CHAPTER 03

컴퓨터에게 작업을 시키는 기호 연산자

연산자는 컴퓨터에 계산을 부탁하기 위해 사용하는 기호입니다.

연산자에 대한 내용은 비단 Java에만 해당하는 내용이 아닙니다. 프로그래밍의 기초적인 부분이라고 생각합니다. 다른 언어에 대한 경험이 있다면 쉽게 보실 수 있는 내용입니다. 흔히들 연산자라고 하면 더하기, 빼기, 곱하기, 나누기를 생각합니다. 물론 이것들이 연산자가 아니라는 말은 아닙니다. 다만, 연산자라는 것을 좀 더 근본적으로 이해하시면 좋겠습니다. 연산자라는 것은 "자 이제 여기서는 계산을 좀 했으면 좋겠어."라는 개발자들의 의사 표시입니다. Java 언어에서도 많은 종류의 연산자들이 있고 이를 처리하는 방식이 있습니다.

종류	설명	연산자
단항 연산자	변수 하나에 붙이는 연산자	++,--, +, -, !(타입)
산술 연산자	숫자 연산/시프트 연산	*, /, %,
		+, -
		〈〈, 〉〉, 〉〉〉
비교 연산자	부등호 연산	〈, 〉, 〈=, 〉=, instance of
	같다/다르다 연산	==, !=
논리 연산자	AND/OR	&, ^, |, &&, ||,
삼항 연산자	간단한 제어 처리	?, :
할당 연산자	연산의 결과를 담을 때 사용	=

표

1 산술연산: 더하기, 빼기, 곱하기, 나누기, 나머지

산술연산은 이미 초등학교 시절에 다 배운 연산자입니다. 다만, 나머지 연산자로 % 기호를 이용한다는 점은 봐두셔야 하고, 나눗셈 같은 경우 정수와 정수의 연산으로는 계산하지 않도록 주의하시면 됩니다.

산술연산에서 가장 많이 쓰이는 연산자
+, –, *, /
% (나머지)
ex) 10 % 4 → 나머지 2

연산자에는 우선순위가 있습니다. 예를 들어 곱셈(*), 나눗셈(/), 나머지(%) 연산자는 더하기(+), 빼기(–) 연산자보다 우선합니다. 하지만, 이런 소소한 규칙보다는 괄호를 이용해서 연산의 우선순위를 높여주는 방법만을 알고 있으면 됩니다.

예제 | 연산을 하고 결과를 담는 변수

```
public class OperEx {
	public static void main(String[] args) {
		int x = 10;
		int y = 2;
		int addResult = x + y;
		int minusResult = x - y;
		System.out.println(addResult);
		System.out.println(minusResult);
	}
}
```

```
12
8
```

2 자동증감 연산자: ++, --

사실 연산자에는 하나의 비밀이 숨겨져 있습니다. 연산자를 이용하고 나면 반드시 그 결과를 어떤 방식으로든 받아주어야 한다는 사실입니다. 다만, 이 법칙에 지배를 벗어나는 연산이 바로 자

동증감 연산입니다.

자동증감 연산자는 별도의 연산 결과를 처리하지 않는 연산자를 의미합니다. ++,— 연산자는 연산 대상에 1을 증가하거나 감소한 후에 연산의 결과를 원래의 변수에 자동으로 대입해줍니다. 다만, 연산자의 위치가 변수의 앞인지 뒤인지에 따라서 결과가 다를 수 있습니다.

예제 | 연산자는 반드시 그 결과를 받거나 출력해야만 합니다.

```
public class OpertTest {
	public static void main(String[] args) {
		int i = 10;
		int j = 100;
		i + j; //컴파일 에러 발생
	}
}
```

위의 코드를 보면 i와 j의 변수를 + 연산자를 이용해서 더해 주었지만, 그 결과를 처리하지 않고 있습니다. 따라서 위의 코드는 컴파일되지 않는 코드입니다. 연산자를 이용한다는 것은 그 결과를 어떤 방식으로든 받거나 처리한다는 것을 의미하기 때문에 1) 변수를 이용해서 결과를 담거나, 2) 어떤 형태로든 연산된 결과를 사용하거나 해야 합니다.

