



# java 6강-static, 상속

---

양 명 속

[[now4ever7@gmail.com](mailto:now4ever7@gmail.com)]



# 목차

---

- static
  - static field
  - static method
  - static 초기화 블록
- 클래스와 배열
- 상속
  - protected
  - 단계별 상속
- Object 클래스
- 기본형 매개변수와 참조형 매개변수

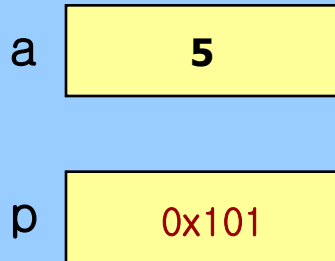


static

---

# 값타입과 참조타입의 메모리 생성 영역

## Stack 메모리 영역



```
int a;  
a=5;
```

```
Person p;  
p=new Person();
```

## Heap 메모리 영역

0x101

name	null
age	0

**display()**

인스턴스화 (객체생성)

```
class Person{  
    String name;  
    int age;  
  
    public void display(){...}  
}
```



# static – Person 클래스

```
class Person
{
    //1. 멤버변수(필드)
    private String name;
    private int age;

    //2. 생성자
    public Person(String name, int age)
    {
        this.name = name;
        this.age = age;
    }

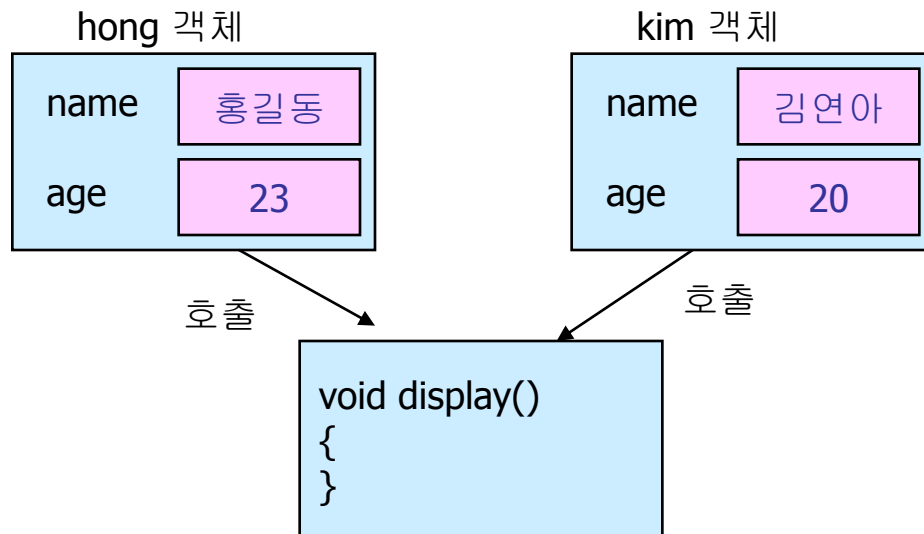
    //3. 메서드
    public void display()
    {
        System.out.println("이름 : " + name);
        System.out.println("나이 : " + age);
    }
}
```

```
이름 : 홍길동
나이 : 23
이름 : 김연아
나이 : 20
```

# Person 클래스-계속

```
class MainClass
{
    public static void main(String[] args)
    {
        Person hong = new Person("홍길동",23);
        hong.display();

        Person kim = new Person("김연아",20);
        kim.display();
    }
}
```



- 메서드는 공유
- 필드는 객체별로 따로 할당

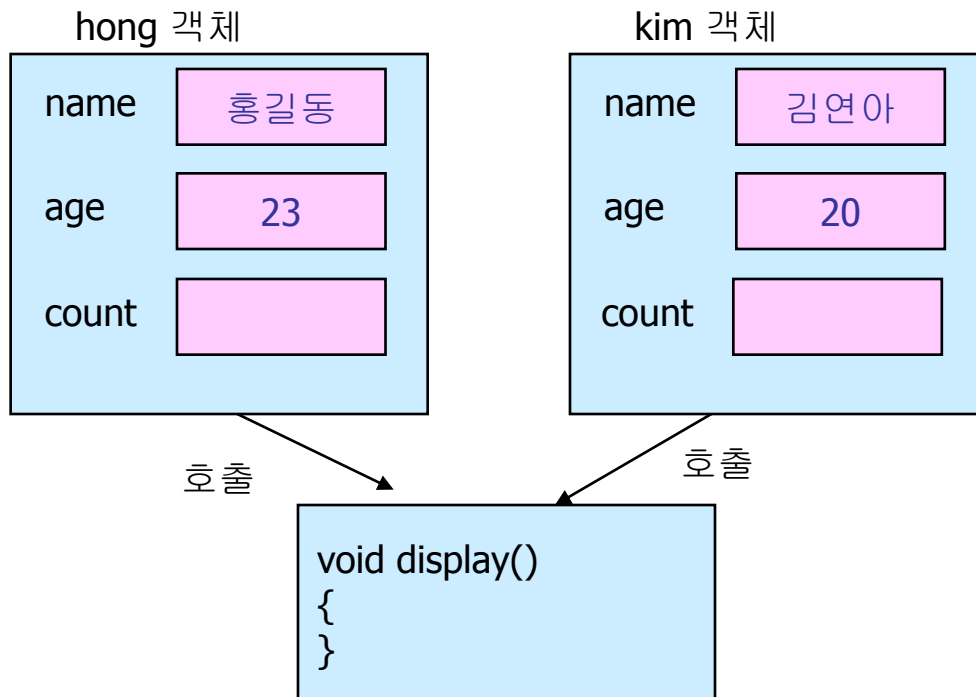


# 인스턴스 멤버

---

- 객체끼리 필드는 각자 따로 가지고 메서드는 공유함
  - 클래스의 필드들은 각 객체별로 할당되어 객체의 고유한 속성을 저장함
  - 동작을 정의하는 메서드는 모든 객체마다 공유됨
    - 같은 클래스로부터 생성된 객체들의 동작은 동일하므로 메서드를 따로 가질 필요는 없음

# count 변수 추가



- 메서드는 공유
- 필드는 객체별로 따로 할당





# 수정된 Person 클래스

```
class Person
{
    //1. 멤버필드
    private String name;
    private int age;
    //static 필드
    private static int count=0;

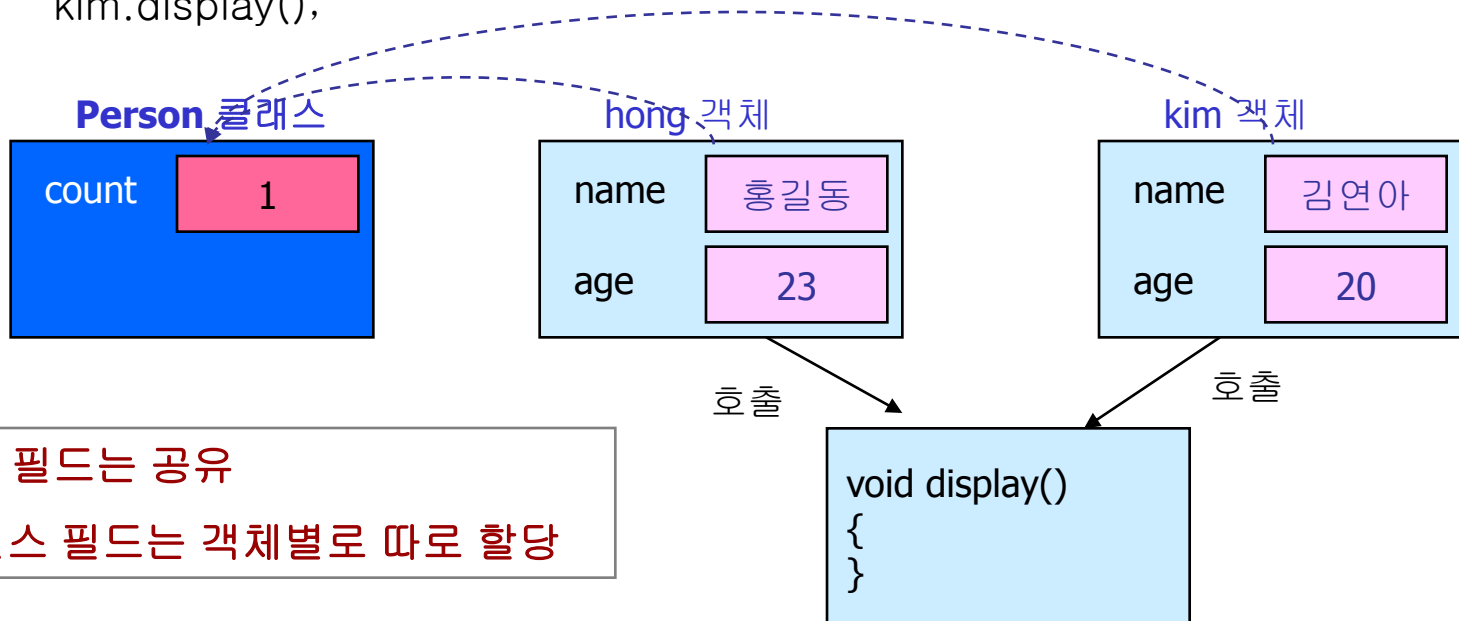
    //2. 생성자
    public Person(String name, int age)
    {
        this.name = name;
        this.age = age;
        System.out.println(++count + "번째 객체 생성!");
    }
    //3. 메서드
    public void display()
    {
        System.out.println("이름 : " + name);
        System.out.println("나이 : " + age);
    }
}
```

```
1번째 객체 생성!
이름 : 홍길동
나이 : 23
2번째 객체 생성!
이름 : 김연아
나이 : 20
```

