

Object – Oriented Programming

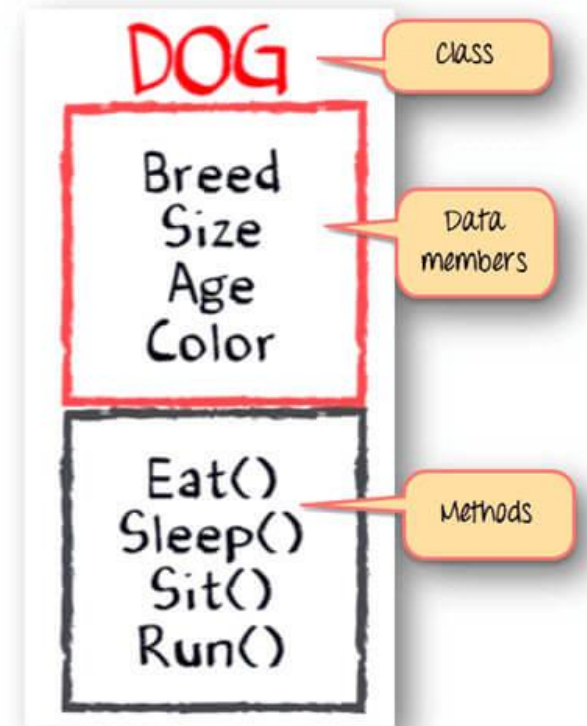
LAB #2. VARIABLES AND STRINGS

Java Language

Java : Object Oriented Programming Language (OOP)

- 객체가 작업을 수행한다.
- 객체는 다른 객체의 작업의 영향을 받는다.
- 객체의 작업을 Method라고 한다.

Java Application : 메인 Method를 사용하는 Java



Java Language

Display 1.1 A Sample Java Program

```
1 public class FirstProgram
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello reader.");
6         System.out.println("Welcome to Java.");
7
8         System.out.println("Let's demonstrate a simple calculation.");
9         int answer;
10        answer = 2 + 2;
11        System.out.println("2 plus 2 is " + answer);
12    }
```

Annotations:

- Name of class (program) points to `FirstProgram`.
- The main method points to `main(String[] args)`.
- A red box highlights `int answer;` on line 9.

SAMPLE DIALOGUE 1

Hello reader.
Welcome to Java.
Let's demonstrate a simple calculation.
2 plus 2 is 4

1. 변수명은 숫자로 시작할 수 없다.
2. 모든 변수는 문자, 숫자, 밑줄(underscore)로만 이루어져야 한다.

Keywords

- 일부 단어는 JVM Library에 의해 선언되어 있으므로 식별자로 사용할 수 없다.

ex) int, String, System, 등

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    int System = 0;  
  
    System.out.println("123");  
}  
  
public int() {  
  
}
```

Variables

- 프로그램 내에서 데이터를 저장하는 용도로 사용

ex) int x = 5;

x

5

- | | | |
|-----------|---|--------------|
| - float | } | 소수점 |
| - double | | |
| - boolean | | True / False |
| - char | | 한 글자의 문자 |
| - byte | } | 정수형 숫자 |
| - short | | |
| - int | | |
| - long | | |

integer types

Type	Bytes	Minimum value	Maximum value
byte	1	$-2^7 = -128$	$2^7 - 1 = 127$
short	2	$-2^{15} = -32,768$	$2^{15} - 1 = 32,767$
int	4	$-2^{31} = -2,147,483,648$	$2^{31} - 1 = 2,147,483,647$
long	8	$-2^{63} = -9,223,372,036,854,775,808$	$2^{63} - 1 = 9,223,372,036,854,775,807$

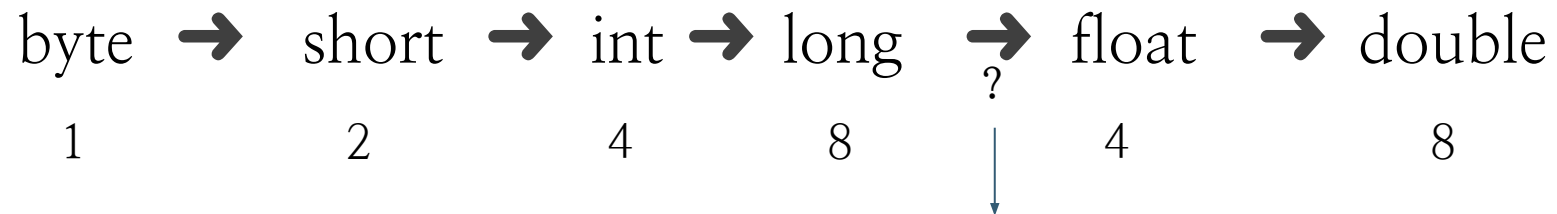
Type Casting

