02_2 Operators and Control Structures

Object-Oriented Programming

이 강의에서는 Java 언어의 operator들과 control structure들에 대해 강의하겠습니다.

Arithmetic Operators

perform mathematical operations

- **Addition (+)**: Adds two operands.
- **Subtraction (-)**: Subtracts the second operand from the first.
- **Multiplication (*)**: Multiplies two operands.
- **Division (/)**: Divides the numerator by the denominator.
- **Modulus (%)**: Returns the remainder of a division.
- **Increment (++)**: Increases the value of an operand by 1.
- **Decrement (--)**: Decreases the value of an operand by 1.

페이지 2

Arithmetic operator는 수학적인 계산을 하는 operator들입니다.
이 operator들에는 덧셈
뺄셈
곱셈
나눗셈
나머지 연산
1 증가
1 감소 가 있습니다.

Example of Arithmetic Operation

```
public class ArithmeticOperators {
    public static void main (String[] args) {
        int a = 11, b = 5;
        int sum = a + b;        // 16
        int difference = a - b; // 6
        int product = a * b;    // 55
        int quotient = a / b;   // 2
        int remainder = a % b;  // 1
        int c = a++; // c = 11
        int d = ++a; // d = 13
        int e = b--; // e = 5
        int f = --b; // f = 3
        System.out.println("sum = " + sum + “\n"+ … );
    }
}
```


페이지 3

Arithmetic operation에 대한 예제를 보겠습니다.
Int type a를 11, b를 5로 initialize했습니다.
Sum, difference, product는 간단합니다.
나눗셈의 경우 int 간의 나누기는 소숫점 이하를 버립니다.
a 나누기 b, 즉 11 나누기 5 는 2.xxx 가 되기 때문에 정수부만 남기면 2가 됩니다.
Remainder, 즉, 나머지 operation의 경우
11 나누기 5의 정수 나눗셈의 경우 몫이 2, 나머지가 1이 되겠습니다.
따라서 나머지는 1이 됩니다.
c = a++ 에서는 a의 값을 일단 먼저 c에 assign한 후 a 값을 1 증가시킵니다.
따라서 c는 기존의 a값인 11을 그대로 가지게 되고
a는 1이 증가되어 이제 a 값은 12가 됩니다.
그 다음 d = ++a 는 prefix increment 이므로
a 값을 먼저 증가 시키고 나서 그 값을 d에 assign 합니다.
따라서 d 값과 a 값 모두 13이 됩니다.
마찬가지로 b– 와 –b 는 postfix와 prefix decrement로
b의 값을 1 감소 시키는 순서가 다릅니다.
e 와 f 값이 각각 5와 3이 되는 것을 잘 따져 보시기 바랍니다.

Comparison Operators

Compare two values and return a boolean result

- **Equal to (==)**: Checks if two operands are equal.
- **Not equal to (!=)**: Checks if two operands are not equal.
- **Greater than (>)**: Checks if the first operand is greater than the second.
- **Less than (<)**: Checks if the first operand is less than the second.
- **Greater than or equal to (>=)**: Checks if the first operand is greater than or equal to the second.
- **Less than or equal to (<=)**: Checks if the first operand is less than or equal to the second.

페이지 4

Comparison operator들은 두 value들의 크기를 테스트하고 그 결과를 boolean 값으로 return 합니다.
Equal to 는 equal sign 두개를 연이어 쓰는데 두 value들이 같은 경우 true를 return 합니다.
Not equal to 는 값이 다른 경우 true 입니다.
Greater than은 왼쪽이 큰 경우 true
Less than은 왼쪽이 작은 경우 true
Greater than or equal to는 왼쪽이 크거나 같은 경우 true
Less than or equal to는 왼쪽이 작거나 같은 경우 true 입니다.

