Chapter 03

구구단 게임

1. 이번에 만들 코드

- 우리가 이번에 만들 게임은 컴퓨터가 구구단 문제를 내고 플레이어(사람)가 답을 맞히는 게임이다.
- 앞에서 만든 게임과의 차이는, 컴퓨터가 구구단 문제를 만들 때 사용하는 두 수 중 하나를 프로그램을 실행할 때 인수(아규먼트)로 줄 수 있다는 점이다.
- 컴퓨터는 주어진 인수와 난수로 만든 수를 곱해서 문제를 만들고, 플레이어가 키보드로 답을 입력하면 정답인지 오답인지를 출력한다.
- 만일 플레이어가 오답을 입력하면 정답을 알려준다.
- 이 프로그램을 통해서 우리는 자바의 연산자들과 인수를 받는 법을 배우고, 데이터형을 자유자재로 변환할 수 있는 방법을 배우게 된다.
- 자바는 C 언어나 C++ 언어와는 달리 데이터형 검사에 엄격하기 때문에 이번에 배우는 데이터형 변환 방법은 매우 중요하다.

2. 연산자

- 덧셈, 뺄셈 등의 연산을 할 때 사용하는 +, - 기호를 연산자라고 합니다.
- 자바에서는 수치 연산을 위한 연산자는 물론이고 비교, 조건, 증가, 감소, 대입 등의 다양한 기능을 하는 연산자들을 준비하고 있다.
- 연산자를 이용하면 명령어를 외우는 불편없이 쉽게 원하는 결과를 얻을 수 있기 때문에 편리하다.
- 그러나 아래 예에서도 볼 수 있듯이 각 연산자는 우선순위가 있기 때문에 주의해야 한다.

```
X = 12 – 3 * 4 + 2
```

- 위 식에서 답은 2 이다.
- 이유는 3*4를 먼저 계산한 후, 12 – 12 + 2를 하기 때문이다.
- 만일 12 -3을 먼저 계산했다면 답은 38 이 되겠지만 *은 +, - 보다 우선순위가 높기 때문에 먼저 계산한 것이다.

2. 연산자

- 자바의 모든 연산자는 우선순위와 연관성을 가지고 있다.
- 여러 연산자가 섞여있는 경우, 우선순위가 높은 것부터 차례대로 계산한다.
- 만일 같은 우선순위의 연산자가 사용된 경우에는 연관성에서 정한 순서대로 계산한다.
- 위의 예에서 우선순위가 높은 3 * 4가 먼저 계산되어 12 – 12 + 2가 된 후에는 –와 +의 우선순위가 같기 때문에 –와 +의 연관성에 따라 왼쪽부터 오른쪽으로 계산한다.
- 자바 연산자의 우선순위와 연관성은 아래 표에 나와 있다.
- 우선순위와 연관성 외에 자바의 연산자에서 반드시 알아야할 또 다른 점은 각 연산자의 연산대상이 될 수 있는 데이터형이 미리 정해져 있다는 점이다.
- 예를 들어 비트 연산지는 정수형만을 사용할 수 있다.
- 실수형이나 논리형의 값을 비트 연산자로 계산하려고 하면 에러가 발생한다.

2. 연산자

- 자바는 데이터형에 대한 제한이 엄격하다.
- 따라서 각 연산자는 미리 정해둔 데이터형으로만 계산할 수 있다.
- 만일 다른 데이터형인 변수를 계산하고 싶을 때는 데이터형 변환 방법을 이용해서 명시적으로 변환시켜야 한다.

우선순위	연관성	연산자	연산대상
1	왼쪽 →오른쪽	++	정수형, 실수형
		--	정수형, 실수형
		양수(+), 음수(-)	정수형, 실수형
		~	정수형
		!	논리형
		(캐스트)	모든 데이터형
2	왼쪽 →오른쪽	*, /, %	정수형, 실수형
3	왼쪽 →오른쪽	+, -	정수형, 실수형
		+	String

2. 연산자

우선순위	연관성	연산자	연산대상
4	왼쪽 →오른쪽	<<, >>, >>>	정수형
5	왼쪽 →오른쪽	<, <=, >, >=	정수형, 실수형
		instanceof	모든 데이터형
6	왼쪽 →오른쪽	==, !=	모든 데이터형
7	왼쪽 →오른쪽	&	정수형, 논리형
8	왼쪽 →오른쪽	^	정수형, 논리형
9	왼쪽 →오른쪽	*, /, %	정수형, 실수형
10	왼쪽 →오른쪽	+, -	정수형, 실수형
11	왼쪽 →오른쪽	|	정수형, 논리형
12	오른쪽 →왼쪽	?:	논리형과 모든 데이터형
13	오른쪽 →왼쪽	=	모든 데이터형
		*=, /=, %=	
		+=, -=	
		<<=, >>=, >>>=	
		&=, ^=, |=	

2. 연산자

산술(arithmetic) 연산자

- 수치 연산을 위해 자바에서 준비해둔 산술 연산자로는 더하기 (+), 빼기(-), 곱하기(*), 나누기(/), 그리고 나머지 (%) 연산자가 있다.
- 나누기 연산자는 주어진 항이 둘 다 정수형일 때는 정수 나누기를 하고, 둘 중 하나라도 실수형일 때는 실수 나누기가 된다.
- 예를 들어 20 / 2는 10 이 되지만, 20.0 / 2 나 20 / 2.0은 10.0 이 된다.
- 산술 연산자 중 특이한 연산자는 나머지 연산자(%) 이다.
- 나머지 연산자는 정수형 인수를 나누고 남은 값을 돌려준다.
- 예를 들어 10 % 3 의 결과는 1 이 된다.

연산자	의미
a + b	더하기
a – b	빼기
a * b	곱하기
a / b	나누기
a % b	나머지

2. 연산자

산술(arithmetic) 연산자

- 다음은 산술 연산자로 주어진 값의 각 자리의 값을 출력하는 예제이다.
- 예를 들어 451이라는 값은 1의 자리는 1, 10의 자리는 5, 100의 자리는 4라고 출력한다.

