

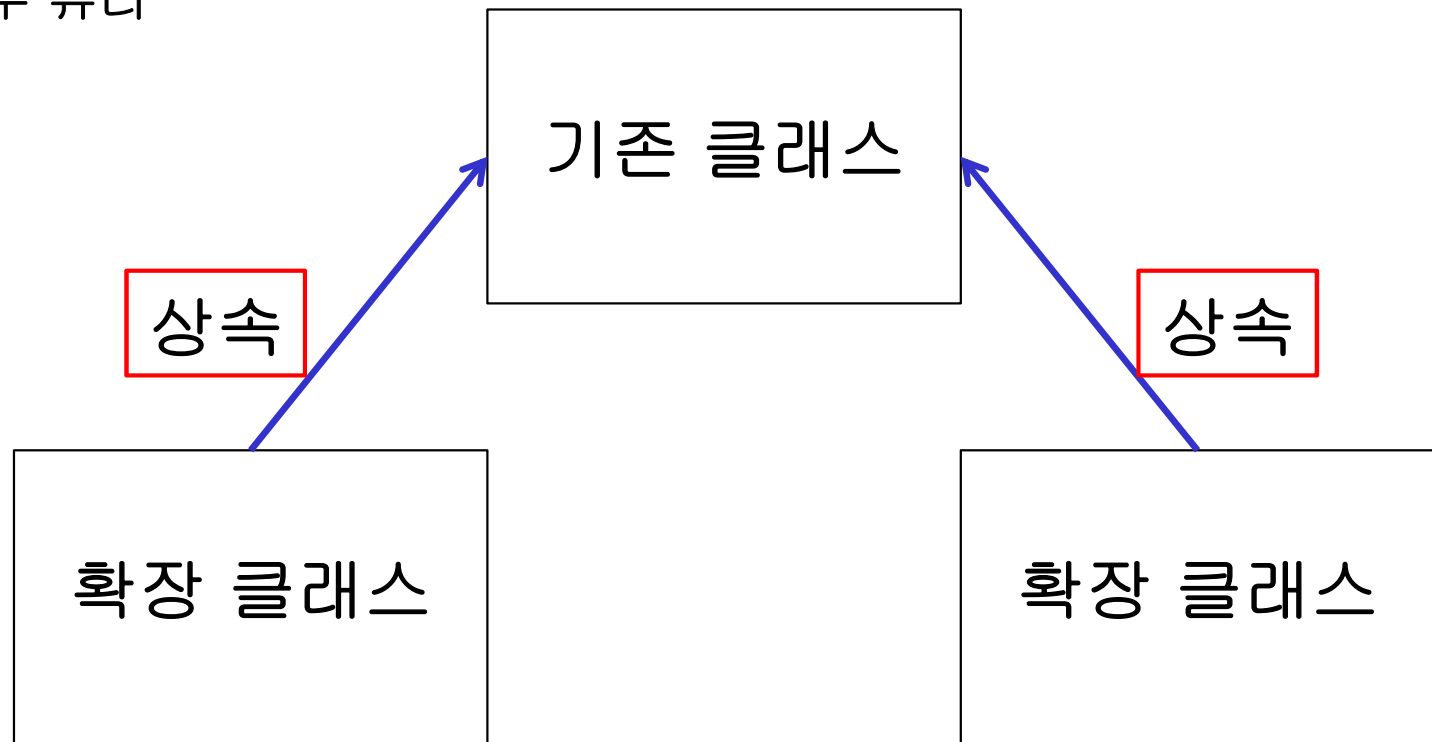
9강

# JAVA\_PROGRAMMING



# Inheritance(상속)

- ❖ OOP(객체지향)에서 중요한 개념
- ❖ 상속은 **기존의 클래스 + 추가적인 기능**
- ❖ 확장(extend) 개념
- ❖ 코드의 재 사용성과 신뢰성을 높인다.
- ❖ 생산성과 유지보수 유리