Example of Comparison Operators

```java
public class ComparisonOperators {
    public static void main(String[] args) {
        int x = 10, y = 20;
        boolean isEqual = (x == y);          // false
        boolean isNotEqual = (x != y);       // true
        boolean isGreater = (x > y);         // false
        boolean isLess = (x < y);            // true
        boolean isGreaterOrEqual = (x >= y); // false
        boolean isLessOrEqual = (x <= y);    // true
    }
}
```


페이지 5

이 예제는 comparison operator의 사용 예를 보여주고 있습니다.
특이 사항이 없으므로 다른 언급 없이 넘어가도록 하겠습니다.

Logical Operators

- Combining multiple boolean expressions
- **AND (&&)**: Returns true if both operands are true.
- **OR (||)**: Returns true if at least one operand is true.
- **NOT (!)**: Reverses the logical state of its operand.

```java
public class LogicalOperators {
    public static void main(String[] args) {
        boolean a = true, b = false;
        boolean andResult = a && b; // false
        boolean orResult = a || b;  // true
        boolean notResult = !a;     // false
    }
}
```


페이지 6

Logical operator는 주로 두 개의 boolean expression을 합친 경우의 boolean 값을 return 합니다.
AND, OR, NOT 의 세 가지가 있으며
이 operation들의 결과는 truth table, 즉, 진리표에 의거하여 계산합니다.
특이한 사항이 없으므로 넘어가도록 하겠습니다.

Assignment Operators

- assign values to variables.
- **Simple assignment (=)**:
  - Assigns the right-hand side value to the left-hand side variable.
- **Compound assignment (+=, -=, *=, /=, %=)**:
  - Combines an arithmetic operation with assignment.

```java
public class AssignmentOperators {
    public static void main(String[] args) {
        int a = 10;
        a += 5; // a = a + 5;  // 15
        a -= 3; // a = a - 3;  // 12
        a *= 2; // a = a * 2;  // 24
        a /= 4; // a = a / 4;  // 6
        a %= 3; // a = a % 3;  // 0
    }
}
```


페이지 7

Assignment operator는 오른쪽 사이드의 값을 왼쪽 사이드의 variable에 assign 합니다.
Compound assignment는 arithmetic operator와 assignment를 합친 것으로
예를 들면 a += 5 는 a = a + 5 를 의미합니다.
나머지는 그냥 넘어가도록 하겠습니다.

Negative Integer (1/2)

- Negative number representation = 2's complement of positive number
  - 1's complement of positive a = ~a
  - 2's complement of positive a = ~a + 1

```
public class NegativeInteger {
    public static void main(String[] args) {
        int a = 5;   // a =  5     (00000101)
        a = ~a;      // a = ~a     (11111010) 1's complement of a
        a += 1;      // a = ~a + 1 (11111011) 1's complement + 1 = 2's complement
        a = -5;      // a = -5     (11111011) 2's complement 5 = -5
    }
}
```


페이지 8

Java 언어에서 음의 정수의 binary 표현은 2's complement를 사용합니다.
먼저 양수인 a 에서 시작하여 1's complement를 구합니다.
그러면 a의 모든 bit가 반전되어 0은 1이 되고 1은 0이 됩니다.
그 후에 1's complement 결과에 1을 더해 주면 최종 음수 표현이 나옵니다.

예제 프로그램을 보면
a는 양수 5로 주어졌습니다.
~a 로 a의 1's complement를 구합니다.
그 다음에 1을 더해 2's complement를 구합니다.
-5와 5의 2's complement를 binary로 프린트 해 보면
두 표현이 같은 것을 알 수 있습니다.
한가지 관찰할 수 있는 것은
양수는 모두 맨 왼쪽 bit가 0으로 시작되고
음수는 1로 시작된다는 것입니다.
따라서 맨 왼쪽 bit를 sign bit라 부릅니다.

