자바 용어 정리

01장

□ **소스**

source [1장 2쪽]

고급 언어로 작성된 내용

□ **컴파일**

compile [1장 2쪽]

컴퓨터가 이해할 수 있도록 0과 1로 이루어진 기계어로 변환하는 과정

□ **객체 지향 프로그래밍**

OOP: Object Oriented Programming [1장 6쪽]

프로그램을 개발하는 기법으로, 부품에 해당하는 객체들을 먼저 만들고 이것들을 하나씩 조립 및 연결해서 전체 프로그램을 완성하는 기법

그것이 알고싶다 **객체 지향 프로그래밍을 할 수 있는 언어는 따로 있다?**

흔히 자바는 객체지향 언어, C언어는 절차지향 언어라고 칭하는데, 그렇다고 해서 자바로 절차 지향 프로그래밍을 할 수 없는 것은 아니다. 단지 자바는 객체 지향 프로그래밍에 적합한 구조로 만들어진 언어일 뿐. 객체 지향 프로그래밍은 프로그래밍의 '패러다임'이다!

- **객체지향 프로그래밍의 특징**
  - 추상화(abstraction)
    실제 세상을 프로그래밍으로 옮길 수 있게, 필요한 기능들만 가져와서 단순화
  - 캡슐화(encapsulation)
    보안적인 기능에 도움을 줌, 규칙 정의, 사용 용도 명확히. 안에 있는 값을 노출하지 않음.
  - 상속성(inheritance)
    부모를 지정하고, 부모의 코드를 가져다가 쓸 수 있는 것. 코드의 재사용성
  - 다형성(polymorphism)
    변수가 여러 가지 모습으로 변할 수 있는 것.
- **클래스**
  - 사물 하나 하나를 기능별로 묶어서 사용하는 것. 각 클래스 안에서 역할에 따라 각 클래스의 기능을 서술해 나감.
  - 사람에 따라, 같은 프로그램을 만들더라도, 여러가지 관점으로 만들 수 있다.
  - 자바에서는 어떤 프로그래밍이든 클래스 안에 속해있다.(public static void main(String[] args) 기능도 class안에 속해 있다)
  - 다른 클래스에서 다른 클래스를 선언 하여 사용할 수 있다.

```
public class 클래스이름 {
        public static void main(String[] args){
    //프로그램 시작 시점
    }
}
```

클래스의 구조

- 멤버변수
  클래스 안의 기능을 끄집어 내서 사용할 때 사용.
  저장할 공간.
  (ex. 화면에 출력 시 사용했던 System.out.println()에서 out은 System이라는 클래스의 out이라는 멤버변수를 사용한 것이다.)

- 메서드(method)
  기능을 나타낸다.
  ()가 항상 붙어있다.
  (ex. System.out.println에서 println()이다.)

- 생성자(constructor)
  처음에 값을 넣어줄 때 사용(default 값)

□ 자바 가상 기계

JVM: Java Virtual Machine [1장 24쪽]

자바 프로그램은 완전한 기계어가 아닌 중간 단계의 바이트 코드이기 때문에, 이것을 해석하고 실행할 수 있게 해주는 가상의 운영체제

* JVM의 메모리 영역

- 메소드 영역: 클래스(~.class)들을 클래스 로더로 읽어 클래스별로 런타임 상수 풀, 필드 데이터, 메소드 데이터, 메소드 코드, 생성자 코드 등을 분류해서 저장한다.
- 힙 영역: 객체와 배열이 생성되는 영역. JVM 스택 영역에서 생성된 변수나 다른 객체의 필드에서 이 곳에 있는 객체와 배열을 참조한다.
- JVM 스택 영역: 스레드마다 하나씩 존재. 스레드가 시작할 때마다 JVM 스택 영역이 할당된다.

□ 예약어

reserved word [2장 49쪽]

이미 해당 프로그래밍 언어에서 사용하기 위해 의미가 정해져 있는 것.

그것이 알고싶다 **예약어를 식별자(변수명/메소드명/함수명)로 사용하면 안되는 걸까?**

프로그래밍 언어 내에서 이미 문법적인 용도로 사용되고 있기 때문에 사용하면 안 된다. 자바에서 예약어를 식별자로 사용할 경우 컴파일러에서 에러 처리하여 실행되지 않는다.

