chapter6 객체지향(1)

ch6-1 객체지향 언어

빠른 변화에 적응하기위해 절차적->객체지향

장점 : 코드의 재사용성↑,유지보수 용이, 중복코드제거

객체지향(oop, object-oriented programing) 핵심개념

-> 캡(슐화), 상(속), 추(상화), 다(향성)

ch6-2 클래스와 객체

클래스 : 객체를 정의해 놓은 설계도

클래스의 용도 : 객체를 생성하는데 사용

객체 : 실제로 존재하는 것. 사물, 개념, 제품

객체의 용도 : 객체가 가지고 있는 기능과 속성에 따라 다름

ch6-3 : 객체의 구성요소 - 속성과 기능

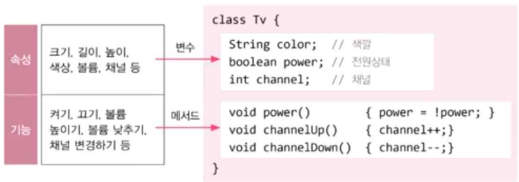
처음 객체지향의 개념 실제세계를 컴퓨터화(코드와)

-> 시행착오를 줄이기 위해

분석과 관찰..

객체 = 속성(변수) + 기능(메서드)

ex)



ch6-4 : 객체와 인스턴스

객체 = 인스턴스(클래스로부터 생성된 객체)

(인스턴스화)

클래스 ----------> 인스턴스(객체)

클래스(설계도)가 왜 필요한가? 객체(제품)생성을 위해

객체(제품)는 왜 필요한가? 객체를 사용하기위해

객체를 사용한다는 것? 객체의 속성과 기능을 사용

ch6-5 하나의 소스파일에 여러 클래스 작성

// 하나의 소스파일에 하나의 클래스가 바람직!

ch6-6,7 객체의 생성과 사용

1. 생성 - ① Tv t; <-참조변수 (리모콘)

② t = new Tv(); <-객체 (new로 연결)

Tv객체는 Tv리모콘으로 밖에 다를 수 없기 때문에 리모콘과 객체의 타입이 Tv로 일치 해야한다.

2. 사용 - 변수 사용 t.channel = 7;

메서드 사용 t.channelDown();

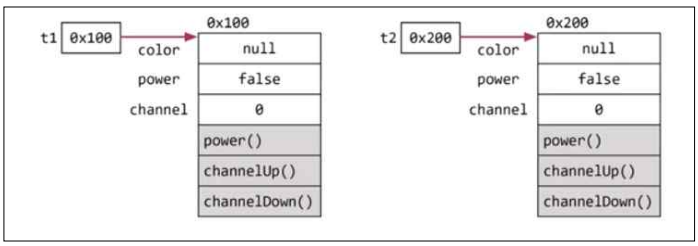
※1.클래스(설계도)생성 ->2.객체(제품)생성 ->3.객체(제품)사용

ch6-7 객체의 생성과 사용 - 예제

Tv t1= new Tv();

Tv t2= new Tv();

-> 별도의 저장공간이 생김



t2=t1 연결시 t2는 0x200와 연결이 끊기고 0x100를 가르킨다.

(0x200객체는 리모콘 없어 사용불가로 G.C로 정리)

ch6-8 객체 배열

객체 배열 == 참조변수 배열

Tv tv1, tv2, tv3; ------> Tv[] tcArr = new Tv[3];