자동증감 연산자는 거의 유일하게 이 법칙을 벗어나는 연산자입니다. 즉 변수의 값이 변하지만, 특별히 새로운 변수나 별도의 할당 없이도 그냥 사용할 수 있습니다. 자동증감은 정수의 값이 1씩 증가하거나 감소하는 것을 의미합니다. 다만 ++이나--가 변수의 앞쪽에도 올 수 있고, 변수의 뒤쪽에도 올 수 있기 때문에 각각의 상황에 맞게 계산해야 합니다.

2.1 ++x, --x는 먼저 더하거나 빼고 계산합니다.

++,--가 앞에 붙는 경우는 간단히 말해서 변수를 사용하기 전(before)에 증감을 먼저 시도하라는 의미입니다. 먼저 해당 변수의 값을 증가시키거나 감소시키고 나머지 연산을 실행된다고 생각하면 됩니다.

예제

```
nt a = 0;
int b = 10;

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

```
1
9
```

위 코드를 보면 ++,--가 변수의 앞쪽에 있는 것을 볼 수 있습니다. 앞쪽에 있기 때문에 실제로 사용되기 이전에 증가나 감소가 먼저 일어나고 이후에 그 값을 사용한 것이 되는 겁니다. 반면에 ++,--가 뒤에 붙는 경우는 말 그대로 후(after)라는 뜻입니다.

예제 ++,-- 기호가 앞에 있으면 변수를 변경한 다음 계산한다.

```
public class AutoEx {
	public static void main(String[] args) {
		int a = 0;
		int b = 10;

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

```
1
9
```

코드를 보면 a 변수의 값은 0이지만 ++이 앞쪽에 붙어 있기 때문에 먼저 증가시킨 후에 변수를 사용하게 됩니다. 따라서 a의 값은 1이 됩니다. b의 경우에는 변수 선언 시점에 10으로 초기화되어 있지만, 이후에 변수의-- 연산을 우선 하게 되고, 그 결과를 System.out.println( )에서 활용하기 때문에 10에서 9로 변경하고 나서 사용됩니다.

2.2 x++, x--는 먼저 사용하고 나서 1씩 더하거나 뺀다.

증감 연산자가 뒤쪽에 오는 경우에는 변수가 사용되고 나서 해당 변수의 값이 1씩 변경되는 것을 의미합니다.

예제 ++,--가 뒤에 붙으면 변수의 값은 이전의 값을 그대로 사용한 후에 변경됩니다.

```
public class AutoEx2 {
	public static void main(String[] args) {
		int a = 0;
		int b = 10;

		System.out.println(a++);
		System.out.println(b++);

		System.out.println("변경 후");
		System.out.println(a);
		System.out.println(b);
	}
}
```

```
0
10
변경 후
1
11
```

위의 코드를 보시면 우선은 증감 기호(++,--)가 뒤쪽에 붙어 있기 때문에 우선 변수가 사용된 후에 값이 변경되는 코드를 의미합니다. 따라서 출력 결과의 위쪽 결과는 동일하게 기존의 0과 10이라는 값이 그대로 유지되는 것을 볼 수 있고, 변경 후에는 아무런 작업이 없어도 변수의 내용이 변경된 것을 확인할 수 있습니다. 즉 증감 기호가 뒤쪽에 붙는 것은 변수가 사용된 후의 코드에서 변경되는 것을 의미합니다.

■ 조금 더 어렵게 자동증감 연산자가 섞여 있는 경우

증감 연산자가 앞에 있는가, 뒤에 있는가의 의미만 정확히 알고 있으면 연산은 어렵지 않습니다. 그럼 조금 더 생각해봐야 하는 연산을 만들어 보도록 합니다.

예제 | 조금 더 복잡한 자동증감 연산

```
public class AutoEx3 {
        public static void main(String[] args) {
                int x = 5;
                int y = 10;
                int z = x+++ ++x + y++;

                System.out.println("z의 값은: " + z);
        }
}
```

위 코드의 결과는 무엇이 될까요?