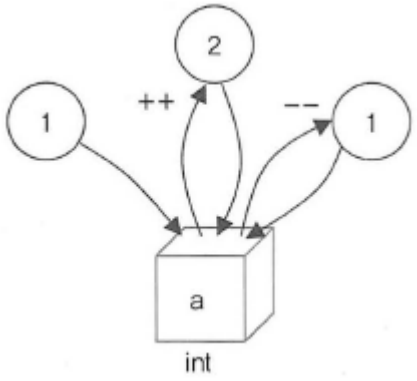
```
1   public class ArithmeticTest {
2       public static void main(String[] args) {
3           int num = 256;
4
5           System.out.println("주어진 수 : " + num);
6           System.out.println("  1의 자리: " + num % 10);          // 나머지 출력
7           num = num / 10;                          // num 값 변경
8           System.out.println(" 10의 자리: " + num % 10);
9           num = num / 10;
10          System.out.println("100의 자리: " + num % 10);
11      }
12  }
```

2. 연산자

증가(increment) 연산자와 감소(decrement) 연산자

- 증가 연산자는 변수의 앞이나 뒤에 붙어서 변수의 값을 하나 증가시킨다.
- 감소 연산자는 반대로 변수의 앞이나 뒤에 붙어서 변수의 값을 하나 감소시킨다.

연산자	의미
++	값을 1증가
--	값을 1감소



- 증가/감소 연산자에서 주의할 점은 이 연산자가 붙는 위치에 따라 증가/감소하는 시점이 다르다는 점이다.
- 증가/감소 연산자가 변수의 앞에 붙을 때는 변수의 값이 사용되기 전에 변수의 값이 1 증가/감소한다.
- 그러나 변수의 뒤에 붙을 때는 변수의 값이 사용된 후에 1 증가/감소한다.

2. 연산자

증가(increment) 연산자와 감소(decrement) 연산자

```
int a = 10;
int b = 10;
int x = a++;        // a는 11, x는 10
int y = ++b;        // b는 11, y는 11
```

- 예를 들어 위 식이 실행된 후의 a와 b 의 값은 모두 11 이지만, X 값은 10이고 y는 11이다.
- X가 10 인 이유는 a가 먼저 x 에 저장된 후에 1 증가되었기 때문이고, y가 11인 이유는 b가 먼저 1 증가되고 y에 저장되었기 때문이다.
- 복잡한 식에서 증가/감소 연산자를 사용하면 자칫 엉뚱한 결과가 나타나기 때문에, 될 수 있으면 간단한 식에서만 사용하거나 변수의 앞이나 뒤로 통일시키는 것이 좋다.

2. 연산자

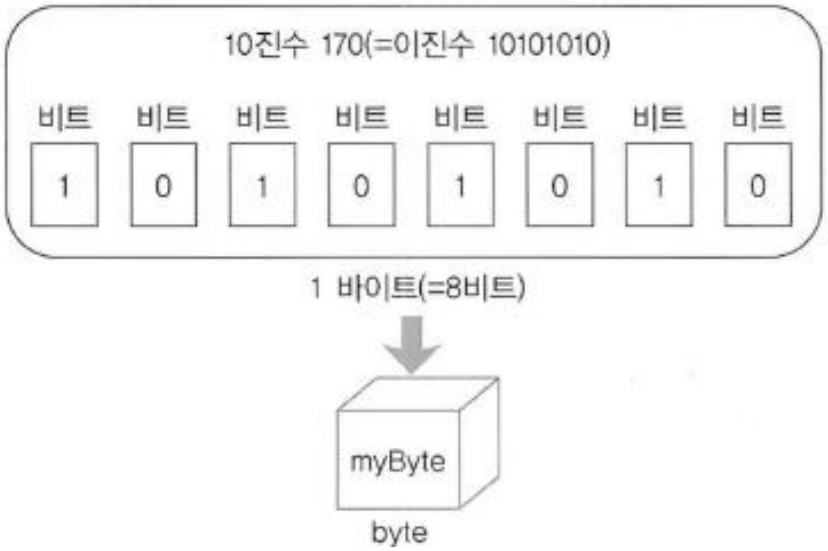
증가(increment) 연산자와 감소(decrement) 연산자

```
1   public class IncDecTest {
2       public static void main(String[] args) {
3           int x, y, z;
4
5           x = 10; y = 5; z = 0;
6           z = x++ - y-- + 1;       // ++나 --가 변수 뒤에 붙은 경우
7           System.out.println("x = " + x + ", y = " + y + ", z = " + z);
8
9           x = 10; y = 5; z = 0;
10          z = ++x - --y + 1;       // ++나 --가 변수 앞에 붙은 경우
11          System.out.println("x = " + x + ", y = " + y + ", z = " + z);
12      }
13  }
```

2. 연산자

비트(bitwise) 연산자

- 컴퓨터에서 다루는 정보는 전기적으로 On인 상태와 Off인 상태로 나타낼 수 있다.
- On을 1, Off를 0으로 표현한다면, 0과 1로 이루어진 2 진수 한 자리 값이 컴퓨터가 다루는 정보의 최소 단위이고, 이를 비트(bit) 라고 한다.
- 다음 그림은 10진수 170이 메모리 1바이트에 어떻게 저장되는지를 보인 것이다.



2. 연산자

비트(bitwise) 연산자

- 프로그래밍을 하다보면 비트 단위의 작업을 해야 할 경우가 있다.
- 비트 연산지는 1과 0으로 표현되는 각 비트 값을 AND, OR, XOR 연산하는데 사용하는 연산자이다.
- 다음의 표는 비트 a와 b를 AND, OR, XOR 연산한 결과이다.
- 다음의 표에서 알 수 있듯이, a & b는 a와 b가 모두 1 일 때만 1 이고, a | b는 a와 b가 모두 0 일 때만 0 이다.
- A ^ b는 a와 b가 같을 때는 0, 다를 때는 1 이다.

a	b	a & b (AND)	a | b (OR)	a ^ b (XOR)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

2. 연산자

비트(bitwise) 연산자

- 비트 값을 왼쪽이나 오른쪽으로 이동한 것을 쉬프트(shift) 라고 한다.
- 자바에서는 비트를 왼쪽이나 오른쪽으로 이동하는 쉬프트 연산지들을 준비하고 있다.
- << 연산자는 정해진 숫자만큼 왼쪽으로 쉬프트하고, >> 연산자는 정해진 숫자만큼 오른쪽으로 쉬프트한다.
- 예를 들어, a가 1 일 때 왼쪽으로 2, 오른쪽으로 1 만큼 쉬프트하면 다음과 같이 된다.

```
int a = 1;              // 0000 0000 0000 0001
int b = a << 2;         // 0000 0000 0000 0100
int c = b >> 1;         // 0000 0000 0000 0010
```