□ 초기값

initial value [2장 50쪽]

변수를 선언하고 처음 저장하는 값

□ 초기화

initialize [2장 50쪽]

변수에 초기값을 주는 행위

□ 리터럴

literal 참고 용어 상수 [2장 60쪽]

소스 코드 내에서 직접 입력된 값

그것이 알고싶다 상수와 리터럴

사실 리터럴은 상수와 같은 의미지만, 프로그램에서는 상수를 "값을 한 번 저장하면 변경할 수 없는 변수"로 정의하기 때문에 이와 구분하기 위해 "리터럴"이라는 용어를 사용한다.

□ 이스케이프 문자

escape character [2장 46쪽]

역슬래시(\) 기호가 붙은 특수한 문자 리터럴

이스케이프 문자	출력
\t	수평 탭
\n ——→ 이 때 n은 new line의 약자!	줄 바꿈
\r	캐리지 리턴
\"	" (큰 따옴표)
\'	' (작은 따옴표)
\\	\
\u16진수	16진수에 해당하는 유니코드

□ 유니코드

unicode [2장 63쪽]

세계 각국의 문자들을 코드값으로 매핑한 국제 표준 규약이다. 유니코드는 하나의 문자에 대해 하나의 코드값을 부여하기 때문에 영문 'A' 및 한글 '가'도 하나의 코드값을 가진다. 자바는 모든 문자를 유니코드로 처리한다.

```
char c1 = 'A';
char c2 = '\u0041';
```

그것이 알고싶다 아스키(ASCII) 코드와 유니코드

유니코드는 영문자 외에 전 세계의 문자를 표현할 수 있도록 설계된 반면 아스키(ASCII: American Standard Code for Information Interchange, 미국 정보 교환 표준 부호) 코드는 7비트로 표현되는 영문자 기반 인코딩이다. 유니코드의 앞부분은 아스키 문자로 할당되어 있다.

□ 자료형

data type [2장 49쪽]

자료의 형태. 자료형에 따라 컴퓨터가 어떻게 처리하는지 달라진다. 자바에서는 기본 타입과 참조 타입으로 구분됨

□ 기본 타입

primitive type [2장 59쪽]

원시 타입이라고도 한다. 정수, 실수, 문자, 논리 리터럴을 저장하는 자료형

□ 참조 타입

reference type [2장 121쪽]

객체의 번지를 참조하는 타입. 배열, 열거, 클래스, 인터페이스 타입이 있다.

그것이 알고싶다 **값에 의한 호출**call by value**과 참조에 의한 호출**call by reference

값에 의한 호출을 할 땐 메소드가 전달인자를 복사하여 사용하고, 메소드 실행이 종료되면 반환하는 방식이기 때문에 전달인자를 직접 수정하는 것이 의미가 없다. 참조란 대상의 주소값을 통해 접근하는 것이고, 참조에 의한 호출은 메소드가 전달인자로 주소값을 넘겨받아 해당 위치에 있는 값에 접근하여 다른 곳에서 쓰일 수 있도록 수정할 수 있다. 자바에서는 무조건 call by value인 것이 아닌가 싶지만, 전달인자로 받은 데이터의 타입에 따라 그 값value이 주소값이 되기도 하고, 객체가 가리키는 값이 되기도 한다.

03장

□ **연산자** — operator

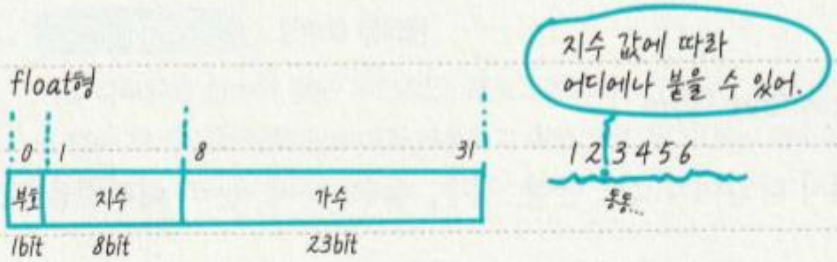
연산에 사용되는 표시나 기호

□ **피연산자** — operand [3장 102쪽]

연산식에서 연산되는 데이터

□ **부동 소수점** — floating point [3장 120쪽]

소수점이 있는 실수 데이터를 저장하는 방식. '부동'은 '떠다니다'의 의미, 소수점이 떠다닌다는 의미에서 부동 소수점이다. 이 방식에서는 최상위 비트MSB: Most Significant Bit를 부호로 결정한다. 최상위 비트가 0이면 양수, 1이면 음수를 뜻한다.



04장

□ 제어문

control statement [4장 134쪽]

실행 흐름을 개발자가 원하는 방향으로 바꿀 수 있도록 해주는 것.