Negative Integer (2/2)

- From 2's complement to negative integer
  - (-a) - 1 = 1's complement of a
  - ~((-a) - 1) = a
  - a is positive integer, so the original binary number represents the negative integer -a
- Ex) A = 11111011
  - A - 1 = 11111010
  - ~(A - 1) = 00000101 = +5
  - So A represents -5

페이지 9

그러면 2's complement로 표현된 음수가
실제로 어떤 정수를 나타내는 지
어떻게 알아낼 수 있을까요?
그것은 2's complement를 구한
역방향으로 계산을 하면 됩니다.
먼저 음수표현에서 1을 뺍니다.
그것을 반전하여 양의정수로 만듭니다.
그러면 -를 붙여 원래 음의 정수가 어떤 수 였는지를 알아낼 수 있습니다.
예를 하나 들어보겠습니다.
A의 표현이 11111011 이었다고 합시다.
이것은 1로 시작하니 분명히 음의 정수 입니다.
여기에서 1을 뺀 A - 1은 11111010 이 됩니다.
그리고 이것을 반전하니 00000101 이 되는데
이 수는 +5를 나타냅니다.
따라서 원래의 수 -A 의 값은 -5 였음을 알 수 있습니다.

Bitwise Operators

Bitwise operators perform operations on bits.

- **AND (&)**: Performs a bitwise AND.
- **OR (|)**: Performs a bitwise OR.
- **XOR (^)**: Performs a bitwise XOR.
- **NOT (~)**: Performs a bitwise NOT.
- **Left shift (<<)**: Shifts bits to the left.
- **Right shift (>>)**: Shifts bits to the right.

페이지 10

Bitwise operator는 variable value로 계산하는 것이 아니라
value의 bit 표현으로 logical 및 shift operation을 합니다.
여기에는 and, or, xor, not 의 logical operation과
Left shift, right shift 가 있습니다.
다음 slide에서 예제를 보도록 하겠습니다.

Example of Bitwise Operators

```java
public class BitwiseOperators {
    public static void main(String[] args) {
        int a = 5, b = -3;          // a =  5       (00000101)
                                    // b = -3       (11111101)
        int andResult = a & b;      // a & b =  5   (00000101)
        int orResult = a | b;       // a | b = -3   (11111101)
        int xorResult = a ^ b;      // a ^ b = -8   (11111000)
        int notResult = ~a;         // ~a = -6      (11111010)
        int leftShift = a << 1;     // a << 1 = 10  (00001010)
        int leftShiftB = b << 1;    // b << 1 = -6  (11111010)
        int rightShift = a >> 1;    // a >> 1 =  2  (00000010)
        int rightShiftB = b >> 1;   // b >> 1 = -2  (11111110)
        int right2Shift = a >> 2;   // a >> 2 =  1  (00000001)
        int right2ShiftB = b >> 2; // b >> 2 = -1  (11111111)
    }
}
```


페이지 11

a는 5, b는 -3으로 초기화 되었는데,
Binary 표현은 각각 00000101, 11111101 입니다.
먼저 a bitwise and (&) b 는 각 대응하는 bit가 모두 1일 경우에만 결과 bit가 1이 됩니다.
따라서 결과는 000000101 이 됩니다.
다음, a bitwise or (|) b 는 각 대응하는 bit 중 하나만 1이라도 결과 bit가 1이 됩니다.
따라서 결과는 11111101 이 됩니다.
다음, a bitwise xor (^) b는 각 대응하는 bit가 다를 경우에만 결과 bit가 1이 됩니다.
따라서 결과는 11111000 이 됩니다.
다음, not (~) a 는 a의 bit를 반전하는 것, 즉 1's complment를 구하는 것입니다.
따라서 결과는 11111010 이 됩니다.
다음 a left shift 1 은 맨 왼쪽의 sign bit를 보전하면서 bit를 하나씩 왼쪽으로 shift합니다.
오른쪽에 새로 들어오는 bit는 0을 넣습니다.
따라서 결과는 00001010 이 됩니다.
다음 b left shift 1 을 할 때에는 맨 왼쪽의 bit가 1로 보전되는지를 보아야 합니다.
다행히 1이 보전되고 있으니 하나씩 왼쪽으로 밀고
오른쪽 끝에 0을 새로 넣어주면 11111010 이 됩니다.
주의해서 보아야 할 것은 b >> 1 입니다.
이 때 왼쪽에서 새로 들어오는 bit는 0이 아니라 1이 되어야 합니다.
원래의 수 b가 음수라서 sign bit가 1이 였기 때문입니다.
따라서 b >> 1 의 값은 11111110 이 됩니다.
다음 a >> 2와 b >> 2 는 각각 a와 b를 right shift하는 것을 두번 연속으로 하면 됩니다.
결과를 잘 확인하기 바랍니다.

