

4.2.4 long 타입은 정말 큰 숫자를 표현할 때에만 사용합니다.

정수를 나타내는 데 있어서 가장 큰 자료형은 long 타입입니다. 2의 64승까지에 해당하는 숫자를 표현할 수 있기 때문에 아주 어마어마하게 큰 숫자가 필요한 경우에 사용합니다.

4.3 소수를 나타내는 float, double

소수를 표현하기 위해서는 float과 double이 있지만 이를 구분하기 위해서 데이터 뒤에 d/D 혹은 f/F를 붙여서 명확히 해주는 것이 좋습니다.

ex) double d = 3.14D;
float f = 1.23F;

만일 아무것도 없는 소수 데이터를 선언하면 double 타입으로 간주합니다.

소수점 이하의 숫자를 표현하기 위해서는 다음 두 가지를 사용합니다.

- float: 4byte 소수를 표현
- double: 8byte 소수를 표현(기본)

소수는 32bit(4byte)인 float가 아닌 64bit(8byte)인 double 타입을 사용합니다. 굳이 이렇게 큰 메모리를 사용하는 기본적인 이유는 소수는 정밀한 연산을 처리하기 때문입니다. 정수의 경우에는 거의 우리가 실생활에서 사용하는 값이 int라는 타입의 범위에 포함되기 때문에 큰 문제가 없지만, 소수는 남들에게 이 값을 사용하는 변수가 double로 선언되어 있는지, float으로 선언되어 있는지를 구분할 수 있게 double일 경우에는 D 혹은 d, float일 경우에는 f, F를 붙여줍니다. 숫자 뒤쪽에 아무런 표시가 없으면 기본인 double 타입입니다.

예제 | double과 float을 이용해서 소수점 선언하기

```
public class VarTest4 {
	public static void main(String[] args) {
		float f = 10.0F;
		double d = 1.2345;

		System.out.println(f);
		System.out.println(d);
```

```
        }
}
```

```
10.0
1.2345
```

■ **소수는 데이터 저장 방식이 다릅니다.**

bit에는 0과 1만 들어갈 수 있습니다. 그리고 각 bit 안에 들어간 숫자는 2의 n 승과 결합해서 표현됩니다. 그런데 곰곰이 생각해 보면 bit 안에 들어가는 데이터를 가지고는 소수점을 표현할 수 없다는 사실을 발견할 수 있습니다. 2의 0승부터 시작해도 역시 정수일 뿐입니다. 이 때문에 double과 float의 경우는 데이터를 저장하기 위해서 특별한 방식으로 데이터를 저장합니다. 조금 어려운 용어를 써야 할 듯합니다. Java의 소수점 연산은 IEEE 754라는 표준을 따라 연산됩니다.

참고

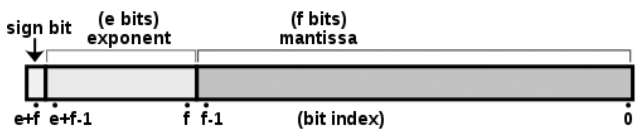
IEEE 754

IEEE 754는 컴퓨터에서 부동소수점을 표현하는 가장 널리 쓰이는 표준이다. ±0 등의 수와 무한, NaN 등의 기호를 표시하는 법과 이러한 수에 대한 연산을 정의하고 있다.

IEEE 754에는 32비트 단정도(single-precision), 64비트 배정도(double-precision), 43비트 이상의 확장단정도(거의 쓰이지 않음), 79비트 이상의 확장배정도(일반적으로 80비트로 구현됨)에 대한 형식을 정의하고 있다. 이 중 32비트 단 정도는 반드시 구현해야 하며, 다른 형식은 선택사항이다. 많은 프로그래밍 언어에서 IEEE 표준을 따르도록 정의하고 있다. 예를 들어 C에서는 float는 단정도, double은 배정도와 대응된다.

구조

IEEE 754의 부동 소수점 표현은 크게 세 부분으로 구성되는데, 최상위 비트는 부호를 표시하는 데 사용되며, 지수 부분(exponent)과 가수 부분(fraction/mantissa)이 있다.



출처: 위키 백과(http://wikipedia.or.kr)

	32비트 부동 소수점	64비트 부동 소수점
부호	1bit	1bit
바이어스된 지수	8bit	11bit
가수	23bit	52bit

표

위의 표를 보시면 바이어스된 지수와 가수라는 것이 보입니다. 간단히 설명드리자면 주어진 데이터를 2진수의 형태로 나타낸 후에 연산을 통해서 데이터를 가지는 숫자(가수)와 데이터를 제대로 표현하기 위해서 연산하는 숫자(지수)로 만들어서 따로 보관합니다. 우선 기본 개념만 설명하자면 데이터와 소수점 이하를 분리하는 방법이라고 할 수 있습니다. 예를 들자면 다음과 같은 방식으로 저장됩니다.

