Chapter 03. 연산자

3.1 부호/증감 연산자

부호 연산자

연산식	설명
+ 피연산자	피연산자의 부호 유지
- 피연산자	피연산자의 부호 변경

- 정수 타입(byte, short, int) 연산의 결과는 int 타입
- 부호를 변경하는 것도 연산이므로 byte / short 타입 부호 변경 후 int 타입 변수에 대입해야 함

증감 연산자

연산식	설명
++피연산자	피연산자의 값을 1 증가시킴
--피연산자	피연산자의 값을 1 감소시킴
피연산자++	다른 연산을 수행한 후에 피연산자의 값을 1 증가시킴
피연산자--	다른 연산을 수행한 후에 피연산자의 값을 1 감소시킴

```
int x = 1;
int y = 1;
int result1 = ++x + 10;     // x를 1 증가 -> result1 = 2 + 12
int result2 = y++ + 10;     // result2 = 1 + 10 -> y를 1 증가
```

3.2 산술 연산자

연산식	설명
피연산자 + 피연산자	덧셈 연산
피연산자 - 피연산자	뺄셈 연산
피연산자 * 피연산자	곱셈 연산
피연산자 / 피연산자	나눗셈 연산
피연산자 % 피연산자	나눗셈의 나머지를 산출하는 연산

- 피연산자가 정수 타입(byte, short, char, int)이면 연산의 결과는 int 타입이다.
- 피연산자가 정수 타입이고 그 중 하나가 long 타입이면 연산의 결과는 long 타입이다.
- 피연산자 중 하나가 실수 타입이면 연산의 결과는 실수 타입이다.

3.3 오버플로우와 언더플로우

- **오버플로우(overflow)** : 타입이 허용하는 최대값을 벗어나는 것
- **언더플로우(underflow)** : 타입이 허용하는 최소값을 벗어나는 것

정수 타입 연산에서 오버플로우 또는 언더플로우가 발생하면 실행 에러가 발생할 것 같지만, 그렇지는 않고 해당 정수 타입의 최소값 또는 최대값으로 되돌아간다.

```
byte value = 127;
value++;
System.out.println(value);  // -128

byte value2 = -128;
value2--;
System.out.println(value2);  // 127
```

항상 해당 타입의 범위 내에서 연산이 수행되도록 코딩에 신경써야 한다. 만약 연산 과정에서 int 타입에서 오버플로우 또는 언더플로우가 발생될 가능성이 있다면 *long 타입*으로 연산을 하도록 해야 한다.

3.4 정확한 계산은 정수 연산으로

산술 연산을 정확하게 계산하고 싶다면 실수 타입을 사용하지 않는 것이 좋다. 부동 소수점 방식을 사용하는 실수 타입에서는 1에서 0.7을 빼더라도 사용자가 원하는 값인 0.3이 출력되는 것이 아닌 0.2999...9993와 같이 출력되기 때문이다. 그렇기에 정확한 계산이 필요하다면 정수 연산으로 변경해서 계산하는 것이 좋다.

3.5 나눗셈 연산 후 NaN과 Infinity 처리

나눗셈 또는 나머지 연산에서 *좌측 피연산자가 정수*이고 *우측 피연산자가 0*일 경우 **예외(ArithmeticException)** 가 발생한다. (무한대의 값을 정수로 표현할 수 없기 때문)

하지만 *좌측 피연산자가 실수*이거나 *우측 피연산자가 0.0 또는 0.0f*이면 예외가 발생하지 않고 연산의 결과는 **Infinity**(무한대) 또는 **NaN**(Not a Number)이 된다. 여기서는 어떤 연산을 하더라도 결과는 계속 Infinity와 NaN이 된다.

```
boolean result = Double.isInfinite(변수);
boolean result = Double.isNaN(변수);
```

3.6 비교 연산자

구분	연산식	설명
동등 비교	피연산자1 == 피연산자2	두 피연산자의 값이 같은지를 검사
동등 비교	피연산자1 == 피연산자2	두 피연산자의 값이 같은지를 검사
크기 비교	피연산자1 > 피연산자2	피연산자1이 큰지를 검사
크기 비교	피연산자1 >= 피연산자2	피연산자1이 크거나 같은지를 검사

