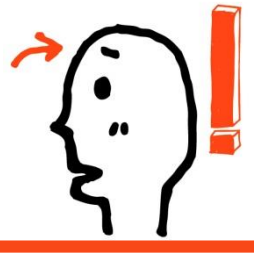
Android Java

6장 객체프로그래밍의 시작(클래스 정의와 인스턴스의 생성)

객체 지향 프로그래밍의 시작

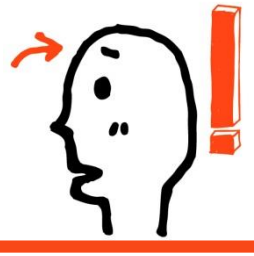


01 클래스와 객체, 인스턴스 개념 구별하기

02 클래스를 구성하는 메소드와 생성자, 그리고 속성

03 자바의 이름 규칙(Naming Rule)

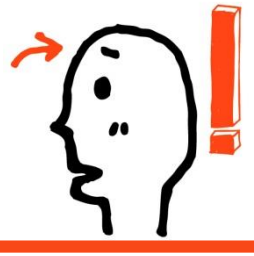
클래스와 객체, 인스턴스 개념 구별하기



객제지향의 목적

- 소프트웨어의 생산성 향상
  - 컴퓨터 산업 발전에 따라 소프트웨어의 생명 주기(life cycle) 단축
  - 객체 지향 언어는 상속, 다형성, 객체, 캡슐화 등 소프트웨어 재사용을 위한 여러 장치 내장
    - 소프트웨어의 재사용과 부분 수정을 통해 소프트웨어를 다시 만드는 부담을 대폭 줄임으로써 소프트웨어의 생산성이 향상
- 실세계에 대한 쉬운 모델링
  - 과거
    - 수학 계산/통계 처리를 하는 등의 처리 과정, 계산 절차가 중요
  - 현재
    - 컴퓨터가 산업 전반에 활용
    - 실세계에서 발생하는 일을 프로그래밍
      - 실세계에서는 절차나 과정보다 일과 관련된 물체(객체)들의 상호 작용으로 묘사하는 것이 용이
  - 실세계의 일을 보다 쉽게 프로그래밍하기 위한 객체 중심의 객체 지향 언어 탄생

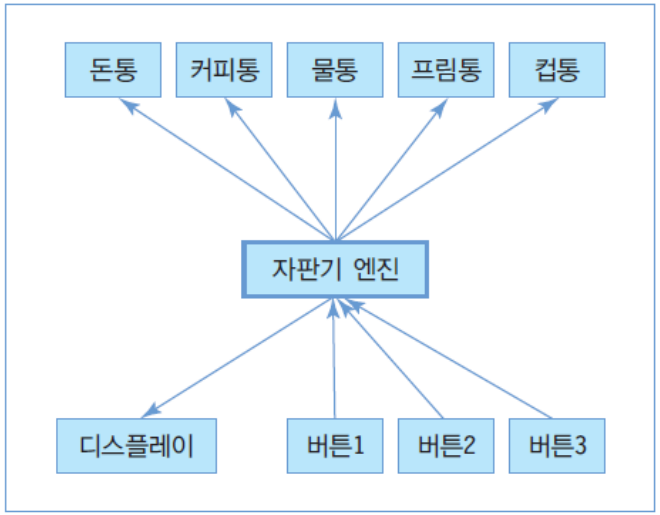
클래스와 객체, 인스턴스 개념 구별하기



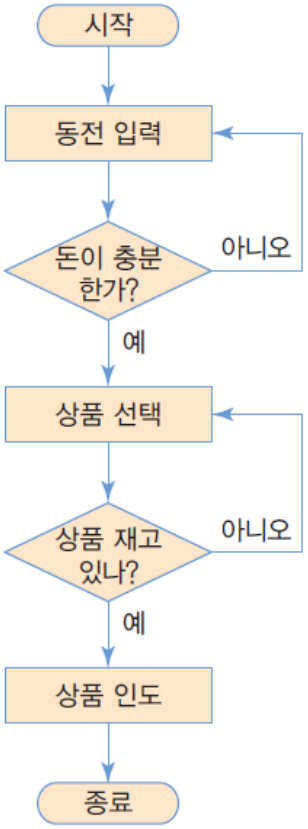
객체지향의 목적

- 절차 지향 프로그래밍
  - 작업 순서를 표현하는 컴퓨터 명령 집합
  - 함수들의 집합으로 프로그램 작성
- 객체 지향 프로그래밍
  - 프로그램을 실제 세상에 가깝게 모델링
  - 컴퓨터가 수행하는 작업을 객체들간의 상호 작용으로 표현
  - 클래스 혹은 객체들의 집합으로 프로그램 작성

커피 자판기

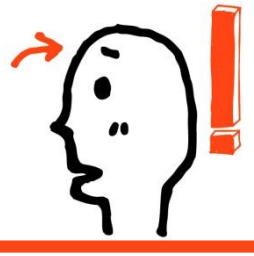


객체지향적 프로그래밍의 객체들의 상호 관련성



절차지향적 프로그래밍의 실행 절차

클래스와 객체, 인스턴스 개념 구별하기



객체 지향 프로그래밍은 일종의 개발 이론으로 프로그램을 설계, 개발하는 방법에 대한 내용

▷ 객체 지향 프로그래밍은 프로그래밍하는 이론이지 문법이 아니다

한 개 이상의 클래스들이 서로 유기적으로 동작할 때 그것을 객체 지향 프로그래밍이라고 함

▷ 클래스는 객체 지향 프로그래밍의 기본이자, 자바 프로그래밍의 기본 단위다

Object = 객체, Oriented = ~지향하는 또는 (기능적으로) 방향 지어진, Programming = 프로그래밍

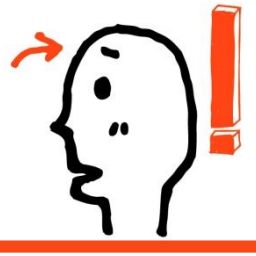
"객체를 목적으로 하는 프로그래밍 방법"

객체를 만드는 게 객체 지향 프로그래밍의 주된 목적

▷ 객체 지향 프로그래밍은 객체를 만들어서 사용하는 것이다

객체 지향 프로그래밍에서 클래스와 객체를 빼놓고 설명할 수 없음

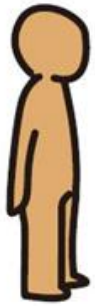
클래스와 객체, 인스턴스 개념 구별하기



- 눈으로 볼 수 있는 모든 것들을 각각 객체라고 부를 수 있음

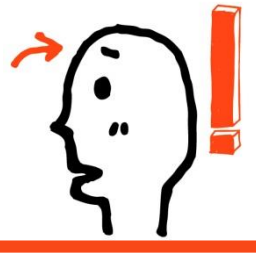
집 객체

차 객체



사람 객체

클래스와 객체, 인스턴스 개념 구별하기

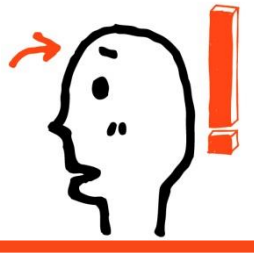


- 걷는 것과 뛰는 것

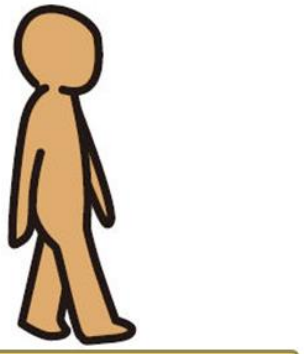
void walk( )

void run( )

클래스와 객체, 인스턴스 개념 구별하기



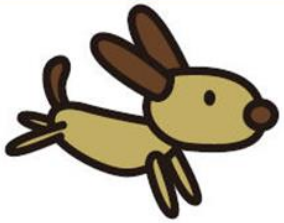
- 사람의 동작과 강아지의 동작을 함수 이름으로 구분하기



void personWalk( )

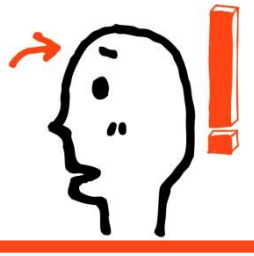
void personRun( )

void dogWalk( )

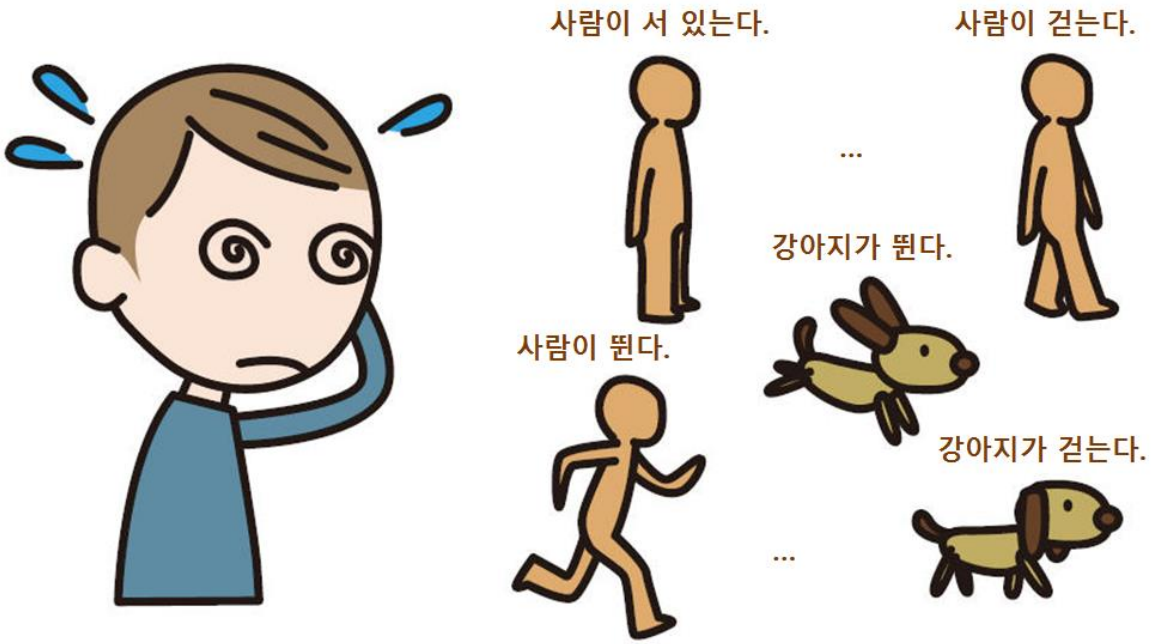


void dogRun( )