//길이가 3인 Tv 타입의 참조변수 배열

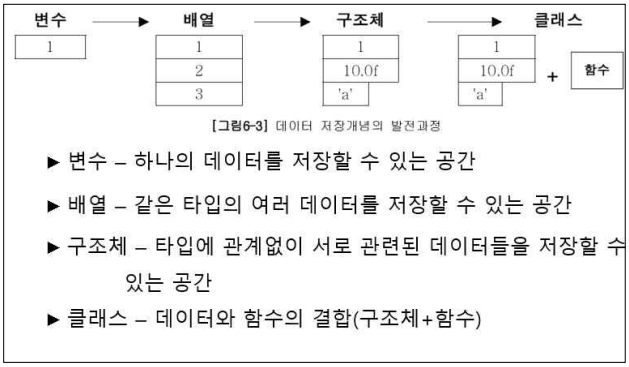
```
Tv[] tvArr = new Tv[3];
tvArr[0] = new Tv();
tvArr[1] = new Tv();
tvArr[2] = new Tv();
```

객체 배열을 만들고 꼭 참조변수 각 번지마다 객체를 넣어줘야 한다.!

ch6-9 클래스의 정의(1)

클래스는 ①설계도, ②데이터+함수, ③사용자 정의타입

클래스 == 데이터 + 함수



[그림6-3] 데이터 저장개념의 발전과정

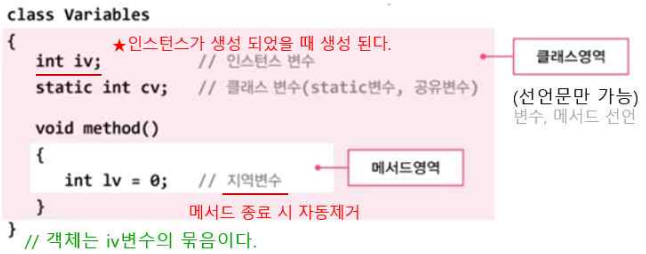
- ▶ 변수 – 하나의 데이터를 저장할 수 있는 공간
- ▶ 배열 – 같은 타입의 여러 데이터를 저장할 수 있는 공간
- ▶ 구조체 – 타입에 관계없이 서로 관련된 데이터들을 저장할 수 있는 공간
- ▶ 클래스 – 데이터와 함수의 결합(구조체+함수)

ch6-10 클래스의 정의(2)

사용자 정의 타입 - 원하는 타입을 직접 만들 수 있다.

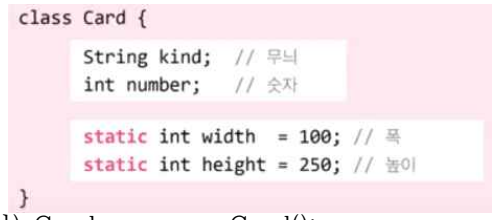
클래스라는 사용자 정의 타입을 만들어서 클래스 안의 서로 다른 타입의 관련 데이터를 같은 클래스 타입으로 묶어서 사용 가능하다.
때문에 유지 보수가 쉽고, 코드가 간결 해진다.

ch6-11 선언위치에 따른 변수의 종류



ch6-12,13 클래스 변수와 인스턴스 변수

카드를 예를 들 때
공통속성은 cv, 개별속성은 iv를 쓴다.

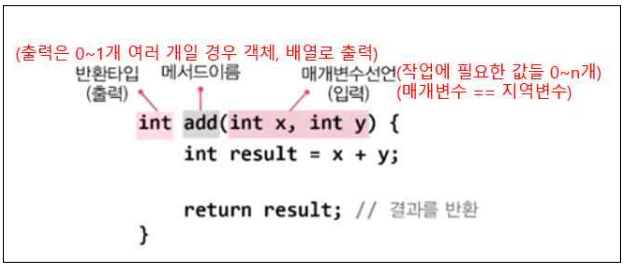


예시) Card c = new Card();
c.kind = "heart";
c.number = 5;
Card.width = 200;
Card.heigh = 300;

cv와 iv를 구분하기 위해 cv는 클래스명을 쓴다.
cv는 공통적으로 쓰이기 때문에 값 변경시 해당 클래스를 참조하고 있는 모든 객체들의 공통 값이 바뀐다.

ch6-14 메서드란?

