객체지향프로그래밍

객체지향프로그래밍

객체지향프로그래밍

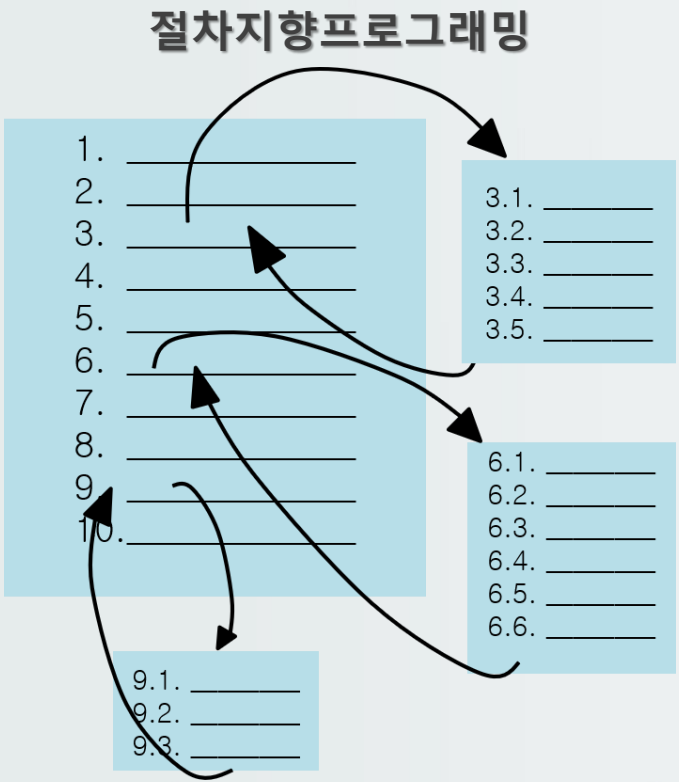
객체지향언어

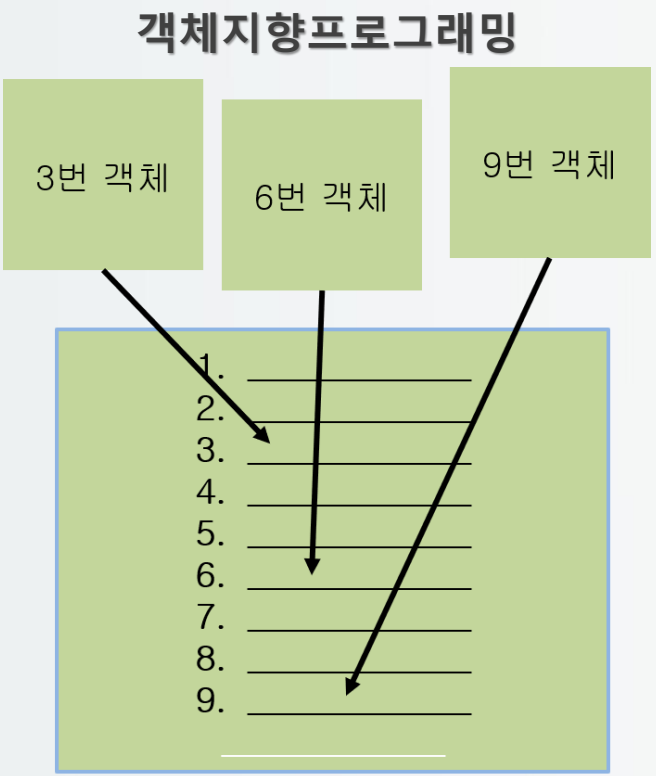
현실 세계는 사물이나 개념처럼 독립되고 구분되는 각각의 객체로 이루어져 있으며, 발생하는 모든 사건들은 객체 간의 상호 작용으로 이루어지며, 이 개념을 컴퓨터로 옮겨 놓아 만들어낸 것

객체지향 프로그래밍

현실세계의 객체(사물, 개념)를 클래스(class)와 객체(object)의 개념으로 컴퓨터에서 구현

절차지향 vs 객체지향


절차지향프로그래밍
1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
3.1.
3.2.
3.3.
3.4.
3.5.
6.1.
6.2.
6.3.
6.4.
6.5.
6.6.
9.1.
9.2.
9.3.