```
int intVariable;  
intVariable = 42;
```

```
double doubleVariable;  
doubleVariable = intVariable;
```

doubleVariable의 값 :
42.0

다음과 같이 더 낮은 타입의 값에 모든 타입의 값을 할당할 수 있다.



<https://diveintodata.org/2014/04/27/float과-long-타입의-implicit-casting/>

boolean type

- boolean 타입은 단 두가지 값만 갖는다.

- true
- false

- boolean의 특정 연산자

- &&
- ||
- !=
- ==

```
boolean x = true;  
boolean y = false;
```

```
System.out.println(x&&y);    // false  
System.out.println(x||y);    // true  
System.out.println(x!=y);    // true  
System.out.println(x==y);    // false
```


Constants

- 상수는 절대 바꿀 수 없는 값이다.
- 상수를 선언하는 방법은 다음과 같다.

```
final int x = 5;
```

Expressions

- 표현식은 다음과 같이 사용한다.

ex)

```
int expression = 4 + 2 * 5;
```

```
System.out.println(5 / 2.0);
```

- Java에서의 Expression 규칙
 - 각 연산자는 우선 순위가 있다.
 - * 와 / 연산자가 + 와 - 연산자보다 우선순위가 높다.
 - 부동 소수점이 사용되는 경우 결과는 부동소수점이다.

Expressions – Priority of Operators

우선순위	연산자	내용
<div>높음</div> <div>↑</div> <div>↓</div> <div>낮음</div>	() , []	괄호
	!, ~, ++, --	부정, 증감 연산자
	*, /, %	곱셈, 나눗셈
	+, -	덧셈, 뺄셈
	<, <=, >, >=	비교
	==, !=	Boolean 연산자 (비교)
	&&	Boolean 연산자 (and)
		Boolean 연산자 (or)

The String Class

- String 클래스는 문자열을 저장하고 처리하는데 사용한다.
- 또한, String 클래스 내에는 문자열을 편하게 처리할 수 있는 여러 Method가 있다.
 - String s = “Java is fun.”;

0	1	2	3	4	5	6	7	8	9	10	11
J	a	v	a		i	s		f	u	n	.

The String Class

String a = “Hello”;

String b = “World”;

String c = a + b;

c

Hello World


- 두 개의 String 변수를 + 연산자를 사용하여 합칠 수 있다.

The String Class

```
String a = "Ten";
```

```
int n = 4;
```

```
String c = a + n;
```

c 

- 다음과 같이 문자열(String), 정수(int) 값을 합쳐 String 형식으로 변환이 가능하다.

The String Class

```
int a = 1;
```

```
int b = 2;
```

```
String c = a + b; ❌
```

- String 변수에 담더라도 더하는 값 중에 String 형식의 값이 없으면 에러가 발생한다.

String Class

Method	설명
substring	한 문자열에서 내용의 일부를 반환
split	문자열을 매개변수로 지정된 분리자로 나누어 문자열 배열 형태로 반환
contains	지정된 문자열이 포함되었는지 검사
endsWith	지정된 문자열로 끝나는지 검사
equals	지정된 문자열과 같은 지 검사
replace	문자열 중에 A를 B로 변경
toLowerCase	모든 문자열을 소문자로 변환
toUpperCase	모든 문자열을 대문자로 변환
trim	문자열의 양 끝의 공백을 제거
valueOf	지정된 값을 문자열로 변환
length	문자열 길이를 반환
charAt	해당 Index의 문자를 반환

substring

String substring(int begin)

String substring(int begin, int end)

한 문자열에서 일부만 추출하는 메소드

