



06장(객체지향언어 1)

객체지향이론의 기본 개념

- 실제 세계는 사물(객체)로 이루어져 있으며, 발생하는 모든 사건들은 사물간의 상호작용이다.

객체지향언어의 주요 특징

- 코드의 재사용성이 높음
- 코드의 관리가 용이
- 신뢰성이 높은 프로그래밍을 가능하게 함
 - 제어자와 메서드를 이용해 데이터 보호하고 올바른 값을 유도
 - 코드의 중복을 제거하여 코드의 불일치로 인한 오작동 방지

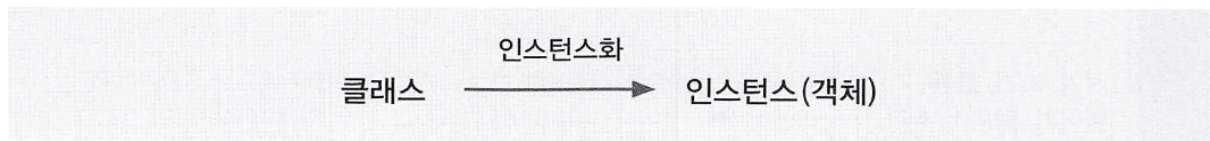
01. 클래스와 객체

!!처음부터 이론을 많이 안다고해서 좋은 설계를 할 수 있는 것은 아님!!

클래스

- 정의 → 객체를 정의해 놓은 것
 - 객체 정의 : 실제로 존재하는 것. 사물 or 개념
- 용도 → 객체를 생성하는데 사용
 - 객체 용도 : 객체가 가지고 있는 기능과 속성에 따라 다름

- 객체를 사용하기 위해선 클래스로 부터 객체를 생성하는 과정이 선행되어야 함



객체의 구성요소

- 속성(property)
 - 멤버변수(member variable), 특성(attribute), 필드(field), 상태(state)
- 기능(function)
 - 메서드(method), 함수(function), 행위(behavior)

인스턴스(객체) 생성하는 방법

- 메모리에 참조변수 ok를 위한 공간이 마련됨
- **new**에 의해 Class의 인스턴스가 메모리의 빈공간에 생성됨
- 대입연산자(=)에 의해 생성된 객체의 주소값이 참조변수 ok에 저장됨

```

Class ok = new Class();

Class ok = null;
ok = new Class();
  
```

인스턴스(객체)

- Class의 인스턴스를 다루기 위해서는 참조변수(ok)가 반드시 필요함!!
 - 참조변수의 타입은 인스턴스의 타입과 일치해야한다.
- 참조변수의 주소값이 변경되어 자신을 참조하고 있는 참조변수가 하나도 없는 인스턴스는
 - GC(Garbage Collector)**에 의해 자동적으로 메모리에서 제거됨
- 객체도 배열로 다루는 것이 가능
 - **각 배열에 객체의 주소가 저장되는 것**

02. 변수

지역변수

- 중괄호 내에서 선언된 변수
- 지역 변수를 선언한 중괄호 내에서만 유효
- 메소드 내부에서만 사용
- 스택메모리 영역

매개변수

- 메소드, 생성자에 넘겨주는 변수
- 메소드가 호출될 때 시작, 메소드가 끝나면 소멸

인스턴스 변수(멤버 변수)

- 메소드 밖, 클래스 안에 선언된 변수, 앞에 **static 예약어가 없음**
- 객체가 생성될 때 시작, 그 객체를 참조하는 다른 객체가 없으면 소멸
- 주로 **private로 선언**, 아니라면 참조변수를 다른 클래스에 사용
- 힙 메모리 영역

클래스 변수, 함수(static)

- 메소드 밖, 클래스 안에 선언된 변수 중 **static 예약어가 있음**
- 클래스가 생성되면 시작, 프로그램 종료시 소멸
- 프로그램 시작하면서 상수와 함께 데이터 영역에 생성
- **public**으로 선언시 접근은 **클래스명**으로 접근(**this**, 객체명 사용 불가)

참조변수(객체)

- 인스턴스를 참조하기 위한 변수

상수(final)

