Java – 객체지향 – 1

컴소과 동아리 REFERENCE

제작자: 김명준

목 차

클래스와 객체

클래스를 작성할 때 유의할 점

public class가 있을 경우,

소스 파일의 이름은 public class와 **이름이 일치해야 함**

public class가 없을 경우,

소스 파일의 이름은 아무거나 상관 없음

즉, **하나의 소스 파일에 둘 이상의 public class가 존재할 수 없음**

클래스와 객체

클래스: 객체를 생성하는 **설계도**

사용자정의 타입

객체: 인간이 분명하게 인지하고 구별할 수 있는

물리적인 또는 개념적인 경계를 지닌 어떤 것
즉, **상태, 행동, 식별자를 지닌 실체**

클래스와 객체

객체의 구성 요소

- 속성(멤버변수): 특성, 상태
- 기능(메서드): 행위

인스턴스: 객체를 구체화한 것

클래스와 객체

인스턴스 생성

```
클래스명 참조변수명; // 객체를 다루는 참조변수 선언
참조변수명 = new 클래스명(); // 객체 생성
```

```
클래스명 참조변수명 = new 클래스명();
```

선언과 생성을 한 줄에 가능

new 연산자는 주소를 생성

클래스와 객체

객체 배열

```
클래스명[] 참조변수명 = new 클래스명[배열의 길이]; // 객체 배열을 다루기 위한 참조변수 선언
참조변수명[인덱스] = new 클래스명(); // 객체 생성
```