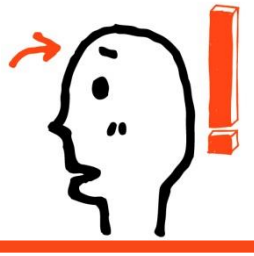
클래스와 객체, 인스턴스 개념 구별하기



- 똑같은 동작을 하는 함수가 너무 많을 때

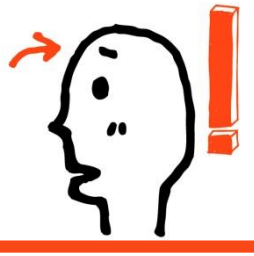


클래스와 객체, 인스턴스 개념 구별하기

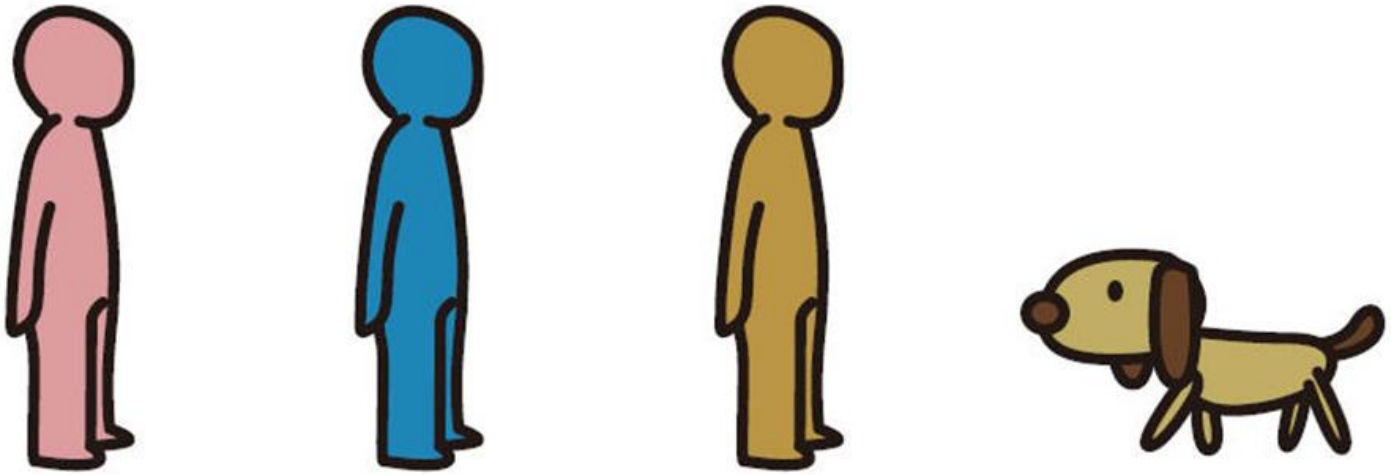


- 사람이 하나의 클래스, 강아지도 하나의 클래스

클래스와 객체, 인스턴스 개념 구별하기

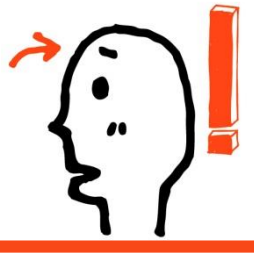


- 각각의 사람이나 강아지를 객체로 보기

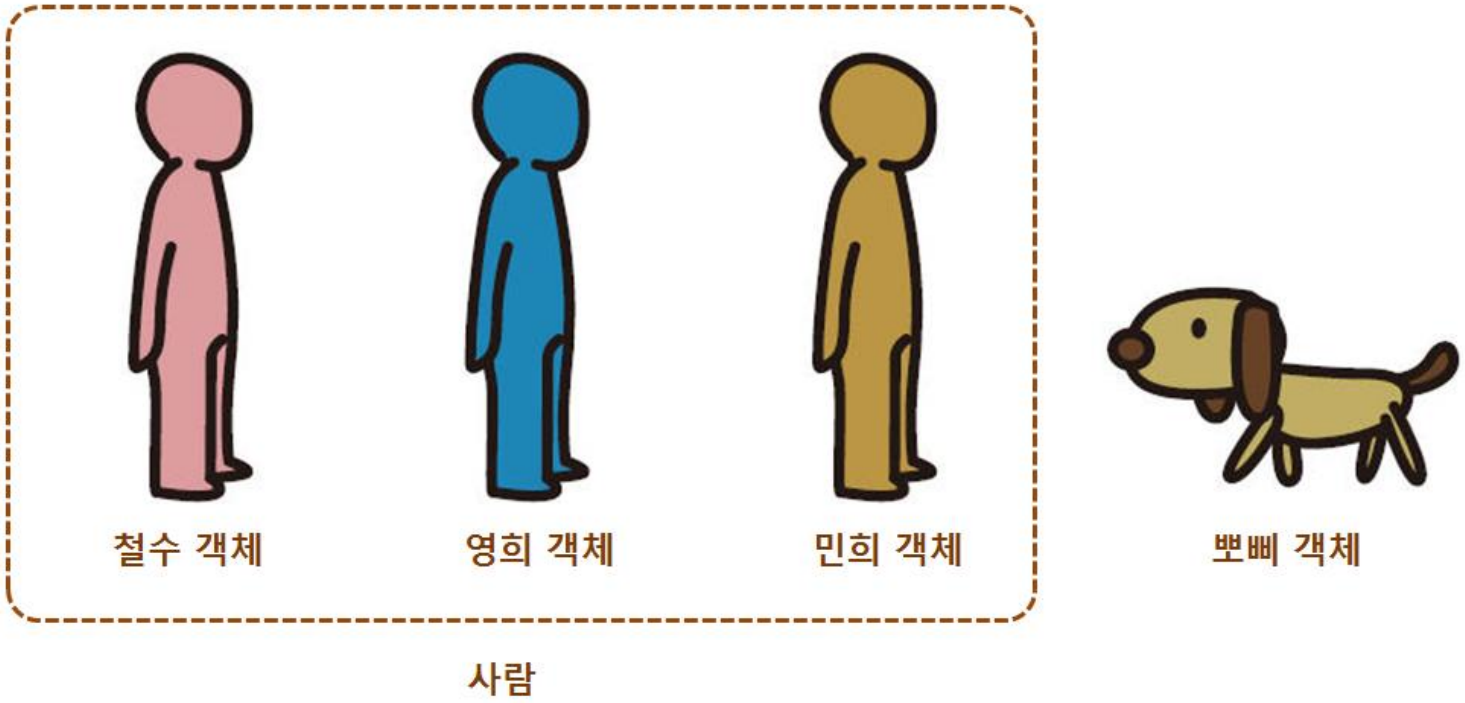


철수 객체 영희 객체 민희 객체 뽀삐 객체

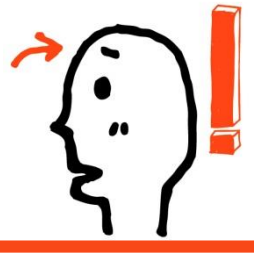
클래스와 객체, 인스턴스 개념 구별하기



- 각각의 객체 중에서 같은 속성을 가진 것들을 묶을 수 있음



클래스와 객체, 인스턴스 개념 구별하기

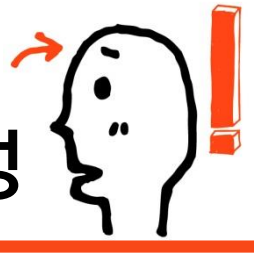


객체는 무엇인가?

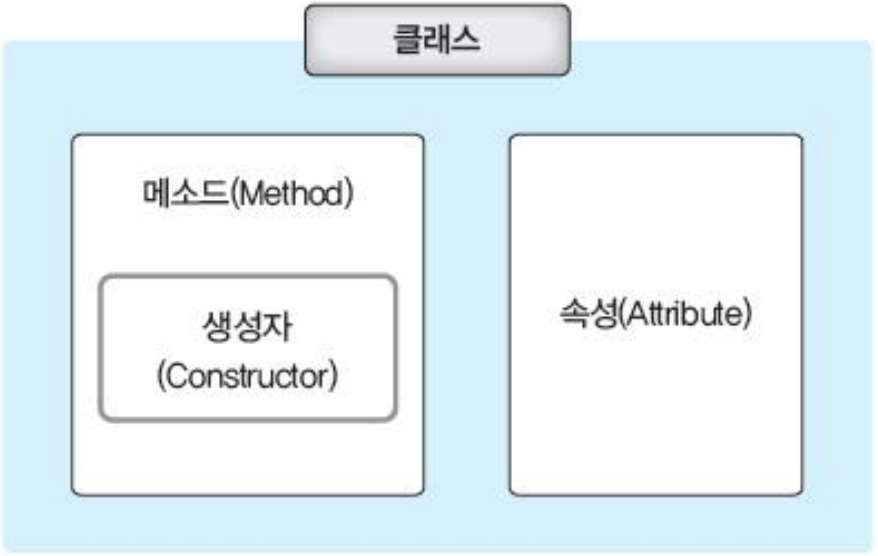
"**객체**는 데이터 혹은 **기능**을 갖고 있어, 그 **목적**을 수행하는 것이다."