2. 연산자

비트(bitwise) 연산자

- 자바의 쉬프트 연산 중 특이한 점은 >>> 연산자이다.
- >>> 연산자는 기본적으로 >> 연산자와 같습니다만, 음수의 경우에는 다른 결과를 돌려주는 연산자이다.
- >> 연산자가 오른쪽으로 비트 값을 쉬프트할 때, 부호(sign) 비트를 왼쪽에 채워 넣는 반면, >>> 연산자는 0을 채운다.
- 따라서 -2를 >> 연산자로 1 만큼 쉬프트하면 -1이 되지만 >>> 연산자로 1만큼 쉬프트하면 양수의 엉뚱한 값(2147483647) 이 되어 버린다.

연산자	의미
a & b	논리곱 (AND)
a | b	논리합 (OR)
a ^ b	XOR
~a	보수
a >> b	a의 비트를 b만큼 오른쪽으로 이동(shift), 왼쪽은 a의 부호(sign) 비트로 채움
a << b	a의 비트를 b 만큼 왼쪽으로 이동(shift), 오른쪽은 0으로 채움
a >>> b	a의 비트를 b만큼 오른쪽으로 이동(shift), 왼쪽은 0으로 채움

2. 연산자

비트(bitwise) 연산자

```
1   public class BitwiseTest {
2       public static void main(String[] args) {
3           int x, y;
4           x = 8;          // 0000 0000 0000 0000 0000 0000 0000 1000
5           y = ~x;         // 1111 1111 1111 1111 1111 1111 1111 0111
6           System.out.println("x = " + x + ", y = " + y + " (~x)");
7
8           // 0000 0000 0000 0000 0000 0000 0000 0000
9           System.out.println(x + " AND " + y + " = " + (x & y));
10          // 1111 1111 1111 1111 1111 1111 1111 1111
11          System.out.println(x + " OR " + y + " = " + (x | y));
12          // 1111 1111 1111 1111 1111 1111 1111 1111
13          System.out.println(x + " XOR " + y + " = " + (x ^ y));
14
```

2. 연산자

비트(bitwise) 연산자

```
15          // 0000 0000 0000 0000 0000 0000 0010 0000
16          x = x << 2;
17          // 1111 1111 1111 1111 1111 1111 1101 1100
18          y = y << 2;
19          System.out.println("x = " + x + " (x<<2), y = " + y + " (y<<2)");
20
21          x = x >> 2;          // 0000 0000 0000 0000 0000 0000 0000 1000
22          y = y >> 2;          // 0011 1111 1111 1111 1111 1111 1111 0111
23          System.out.println("x = " + x + " (x>>2), y = " + y + " (y>>2)");
24
25          x = y >> 2;          // 1111 1111 1111 1111 1111 1111 1111 1101
26          y = y >>> 2;         // 0011 1111 1111 1111 1111 1111 1111 1101
27          System.out.println("x = " + x + " (y>>2), y = " + y + " (y>>>2)");
28      }
29  }
```

2. 연산자

비트(bitwise) 연산자

- **아래처럼 8을 보수 연산자(~)를 이용해서 보수로 바꾸면 – 8이 되어야 할 텐데, -8이 아니고 -9가 되었다. 이유가 무엇일까?**

```
...
x = 8;
y = ~x;
...
```

- 보수 연산자(~)는 비트 반전된 값을 구하는 단항 연산자이다.
- 정확히는 '1 의 보수(unary bitwise complement) 연산자'라고 부를 수 있다.
- 그런데 컴퓨터에서 음수(-)를 표현할 때는 2의 보수(binary bitwise complement)를 사용하고 있다.
- 2의 보수라는 것은 모든 비트를 반전시켜 '1의 보수'를 만든 후 1을 더한 것이다(1의 보수 + 1).
- 따라서 2진수 00001000 (10진수 8)의 1의 보수인 11110111을 2의 보수법으로 10진수로 바꾸면, -9가 된다.

2. 연산자

비트(bitwise) 연산자

- 아래처럼 8을 보수 연산자(~)를 이용해서 보수로 바꾸면 – 8이 되어야 할 텐데, -8이 아니고 -9가 되었다. 이유가 무엇일까?

```
...
x = 8;
y = ~x;
...
```

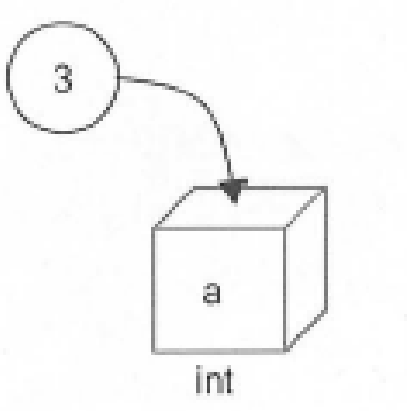
```
00001000 → 8
    ↓ 보수 연산
11110111 → 8의 보수 연산 결과

11110111 → 8의 보수 연산 결과
00001000
      +1
00001001 → 9(8의 보수 연산 결과의 '2의 보수')
```

- 따라서, 어떤 x 변수의 보수 연산한 결과는 (-x) – 1 이 되는 것이다.

2. 연산자

- **대입(assignment) 연산자**
  - 자바에서 기본적인 대입 연산자는 =이다.
  - 대입 연산자(=)는 다른 연산자와는 달리 주어진 인수를 처리해서 값을 돌려주지 않고, = 연산자 오른쪽에 있는 변수나 상수의 값을 왼쪽의 변수에 저장한다.



2. 연산자

대입(assignment) 연산자

- 대입 연산자는 앞에서 배운 산술 연산자와 조합해서 왼쪽 인수가 반복되는 의미로 사용되기도 한다.
- 예를 들어 a += b는 a=a + b와 같은 의미이다.
- 또한 앞에서 배운 비트 연산자와도 결합할 수 있다.
- 예를 들어 a &= b는 a = a & b와 같은 의미이다.