```
String s = "Java Programming.txt"
```

```
String s1 = s.substring(0,4);
```

```
String s2 = s.substring(10);
```

```
String s3 = s.substring( s.length() - 4 );
```

결과

s1 = "Java"

s2 = "amming.txt"

s3 = ".txt"

split

```
String[] split(String regex)
```

문자열을 지정된 분리자로 나누어 문자열 배열 형태로 반환한다.

```
String colors = "black,white,red,blue,yellow";
```

```
String[] color_arr = colors.split(",");
```

```
color_arr.length;
```

결과

```
arr[0] = "black"  
arr[1] = "white"  
arr[2] = "red"  
arr[3] = "blue"  
arr[4] = "yellow"  
color_arr.length = 5
```

split(".") delimiter dot java error split(".") -> ("\\.")

<https://stackoverflow.com/questions/7935858/the-split-method-in-java-does-not-work-on-a-dot>

contains

Boolean contains(String s)

지정된 문자열이 포함되었는지 검사한다.

```
String s = "abcdefg";
```

```
Boolean b = s.contains("ef");
```

결과

b = true

endsWith

Boolean endsWith(String suffix)

문자열의 끝에 해당 문자열이 있는지 검사한다

↔ startsWith(String prefix)

```
String file = "Hello.cpp";
```

```
Boolean b = file.endsWith("cpp");
```

결과

b = true

equals

Boolean equals(String s)

지정된 문자열과 같은지 검사한다.
대소문자를 구분한다.

```
String s = "Hello World";  
Boolean b = s.equals("Hello World");  
Boolean b2 = s.equals("hello world");
```

결과

b = true
b2 = false

※ 대소문자 구분 하지 않고 검사하는 메소드 : equalsIgnoreCase(String s)

compareTo

Boolean compareTo(String s)

지정된 문자열과 같은지 각 문자의 유니코드값에 근거해 검사한다.
반환 값은 int형이고, 대소문자를 구분한다.

```
String s = "Hello World";  
int i = s.compareTo("Hello World");  
int j = s.compareTo("hello world");
```

결과

i = 0
j = -32

※ 대소문자 구분 하지 않고 검사하는 메소드 : compareToIgnoreCase(String s)

replace

String replace(String a, String b)

문자열에 있는 a 문자열을 b로 변경한다.

String s = "Gildong Hong"

String n = s.replace("Hong", "Go");

결과

n = "Gildong Go"

toLowerCase

String toLowerCase()

모든 문자열을 소문자로 변환하여 반환한다.

```
String s = "Hello";
```

```
String n = s.toLowerCase();
```

결과

n = "hello"

toUpperCase

String toUpperCase()

모든 문자열을 대문자로 변환하여 반환한다.

```
String s = "Hello";
```

```
String n = s.toUpperCase();
```

결과

n = "HELLO"

trim

String trim()

문자열 양 끝의 공백을 제거한다.

```
String s = "  Hello  ";
```

```
String n = s.trim();
```

결과

n = "Hello"

valueOf

static String valueOf()

Boolean
char
int
long
float
double

특정 값을 문자열로 변환하여 반환한다.

```
String a = String.valueOf(true);
```

```
String b = String.valueOf(100);
```

```
String c = String.valueOf('c');
```

```
String d = String.valueOf(10.0);
```

결과

```
a = "true"  
b = "100"  
c = "c"  
d = "10.0"
```

length

```
int length()
```

문자열의 길이를 반환한다.