- 우선 x++을 하게 되면 x의 값은 5가 된다. 그리고 나서 x의 값은 6으로 변경된 상태가 된다.
- x 값이 6으로 변경된 후에 ++x를 하게 되면 x의 값을 7로 변경한 후에 사용한다.
- y++의 값은 10으로 사용된 후에 증가하기 때문에 10이 된다. 따라서 최종 결과는 5+7+10이 된다.

따라서 실행 결과는 다음과 같습니다.

```
z의 값은: 22
```

3 동등 비교(Equality) 및 관계(Relational) 연산자

비교 및 관계 연산자는 부등호나, 같다, 틀리다 연산을 하는 데 사용합니다. 유심히 봐 두셔야 하는 것은 '같다(==)', '다르다(!=)'의 연산자입니다.

프로그램을 만들면서 가장 많이 하는 작업이 바로 '비교'입니다.
- a == b → a와 b 변수의 내용물이 같으면 true를 반환합니다.
- a != b → a와 b 변수의 내용물이 같지 않으면(NOT) true를 반환합니다.
- 기타 〉, 〈, 〉=, 〈= 연산자는 수학의 연산과 마찬가지로 부등호 연산입니다.

4 비트 연산자: |, &, ^

개발자가 눈에 보이는 연산 기호보다 컴퓨터가 쉽게 이해하는 연산 기호를 사용하면 분명히 속도 면에서 향상이 있습니다. 비트 연산자라는 것이 바로 그런 연산자입니다. 개발자가 직접 비트(bit)를 조작하면 연산이 빨라진다는 장점이 있습니다. 하자만 CPU의 성능이 비약적으로 향상된 요즘에는 많이 사용되지는 않습니다.

기호	설명
| (OR)	양쪽 데이터의 비트의 값을 OR 조건으로 따져서 한쪽 bit가 1이면 무조건 1이라는 결과를 낸다.
& (AND)	양쪽의 비트가 모두 1이면 1, 아닐 때는 0
^ (XOR)	양쪽의 비트가 서로 다른 비트를 가져야만 1, 아닐 경우는 0

표

간단하게 OR 연산자를 예를 들어서 설명해 보도록 합니다.

예제

```
public class OperEx {
	public static void main(String[] args) {
		int x = 3;
		int y = 2;

		System.out.println(x|y);
	}
}
```

```
3
```

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
10진수 3	0	0	0	0	0	0	1	1
10진수 2	0	0	0	0	0	0	1	0
3 | 2	0	0	0	0	0	0	1	1

결과 3

' | (OR)'연산은 어느 한 쪽만 1이면 1로 처리

그림 1 비트 연산의 OR

AND 연산은 양쪽의 비트 값이 1인 경우에만 1로 나오는 연산이므로 위의 경우 마지막 자리는 0이 됩니다. 따라서 결과는 2에 해당하는 값만 나오게 됩니다.

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
10진수 3	0	0	0	0	0	0	1	1
10진수 2	0	0	0	0	0	0	1	0
3 & 2	0	0	0	0	0	0	1	0

결과 2

'&(AND)'연산은 양쪽 모두 1이면 1로 처리

그림 2 비트 연산의 AND

XOR 연산은 양쪽의 비트의 값이 서로 다른 경우에만 1을 반환합니다. 7 ^ 3을 가지고 한번 해보겠습니다.

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
10진수 7	0	0	0	0	0	1	1	1
10진수 3	0	0	0	0	0	0	1	1
7^3	0	0	0	0	0	1	0	0

결과 4

'^(XOR)'연산은 양쪽 값이 다른 경우만 1로 처리

그림 3 비트 연산의 XOR

위의 설명을 간단한 예제로 만들어 보겠습니다.

예제

```
public class BitOper {
	public static void main(String[] args) {
		//OR 연산
		int x = 3 | 2;
		System.out.println("OR 연산 : " + x);
		//AND 연산
		int y = 3 & 2;
		System.out.println("AND 연산 : " + y);
```

```
                //^XOR 연산
                int z = 7 ^ 3;
                System.out.println("XOR 연산 " +z);
        }
}
```

```
OR 연산 : 3
AND 연산 : 2
XOR 연산 : 4
```

5 논리 연산자: &&, ||

논리 연산자는 여러 개의 조건을 종합적으로 검토해야 하는 경우에 많이 사용합니다.