연산자	의미	연산자	의미
a = b	a에 b를 대입	a &= b	a = a & b
a += b	a = a + b	a |= b	a = a | b
a -= b	a = a – b	a ^= b	a = a ^ b
a *= b	a = a * b	a <<= b	a = a << b
a /= b	a = a / b	a >>= b	a = a >> b
a %= b	a = a % b	a >>>= b	a = a >>> b

2. 연산자

대입(assignment) 연산자

```
1   public class AssignmentTest {
2       public static void main(String[] args) {
3           int x, y, z;
4
5           x = y = z = 1;
6           z += x + y;                 // z = z + x + y와 동일
7           System.out.println("x = " + x + ", y = " + y + ", z = " + z);
8
9           x += y -= z = 5;                    // z = 5; y = y - z; x = x + y와 동일
10          System.out.println("x = " + x + ", y = " + y + ", z = " + z);
11      }
12  }
```

2. 연산자

논리(logical) 연산자

- 논리 연산자는 앞에서 배운 비트 연산자의 &, |, ~와 흡사하다.
- 차이점은 연산 대상이 논리형 이라는 점이다.
- 다음 표는 두 논리형 데이터 a와 b에 대한 &&, || , 연산의 결과이다.

a	b	a && b (AND)	a || b (OR)	!a
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

2. 연산자

논리(logical) 연산자

```
1   public class LogicalTest {
2       public static void main(String[] args) {
3           boolean x, y, z, b;
4
5           x = false;
6           y = z = true;
7           b = x || y && z;          // b = x || (y && z)와 동일
8           System.out.println("x = " + x + ", y = " + y + ", z = " + z + ", b = " + b);
9       }
10  }
```

2. 연산자

논리(logical) 연산자

- **자바는 최단 평가(Short Circuit Evaluation)를 한다고 한다.**
- **최단 평가라는 것은 어떤 것인가?**
  - 자바는 &&(AND)에서는 조건이 false(거짓)이거나 ||(OR) 에서 조건이 true인 경우에는 그 이후에 나오는 조건은 평가하지 않는데, 이를 최단 평가(Short Circuit Evaluation)라고 한다.
  - 예를 들어

```
(x>5) && (y<2) && (z==3)
```

  - 에서 x>5가 false라면 나머지 조건이 어떻든 항상 false가 된다. 또, 다음 예에서

```
(x>5) || (y<2) || (z==3)
```

  - X>5가 true라면 나머지 조건이 어떻든 항상 true가 될 수밖에 없다.

2. 연산자

논리(logical) 연산자

- 다음 예제는 if문에서 최단 평가를 하는 경우이다.
- 자바가 최단 평기를 하는 것을 모르면 큰 실수를 할 수 있다.
- 다음 예제의 8행에서 ++a하면 3이기 때문에 ++a > 3이 false가 되어 ++b는 아예 수행이 되지 않는다.
- 따라서 a의 값은 3으로 증가되지만, b의 값은 변화가 없다.

```
1   public class ShortCircuitTest {
2       public static void main(String[] args) {
3           int a = 2;
4           int b = 4;
5
6           if ((++a > 3) && (++b > 8)) {
7               System.out.println("if문이 수행되었습니다.");
8           }
9           System.out.println("a : " + a + ", b : " + b);
10      }
11  }
```

2. 연산자

관계 연산자

- 관계 연산자는 두 값의 크기를 비교하여 결과에 따라 true(참) 또는 false (거짓)을 돌려주는 연산자이다.
- 관계 연산자는 쉽게 이해 할 수 있지만 같음을 나타내는 == 연산자와 다름을 나타내는 != 연산자는 주의해야 한다.
- 앞에서 배운 대로 자바에서는 =를 오른쪽에 있는 변수나 상수의 값을 왼쪽 변수에 저장하는 대입 연산자로 사용하고 있기 때문에, 두 값이 같은지를 묻는 연산자로는 ==을 쓰고 있다.
- ==를 사용할 곳에서 실수로 =를 사용하면 엉뚱한 결과가 나올 수 있다.
- 다행히 자바는 연산자의 연산대상이 될 수 있는 데이터형에 대한 검사가 철저하기 때문에, 컴파일 과정에서 에러로 잡아준다.
- 하지만 =와 ==의 차이를 정확하게 알고 있는 것이 좋다.
- 다름을 나타낼 때는 논리 연산자에서 !(NOT) 연산자를 ==과 합쳐서 !=로 표시한다.

2. 연산자

관계 연산자

- 흡사 명령어처럼 생긴 instanceof 연산자는 클래스와 객체의 관계를 알고 싶을 때 사용한다.
- 우리는 앞장에서 Date 클래스로 today라는 변수를 생성했다.
- 이때 만일,

```
if (today instanceof Date){
    System.out.println("true");
}else{
     System.out.println("false");
}
```

- 라고 쓰면 값은 true가 된다.
- Instanceof는 객체가 어떤 클래스로 만든 객체(인스턴스)인지 알고 싶을 때 매우 유용한 연산자이다.

2. 연산자

관계 연산자

연산자	의미
a > b	a가 b보다 크면 true
a >= b	a가 b보다 크거나 같으면 true
a < b	a가 b보다 작으면 true
a <= b	a가 b보다 작거나 같으면 true
a == b	a와 b가 같으면 true
a != b	a와 b가 같지 않으면 true
a instanceof b	a객체가 b 클래스로 생성한 객체(인스턴스)이면 true

2. 연산자

관계 연산자

```
1   public class RelatedTest {
2       public static void main(String[] args) {
3           int x, y, z;
4           boolean b;
5
6           x = y = z = 1;
7           b = ((x-- > 0) || (++y != 0) && (--z == 0));          // 실행순서에 주의
8           System.out.println("x = " + x + ", y = " + y + ", z = " + z + ", b = " + b);
9       }
10  }
```

2. 연산자

조건 (conditional) 연산자

- 조건 연산자는 항을 3개 사용한다고 해서 삼항 연산자라고도 하는데, 조건문을 대신해서 시용할 수 있다.
- 다음처럼 조건 연산자는 대부분 대입 연산자와 함께 사용한다.

```
변수 = 조건 ? 값1 : 값2
```

- 이때 조건이 참이면 값1이 변수에 대입되고 거짓이면 값2가 변수에 대입된다.