# Inheritance(상속)

- ❖ 기존의 클래스를 재사용하여 새로운 클래스를 작성하는 것

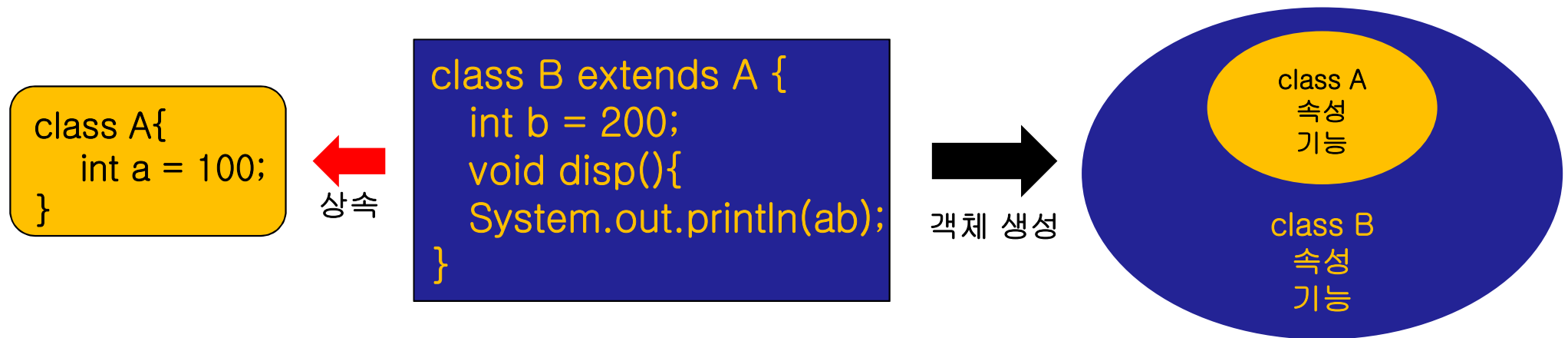
접근 제한자 [지정 예약어] class 클래스명 **extends** 상위 클래스명  
[인터페이스 구현 시 **implements** 상위 인터페이스]

```
{  
    내용..  
}
```

```
class A{  
    int a = 100;  
}  
  
class B extends A{    //A클래스를 상속하여 B클래스 정의  
    int b = 200;      //b 필드 추가  
    void disp(){      //disp()메서드 추가  
        System.out.println("x = "+a+" y = "+b);  
    }  
}  
  
public class test {  
    public static void main(String[] ar) {  
        B ob1 = new B();  
        ob1.disp();  
    }  
}
```

# 예제

- ❖ 상속 개념
- ❖ 상속한 클래스(자식 클래스)로 객체를 생성하면 부모의 내용을 모두 포함한다.
- ❖ 상속관계를 맺어주면 하위클래스들에서 사용하는 공통적인 기능은 상위 클래스에서 관리를 하도록 만들어 준다.
- ❖ 코드의 관리가 용이
- ❖ 단일 상속만 가능하다.(다이아몬드 상속 문제)



# 클래스의 관계(is-a, has-a)

❖ 클래스를 만들 때 클래스들 간에 관계에 따라 관계 형성을 고려

- ✓ is ~ a 관계가 성립된다면 상속관계가 유리
  - ~는 ~다.
- ✓ has ~ a 관계가 성립된다면 포함관계가 유리
  - ~는 ~를 가지고 있다.

예)