각 요소마다 new 연산자를 사용

해 인스턴스를 생성해야 함

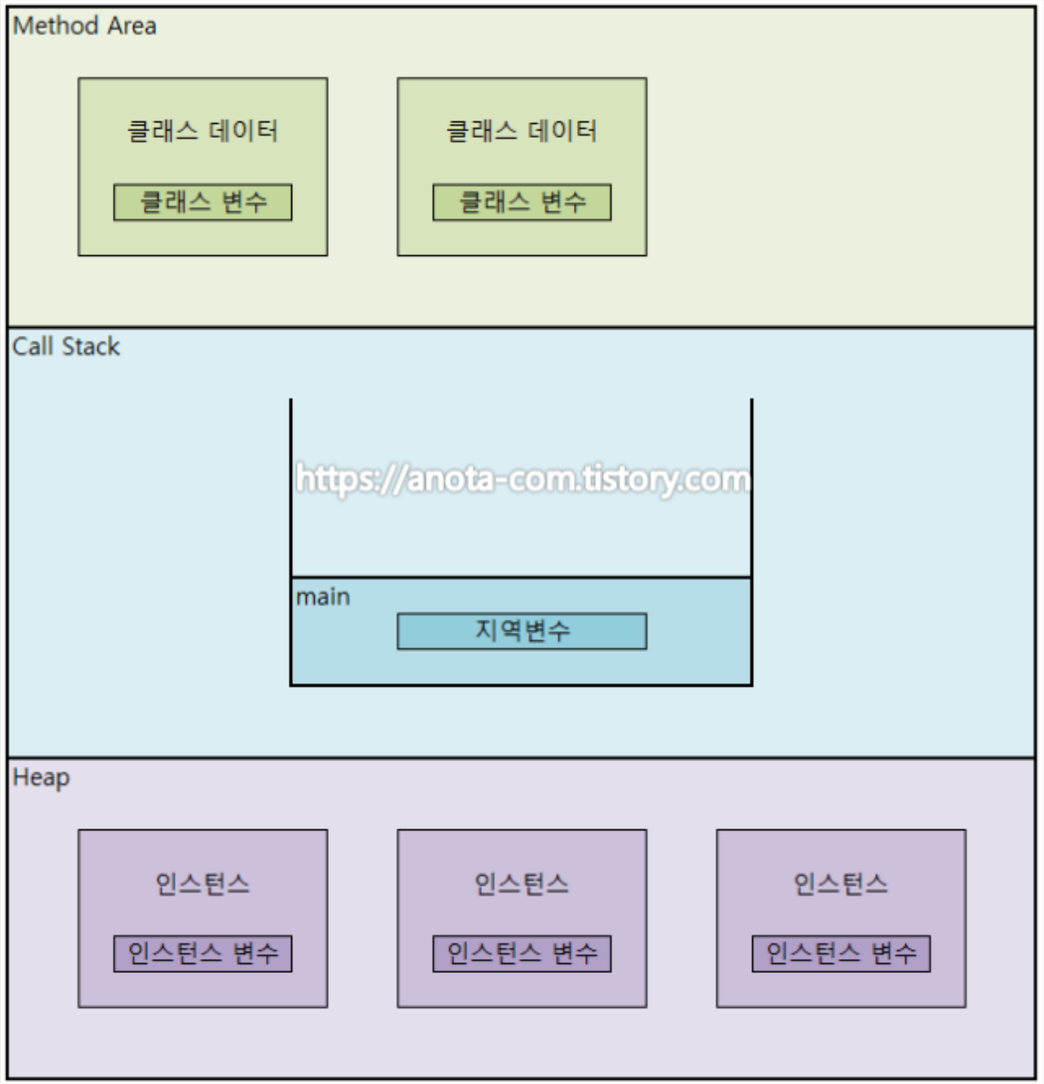
→ 반복문이나 초기화 블록 사용

변수와 메소드

변수의 선언 위치에 따라 종류와 범위가 결정

	변수의 종류	선언 위치	생성 시기
Method 영역	클래스 변수	클래스 영역	클래스가 메모리에 올라갈 때
Heap 영역	인스턴스 변수		인스턴스 생성되었을 때
Stack 영역	지역 변수	클래스 영역 이외의 영역	변수 선언문 수행되었을 때

변수와 메소드



Method Area:

클래스 정보와 클래스변수가 저장되는 곳

Call Stack:

메소드가 호출되면 메모리 공간을 할당받고

종료되면 메모리를 반환

Heap Area:

인스턴스가 생성되는 공간, new 연산자를

통해 생성되는 모든 것은 여기에 생성

변수와 메소드

인스턴스변수: 각 인스턴스의 **개별적인** 저장공간

특징

- 인스턴스 생성 후에 사용 가능(new 연산자)
- “참조변수.인스턴스변수”로 접근

→ 인스턴스마다 **서로 다른 값을 가져야 한다면**,

인스턴스변수로 선언

변수와 메소드

클래스변수: 같은 타입의 **모든 인스턴스들이 공유**하는 변수
static 키워드 + 인스턴스변수

특징

- 인스턴스 생성 없이 사용 가능
- “클래스.클래스변수”로 접근

→ 인스턴스가 **모두 같은 값을 가져야 한다면**,

클래스변수로 선언

변수와 메소드

지역변수: 클래스 영역 이외에서 선언된 변수

특징

- 메소드 내에 선언되며, 메소드 종료와 함께 소멸
- {}이 유효 스코프
- **반드시 초기화를 해줘야 함**

변수와 메소드

메소드: 특정 작업을 수행하는 문장을 하나로 묶은 것

함수와 동일하나, 클래스 영역 안에서만 정의 가능

메소드의 장점

- 코드의 중복 제거
- 코드의 관리가 용이
- 코드의 재사용성 증가
- 코드가 간결해져 이해하기 쉬움

하나의 메소드는 한 가지 기능만 수행하도록 작성

변수와 메소드

메소드의 정의

```
반환타입 메소드이름(매개변수, ...) // 선언부
{
    수행할 문장들 // 구현부
}
```

반환타입이 없을 경우에는 **void**를 사용

반환타입이 void가 아닐 경우에는 구현부에 무조건 return문이 있어야 함

반환값이 2개 이상이라면 배열이나 객체를 통해 반환

반환타입은 기본형이 아닌 참조형도 가능

변수와 메소드

메소드 호출: “참조변수.메소드”

메소드 호출의 특징

- 메소드는 호출되면 메모리를 스택에 할당받음
- **스택 맨 위에 있는 메소드가 현재 실행 중인 메소드**
- **아래에 있는 메소드가 바로 위의 메소드를 호출한 메소드**

