자바 왕기초

URL	
강의 번호	andriod & web
복습	☐
유형	코드잇
자료	
작성일시	@2021년 7월 29일 오후 11:16

> 코드잇 자바 왕기초 강의로, 자바를 배우는 것과 동시에 CS에 대해서 배우는 시간을 가져봅시다.

01 Introduction to Java Programming

- 꾸준한 인기 언어, 자바
- 자바와 가상머신
- Mac 및 Windows 설치 (인텔리제이 사용)

02 Hello, Java!

- 자바와 객체 지향
  - 자바 : 처음부터 객체 지향 언어로 만들어짐
  - 객체지향 프로그래밍(Object Oriented Programming, OOP) : 프로그램을 작성하는 기법, 부품에 해당하는 객체(Object)를 먼저 만들고, 이것들을 하나씩 조립 및 연결해서 전체 프로그램을 완성하는 기법
  - 객체 지향 잘 적용된 언어는 코드의 구조가 명확하기 때문에 코드 이해 쉽고, 관리와 유지 보수가 효율적이다.

  - 자바는 객체 지향의 개념이 언어에 강하게 드러나는 특징을 가지고 있다.
    - 그래서 자바는 한 줄을 출력하는 데도, public, class, static, void, System, out 등 많은 용어가 등장한다.
    - 그리고 그만큼 많은 의미를 코드에서 전달하고 있다.
    - 객체 지향적 구조와 설계를 정확히 표현하는 것에 특화된 언어라고 할 수 있다.
      → 객체 지향을 배우지 않고 자바를 이해할 수 없다.

- 프로그래밍 기초
- Hello World

```
public class HelloWorld {
    public static void main(Stringp[] args) {
        System.out.println("Hello, World!");
    }
}
```

  - public class HelloWorld
    - public : 누구나 사용할 수 있는 클래스를 지정하는 접근 제어자,
      - 접근 제어자 : public, private, protected 등이 있다.
    - class : 객체 지향 프로그래밍의 기본 단위
    - HelloWorld : 클래스의 이름

- public static void main(String[] args)
  - main : 메소드, 모든 자바 프로그램에 항상 포함되어 있다.
  - static : (후에 소개)
  - void : 되돌려주는 값이 없다는 의미. 만약 실행이 끝난 후, 숫자형을 돌려줄 예정이라 한다면, void 대신 숫자를 의미하는 int를 쓰면 됨.
    - 메인 메소드인 main은 보통 아무 것도 되돌려주지 않기 때문에, void를 붙여준다.
  - String[] args : 메소드에 전달되는 값으로, 파라미터라고 한다.
    - String[] : 문자열
    - args : 변수
    - 이 의미는 args라는 이름의 문자열 변수가 메소드에 전달된다는 뜻
- System.out.println("Hello World!");
  - System : 자바에서 미리 직접 만들어둔 클래스, 입력과 출력 등 시스템에 관련된 기능들을 모아둔 클래스이다.
  - out : System 클래스 안에는 여러 기능이 모여 있어 System.in, System.out, System.err 등으로 분리되어 있다. out은 출력과 관련된 기능이 들어 있다.
  - println() : print(출력하다) + ln(line) → 원하는 것을 출력하고, 엔터를 치듯이 다음 줄(line)로 넘겨주는 역할을 한다. 기능을 구현한 부분이므로 메소드라고 부른다.

