

## < 객체 >

### 1. 객체 지향 프로그래밍 (Object Oriented Programming) : OOP

➔ 객체를 사용해서 프로그램을 작성하는 것

➔ 객체는 클래스에 의해 만들어짐

> c++ : 클래스에 의해 만들어진 변수를 객체라 함. ( 클래스형 변수명; )

> java : 힙 메모리에 할당된 클래스에 의한 기억공간을 객체라 함, 클래스에 의한 변수는  
레퍼런스[변수]라고 함.

( 클래스형 레퍼런스[변수] = new 클래스명(); )

### 2. OOP의 3대특징 + 1대특징 -> 4대특징

➔ 추상화(Abstraction)

프로그램에서 필요한 공통적인 기능 및 속성들을 추출하고, 불필요한 것을 제거하는 과정

중요한 건 내가 구현하고자 하는 프로그램이 어떤 프로그램인지, 어떤 목적을 가지고 만드는지 생각할 것 !!

➔ 캡슐화(EnCapsulation)

변수 -----> 배열 -----> 구조체

값1개            같은 자료형의 값 여러 개            자료형이 다른 변수들의 묶음

> 클래스(Class) : 구조체 + 정보은닉(접근에 제한을 설정)

> 캡슐화 : 클래스 안의 멤버변수(Field)에 클래스 밖에서는 접근 못하게 하는 것

→ private가 원칙!!

➔ 상속(Inheritance)

➔ 다형성(Polymorphism)

ex)

```
class 클래스명 {
```

```
    ** 멤버변수 == 속성 == 필드
```

```
    private 자료형 변수명;    // 클래스 밖에서는 해당 필드에 절대 접근 X
```

```
    // 해당 필드 값을 처리하는 목적으로 하는 함수(메소드)를 클래스 안에 작성
```

```
    // 메소드를 클래스 밖에서 사용하는 구조
```

```
    // 클래스 밖에서 사용할 메소드는 public으로 처리
```

```
    // 메소드가 필드를 처리하는 기능에 따라
```

```
    ** 생성자함수(Constructor)
```

- 객체가 힙에 할당될 때 객체 안에 만들어지는 필드의 초기화를 담당
- new 할 때 실행되는 함수 (new 할 때 초기값을 생성자 쪽으로 전달하면 됨)
- 생성자 함수가 전달된 초기값을 받아서, 필드에 기록

```
    ** 메소드(Method)
```

- Getter 메소드 : 할당된 필드(인스턴스 변수라고 함)에 기록된 값을 읽어서

```
        요구하는 쪽으로 읽은 값을 넘기는 메소드
```

```
        >> instance == object
```

```
        >> 인스턴스변수 == 객체 안에 있는 필드들
```

- Setter 메소드 : 할당된 필드 값을 변경하는 메소드

```
        바꿀 새 값을 전달받아 해당 필드에 대입
```

- 일반 메소드 : 다른 용도(기능)를 처리하기 위한 메소드

```
}
```

### 3. 클래스

[접근제한자] [예약어] class 클래스명 { }

- 접근제한자 : public, default(생략됨)
- 예약어 : final(종단, 끝), abstract(미완성된, 추상)
- class 클래스명 { } : default 클래스라고 함 -> 해당 패키지 안에 있는 클래스들끼리만 import없이 사용가능
- public class 클래스명 { } : 패키지 밖의 다른 클래스에서 사용하고자 할 경우, import 하면 사용 가능
- public final class 클래스명 { } : 종단 클래스라고 함 -> 상속에 사용 못함  
클래스의 기능 확장을 막기 위해 사용
- public abstract class 클래스명 { } : 추상 클래스라고 함 -> 미완성된 클래스  
스스로 객체 생성 못하는 클래스  
abstract 메소드가 있다면 상속을 통한 후손 클래스의 기능 구현으로 활용 가능한 클래스

### 4. 멤버변수 : Field 작성

[접근제한자] [예약어] 자료형 변수명 [= 초기값];