- **객체(object)** : 어떤 목적이나 기능을 가진 것
- 사물 혹은 어떤 개념들이 단어로 존재, 우리 주위에 있는 모든 것들은 현실 세계의 객체
- 객체 지향 프로그래밍에서도 마찬가지, JVM에서 존재하는 모든 것들이 객체가 되는 것
- 객체 지향 프로그램도 각각 기능과 목적을 갖고 있는 한 개 또는 그 이상의 객체들로 구성되고 그 객체들이 서로 유기적으로 동작하면서 전체 프로그램이 동작

클래스를 구성하는 메소드와 생성자, 그리고 속성



클래스를 구성하는 핵심 요소



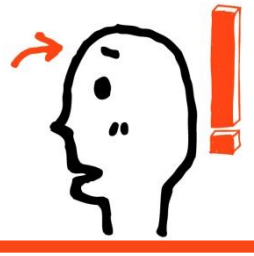
△ 그림 5-3 클래스의 구성

▷ **"클래스 = 데이터 영역 + 기능 영역"**

▷ 객체의 기능(function, behavior)은 클래스의 메소드로부터 만들어지며 객체의 데이터는 클래스의 속성으로부터 만들어짐

▷ 클래스는 여러 개의 메소드와 속성을 포함할 수 있음

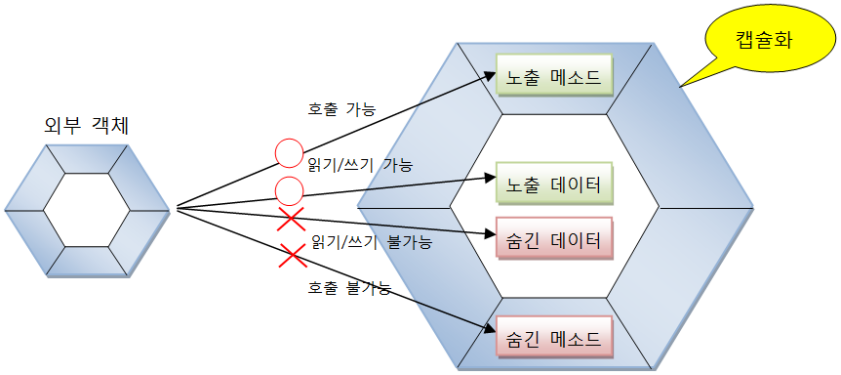
클래스와 객체, 인스턴스 개념 구별하기



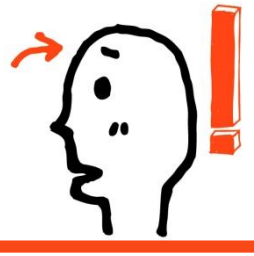
객체 지향 프로그래밍의 특징

캡슐화

- 객체의 필드, 메소드를 하나로 묶고, 실제 구현 내용을 감추는 것
- 외부 객체는 객체 내부 구조를 알지 못하며 객체가 노출해 제공하는 필드와 메소드만 이용 가능
- 필드와 메소드를 캡슐화하여 보호하는 이유는 외부의 잘못된 사용으로 인해 객체가 손상되지 않도록 하기 위함
- 자바 언어는 캡슐화된 멤버를 노출시킬 것인지 숨길 것인지 결정하기 위해 접근 제한자(Access Modifier) 사용



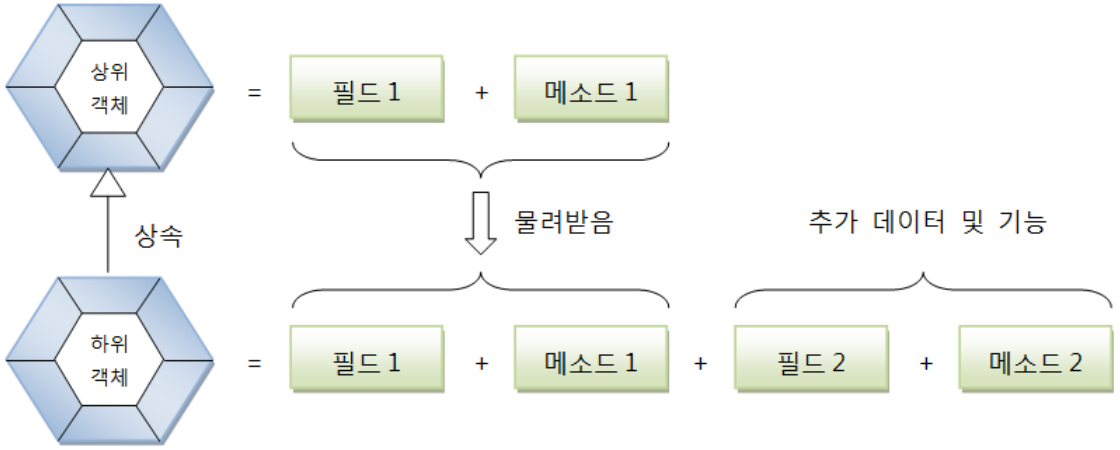
클래스와 객체, 인스턴스 개념 구별하기



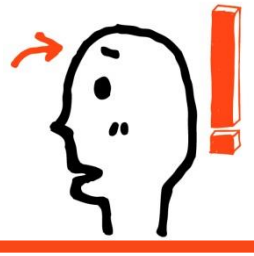
객체 지향 프로그래밍의 특징

상속

- 상위(부모) 객체의 필드와 메소드를 하위(자식) 객체에게 물려주는 행위
- 하위 객체는 상위 객체를 확장해서 추가적인 필드와 메소드를 가질 수 있음
- 상속 대상: 필드와 메소드
- 상속의 효과
  - 상위 객체를 재사용해서 하위 객체를 빨리 개발 가능
  - 반복된 코드의 중복을 줄임
  - 유지 보수의 편리성 제공
  - 객체의 다형성 구현



클래스와 객체, 인스턴스 개념 구별하기

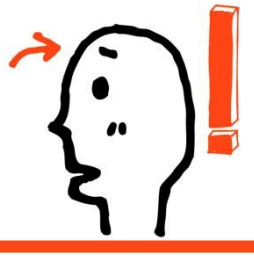


객체 지향 프로그래밍의 특징

다형성 (Polymorphism)

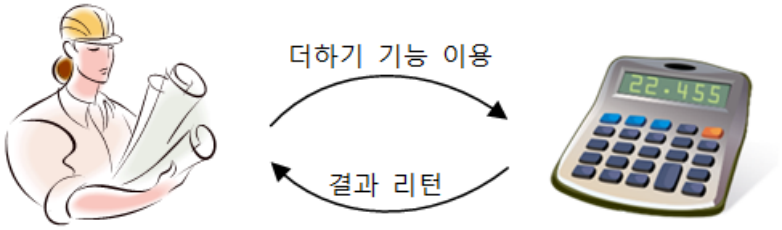
- 같은 타입이지만 실행 결과가 다양한 객체를 대입할 수 있는 성질
  - 부모 타입에는 모든 자식 객체가 대입
  - 인터페이스 타입에는 모든 구현 객체가 대입
- 효과
  - 객체를 부품화시키는 것 가능
  - 유지보수 용이

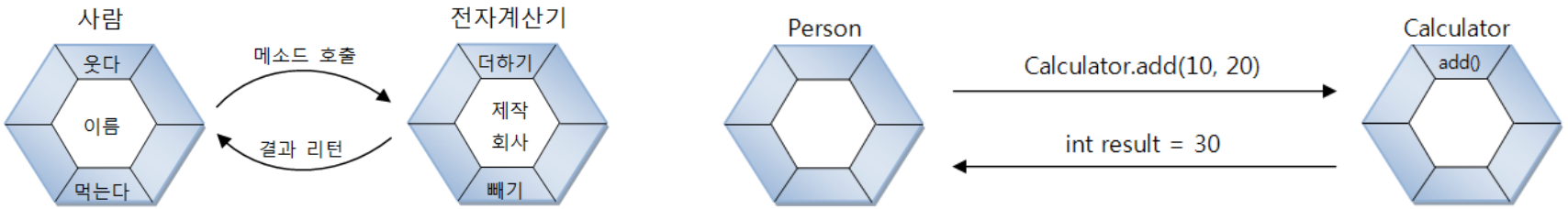
클래스와 객체, 인스턴스 개념 구별하기



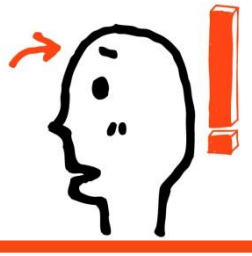
객체의 상호작용

▷ 객체들은 서로간에 기능(동작)을 이용하고 데이터를 주고 받음





클래스와 객체, 인스턴스 개념 구별하기



객체는 무엇인가?

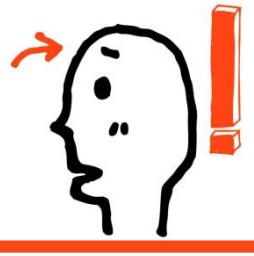
객체 객체 객체

"나는 과일장수에게 두 개의 사과를 구매했다!"