연산자	의미
(a > b) ? a : b	(a > b)라는 조건이 true(참)이면 a를, false(거짓)이면 b를 수행

2. 연산자

조건 (conditional) 연산자

- 조건 연산자는 모두 분기문으로 바꿀 수 있다.

```
positive = (x > 0) ? true : false;
```

- 위의 조건 연산자는 다음처럼 분기문으로 바꿀 수 있다.

```
if (x > 0) {
    positive = true ;
}else {
    positive = false;
}
```

2. 연산자

■ 조건 (conditional) 연산자

```
1   public class ConditionTest {
2       public static void main(String[] args) {
3           int hour, min, sec;
4
5           hour = 13;
6           min = 30;
7           sec = 25;
8
9           String ampm;
10          ampm = (hour >= 12) ? "PM" : "AM";          // 오전/오후 결정
11           // 24시간 표기를 12시간 표기로 변경
12          hour = (hour >= 12) ? (hour - 12) : hour;
13          System.out.print(ampm + " " + hour + ":" + min + ":" + sec);
14      }
15  }
```

3. 형변환의 이해

- 우리는 앞장에서 자바의 데이터형에 대해서 배웠다.
- 자바에서 명령어를 사용할 때는 미리 정해진 데이터 형을 사용해야 하고, 다른 데이터 형을 사용하고 싶을 때는 명시적으로 형 변환을 해줘야 한다.
- 자바에 대해 잘 모르는 프로그래머들 중에는 '자바는 형변환을 반드시 해야 하기 때문에 불편하다'는 잘못된 인식을 가지는 경우가 많은데, 이는 형변환 방법을 제대로 공부하지 않았기 때문이다.
- 프로그래머가 명시적으로 형변환을 하도록 한 점은 자바의 단점이 아니고 장점이다.

3. 형변환의 이해

- 수십만 라인이나 되는 프로그램 중 어디선가 멋대로 형변환이 일어나 전체 프로그램의 결과가 엉뚱한 값이 되는 일은 흔한 일이다.
- 예를 들어 월급 계산을 하는 프로그램에서 double형인 금액을 int형에 저장하는 과정에서 소수점 이하가 잘리거나 크기 문제로 값이 변경되면 심각한 결과가 나타날 수 있다.
- 그래서 자바에서는 프로그래머가 의도적으로 형변환을 하지 않는 이상, 자동으로 형변환이 일어나는 것을 금지시킨 것이다.
- 프로그래머가 의도적으로 형변환을 하는 방법은 크게 두 가지가 있다.
- 기본 데이터형 간에 변환하고 싶을 때는 캐스팅이라는 방법을 쓰고 기본 데이터형과 클래스로 만든 객체간에 변환하고 싶을 때는 랩퍼(wrapper) 클래스를 사용한다.

3. 형변환의 이해

캐스팅

- 앞장의 키보드에서 입력받기 절에서 키보드로부터 1 문자를 읽는 프로그램을 만들었을 때 다음처럼 해서 에러가 났다.

```
...
char ch;
ch = System.in.read();
...
```

- 그 이유는 System.in.read( ) 메서드가 int형의 값을 돌려주기 때문에, int형 데이터를 char형인 ch에 저장할 수 없었기 때문이다.
- 다음처럼 캐스팅을 하면 해결된다.

```
...
char ch;
ch = (char) System.in.read();
...
```

3. 형변환의 이해

캐스팅

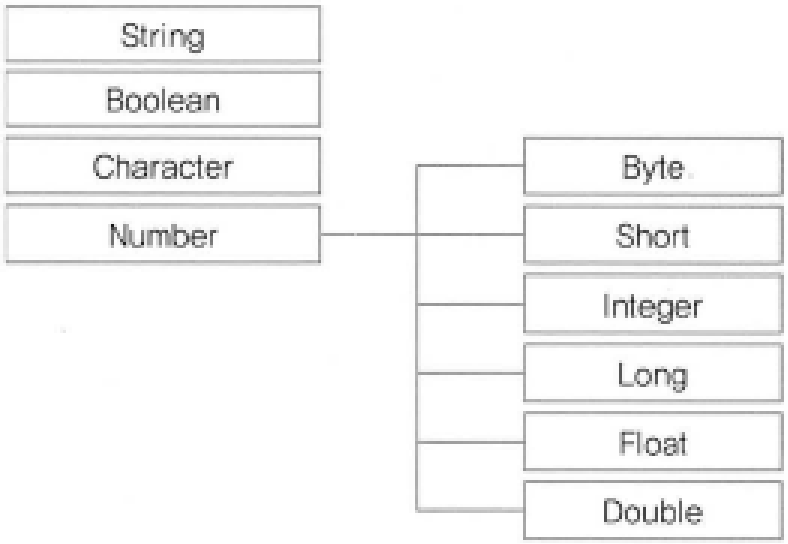
```
1   public class CastTest {
2       public static void main(String[] args) {
3           int myInt;
4           float myFloat = (float) 3.0;          // double형을 float형으로 변환
5           double myDouble;
6           myInt = (int) myFloat;  // float형을 int형으로 변환
7           myDouble = myInt;
8           System.out.print("myInt = " + myInt + ", myFloat = " + myFloat + ",
    myDouble = " + myDouble);
9       }
10  }
```

3. 형변환의 이해

랩퍼(wrapper) 클래스

- 자바에서는 기본 데이터형 외에 클래스를 생성해서 만드는 객체가 있다.
- 기본 데이터형 간의 형변환은 캐스팅을 사용하면 됩니다만, 기본 데이터형과 객체간에 데이터를 주고받을 때는 캐스팅을 사용할 수 없다.
- 이런 경우를 위해 미리 자바에서 준비해둔 것이 아래와 같은 랩퍼 (wrapper) 클래스이다.
- 아래 그림에서 String 클래스는 엄밀히 말해 랩퍼 클래스가 아니지만 랩퍼 클래스와 함께 쓰이는 일이 많기 때문에 함께 표시했다.