구분	연산식	설명
크기 비교	피연산자1 < 피연산자2	피연산자1이 작은지를 검사
크기 비교	피연산자1 <= 피연산자2	피연산자1이 작거나 같은지를 검사

피연산자의 타입이 다를 경우에는 **비교 연산을 수행하기 전에 타입을 일치**시킨다.

```
'A' == 65 // true
3 == 3.0 // true
```

부동 소수점 방식을 사용하는 실수 타입은 0.1을 정확히 표현할 수 없고 float 타입과 double 타입의 정밀도 차이도 나기 때문에 다음 코드는 false가 산출된다. 해결책은 **피연산자를 float 타입으로 강제 변환 후** 비교 연산을 하면 된다.

```
0.1f == 0.1 // false
```

문자열을 비교할 때에는 동등(==, !=) 연산자 대신 **equals()** 와 **!equals()** 를 사용한다.

3.7 논리 연산자

구분	연산식	설명
AND(논리곱)	**&&** 또는 **&**	피연산자 모두가 true일 경우에만 연산 결과가 true
OR(논리합)	**||** 또는 **|**	피연산 중 하나만 true이면 연산 결과는 true
XOR(배타적 논리합)	**^**	피연산자가 하나는 true이고 다른 하나가 false일 경우에만 연산 결과가 true
NOT(논리 부정)	**!**	피연산자의 논리값을 바꿈

- && 또는 || : 앞의 피연산자가 조건에 해당할 경우 뒤의 피연산자를 평가하지 않고 결과 산출. 더 효율적임.
- & 또는 | : 두 피연산자 모두 평가

3.8 비트 논리 연산자

- bit 단위로 논리 연산을 수행.
- 0과 1이 피연산자가 되므로 2진수 0과 1로 저장되는 정수 타입(byte, short, int, long)만 피연산자가 될 수 있음.

구분	연산식	설명
AND(논리곱)	**&**	두 비트 모두 1일 경우에만 연산 결과가 1
OR(논리합)	**|**	두 비트 중 하나만 1이면 연산 결과는 1
XOR(배타적 논리합)	**^**	두 비트 중 하나는 1이고 다른 하나가 0일 경우 연산 결과는 1

구분	연산식	설명
NOT(논리 부정)	~	보수

비트 논리 연산자는 byte, short, char 타입 피연산자를 **int 타입으로 자동 변환**한 후 연산을 수행한다. 따라서 연산 결과도 int 타입이 되므로 int 변수에 대입해야 한다.

왜 필요할까?
소형 임베디드 장치의 C 프로그램에서 외부 서버의 자바 프로그램으로 데이터를 전달한다고 가정하자. C 언어에는 uint8_t 타입이 있는데 이는 1byte 크기를 가지면서 0~255 값의 범위를 가진다. C 프로그램이 uint8_t 타입 136을 2진수로 보내면, 자바는 2진수를 -120로 읽게 된다. 자바는 최상위 비트가 1이면 음수로 인식하기 때문이다. 이때 -120을 C 프로그램이 보낸 136으로 복원하고 싶다면 **-120과 255를 비트 논리곱(&) 연산을 수행**하면 된다.

```
byte receiveData = -120;
int unsignedInt = receiveData & 255;    // 136
```

또는 자바가 개발자의 편리성을 위해 **Byte.toUnsingedInt()** 코드를 제공하므로 이를 사용해도 된다.

```
byte receiveData = -120;
int unsignedInt = Byte.toUnsingedInt(receiveData);    // 136
```

3.9 비트 이동 연산자

구분	연산식	설명
이동(shift)	a << b	정수 a의 각 비트를 b만큼 왼쪽으로 이동 오른쪽 빈자리는 0으로 채움 a x 2^b와 동일한 결과가 됨
이동(shift)	a >> b	정수 a의 각 비트를 b만큼 오른쪽으로 이동 왼쪽 빈자리는 최상위 부호 비트와 같은 값으로 채움 a / 2^b와 동일한 결과가 됨
이동(shift)	a >>> b	정수 a의 각 비트를 b만큼 오른쪽으로 이동 왼쪽 빈자리는 0으로 채움