1. 문장들을 묶어 놓은 것
 - 작업 단위로 문장들을 묶어서 이름 붙인 것
2. 값을 받아서 처리하고 결과를 반환(출력)



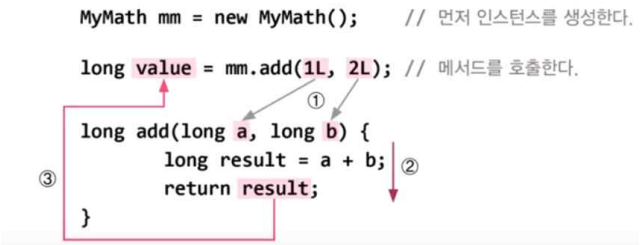
메서드의 장점
- 코드의 중복코드제거, 관리용이, 재사용 가능
메서드의 작성
- 반복적으로 수행되는 여러 문장을 메서드로 작성
- 하나의 메서드는 한 가지 기능만 수행하도록 작성

ch6-17,18,19 메서드의 호출, 실행 흐름

메서드이름(값1, 값2 ...); //메서드 호출 하는 방법

print99danAll(); //단순히 메서드의 구구단을 출력
int result = add(3, 5) //값을 보내 작업결과를 return

<실행흐름>



ch6-20 return문

실행 중인 메서드를 종료하고 호출한 곳으로 돌아간다.

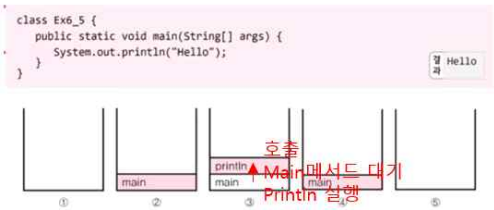
-메서드가 작업을 마쳤을 때 return문을 써야하지만 반환타입이 void일때 생략 가능하다.(컴파일러가 자동추가)
-return문은 참일 때만 실행가능 하기 때문에 반환 결과가 거짓일 때도 명시 해줘야한다.

ch6-21 반환값

반환값이 void가 아닐 때 반환할 결과(result)와 반환타입(메서드 선언시 타입)과 결과값을 담을 변수의 타입이 일치해야 한다.

ch6-22 호출 스택

스택 : 밑이 막힌 상자. 위에 차곡차곡 쌓인다.
-넣을 때 쌓이고 꺼낼 때 최근에 넣은 것부터 꺼낸다.
-메서드 수행에 필요한 메모리가 제공되는 공간
-메서드 호출시 호출스택에 메모리 할당, 종료 시 해제