# 수정된 Person 클래스

```
class PersonTest
{
    public static void main(String[] args)
    {
        Person hong = new Person("홍길동",23);
        hong.display();

        Person kim = new Person("김연아",20);
        kim.display();
    }
}
```



- **static** 필드는 공유
- 인스턴스 필드는 객체별로 따로 할당

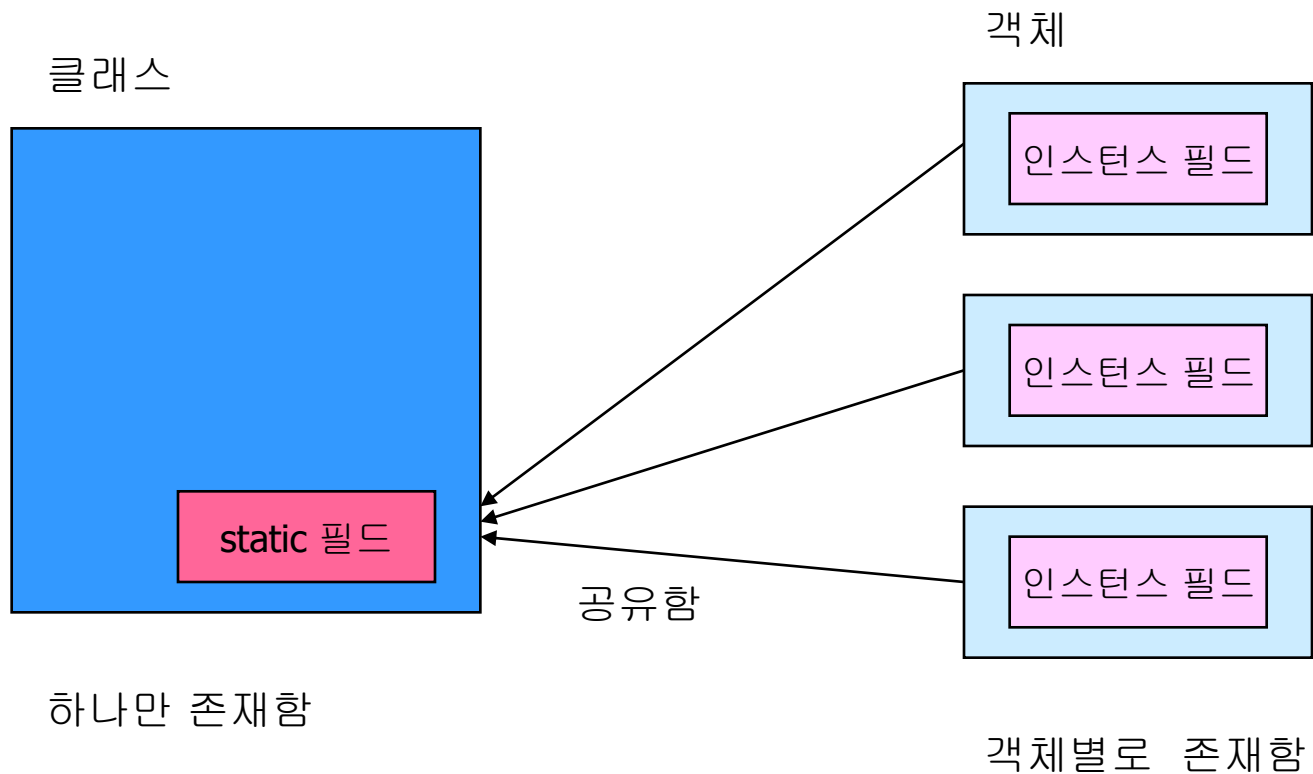


# static 변수

---

- static 변수(클래스 변수)
  - 개별 객체에 소속되지 않으며 클래스에 직접 소속됨
  - 객체가 아무리 많이 생성되어도 static 변수는 단 하나만 생성되며, 객체가 전혀 없어도 하나는 생성됨
  - 메모리에 딱 한번만 생성되며 모든 객체가 공유함
    - 어떤 객체에서 static 변수의 값을 변경하면 같은 클래스에 속한 모든 객체들이 이 변경의 영향을 받음
  - 클래스에 속한 모든 객체들이 공유해야 하는 전역적인 설정 정보나 읽기만 하는 참고 정보들이 static 변수로 선언됨
  - 각 필드 영역마다 같은 값을 가지는 필드가 있어야 할 경우, 굳이 각 영역마다 필드를 만들 필요 없이, static 변수로 선언해서 사용