0.1×10^{-5} → 가수: 1, 지수: −5, 밑수: 10(실제로는 2진수이므로 2)

4.4 참, 거짓을 나타내는 boolean

Java에서 '참, 거짓' 연산은 다른 언어들과 달리 0/1을 사용하지 않고 무조건 true/false라고만 표현합니다.

boolean은 단순하게 true 혹은 false 값만을 가집니다. 주로 상황에 따라서 다르게 동작해야 할 필요가 있는 제어문 같은 곳에서 많이 사용됩니다. 주의할 점은 C 언어에서는 0, 1이라는 숫자가 true, false를 의미했지만, Java에서는 명확히 true, false로 사용한다는 점입니다. boolean은 변수 그 자체의 선언보다는 특정한 로직 제어 시에 더 많이 사용됩니다. 예를 들어 홀, 짝수의 구분이라든가 데이터 값의 검증이 성공했는지의 여부처럼 제어의 판단이 되는 경우에 많이 사용됩니다.

예제 | boolean을 선언하고 사용하기

```
public class BooleanTest {
	public static void main(String[] args) {
		boolean b = true;
		boolean c = false;
		System.out.println( "  B :" + b );
		System.out.println( "  C :" + c );
	}
}
```

```
B :true
C :false
```

4.5 글자(문자) 하나를 나타내는 char

char는 알파벳 글자 하나, 한글 문자 하나를 의미한다고 생각하면 쉽습니다. 다만, 주의할 것은 char를 이용할 때는 작은따옴표('')를 이용해서 표시한다는 것과 2byte로 데이터를 표현한다는 사실입니다.

서유럽의 알파벳 기반 문자들의 경우에는 기호 하나에 숫자 값을 하나 주면 byte의 범위 내의 값으로 충분히 글자 하나와 숫자 하나를 표현할 수 있습니다(예를 들어 A라는 글자는 65와 같이 하나씩 연결하는 방식). 하지만, 한국어나 중국어, 일본어와 같은 문자들은 하나의 byte에 글자를 표현하기에는 부족하기 때문에 2byte 이상의 공간에 글자를 표현합니다.

예제 | 글자 하나를 의미하는 char

```
public class CharEx {
	public static void main(String[] args) {
		char a ='A';
		char b ='가';

		System.out.println( "  a :" + a );
		System.out.println( "  b :" + b );
	}
}
```

```
a :A
b :가
```

5 변수를 선언할까 말까?: 변수를 선언하는 판단 기준 가이드

여러분이 프로그래밍이 익숙하지 않다면 변수 대신에 직접 코드에 값을 넣는 경우가 많습니다. 대신 나중에 수정하거나 데이터의 수치가 변경되면 변수로 작업하지 않은 것을 후회하게 되기 때문에 이참에 변수를 선언하는 기준이 되는 몇 가지 상황을 정리해볼까 합니다.

- 매번 다른 데이터를 사용할 때
- 연산한 결과를 보관할 때
- 한 번 선언한 데이터를 여러 곳에서 사용할 때

5.1 프로그램을 실행하는 사람의 입력에 의해 매번 값이 바뀔만한 부분

예를 들어 "우주왕복선의 비행 궤도를 구하라.", "미사일의 탄도를 예상하라.", "신형 자동차의 연비를 구하라." 등의 이런 거창한 제목들이 바로 프로그램을 만드는 이유입니다. 아주 힘들고 복잡한 작업을 컴퓨터에게 대신시키자는 뜻입니다. 그런 로직은 한 번만 만들어 두고 매번 바뀌는 데이터의 값에 따라서 다른 결과를 얻어야 합니다. 사용자가 매번 다른 값을 입력하고 그 값에 따라서 만들어진 프로그램을 그때마다 변경한다면 무척 귀찮은 작업입니다. 따라서 이런 경우에는 변수를 하나 선언해서 사용하는 방식이 편리합니다. 조금 뒤에 이런 예제를 하나 만들어 보겠습니다.

프로그램을 만들다 보면 사용자의 입력 값을 하나의 변수로 받은 후에 그 변수를 이용해서 여러 곳에서 추가적인 작업을 수행하는 경우가 많으므로 보통 프로그램을 사용하는 사용자들의 입력 데이터는 별도의 변수로 선언하는 경우가 많습니다.