아래 있는 메서드가 위에 메서드를 호출한 것
맨 위의 메서드 하나만 실행, 나머지는 대기

ch6-23,24 기본형 매개변수, 참조형 매개변수

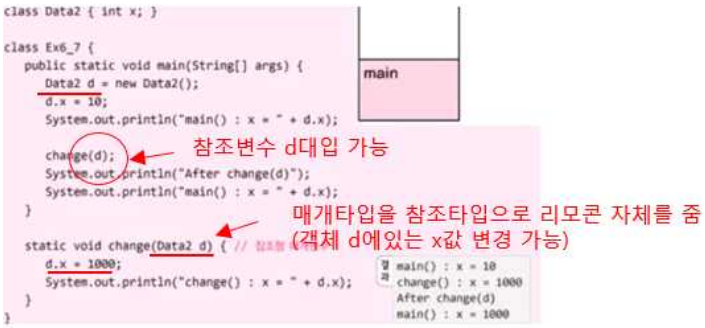
기본형(8개) 매개변수 - 변수의 값을 읽기만

참조형(리모콘) 매개변수 -변수의 값을 읽고, 변경

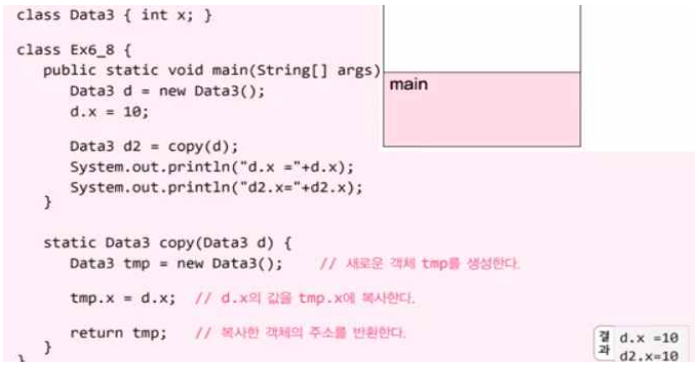
<기본형>

```
static void change(int x) {  // 기본형 매개변수
    x = 1000;
    System.out.println("change() : x = " + x);
```

<참조형>



ch6-25 참조형 반환타입



ch6-26,27 static 메서드와 인스턴스 메서드

▶인스턴스 메서드

-인스턴스 생성후, '참조변수.메서드이름()'으로 호출

-인스텐스 멤버(iv,im)관련 작업을 하는 메서드

-메서드 내에서 인스턴스 변수(iv) 사용가능

▶static 메서드

-객체생성 없이 '클래스이름.메서드 이름()'으로 호출

-인스턴스 멤버(iv,im)와 관련없는 작업을 하는 메서드

-메서드 내에서 인스턴스 변수(iv) 사용불가

객체는 iv의 묶음!! iv를 쓰지 않을거면 객체 생성 필요 없다.

ch6-28 static은 언제 붙여야 할까?

속성(멤버 변수)중에서 공통속성에 staic을 붙인다.

iv,im을 사용 안하면 static메서드, 사용하면 인스턴스 메서드를 쓴다.

★ 멤버변수와 메서드의 static용도가 다름

★ 변수는 공통사용 여부, 메서드는 iv,im 사용 여부

ch6-28,29 메서드 간의 호출과 참조

static 메서드는 인스턴스 변수, 인스턴스 메서드를 사용할 수 없다.

-> 인스턴스 멤버(iv, im)는 객체 생성 후 호출 가능 한데 static메서드 호출시 객체 생성이 되어 있다는 보장이 없음!!

ch6-30,31 오버로딩(overloading)

한 클래스 안에 같은 이름의 메서드가 여러개 있는 것

매개변수는 다르지만 같은 기능을 수행

오버로딩 성립조건

1. 메서드 이름이 같아야한다.(=수행 작업이 같음을 의미)
2. 매개변수의 개수 또는 타입이 달라야한다.
3. 반환 타입은 영향이 없다.

ch6-32 생성자 (constructor)

- 인스턴스가 생성될 때마다 호출되는
  '인스턴스(객체=iv묶음)초기화 메서드'
- 인스턴스 생성시 수행할 작업에 사용

```
Time t = new Time();
t.hour = 12;
t.minute = 34;
t.second = 56;
```

=

```
Time t = new Time(12, 34, 56);
```

생성자 호출

iv초기화시 편리를 위해 사용!

규칙

-이름이 클래스 이름과 같아야 한다.
(생성자 오버로딩가능) ㅐ

-리턴값이 없다. 대입만.. (void 안붙임)

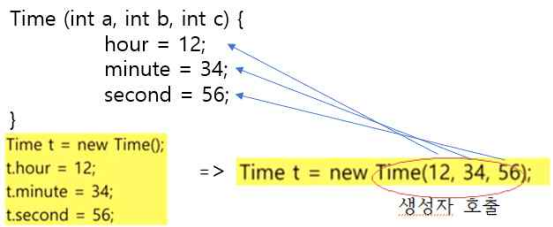
-모든 클래스는 반드시 생성자를 가진다.

ch6-33 기본 생성자 (default constructor)

-생성자가 하나도 없을 때만 컴파일러가 기본 생성자를 자동 주가 해준다.