기본 데이터형	랩퍼 클래스
boolean	Boolean
byte	Byte
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double



3. 형변환의 이해

랩퍼(wrapper) 클래스

- 영단어 랩퍼 (wrapper)는 감싼다는 의미로, 랩퍼 클래스는 기본 데이터형을 감싸서 다른 데이터형으로 바꿀 수 있도록 해준다.
- 예를 들어 int 형 값을 다른 데이터 형을 바꾸고 싶다면, int형과 관련된 Integer 클래스를 사용해서 다음처럼 하면 된다.

```
int myInt= 10; // 사용할 Int형 데이터
Integer wrapInt= new Integer(myInt);    // int형 데이터 10으로 만든 Integer형 객체

byte myByte = wrapInt.byteValue();              // Integer형 객체 → byte형
short myShort = wrapInt.shortValue();           // Integer형 객체 → short형
int myInt = wrapInt.intValue();                 // Integer형 객체 → int형
long myLong = wrapInt.longValue();              // Integer형 객체 →  long형
float myFloat = wrapInt.floatValue();           // Integer형 객체 → float형
double myDouble = wrapInt.doubleValue();        // Integer형 객체 → double형
String myString = wrapInt.toString();           // Integer형 객체 → String형
```

3. 형변환의 이해

랩퍼(wrapper) 클래스

- 랩퍼 클래스에 있는 메서드들 중 중요한 메서드들은 다음의 표와 같다.
- 이 중 toString( )과 valueOf( )는 랩퍼 클래스와 String형 간의 변환을 위해 자주 쓰는 메서드로 자주 사용되기 때문에 꼭 알아두어야 한다.
- 예를 들어 float형 데이터를 String형으로 저장하거나 String형을 int형으로 저장하려면 다음처럼 하면 된다.

```
// float형 → String형
float myFloat= 12.34F;
Float wrapFloat= new Float(myFloat);    // float형 → Float형 객체
String myString= wrapFloat.toString( ); // Float형 객체 → String형 객체

// String형 → int형
String myString= "4225";
Integer wrapInteger= Integer.valueOf(myString); // String형 객체 →  Integer형 객체
int myInt= wrapInteger.intValue( );                    // Integer형 객체 → int형
```

3. 형변환의 이해

랩퍼(wrapper) 클래스

메서드	의미
toString( )	랩퍼 클래스가 가진 값을 String 형으로 변환
valueOf(String s)	String형 값을 해당 랩퍼 클래스의 객체로 변환
byteValue( )	랩퍼 클래스가 가진 값을 byte형으로 변환
shortValue( )	랩퍼 클래스가 가진 값을 short형으로 변환
intValue( )	랩퍼 클래스가 가진 값을 int형으로 변환
longValue( )	랩퍼 클래스가 가진 값을 long 형으로 변환
floatValue( )	랩퍼 클래스가 가진 값을 float형으로 변환
doubleValue( )	랩퍼 클래스가 가진 값을 double형으로 변환

3. 형변환의 이해

랩퍼(wrapper) 클래스

```
1   public class WrapperTest {
2       public static void main(String[] args) {
3           int myInt = 5316;
4           System.out.println("myInt = " + myInt);
5   // Integer wlpint = new Integer(myInt);   String myString = wlpInt.toString();
6           // int형 -> Integer형 객체 -> String형 객체
7           String myString = Integer.toString(myInt);
8           System.out.println("myString = " + myString);
9
10          // 5316 -> 5314
11          myString = myString.replace('6', '4');
12
```

3. 형변환의 이해

랩퍼(wrapper) 클래스

```
13          // String형 객체 -> Float형 객체 -> float형
14          float myFloat = Float.valueOf(myString).floatValue();
15          System.out.println("myFloat = " + myFloat);
16      }
17  }
```

4. main() 메서드와 인수

- 자바가 C 언어 등과 가장 큰 차이점은 클래스를 기반으로 하는 객체지향 프로그래밍 언어라는 점이다.
- 물론 C++ 언어도 객체지향 언어라고는 하지만 C++ 언어는 C 언어의 연장선에 있기 때문에 비객체지향적인 요소를 많이 가지고 있다.
- 이에 비해 자바는 철저하게 객체지향 원칙을 지키고 있다.
- 객체지향 프로그래밍 언어는 객체 즉 클래스를 기반으로 하는 언어이다.
- 따라서 자바는 모든 명령어가 클래스 내부에 존재해야 한다.
- C언어의 함수에 해당하는 것이 자바에서는 메서드인데, 자바에서는 메서드가 절대로 독립적으로 존재할 수 없다.
- 반드시 클래스 내부에서 선언되고 사용되어야 한다.
- 또한 모든 클래스는 사용되기 전에 생성해야 한다.
- 우리가 앞에서 Random이나 Date 등의 클래스를 사용했을 때도, 사용하기 전에 new 명령으로 객체를 생성했었다.

4. main() 메서드와 인수

- 그런데, 여기에는 모순이 있다.
- 모든 메서드가 클래스 안에서 선언되고, 클래스가 사용되기 전에 생성되어야 한다면, 모든 명령어는 메서드 내에서 실행해야 하는데 대체 최초에 클래스를 생성하는 명령어는 어떻게 실행시킬 수 있는 걸까?
- 이것을 자바 프로그래머들은 일명 '닭-달걀 문제' 라고 부른다.
- 과연 어느 것이 먼저인지 알 수 없다는 뜻이다.
- 이러한 '닭이 먼저냐, 달걀이 먼저냐'하는 문제를 해결하기 위한 메서드가 main( ) 메서드이다.
- Main() 메서드는 자바 프로그램을 실행시키는 자바 가상머신(Java Virtual Machine, JVM) 이 호출하는 최초의 메서드이다.

4. main() 메서드와 인수

- 예를 들어 우리가

```
>>java Hello
```

- 라고 명령을 내리면, 자바 가상머신은 Hello라는 클래스 내의 main() 메서드를 찾아 실행시키고, 만일 main( ) 메서드가 없다면 실행시킬 수 없다는 에러 메시지를 출력하는 것이다.
- 자바에서는 몇 개의 클래스를 사용하든, 어떤 메서드를 사용하든 처음 시작은 main( ) 메서드이다.

4. main() 메서드와 인수

- 따라서 다른 메서드는 우리 마음대로 선언할 수 있지만, 이 main( ) 메서드는 자바 가상머신이 찾을 수 있도록 다음처럼 선언해야 한다.