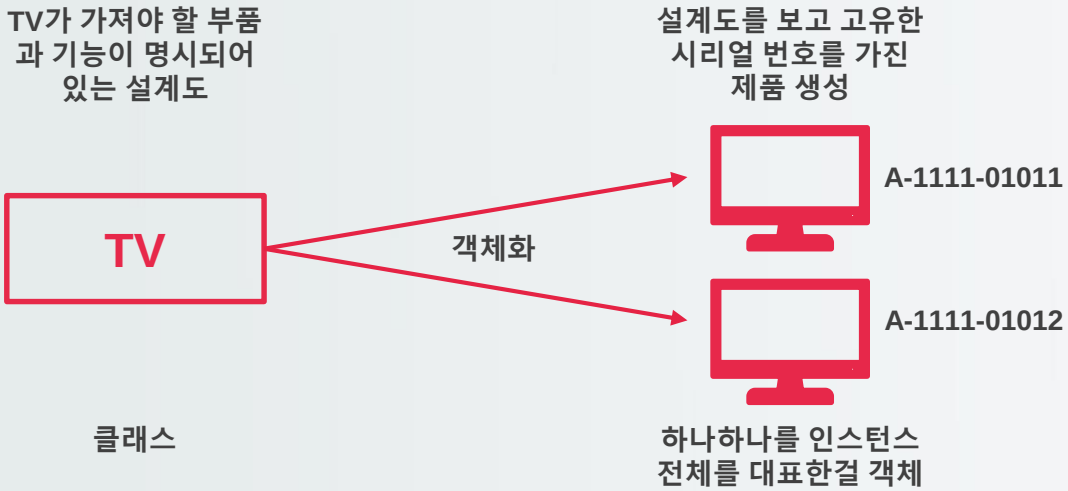

객체지향프로그래밍
3번 객체
6번 객체
9번 객체
1.
2.
3.
4.
5.
6.
7.
8.
9.

객체

객체

객체

사전적 의미 : 실체 하는 모든 사물
객체지향 언어에서 객체의 개념은 new 연산자를 통해 클래스의 설계대로 데이터를 메모리에 할당한 결과물(instance)



객체

클래스

객체를 정의해 놓은 것. 객체의 설계도, 틀
사물이나 개념의 공통 요소(속성, 기능)를 용도에 맞게 추상화

추상화

사전적 의미 : 구체적 사물들의 공통된 특징
프로그램에서 필요한 기능/속성을 추출하고, 불필요한 것을 제거하는 과정

객체

추상화

판매를 위한 TV를 만들기 위해 TV에 필요한 속성과 기능을 정의하기 위한 추상화

속성

- 크기
- 전원
- 볼륨
- 채널

기능

- 전원 on/off
- 볼륨 조절
- 채널 변경

객체

추상화 → 클래스 작성

추상화 한 결과물을 객체지향 프로그래밍 언어를 사용해서 작성
속성 : 추상화를 통해 나온 속성을 저장할 변수(멤버변수)의 데이터 타입과 이름을 정의

구분	자료형	변수명
크기	int	inch
전원	boolean	power
채널	int	channel
볼륨	int	volume

객체

추상화 → 클래스 작성

추상화 한 결과물을 객체지향 프로그래밍 언어를 사용해서 작성
기능 : 추상화를 통해 나온 기능을 메소드로 정의하여 구현

구분	메소드 명	설명
전원 on/off	power()	현재 power의 값을 반대로 변경
채널 변경	channelUp()	현재 채널에서 1 증가
	channelDown()	현재 채널에서 1 감소
볼륨 조절	volumeUp()	현재 볼륨에서 1 증가
	volumeDown()	현재 볼륨에서 1 감소

객체

변수의 종류

변수의 선언 위치에 따라 구분

1. 지역변수
   - 해당지역(메소드)에서만 사용 가능한 변수
2. 전역변수(멤버변수,인스턴스변수)
   - 해당 클래스 내부에서만 사용이 가능한 변수
   - 객체 생성 시 해당 객체에서만 사용이 가능
3. 정적변수(클래스 변수)
   - 클래스와 관계없이 사용 가능한 변수
   - static 키워드를 이용해서 선언

※ 지역변수는 반드시 직접 초기화해서 사용해야 하지만, 전역변수, 정적변수는 선언만 하더라도 기본값으로 초기화 된다.(배열선언 시 초기화 값과 동일)

메소드

변수의 종류

```
public class TestClass{

    public static int size;

    public int num1;

    public void method1(){
        int num2 = 100;
    }
}
```

클래스 영역

메소드 영역

변수 선언

1. 속성에 해당하는 변수는 전역변수로 선언
2. 기능을 구현하기 위해 사용하는 변수는 지역변수로 선언
3. 여러 기능에서 공동으로 사용해야 하는 변수는 전역변수로 선언
4. 정적변수는 특별한 케이스를 제외하고는 사용하지 않음