# static 변수



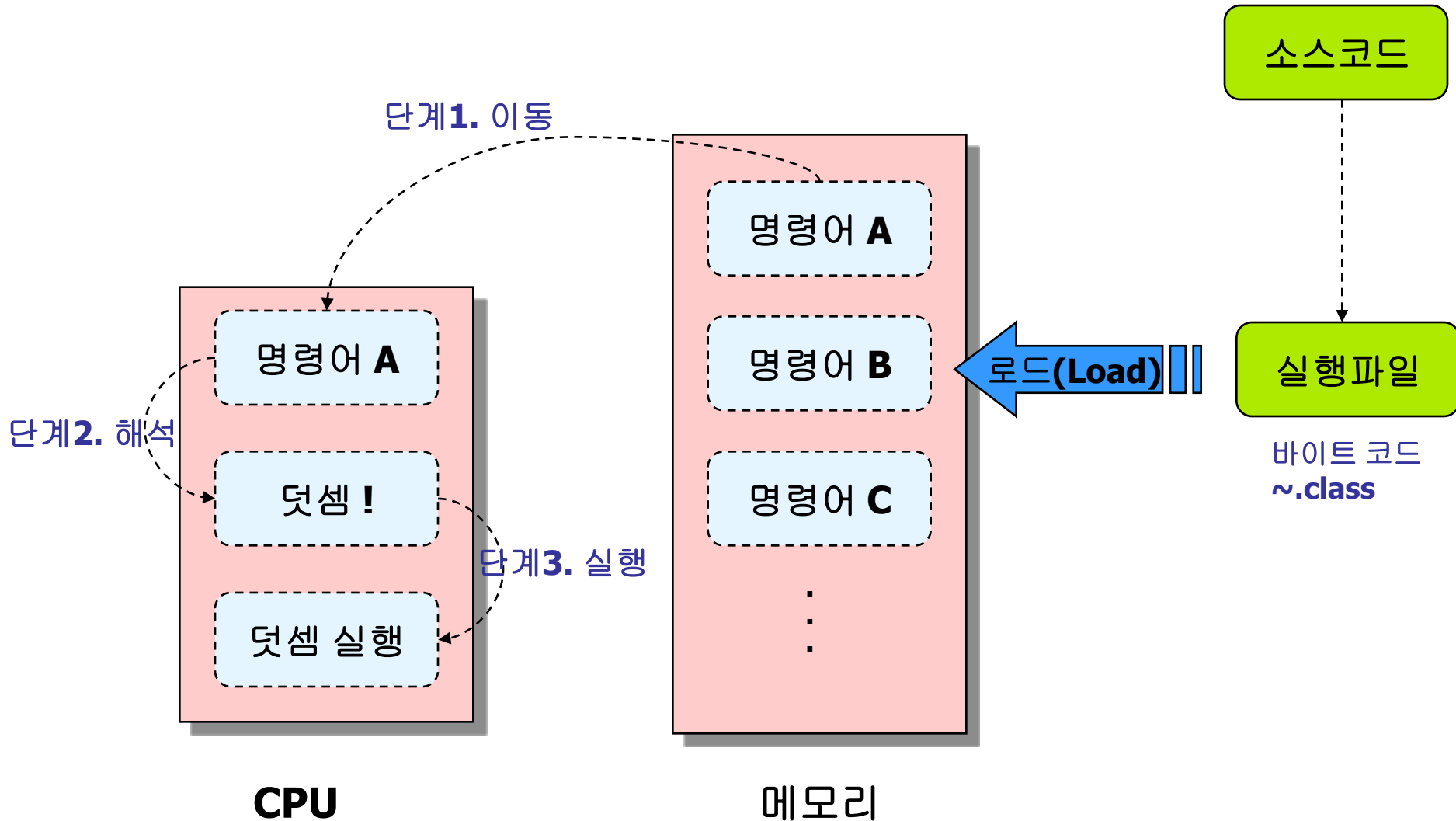


# 정적멤버 선언 – static 키워드

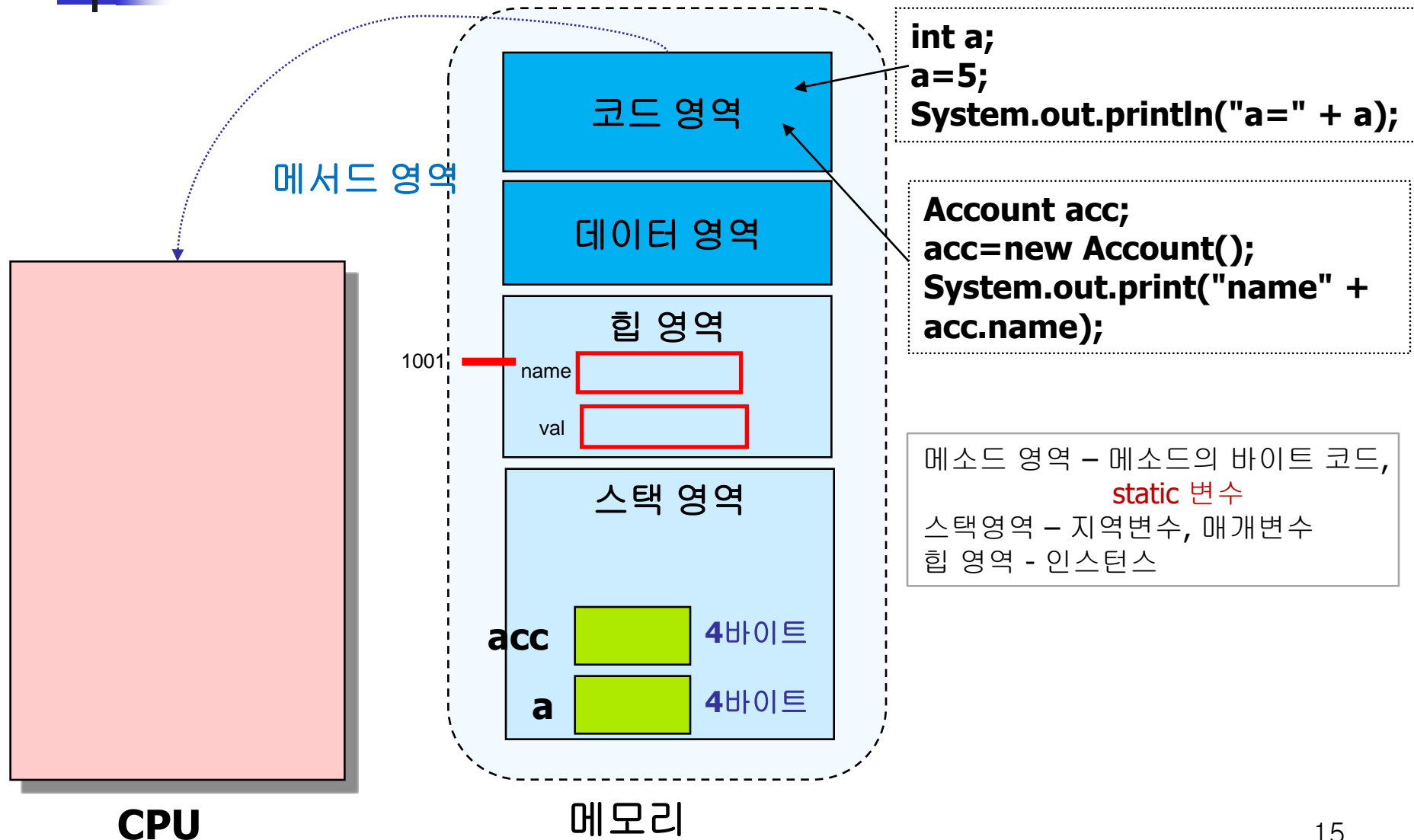
- 정적(static)멤버
  - static 키워드를 사용해서 선언하는 필드나 메서드
  - static – '**클래스의**', '**공통적인**'의 의미를 가지고 있음
- static으로 선언 – 클래스의 복사물인 인스턴스로 참조하지 않고 **클래스 차원**에서 바로 호출이 가능
  - 클래스에 대한 인스턴스를 생성하지 않아도 해당 클래스의 메서드를 호출해서 사용할 수 있음
  - **클래스의 이름으로 접근**
  - **static 변수** – 클래스가 메모리에 로드될 때 생성됨

# 프로그램의 실행과정

```
int a=5;  
a++;  
System.out.println("a=" + a);
```



# 자바 가상 머신의 메모리 모델



# static 변수의 초기화 시점

- 다른 언어로 구현된 프로그램은 컴파일이 완료되면 하나의 실행파일이 만들어짐
- 자바 프로그램은 컴파일이 완료되어도 하나의 실행파일로 만들어지지 않고, 여러 개의 클래스 파일들만 생성됨
- 하나의 실행파일로 만들어진 프로그램은 실행이 되기 위해서 실행파일 전부가 한꺼번에 메모리공간에 올라가야 함
- 자바는 필요한 만큼만(필요한 클래스 파일만-바이트 코드, ~.class) 메모리 공간에 올리는 방식으로 실행됨
- static 변수가 초기화되는 시점 - JVM에 의해서 클래스가 메모리 공간에 올라가는 순간임

예) 하나의 자바 프로그램이 총 3개의 클래스 파일로 이뤄져 있다고 가정

MainFunc.class, Client.class, Listener.class

- 프로그램의 실행을 위해 c:\ >java MainFunc
- JVM은 MainFunc.class 하나만 메모리에 올려서 프로그램을 실행함
- 이후에 Client.class가 필요한 상황을 만나면, 그 때에 가서야 Client.class 파일을 메모리에 올림

- JVM은 실행과정에서 필요한 클래스의 정보를 메모리에 로딩한다.
- 바로 이 Loading 시점에서 static 변수가 초기화된다.

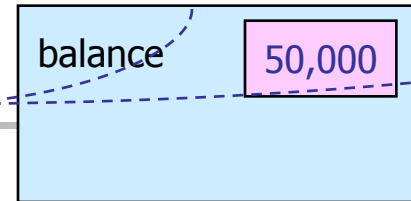


# 예제

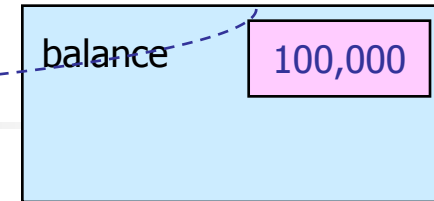
## BankAccount 클래스



## ba1 객체



## ba2 객체



```
class BankAccount
{
```

```
    private int balance ;                // 원금
    public static final double interest = 0.02 ;    // 이자율
    public static int totalBalance ;      // 각 객체의 원금의 합계
```

```
    public BankAccount(int balance)      // 생성자
    {
```

```
        this.balance = balance;
```

```
    }
```

```
    public int getBalance()              // getter/setter
    {
```

```
        return this.balance;
```

```
    }
```

```
    public void setBalance(int balance)
    {
```

```
        this.balance=balance;
```

```
    }
```

```
    public void findTotal()
    {
```

```
        totalBalance += balance;
```

```
    }
```

```
}
```

```
    public static int getTotalBalance()
    {
        return totalBalance;
    }
    public static void setTotalBalance(int totalBal)
    {
        totalBalance = totalBal;
    }
}
```



# 예제 1

계좌 1의 원금	: 50000	, 이율: 0.02
계좌 2의 원금	: 100000	, 이율: 0.02
전 계좌의 원금 합계	: 150000	

```
public class BankAccountTest
{
    public static void main(String[] args)
    {
        BankAccount ba1 = new BankAccount(50000);           // 초기값 부여
        BankAccount ba2 = new BankAccount(100000);
        ba1.findToal();
        ba2.findToal();

        System.out.println("계좌 1의 원금 : " + ba1.getBalance() + ", 이율:"
                           + BankAccount.interest);
        System.out.println("계좌 2의 원금 : " + ba2.getBalance() + ", 이율:"
                           + BankAccount.interest);
        System.out.println("전 계좌의 원금 합계 : " + BankAccount.totalBalance);
    }
}
```

```
System.out.println("모든 고객의 원금의 합계 : "+ BankAccount.getTotalBalance());
```



# static 초기화 블록

- static 초기화 블록(클래스 초기화 블록)
  - static 변수(클래스 변수)의 복잡한 초기화에 사용됨
  - 초기화 블록 내에는 메서드 내에서와 같이 조건문, 반복문, 예외처리 구문 등을 사용할 수 있으므로, 초기화 작업이 복잡하여 명시적 초기화만으로는 부족한 경우 사용
  - 클래스가 메모리에 처음 로딩될 때 한번만 수행됨
    - 클래스가 처음 로딩될 때 클래스변수들이 자동적으로 메모리에 만들어지고, 바로 클래스 초기화 블록이 클래스변수들을 초기화하게 됨

```
static{ }
```



# 예제

- 명시적 초기화를 통해 배열 **arr**을 생성
- **static** 초기화 블록을 이용해서 배열의 각 요소들을 임의의 값으로 채웠다

코드

```
class StaticBlock{  
    static int[] arr = new int[10];
```

```
    static{  
        for(int i=0;i<arr.length;i++){  
            arr[i]=(int)(Math.random()*10+1);  
        }  
    }  
}
```

```
public class StaticBlockTest {  
    public static void main(String[] args) {  
        for(int i=0;i<StaticBlock.arr.length;i++){  
            System.out.println(StaticBlock.arr[i]);  
        }  
    }  
}  
  
} //class
```

바이트코드(메서드 등 다 올라감)

static 변수부터 만들어지고 인스턴스는 나중에

static 변수는 코드가 올라갈때 만들어짐



# 멤버변수의 초기화 시기와 순서

- 멤버변수 - 인스턴스 변수, 클래스 변수(static변수)