논리 연산자는 '제어문'에서 여러 개의 조건을 검사하기 위해서 많이 사용합니다.
- A && B → A 조건도 '참'이고 B 조건도 '참'인 경우만 '참'으로 판단합니다.
- A || B → A, B 조건 둘 중에서 하나라도 '참'이라면 '참'으로 판단합니다.

논리 연산자의 경우에는 여러 가지 상황들이 종합적으로 반영되는 경우에 많이 사용합니다. 예를 들어 "오늘 비가 오고, 온도가 낮으면 차를 타고 가겠다."라든가, "지하철이나, 버스를 타고 오면 물건을 살 수 없다."라든가 하는 논리적인 상황들을 결합해서 사용합니다. 프로그래밍 관점에서 말하자면 로그인 같은 로직을 생각하면 됩니다. "아이디가 일치하고, 패스워드가 일치하면 회원이라고 판단한다."와 같은 경우가 가장 간단한 예입니다. 논리 연산자로는 &&, || 연산자를 기억하시면 됩니다.

기호	설명
&& (AND)	'조건 1 && 조건 2'인 경우에 조건 1 역시 true이고 조건 2 역시 true일 경우에 true로 판단한다. 만일 조건 1이 false일 경우에는 뒤의 조건을 검사하지 않는다.
|| (OR)	'조건 1 || 조건 2' 두 가지 중에서 어느 한 조건이 true인 경우는 true로 판단한다. 만일 앞의 조건이 true인 경우는 뒤의 조건을 검사하지 않는다.

표

6 기타 연산자: 삼항 연산자, 할당 연산자

삼항 연산자는 3가지로 구성되는 연산자입니다. 삼항 연산자는 '?'를 이용해서 표현하는데 요약하자면 다음 장에서 배우는 if ~ else 구문을 간단히 사용한 방식이라고 생각하면 됩니다(실제로 코드를 작성할 때에는 if ~ else를 더 많이 사용합니다).

삼항 연산자
int a = 10;
char c = a == 10? 'O':'X';
a == 10의 연산 결과가 true일 때는 'O'
a == 10의 연산 결과가 false일 때는 'X'
삼항 연산자는 결과 데이터를 변수로 받거나 처리해야 합니다.

예제 | 삼항 연산자는 일종의 제어문 역할을 합니다.

```
public class TriOper {
	public static void main(String[] args) {
		int a = 10;

		char c =
			a == 10? 'O': 'X';
		System.out.println(c);
	}
}
```

```
O
```

6.1 =은 같다가 아닌 '할당(Assign) 연산자'입니다.

=은 앞서 변수에서 설명했듯이 '할당'이라는 확실한 의미가 있는 연산자입니다. Java에서의 변수 처리는 데이터를 복사한다는 개념만이 있기 때문에 처음에 공부할 때부터 정확한 용도를 알아두어야 합니다.

int a = 10; → a 변수 상자에 10이라는 데이터를 할당합니다.
a = a + 10; → a 변수 상자(왼편)에 a 변수 상자의 내용물을 꺼내서(오른편) 10을 더한 결과를 담습니다.
int b = a; → b라는 변수 상자에는 a 상자의 내용물을 복사(Copy)해서 담아줍니다.

■ **연산자의 응용: +=, -=**

할당 연산자의 경우 그냥 사용되는 경우도 많지만, 코드 내에 변수가 여러 번 나오면 그만큼 메모리상에 상자가 존재했다가 사라져야 하기 때문에 좀 더 간결하게 사용하는 경우도 있습니다.

예제

```
int a = 10;
a += 100; // 'a = a+ 100'을 줄여서 표현할 때 사용합니다.
```

'a = a + 100;'이라는 코드를 간결하게 표현하기 위해서 'a += 100;'과 같이 줄여서 표현하기도 합니다.