Ternary Operator

- The ternary operator is a shorthand for the if-else statement.
- **Syntax**: condition ? value if true : value if false

```java
public class TernaryOperator {
    public static void main(String[] args) {
        int a = 10, b = 20;
        int max = (a > b) ? a : b; // 20
    }
}
```


페이지 12

Ternary Operator는 if-else 문을 간단히 표현한 것입니다.
(Condition) ? (true_part) : (false_part) 형태 입니다.
condition이 true 이면 true_part를, 아니면 false_part를 return 합니다.
이를 if-else로 바꾸면
If (Condition) true_part
else false_part 가 되겠습니다.
예제 프로그램에서 a 가 b 보다 크다는 condition이 true 라면 a를,
아니라면 b를 max에 assign 합니다.

if Statement (1/2)

```java
int num = 5;
if (num > 5) {
    System.out.println("Number is greater than 5");
}

num = 2;
if (num > 5) {
    System.out.println("Number is greater than 5");
}
else {
    System.out.println("Number is not greater than 5");
}
```


페이지 13

If 문의 body는 조건이 맞으면 실행이 됩니다.
첫번 파트는 num 이 5보다 크다는 조건이 맞다면 block안의 body가 실행됩니다.
두번째 파트에서는 num이 5보다 크다는 조건이 맞다면 위의 block이
조건이 만족되지 않는다면, 즉 조건을 계산해 본 결과가 false라면
아래의 else block이 실행됩니다.

if Statement (2/2)

```
if (A) { U }
else if (B) { V }
else if (C) { X }
else { Y }
Z

if condition A is true, U and goto Z
if condition B is true, V and goto Z
if condition C is true, X and goto Z
else Y and goto Z
```


페이지 14

이 multiple if-else 문은
여러가지 condition들 중에 맞는 하나를 골라서 수행하도록 합니다.
Condition A가 true이면 U를 수행한 후 Z로 건너뛰고,
Condition B가 true이면 V를 수행한 후 Z로 건너뛰고,
Condition C가 true이면 X를 수행한 후 Z로 건너뛰고,
A, B, C가 모두 아닌 경우 Y를 수행하고 Z로 진행하게 됩니다.
뒤에 나오는 switch 문과 비슷한 역할을 수행한다고 보면 되겠습니다.

switch Statement (1/3)

```java
import java.util.Scanner;

public class SwitchStatement {
    public static void main(String[] args) {
        int score;
        char grade;
        Scanner keyboard = new Scanner(System.in);

        System.out.print("What is your score? ");
        score = keyboard.nextInt();
        int scoreOverTen = score / 10;
```

```
Prompt: What is your score? 73
```


페이지 15

Switch 문의 활용법을 보여주는 예제를 보겠습니다.
Keyboard input을 받는데 사용하는 Scanner class object를 활용하기 위해
java.util.Scanner package를 import 하였습니다.
점수를 받아들여 저장할 int score variable과
성적 문자를 저장할 character grade를 declare 하였습니다.
Keyboard 입력을 받을 Scanner object를 하나 생성하고
Screen에 "What is your score?" 라는 prompt를 보여줍니다.
keyboard로 사용자가 점수를 입력하면 이를 score로 받아 들입니다.
Keyboard 인풋에 대해서는 다음 슬라이드 노트에서 자세히 살펴보겠습니다.
이제 score를 10으로 나눈 몫을 scoreOverTen에 저장해서,
9, 8, 7, .. 과 같이 한자리 숫자 점수로 만들었습니다.

switch Statement (2/3)

```
switch (scoreOverTen) {
    case 10:
    case 9:
        grade = 'A';
        break;
    case 8:
        grade = 'B';
        break;
    default:
        grade = 'C';
}

System.out.println("Score: " + score + " Grade: " + grade + "\n");
```