```
String s = "Hello";
```

```
int n = s.length();
```

결과

n = 5

charAt

`char charAt(int index)`

해당 index의 문자를 반환한다.

`String s = "abcde";`

`char c = s.charAt(3);`

결과

`c = d`

Control Flow

The `if-then` statement is the most basic of all the control flow statements. It tells your program to execute a certain section of code *only if* a particular test evaluates to `true`.

```
void applyBrakes() {  
    // the "if" clause: bicycle must be moving  
    if (isMoving){  
        // the "then" clause: decrease current speed  
        currentSpeed--;  
    }  
}
```

In addition, the opening and closing braces are optional, provided that the "then" clause contains only one statement:

```
void applyBrakes() {  
    // same as above, but without braces  
    if (isMoving)  
        currentSpeed--;  
}
```

The `if-then-else` statement provides a secondary path of execution when an "if" clause evaluates to `false`.

```
void applyBrakes() {  
    if (isMoving) {  
        currentSpeed--;  
    } else {  
        System.err.println("The bicycle has already stopped!");  
    }  
}
```

Control Flow

The `switch` statement can have a number of possible execution paths. A `switch` works with the `byte`, `short`, `char`, and `int` primitive data types.

It also works with *enumerated types* (`Enum Types`), the `String` class, and a few special classes that wrap certain primitive types: `Character`, `Byte`, `Short`, and `Integer`.

Deciding whether to use `if-then-else` statements or a `switch` statement is based on **readability and the expression** that the statement is testing.

An `if-then-else` statement can test expressions based on **ranges of values or conditions**,

whereas a `switch` statement tests expressions based only on **a single integer, enumerated value, or `String` object**.

In this case, `August` is printed to standard output.

```
public class SwitchDemo {
    public static void main(String[] args) {

        int month = 8;
        String monthString;
        switch (month) {
            case 1: monthString = "January";
                    break;
            case 2: monthString = "February";
                    break;
            case 3: monthString = "March";
                    break;
            case 4: monthString = "April";
                    break;
            case 5: monthString = "May";
                    break;
            case 6: monthString = "June";
                    break;
            case 7: monthString = "July";
                    break;
            case 8: monthString = "August";
                    break;
            case 9: monthString = "September";
                    break;
            case 10: monthString = "October";
                    break;
            case 11: monthString = "November";
                    break;
            case 12: monthString = "December";
                    break;
            default: monthString = "Invalid month";
                    break;
        }
        System.out.println(monthString);
    }
}
```

Control Flow

This is the output from the code (without `break;`):

August
September
October
November
December

```
public class SwitchDemoFallThrough {  
  
    public static void main(String[] args) {  
        java.util.ArrayList<String> futureMonths =  
            new java.util.ArrayList<String>();  
  
        int month = 8;  
  
        switch (month) {  
            case 1: futureMonths.add("January");  
            case 2: futureMonths.add("February");  
            case 3: futureMonths.add("March");  
            case 4: futureMonths.add("April");  
            case 5: futureMonths.add("May");  
            case 6: futureMonths.add("June");  
            case 7: futureMonths.add("July");  
            case 8: futureMonths.add("August");  
            case 9: futureMonths.add("September");  
            case 10: futureMonths.add("October");  
            case 11: futureMonths.add("November");  
            case 12: futureMonths.add("December");  
                    break;  
            default: break;  
        }  
  
        if (futureMonths.isEmpty()) {  
            System.out.println("Invalid month number");  
        } else {  
            for (String monthName : futureMonths) {  
                System.out.println(monthName);  
            }  
        }  
    }  
}
```


Control Flow

`SwitchDemo2`, shows how a statement can have multiple `case` labels. The code example calculates the number of days in a particular month:

This is the output from the code:

Number of Days = 29

```
class SwitchDemo2 {
    public static void main(String[] args) {

        int month = 2;
        int year = 2000;
        int numDays = 0;

        switch (month) {
            case 1: case 3: case 5:
            case 7: case 8: case 10:
            case 12:
                numDays = 31;
                break;
            case 4: case 6:
            case 9: case 11:
                numDays = 30;
                break;
            case 2:
                if (((year % 4 == 0) &&
                    !(year % 100 == 0))
                    || (year % 400 == 0))
                    numDays = 29;
                else
                    numDays = 28;
                break;
            default:
                System.out.println("Invalid month.");
                break;
        }
        System.out.println("Number of Days = "
                           + numDays);
    }
}
```

Control Flow

you can use a `String` object in the `switch` statement's expression.