## 클래스 변수의 초기화 시점

- 클래스가 처음 로딩될 때 단 한번 초기화 됨

## 인스턴스 변수의 초기화 시점

- 인스턴스가 생성될 때마다 각 인스턴스별로 초기화가 이루어짐

## 클래스 변수의 초기화 순서

- 기본값(default값) => 명시적 초기화 => static 초기화 블록

## 인스턴스 변수의 초기화 순서

- 기본값(default값) => 명시적 초기화 => 생성자

클래스 변수는 항상 인스턴스 변수보다 먼저 생성되고 초기화 됨

```
class AAA {
    static int cv=1; //명시적 초기화
    int iv=1; //명시적 초기화
```

static 초기화 블록

AAA.cv=2

인스턴스 초기화 블록

생성자!!

obj.iv=3

```
static{ //클래스 초기화 블록
    cv=2;
    System.out.println("static 초기화 블록");
}
```

```
{ //인스턴스 초기화 블록
    iv=2;
    System.out.println("인스턴스 초기화 블록");
}
```

```
AAA(){ //생성자
    iv=3;
    System.out.println("생성자!!");
}
```

}//class

```
class InitTest{
    public static void main(String[] args) {
        System.out.println("AAA.cv="+AAA.cv);
        AAA obj = new AAA();
        System.out.println("obj.iv="+obj.iv);
    }
}
```

클래스 초기화			인스턴스 초기화			
기본값	명시적 초기화	클래스 초기화블럭	기본값	명시적 초기화	인스턴스 초기화블럭	생성자
cv 0	cv 1	cv 2	cv 2	cv 2	cv 2	cv 2
			iv 0	iv 1	iv 2	iv 3
1	2	3	4	5	6	7



# static 메서드

- 클래스에 소속되며 개별 객체에 대한 동작이 아닌 클래스 차원의 동작을 처리함
- 특정 객체에 대한 처리를 하는 것이 아니므로 호출하는 객체에 대한 참조자 this를 전달받지 않음
- 생성된 객체가 전혀 없어도 호출할 수 있음
  - 특정 객체에 소속되지 않으므로 클래스로부터 호출해야 함
- 정적 메서드는 this가 없으므로 인스턴스 필드는 참조할 수 없음 아직 없는 애라서 인스턴스는 Static을 참조할수없음
- 클래스 소속의 static 필드만 액세스할 수 있음



# static 메서드

## ■ static 메서드

- 클래스의 객체를 여러 개 만들면 메서드는 같기 때문에 한 번만 메모리에 자리잡고, 필드는 각각 다르기 때문에 각각의 인스턴스마다 따로 생성됨
- 자주 사용하는 메서드인 경우 메서드 하나를 사용하기 위해서 매번 큰 인스턴스 전체를 메모리에 load하는 것은 비효율적
- static멤버로 선언해서 인스턴스 없이 바로 클래스 차원에서 호출해서 사용
  - 한 개의 메서드를 사용하기 위해서 객체 전체를 메모리에 로딩하는 일을 생략시켜줌
- static 메서드는 static 데이터만 접근 가능, 인스턴스변수에 접근 불가
  - static 메서드에서는 static 멤버변수만 사용가능
  - non-static 데이터는 new로 클래스를 인스턴스화 시킨 후 접근해야 함  
이걸 메모리에 올리는 작업이라한다



## 예제 1

두 수의 합	: 30
두 수의 차	: -20

```
class Calculator{
    //static 메서드
    public static int add(int a, int b){
        int res = a+b;
        return res;
    }

    //instance 메서드
    public int minus(int a, int b){
        return a-b;
    }
}

class CalculatorTest1 {
    public static void main(String[] args) {
        //static 메서드 호출 => 클래스명.메서드()
        int result = Calculator.add(10, 20);
        System.out.println("두 수의 합 : " + result);
        //int a = Integer.parseInt("123");

        //instance 메서드 호출 => 객체 생성 후 참조변수.메서드()
        //객체 생성
        Calculator cal = new Calculator();
        //메서드 호출
        int res = cal.minus(10, 30);
        System.out.println("두 수의 차 : " + res);
        /*
            Scanner sc = new Scanner(System.in); //Scanner 객체 생성
            String str = sc.nextLine(); //메서드 호출
        */
    }
}
```

## 예제2

```
public class StaticTest{
    private int num1=10; //인스턴스 변수
    private static int num2=20; //static 변수

    public int add(){ //instance 메서드
        //non-static 메서드에서는 static 필드에 바로 접근 가능
        return num1 + num2;
    }

    public static int multiply(int a, int b){ //static 메서드 - static만 접근 가능
        //int c= num1 * num2; //static에서는 static만 접근 가능하므로 에러
        int c= a*b;
        return c;
    }

    public static void main(String[] args){
        int result = multiply(10, 7); //static 메서드 호출=>클래스명.메서드() => 같은 클래스이므로
        클래스명 생략하고, 메서드()만 호출 가능
        System.out.println("곱하기:" + result);

        //static 메서드는 static 데이터만 접근 가능,
        //non-static 데이터는 new로 클래스를 인스턴스화 시킨 후 접근해야 함
        //result = add(); //에러
        StaticTest obj = new StaticTest();
        result = obj.add();
        System.out.println("더하기:"+ result);
    }
}
```



# 예제

---

- 책 판매 정보 처리(BookSales Class)
  - 필드 : 책제목, 판매수량, 단가, 판매금액
  - static 필드 : 모든 판매에 대한 총액(누적 판매금액)
  - 생성자 : 인스턴스 필드 초기화(책제목, 판매수량, 단가)
  - 메서드
    - 각 판매금액 구하는 메서드
      - 판매수량\*단가
    - 누적 판매금액 구하는 메서드 :
      - 총액(판매금액 누적) 구하기 => 총액 += 판매금액
- main()에서 사용자로부터 책제목, 판매수량, 단가를 입력 받아, 객체 생성하여 판매금액을 구하고, 총액을 구한 후
  - 화면 출력하기

# 예제

```
책제목, 수량, 단가를 입력하세요!  
c#  
3  
20000  
판매금액=60000, 누적판매금액=60000  
그만하시겠습니까?(Q)uit  
n  
책제목, 수량, 단가를 입력하세요!  
js  
5  
10000  
판매금액=50000, 누적판매금액=110000  
그만하시겠습니까?(Q)uit  
q  
계속하려면 아무 키나 누르십시오 . . .
```



# 예제

```
import java.util.Scanner;
class BookSales
{
    //멤버필드
    private String title;
    private int quantity, price, salesPrice;
    //static필드
    private static int totalSalesPrice; //총 판매금액(각 객체들의 판매금액의 합계)
    //생성자
    public BookSales(String title, int quantity, int price){
        this.title = title;
        this.quantity = quantity;
        this.price = price;
    }

    //getter/setter
    public int getSalesPrice(){
        return salesPrice;
    }
    public static int getTotalSalesPrice(){
        return totalSalesPrice;
    }

    //메서드
    public void findSalesPrice()
    {
        salesPrice = price*quantity;
    }
}
```



# 예제

---

```
    public void findTotal(){
        totalSalesPrice += salesPrice;
    }
}
//
class Ex9_1
{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        while(true) {
            System.out.println("책제목, 수량, 단가를 입력하세요!");
            String title = sc.nextLine();
            int quantity = sc.nextInt();
            int price = sc.nextInt();

            BookSales obj = new BookSales(title, quantity, price);
            obj.findSalesPrice();
            obj.findTotal();

            System.out.println("판매금액=" + obj.getSalesPrice() + ",누적판매금액="
                               + BookSales.getTotalSalesPrice());

            System.out.println("그만하시겠습니까?(Q)uit");
            sc.nextLine();
            String str = sc.nextLine();
            if (str.toUpperCase().equals("Q")) break;
        }
    }
}
```



# static 멤버

---

- 하나만 존재, 수명이 길며, 언제든지 참조할 수 있다는 점에서 static 멤버는 사실상 다른 언어의 전역함수, 전역 변수와 같은 개념
- 다만, 필요하다면 숨길 수 있고, 클래스의 범주에 논리적으로 포함된다는 점이 다름
- main() 메서드는 static 메서드
  - 객체를 만들기 전에 호출되어야 하므로 정적 메서드임

# 변수의 종류

- 클래스 영역에 선언되어 있으면 멤버변수
- 메서드 내부에 선언되어 있으면 지역변수
- 멤버변수 중 **static**이 붙은 것은 클래스 변수,  
붙지 않은 것은 인스턴스 변수

## ■ 클래스 변수(static 변수)

- 모든 객체가 공유하는 변수, 클래스차원에서 단 하나만 생성
- 인스턴스를 생성하지 않고도 언제라도 바로 사용할 수 있음
- 클래스 이름으로 접근
- 클래스가 로딩될 때 생성되어 프로그램이 종료될 때까지 유지됨
- 디폴트값으로 초기화됨

## ■ 인스턴스 변수

- 클래스 영역에서 선언된 변수
- 클래스 내의 여러 메서드에서 사용 가능, 클래스 외부에서도 접근 가능하게 할 수 있음
- 클래스의 인스턴스를 생성할 때 만들어짐
- 인스턴스마다 각기 다른 값을 가질 수 있다
- 0(숫자필드), false(논리형), null(참조형)값으로 초기화함