- 조건문: if문, switch 문
- 반복문: for문, while문, do-while문

□ 조건문

conditional statement [4장 134쪽]

특정 조건을 만족할 때 코드를 실행하는 문법

- if문: 괄호 내의 조건식이 참이면 블록 내의 문장을 실행한다.
- else문: if문의 조건식이 거짓이면 블록 내의 문장을 실행한다. 필요 없으면 없어도 된다.
- else if문: if문의 조건식이 거짓일 때 실행시킬 코드에 추가 조건을 걸고 싶을 때 사용한다. 마찬가지로 필요 없으면 else if를 사용하지 않아도 된다.

```
if (a > 0)
  //중괄호를 사용하지 않으면 조건식이 참일 때 한 줄만 실행
  System.out.println("a는 양수입니다.");
else if (a == 0) {
  System.out.println("a는 0입니다.");
} else {
  System.out.println("a는 음수입니다.");
}

switch(ch) {
case 'a':
  //이 케이스에서 break 문이 없기 때문에 블록을 빠져나가지 않고 다음
  case 문으로 넘어간다
case 'b':
  System.out.println("ch는 a 혹은 b입니다"); //a일 경우에도 출력된다.
  break;
case 'c':
  System.out.println("ch는 c입니다");
  break;
default:
  System.out.println("a도 b도 c도 아닙니다");
  break; //이 break문을 적지 않아도 이 블록을 빠져나갈 수 있다.
}
```

□ 반복문

loop

[4장 148쪽]

특정 조건을 만족하는 동안 반복해서 실행하는 문법

- for문: 반복 횟수가 정해진 경우에 주로 사용

```
//for문 예시
for(int i = 0; i < n; i++)
  //0부터 n-1까지 i를 증가시키며 코드 실행 (n회 반복)
```

- while문: 반복 횟수를 모를 때 주로 사용

```
//while문 예시
while(true) {
  result *= 3;
  if(result >= 100) //이 조건을 만족시켜야 반복문 종료
        break;
}
```

• do-while문: 조건 만족 여부와 상관없이 코드를 먼저 실행하고, 그 다음 루프부터 조건을 검사

```
//do-while문 예시
do {
  printf("어머 이건 꼭 출력 되어야 해...! ");
} while(false); //1회만 출력
```

□ **무한 루프** infinite loop [4장 155쪽]

무한히 반복을 하고 싶거나 정해진 횟수 없이 일정한 조건을 충족하면 빠져나오게 하고 싶을 때 사용하는 것.

```
//for문을 사용한 무한 루프
for(;;) {
  //반복하고 싶은 코드
}

//while문을 사용한 무한 루프
while(true){
  //반복하고 싶은 코드
}
```

05장

□ 쓰레기 수집기

garbage collector 참고 용어 오버라이딩, 인스턴스 [5장 167쪽]

쓰레기 수집은 메모리 관리 기법 중 하나이다. 자바에선 JVM의 쓰레기 수집기를 이용해 자동적으로 사용하지 않는 객체를 메모리에서 제거한다.

그것이 알고싶다 **객체를 제거하기 전 수행하고 싶은 일이 있을 때**

쓰레기 수집기가 객체를 제거하기 전에 중요한 데이터를 저장하고 싶을 때, 클래스 내에 객체 소멸자(finalize())를 재정의하여 객체가 소멸될 때 실행할 코드를 입력할 수 있다.

□ null

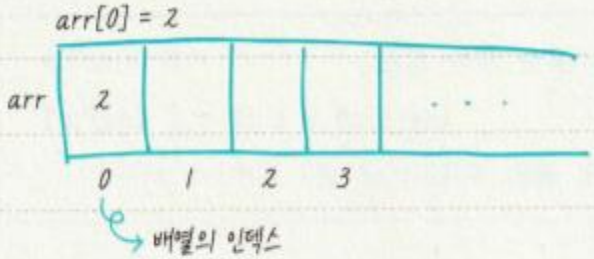
참조 타입 변수가 객체를 참조하지 않는다는 의미의 값. '널'이라고 읽는다. null값도 초기값으로 사용할 수 있기 때문에 null로 초기화된 참조 변수는 스택 영역에 저장된다. [5장 169쪽]

□ 배열

array [5장 170쪽]

데이터를 연속된 공간에 나열하고, 각 데이터에 인덱스를 부여해 놓은 자료구조

→ 이러한 성질 때문에 중간 인덱스의 값을 삽입하거나 삭제하기 어렵고, 선언 후 배열의 인덱스 수를 늘릴 수 없다.