예제 매번 도형의 길이와 높이 값이 달라져야 한다면 변수를 고려합니다.

```
public class VarNeed1 {
        public static void main(String[] args) {
                int width = 100;
                int height = 300;

                System.out.println("이 도형이 사각형일 경우");
                System.out.println(width * height);
                System.out.println("이 도형이 삼각형일 경우");
                System.out.println( ( width * height ) / 2);
        }
}
```

앞의 코드에서는 width와 height의 값은 사용자가 입력해서 사용할 가능성이 큽니다. 즉 원하는 데이터가 자주 변경될 것이라는 판단이 서면 변수로 만드는 습관을 가지시는 것이 좋습니다.

5.2 어떤 실행 결과를 저장하고 싶을 때

예를 들어 1에서 10까지의 전체 합을 구하는 경우를 생각해 보겠습니다. 여러분은 계산하기 위해서 종이에 임시로 결과를 적거나, 손가락을 이용하거나 머리가 좋으신 분들은 머릿속에서 순간순간 결과를 임시로 보관하려 할 겁니다. 프로그래밍에서도 마찬가지입니다. 무언가 연산을 하거나 계산을 하는 데 있어서 잠시 그 결과를 보관하고 싶을 때 변수를 사용합니다. 즉 연산한 결과의 임시 보관소를 변수라고 생각하시면 편합니다.

```
sum = 1 + 2 + 3 + 4 + 5
total = sum + 1000
```

위와 같은 경우라면 sum은 연산한 결과를 잠시 보관해 두고, 1000이라는 값을 이곳에 추가로 연산할 때의 축약형으로 사용되었습니다.

5.3 한번 만들어진 값이 여러 곳에서 사용될 듯하면 변수를 이용

앞에서 설명한 내용과 비슷하긴 하지만 결과적으로 변수는 하나의 데이터를 여러 곳에서 사용하다가 쉽게 변경할 수 있게 하기 위해서 선언하고 사용한다는 것이 가장 정석적인 해석입니다. 따라서 여러분이 10이라는 값을 여러분의 코드 내에서 여러 번 써야 한다고 판단하시면 바로 변수를 선언해 주는 것이 좋습니다. 변수는 데이터를 변경할 때 한 곳만 고치기 위해서 사용한다는 점을 반드시 기억하시기 바랍니다.

6 형 변환(Type Conversion): 데이터를 다른 종류의 상자로

변수에서 가장 중요한 내용은 기본 자료형의 종류와 형 변환입니다. 형 변환이란 어떤 데이터를 다른 타입의 데이터로 취급하는 방법입니다. 예를 들어 숫자를 문자로 간주하거나, 문자를 숫자로 간주해야 할 필요가 있을 때 형 변환을 하게 됩니다. 좀 더 쉽게 이해하시려면 변수를 하나의 상자로 생각하면 좋습니다. 즉 어떤 상자의 내용물을 다른 종류의 상자로 옮기는 것이라고 생각하시면 좋을 듯합니다. 형 변환이라고 할 수 있는 것은 주로 다음과 같은 것을 의미합니다.

• 하나의 데이터를 크기가 작거나 큰 상자로 복사할 때

```
int a = 10;
long b = a;
```

• 특정 타입의 데이터를 다른 타입의 데이터로 사용할 때

```
char c = 'A';
int i = c;
```

• 연산의 결과를 다른 타입의 데이터로 사용할 때

6.1 자동(묵시적) 형 변환(Automatic Type Conversion)

형 변환은 다시 두 가지로 나눠 볼 수 있는데, 자동(묵시적) 형 변환과 명시적 형 변환이 그것입니다.

자동 형 변환은 개발자가 별도의 작업을 하지 않아도 알아서 처리된다는 뜻입니다. 즉 JVM 내에서 알아서 해석되는 변환 작업이라고 생각할 수 있습니다. 가장 쉽게 생각해 보면 1G 메모리 카드에 있는 데이터를 2G 용량의 메모리 카드로 옮기는 작업을 할 때는 특별히 별도의 작업을 하지 않아도 되는 것과 비슷한 원리입니다.

자동 형 변환에 대해서 알아보도록 합시다. 이를 위해 다시 한번 변수의 크기를 유심히 보실 필요가 있습니다.

사용되는 바이트의 크기	기본 자료형
1byte (8bit)	byte
2byte	short, char
4byte	int, float
8byte	long, double

표

작은 상자에서 큰 상자로 옮긴다는 표현은 간단하게 다음과 같은 코드를 생각하시면 됩니다.