## ■ 지역변수(Local variables)

- 메서드 내부에서만 사용 가능한 지역변수, 메서드 내에서 선언되는 변수
- 메서드가 시작될 때 생성
- 메서드를 빠져나갈 때 사라짐

## ■ ※ 블럭변수 – 메서드내의 또 다른 블록(if, for등)내에서 선언된 변수



# 실습-할인판매를 위한 판매가격 계산하기

- 학생은 15% 할인하는 분식점에서 판매가격 계산하기 (FoodSale 클래스 정의하기)
  - 멤버변수 - 메뉴, 수량, 단가, 판매가격
  - static변수 - 할인률, **판매가격의 총합계**
  - 기능
    - 1) 판매가격을 계산하는 기능
      - 할인금액 = 수량\*단가\*할인률
      - 판매가격 = 수량\*단가-할인금액
    - 2) 판매가격의 총 합계를 구하는 기능

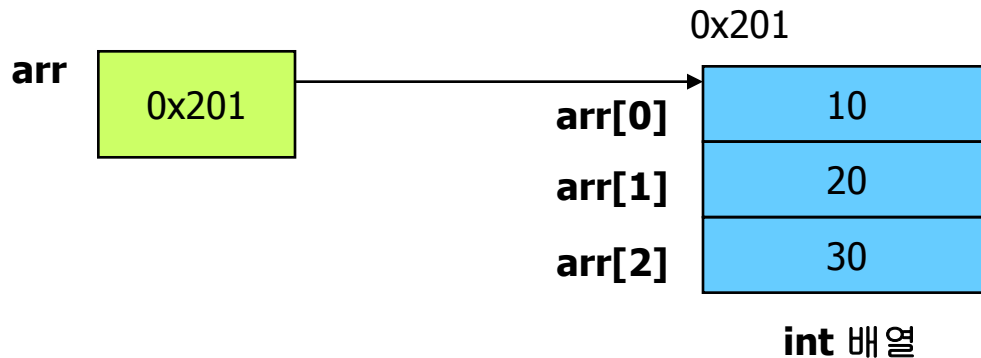
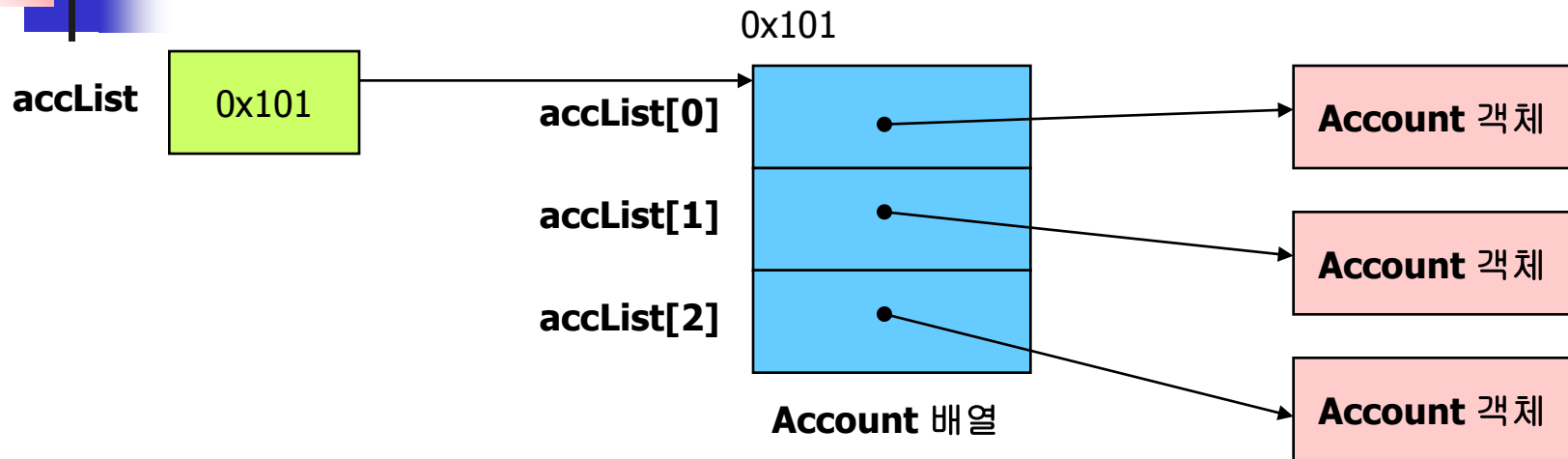
```
메뉴, 수량, 단가를 입력하세요!
김치찌개
4
5000
판매금액=₩17,000
그만하시겠습니까?(Q)uit
Q
계속하려면 아무 키나 누르십시오 . . .
```



# 클래스와 배열

---

# 객체를 배열에 저장하기



```
int[] arr = new int[3];  
Account[] accList = new Account[3];
```

# 여러 객체를 하나의 배열로 다루기

## ■ Account 객체들을 배열에 저장하기

```
===계좌번호, 잔액, 출금액을 입력하세요===  
100-1234  
10000  
1000  
===계좌번호, 잔액, 출금액을 입력하세요===  
110-4567  
20000  
7000  
===계좌번호, 잔액, 출금액을 입력하세요===  
120-7895  
30000  
5000
```

```
*****은행 고객 리스트*****  
계좌번호 : 100-1234, 잔액:9000  
계좌번호 : 110-4567, 잔액:13000  
계좌번호 : 120-7895, 잔액:25000
```

1. 배열 생성
2. For문 안에서
  - 사용자에게 입력받기(계좌번호, 잔액)
  - Account객체 생성해서 배열에 넣기
3. 화면출력

첫번째 고객의 잔액: 15000  
두번째 고객의 잔액: 19000

# 예제 1

```
public class AccountTest
{
    public static void main(String[] args)
    {
        //int[] arr=new int[3];
        //arr[0] = 7;
        Account[] accList = new Account[3];

        Account acc1 = new Account("100-1123",10000);
        Account acc2;
        acc2= new Account("110-2128",20000);
        //Account acc3 = new Account("120-7129",30000);

        accList[0]=acc1;
        accList[1]=acc2;
        accList[2]=new Account("120-7129",30000);

        accList[0].deposit(5000);
        accList[1].withdraw(1000);
        System.out.println("첫번째 고객의 잔액: "+accList[0].getBalance());
        System.out.println("두번째 고객의 잔액: "+accList[1].getBalance());
    }
}
```

```
/*
accList[0].display();
accList[1].display();
accList[2].display();
*/
for (int i=0;i<accList.length ;i++ )
{
    accList[i].display();
} //for
```

```
//메서드 호출
//acc1.deposit(5000);
```



# 예제 1

```
class Account{
    //멤버 변수
    private String acclD; //계좌번호
    private int balance; //잔액
    //생성자
    public Account(String acclD, int balance){
        this.acclD=acclD;
        this.balance=balance;
    }
    //getter/setter
    public String getAcclD(){
        return acclD;
    }
    public void setAcclD(String acclD){
        this.acclD=acclD;
    }
    public int getBalance(){
        return balance;
    }
    public void setBalance(int balance){
        this.balance=balance;
    }
}
```

```
//멤버 메서드
public void deposit(int money){//입금
    balance += money;
}
public void withdraw(int money){//출금
    balance -= money;
}
//계좌정보를 출력하는 메서드
public void display(){
    System.out.println("계좌번호 : " + acclD);
    System.out.println("잔액 : " + balance);
}
}
```



## 예제2

---

```
import java.util.Scanner;
class AccountTest2{
    public static void main(String[] args){
        Account[] accList = new Account[3];

        for (int i=0;i<accList.length ;i++ )
        {
            System.out.println("===계좌번호, 잔액, 출금액을 입력하세요===");
            Scanner sc = new Scanner(System.in);
            String accId = sc.nextLine();
            int balance = sc.nextInt();
            int money = sc.nextInt();

            accList[i]=new Account(accId, balance);
            accList[i].withdraw(money);
        }

        System.out.println("\n*****은행 고객 리스트*****");
        for (int i=0;i<accList.length ;i++ )
        {
            System.out.println("계좌번호 : " + accList[i].getAcId()+" , 잔액:" +
accList[i].getBalance());
        }
    }
}
```

# 예제-클래스내에 배열이 있는 경우

```
국어, 영어, 수학 점수를 입력하세요  
85  
97  
96  
총점=278, 평균=92.67
```

- 성적 클래스

- 필드

- 과목 배열(3과목)

- 생성자(과목 배열 초기화)

- 메서드

- 총점 구하기

- 평균 구하기

- main()에서

- 사용자로부터 국어, 영어, 수학 점수를 입력 받고, 성적 객체 생성 후 총점, 평균 구하여 화면 출력





## 예제-클래스내에 배열이 있는 경우

```
import java.util.Scanner;
class Score{
    private int[] subject;

    //생성자
    public Score(int[] subject)
    {
        this.subject = subject;
    }
    public int[] getSubject(){
        return subject;
    }
    public void setSubject(int[] subject){
        this.subject = subject;
    }
    //총점, 평균을 구하는 메서드
    public int findSum(){
        int sum=0;
        for (int i=0;i<subject.length ;i++ ){
            sum += subject[i];
        }
        return sum;
    }
    public float findAverage(){
        return findSum()/3f;
    }
}
```



# 예제-클래스내에 배열이 있는 경우

```
public class ScoreTest2{
    public static void main(String[] args){
        int[] subject = new int[3];
        System.out.println("국어, 영어, 수학 점수를 입력하세요");
        Scanner sc = new Scanner(System.in);

        for (int i=0;i<subject.length ;i++ ){
            subject[i] =sc.nextInt();
        }
        Score obj = new Score(subject);
        System.out.println("총점=" + obj.findSum() +", 평균=" + obj.findAverage());

        //-----참고 : 배열 getter/setter
        obj.setSubject(subject);
        System.out.println("\n====배열 getter/setter====");
        int[] arr = obj.getSubject();
        for (int i=0;i<arr.length ;i++ ){
            System.out.print(arr[i] +"Wt");
        }
    }
}
```



# 실습

- Student 클래스 만들기
  - 멤버변수 - 이름(name), 학번(idNo)
  - 생성자
  - 이름, 학번을 출력하는 메서드
- 1. main()에서 Student 객체를 생성하여 생성자에 의해 멤버변수에 값을 넣어주고, 출력 메서드를 호출하여 화면에 출력하기
  - 학생 1명에 대한 처리 - Student 객체 1개 생성
- 2. Student 객체를 3개 더 생성하고, 배열에 저장한다
  - 학생 3명의 정보를 배열에 저장
  - for루프 돌리면서 출력 메서드를 호출하여 저장된 Student 객체들의 정보를 출력한다.