객체

객체지향 3대 원칙

1. 캡슐화
2. 상속
3. 다형성

객체

캡슐화

추상화를 통해 정의된 데이터들과 기능을 하나로 묶어 관리하는 기법
클래스의 가장 중요한 목적인 데이터의 접근제한을 원칙으로 하여, 클래스 외부에서 데이터의 직접 접근을 막고, 대신 데이터를 처리하는 함수(메소드)들을 클래스 내부에 작성하여 데이터에 접근하는 방식

캡슐화를 위해 필요한 것

1. 접근제어 지시자를 통한 데이터 접근 제한
2. 데이터 처리 함수 (getter,setter)

객체

접근제어지시자(접근제한자)

클래스, 변수, 메소드에 접근할 때 허용 범위

구분	해당 클래스 내부	같은 패키지	후손 클래스	전체
public	O	O	O	O
protected	O	O	O	
default	O	O		
private	O			

접근제어 범위 크기
public > protected > default > private

객체

정보은닉

→ 정보를 숨기는 것
→ 의도치 않은 외부 접근에 대한 오류를 방지하기 위함
→ 만들어진 객체가 개발자의 의도대로 사용 될 수 있도록 하기 위함
→ 클래스의 멤버 변수는 일반적으로 private 영역에 저장

```
public class TV{
    private int inch;
    private boolean power;
    private int channel;
    private int volume;
}
```

※ 변수의 접근제어 지시자를 private로 설정하여 클래스 외부에서 직접접근을 제한

객체

getter/setter

- 변수 선언 시 정보은닉을 위해 private를 사용하면 외부에서 해당 변수의 데이터를 입력하는 것도 불가능하고, 변수안의 데이터 값을 사용하는 것도 불가능

- 객체화를 하여 사용하는 경우 데이터에 값을 사용하기도 해야 하고, 필요에 따라서는 변경도 해야 함

- 메소드는 일반적으로 접근제어 지시자를 public으로 사용하기 때문에 해당 기능을 담당하는 메소드를 getter/setter라고 함

객체

getter

- 멤버변수의 값을 읽어와서 호출한 쪽으로 읽은 값을 넘기는 메소드

```
public class TV{
    private int inch;
    private boolean power;
    private int channel;
    private int volume;

    public boolean getPower(){
        return power;
    }
}
```

getter메소드 작성 방법

1. 접근제어 지시자는 public
2. 리턴타입은 멤버변수 자료형
3. 메소드 이름은 get변수명
   → 카멜 표기법 유지
4. 현재 변수의 값을 리턴

객체

setter

- 멤버변수에 변경할 값을 전달 받아서 멤버변수 값을 변경하는 메소드

```
public class TV{
    private int inch;
    private boolean power;
    private int channel;
    private int volume;

    public void setPower(boolean power){
        this.power = power;
    }
}
```

※ this 레퍼런스
멤버변수와 매개변수의 변수명이 동일할 때 메소드 내부에서 변수명 사용 시 자동으로 지역변수를 호출하기 때문에 멤버변수를 호출하기 위한 키워드

setter메소드 작성 방법
1. 접근제어 지시자는 public
2. 리턴타입은 void
3. 메소드 이름은 set변수명
   → 카멜 표기법 유지
4. 매개변수로 받은 값을 변수에 대입

객체

생성자

- new 연산자를 통해 실행되는 메소드로 객체 안에서 만들어지는 멤버 변수의 초기화를 담당

- 생성자를 통해 전달된 초기 값이 있다면, 이를 멤버변수에 기록

- 생성자의 종류는 기본생성자와 매개변수 있는 생성자로 나뉨

생성자 규칙

1. 생성자는 메소드 선언과 유사하나 리턴값이 존재하지 않음
   → void가 아니라 없음
2. 생성자의 이름은 클래스명과 동일

객체

생성자

```
public class TV{
    private int inch;
    private boolean power;
    private int channel;
    private int volume;

    public TV(){

    }

}
```

매개변수가 없는 생성자 → 기본생성자

객체

생성자

```
public class TV{
    private int inch;
    private boolean power;
    private int channel;
    private int volume;

    public TV(int inch, boolean power, int channel, int volume){
        this.inch = inch;
        this.power = power;
        this.channel = channel;
        this.volume = volume;
    }

}
```

매개변수가 있는 생성자

객체

생성자

- 클래스 작성 시 생성자를 작성하지 않는 경우 JVM이 기본생성자를 자동으로 생성
- 매개변수가 있는 생성자를 작성하는 경우 JVM이 기본생성자를 자동으로 생성하지 않음
- 메소드 오버로딩을 이용하여 여러 개의 생성자 사용 가능
- 일반적으로 기본생성자, 모든 멤버변수를 초기화 하는 매개변수가 있는 생성자를 작성하고, 필요에 따라서 추가로 생성자를 작성.