예제

```
public class CastingTest {
        public static void main(String[] args) {
                //작은 상자의 내용물을 큰 상자로 복사
                byte b = 10;
                int i = b;
                System.out.println(i);
        }
}
```

byte 타입의 변수 b는 10이라는 값을 담고 있고, int 타입 변수 i는 b를 담고 있습니다. 다른 것은 생각하지 말고 상자의 크기만 보겠습니다.

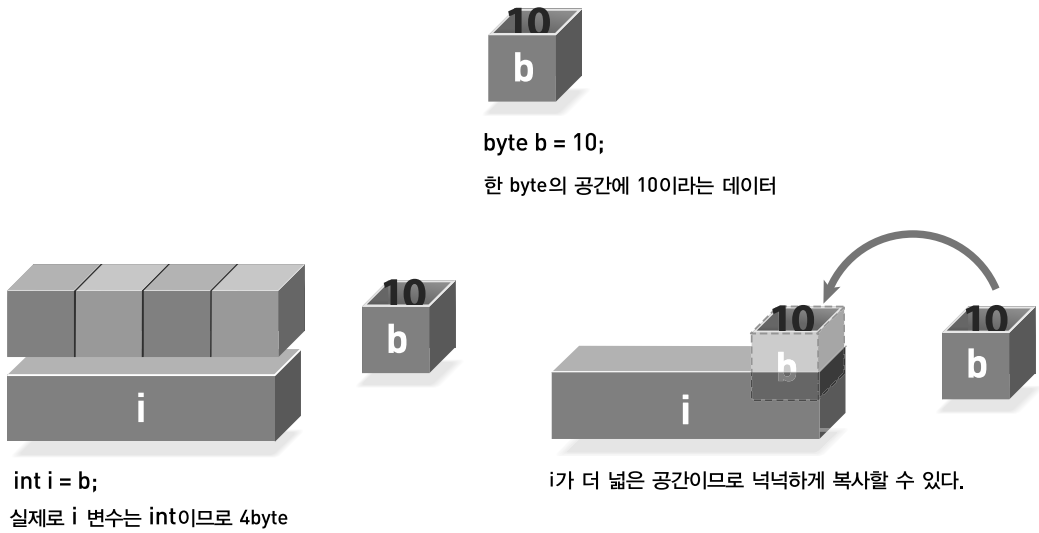


그림 14 묵시적 형 변환

그림에서 보는 바와 같이 공간이 넓은 상자에 작은 상자의 내용물을 옮길 때에는 아무런 제약이 없습니다. 작은 상자의 내용물을 큰 상자에 넣으니 당연히 원래의 값을 잃어버리는 일은 발생하지 않습니다. 다음 예제를 보면서 설명합니다.

예제

```
//자료형이 다른 데이터
char c = 'A';
int j = c;
System.out.println(j);
```

c라는 이름의 변수는 알파벳 문자 'A'를 담고 있습니다. 밑의 라인의 코드를 보면 j라는 숫자 타입의 변수가 c를 담고 있습니다. 여기서 중요하게 보셔야 하는 것은 상자의 크기입니다. char는 2byte의 상자이고, int는 4byte의 상자이므로 작은 상자(c)를 큰 상자(j)에 넣을 수 있게 됩니다.

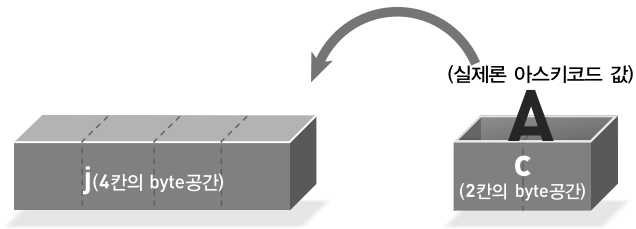


그림 15 작은 상자에서 큰 상자로

그럼 여기서 원래의 char c의 값은 어떻게 되었을까요? int는 당연히 정수 숫자만을 의미하기 때문에 'A'라는 글자가 화면에 찍히지 않습니다. 위의 코드를 실행해보면 결과는 다음과 같이 나옵니다.

```
65
```

결과가 이상한 값이 나온다고 생각하시죠? 위의 결과는 A라는 글자에 대한 ASCII 코드 값입니다. ASCII 코드라는 말은 모든 프로그래밍에서 많이 나오는 말이기 때문에 알아두시면 많은 도움이 됩니다.