이름:홍길동  
학번:2012001

이름:김연아  
학번:2012002

이름:유재석  
학번:2012003

이름:김유정  
학번:2012004



# 과제-전화번호 관리 프로그램

## ■ 3단계

- 배열을 이용해서, 프로그램 사용자가 입력하는 정보가 최대 100개 까지 유지되도록 변경하기
- 다음의 기능을 추가
  - 저장- 이름, 전화번호, 생년월일 정보(PhoneInfo 객체)를 배열에 저장
  - 전체조회 - 모든 사람들의 데이터를 출력한다.
  - 검색- 이름을 기준으로 데이터를 찾아서 해당 데이터의 정보를 출력해 준다
    - String 클래스의 equals()메서드 이용
  - 삭제- 이름을 기준으로 데이터를 찾아서 삭제의 과정을 진행한다
- 동명이인의 데이터가 존재하지 않는다고 가정한다
- 데이터의 삭제는 다음의 형태로 이루어진다
  - 배열의 중간에 저장된 데이터를 삭제할 경우, 해당 요소의 뒤에 저장된 요소들을 한 칸씩 앞으로 이동시키는 형태로 삭제를 진행한다.

선택하세요...  
1. 데이터 입력 (저장)  
2. 전체 데이터 조회  
3. 데이터 검색  
4. 데이터 삭제  
5. 프로그램 종료  
선택: 1  
데이터 입력을 시작합니다..  
이름: 홍길동  
전화번호: 010-100-2000  
생년월일: 88-08-19  
데이터 입력이 완료되었습니다.

선택하세요...  
1. 데이터 입력  
2. 전체 데이터 조회  
3. 데이터 검색  
4. 데이터 삭제  
5. 프로그램 종료  
선택: 1  
데이터 입력을 시작합니다..  
이름: 김연아  
전화번호: 010-300-9999  
생년월일:  
데이터 입력이 완료되었습니다.

선택하세요...  
1. 데이터 입력  
2. 전체 데이터 조회  
3. 데이터 검색  
4. 데이터 삭제  
5. 프로그램 종료  
선택: 2  
-----전체 데이터 조회-----  
name: 홍길동  
phone: 010-100-2000  
birth: 88-08-19  
  
name: 김연아  
phone: 010-300-9999  
-----

선택하세요...  
1. 데이터 입력  
2. 전체 데이터 조회  
3. 데이터 검색  
4. 데이터 삭제  
5. 프로그램 종료  
선택: 5  
프로그램을 종료합니다.

선택하세요...  
1. 데이터 입력  
2. 전체 데이터 조회  
3. 데이터 검색  
4. 데이터 삭제  
5. 프로그램 종료  
선택: 3  
데이터 검색을 시작합니다..  
이름: 홍길동  
name: 홍길동  
phone: 010-100-2000  
birth: 88-08-19  
데이터 검색이 완료되었습니다.

선택하세요...  
1. 데이터 입력  
2. 전체 데이터 조회  
3. 데이터 검색  
4. 데이터 삭제  
5. 프로그램 종료  
선택: 4  
데이터 삭제를 시작합니다..  
이름: 홍길동  
데이터 삭제가 완료되었습니다.



상속

---



# 객체지향언어의 3대 특징

---

- 1) 캡슐화(은닉성)
  - 클래스 내부에서 노출해야 되는 최소한의 부분을 제외한 나머지를 숨기는 특징
  - 필요한 기능만 노출하고 나머지를 감추는 것
  - 구성요소와 행위가 객체에 의해서 포장되어 있음



# 객체지향언어의 3대 특징

## ■ 2) 상속성

- 객체지향언어 – 클래스를 만들어 놓고, 필요할 때 객체를 생성해서 사용하기만 하면 됨, 한 번 만들어 놓으면 재사용이 용이
- 상속성 – 상위 클래스의 구성요소, 행위를 그대로 물려받아 사용하고, 자신만의 구성요소와 행위는 추가해서 사용
- 예) 사람 클래스 – 남자 클래스, 여자 클래스로 구분하여 생성
  - 남자 클래스도 사람클래스의 보다, 숨쉬다, 말하다를 똑같이 만들어야 되는 경우
    - 사람 클래스 밑에 오는 클래스는 사람 클래스의 구성요소와 행위를 그대로 불러서 사용할 수 있게 하는 것 – 상속





# 객체지향언어의 3대 특징

---

## ■ 3) 다형성

- 상속 클래스의 행위와 자신 클래스의 행위가 같긴 하지만 내용이 달라져야 하는 경우
- 같은 행위를 상속 받았지만, 방식이 다를 때는 다시 정의해서 사용하는 것
- 재 정의를 통해서 다형성을 보장해줌

# 클래스 상속(Inheritance)

## ■ 상속(Inheritance)

- 기존의 클래스를 재사용하여 새로운 클래스를 작성하는 것
- 이미 만들어진 클래스의 멤버들을 물려받아 새로운 클래스를 정의하는 기법
- 공통되는 부분을 Base 클래스로 추상화하고, 이를 상속하면서 각각의 특징을 드러낼 수 있도록 Derived 클래스를 정의함
- 하위클래스가 상위클래스를 상속 받았을 때, 하위클래스는 상위클래스의 모든 권한을 갖게 됨, 상위 클래스의 모든 것을 이용할 수 있음
  - 부모 클래스로부터 상속을 받은 자식 클래스는 부모가 가지고 있던 모든 것을 물려 받음
  - 클래스를 상속받게 되면 상위(부모) 클래스 내부의 멤버들을 가져다가 사용할 수 있음
- 상속을 하고자 한다면 `extends` 사용



# 클래스 상속(Inheritance)

## ■ 상속의 장점

- 보다 적은 양의 코드로 새로운 클래스를 작성할 수 있고 코드를 공통적으로 관리할 수 있기 때문에 코드의 추가 및 변경이 매우 용이함
- 코드의 재사용성을 높임
- 코드의 중복을 제거 => 프로그램의 생산성과 유지보수에 크게 기여함
- 상위(super) 클래스 – 기본 (base) 클래스, 부모 클래스, 조상 클래스
- 하위(sub) 클래스 – 파생 (derived) 클래스, 자식 클래스, 자손 클래스

- 자식 클래스의 멤버 개수는 부모 클래스와 같거나 많다.



# 상속

---

- 형식

```
class 클래스 이름 extends 부모 클래스
```

- 예

```
class Parent
{
    //부모가 가지고 있는 코드
}

class Child extends Parent
{
}
```

# 예제 1-상속

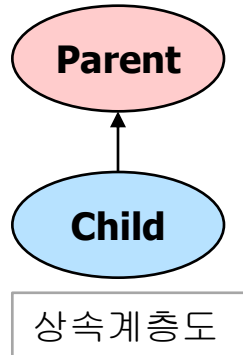
나는 자식  
나이는 7  
물려받은 유산은 10000원

class Parent

```
{  
    protected String name;  
    protected int age;  
    protected int money = 10000;  
}
```

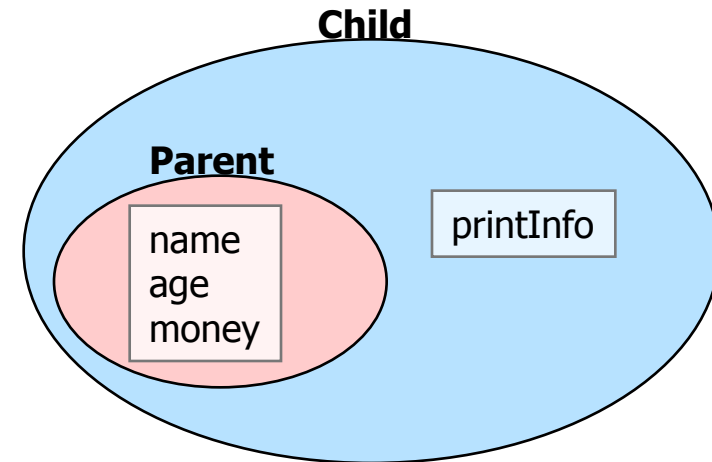
class Child **extends** Parent

```
{  
    Child()  
    {  
        this.name = "자식";  
        this.age = 7;  
    }  
    public void printInfo()  
    {  
        System.out.println("나는 "+ this.name);  
        System.out.println("나이는 "+ this.age);  
        System.out.println("물려받은 유산은 "+ this.money + "원");  
    }  
}
```