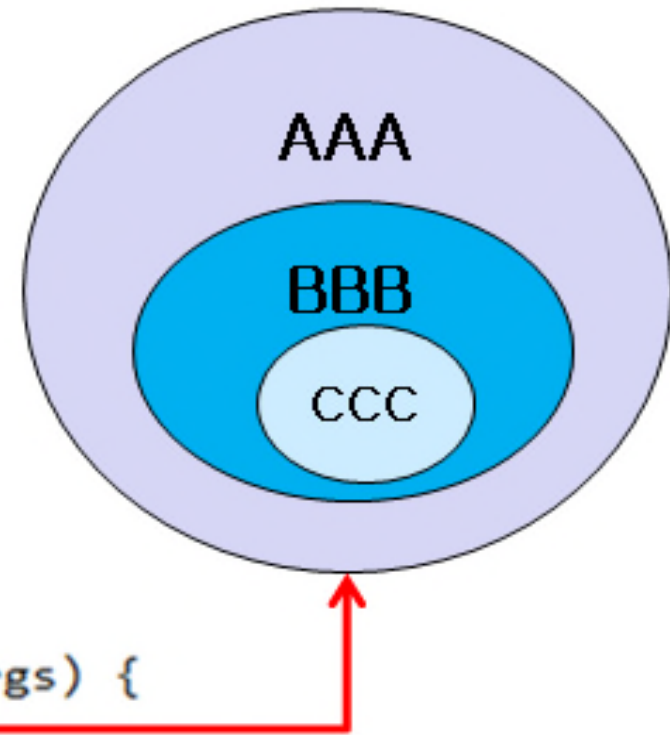
- 학생은 사람이다.(is ~ a)
- 자동차는 엔진을 가지고 있다. (has ~ a)

❖ 프로그램에서 사용할 모든 클래스를 분석하여 최대한 많은 관계를 만들어 주는 것이 코드의 재사용성, 신뢰성 등을 높인다.

# 클래스 포함( has ~ a 관계 )

- ❖ 포함 관계는 여러 번 다중으로 포함이 될 경우 효율성이 떨어진다.
- ❖ 관계성을 보고 is ~ a 관계라면 상속을 사용하는 것이 효율적이다.

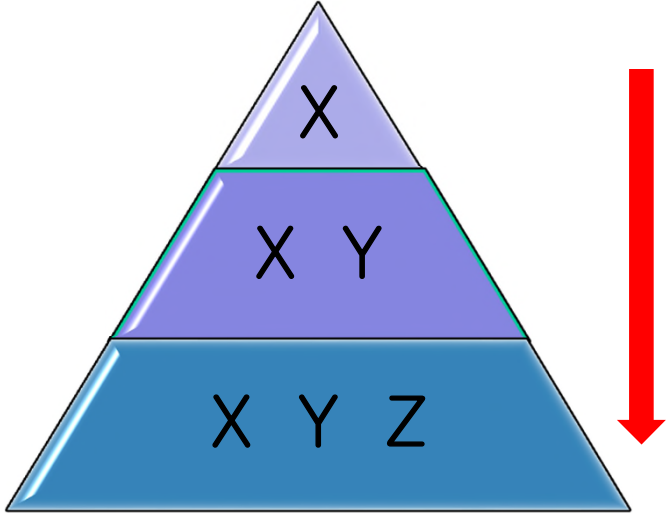
```
class AAA{
    int x;
}
class BBB{
    int y;
    AAA a = new AAA();
}
class CCC{
    int z;
    BBB b = new BBB();
}
public class test1 {
    public static void main(String[] args) {
        CCC c = new CCC();
        c.b.a.x = 10; //비효율적이다.
        System.out.println(c.b.a.x);
    }
}
```



# 상속 예제

- ❖ 상속을 받게 되면 클래스 내부의 멤버를 쉽게 참조하여 사용할 수 있다.
- ❖ 기능을 포함하는 개념이 아닌 기능이 추가된 형태이다.

```
class AAA{  
    int x;  
}  
class BBB extends AAA{  
    int y;  
}  
class CCC extends BBB{  
    int z;  
}  
  
public class test1 {  
    public static void main(String[] args) {  
        CCC c = new CCC();  
        c.x = 10; c.y = 20; c.z = 30;  
        System.out.printf("x:%d y:%d z:%d",c.x,c.y,c.z);  
    }  
}
```