```
public class StringSwitchDemo {  
    public static int getMonthNumber(String month) { ... }  
    public static void main(String[] args) { ... }  
}
```

```
public static void main(String[] args) {  
    String month = "August";  
    int returnedMonthNumber =  
        StringSwitchDemo.getMonthNumber(month);  
  
    if (returnedMonthNumber == 0) {  
        System.out.println("Invalid month");  
    } else {  
        System.out.println(returnedMonthNumber);  
    }  
}
```

The output from this code is 8.

Note: This example checks if the expression in the `switch` statement is `null`. Ensure that the expression in any `switch` statement is not null to prevent a `NullPointerException` from being thrown.

```
public static int getMonthNumber(String month) {  
    int monthNumber = 0;  
  
    if (month == null) {  
        return monthNumber;  
    }  
    switch (month.toLowerCase()) {  
        case "january":  
            monthNumber = 1;  
            break;  
        case "february":  
            monthNumber = 2;  
            break;  
        case "march":  
            monthNumber = 3;  
            break;  
        case "april":  
            monthNumber = 4;  
            break;  
        case "may":  
            monthNumber = 5;  
            break;  
        case "june":  
            monthNumber = 6;  
            break;  
        case "july":  
            monthNumber = 7;  
            break;  
        case "august":  
            monthNumber = 8;  
            break;  
        case "september":  
            monthNumber = 9;  
            break;  
        case "october":  
            monthNumber = 10;  
            break;  
        case "november":  
            monthNumber = 11;  
            break;  
        case "december":  
            monthNumber = 12;  
            break;  
        default:  
            monthNumber = 0;  
            break;  
    }  
  
    return monthNumber;  
}
```

Control Flow

The `while` statement continually executes a block of statements while a particular condition is `true`. Its syntax can be expressed as:

```
while (expression) {  
    statement(s)  
}
```

The Java programming language also provides a `do-while` statement, which can be expressed as follows:

```
do {  
    statement(s)  
} while (expression);
```

Control Flow

The `for` statement provides a compact way to iterate over a range of values. Programmers often refer to it as the "for loop" because of the way in which it repeatedly loops until a particular condition is satisfied. The general form of the `for` statement can be expressed as follows:

```
for (initialization; termination; increment) {  
    statement(s)  
}
```

```
class EnhancedForDemo {  
    public static void main(String[] args){  
        int[] numbers =  
            {1,2,3,4,5,6,7,8,9,10};  
        for (int item : numbers) {  
            System.out.println("Count is: " + item);  
        }  
    }  
}
```

```
class ForDemo {  
    public static void main(String[] args){  
        for(int i=1; i<11; i++){  
            System.out.println("Count is: " + i);  
        }  
    }  
}
```

```
// infinite loop  
for ( ; ; ) {  
    // your code goes here  
}
```

When using this version of the `for` statement, keep in mind that:

- The *initialization* expression initializes the loop; it's executed once, as the loop begins.
- When the *termination* expression evaluates to `false`, the loop terminates.
- The *increment* expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment or decrement a value.

Control Flow

The `break` statement has two forms: labeled and unlabeled.

This program's output is:

```
Found 12 at index 4
```

```
class BreakDemo {
    public static void main(String[] args) {
        int[] arrayOfInts =
            { 32, 87, 3, 589,
              12, 1076, 2000,
              8, 622, 127 };
        int searchfor = 12;

        int i;
        boolean foundIt = false;
        for (i = 0; i < arrayOfInts.length; i++) {
            if (arrayOfInts[i] == searchfor) {
                foundIt = true;
                break;
            }
        }
        if (foundIt) {
            System.out.println("Found " + searchfor + " at index " + i);
        } else {
            System.out.println(searchfor + " not in the array");
        }
    }
}
```

Control Flow

The `break` statement has two forms: labeled and unlabeled.

This is the output of the program.

Found 12 at 1, 0