3.10 대입 연산자

- 우측 피연산자의 값을 좌측 피연산자인 변수에 대입
- 우측 피연산자에는 리터럴 및 변수, 그리고 다른 연산식이 올 수 있음
- 단순히 값을 대입하는 *단순 대입 연산자*
- 정해진 연산을 수행한 후 결과를 대입하는 *복합 대입 연산자*

```java
public class AssignmentOperatorExample {
    public static void main(String[] args) {
        int result = 0;
        result += 10;
        System.out.println("result=" + result);     // result=10
        result -= 5;
        System.out.println("result=" + result);     // result=5
        result *= 3;
        System.out.println("result=" + result);     // result=15
        result /= 5;
        System.out.println("result=" + result);     // result=3
        result %= 3;
        System.out.println("result=" + result);     // result=0
    }
}
```

3.11 삼항(조건) 연산자

- **조건식 ? 값 또는 연산식 : 값 또는 연산식**
  ? 앞의 피연산자는 boolean 변수 또는 조건식이 오므로 조건 연산자라고도 한다. 이 값이 true이면 콜론(😃 앞의 피연산자가 선택되고, false이면 콜론 뒤의 피연산자가 선택된다.

```java
public class ConditionalOperationExample {
    public static void main(String[] args) {
        int score = 85;
        char grade = (score > 90) ? 'A' : ((score > 80) ? 'B' : 'C');
        System.out.println(score + "점은 " + grade + "등급입니다.");
        // 85점은 B등급입니다.
    }
}
```

3.12 연산의 방향과 우선순위

연산자	연산 방향	우선순위
증감(++, --), 부호(+, -), 비트(~), 논리(!)	<-	높음
산술(*, /, %)	->	
산술(+, -)	->	
쉬프트(<<, >>, >>>)	->	
비교(<, >, <=, >=, instanceof)	->	
비교(==, !=)	->	
논리(&)	->	

연산자	연산 방향	우선순위
논리(^)	->	
논리(|)	->	
논리(&&)	->	
논리(||)	->	
조건(?😃	->	
대입(=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=)	<-	낮음

+ 먼저 처리해야 할 연산을 괄호()로 묶는 것 추천

Chapter 04. 조건문과 반복문

4.1 코드 실행 흐름 제어

- (흐름)**제어문** : main() 메소드의 시작 중괄호에서 끝 중괄호까지 위에서부터 아래로 실행하는 흐름을 개발자가 원하는 방향으로 바꿀 수 있도록 해주는 것.

조건문	반복문
if 문, switch 문	for 문, while 문, do-while 문

4.2 if 문

```
public class IfNestedExample {
    public static void main(String[] args) {
        int score = (int)(Math.random()*20) + 81;
        System.out.println("점수: " + score);

        String grade;

        if(score>=90) {
            if(score>=95) {
                grade = "A+";
            } else {
                grade = "A";
            }
        } else {
            if(score>=85) {
                grade = "B+";
            } else {
                grade = "B";
            }
        }

        System.out.println("학점: " + grade);
```

```
    }
}
```

4.3 switch 문

```
public class SwitchExample {
    public static void main(String[] args) {
        int num = (int)(Math.random()*6) + 1;

        switch(num) {
            case 1:
                System.out.println("1번이 나왔습니다.");
                break;
            case 2:
                System.out.println("2번이 나왔습니다.");
                break;
            case 3:
                System.out.println("3번이 나왔습니다.");
                break;
            case 4:
                System.out.println("4번이 나왔습니다.");
                break;
            case 5:
                System.out.println("5번이 나왔습니다.");
                break;
            default:
                System.out.println("6번이 나왔습니다.");
        }
    }
}
```

- `default`는 생략이 가능하며, `break`를 사용하지 않는다면 그 다음 `case`를 실행하게 된다.
- `switch` 문의 괄호에는 정수 타입 (byte, char, short, int, long)과 문자열 타입(String) 변수만이 사용 가능하다.