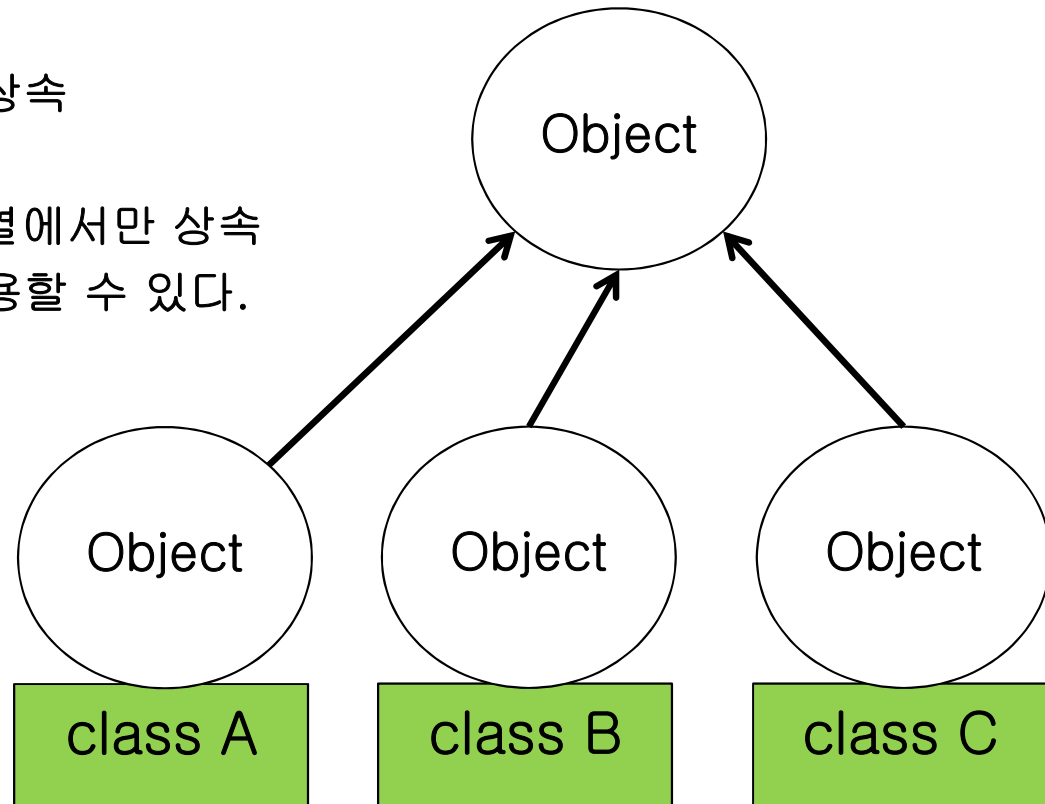
# Quiz

- ✓ 포함관계를 이해하기 위한 문제
  - ✓ has - a : 무엇 무엇을 가지고 있다.
  
- ❖ Handphone 클래스를 작성
  - 전화번호부 기능
  - 전화기능
  - 문자기능
  
- ❖ Person 클래스를 작성
  - Handphone 클래스를 포함 관계로 작성
  
- ❖ Person클래스로 객체를 여러 개 생성하여  
각각 다른 Handphone정보를 갖도록 테스트



# Object class

- ❖ 모든 클래스는 **Object** class 를 상속 받아서 사용
- ❖ java.lang.Object
  - java의 최상위 클래스 이다.
  - 모든 클래스는 Object로부터 상속
  - 컴파일러가 자동으로 추가
  - interfaces가 아니며 class 계열에서만 상속
  - Object안의 메소드를 모두 사용할 수 있다.



# Object class

## ❖ Object class

Object method	설명
protected Object clone()	객체의 복사본을 만든다.
public boolean equals(Object obj)	객체 비교(true / false)
protected void finalize()	객체 소멸 시 호출(필요 시 오버라이딩)
public Class getClass()	객체 자신의 클래스 정보를 담은 클래스를 반환
public int hashCode()	객체 자신의 해시코드 반환
public String toString()	객체 자신의 정보를 문자열로 반환
public void notify()	객체를 사용하려는 스레드 하나만 활성화
public void notifyAll()	객체를 사용하려는 스레드 모두를 활성화
public void wait()	다른 스레드가 notify() 또는 notifyAll()을 호출 할 때까지 현재 스레드를 기다리게 하는 메서드들. (무한대) (지정된 시간)
public void wait(long timeout)	
public void wait(long timeout, int nano)	

# Object class

## ❖ Object class의 멤버(자동 상속 확인)

The image shows a screenshot of an IDE with two code snippets and their corresponding member lists.