객체

소멸자

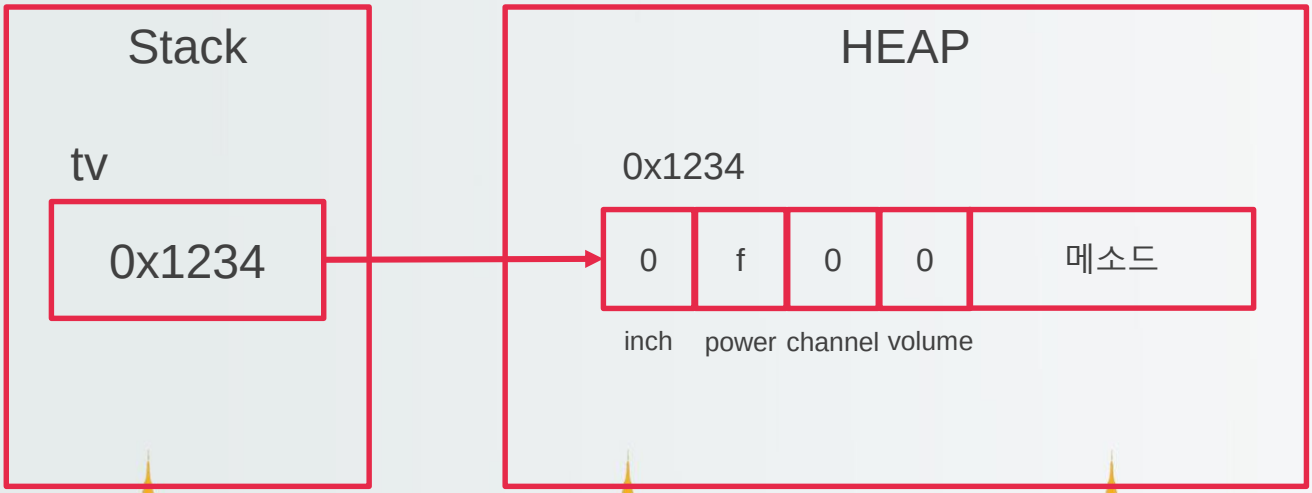
- 생성자와 반대로 객체가 소멸할 때 자동으로 호출되는 메소드
- 타언어에 존재하지만 자바는 언어 특성상 소멸자가 존재하지 않음
- GC(Garbage Collector)가 메모리를 자동으로 정리하기 때문에 소멸자가 필요하지 않음

객체

객체(instance)의 할당

new 생성자();를 사용하여 객체를 생성하면 heap메모리 공간에 서로 다른 자료형의 데이터가 연속으로 나열 할당된 객체공간이 만들어지는데 이를 인스턴스라고 한다.

TV tv = new TV();