- **final 예약어**를 사용한 변수
- 변경할 수 없는 값을 선언할 때 사용
- 프로그램 종료시까지 메모리에 있음

03. 메서드

메서드

- 특정 작업을 수행하는 일련의 문장들을 하나로 묶어 이름을 지어논 것
- 높은 재사용성, 중복된 코드의 제거, 프로그램의 구조화
- 선언부 → 메서드의 이름, 매개변수, 반환타입을 명시
- 구현부 → 실행문을 작성 { }
- **반환타입이 'void'가 아닌 경우, 구현부({ })에는 return이 반드시 존재함**
 - void에 return은 생략해도 컴파일러가 메서드 마지막에 자동적으로 삽입시킴
 - return 값은 주로 변수가 오지만, 수식이 오는 경우도 있다. (x + y);
 - return은 매개변수 유효성검사로도 사용한다. (중간에 return시켜 실행을 중지시키는 용도가 이 경우)
 - 참조형 타입의 반환은 '객체의 주소'일 뿐이다.

```
public void method1(){
    '실행문';
}
```

재귀호출(recursive call)

- 자기 자신을 메서드 구현부에 선언하여 반복적으로 불러들이는 것

인스턴스 메서드

- 인스턴스 변수와 관련된 작업을 하는 메서드
- 인스턴스를 생성해야만 구현할 수 있다.

클래스 메서드(static method)

- 인스턴스와 관계없는 메서드

static 멤버는 변수나 메서드에 static을 갖고있는 녀석들만 호출할 수 있다.

인스턴스 멤버는 변수나 메서드를 둘 다 호출할 수 있다.

04. JVM 메모리 구조

메서드 영역(method area)

- Class data를 여기에 저장, 이 대 클래스의 클래스변수도 이 영역에 생성

힙(heap)

- 인스턴스가 생성되는 공간

스택(call stack 또는 execution stack)

- 메서드의 작업에 필요한 메모리 공간을 제공
- 생성 삭제가 이루어지는 공간
- 특징
 - 메서드가 호출되면 수행에 필요한 만큼의 메모리를 스택에 할당 받음
 - 수행을 마치면 사용했던 메모리를 반환하고 스택에서 제거됨
 - 스택의 제일 위에 있는 메서드가 현재 실행중인 메서드
 - 아래에 있는 메서드가 바로 위의 메서드를 호출한 메서드

05. 오버로딩

오버로딩

- 한 클래스 내에 같은 이름의 메서드를 여러개 정의 하는 것
- 조건
 - 메서드 이름이 같아야 함
 - 매개변수의 개수 or 타입이 달라야 함
- 같은 기능을하는 여러이름의 함수를 생성할 필요가 없어짐(메서드 이름을 절약함)

가변인자

- ‘타입... 변수명’과 같은 형태로 매개변수에 선언
- 내부적으로 배열을 이용하는 것
- 갯수가 정해져 있지 않은 같은타입들을 선언할 때 사용
- 반드시 마지막 위치에 선언되어야 함

06. 생성자

생성자

- 인스턴스가 생성될 때 호출되는 ‘인스턴스 초기화 메서드’
- 생성자의 이름은 클래스의 이름과 같아야 함
- 생성자는 리턴 값이 없음
- 연산자 new가 인스턴스를 생성하는 것이지 생성자가 인스턴스를 생성하는 것이 아님!!!!

생성자 생성과정

- 연산자 new에 의해 메모리(heap)에 인스턴스가 생성

- 생성자 클래스명()가 호출되어 수행
- 연산자 new의 결과로, 생성된 클래스의 주소가 반환되어 참조변수에 저장

기본 생성자(Default Constructor)

- 기본적으로 생성하는 생성자
- 클래스에 정의된 생성자가 없을 시 컴파일러가 자동으로 추가

매개변수가 있는 생성자

- 매개변수를 선언하여 호출시 값을 넘겨받아 인스턴스의 초기화작업을 하는 생성자

this()