Switch Expressions

```
public class SwitchValueExample {
    public static void main(String[] args) {
        String grade = "B";

        //Java 11 이전 문법
        int score1 = 0;
        switch(grade) {
            case "A":
                score1 = 100;
                break;
```

```
            case "B":
                int result = 100 - 20;
                score1 = result;
                break;
            default:
                score1 = 60;
        }
        System.out.println("score1: " + score1);

        //Java 12부터 가능
        int score2 = switch(grade) {
            case "A" -> 100;
            case "B" -> {
                int result = 100 - 20;
                //Java 13부터 가능
                yield result;
            }
            default -> 60;
        };
        System.out.println("score2: " + score2);
    }
}
```

- 스위치된 값 변수에 바로 대입 가능.
- 단일 값일 경우 화살표 오른쪽에 값을 기술하고, 중괄호를 사용할 경우에는 `yield` 키워드로 값을 지정. (단, `default` 반드시 존재해야 함)

4.4 for 문

```
public class SumFrom1To100Example {
    public static void main(String[] args) {
        int sum = 0;
        int i;

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

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

- 초기화식, 조건식, 증감식 둘 이상 가능
- 초기화식에서 부동 소수점을 쓰는 *`float` 타입은 사용 금지.*

중첩된 for 문

```
public class MultiplicationTableExample {
    public static void main(String[] args) {
        for (int m=2; m<=9; m++) {
            System.out.println("*** " + m + "단 ***");
            for (int n=1; n<=9; n++) {
                System.out.println(m + " x " + n + " = " + (m*n));
            }
        }
    }
}
```

4.5 while 문

```
import java.util.Scanner;

public class KeyControlExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        boolean run = true;
        int speed = 0;

        while(run) {
            System.out.println("-----------------------------");
            System.out.println("1. 증속 | 2. 감속 | 3. 중지");
            System.out.println("-----------------------------");
            System.out.print("선택: ");

            String strNum = scanner.nextLine();

            if(strNum.equals("1")) {
                speed++;
                System.out.println("현재 속도 = " + speed);
            } else if(strNum.equals("2")) {
                speed--;
                System.out.println("현재 속도 = " + speed);
            } else if(strNum.equals("3")) {
                run = false;
            }
        }

        System.out.println("프로그램 종료");
    }
}
```

4.6 do-while 문

```
import java.util.Scanner;

public class DoWhileExample {
    public static void main(String[] args) {
        System.out.println("메시지를 입력하세요.");
        System.out.println("프로그램을 종료하려면 q를 입력하세요.");

        Scanner scanner = new Scanner(System.in);
        String inputString;

        do {
            System.out.print(">");
            inputString = scanner.nextLine();
            System.out.println(inputString);
        } while( ! inputString.equals("q") );

        System.out.println();
        System.out.println("프로그램 종료");
    }
}
```

4.7 break 문

```
public class BreakExample {
    public static void main(String[] args) throws Exception {
        while(true) {
            int num = (int)(Math.random()*6) + 1;
            System.out.println(num);
            if(num == 6) {
                break;
            }
        }
        System.out.println("프로그램 종료");
    }
}
```

- 중첩된 반복문에서 바깥쪽 반복문까지 종료시키려면 바깥쪽 반복문에 **이름(레이블)**을 붙이고, `break 이름;`을 사용.

```
public class BreakOutterExample {
    public static void main(String[] args) throws Exception {
        Outter: for(char upper='A'; upper<='Z'; upper++) {
            for(char lower='a'; lower<='z'; lower++) {
                System.out.println(upper + "-" + lower);
                if(lower=='g') {
                    break Outter;
                }
```

```
            }
        }
        System.out.println("프로그램 실행 종료");
    }
}
```

4.8 continue 문

```
public class ContinueExample {
    public static void main(String[] args) throws Exception {
        for(int i=1; i<=10; i++) {
            if(i%2 != 0) {
                continue;
            }
            System.out.print(i + " ");
        }
    }
}
```