참고

아스키(ASCII) 또는 미국 정보 교환 표준 부호(American Standard Code for Information Interchange)는 영문 알파벳을 사용하는 대표적인 문자 인코딩이다. 아스키는 컴퓨터와 통신 장비를 비롯한 문자를 사용하는 많은 장치에서 사용되며, 대부분의 문자 인코딩이 아스키에 기반을 둔다.

아스키는 1967년에 표준으로 제정되어 1986년에 마지막으로 개정되었다. 아스키는 7비트 인코딩으로, 33개의 출력 불가능한 제어 문자들과 공백을 비롯한 95개의 출력 가능한 문자들로 이루어진다. 제어 문자들은 역사적인 이유로 남아 있으며 대부분은 더 이상 사용되지 않는다. 출력 가능한 문자들은 52개의 영문 알파벳 대소문자와, 10개의 숫자, 32개의 특수문자, 그리고 하나의 공백 문자로 이루어진다.

아스키가 널리 사용되면서 다양한 아스키 기반의 확장 인코딩들이 등장했으며, 이들을 묶어서 아스키라고 부르기도 한다. 대표적으로 7비트 인코딩을 유지한 ISO/IEC 646과, 원래 아스키코드 앞에 비트 0을 넣어 8비트 인코딩을 만든 IBM 코드 페이지와 ISO 8859가 있다. 이 인코딩들은 언어 군에 따라 같은 숫자에 서로 다른 문자가 배당된 경우가 많다.

10진	문자	10진	문자	10진	문자	10진	문자
0	NUL	32	SP	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	“	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p

17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	←	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	→	94	^	126	~
31	US	63	?	95	_	127	DEL

표 아스키 코드표

자동 형 변환이라는 것은 결국 원래 데이터가 들어 있는 작은 상자를 보다 큰 상자에 넣는 행위를 의미합니다. 새로 담으려고 하는 상자가 원래 큰 상자이기 때문에 원래의 값이 그대로 유지되며, 별도의 작업을 하지 않아도 됩니다.

6.2 명시적 형 변환(Explicit Type Conversions): Casting

명시적 형 변환은 '큰 용량의 상자에서 작은 용량의 상자로 옮기는 작업'으로 이해하면 됩니다. 즉 옮길 곳이 더 작은 공간이기 때문에 데이터를 잃어버릴 수 있습니다. 따라서 개발자가 이 사실을 정확히 알 수 있도록 컴파일 시에 알려줍니다.

명시적 형 변환은 큰 상자(큰 범위의 타입) 안의 내용물을 작은 상자(작은 범위의 타입) 쪽으로 담는 것을 의미합니다. 흔히 이 작업을 캐스팅(Casting)이라고 하기도 합니다.

```
int x = 1;
byte y = x;
```

x는 int 타입의 변수이므로 4byte(4칸의 상자)를 차지하는 큰 상자입니다. 반면에 y는 byte 타입의 변수이므로 1byte(1칸의 상자) 공간의 크기입니다. 즉 1이라는 값은 큰 상자에 우선 담은 다음에 y라는 작은 상자로 옮기려고 하는 것을 의미합니다.

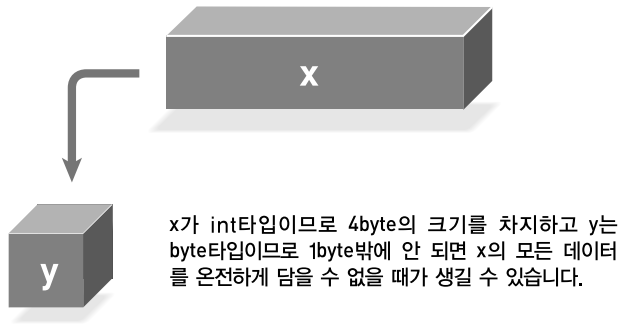


그림 16 큰 상자의 데이터를 작은 상자로

원래 큰 상자 안에 있는 내용물을 작은 상자로 넣으려면 어떤 방법을 써야 할까요? 가장 무식하면서도 단순한 해결책이 하나 있습니다. 그것은 꾸깃꾸깃 꾸겨서 넣는 방법입니다. 아니면 잘라내는 방법이 있습니다. 프로그래밍에서 처리 방법은 이 두 가지 방법 중에 후자의 방법에 더 가깝습니다. 어떤 방법을 선택하든 간에 뭔가 추가적인 작업은 반드시 필요하다는 얘기입니다. 이럴 때에는 코드를 이렇게 만들어 주어야 합니다.