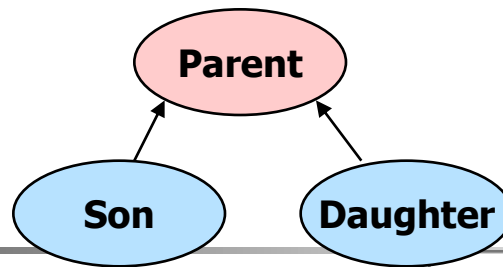


class Inheritance

```
{  
    public static void main(String[] arg)  
    {  
        Child hong = new Child();  
        hong.printInfo();  
    }  
}
```



## 예제2



```
나는 아들
나이는 ?
=====
나는 딸
나이는 10
```

```
class Parent
{
    protected String name;
    protected int age;
}

class Son extends Parent
{
    Son(){
        this.name = "아들";
        this.age = 7;
    }
    public void printInfo(){
        System.out.println("나는 "+ this.name);
        System.out.println("나이는 "+ this.age);
    }
}

class Daughter extends Parent
{
    Daughter() {
        this.name = "딸";
        this.age = 10;
    }
    public void printInfo(){
        System.out.println("나는 "+ this.name);
        System.out.println("나이는 "+ this.age);
    }
}
```

```
class Inheritance2
{
    public static void main(String[] arg)
    {
        Son s = new Son();
        s.printInfo();

        System.out.println("=====");

        Daughter d = new Daughter();
        d.printInfo();
    }
}
```

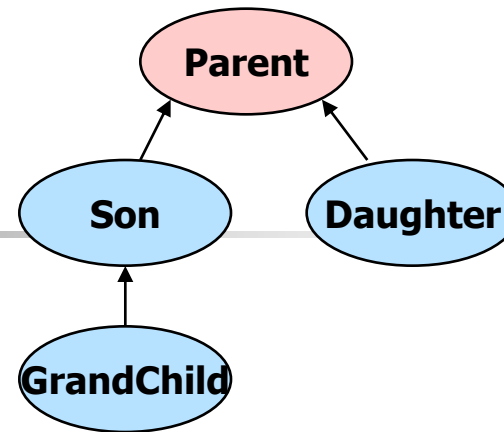
- Son과 Daughter간에는 서로 아무런 관계도 성립되지 않음
- Son과 Daughter 클래스에 공통적으로 추가되어야 하는 멤버가 있다면, 공통 부모인 Parent 클래스에 추가  
⇒ 같은 내용의 코드를 하나 이상의 클래스에 중복적으로 추가해야 하는 경우에는 상속관계를 이용해서 **코드의 중복을 최소화**해야 함

## 예제3

```
class Parent
{
    protected String name;
    protected int age;
}

class Son extends Parent
{
    Son(){
        this.name = "아들";
        this.age = 35;
    }
    public void printInfo(){
        System.out.println("나는 "+ this.name);
        System.out.println("나이는 "+ this.age);
    }
}

class GrandChild extends Son
{
    GrandChild()
    {
        this.name = "손자";
        this.age = 2;
    }
}
```



```
나는 아들
나이는 35
=====
나는 딸
나이는 30
=====
나는 손자
나이는 2
```

- Parent 클래스에 추가된 멤버변수 name, age는 Parent 클래스의 모든 자식에 추가됨
- 부모 클래스만 변경해도 모든 자식 클래스에 영향을 미치기 때문에, 클래스간의 상속관계를 맺어주면 자식 클래스들의 **공통적인 부분은 부모클래스에서 관리하고 자식 클래스는 자신에 정의된 멤버들만 관리**하면 되므로 각 클래스의 코드가 적어져서 관리가 쉬워짐

-자식 클래스의 인스턴스를 생성하면 부모 클래스의 멤버와 자식 클래스의 멤버가 합쳐진 하나의 인스턴스로 생성됨

## 예제3

```
class Daughter extends Parent
```

```
{
    Daughter(){
        this.name = "딸";
        this.age = 30;
    }
    public void printInfo(){
        System.out.println("나는 "+ this.name);
        System.out.println("나이는 "+ this.age);
    }
}
} //
class Inheritance3
{
    public static void main(String[] arg){
        Son s = new Son();
        s.printInfo();
        System.out.println("=====");

        Daughter d = new Daughter();
        d.printInfo();
        System.out.println("=====");

        GrandChild g = new GrandChild();
        g.printInfo();

    }
}
```



## 예제4

- person 클래스 정의
  - 필드 : 이름, 나이
  - getter/setter 메서드
- person 클래스를 상속받는 Student 클래스 정의
  - 필드 : 전공
  - getter/setter 메서드
- 메인 메서드
  - 사용자로부터 이름, 나이, 전공을 입력 받아 Student 객체 생성 후 화면에 출력

```
이름, 나이, 전공을 입력하세요  
홍길동  
20  
영어  
=====
```

```
이름: 홍길동  
나이: 20  
전공: 영어
```



## 예제 4

```
import java.util.*;
class Person
{
    protected String name;
    protected int age;
    public void setName(String name)
    {
        this.name = name;
    }
    public String getName()
    {
        return this.name;
    }
    public void setAge(int age)
    {
        this.age = age;
    }
    public int getAge()
    {
        return this.age;
    }
}
```

```
class Student extends Person
{
    private String major;

    public void setMajor(String major)
    {
        this.major = major;
    }
    public String getMajor()
    {
        return this.major;
    }
}
```



## 예제 4

---

```
class PersonTest{
    public static void main(String[] arg){
        System.out.println("이름, 나이, 전공을 입력하세요");
        Scanner sc=new Scanner(System.in);
        String name = sc.nextLine();
        int age = sc.nextInt();
        sc.nextLine();
        String major = sc.nextLine();
        System.out.println("=====");

        Student s = new Student();
        s.setName(name);
        s.setAge(age);
        s.setMajor(major);

        System.out.println("이름:"+s.getName());
        System.out.println("나이:"+s.getAge());
        System.out.println("전공:"+s.getMajor());
    }
}
```



# 접근 제한자

- 접근 제어자가 사용될 수 있는 곳 - 클래스, 멤버변수, 메서드, 생성자
- 1) private - 같은 클래스 내에서만 접근 가능
  - 2) default(생략형) - 같은 패키지 안에 있는 클래스들끼리만 접근 가능
  - 3) **protected** - 같은 패키지는 물론 다른 패키지일지라도  
상속 관계가 있으면 접근 가능  
(상속받은 자식 클래스에서 접근 가능)
  - 4) public : 어디서나 접근 가능

public > protected > default > private



# protected 접근자

---

## ■ protected

- 외부에 자신의 멤버를 감추고 자기의 자식 클래스에게만 멤버를 노출함
- 다른 패키지에 존재할지라도 상속관계에 놓이면 접근을 허용
- 자식에게 물려주고 싶은 멤버가 있으면 protected로 선언
- 부모의 protected 멤버필드는 실행 타임에는 default 이면서 자식에게는 완전한 public이 됨
  - 객체의 메모리를 생성한 후 점(.)찍고 접근할 때는 완벽한 default이면서 상속관계의 클래스 디자인타임에는 완벽한 public
  - 클래스 외부에서 보면 default로 보이고, 상속의 관계에서 보면 public으로 보임

## ■ private

- 자신의 자식 클래스에게도 자신의 멤버를 감춤



# 예제

---

```
class Parent
{
    private int num1;
    protected int num2;
}
class Child extends Parent
{
    public void putData()
    {
        //num1 = 10; //에러
        num2 = 20;
    }
}
class ProtectedTest1
{
    public static void main(String[] arg)
    {
        Child c = new Child();
        //c.num1 = 30; //에러
        //c.num2 = 40; //Parent 와 ProtectedTest1가 다른 패키지에 있다면 에러
        c.putData();
    }
}
```

# 예제-protected

이름 : 아버지  
이름 : 호랑이  
이름 : 아버지  
이름 : 메뚜기

```
class Father
{
    private String name;
    protected String nickname;

    public Father(){
        name ="아버지";
        nickname="호랑이";
    }
    public String getName()
    {
        return name;
    }
    public void setName(String name)
    {
        this.name =name;
    }
    public String getNickname()
    {
        return nickname;
    }
    public void setNickname(String nickname)
    {
        this.nickname = nickname;
    }
}
```



# 예제-protected

---

```
class Son extends Father
```

```
{  
    public void display()  
    {  
        //System.out.println(name); //에러 : 상위클래스의 private에 직접접근 불허  
        System.out.println(getName());  
        System.out.println(nickname + "Wn");  
    }  
}
```

```
class ProtectedTest2{
```

```
    public static void main(String[] arg)  
    {  
        Son s = new Son();  
        s.display();  
        //s.name = "아들"; //에러  
        //s.nickname = "메뚜기"; //다른 패키지라면 에러  
        s.setName("아들");  
        s.setNickname("메뚜기");  
        s.display();  
    }  
}
```





# 실습

공통되는 부분을 부모 클래스로 추상화하고, 이를 상속하면서 각각의 특징을 드러낼 수 있도록 자식 클래스 정의

- Human 클래스 (부모 클래스)
  - 이름, 나이
  - getter/setter 만들기
- Teacher 클래스 (자식 클래스)
  - 필드 : 과목
  - getter/setter 만들기
  - 메서드 : 하는 일-work()
    - 가르친다(System.out.println으로 가르친다는 내용을 화면 출력)
- Programmer (자식 클래스)
  - 필드 : 개발경력
  - getter/setter 만들기
  - 메서드 : 하는 일
    - 프로그래밍한다.
- main()에서는
  - (1) 이름, 나이, 과목을 입력 받아
    - Teacher 객체 생성 후 값을 넣어준 후, 화면 출력 (getter/setter 이용)
    - 하는 일 메서드 호출
  - (2) 이름, 나이, 개발경력을 입력 받아 처리
    - Programmer 객체 생성 후 처리