- 접근제한자 : - private(원칙!!), ~ default(생략됨) , # protected, + public
- 예약어 : final(수정 불가능), static, final static(static final) (정적메모리 할당)
- (-)private : 클래스 안에서만 접근 가능 -> 클래스 밖에서 접근 불가능
- (~)default : package private(같은 패키지안) 같은 패키지 안에 있는 다른 클래스들이 접근 가능함 -> 다른 패키지에 있는 클래스는 접근 불가능
- (#)protected : default의 범위 + 다른 패키지의 상속 받은 후손 클래스까지 접근 가능  
-> 후손클래스 외의 다른 패키지에 있는 클래스는 접근 불가능
- (+)public : 공개를 의미함 -> 패키지 안, 밖 모든 클래스가 접근 가능함

## < 변수 >

### \* 멤버 변수 (인스턴스 변수)

- 메모리 할당 시점 : new라는 연산자를 통한 인스턴스 생성 시
- 소멸 시점 : 인스턴스 소멸 시

### \* 클래스 변수

- 클래스 영역에서 static이라는 키워드가 붙은 변수
- 메모리 할당 시점 : 프로그램 실행 시
- 소멸 시점 : 프로그램 종료 시

### \* 지역 변수

- 메모리 할당 시점 : 메소드 호출(실행) 시
- 소멸 시점 : 메소드 종료 시

## 5. 생성자(Constructor) 함수 작성

- public이 원칙!!
- 반환형 없음
- 함수명은 반드시 클래스명과 같아야 된다.
- 오버로딩(Overloading) 가능함 -> 같은 이름의 함수가 여러 개 작성되는 경우  
-> 반드시 매개변수를 다르게 구성해야 함

### \* 기본 생성자 : **public 클래스명() { }** → 매개변수 없는 생성자

- 클래스에 생성자가 없는 경우, 클래스 사용 시에 자바가상머신(JVM)이 자동으로 기본 생성자를 하나 만들어서 실행함
- 클래스가 상속에 사용되거나 (부모클래스), 또는 매개변수가 있는 생성자를 만들 경우에는 반드시 기본 생성자 작성해야 됨

\* 매개변수 생성자 : **public** 클래스명( 자료형 변수명, 자료형 변수명 ...) {

**[this.]필드명 = 변수명;**

**}**

- 객체 생성 시(할당 시, new 할 때) 원하는 초기값을 인스턴스변수의 초기값으로 기록 원할 때 사용됨
- 생성 시 초기 값을 생성자를 통해 전달하면,  
    클래스명 레퍼런스 = new 생성자( 초기값: 전달값(argument) );  
    → 생성자의 준비된 매개변수가 받아서 필드에 대입함

< **this** 레퍼런스 >

- 생성자와 메소드 안에 무조건 존재함(언제든 사용가능)
- 할당된 객체를 가리킴
- this함수 실행 시 전달되는 객체의 주소를 자동으로 받음. (레퍼런스.메소드명());
- this.필드명 : 현재 객체 안의 필드를 말함

## 6. 메소드 작성

\* 일반 메소드 : [접근제한자][예약어] 반환자료형 메소드명( [자료형 매개변수명, ...] ) {

    처리할 내용 작성

**return;**      → void

**return 값;**   → 값의 자료형

**}**

- 접근제한자 : + **public**, ~ **default**, # **protected**, - **private**
- 예약어 : **static**, **final**(상속 시 오버라이딩 불가), **abstract**(미완성된),  
    **synchronized**(동기화 처리), **static final**(final static)

**\* Setter : Set 메소드**

```
public void set필드명(자료형 매개변수명) {  
    [this.]필드 = 매개변수;  
}
```

- 객체 안의 필드(인스턴스변수)의 값을 변경하는 메소드

**\* Getter : Get 메소드**

```
public 반환자료형 get필드명() {  
    return [this.]필드;  
}
```

- 필드에 기록된 값을 읽어 요구하는 쪽으로 넘기는 메소드 ( 필드 값 조회용 )