Chap 6. 객체지향 프로그래밍 I

◈ 객체지향이란?
- 프로그램을 묶음 단위로 잘게 쪼개서 가져다 쓰기 편하게 만들어 놓은 것들

◈ 특징
- 캡슐화 : 캡슐로 감싸서 내부 변수들을 보호하는 것
- 상속 : 코드를 물려받아서 재사용함으로써 중복 방지
- 추상화 : 미리 틀을 만들어서, 누락 부분이 없게 해주는 것 (구조화)
- 다형성 : 하나의 틀을 여러가지 기능과 용도로 써먹게 하는 것

→ 코드 재사용, 유지/보수, 중복된 코드 및 오류 제거 목적

◈ 클래스와 객체, 인스턴스
- 클래스 : 변수와 메서드를 정의해 놓는 공간, 객체를 생성하는 용도로 사용
- 객체 : 클래스 내에 정의된 변수와 메서드를 전부 지칭
  . 클래스에 의해 정의되며, 메모리에 생성됨, 멤버(속성과 기능)의 개별 집합체
- 인스턴스 : One of 객체, 특정 클래스에서 생성된 객체를 지칭
  eg) Car 인스턴스 : Car 클래스 내에서 생성된 변수와 메서드

◈ 인스턴스화(instantiate)
- 특정 클래스 내에서 실제로 객체를 만드는 행위, 과정

◈ 변수의 초기화 : 멤버변수(iv, cv)는 자동 초기화, 지역변수(lv)는 수동 초기화
- boolean : false - char : \n0000 - long : 0L - 참조변수 : Null
- byte, short, int : 0 - float : 0, 0f - double : 0.0d, 0.0

◈ 멤버의 종류 (멤버는 변수, 메서드로 구성)
- 클래스 멤버 (= 스태틱 멤버) : 앞에 static이 붙는 변수, 메서드
  . 가장 먼저 생성되고 메모리 uploaded
  . 인스턴스 멤버를 호출할 수 없다. b/c 인스턴스 멤버 not uploaded
- 인스턴스 멤버 : static이 안붙는 변수, 메서드

◈ 변수의 종류 (타입에 의한 구분)
- 기본형 변수 (primitive type, data type)
  . boolean, char, byte, short, int, long, float, double 총 8개
- 참조형 변수 (reference type)
  . 특정 클래스에서 생성된 인스턴스들의 주소를 저장하는 변수
  . 메서드의 매개변수(parameter)로 참조변수를 넣으면, 메서드에서 직접 인스턴스들에 접근이 가능하므로, 여러개의 리턴값을 갖는 효과를 가진다.

◈ 변수의 종류 (위치에 의한 구분)
- 멤버변수 : 클래스 안에 선언되어 있는 변수들(cv, iv)
  . 클래스 변수 (class variable) = 스태틱 변수(static variable)
    → 공통적으로 사용하는 변수, 클래스가 선언되면 메모리 로딩됨
    → 모든 인스턴스가 공유하는 변수, public을 붙이면 전역변수가 됨
    → 자동 초기화 됨.
  . 인스턴스 변수(instance variable)
    → 해당 클래스의 객체를 생성할 때 만들어지고 메모리 로딩됨
    → 별도로 초기화 필요함 from 초기화 블럭 or 생성자
- 지역변수 : 클래스 안에서도 특정 메서드 안에 있는 변수(lv), 매개변수도 포함
  . 특정 메서드가 종료되면 메모리에서 제거됨 by Garbage collector

※ 클래스 변수를 사용할 때는 명시적으로 “클래스이름.클래스 변수”로 사용

◈ 메서드의 종류
- 클래스 메서드 (= Static method) : 인스턴스 생성없이 호출 가능
- 인스턴스 메서드 : 인스턴스 생성(예. *Car c = new Car()*)을 해야 호출 가능

◈ 인스턴스의 접근
- 특정 클래스 내의 인스턴스는 참조변수를 통해서만 접근 가능
- 이 참조변수의 타입은 인스턴스 타입과 일치해야 함

◈ 데이터 저장 개념의 발전
- 변수(단일) → 배열(집합) → 구조체(여러 타입) → 클래스(+ 메서드)

◈ Naming Rule

- 클래스, 구조체 : 파스칼 표기법	class PlayerManager
- 지역변수, 매개변수 : 카멜 표기법	int dailyUserTable
- 메서드 : 동사(명령형) + 명사(목적어)	GetAge()
- Boolean 메서드 : 조동사/Be동사 + 동사	IsAlive, CanAccept
- 상수 : 전부 대문자로 하되 밑줄로 분리	SOME_CONSTANT
- Boolean 변수 : 앞에 b를 붙임	bFired
- 인터페이스 : 앞에 I를 붙임	ISomeInterface
- 열거형 : 앞에 E를 붙임	EDirection
- 구조체 : 앞에 S를 붙임	SUserID
- Private 멤버 변수 : 앞에 m, 파스칼	private int mAge
- 그 외 모든 메서드 : 전부 파스칼	

◈ Return 0, 1, -1

0 : 정상종료 1 : 정상종료 but 뭔가 있음 -1 : 에러발생

◈ 오버로딩 (overloading)
- 한 클래스 내에 같은 이름의 메서드를 여러개 정의하는 것
- 매서드 이름이 같고, 매개변수의 개수 또는 타입이 달라야 한다.
- 리턴 타입으로 오버로딩은 불가