03 변수와 연산

- 변수
  - 변수 선언은 앞에 자료형을 써주고 뒤에 변수 이름을 써주면 된다.

```
type variableName;
```

  - 변수 이름 규칙
    1. 대소문자 구분

2. 숫자로 시작할 수 없다.
3. 밑줄(_)과 달러 표시($)를 사용할 수 있지만, 사용하지 않는 것이 좋다.
    - $: 자동 생성되는 변수명
    - _ : 보통 상수 이름에 쓰임
4. class, public과 같은 자바의 예약어는 변수명으로 쓸 수 없다.
5. 자바의 변수명은 '카멜 케이스(camelCase)'라는 기법으로 작성하는 것이 좋다. (권장사항)
    - 카멜 케이스에서 첫 번째 글자는 소문자고, 그 후에 새로운 단어의 첫 번째 글자는 대문자입니다.
    - 예시 : myName, someRidiculouslyLongName

- 값 넣어주기
    - 유형 1 : 선언, 초기값 따로

```
int age;
age = 27;
```

    - 유형 2 : 선언 + 초기값

```
int age = 27;
```

- 변수 사용 : 다른 언어와 같다. 변수에 값을 대입할 수 있고, 연산을 할 수도 있다.

```
int age = 27;
int num = 13;

System.out.println(num + age);  // 27 + 13 출력

age = num;
System.out.println(num + age);  // 13 + 13 출력

age = age + 1;
System.out.println(num + age);  // 13 + 14 출력
```

- println과 print의 차이
  - println : 출력 후 자동으로 줄 바꿈
  - print : 출력 기능. 줄 바꿈을 하지 않음

- 자료형
  - 기본 자료형(Primitive Types)

Type	Bits	Range of Values
byte	8bits	-2^7 ~ 2^7-1 (-128 ~ 127)
short	16bits	-2^15 ~ 2^15-1 (-32768 ~ 32767)
int	32bits	-2^31 ~ 2^31-1 (-2147483648 ~ 2147483647)
long	64bits	-2^63 ~ 2^63-1 (-9223372036854775808 ~ 9223372036854775807)
float	32bits	*single-precision 32-bit IEEE 754 floating point
double	64bits	*double-precision 64-bit IEEE 754 floating point
char	16bits	\u0000 ~ \uffff (0 ~ 2^15-1)
boolean	*VMD	true, false

type	기본 값
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
boolean	false

- 숫자형 : 정수형과 소수형이 존재
  - 정수형 : byte, short, int, long
    - 자바는 int를 정수형의 메인으로 사용한다. 즉, 정수를 입력하면 기본적으로 int로 간주
    - int나 다른 data type의 범위에서 벗어나는 값을 써준다면 오류가 나온다. 만약 int 범위에서 벗어난다면 long과 숫자 뒤에 L을 붙여주면 된다.

  - 소수형 : float, double
    - float과 double은 정밀도(Precision) 혹은 정확도에 차이가 있다.
    - double이 더 정밀하게 값을 보관할 수 있어서 자바는 double을 소수형의 기본으로 사용한다.
    - float를 사용하기 위해서는 소수를 쓰고, 뒤에 f를 붙이면 된다.

- 글자(Character) : char

    - char : Character 하나를 담는 자료형, 딱 글자 하나만 넣어줄 수 있고, 작은 따옴표로 글자를 둘러싸야 한다.
    - 글자 여러개를 담고 싶다면 String을 사용해야 한다. 큰 따옴표로 글자를 둘러싸야 한다.

- 불린(Boolean) : true, false가 가능. 참과 거짓을 담는 자료형
- 문자열 : String, 기본 자료형이 아니라 클래스이다.
    - 클래스를 변수의 형으로 쓰고 변수를 선언하면 그 변수는 클래스의 인스턴스를 담을 수 있다.
    - String은 큰 따옴표(")로 둘러싸인 글자들을 적어 만들 수 있다.

```
String a = "Hello, I'm ";
String b = ".";

System.out.print(a);
System.out.print(26);
System.out.println(b);

// 결과 : Hello, I'm 26.
```

- 연산자
    - 문자열 연산 : 문자열 사이에 덧셈(+) 연산자를 사용하면 문자열을 연결해줄 수 있다.
        - 문자열과 숫자를 더하면 숫자가 저절로 문자열로 바뀌고, 문자열과 불린을 더하면 불린이 저절로 문자열로 바뀐다.

```
String myString = "Hello " + "Codeit!!";
System.out.println(myString);
System.out.println("I am " + 27 + " years old.");
System.out.println("The result is " + false + ".");

// 결과
// Hello Codeit!!
// I am 27 years old.
// The result is false.
```