변수와 메소드

매개변수의 종류: 기본형 매개변수, 참조형 매개변수
매개변수: 지역변수 → 메소드가 종료되면 소멸

기본형 매개변수: Read Only → 값을 넘김

즉, **값을 복사하여 넘기므로 기존 값에 영향 X**

참조형 매개변수: Read & Write → 주소를 넘김

주소를 복사하여 넘기므로 주소에 저장된 값에 영향 O

매개변수는 사용자가 입력하는 것이므로 **유효성 검사**를 해야 함

인스턴스변수 사용 여부에 따라 결정

인스턴스메소드: 메소드 내에서 인스턴스변수 사용 가능

- 인스턴스 생성 후 “참조변수.메소드”로 호출
- 클래스변수도 사용 가능

클래스메소드: 메소드 내에서 인스턴스멤버 사용 불가

- 인스턴스 생성 없이 “클래스.메소드”로 호출
- static + 인스턴스클래스

→ **클래스메소드를 사용할 때 인스턴스가 생성되었는지 알 수 없음**

오버로딩

메소드 오버로딩

오버로딩: 하나의 클래스 내에서

같은 이름의 메소드를 여러 개 정의하는 것

매개변수는 다르지만 같은 기능을 수행해야 함

오버로딩의 조건

- 메소드의 이름이 같아야 함
- 매개변수의 개수와 타입이 달라야 함

반환타입과 매개변수의 이름은 상관없음

메소드 오버로딩

오버로딩의 장점

- 메소드의 이름을 기억하기 쉽고 기능을 예측할 수 있음
- 메소드의 이름을 절약할 수 있음

가변인자를 사용한 메소드는 가능하면 오버로딩 X → 구별하기 힘듦

가변인자: 매개변수의 개수를 동적으로 지정하는 것

- 타입... 변수명으로 선언
- **매개변수 중에서 제일 마지막에 선언해야 함**

생성자

생 성 자

생성자: **인스턴스 초기화 메소드**
오버로딩이 가능, 매개변수 존재

생성자의 조건

- **생성자의 이름은 클래스의 이름과 동일해야 함**
- **생성자는 반환 값이 없음**

모든 클래스에는 반드시 하나 이상의 생성자가 있어야 함

생성자

기본 생성자: **클래스에 생성자가 하나도 없을 때만**,

컴파일러가 기본으로 추가해주는 생성자

```
class 클래스 {
    // 클래스() {} // 기본 생성자
}
```

생성자

this(): **같은 클래스의 다른 생성자**를 호출할 때 사용

생성자의 첫 문장에서만 가능

this: **인스턴스 자신을 가리키는 참조변수**

클래스메소드에서 사용 불가

인스턴스변수와 지역변수를 구별하기 위해 사용

생성자

인스턴스 복사 방법

- 생성자 이용
- Object의 clone() 이용

생성자 이용: 생성자를 매개변수로 받아서 복사

```
클래스(클래스 참조변수) {
    인스턴스변수 = 참조변수.인스턴스변수
}
```

변수의 초기화

변수의 초기화

멤버변수는 초기화 생략 가능

지역변수는 사용 전에 무조건 초기화를 해야 함

멤버변수들은 대부분 0으로 초기화됨

논리형은 false, 참조형은 null

변수의 초기화

멤버변수의 초기화 방법

- 명시적 초기화: 대입 연산자 =
- 생성자
- 초기화 블럭
  - 인스턴스 초기화 블록: {}
  - 클래스 초기화 블록: static {}

변수의 초기화

클래스변수 초기화 시점: **클래스가 처음 로딩될 때 단 한 번**

인스턴스변수 초기화 시점: **인스턴스가 생성될 때 마다**

초기화 순서: **자동 → 간단 → 복잡**

클래스 초기화			인스턴스 초기화			
기본값	명시적 초기화	클래스 초기화블럭	기본값	명시적 초기화	인스턴스 초기화블럭	생성자
cv 0	cv 1	cv 2	cv 2 iv 0	cv 2 iv 1	cv 2 iv 2	cv 2 iv 3
1	2	3	4	5	6	7

감사합니다

컴소과 동아리 REFERENCE