페이지 16

Switch keyword 다음의 condition인 scoreOverTen 의 값으로 case를 정하게 됩니다.
scoreOverTen이 10 또는 9일 경우, 즉, score가 90에서 100점 일 경우,
A 학점을 주고 break를 만나 switch문을 빠져나오게 됩니다.
scoreOverTen이 8일 경우, 즉, 80점대 점수일 경우
B 학점을 주고 break를 만나 switch문을 빠져나옵니다.
위의 두 case가 아닌 경우에는
Default case를 수행하여 C 학점을 부여합니다.
마지막으로 점수와 학점을 프린트합니다.

switch Statement (3/3)

```
        System.out.print("Choose your menu (Americano, CafeLatte): ");
        String menu = keyboard.next();
        int sales = 0;

        switch (menu) {
            case "Americano":
                sales += 3500;
                break;
            case "CafeLatte":
                sales += 4500;
                break;
            default:
                System.out.println("Wrong coffee menu.. system exit...");
                System.exit(0);
        }

        System.out.println("Sales: " + sales);
    }
}
```


페이지 17

이 부분은 switch 문이 condition으로 String type을 사용할 수 있다는 것을
보여주고 있습니다.
Americano와 CafeLatte 중에 하나의 coffee를 선택하여
그 이름을 입력하게 합니다.
Sales 변수를 0으로 초기화 하여 주문이 들어오는 가격을
오늘 매출에 더하려 합니다.
입력을 저장한 String 변수 menu가 “Americano” 인 경우
sales에 3500을 더한 후
break를 만나 switch문을 빠져 나옵니다.
menu가 “CafeLatte” 인 경우
sales에 4500을 더한 후
break를 만나 switch문을 빠져 나옵니다.
만약 위의 두가지 case가 아닌
다른 coffee 이름이 입력되는 경우
Coffee 주문이 잘못되었다고 프린트 하고
프로그램을 강제로 끝내게 됩니다.
프로그램을 강제로 끝내는데 사용하는 명령은 System.exit(0) 입니다.
Switch를 빠져 나온 후 현재의 sales 값을 프린트 합니다.

for Loop (1/2)

```java
public class ForLoop {
    public static void main(String[] args) {
        // basic for loop
        for (int i = 0; i < 5; i++) {
            System.out.println(i);
        }
        // nested for loop
        for (int i = 1; i <= 3; i++) {
            for (int j = 1; j <= 3; j++) {
                System.out.println("i: " + i + ", j: " + j);
            }
        }
```

```
0
1
2
3
4

i: 1, j: 1
i: 1, j: 2
i: 1, j: 3
i: 2, j: 1
i: 2, j: 2
i: 2, j: 3
i: 3, j: 1
i: 3, j: 2
i: 3, j: 3
```


페이지 18

For loop는 반복 횟수가 미리 정해져 있는 경우 사용하기 편리합니다.
Basic한 for loop는 인덱스가 되는 변수의 초기값, 반복 유지 조건,
인덱스 변화식으로 이루어 집니다.
예를 들면 이 경우에는 i 값이 0부터 시작되고,
i가 5보다 작은 동안 계속 반복되며
반복마다 i를 1씩 증가 시킵니다.
그래서 output은 0, 1, 2, 3, 4 가 나오게 됩니다.

그 아래 쪽은 nested for loop 입니다.
i가 1부터 3까지 1씩 증가하는 동안
각 i에 대해 j는 1부터 3까지 1씩 증가하게 됩니다.
따라서 가운데에 있는 System.out.println 문은
3 곱하기 3 은 9. 즉, 아홉번 수행되게 됩니다.

output에 보면 i가 1인 동안 j가 1, 2, 3으로 차례로 변하고
다시 i가 2인 동안 j가 변하고 하는 것을 볼 수 있습니다.

for Loop (2/2)

```
        // for-each
        int[] numbers = {1, 2, 3, 4, 5};
        for (int n : numbers) {
            System.out.println(n);
        }
    }
}
```