        - 문자열 사이에 특수한 문자를 표현하고 싶을 때, 역슬래시(\)를 사용하면 된다.

```
System.out.println("데카르트는 \"나는 생각한다. 고로 존재한다.\"라고 말했다.");

// 결과 : 데카르트는 "나는 생각한다. 고로 존재한다."라고 말했다.
```

- 이렇게 \를 이용하여 문자열 안에 확장된 표현을 하는 문자를 **이스케이프 문자(Escape Character)**라고 한다.
- 종류
  - \t : 탭
  - \b : 백스페이스
  - \n : 줄 바꿈(new line)
  - \r : 줄 바꿈 (carriage return)
  - \f : 폼 피드(form feed)
  - ' : 작은 따옴표
  - " : 큰 따옴표
  - \ : 역슬래쉬
- 줄 바꿈을 위해서는 맥에서는 "\n", 윈도우즈에서는 "\r\n"을 사용해야 한다.

- 숫자 연산
  - + : 덧셈
  - - : 뺄셈
  - * : 곱셈
  - / : 나눗셈
    - 소수형은 정수형보다 랭크가 높기 때문에 소수형과 정수형 간의 연산의 결과값으로는 소수형이 나온다.
  - % : 나머지
  - 단항 연산자 : -, ++, - - 등
  - 불린 연산

- 비교 연산자(Comparison Operators) : 두 숫자형 값을 비교할 싶을 때 사용

```
int a = 3;
int b = 5;

System.out.println(a > b);  // a가 b보다 크다
System.out.println(a >= b); // a가 b보다 크거나 같다
System.out.println(a < b);  // a가 b보다 작다
System.out.println(a <= b); // a가 b보다 작거나 같다
System.out.println(a == b); // a가 b와 같다
System.out.println(a != b); // a가 b와 같지 않다

// 결과
// false
// false
// true
// true
// false
// true
```

- 불린 연산자 (Boolean Operators)
  - AND(&&) → 양쪽 다 true이어야만 true가 나온다. 한쪽이라도 false면 false가 나온다.
  - OR(||) → 양쪽 다 false이어야만 false가 나오고, 한쪽이라도 true면 true가 나온다.
  - NOT(!) → 불린 값을 뒤집어준다.

- 형 변환
  - Literal : 소스코드의 고정된 값을 대표하는 용어

```
int myInt = 123;
byte myByte = 38;
short myShort = 2;
long myLong = 12345678910L; // 12345678910는 오류 발생
```

  - 123, 38, 2는 '정수 리터럴'이다.
  - 12345678910L은 '롱 리터럴'이다.

  - 랭크

Type	Size	Range
byte	1 byte	-128 ... 127
short	2 byte	-32,768 ... 32,767
int	4 byte	-2,147,483,648 ... 2,147,483,647
long	8 byte	-9,223,372,036,854,775,808 ... 9,223,372,036,854,775,807
float	4 byte	1.4023985 x 10^-45 ... 3.4028235 x 10^38
double	8 byte	4.940656458412465 x 10^-324 ... 1.797693134862316 x 10^308

- 위에서부터 아래로 갈수록 랭크가 높은 자료형이다.

- 형 변환 : 랭크에 따라 형 변환이 가능하다.

to \ from	byte	short	int	long	float	double
byte	-	X	X	X	X	X
short	O	-	X	X	X	X
int	O	O	-	X	X	X
long	O	O	O	-	X	X
float	O	O	O	O	-	X
double	O	O	O	O	O	-

- 바꾸고자 하는 형(to)이 기존 형(from)보다 넓은 데이터를 담을 수 있는 자료형일 경우 특별한 처리 없이 형을 변환할 수 있다.