```
class BreakWithLabelDemo {
    public static void main(String[] args) {
        int[][] arrayOfInts = {
            { 32, 87, 3, 589 },
            { 12, 1076, 2000, 8 },
            { 622, 127, 77, 955 }
        };
        int searchfor = 12;

        int i;
        int j = 0;
        boolean foundIt = false;

        search:
        for (i = 0; i < arrayOfInts.length; i++) {
            for (j = 0; j < arrayOfInts[i].length;
                j++) {
                if (arrayOfInts[i][j] == searchfor) {
                    foundIt = true;
                    break search;
                }
            }
        }

        if (foundIt) {
            System.out.println("Found " + searchfor + " at " + i + ", " + j);
        } else {
            System.out.println(searchfor + " not in the array");
        }
    }
}
```

Control Flow

```
class ContinueDemo {
    public static void main(String[] args) {
        String searchMe = "peter piper picked a " + "peck of pickled peppers";
        int max = searchMe.length();
        int numPs = 0;

        for (int i = 0; i < max; i++) {
            // interested only in p's
            if (searchMe.charAt(i) != 'p')
                continue;

            // process p's
            numPs++;
        }
        System.out.println("Found " + numPs + " p's in the string.");
    }
}
```

Here is the output of this program:

```
Found 9 p's in the string.
```

Control Flow

Here is the output from this program.

Found it

```
class ContinueWithLabelDemo {
    public static void main(String[] args) {

        String searchMe = "Look for a substring in me";
        String substring = "sub";
        boolean foundIt = false;

        int max = searchMe.length() -
            substring.length();

        test:
        for (int i = 0; i <= max; i++) {
            int n = substring.length();
            int j = i;
            int k = 0;
            while (n-- != 0) {
                if (searchMe.charAt(j++) != substring.charAt(k++)) {
                    continue test;
                }
            }
            foundIt = true;
            break test;
        }
        System.out.println(foundIt ? "Found it" : "Didn't find it");
    }
}
```


Scanner Class

java.util에 포함되어 있는 class

➔ `import java.util.Scanner`

키보드 입력을 받는 역할을 수행

```
Scanner keyboard = new Scanner(System.in);
```

Scanner Class

Method	설명
next	키보드에서 입력된 공백까지의 값을 반환
nextInt	키보드에서 입력된 int 값을 반환
nextDouble	키보드에서 입력된 double 값을 반환
nextLine	키보드에서 입력된 'Wn'까지의 값을 반환

Scanner 사용법

1. import java.util.Scanner; 를 첫 줄에 입력
2. Scanner 객체를 main 메소드에 생성
3. 입력 값은 1줄로 입력하므로 nextLine() 메소드를 사용

```
1 package lab02;  
2  
3 import java.util.Scanner;  
4  
5 public class Lab02 {  
6     public static void main(String[] args) {  
7         Scanner scan = new Scanner(System.in);  
8         String input = scan.nextLine();  
9  
10        System.out.println(input);  
11    }  
12 }
```

Gil Dong Go, Homework.ppt

Gil Dong Go, Homework.ppt

실습

위의 메소드들을 활용하여 실습을 진행할 것

- 이름은 각자의 이름을 입력할 것(Scanner 클래스 사용)

공백

입력 : gil dong go, homework.ppt

출력 : Name Length(Korean) : 3
G.D.Go submitted Homework.pdf

실습 제출 방법

다음 실습 부터 깃랩으로 실습 코드 제출

메일 양식

제목: [OOP lab 수업번호] 이름 질문 내용

*수업번호: 12334 또는 12335

메일 주소: Lab01_Eclipse-2023 슬라이드 3페이지 참조

jehakim22oct@hanyang.ac.kr