## 실습

이름, 나이, 과목을 입력하세요

홍길동

20

닷넷

=====

이름: 홍길동

나이: 20

과목: 닷넷

가르치는 일을 합니다

이름, 나이, 개발경력을 입력하세요

김연아

25

3

=====

이름: 김연아

나이: 25

개발경력: 3

프로그래밍을 합니다

## 예제 - 단계별 상속

```


GrandFather 생성자
Father 생성자
Child 생성자
GrandFather의 displayGrand()
0      1      2
Father의 displayFather()
0      1      2      3      4
Child의 displayChild()
0      1      2      3      4      5      6
    
```

```

class GrandFather
{
    public GrandFather() {
        System.out.println("GrandFather 생성자");
    }
    public void displayGrand() {
        System.out.println("GrandFather의 displayGrand()");
        for(int i=0; i<3; i++) {
            System.out.print( i + "Wt");
        }
        System.out.println();
    }
}

class Father extends GrandFather
{
    public Father() {
        System.out.println("Father 생성자");
    }

    public void displayFather() {
        System.out.println("Father의 displayFather()");
        for(int i=0; i<5; i++){
            System.out.print( i + "Wt");
        }
        System.out.println();
    }
}
    
```



## 예제-계속

```
class Child extends Father
```

```
{
```

```
    public Child(){
```

```
        System.out.println("Child 생성자");
```

```
    }
```

```
    public void displayChild() {
```

```
        System.out.println("Child의 displayChild()");
```

```
        for(int i =0; i<7; i++){
```

```
            System.out.print( i + "Wt");
```

```
        }
```

```
        System.out.println();
```

```
    }
```

```
}//
```

```
public class InheritanceTest2
```

```
{
```

```
    public static void main(String[] arg) {
```

```
        Child c = new Child();
```

```
        c.displayGrand();
```

```
        c.displayFather();
```

```
        c.displayChild();
```

```
    }
```

```
}//
```



## 상속

---

- **Father** 클래스 객체는 **GrandFather** 클래스의 것을 자신의 것처럼 호출
- **Child** 클래스는 최하위 클래스인데도 그 상위의 클래스에 속해있는 메서드를 모두 사용할 수 있음
- 생성자의 호출
  - 상속과정에서 상위레벨 클래스의 메모리가 생성되지 않는다면 자식레벨의 메모리는 생성할 수 없음
    - 메모리는 최상위클래스부터 차례대로 생성됨
    - **Father** 클래스로 객체를 만들었다면 **Father** 클래스가 상속 받은 모든 상위레벨의 생성자가 차례대로 호출 되어지고, 제일 마지막에 자신의 것이 호출됨



# 상속

---

## ■ 상속의 특징

- 생성자는 상속되지 않는다. 멤버만 상속된다.
- 생성자는 상위클래스로부터 상속되지 않고 호출됨
- 클래스는 중복상속을 할 수 없고 단일 상속만이 가능함
  - 자바에서는 중복상속(다중 상속)을 허용하지 않음
  - 2개의 클래스로부터 동시에 상속을 받을 수 없다는 뜻
- 인터페이스는 중복상속이 가능함

생성자는 상속되지 않는다. 멤버만 상속된다.



# 실습-단계별 상속

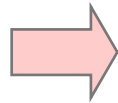
- Person – 부모 클래스
  - 이름, 나이
  - getter/setter
- Student – Person의 자식 클래스
  - 학번
  - getter/setter
  - 메서드 – study()
    - 공부한다 출력
- Graduate – Student의 자식 클래스
  - 전공
  - getter/setter
  - 메서드 – writeThesis()
    - 논문을 쓴다 출력
- main()에서 Graduate 객체 생성

```
이름, 나이, 학번, 전공을 입력하세요
홍길동
20
200917001
컴공
=====
이름: 홍길동
나이: 20
학번: 200917001
전공: 컴공
논문쓴다
```

# Object 클래스 - 모든 클래스의 조상

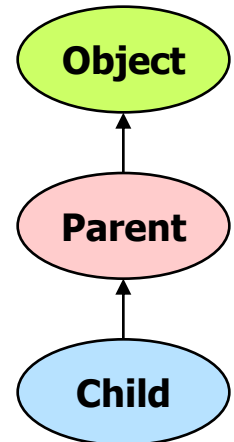
- Object 클래스 - 모든 클래스 상속계층도의 제일 위에 위치하는 조상 클래스
- 다른 클래스로부터 상속받지 않는 모든 클래스들은 자동적으로 Object 클래스로부터 상속받게 함으로써 이것을 가능하게 함

```
class Person{  
....  
}
```



```
class Person extends Object{  
....  
}
```

컴파일러는 자동적으로 `extends Object`를 추가하여 `Person` 클래스가 `Object` 클래스로부터 상속받도록 함



- 자바의 모든 클래스들은 Object 클래스의 멤버들을 상속받기 때문에 Object 클래스에 정의된 멤버들을 사용할 수 있음
  - `toString()`, `equals(Object o)` => Object 클래스에 정의된 메서드들
  - Object 클래스에는 모든 인스턴스가 가져야 할 기본적인 11개의 메서드가 정의되어 있음





## 자료형 별 디폴트 값

- 인스턴스 변수에 별도의 초기화를 진행하지 않으면, 모든 인스턴트 변수는 디폴트 값으로 초기화됨

자료형	default 값
int	0
long	0
double	0.0
boolean	false
String (or Object)	null

# 실습1

- 은행계좌정보를 담을 수 있는 Account 클래스를 상속하는 KBAccount 클래스 정의하기
- Account 클래스
  - 멤버변수 : 계좌번호, 계좌잔액
  - getter/setter
- KBAccount 클래스
  - 멤버변수 : 이체한도 추가
    - Account 클래스가 지니고 있는 멤버변수 이외에 고객별 이체한도 정보를 담고 있는 멤버변수를 지녀야 함
  - getter/setter
  - 계좌번호, 잔액, 이체한도를 화면에 출력하는 메서드
- 메인메서드
  - KBAccount 객체 생성
  - 메서드를 호출하여 계좌번호, 잔액, 이체한도를 화면에 출력

```
계좌번호, 잔액, 이체한도를 입력하세요
100-05-2456
790000
2000000
=====
계좌번호:100-05-2456
계좌잔액:790000
이체한도:2000000
```



# 기본형 매개변수와 참조형 매개변수

---



# 기본형 매개변수와 참조형 매개변수

- 자바에서는 메서드를 호출할 때 매개변수로 지정한 값을 메서드의 매개변수에 복사해서 넘겨줌
  - 매개변수의 타입이 기본형일 때는 기본형 값이 복사되겠지만, 참조형이면 인스턴스의 주소가 복사됨
  - 메서드의 매개변수를 기본형으로 선언하면 단순히 저장된 값만 얻지만, 참조형으로 선언하면 값이 저장된 곳의 주소를 알 수 있기 때문에 값을 읽어 오는 것은 물론 값을 변경하는 것도 가능

- 기본형 매개변수 - 변수의 값을 **읽기만** 할 수 있다. (read only)  
- call by value
- 참조형 매개변수 - 변수의 값을 **읽고 변경할** 수 있다. (read & write)  
- call by reference



## 예제- 기본형 매개변수

```
main() : x = 10  
change() : x = 1000  
After change(d.x)  
main() : x = 10
```

```
class Data {  
    int x;  
}
```

```
class ParameterTest {  
    public static void main(String[] args) {  
        Data d = new Data();  
        d.x = 10;  
        System.out.println("main() : x = " + d.x);  
  
        change(d.x);  
        System.out.println("After change(d.x)");  
        System.out.println("main() : x = " + d.x);  
    }  
}
```

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



## 예제- 참조형 매개변수

```
main() : x = 10  
change() : x = 1000  
After change(d)  
main() : x = 1000
```

```
class Data {  
    int x;  
}  
  
class ParameterTest2 {  
    public static void main(String[] args) {  
        Data d = new Data();  
        d.x = 10;  
        System.out.println("main() : x = " + d.x);  
  
        change(d);  
        System.out.println("After change(d)");  
        System.out.println("main() : x = " + d.x);  
    }  
  
    static void change(Data d) { // 참조형 매개변수  
        d.x = 1000;  
        System.out.println("change() : x = " + d.x);  
    }  
}
```

# 예제-참조형 매개변수(배열)

```
class ParameterTest3 {  
    public static void main(String[] args) {  
  
        int[] x = {10}; // 크기가 1인 배열. x[0] = 10;  
        System.out.println("main() : x = " + x[0]);  
  
        change(x);  
        System.out.println("After change(x)");  
        System.out.println("main() : x = " + x[0]);  
  
    }  
  
    static void change(int[] x) { // 참조형 매개변수  
        x[0] = 1000;  
        System.out.println("change() : x = " + x[0]);  
    }  
}
```

```
main() : x = 10  
change() : x = 1000  
After change(x)  
main() : x = 1000
```