예제

```
int x = 1;
byte y = (byte)x;
```

x 앞에 ( )를 이용해서 byte 부분만큼의 상자로 줄이는 겁니다. 이렇게 줄이는 것을 명시적 형 변환이라고 합니다(간단히 캐스팅이라고 하면 주로 명시적 형 변환을 말하는 겁니다).

■ 명시적 형 변환은 데이터가 잘릴 수 있습니다.

명시적 형 변환에서 알아 두셔야 할 점은 사항은 바로 데이터가 잘려나간다는 개념입니다.

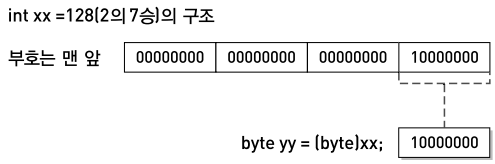
예제 | 형 변환하면서 원래의 데이터가 변경되는 경우

```
public class DownCastingEx {
```

```
        public static void main(String[] args) {
                int xx = 128;
                byte yy = (byte)xx;
                System.out.println(yy);
        }
}
```

```
-128 ← 128이라는 데이터가 변질됩니다.
```

int 타입으로 만들어진 변수 xx는 128이라는 값을 정상적으로 저장하고 있습니다. 다음 라인에서 yy는 byte로 더 작은 크기의 상자이므로 (byte)xx로 xx를 줄여낸 다음 처리하고 있습니다. 큰 상자의 것을 작은 상자로 줄여서 넣었기 때문에 문법적으로 위의 코드는 전혀 문제가 없습니다. 그런데 문제는 값에 있습니다. xx라는 변수는 128이라는 값을 가집니다. int는 4byte의 구조이므로 아래의 그림처럼 데이터가 차 있습니다.



작은 공간으로 옮기고 나니 2의 7승이지만 앞자리가 1이므로 음수가 됩니다.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	0	0	0	0	0	0	0
-128 +	0 +	0 +	0 +	0 +	0 +	0 +	0

결과 -128

그림 17

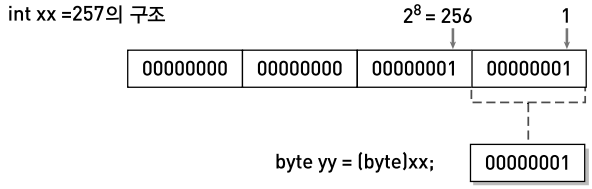
앞에서 2진법을 공부하면서 byte라는 타입이 가질 수 있는 값의 범위가 -128에서 127까지만 되는 이유를 설명해 드렸을 겁니다. 생각해 보시면 (byte)처럼 상자를 줄이게 되면 결국에 남는 데이터는 마지막 1byte만을 취하게 됩니다. 좀 더 쉽게 도마뱀 꼬리를 생각하시면 됩니다. 원래 몸통까지 있었는데 위급해 지면서 꼬리를 끊어 내듯이 마지막 1byte 데이터만 끊어 내서 생각해 보겠습니다. 몸통에서 필요한 마지막 byte만을 구성해 보면 위와 같습니다. 그러면 위의 값을 화면에 찍으면 어떻게 될까요? 예전에도 언급했지만 맨 앞에 나오는 숫자 1은 128이라는 값인 동시에

부호가 된다고 말씀드린 적이 있습니다. 따라서 -128이라는 값만이 남아 있게 됩니다. 한 번만 더 연습해보도록 합니다.

예제

```
int xx = 257;
byte yy = (byte)xx;
System.out.println(yy);
```

이전의 코드를 살짝 수정해 보았습니다. 그럼 또다시 바이트 안쪽 구성을 살펴볼까요?



출력 결과는 1

32개의 데이터 중에서 뒤의 8개만 담다보니
데이터를 읽어버리는 사태가 발생합니다.

그림 18 잃어버리는 데이터

값이 257이기 때문에 1byte의 범위를 넘어선 데이터이고 맨 앞에는 256(2의 8승)에 해당하는 값이 쓰여 있을 겁니다. 그리고 1에 해당하는 값은 2진수의 맨 마지막에 쓰여 있을 겁니다. 이런 상태에서 8개의 bit만 끊어서 구성하면 결국 남은 데이터는 1에 불과합니다. 따라서 화면에는 1만 출력됩니다.

정리해보면 결국 다운 캐스팅(Down Casting)이라는 것은 큰 상자에서 작은 상자로 데이터를 옮기는 동안 자신에 필요한 크기로 잘라낸다는 개념입니다. 그리고 잘라내게 되면 원하지 않는 데이터가 들어갈 수 있는데 이 이유는 바이트의 데이터가 부분부분 잘려나가면서 캐스팅되기 때문입니다.