**Left Snippet (test.java):**

```
public class test{  
    public static void main(String[] args){  
        Object ob = new Object();  
        ob.  
    }  
}
```

**Left Member List (Object class):**

- equals(Object obj) : boolean - Object
- toString() : String - Object - 11 %
- getClass() : Class<?> - Object - 3 %
- equals(Object obj) : boolean - Object
- getClass() : Class<?> - Object
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- toString() : String - Object
- wait() : void - Object
- wait(long timeout) : void - Object
- wait(long timeout, int nanos) : void - Object

**Right Snippet (Tv.java):**

```
class Tv{ //class Tv extends Object 자동 추가  
    boolean power;  
    int channel;  
    int volume;  
}  
  
public class test{  
    public static void main(String[] args){  
        Object ob = new Object();  
        Tv tv = new Tv();  
        tv.  
    }  
}
```

**Right Member List (Tv class):**

- △ channel : int - Tv
- △ power : boolean - Tv
- △ volume : int - Tv
- equals(Object obj) : boolean - Object
- getClass() : Class<?> - Object
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- toString() : String - Object
- wait() : void - Object
- wait(long timeout) : void - Object
- wait(long timeout, int nanos) : void - Object

A red arrow points from the 'notifyAll()' method in the left list to the 'notifyAll()' method in the right list, indicating inheritance.

# Overriding(재정의)

❖ 부모 클래스로부터 상속받은 메서드를 자식클래스에서 재정의

❖ 조건

- 선언부는 같아야 한다.
- 접근지정자가 작아질 수 없다.
- 조상의 예외 개수보다  
자식의 예외 개수가 많을 수 없다.

➤ 오버로딩(중복 정의)

➤ 오버라이딩(재정의)

```
class Point2D{
    int x;
    int y;

    void disp(){
        System.out.println("x : " + x);
        System.out.println("y : " + y);
    }
}
class Point3D extends Point2D{
    int z;
    void disp(){ //오버라이딩(재정의)
        System.out.println("x : " + x);
        System.out.println("y : " + y);
        System.out.println("z : " + z);
    }
}
```

# Object class

## ❖ toString()

```
class A1    //extends Object 자동추가
{
    //정의된 내용 없음
}
class A2    //extends Object 자동추가
{
    public String toString(){
        return "객체의 설명을 달아 오버라이딩한 메서드";
    }
}
public class Test{
    public static void main(String[] args){
        A1 ob1 = new A1();    // A1 클래스로 객체 생성
        System.out.println(ob1.toString()); //기본 객체의 HashCode반환
        A2 ob2 = new A2();    // A2 클래스로 객체 생성
        System.out.println(ob2.toString()); //오버라이딩한 객체설명을 반환
    }
}
```

### <실행결과>

A1@39ce508a

객체의 설명을 달아 오버라이딩한 메서드

# Object class

- ❖ equals()
- ❖ 주석을 풀고 실행하면 결과는 true

```
class A1      //extends Object 자동추가
{
    int x;
    /* public boolean equals(Object o){
        A1 tmp = (A1)o;
        if(this.x == tmp.x) return true;
        else return false;
    }
    */
}

public class Test{
    public static void main(String[] args){
        A1 ob1 = new A1(); // A1 클래스로 객체 생성
        ob1.x = 10;
        A1 ob2 = new A1(); // A1 클래스로 객체 생성
        ob2.x = 10;
        System.out.println(ob1.equals(ob2)); //객체 비교
    }
}
```

<실행결과>

false



# super

## ❖ super

- 부모의 객체를 참조하는 키워드(부모 객체를 명시한다.)

```
class A{  
    int x;  
    A(){  
        x = 10;  
    }  
    A(int x) {  
        this(); //자신의 생성자  
        this.x += x; //자신을 참조  
    }  
}
```