6.2 홀수 짝수 프로그램 만들기

변수에 대한 개념을 학습했으니 이번에는 임의의 숫자를 하나 발생시켜서 그 숫자가 홀수인지 짝수인지를 계산하는 프로그램을 만들어볼까 합니다. 마찬가지로 실행 결과를 먼저 보고 생각해보도록 합니다.

```
임의의 값은: 64
나머지 값은: 0
```

물론 위의 임의의 값은 프로그램을 실행할 때마다 매번 변경되는 값입니다. 홀수 짝수는 2로 나눈 나머지의 값이 0인지 1인지에 대한 문제입니다. 다음 장에서 제어문을 배우면 좀 더 우아한 메시지를 보여줄 수도 있습니다만, 아직은 0, 1 정도로만 표현하겠습니다.

■ **배운 내용 중에서**

그럼 위의 프로그램에서 우리가 배운 내용 중에 사용할 수 있는 부분을 생각해 보겠습니다.

- 숫자로 변수를 하나 선언할 수 있다.
- '%'를 이용하면 나머지 연산을 할 수 있다.

■ **배우지 않은 내용 찾아내기**

• 임의의 숫자를 어떻게 발생해야 하는지를 모른다.

임의의 숫자를 발생하는 것은 주로 Math.random( )이나 Random.nextInt( )라는 것을 이용합니다. Random 클래스를 이용하는 예제를 간단히 만들어 보도록 하겠습니다.

예제

```
import java.util.Random;
public class RandomEx {
        public static void main(String[] args) {
                Random r = new Random();
                int randomValue = r.nextInt(100);
                System.out.println(randomValue);
        }
}
```

1_ import라는 것을 이용해서 임의의 숫자 발생기능을 가져다 사용할 수 있게 합니다.

2_ 임의의 숫자 발생기를 하나 선언합니다. 아직은 자세한 내용은 모르셔도 됩니다. 다만, 'Random r = new Random( )'이라는 것이 임의의 숫자 발생기이고, 이를 위해서 라인 1에 작업한다는 사실만 알고 있으면 됩니다.

3_ r.nextInt(100)이라는 부분이 바로 핵심적인 부분입니다. nextInt(100)은 0에서부터 100 미만의 임의의 정수를 발생시켜주는 장치입니다. 발생한 정수를 randomValue라는 것으로 받아들였습니다.

■ **만들어보기**

예제

```
import java.util.Random;

public class OddAndEven {
        public static void main(String[] args) {
                Random r = new Random();
                int value = r.nextInt(100);
                int odd = value%2;
```

```
                System.out.println("임의의 값은: " + value);
                System.out.println("나머지 값은: " + odd);
        }
}
```

```
임의의 값은: 29
나머지 값은: 1
```

'Random r = new Ramdom( );' 코드는 임의의 숫자를 가져오기 위한 특별한 장치를 준비합니다(이 장치에 대해서는 객체지향 프로그래밍에서 자세히 배우게 됩니다).

'int value = r.nextint( );'에서는 0에서 100 미만의 숫자를 만들어주는 소스 코드를 실행합니다. 'int odd = value%2;' 코드에서는 만들어진 숫자를 2로 나눈 나머지 값을 구합니다. 2로 value 값을 나누는 경우의 나머지 값은 0 혹은 1밖에 될 수 없습니다. 현재까지의 학습한 내용이라면 아직은 나머지 값을 연산하고 결과를 화면에 보여주는 정도입니다. 하지만, 나중에 제어문을 학습하고 나면 홀수나 짝수라는 메시지를 화면에 보여줄 수 있습니다.

7 16진수와 8진수

우리가 주로 사용하는 것은 10진수이지만 컴퓨터가 좀 더 편하게 사용하는 수인 16진수와 8진수에 대해서도 약간의 지식이 있으면 좋습니다. 16진수와 8진수는 말 그대로 숫자의 단위가 16, 8이라는 의미입니다. 8진수의 경우에는 0에서 7까지의 값을 사용하고, 16진수의 경우에는 0~9, A, B, C, D, E, F까지 알파벳 6개를 사용하므로 표기법을 알아 두시면 됩니다.