객체

객체의 등장 배경

1개의 자료형
1개의 데이터
저장을 위한 공간

변수

1개의 자료형
여러 개의 데이터
저장을 위한 공간

배열

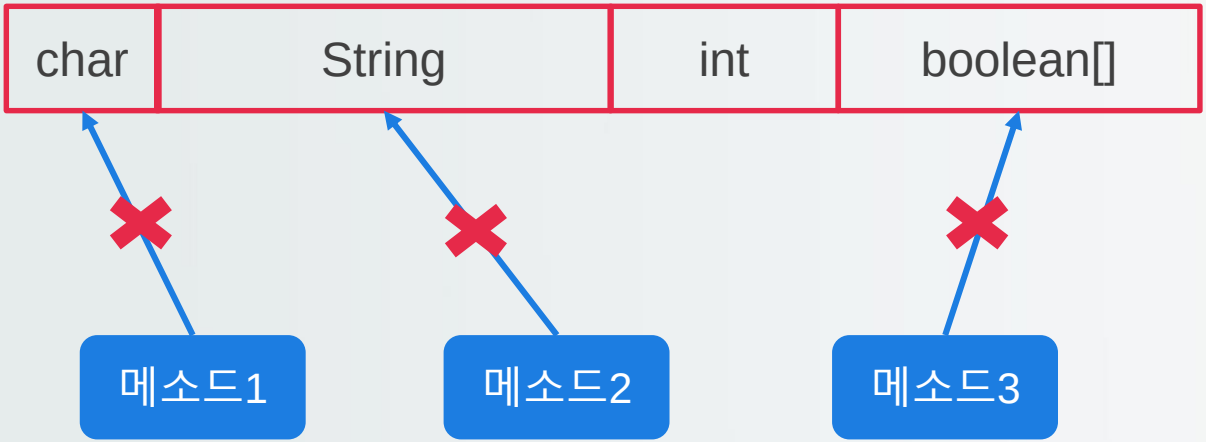
여러 개의 자료형
여러 개의 데이터
저장을 위한 공간

구조체

객체

객체의 등장 배경

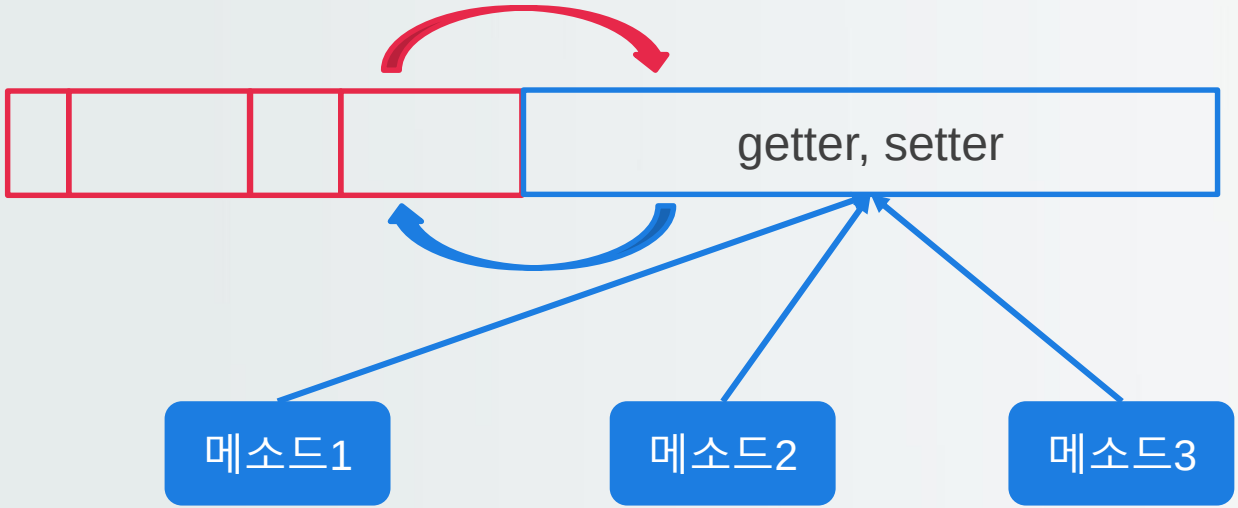
여러 개의 데이터 타입을 가진 구조체에 보안문제로 직접 접근을 금지
→ 데이터를 사용하기 위해 만들었는데 데이터에 대한 접근이 불가능함



객체

객체의 등장 배경

클래스 내부에 데이터를 처리하는 메소드를 이용하여 데이터에 접근



클래스 분류

클래스 분류

클래스 분류

자바에서 데이터를 표현하는 클래스와 해당 데이터를 운영하는 클래스를 별도로 나누어 사용

데이터가 되는 클래스

→ Entity, VO, DTO, Bean Class

데이터를 운영하는 클래스

→ Control Class

※ 컨트롤 클래스에서 Entity클래스를 변수로 사용하여 프로그램을 작성하는 구조로, 이 경우 컨트롤 클래스가 Entity 클래스를 포함하고 있어 has a 관계라고 함(A has a B)

클래스 분류

데이터가 되는 클래스

→ Entity, VO, DTO, Bean Class

```
Public class Student{
    private String name;
    public void setName(String name){
        this.name = name;
    }
    public String getName(){
        return name;
    }

}
```

클래스 분류

데이터를 운영하는 클래스

→ Control Class

```
public class StudentMgr{
    private Student s;
    public void main(){
        s = new Student();
        System.out.println(s.getName());
    }

}
```

객체배열

객체배열

객체배열

객체를 저장하는 배열로 기본자료형 배열과 선언과 할당이 동일하고 자료형 부분에 클래스명으로 지정하여 활용

선언

클래스명[] 변수명 = new 클래스명[배열크기];

TV [] tvArr = new TV[10]

객체배열

할당 후 값 저장

초기화할 인덱스를 선택 후 생성자를 이용하여 대입
참조형 변수 배열과 동일하게 값을 사용하기 위해서는 반드시 할당해야 함

```
tvArr[0] = new TV();
tvArr[1] = new TV();
tvArr[2] = new TV();
```

※생성자를 통한 값 대입을 하지 않는 경우 null이 들어있어 객체를 사용할 수 없음

객체배열 구조

객체배열 구조

TV [] arr[] = new TV[2]
arr[0] = new TV();
arr[1] = new TV();