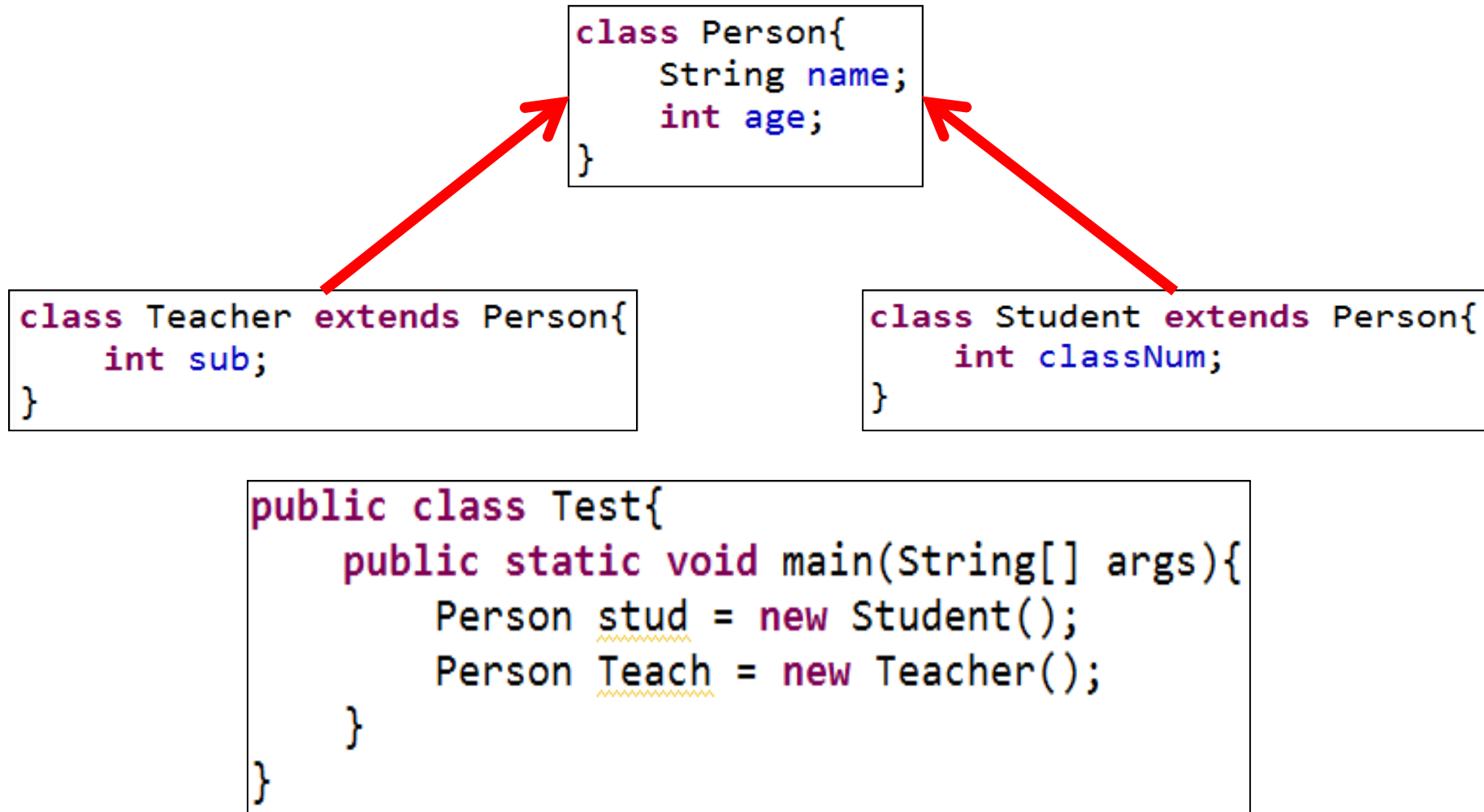


```
class B extends A{  
    int y;  
    B(){  
        this(15); //자신의 생성자  
        y = 0;  
    }  
    B(int y) {  
        super(y); //부모생성자호출  
        this.y = y; //자신을 참조  
    }  
}
```

참조 변수	생성자
호출한 객체를 참조(자신)	자신의 생성자
상속받은 객체를 참조(부모)	부모의 생성자

# 다형성

- ❖ 상속 관계에서 부모의 참조 변수로 자식의 객체를 참조 할 수 있다.