데이터 행위, 기능

객체지향 프로그래밍에서는 나, 과일장수, 사과라는 객체를 등장 시켜서 두 개의 사과 구매라는 행위를 실체화 한다.

클래스와 객체, 인스턴스 개념 구별하기



객체는 무엇인가?

과일장수 객체의 표현

과일장수는 과일을 팝니다. 행위
과일장수는 사과 20개, 오렌지 10개를 보유하고 있습니다. 상태
과일장수의 과일판매 수익은 50,000원입니다. 상태

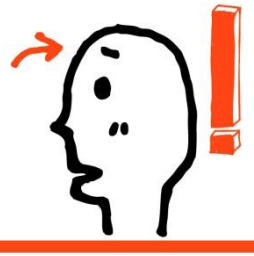
과일장수의 데이터 표현

보유하고 있는 사과의 수: int numOfApple;
판매 수익: int myMoney;

과일장수의 행위 표현

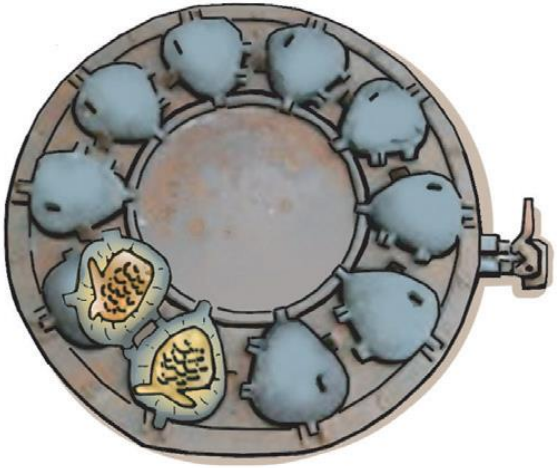
```
int saleApple(int money)      // 사과 구매액이 메소드의 인자로 전달
{
    int num = money/1000;     // 사과가 개당 1000원이라 가정
    numOfApple -= num;    // 사과의 수가 줄어들고,
    myMoney += money;     // 판매 수익이 발생한다.
    return num;           // 실제 구매가 발생한 사과의 수를 반환
}
```

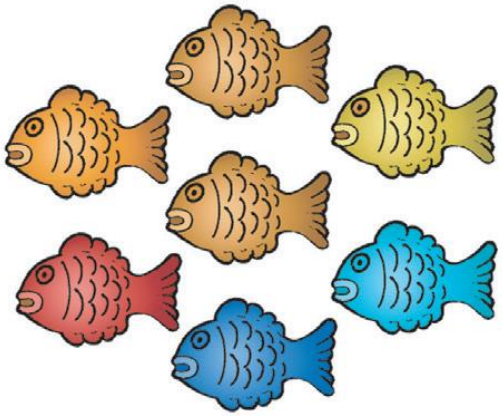
클래스와 객체, 인스턴스 개념 구별하기



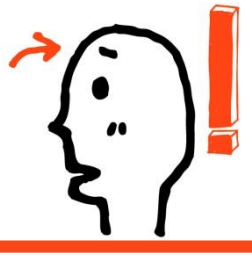
기능 명세서와 같은 클래스

붕어빵 틀은 클래스이며, 이 틀의 형태로 구워진 붕어빵은 바로 객체입니다. 붕어빵은 틀의 모양대로 만들어지지만 서로 조금씩 다릅니다.
치즈붕어빵, 크림붕어빵, 앙코붕어빵 등이 있습니다.
그래도 이들은 모두 붕어빵입니다.





클래스와 객체, 인스턴스 개념 구별하기



기능 명세서와 같은 클래스

클래스: 사람

이름, 직업,나이, 성별, 혈액형
밥 먹기, 잠자기, 말하기, 걷기

이름 최승희
직업 의사
나이 45
성별 여
혈액형 A

객체 : 최승희

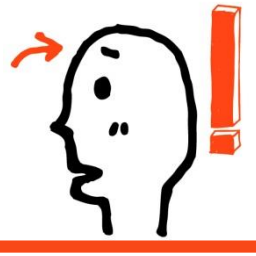
이름 이미녀
직업 골프선수
나이 28
성별 여
혈액형 O

객체 : 이미녀

이름 김미남
직업 교수
나이 47
성별 남
혈액형 AB

객체:김미남

클래스와 객체, 인스턴스 개념 구별하기



클래스(class)라는 틀 기반의 객체 생성

객체 생성에 앞서 선행되어야 하는 클래스의 정의

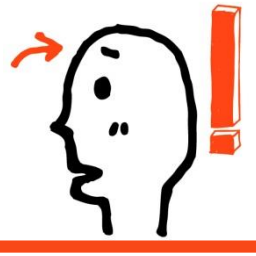
```
class FruitSeller
{
    int numOfApple=20;
    int myMoney=0;

    public int SaleApple(int money)
    {
        int num = money/1000;
        numOfApple -= num;
        myMoney += money;
        return num;
    }
}
```

변수 선언

메소드 정의

FruitSeller라는 이름의 틀 정의

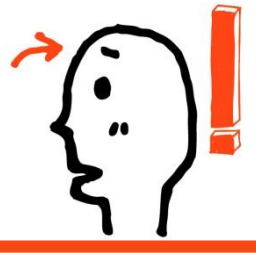
클래스와 객체, 인스턴스 개념 구별하기



- '과일장수' 클래스 정의와 키워드 final

```
class FruitSeller
{    변수 APPLE_PRICE를 상수화한다.
    final int APPLE_PRICE=1000;
    int numOfApple=20;
    int myMoney=0;

    public int saleApple(int money) 과일 판매 기능의 표현
    {
        int num=money/APPLE_PRICE;
        numOfApple-=num;
        myMoney+=money;
        return num;
    }
    public void showSaleResult()오늘 좀 많이 파셨어요? 에 대한 대답
    {
        System.out.println("남은 사과 : " + numOfApple);
        System.out.println("판매 수익 : " + myMoney);
    }
}
```

클래스와 객체, 인스턴스 개념 구별하기



▶ '나(me)' 클래스 정의

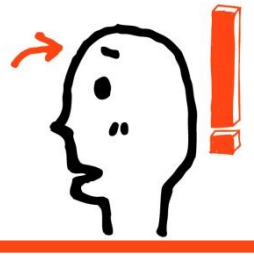
과일 구매자 관점에서의 나!

```
class FruitBuyer
{
    int myMoney=5000; 소유 현금
    int numOfApple=0; 소유 과일

    public void buyApple(FruitSeller seller, int money)
    {
        numOfApple+=seller.saleApple(money);
        myMoney-=money;
    }
    public void showBuyResult()
    {
        System.out.println("현재 잔액 : " + myMoney);
        System.out.println("사과 개수 : " + numOfApple);
    }
}
```

과일 구매행위의 표현

네가 가지고 있는 돈과 과일의 수는?

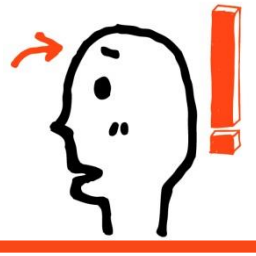
클래스와 객체, 인스턴스 개념 구별하기



인스턴스란?

- **인스턴스** : 객체 지향 프로그래밍에서 객체는 어떤 클래스를 사용해서 만들어진 것
  이때 그 객체가 메모리에 할당되어 실제 메모리를 차지하는 것
- 객체와 인스턴스는 서로 비슷한 개념
- 정확하게 구별하면 인스턴스가 객체보다는 큰 의미, 객체는 인스턴스의 한 종류

클래스와 객체, 인스턴스 개념 구별하기



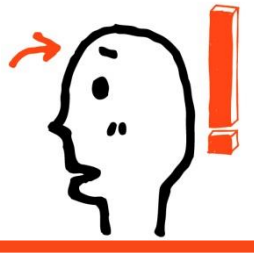
인스턴스란?

△ 그림 5-2 클래스, 객체 그리고 인스턴스

▷ 객체는 하나의 클래스로부터 여러 개가 생성될 수 있음

▷ **인스턴스화** : 클래스로부터 새로운 객체를 만드는 과정

클래스와 객체, 인스턴스 개념 구별하기



클래스를 기반으로 객체 생성하기

클래스 (틀)

```
class FruitSeller
{
    final int APPLE_PRICE=1000;
    int numOfApple=20;
    int myMoney=0;
    public int saleApple(int money)
    {
        . . . . .
    }
    public void showSaleResult( )
    {
        . . . . .
    }
}
```

인스턴스화 (instantiation)

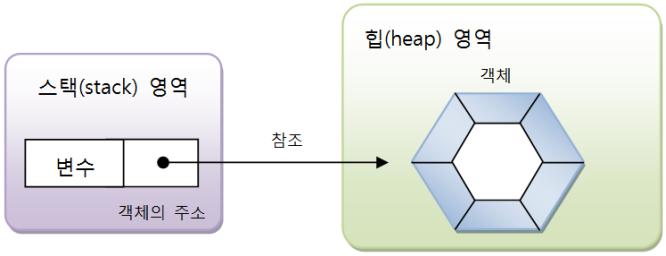
객체 (인스턴스) (실체)

```
final int APPLE_PRICE=1000;
int numOfApple=20;
int myMoney=0;
```

변수

```
public int saleApple(int money)
{
    . . . . .
}
public void showSaleResult( )
{
    . . . . .
}
```