★ 하나의 클래스에서 기본 생성자 이외 매개변수가 있는 생성자가 있을시 컴파일러가 기본 생성자를 생성해 주지 않기 때문에 직접 기본 생성자를 명시해 줘야한다.

ch6-34 매개 변수가 있는 생성자



밖에 쓸 코드를 안에 쓴 것!

코드가 간결해지고 편리하게 사용, 초기화가 쉽다.

ch6-36 생성자 this()

-같은 클래스 안 생성자에서 다른 생성자 호출할 때 사용한다.

(iv초기화시 코드의 중복을 제거 하기 위해서 생성자들끼리 서로 호출한다.)

-다른 생성자 호출시 첫 줄에서만 사용가능

```
class Car2 {
    String color;     // 색상
    String gearType;  // 변속기 종류 - auto(자동), manual(수동)
    int door;         // 문의 개수

    Car2() {
        this("white", "auto", 4);
    }

    Car2(String color) {
        this(color, "auto", 4);
    }

    Car2(String color, String gearType, int door) {
        this.color = color;
        this.gearType = gearType;
        this.door = door;
    }
}
```

Car2(String color, String gearType, int door)를 호출

ch6-37 참조변수this()

(생성자this()와 완전다름 주의)

-인스턴스 자신을 가리키는 참조변수
(객체주소가 저장되어 있음)

-인스턴스 메서드(생성자 포함)에서 사용가능

-지역변수(iv)와 인스턴스 변수 (iv)를 **구별**할 때 사용

this는 모든 im에 지역변수로 숨겨져 존재하기 때문에 선언하지 않아도 바로 쓸 수 있다.

```
class MyMath2 {
    long a, b; // this.a, this.b

    MyMath2(int a, int b) { // 생성자
        this.a = a;
        this.b = b;
    }
    long add() {   // 인스턴스 메서드
        return a + b;  // return this.a + this.b;
    }
    static long add(long a, long b) { // 클래스 메서드(static메서드)
        return a + b;
```

·인스턴스 메서드에서 생략가능

·lv와 iv의 이름이 같은때는 구분을 위해 this 꼭 명시

·클래스 메서드에서 사용불가 (iv가 사용 안되기 때문)

ch6-38 변수의 초기화

지역변수(lv)는 수동으로 초기화 해야함(꼭!!!!!)

-사용이 빈번한 메서드를 호출할 때마다 lv를 자동으로 0 초기화 한다면 메모리 낭비-성능↓

인스턴스변수(iv)는 자동으로 초기화 된다.

-인스턴스화 할 때 객체에 생명주기가 긴 많은 변수, 배열등을 수동으로 하기엔 너무 많기 때문

ch6-39 멤버변수(iv, cv)의 초기화

1. 명시적 초기화(간단)

```
int door = 4;              // 기본형(primitive type) 변수의 초기화
Engine e = new Engine();   // 참조형(reference type) 변수의 초기화
```

참조형 변수는 객체주소를 넣음으로써 초기화 한다.

2. 초기화 블록(복잡)

- 인스턴스 초기화 블록 : {} <-iv 초기화 할 때 사용
- 클래스 초기화 블록 : static{} <-cv 초기화 할때 사용

3. 생성자

iv초기화, 복잡한 초기화

순서 : 자동 --> 간단 --> 복잡
0 = static,생성자

정리---

1. 명시적초기화 (=) <--간단초기화
2. 초기화 블록 -{}, static{} <--복잡초기화(cv,iv초기화)
3. 생성자 <--복잡초기화(iv초기화)

cv 초기화 시점 : 클래스가 처음 로딩 될 때 한번

iv 초기화 시점 : 인스턴스가 생설 될 때 마다

객체를 만들면

cv가 먼저 초기화 되고 iv가 초기화 된다. cv->iv

객체를 또 만들면 cv는 그대로 iv초기화는 반복