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

- 여기서 public은 이 클래스 외부에서 호출할 수 있다는 뜻이고, static은 클래스를 생성하기 전에 쓸 수 있다는 뜻이다.
- Void는 main( ) 메서드가 호출된 후에 돌려주는 값이 없다는 뜻이고, String[] args는 String 클래스의 배열로 된 인수(아규먼트)를 받는다는 의미이다.
- Public, static 등의 객체지향 관련 명령어들과 void, 배열 등은 뒷 장에서 자세히 배울 것이다.

4. main() 메서드와 인수

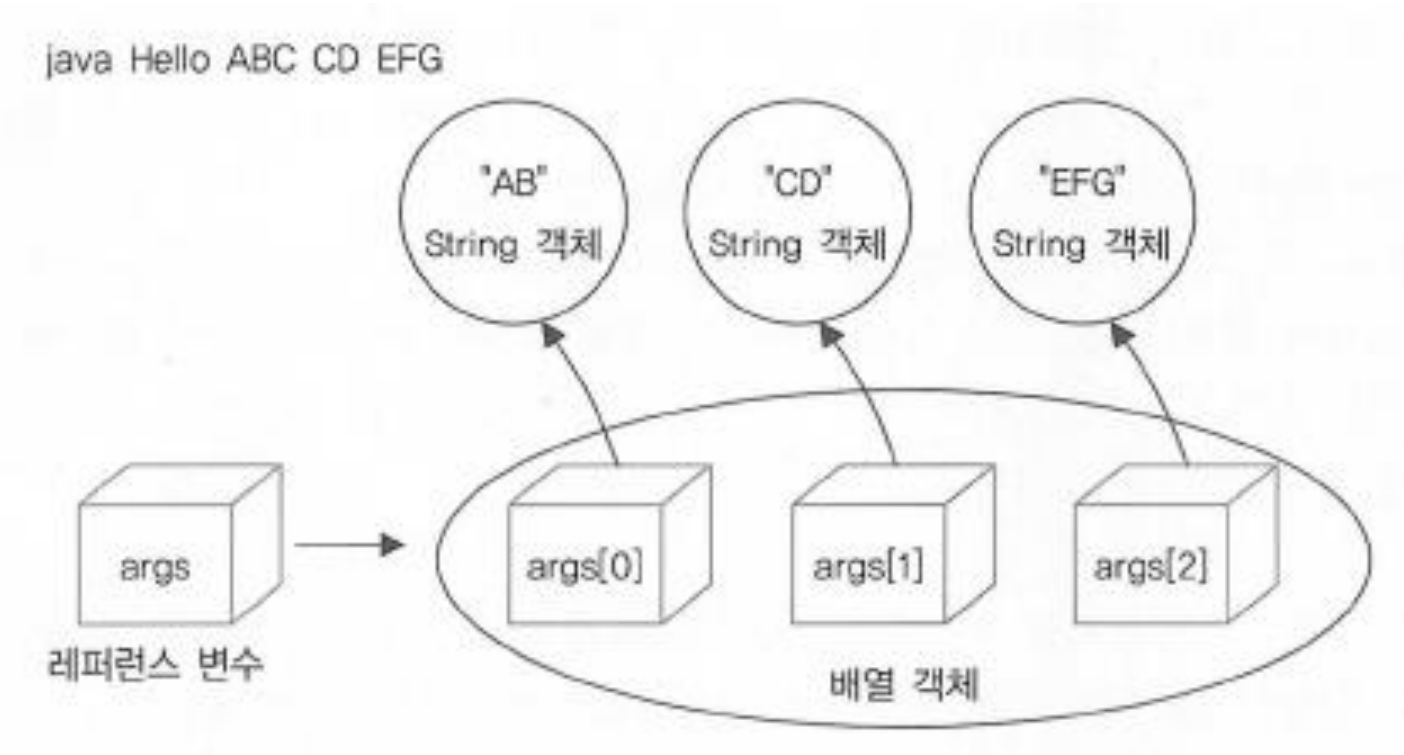
- 위 main( )의 선언에서 인수를 받는다는 의미로 String[] arg라고 선언했다.
- 뒷 장에서 배열을 학습할 때 자세히 배우겠지만, String[] args는 String args[ ] 와 같은 의미이다.
- 다른 메서드는 메서드를 호출할 때 전달하는 인수를 받습니다만 main( )의 경우는 처음 프로그램을 실행할 때 주는 인수가 전달된다.
- 예를 들어

```
>>java Hello AB CD EFG
```

- 라고 명령을 내리면, 자바 가상머신은 Hello라는 클래스의 main() 메서드를 실행시키면서, 1번째 인수로 AB, 2 번째 인수로 CD, 3번째 인수로 EFG를 전달한다.
- 이 인수는 모두 String형이다.