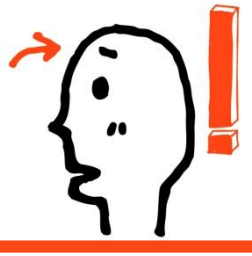
메소드



참조변수의 선언 = 인스턴스의 생성

```
FruitSeller seller = new FruitSeller();
FruitBuyer buyer = new FruitBuyer();
```

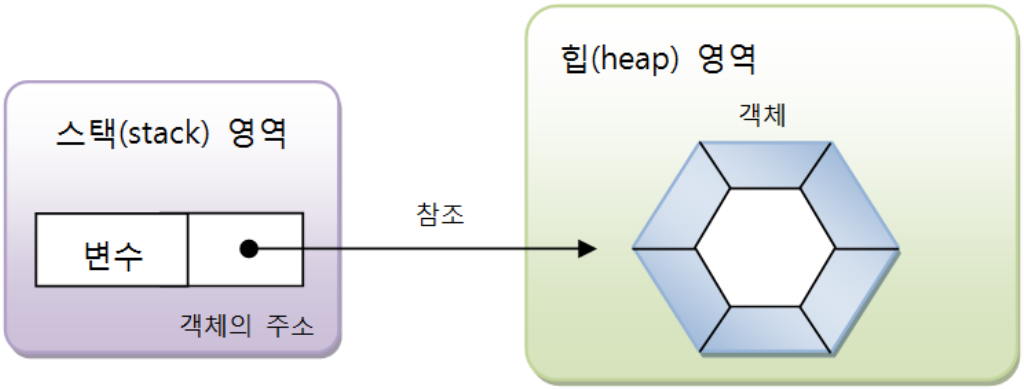
클래스와 객체, 인스턴스 개념 구별하기



클래스 참조 변수

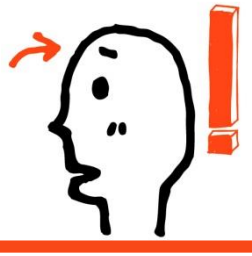
- new 연산자는 객체를 생성 후, 객체 생성 번지 리턴하고, 객체의 번지를 변수에 저장 (참조 타입 변수)
- 힙 영역의 객체를 사용하기 위해 사용

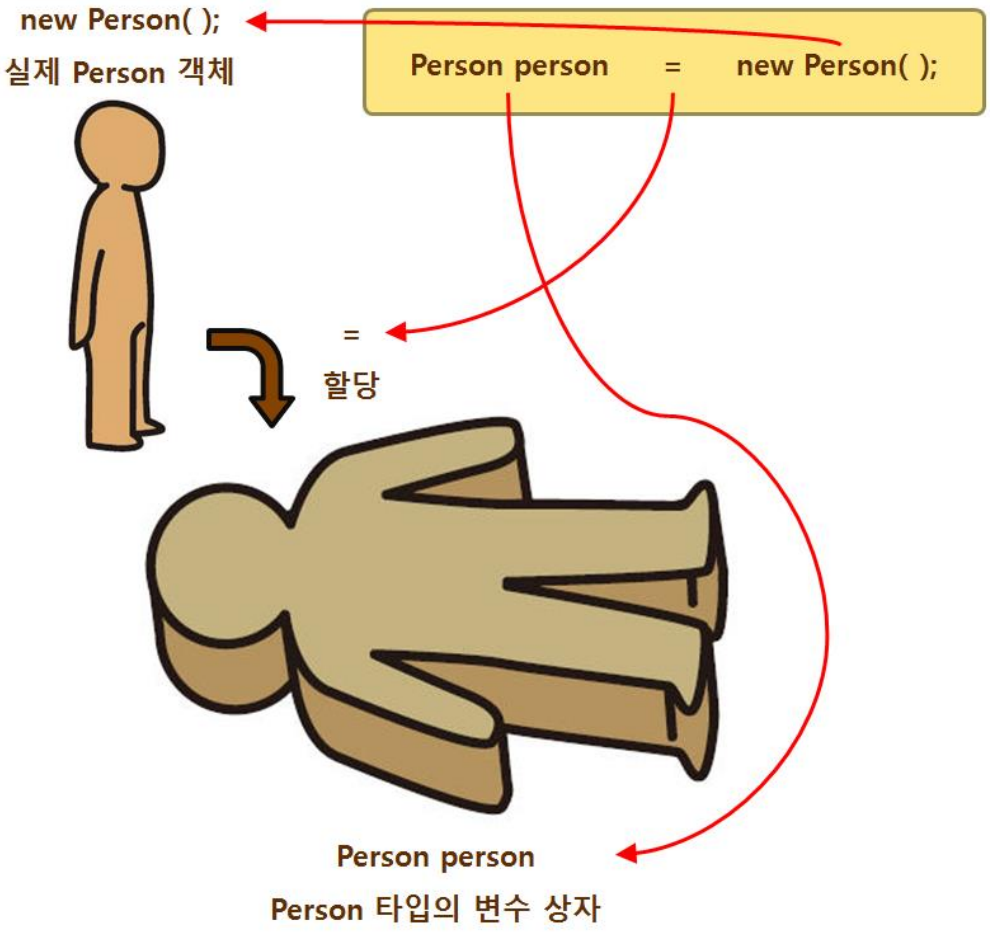
```
클래스 변수;
변수 = new 클래스();

클래스 변수 = new 클래스();
```



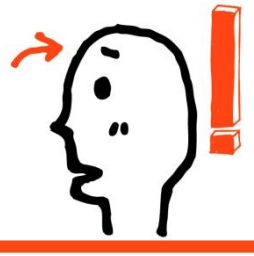
- 클래스()는 생성자를 호출하는 코드

클래스와 객체, 인스턴스 개념 구별하기

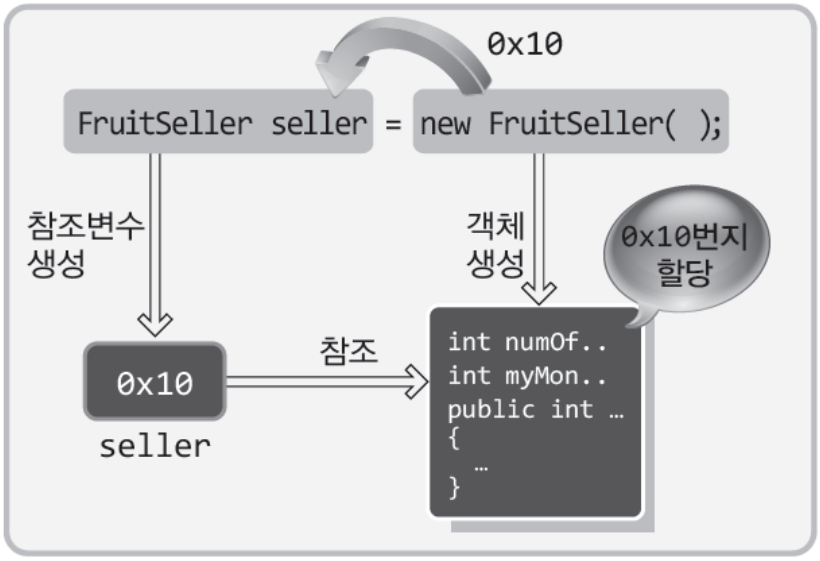




클래스와 객체, 인스턴스 개념 구별하기

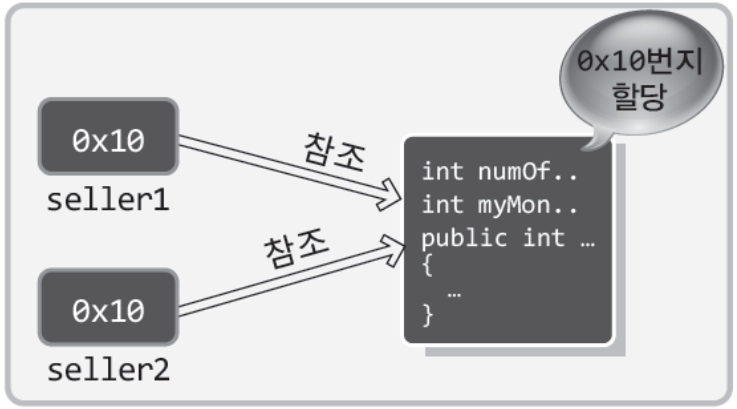


객체생성과 참조의 관계를 정확히 말하라!