큰 범위 타입(long이나 double)의 변수를 작은 범위 타입의 변수로 처리하게 되면 원래의 bit 공간 안에 들어 있는 데이터를 잃어버릴 수 있습니다. 따라서 개발자가 반드시 이 상황을 알 수 있도록 '( )'를 이용해서 처리해야 합니다.

6.3 상자의 크기가 같더라도 변수의 타입이 다른 경우

상자의 크기(몇 바이트인지)가 같다면 다음으로 고려해야 하는 것은 상자의 유형입니다. 즉, 같은 크기라고 해도 char(2byte)는 글자를 표현하는 데에만 사용하고, short(2byte)는 숫자를 표현하기 위해 사용합니다.

예제

```
//같은 크기
char c1 = 'A';
short s1 = c1; //컴파일 불가
float f1 = 13.3F;
int i1 = f1; //컴파일 불가
```

위의 코드는 컴파일되지 않는 코드입니다. char 타입은 2byte의 공간을 차지하는 상자이고, short 역시 2byte의 크기를 차지하는 상자이므로 내용물을 옮기는 데에는 아무 이상이 없지만, 같은 크기의 상자라 하더라도 타입이 완전히 달라지면 캐스팅 작업이 필요합니다.

> 변수의 타입이 달라지면 원래의 데이터를 잃어버릴 수 있다는 것을 생각해 보면 어째서 개발자가 명시적 형 변환을 할 때마다 확인해야 하는지를 이해할 수 있습니다. 즉 데이터의 변질이라는 근본적인 현상을 이해하면 됩니다. 원래의 데이터가 다른 타입으로 변하면서 개발자가 의도하지 않았던 데이터의 변화가 있을지도 모를 때에는 명시적인 형 변환을 해야 합니다.

■ **소수(double, float)는 주의해야 합니다.**

형 변환을 설명하면서 이 부분을 어떻게 설명해야 하나 많이 고민했습니다. 우선은 대부분의 기초적인 서적에서는 이런 부분을 그냥 '이러면 이렇게 됩니다'와 같은 방식으로만 설명했기 때문입니다. 하지만, 알아두면 나쁠 것은 없다고 생각해서 설명하도록 하겠습니다.

예제

```
//조심하세요.
long lval = 1234L; //(8byte)
float fval = lval; //(4byte)
System.out.println(fval);
```

상자의 크기만으로 본다면 lval은 long 타입이고 8byte의 크기입니다. 조금 더 명확하게 붙여주기 위해서 L을 붙여주었습니다. 다음에 나오는 fval은 float이므로 4byte입니다. 만일 우리가 배운 이론 대로 라면 위의 선언은 잘못된 것이므로 컴파일할 수 없는 코드입니다. 하지만, 실제로 위의 코드는 전혀 문제가 없는 코드입니다. 큰 상자의 내용물을 작은 상자에 넣는데 아무런 조치를 하지 않아도 문제가 없는 이유는 소수일 때는 데이터를 보관하는 데 있어서 일종의 연산 작업이 이루어지기 때문입니다. 즉 double, float는 데이터를 '000000'과 같은 방식으로 표현하는 것이 아니라, 연산의 결과로 '부호 + 지수 + 가수'의 형태가 되므로 일반적인 형 변환의 법칙에서는 조금 예외라고 생각하시면 됩니다.

6.4 자료형이 바뀌는 경우: 정수와 소수의 연산은 소수

소수를 나타내는 double과 float을 사용하는 경우에 값이 자동으로 바뀔 수 있는 상황이 꽤 빈번합니다. 간단한 예를 들어 보겠습니다.

```
int x = 10;
int y = 3;
```

만일 x 값 10을 y 값 3으로 나누게 되면 어떻게 될까요? 우리의 상식으로는 소수 3.3333...의 값이 나온다는 사실을 알고 있습니다. 참고로 나중에 배우겠지만, 나눗셈 연산은 '/'를 이용합니다.

예제

```
int x = 10;
int y = 3;
System.out.println(x/y);
```

```
3
```

컴퓨터는 기본적으로 멍청하고 바보입니다. 정수로 된 숫자와 정수 숫자의 연산은 당연하게도 숫자가 나올 것이라고 짐작합니다. 컴퓨터는 정수와 정수의 연산은 무조건 정수로 연산됩니다. 따라서 제대로 연산하려면 아래의 규칙을 알아 둘 필요가 있습니다.

• 정수와 소수의 연산은 소수로 자동 변환된다.