- 생성자 안에서 다른 생성자를 호출 할 때 사용
- 반드시 첫 줄에서만 호출이 가능
- Default값을 설정할 때 사용

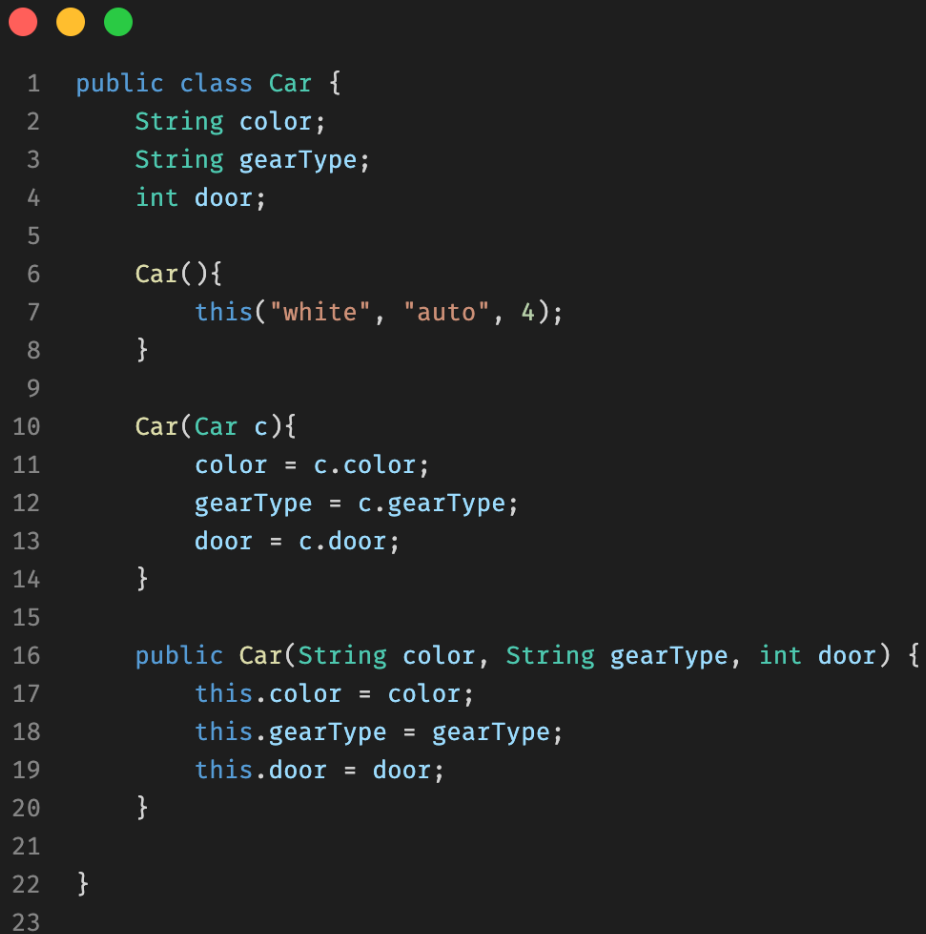
ex) ORACLE의 SYSDATE처럼!!

this

- 생성자 안에서 인스턴스변수를 초기화 시킬때 사용
- 같은 이름으로 매개변수와 인스턴스변수를 지정하는 경우가 많기에 인스턴스변수에 this를 붙여 명시해줌

생성자를 이용한 인스턴스의 복사

- 현재 사용하고 있는 인스턴스와 같은 상태를 갖는 인스턴스를 하나 더 생성할때 생성자를 이용
- 복사하여 생성된 것이므로 서로 같은 상태이지만, 서로 독립적으로 존재하는 별도의 인스턴스가 생성됨



```

1  public class Car {
2      String color;
3      String gearType;
4      int door;
5
6      Car(){
7          this("white", "auto", 4);
8      }
9
10     Car(Car c){
11         color = c.color;
12         gearType = c.gearType;
13         door = c.door;
14     }
15
16     public Car(String color, String gearType, int door) {
17         this.color = color;
18         this.gearType = gearType;
19         this.door = door;
20     }
21
22 }
23

```

07. 변수의 초기화

변수의 초기화

- 멤버변수(클래스변수와 인스턴스변수)와 배열의 초기화는 선택
- 지역변수의 초기화는 필수

명시적 초기화

- 변수를 선언과 동시에 초기화 하는 것

```

1 public class Ex02 {
2     public int solution(String[] babbling) {
3         int answer = 40;
4
5         return answer;
6     }

```

초기화 블록(initialization block)

- 클래스 or 인스턴스 변수의 복잡한 초기화에 사용됨
- 클래스 초기화 블록은 클래스가 메모리에 처음 로딩 될 때 한번만 수행
- 인스턴스 초기화 블록은 생성자와 같이 인스턴스를 생성할 때마다 수행 → **this.~를 사용하는 초기화**

멤버변수의 초기화 시기와 순서

클래스변수의 초기화시점	클래스가 처음 로딩될 때 단 한번 초기화 된다.
인스턴스변수의 초기화시점	인스턴스가 생성될 때마다 각 인스턴스별로 초기화가 이루어진다.
클래스변수의 초기화순서	기본값 → 명시적초기화 → 클래스 초기화 블록
인스턴스변수의 초기화순서	기본값 → 명시적초기화 → 인스턴스 초기화 블록 → 생성자