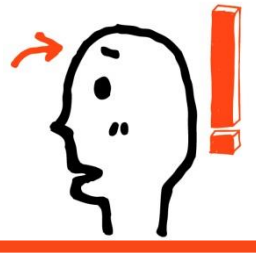


참조변수의 역할

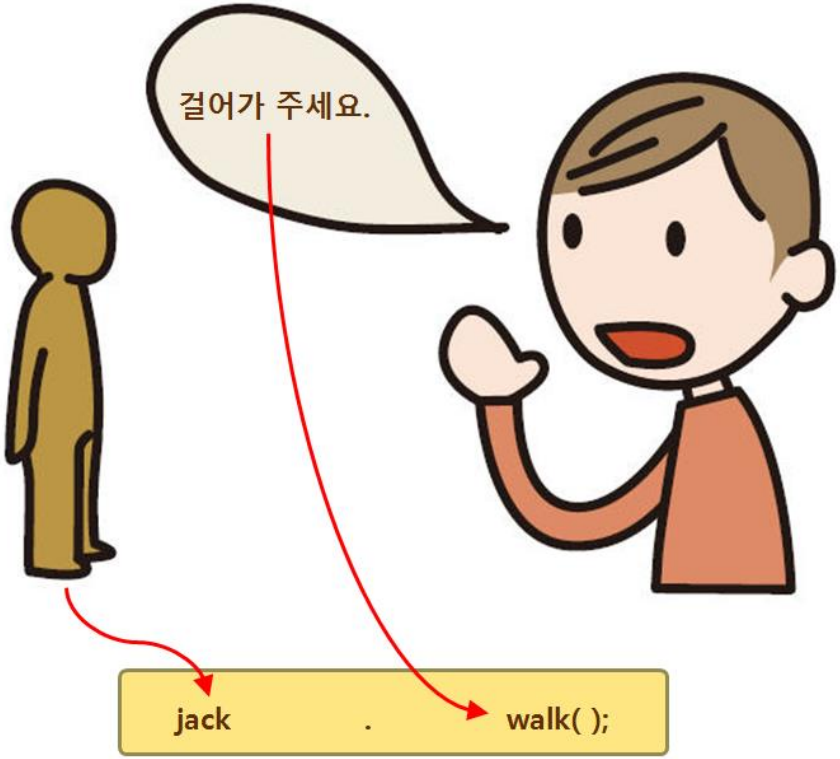
```
FruitSeller seller1 = new FruitSeller();
FruitSeller seller2 = seller1;
```



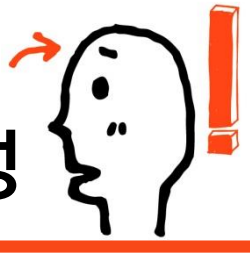
클래스와 객체, 인스턴스 개념 구별하기



- jack이라는 이름으로 된 객체에게 walk()라는 메소드로 명령을 내림



클래스를 구성하는 메소드와 생성자, 그리고 속성



생성된 개체의 접근 방법

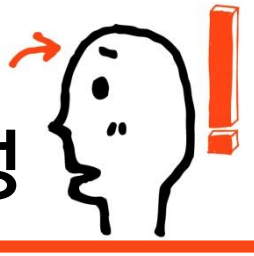
객체 내에 존재하는 변수의 접근

```
FruitSeller seller=new FruitSeller();
seller.numOfApple=20;
```

객체 내에 존재하는 메소드의 접근(호출)

```
FruitSeller seller =new FruitSeller();
seller.saleApple(10);
```

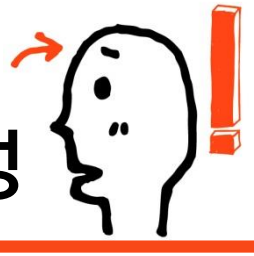
클래스를 구성하는 메소드와 생성자, 그리고 속성



```
class Person {
  String name;
  public void walk(int speed) {
    System.out.println("사람이 " + speed + "km 속도로 걸어갑니다.");
  }

  public void run(int speed) {
    System.out.println("사람이 " + speed + "km 속도로 뛰어갑니다.");
  }
}
```

```
Person person01 = new Person();
person01.name = "철수";
person01.walk(10);
```

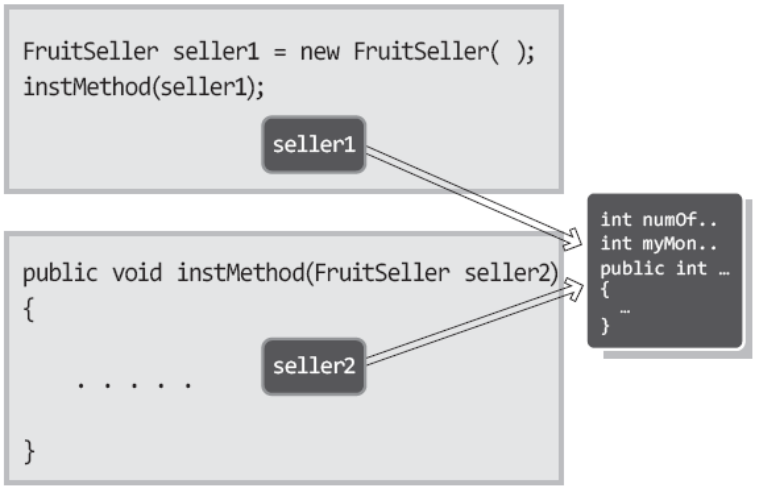
클래스를 구성하는 메소드와 생성자, 그리고 속성



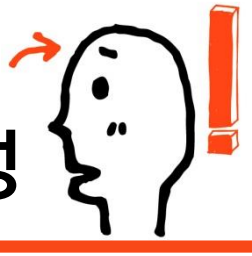
참조변수와 메소드와의 관계

```
public void myMethod()
{
    FruitSeller seller1 = new FruitSeller();
    instMethod(seller1);
    . . . .
}
```

```
public void instMethod(FruitSeller seller2)
{
    . . . . .
}
```



클래스를 구성하는 메소드와 생성자, 그리고 속성



참조변수와 메소드와의 관계

```
class Number
{
    int num=0;
    public void addNum(int n)
    {
        num+=n;
    }
    public int getNumber()
    {
        return num;
    }
}
```

인스턴스 변수라 한다.

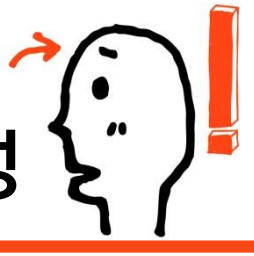
인스턴스 메소드라 한다.

실행결과

```
메소드 호출 전 : 0
메소드 호출 후 : 12
```

```
class PassInstance
{
    public static void main(String[] args)
    {
        Number nInst=new Number();
        System.out.println("메소드 호출 전 : "+nInst.getNumber());
        simpleMethod(nInst);
        System.out.println("메소드 호출 후 : "+nInst.getNumber());
    }
    public static void simpleMethod(Number numb)
    {
        numb.addNum(12);
    }
}
```

클래스를 구성하는 메소드와 생성자, 그리고 속성



사과장수 시뮬레이션 완료

```
class FruitSalesMain1
{
    public static void main(String[] args)
    {
        FruitSeller seller = new FruitSeller();
        FruitBuyer buyer = new FruitBuyer();
        buyer.buyApple(seller, 2000);   // seller에게 2000원어치 사과 구매
        System.out.println("과일 판매자의 현재 상황");
        seller.showSaleResult();

        System.out.println("과일 구매자의 현재 상황");
        buyer.showBuyResult();
    }
}
```

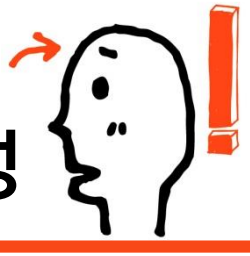
메소드 호출 → 메시지 전달(passing)

```
public void buyApple(FruitSeller seller, int money)
{
    numOfApple+=seller.saleApple(money);
    myMoney-=money;
}
```

메시지 전달은 두 객체간의 대화방법이다. 위 예제에서의 buyApple 메소드가 의미하는 바는 다음과 같다.

"아저씨 사과 2000원어치 주세요!"

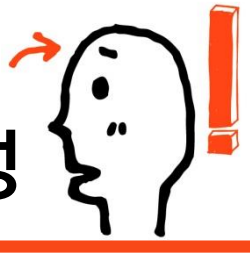
클래스를 구성하는 메소드와 생성자, 그리고 속성



객체를 생성하기 위한 특별한 메소드, 생성자

- **생성자(Constructor)** : 클래스를 객체로 만들 때 인스턴스 과정에 사용되는 메소드의 한 종류, new 키워드와 함께 호출
- **초기화** : 객체를 인스턴스하는 과정에 필요한 구문이 있는 경우에는 생성자 메소드의 실행부에 구현
- 이 과정을 거치면 생성자 안에 있는 구문들이 실행되는 동시에 객체가 JVM 위에 생성

클래스를 구성하는 메소드와 생성자, 그리고 속성



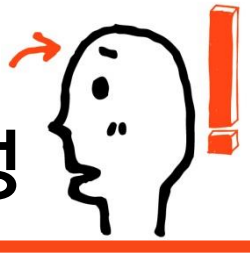
객체를 생성하기 위한 특별한 메소드, 생성자

- 클래스에 선언된 생성자는 언제, 어떻게 호출?
  → **new 키워드**를 사용

```
사용법 : [클래스 이름] [변수 이름] = new [생성자 이름]
사용예 : HelloWorld helloWorld = new HelloWorld();
```

- 자바에서 객체를 생성할 때는 new 키워드를 사용
- 이때 JVM은 해당 클래스의 생성자들 중 코드에 알맞은 생성자를 찾아 호출
- new 키워드를 사용해서 매개변수가 있는 생성자 구문을 사용하면 매개변수의 데이터형과 맞는 생성자를 실행
- 만약 선언된 생성자 중 적당한 것이 없다면 JVM은 에러를 발생

클래스를 구성하는 메소드와 생성자, 그리고 속성

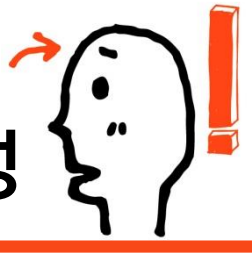


객체를 생성하기 위한 특별한 메소드, 생성자

생성자를 선언할 때 유의할 사항

- 모든 클래스는 반드시 생성자가 있어야 한다.
  선언하지 않으면 컴파일러가 자동 생성한다.
- 생성자 이름은 클래스 이름과 반드시 같아야 한다.
- 생성자는 상속되지 않는다.
- new 키워드를 사용해서 호출한다.
- 여러 개의 생성자를 만들어도 상관없으나
  매개변수의 개수와 데이터형이 중복되어서는 안 된다.