- 타입 캐스팅(Type Casting)
  - 값(혹은 변수) 앞에 (자료형) (ex : (int) x) 을 적어두면 강제적으로 형을 변환시킬 수 있다.

- 물론 형 변환이 가능한 경우에만 가능하다.
- 숫자 자료형들 사이에서는 모두 가능하다.

```
int a = 3;
double b = (double) a;
long c = (long) a;

System.out.println(b);
System.out.println(c);

// 결과
// 3.0
// 3
```

- 더 큰 랭크의 값을 더 작은 랭크의 변수에 담는 것도 가능하지만, 데이터의 손실이 있다는 것을 주의해야 한다.

```
double pi = 3.14;
int myInt = (int) pi;  // 데이터 손실 (소수 부분)
System.out.println(myInt);

// 결과 : 3
```

```
int a = 9, b = 5;
System.out.println(a / b);
System.out.println((double) a/ b);

// 결과
// 1
// 1.8
```

04 조건문과 반복문

- if, else if, else

```
// 기본 문법

if (조건부분 1) {
    // 수행 부분 1
}

else if (조건부분 2) {
    // 수행 부분 2
}
```

```
else if (조건부분 3) {
    // 수행 부분 3
}

else {
    // 수행 부분 4
}
```

- 조건문의 기본 구조이다. C/C++와 동일하다.

- switch문
  - switch문의 조건 부분은 숫자, 문자열 등의 결과값을 내는 식이다.
  - 합격-불합격, 옳다-그르다 등과 같은 방식으로 흐름을 나누는(if-else)것보다 여러 개의 동등한 조건의 나열에는 switch문이 더 편리하다.

```
switch (i % 3) {  // i : 불린이 아닌 식, 변수, 메소드
    case 0:
        System.out.println("C 구역입니다.");
        break;

    case 1:
        System.out.println("A 구역입니다.");
        break;

    default:
        System.out.println("B 구역입니다.");
        break;
}
```

- while문

```
while (조건 부분) {
    수행 부분
}
```

  - while문은 조건 부분이 true인 동안 계속 반복해서 수행 부분을 실행하는 구문입니다.
  - 수행부분에서 조건이 false가 되도록 바꾸어 주거나 break;를 통해서 반복문을 종료시킬 수 있다.
  - while문은 반복 횟수를 정확히 알 수 없는 동작, 혹은 특별한 조건에서만 멈추어야 할 경우 등에서 자주 사용한다.

- for문과 비교하면, for문은 반복 횟수를 while문보다 직관적으로 조절할 수 있다는 점이 차이점이다.

- for문

```
for (초기화식; 종결 제어식; 증감 제어식) {
    // 수행 부분
}
```

  - for문의 조건 부분
    1. 초기화식 (initialization)
    2. 종결 제어식 (termination)
    3. 증감 제어식 (increment)

  - for문을 사용하는 경우
    - for문은 while문과 달리 초기화식이 있고 for문 안에서만 쓸 수 있는 변수를 만들 수 있다는 장점 때문에
      1. 반복의 인덱스가 필요한 경우
      2. 반복의 최대 횟수가 정해진 경우
      3. 갯수가 정해진 데이터 셋(배열, 리스트 등)의 내용을 하나씩 봐야 할 경우

      에 주로 사용한다.

05 배열

- 배열

  배열을 쓰면 변수 하나에 값을 여러 개 담을 수 있다.

  - 배열 만드는 법

```
// 1. 선언과 동시에 빈 배열 생성(권장)
int[] intArray = new int[5]; // 크기 5의 빈 배열

// 2. 선언 후, 배열 생성
```

```
int[] intArray;
intArray = new int[5];  // 크기 5의 빈 배열

// 3. 리터럴로 생성
int[] intArray = {1, 2, 3, 4, 5};

// 3번 방식 중 다음 방식은 오류가 발생한다.
// int[] intArray;
// intArray = {1, 2, 3, 4, 5};
```

- 배열 사용
  - 값 대입하는 방법

```
intArray[0] = 1;
intArray[1] = 2;
intArray[2] = 3;
intArray[3] = 4;
intArray[4] = 5;
```

  - 인덱스 범위에서 벗어나면

```
intArray[5] = 6;

// 결과
// Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
```

  - 다음과 같은 에러가 발생하니, 인덱스를 조심히 다루어야 한다.