- 16진수는 숫자 앞에 '0x'를 붙여준다.
- 8진수는 숫자 앞에 '0'을 붙인다.

예제

```
public class EtcEx {
	public static void main(String[] args) {
		int a = 0x11;//16진수
		System.out.println(a); //← 17

		int b = 011; //8진수
		System.out.println(b); //← 9
	}
}
```

```
17
9
```

변수 a는 16진수이기 때문에 10이 넘으면 16이라는 값이 추가됩니다. 따라서 a는 16 + 1 = 17의 값을 가집니다. 변수 b는 앞에 '0'을 붙인 8진수입니다. 8진수를 이용할 때 주의해야 하는 사항은 8진수는 0에서 7까지의 숫자만을 활용하기 때문에 '09'와 같은 선언은 불가능하다는 점입니다. 라인 7에서 11이라는 값은 8을 기준으로 하기 때문에 8+1의 값인 9가 됩니다.

8 프로그램에서 쓰이는 특수문자

프로그래밍을 하다가 보면 자주 마주치는 특수문자들이 몇 가지 있습니다. 이러한 특수문자와 표기법 정도를 알아 두시면 유용할 듯합니다.

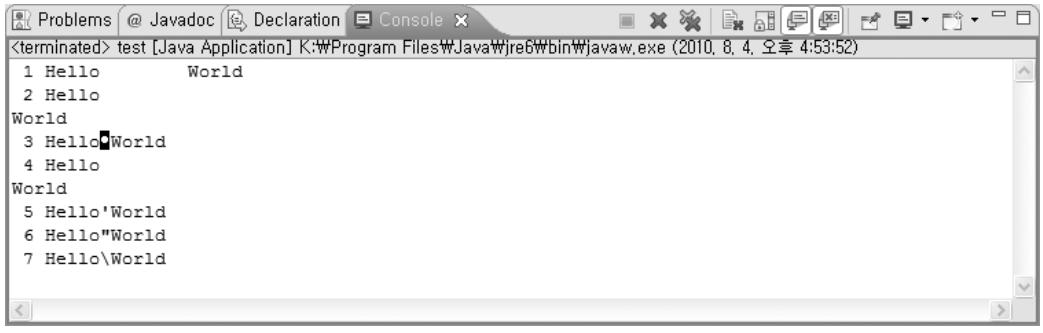
표기	의미
char c ='\t' (역슬래시+t)	탭 간격
\n	줄 바꿈
\b	백스페이스(한 칸 뒤로 이동)
\r	Carriage Return (프린터나 타자기에서 사용되던 개념인데, 컴퓨터의 텍스트 파일에서도 사용되고 있습니다. '커서를 아래로 내리는 동작'과 '커서의 행의 맨 앞으로 보내는 동작'을 합치면, 다음 줄로 행 바꿈이 됩니다.)
\'	작은따옴표
\"	큰따옴표
\u	유니코드 문자

표

특수문자는 주로 문자열을 처리할 때 많이 사용하게 됩니다. 간단한 예제를 통해 알아보도록 합니다. 특히 주의해서 보실 것은 '\n', '\b', '\r'입니다. 이 중 화면에 줄을 바꿀 때에는 주로 '\n'을 사용합니다.

예제

```
public class StringLiteralTest {
	public static void main(String[] args) {
		System.out.println(" 1 Hello\tWorld" );
		System.out.println(" 2 Hello\nWorld" );
		System.out.println(" 3 Hello\bWorld" );
		System.out.println(" 4 Hello\rWorld" );
		System.out.println(" 5 Hello\'World" );
		System.out.println(" 6 Hello\"World" );
		System.out.println(" 7 Hello\\World" );
	}
}
```



프로그래밍을 구성하는 요소인 데이터는 주로 변수로 표현됩니다. 그리고 프로그램의 로직은 지금 배운 연산자라는 것과 제어문이 결합되어 만들어집니다. 다음 장에서는 제어문에 대해서 공부하도록 하겠습니다.