클래스를 구성하는 메소드와 생성자, 그리고 속성



두 명의 과일장수와 한 명의 구매자

서로 다른 인스턴스의 생성은, 인스턴스 변수의 초기화라는 문제를 고민하게 한다.

- 과일장수1 : 보유하고 있는 사과의 수는 30개이고, 개당 가격은 1,500원
- 과일장수2 : 보유하고 있는 사과의 수는 20개이고, 개당 가격은 1,000원

```
class FruitSeller
{
    int numOfApple;
    int myMoney;
    int APPLE_PRICE;        // 이전 예제에서는 final로 선언되었다!
    . . . . .
    public void initMembers(int money, int appleNum, int price)
    {
        myMoney=money;
        numOfApple=appleNum;
        APPLE_PRICE=price;
    }
}
```

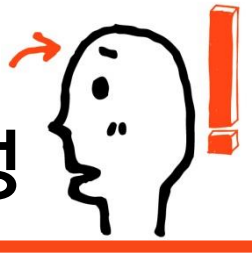
APPLE_PRICE의 값이 변경되어야 하므로 final이 빠진다.

바람직하지 않다!

- final이 빠지므로..
- 초기화 과정의 불편함..

```
public static void main(String[] args)
{
    FruitSeller seller1 = new FruitSeller();
    seller1.initMembers(0, 30, 1500);
    FruitSeller seller2 = new FruitSeller();
    seller2.initMembers(0, 20, 1000);
    FruitBuyer buyer = new FruitBuyer();
    buyer.buyApple(seller1, 4500);
    buyer.buyApple(seller2, 2000);
    . . . . . .
}
```

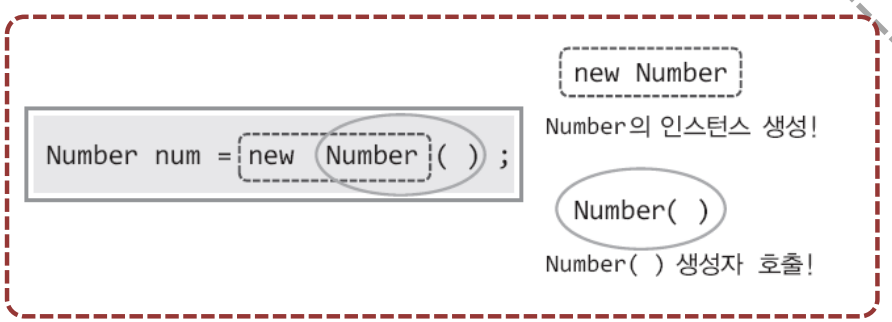
클래스를 구성하는 메소드와 생성자, 그리고 속성



두 명의 과일장수와 한 명의 구매자

```
class Number
{
    int num;
    public Number()     // 생성자!
    {
        num=10;
        System.out.println("생성자 호출!");
    }
    public int getNumber()
    {
        return num;
    }
}
```

```
public static void main(String[] args)
{
    Number num1=new Number();
    System.out.println(num1.getNumber());
    Number num2=new Number();
    System.out.println(num2.getNumber());
}
```



실행결과

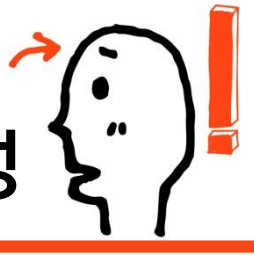
```
생성자 호출!
10
생성자 호출!
10
```

생성자의 조건

- 클래스의 이름과 동일한 이름의 메소드
- 반환형이 선언되어 있지 않으면서, 반환하지 않는 메소드

자바 인스턴스 생성시 생성자는 반드시 호출된다!

클래스를 구성하는 메소드와 생성자, 그리고 속성



값을 전달하는 생성자

```
Number num = new Number(10);
Number num = new Number(30);
```

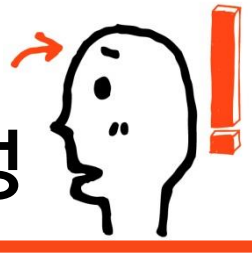
호출되는 생성자

```
public Number(int n)
{
    . . . .
}
```

```
class FruitSeller
{
    int numOfApple;
    int myMoney;
    final int APPLE_PRICE;
    public FruitSeller(int money, int appleNum, int price)
    {
        myMoney=money;
        numOfApple=appleNum;
        APPLE_PRICE=price;
    }
    . . . . . .
}
```

```
public static void main(String[] args)
{
    FruitSeller seller1 = new FruitSeller(0, 30, 1500);
    FruitSeller seller2 = new FruitSeller(0, 20, 1000);
    . . . . . .
}
```

클래스를 구성하는 메소드와 생성자, 그리고 속성



디폴트 생성자

```
FruitSeller seller = new FruitSeller();
FruitBuyer buyer = new FruitBuyer();
```

호출되는 디폴트 생성자

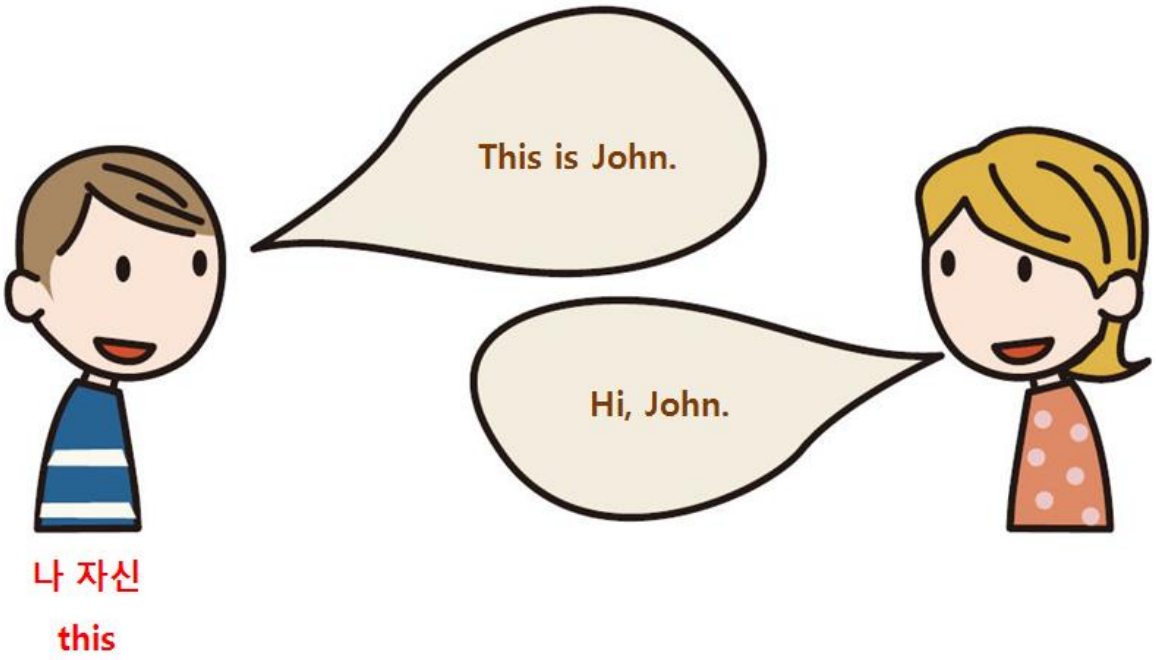
```
public FruitSeller()
{
    // 텅 비어있다!
}
```

생성자를 정의하지 않았을 때에만 삽입!

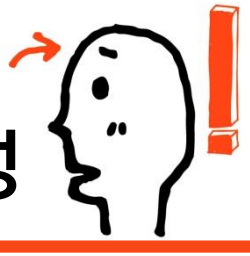
생성자가 없어도 인스턴스 생성이 가능한 이유는 자동으로 삽입되는 디폴트 생성자에 있다.