- 런타임
  - 런타임 : 코드 작성 시점이 아니라 실제 실행될 때를 뜻한다.
  - 위의 오류는 문법적으로 오류는 없지만, 실제 실행을 해서 접근하려 할 때 문제가 생기는 것인데,
  - 예외 처리를 위해 '컴파일 시점', '런타임 시점'에 대해 알아야 한다.

- 변수명 뒤에 인덱스가 들어간다는 것 말고는 배열은 일반적인 변수와 사용법이 같다.

```
// 값을 대입할 때는
intArray[0] = 1;
intArray[1] = 2;

// 값을 읽을 때는
System.out.println(intArray[0] + intArray[1]);  // 1 + 2

// 결과 : 3
```

- 앨리어싱(Aliasing)
  - Alias : 가명

```
int[] arr1 = {1, 2, 3, 4, 5};
int[] arr2 = arr1;

arr1[0] = 100;
System.out.println(arr2[0]);

// 결과 : 100
```

  - arr1을 arr2에 지정해줬을 때, 두 변수는 같은 주소를 가리키게 된다.
    - arr2는 arr1의 가명이라고 할 수 있다.

  - arr1을 arr2로 새롭게 복사하고 싶을 때 **clone** 메소드를 사용하면 된다.

```
int[] arr1 = {1, 2, 3, 4, 5};
int[] arr2 = arr1.clone();

arr1[0] = 100;
System.out.println(arr1[0]);
System.out.println(arr2[0]);

// 결과
// 100
// 1
```

    - 배열은 복사가 된 것이므로 arr1과 arr2는 서로 다른 배열이기 때문에, arr1[0]을 수정해도 arr2[0]에는 영향을 미치지 않고, 1이 출력된다.

- for-each

```
for (int i : intArray) {
    System.out.println(i);
}
```

    - 이렇게 쓰면, 처음에 수행 부분으로 들어갈 때 i는 intArray의 0번 인덱스의 값(원소)을 갖게 되고, 그 다음 들어갈 때는 1번 인덱스의 값(원소)을 갖게 되고... 이런 식으로 배열의 마지막 값(원소)까지 갖게 된다.

- 다중 배열 : 2차원 이상의 구조를 표현할 때 사용한다.

```
int[][] multiArray;
```

    - 초기값 설정하는 방법

```
// 1 2  3  4
// 5 6  7  8
// 9 10 11 12

int[][] multiArray = {
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12}
};
```

    - 위의 예시는 int[4] 배열 세 개가 묶인 배열이라고 볼 수 있다. 즉, multiArray[0]의 자료형은 int[4]이고 내용은 {1, 2, 3, 4}인 것이다.

    - 생성
        - 3 x 4 사이즈의 빈 배열

```
int[][] multiArray = new int[3][4];
```

        각 대괄호 사이에 사이즈를 넣어주면 된다.
        일반적으로 '행(줄)'을 첫 번째 대괄호에, '열(칸)'을 두 번째 대괄호에 넣는다.

    - 사용
        - multiArray[0], multiArray[1], multiArray[2] 모두 int[4]의 자료형을 갖게 된다.

- 그렇기 때문에 multiArray[0]을 일반적인 배열 탐색법으로 탐색할 수 있다.

```
for (int i = 0; i < multiArray.length; i++){
    for (int j = 0; j < multiArray[i].length; j++) {
        multiArray[i][j] = (i * 4 + 1) + j;
    }
}
```