페이지 19

For 문에는 array (배열) 내의 각 element들에 대해
순서대로 주어진 일을 하게 하는 기능도 있습니다.
이 예에서는 numbers array가 1, 2, 3, 4, 5 값을 가지고 있습니다.
for each 기능을 이용하여
for (int number : numbers) 라고 해 주면
numbers array 내의 모든 element 들을 순서대로 돌면서
variable n이 1, 2, 3, 4, 5 값을 차례로 가지게 됩니다.
따라서 위 예제의 output은 1, 2, 3, 4, 5 가 됩니다.

while Loop

```java
public class WhileLoop {
    public static void main(String[] args) {
        int i = 0;
        while (i < 5) {
            System.out.println(i);  // 0 1 2 3 4
            i++;
        }

        i = 0;
        while (true) {  // infinite loop
            if (i >= 5) {  // using break statment to exit from the loop
                break;
            }
            System.out.println(i);  // 0 1 2 3 4
            i++;
        }
    }
}
```


페이지 20

while loop는 반복문을 구성하는데 있어 가장 자유도가 높은 control structure 입니다.
i를 0으로 초기화 한 후
i를 1씩 증가시키면서
i < 5 의 condition을 만족하는 동안
while문의 body를 실행합니다.
따라서 output은 0 1 2 3 4 가 됩니다.
i를 다시 0으로 초기화 하고
두번째 while loop에서는
괄호안의 condition을 true로 하여
무한 loop를 실행하도록 만들어 놓았습니다.
대신 body 안에 i가 5보다 같거나 커진 경우
break를 실행하면서 while loop를 탈출하게 됩니다.
따라서 두번째 while loop의 output은
첫번째 while 문과 마찬가지로
0 1 2 3 4 가 됩니다.

do-while Loop

```java
public class DoWhileLoop {
    public static void main(String[] args) {
        int i = 0;
        do {
            System.out.println(i); // 0 1 2 3 4
            i++;
        } while (i < 5);

        i = 0;
        do {
            System.out.println("printed at least once.");
            i++;
        } while (i < 0);

    }
}
```


페이지 21

do while loop는 while loop와는 달리 while과 condition이 맨 아래 있습니다.
i를 0으로 초기화 하였고
do while 문을 시작하는 do keyword가 나옵니다.
Body 안에서 i를 print 하고
i를 1 증가 시킵니다.
Body 끝에 while 과 condition이 자리하고 있습니다.
여기서는 i < 5 라는 condition을 test하는데
이 condition을 만족하면 다시 do로 올라가서 body를 반복하는 것입니다.
따라서 첫번째 do while loop의 output은 0 1 2 3 4 입니다.
두번째 do while loop 전에 i를 다시 0으로 하였습니다.
do while loop의 특징은 반드시 body를 한번 이상은 실행 한다는 것입니다.
따라서 "printed at least once" 라는 String은 첫번째 body 실행 시에 print됩니다.
그러나 i가 바로 1로 증가되고
아래의 while condition은 (i < 0) 이기 때문에
i가 1이 되는 첫번째 비교때 do while loop를 빠져나오게 됩니다.

break and continue

```java
public class LoopControl {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            if (i == 5) {
                break; // exit out the for loop
            }
            System.out.println(i);
        }
        for (int i = 0; i < 10; i++) {
            if (i % 2 == 0) {
                continue; // go up to for
            }
            System.out.println(i);
        }
    }
}
```

```
0
1
2
3
4
1
3
5
7
9
```


페이지 22

이 예제에서는 break와 continue 문의 기능을 보여줍니다.
첫번째 for loop에서 i가 0부터 증가해서 5가 되는 순간
break가 실행되고 for loop를 빠져나오게 됩니다.
따라서 output은 0 1 2 3 4 입니다.
두번째 for loop에서는 i를 2로 나눈 나머지가 0이 되는 경우에는
즉 i가 짝수인 경우에는
continue문이 실행되어 그 아래에 있는 print 명령을 실행하지 않고
for가 있는 line으로 건너뛰게 됩니다.
따라서 i가 홀수 일때에만 print되기 때문에
두번째 for loop의 output은 1 3 5 7 9 입니다.