□ 인덱스

index [5장 178쪽]

0에서부터 "문자열(혹은 배열) 길이−1"까지의 번호를 매긴 것. 배열과 같은 선형 자료구조는 인덱스로 원소에 접근할 수 있다.

□ 열거 타입

enumeration type [5장 203쪽]

한정된 값만을 갖는 자료형

```
enum Week {
  SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}
```

↓ 열거 상수

이클립스 단축키

1. 펑션키

F1 : 도움말

F3 : 클래스, 메소드, 속성이 선언된 위치로 이동

F4 : 클래스의 계층구조 확인(Type Hierarchy view)

F5 : 디버깅 시 선택된 행의 메소드 내부로 이동 (Step In)

F6 : 디버깅 시 선택된 행의 아래로 이동 (Step Over)

F7 : 디버깅 시 실행중인 메소드 외부로 이동(Step Return)

F8 : 디버깅 시 다음 디버그 포인트(중단점)로 이동 (Resume)

F11 : 디버그 모드로 실행 (Debug)

F12 : Editor창으로 이동

이클립스 단축키

2. Ctrl + 단축키

Ctrl + / : 한 줄 주석(//) 처리 또는 해제

Ctrl + 콤마(,) 또는 점(.) : 다음/이전 에러(경고)로 이동

Ctrl + F6 또는 E : Editor 창 간의 이동 (파일간의 이동)

Ctrl + F7 : View 창 간의 이동 (Console, Problems 등)

Ctrl + F8 : Perspectives 창 간의 이동 (Java, Debug 등)

Ctrl + F11 : 실행 모드로 실행 (Run)

Ctrl + 1 : 빠른 수정 (에러에 대한 수정할 코드 추천)

Ctrl + D : 한 줄 삭제

Ctrl + F : 문자열 찾기 (Find/Replace 다이얼로그)

Ctrl + H : 문자열 찾기 (Search 다이얼로그)

Ctrl + I : 들여쓰기 자동 적용

Ctrl + K : 현재 선택된 문자열과 동일한 문자열 찾기

Ctrl + L : 행 번호를 입력하여 특정 행으로 이동 (Go to Line 다이얼로그)

Ctrl + M : 현재 View / Editor 를 최대화

Ctrl + N : 새로운 파일 / 프로젝트 생성

Ctrl + O : 메소드 또는 속성 이동

Ctrl + Q : 마지막으로 편집한 곳으로 이동

Ctrl + T : 클래스 계층 구조 팝업

Ctrl + W : 파일 닫기

Ctrl + Space : 코드 자동완성

이클립스 단축키

3. Alt + 단축키

Alt + 화살표(up, down) : 현재 라인을 한 줄씩 위(아래)로 이동

4. Ctrl + Shift + 단축키

Ctrl + Shift + F4 : 열린 파일 모두 닫음

Ctrl + Shift + T : 클래스 명 찾는 창 띄우기

Ctrl + Shift + R :파일(클래스 포함) 찾기

Ctrl + Shift + Space : 메소드의 파라미터 목록 표시 (Ctrl + Space와는 다르게 메소드만 표시)

Ctrl + Shift + O : import 자동 추가 / 삭제

Ctrl + Shift + B : 현재 행의 중단점(Break Point) 설정 / 해제

이클립스 단축키

Ctrl + Shift + F : 코드 형식 정리

Ctrl + Shift + G : 선택한 메소드, 속성이 사용된 모든 곳을 검색 (Search view)

Ctrl + Shift + L : 모든 단축키 정보 표시

Ctrl + Shift + X : 대문자 변환

Ctrl + Shift + Y : 소문자 변환

Ctrl + Shift + K : 현재 선택된 문자열과 동일한 문자열 반대로 찾기 (Ctrl + K 와 반대)

5. Alt + Shift + 단축키

Alt + Shift + R : 클래스명, 메소드명, 필드명 위에서 누르면, 이름을 바꿀수 있다. 참조하는 곳도 모두 바꿔준다.

Alt + Shift + L : 선택한 코드를 지역변수로 선언해서 참조해준다.

Alt + Shift + M : 선택한 코드를 메소드로 생성해준다.

이클립스 단축키

6. Ctrl + Alt + 단축키

Ctrl + Alt + G : 전체 파일에서 선택된 문자열과 동일한 문자열 찾기 (Search view)

Ctrl + Alt + 화살표(up, down) : 현재 라인을 위(아래)로 복사

출처 : http://ralf79.tistory.com/24

출처 : http://library1008.tistory.com/35