앞의 코드에서 y를 정수가 아닌 double 타입의 값으로 변경해 보도록 합니다.

예제

```
public class PromotionTest {
        public static void main(String[] args) {
                int x = 10;
                double y = 3;
                System.out.println(x/y);
        }
}
```

```
3.3333333333333335
```

실행 결과에서 보는 것처럼 결과가 원하는 대로 소수로 연산됩니다.

7 만들어 봅시다: 화면에서 숫자 입력받기

변수의 개념을 조금 더 연습할 겸해서 화면으로부터 숫자를 입력받는 프로그램을 만들어볼까 합니다.

7.1 실행 시나리오

프로그램을 실행하면 다음과 같은 결과를 만들려고 합니다.

```
화면에 숫자를 입력해 주세요!
13 ← 입력 후 Enter
당신이 입력한 숫자는 13
```

■ 배운 내용 활용하기

그럼 위의 프로그램에서 우리가 배운 내용 중에 써먹을 수 있는 부분을 생각해 보겠습니다. 아직 배운 내용이 많지는 않지만 그래도 몇 가지는 활용할 수 있을 듯합니다.

• 사용자가 입력하는 부분을 받기 위해서는 변수를 하나 선언하는 것이 좋다.
• 숫자를 변수로 사용하기 위해서는 int라는 타입을 활용한다.
• System.out.println( )을 통해서 환영 메시지를 화면에 출력하거나 변수를 화면에 출력한다.

■ **배우지 않은 내용 찾아내기**

그럼 우리가 아직 배우지 않은 내용은 무엇이 있을지 생각해 보겠습니다.

• 화면에서 사용자가 키보드에 입력하는 글자를 입력받는 방법을 모른다.

키보드로 입력받는 부분을 만들어 내기 위해서 Scanner라는 특이한 기능을 사용할 겁니다. 아직은 여러분이 어떤 기능인지 자세히 아실 만한 건 아니지만 간단하게 예제를 하나 만들어 보도록 하겠습니다.

예제 | 사용자의 입력을 처리하는 UserInputTest

```
import java.util.*;

public class UserInputTest {
        public static void main(String[] args) {
                Scanner s = new Scanner(System.in);
                System.out.println("화면에 숫자를 입력하세요.");
                System.out.println(s.nextInt());
        }

}
```

```
화면에 숫자를 입력하세요.
123 ← 사용자 입력
123
```

Scanner는 JDK1.5 이상에서만 사용할 수 있는 기능입니다. JDK1.4 이하에서는 Scanner 기능을 사용하실 수 없습니다. Scanner를 사용하려면 다음과 같은 단계를 거칩니다.

1_ 소스의 맨 위에 'import java.util.*;' 구문을 추가합니다(이것은 Scanner 등의 특별한 기능을 모아둔 것을 이 소스에서 사용한다는 의미입니다(전문적으로 패키지(Package)라고 합니다.)

2_ 'Scanner scanner = new Scanner(System.in);'이라는 구문을 추가합니다. 유심히 보실 부분은 System.in입니다. 기존에는 System.out.println( );을 이용했습니다. 이것이 화면에서 우리가 원하는 변수나 메시지를 화면으로 출력(output)하는 기능을 했었다면 System.in은 반대로 입력(input)기능이라는 것을 봐 두실 필요가 있습니다. 자세한 내용은 I/O(입출력 프로그래밍) 부분에서 좀 더 설명합니다.

3_ 실제로 사용자가 키보드로 입력하는 데이터를 처리하는 부분은 scanner.nextInt( )입니다. 이것은 키보드에서 입력되는 문자를 숫자로 만들어 주는 부분입니다. 보시면 nextInt( )라고 int라는 용어가 나온 것이 보입니다.

간단한 소스입니다만 이것을 조금만 변경해서 완성해 보겠습니다.

7.2 만들어보기

예제

```
import java.util.*;

public class GetUserNumber {
        public static void main(String[] args) {
                Scanner scanner = new Scanner(System.in);
                System.out.println("화면에 숫자를 입력해 주세요.");
                int input = scanner.nextInt();
                System.out.println("당신이 입력한 숫자는" + input);
        }
}
```

```
화면에 숫자를 입력해 주세요.
1234 ← 사용자 입력
당신이 입력한 숫자는 1234
```

아직은 단순히 숫자에 대한 처리만 가능하므로 문자를 입력하면 에러를 확인하게 될 겁니다. 앞으로 학습을 진행해 나가면서 좀 더 완전한 프로그램을 작성해보도록 합시다.