4. main() 메서드와 인수

>>java Hello AB CD EFG



4. main() 메서드와 인수

- 만일

```
>>java Hello 120 34
```

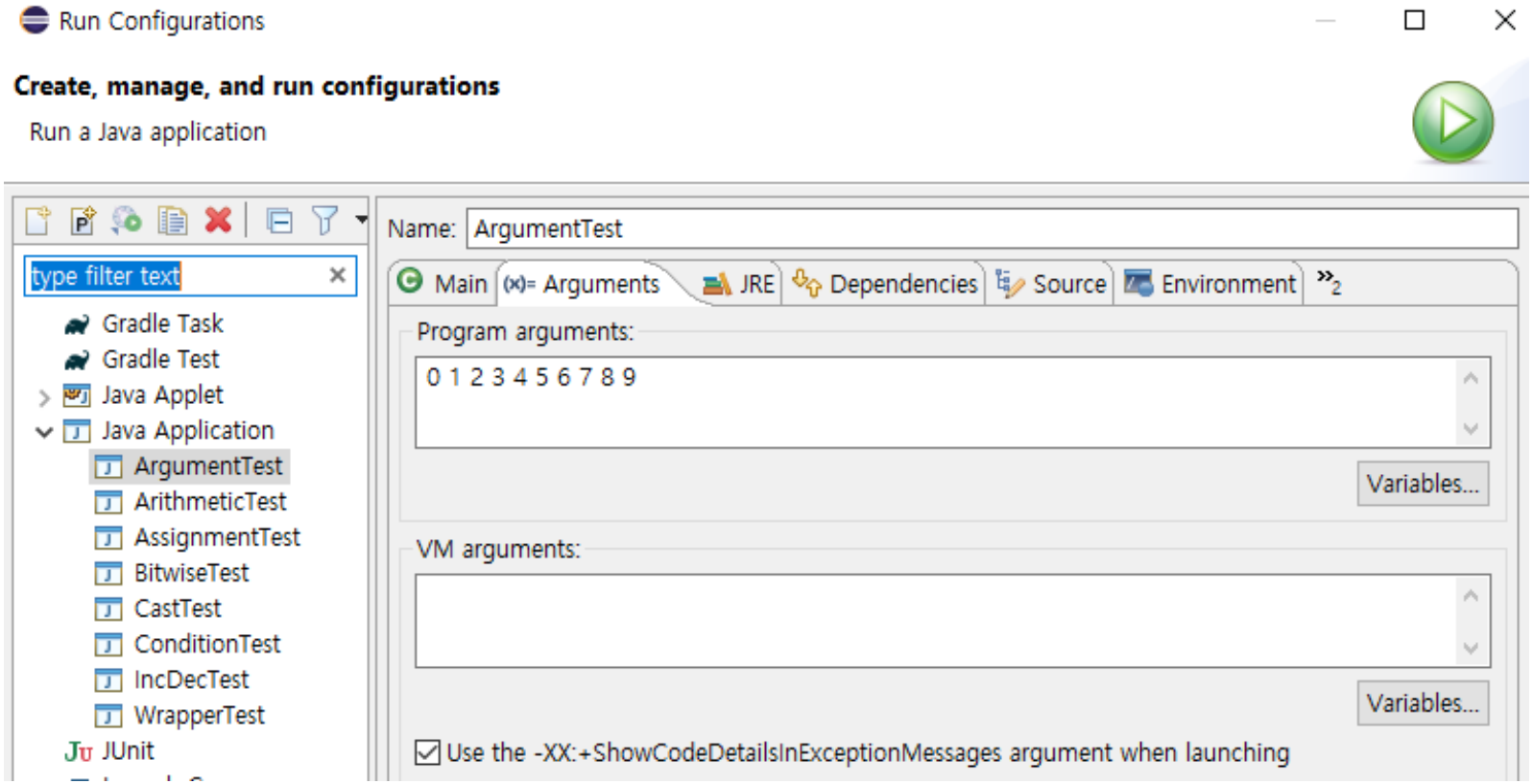
- 라고 명령을 내려도, 1번째 인수는 120, 2번째 인수는 34인 것은 마찬가지지만, 120은 수치형 데이터가 아니고 String형이다.
- 따라서 숫자로 사용하고 싶다면, 수치형으로 데이터 형을 변환시켜야 한다.
- 다음은 main( ) 메서드가 받은 인수를 순서대로 출력하는 예제이다.
- 주의할 점은 비록 1번째, 2번째, 3번째... 등으로 표현하지만 자바에서는 0부터 시작하기 때문에 1번째 인수는 args[1]이 아니고, args[0]이라는 점이다.
- 따라서 1번째, 2번째, 3번째 인수는 각각 args[0], args[1], args[2]에 대응한다.

4. main() 메서드와 인수


```
1  public class ArgumentTest {
2      public static void main(String[] args) {
3          for(int i = 0; i < args.length; i++) {          // 첨자는 0부터 시작
4              System.out.println(i + ": " + args[i]);
5          }
6      }
7  }
```

4. main() 메서드와 인수

- 이클립스에서 인수값 입력하는 방법
  - [Run]-[Run Configurations]를 선택하여 대화상자를 연다.
  - Arguments 탭을 선택한 후, Program arguments에 main() 메서드에 전달할 인수값을 입력한다.
  - [Run] 버튼을 눌러 실행하면 인수값이 전달된다.



5. 구구단 게임 만들기

- 구구단 게임은 인수로 값을 주면, 이 값과 난수로 만든 값을 곱해 문제를 만들고 플레이어(사람)가 입력한 값과 비교해서 정답여부를 알려주는 게임이다.
- 먼저 프로그램을 실행 시킬 때 주는 인수를 얻는데, 이 인수는 String 형이므로 랩퍼 클래스를 사용해서 int형 으로 바꿔 저장한다.
- 이때 만약 인수로 준 값이 없으면 난수로 1부터 9까지의 값 중 하나를 만든다.
- 이렇게 구한 x값과 난수로 만든 y값을 곱해서 num 값을 만들고, 플레이어(사람)가 입력한 값과 비교하는데, 플레이어가 입력한 값도 String형이기 때문에 역시 랩퍼 클래스를 이용해서 int형으로 바꾼 후 비교한다.
- Num 값과 플레이어가 입력한 값이 같으면 정답임을 출력하고 다르면 정답을 알려준다.

5. 구구단 게임 만들기


```
1   import java.util.*;
2   import java.io.*;
3
4   public class GameJave2_04 {
5       public static void main(String[] args) throws IOException{
6           int x, y;
7           Random r = new Random();
8           Scanner stdIn = new Scanner(System.in);
9           if(args.length == 1) {    // 인수가 있으면 인수로 준 값으로 문제 출제
10              // 인수는 String형이므로 랩퍼 클래스를 사용하여 int형으로 변환
11              x = Integer.valueOf(args[0]).intValue();
12          }else {
13              x = Math.abs(r.nextInt() % 9) + 1;
14          }
15          y = Math.abs(r.nextInt() % 9) + 1;
16
```

5. 구구단 게임 만들기


```
17          int num = x * y;
18          System.out.println();
19          System.out.println("* 구구단 " + x + "단에 대한 문제입니다.");
20          System.out.println();
21          System.out.print(x + " * " + y + " = ");
22
23          //BufferedReader in = new BufferedReader(new
        InputStreamReader(System.in));
24          String user;
25          user = stdIn.nextLine(); // 키보드로부터 입력
26
27          // 키보드에서 입력받은 값은 String형이므로 랩퍼클래스를 사용하여 int형으
        로 변환
28
29          int inputNum = Integer.valueOf(user).intValue();
30
```

5. 구구단 게임 만들기


```
31              if(num == inputNum) {
32                  System.out.println("맞았습니다!");
33              }else {
34                  System.out.println("틀렸습니다. 정답은 " + num + "입니다.");
35              }
36          }
37      }
```

Thank You