◈ 가변인자 (varargs)
- 매개변수의 개수를 동적으로 할당, 가장 마지막에 선언해야 함
- 배열을 노출하기 때문에 type 안정성이 떨어진다.
- 가변인자를 사용한 메서드는 오버로딩하지 말것

eg) printf(String format, Object… args)

◈ 생성자 (Constructor) : 클래스의 이름과 같아야 하며, return 값이 없다.
- 기본 생성자 (default constructor) : 생성자가 하나도 없을 때, 컴파일러가 제공
- 매개변수 생성자 (parameter) : 기본 생성자에서 오버로딩한 개념

◈ this와 this()
- this : 메서드/생성자에 입력된 매개변수와 지역변수를 구분하기 위함
  . this는 참조변수이며, 클래스의 인스턴스 자신을 가리킨다.
- this() : 한 생성자에서 같은 클래스내 다른 생성자를 호출할 때 사용
  . 첫 줄에서만 호출 가능,
  . Static 메서드(Class 메서드)에서는 this 사용 불가 b/c 인스턴스 자신의 주소를 저장하고 있으며, 인스턴스 메서드에 숨겨진 채로 존재하기 때문.

eg)
```
Car() {
   this("white", "auto", 4);
}
Car(String color, String gearType, int door) {
   this.color = color;    this.gearType = gearType;    this.door = door;
}
```

◈ 생성자를 이용한 인스턴스 복사

- 생성자의 매개변수로 해당 클래스의 참조변수를 넣으면 인스턴스 복사 가능

eg)
```
Car(Car c) {
        color = c.color;
        gearType = c.gearType;
        door = c.door; }
```

◈ 멤버 변수의 초기화 방법

- 명시적 초기화(explicit initialization) : 변수 선언과 동시에 초기화하는 것
- 생성자(constructor)
- 초기화 블럭(initialization block)
  . 인스턴스 초기화 블럭 : 인스턴스 변수 초기화, 인스턴스 생성시마다 수행
  . 클래스 초기화 블럭 : 클래스 변수(static)를 초기화, 클래스 로딩시 한번 수행
- 기본값 → 명시적 초기화 → 클래스 초기화 블럭 → 인스턴스 초기화 블럭 → 인스턴스 생성자

※ 생성자마다 공통 수행 코드가 있을때 인스턴스 초기화 블럭에 넣어준다.

<연습문제>

- 간단한 조건문은 삼항연산자(ternary operator)를 사용해서 한줄로 표현할 것
  (condition) ? true : false; eg) return num + (isKwang? "K" : "");

- 기본 생성자에서 오버로딩된 생성자를 호출하여 초기화 하는 코드 구현 필요
  this(1, true);

- 반올림 계산법 2가지 (소수점 2째 자리에서) + 리터럴 뒤에 f 주의
  1) Math.round(this.average*10)/10f
  2) (int)(getTotal() / 3f * 10 + 0.5f) / 10f;

- 간단한 계산에서 메서드를 호출하면 비용이 많이든다.

- static 메서드(클래스 메서드)는 인스턴스 변수를 사용하지 않으므로, 변수 사용을 위해서는 매개변수(지역변수)로 받아야 한다. 이를 거꾸로 뒤집으면, 인스턴스와 관계가 없는 메서드는 static으로 선언할 수 있다는 얘기가 된다.
- 즉, 인스턴스 변수를 사용하면 인스턴스 메서드로 하고, 그렇지 않으면 static 메서드로 하면 된다.

```
static double getDistance(int x, int y, int x1, int y1) {
        return Math.sqrt((x-x1)*(x-x1) + (y-y1)*(y-y1)); } // x, y는 지역변수

double getDistance(int x, int y) {
        return Math.sqrt((x-x1)*(x-x1) + (y-y1)*(y-y1)); } // x, y는 인스턴스변수
```

- 지역변수는 메서드가 종료되면 자동 소멸된다
- 힙(heap) 영역 : 인스턴스가 생성되는 영역
- 호출스택(call stack, execution stack) : 지역변수가 생성되는 영역
- 메서드 영역(method area) : 클래스 변수, 클래스 데이터 저장

- static 메서드 관련

```
int iv = 10;
static int cv2 = iv;  // 에러 : static 변수의 초기화에 iv 사용 불가
                         인스턴스 생성하면 사용 가능

static void staticMethod1()  {
  System.out.println(iv); } // 에러 : static 메서드에서 iv 사용 불가

static void staticMethod1()  {
  instanceMethod1(); }  // 에러 : static 메서드에서 인스턴스메서드 사용 불가
```

- 참조형 매개변수 주의점

```
class Exercise6_19 {
 public static void change(String str) {
   str += "456";  }

 public static void main(String[] args) {
  String str = "ABC123";
  System.out.println(str);
  change(str);
  System.out.println("After change:" + str);  }
```

결과 : ABC123 &n After change : ABC123

- 배열, String의 길이를 세는 방법 : arr.length => 배열 / arr.length() => String

- 배열의 셔플 로직 : 배열 크기의 2배 횟수로 섞는다. 너무 많이 섞으면 성능저하

```
for(int x=0; x<arr.length*2; x++) {
        int i = (int)(Math.random()*arr.length);
        int j = (int)(Math.random()*arr.length);
        int tmp = arr[i];
        arr[i] = arr[j];
        arr[j] = tmp;  }
```

- 배열에서 최대값 찾는 로직

```
int max = arr[0];
for(int i=1; i<arr.length; i++) {
        if(arr[i] > max)
        max = arr[i]; }
return max;
```

- 절대값 반환하는 로직 : return value >=0? value : -value;