클래스를 구성하는 메소드와 생성자, 그리고 속성

this



클래스를 구성하는 메소드와 생성자, 그리고 속성

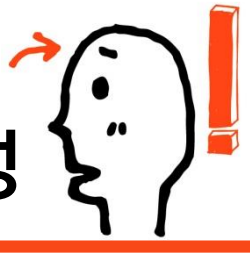


this

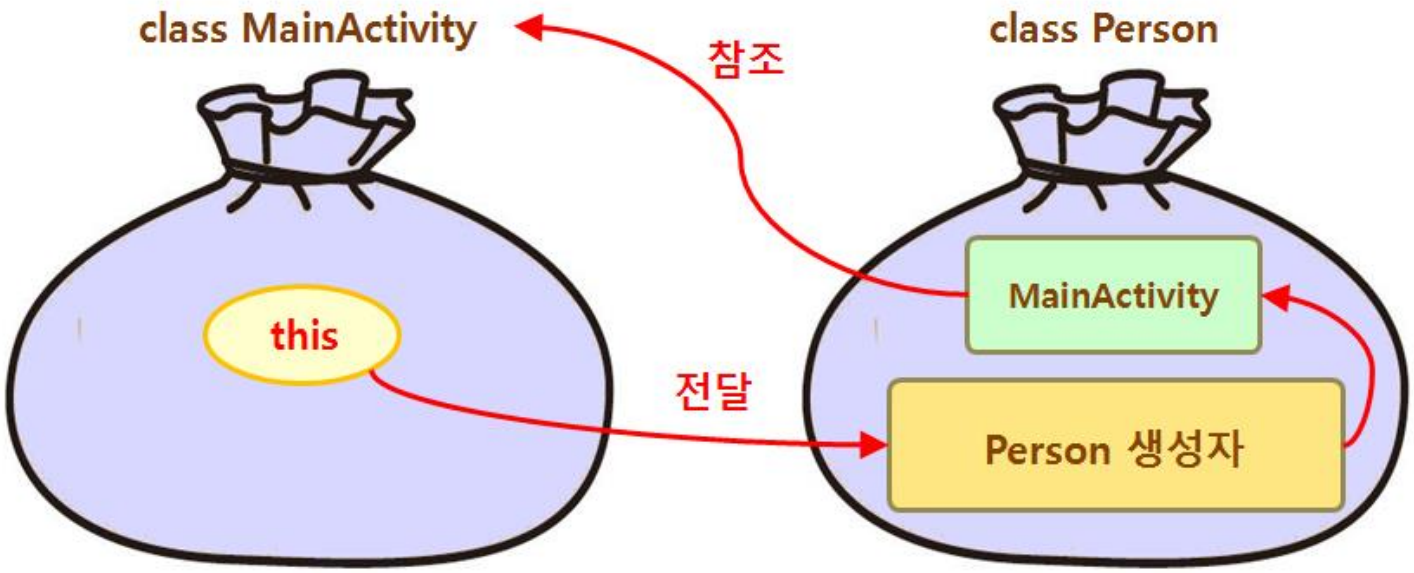
- 객체(인스턴스) 자신의 참조(번지)를 가지고 있는 키워드
- 객체 내부에서 인스턴스 멤버임을 명확히 하기 위해 this. 사용
- 매개변수와 필드명이 동일할 때 인스턴스 필드임을 명확히 하기 위해 사용

```
Car(String model) {
   this.model = model;
}
void setModel(String model) {
   this.model = model;
}
```

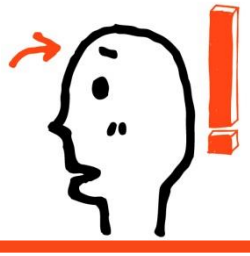
클래스를 구성하는 메소드와 생성자, 그리고 속성



this(생성자에 파라미터로 전달하기)



Self-Check(혼자 풀어보기)



구슬치기

다음 조건을 만족하는 클래스를 정의해보자

- 어린아이가 소유하고 있는 구슬의 정보를 담을 수 있다
- 놀이를 통한 구슬의 주고 받음을 표현하는 메소드가 존재한다
- 어린이의 현재 보유자산(구슬의 수)을 출력하는 메소드가 존재한다.

위의 두번째 조건은 두 아이가 구슬치기를 하는 과정에서 구슬의 잃고 얻음을 의미하는 것이다.

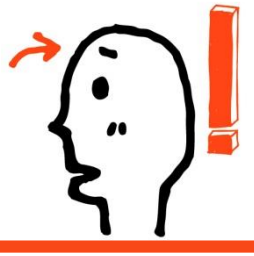
위의 조건을 만족하는 클래스를 정의하였다면, 다음 조건을 만족하는 인스턴스를 각각 생성하자.

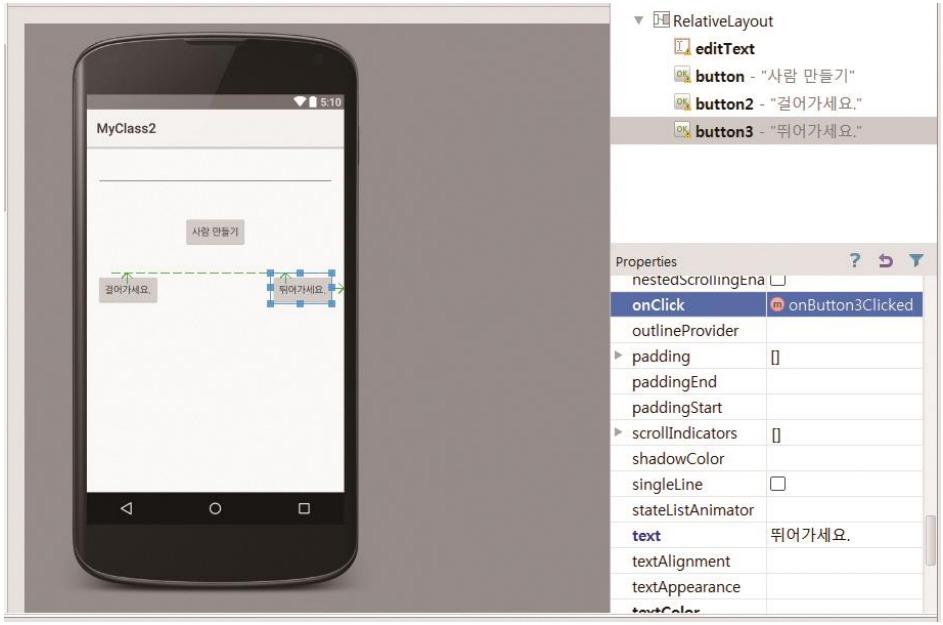
- 어린이1의 보유자산 -> 구슬 15개
- 어린이2의 보유자산 -> 구슬 9개

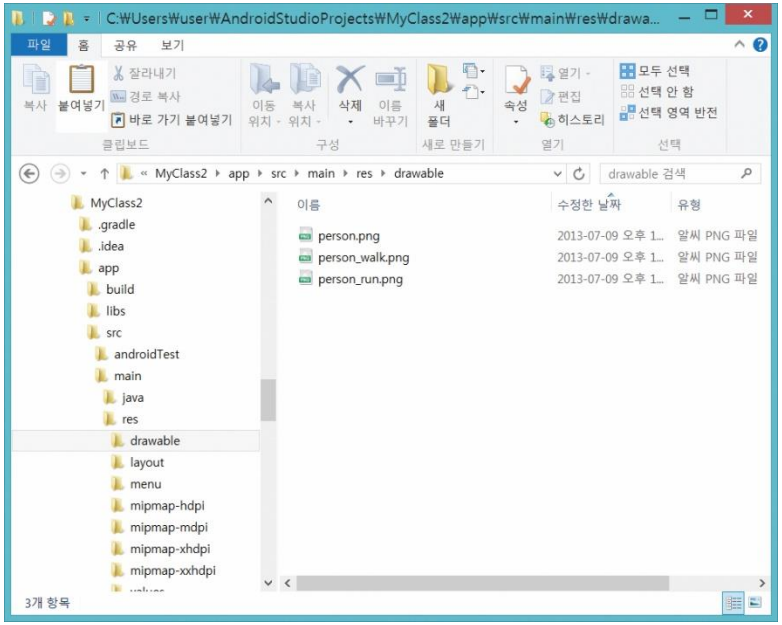
인스턴스의 생성이 완료되면 다음의 상황을 main 메소드 내에서 시뮬레이션하자

- 1차 게임에서 어린이1은 어린이2의 구슬 2개를 획득한다.
- 2차 게임에서 어린이2는 어린이1의 구슬 7개를 획득한다.

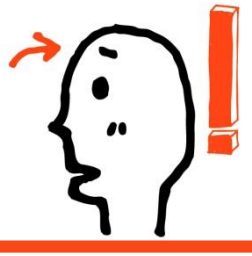
MyClass2 프로젝트 만들고 화면 구성하기







화면 레이아웃을 위한 XML에서 이미지 파일 참조하기



MyClass2 프로젝트 파일

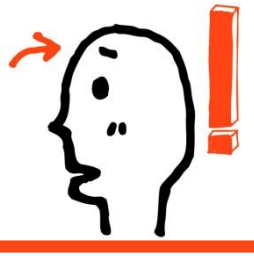
/res/drawable-xhdpi/person.png

참조

화면 레이아웃 XML (activity_main.xml)

@drawable/person

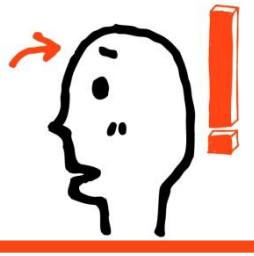
시스템의 표준 출력으로 보여주기



```
public void onButton1Clicked(View v) {
  String name = editText1.getText().toString();
  createPerson(name);
}
```

```
public void createPerson(String name) {
  person01=new Person(name, this)
}
```

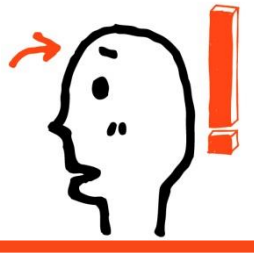
실행하여 결과 확인하기




9:59
MyClass2
john
사람 만들기
걸어가세요.
뛰어가세요.
john이(가) 10km 속도로 걸어갑니다.


10:00
MyClass2
john
사람 만들기
걸어가세요.
뛰어가세요.
john이(가) 10km 속도로 뛰어갑니다.

Self-Check



1. Singer 클래스를 만들고 그 안에 name, age 라는 속성을 포함하도록 변수를 선언합니다
2. MainActivity.java 파일을 열고 첫 번째 입력 상자에 '트와이스 ' 라는 글자를 입력합니다. 그리고 두번째 입력상자에 '25'라는 글자를 입력한 후 [설정] 버튼을 누르면 그 정보가 설정되도록 코드를 입력합니다. 이때 입력한 정보는 라디오 버튼으로 선택한 이미지뷰로 표현되는 Singer 객체에 설정합니다. 라디오 버튼이 선택되어 있는 확인하려면 라디오 버튼 객체의 isChecked 메서드를 호출합니다. isChecked 메서드에서 true를 리턴하면 선택되어 있는 것이고, false를 리턴하면 선택되어 있지 않은 것입니다.
3. 정보를 설정한 후 이미지뷰를 누르면 해당 객체에 설정한 정보가 토스트로 보이도록 합니다.