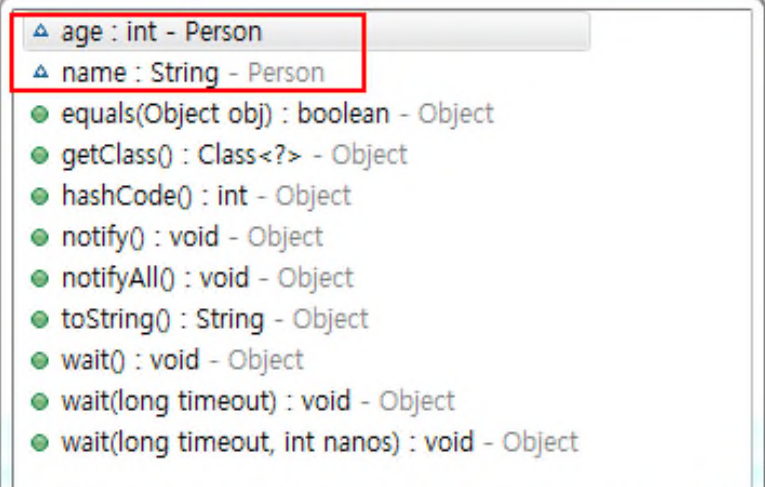




# Up-casting

❖ 부모의 참조 변수가 자식 객체를 참조하는 형태

```
class Person{
    String name;
    int age;
}
class Student extends Person{
    int classNum;
}
public class Test{
    public static void main(String[] args){
        Student ob1 = new Student();
        Person ob2 = ob1;    //업 캐스팅
        ob2.
    }
}
```



△ age : int - Person  
△ name : String - Person  
● equals(Object obj) : boolean - Object  
● getClass() : Class<?> - Object  
● hashCode() : int - Object  
● notify() : void - Object  
● notifyAll() : void - Object  
● toString() : String - Object  
● wait() : void - Object  
● wait(long timeout) : void - Object  
● wait(long timeout, int nanos) : void - Object

- ✓ 부모의 참조 변수는 자식을 참조하지만 자식의 멤버를 볼 수 없다.
- ✓ 실제 객체는 자식객체이므로 객체가 변한 것은 아니다.

# Down-casting

- ❖ 자식 참조변수에 부모가 참조하는 자식객체를 대입

```
class Person{
    String name;
    int age;
}
class Student extends Person{
    int classNum;
}
public class Test{
    public static void main(String[] args){
        Person ob1 = new Person(); //부모 객체 생성
        Person ob2 = ob1; //ob1객체 참조
Line 12 Student ob3 = (Student)ob2; //다운캐스팅

        Student ob4 = new Student(); //자식 객체 생성
        Person ob5 = ob4; //ob4객체 참조 (업캐스팅)
        Student ob6 = (Student)ob5; //다운캐스팅
    }
}
```

- ✓ 부모 참조변수가 참조하는 객체가 자식 객체일 때 다시 자식의 참조변수에 대입하는 것이 다운캐스팅
- ✓ 자식 참조변수가 부모를 참조할 때 컴파일에는 문제가 없으나 실행 시 예외

Exception in thread "main" [java.lang.ClassCastException](#): Person cannot be cast to Student  
at Test.main([Test.java:12](#))

# instanceof

- ❖ instanceof 연산자
- ❖ 참조변수가 참조하는 객체의 타입을 확인하는 기능
- ❖ boolean 값을 반환
  - 형식

참조변수 instanceof 클래스명

```
class TV{
    boolean power;
    int channel;
    int volume;
}
public class Test{
    public static void main(String[] args){
        TV tv = new TV();
        boolean bool = tv instanceof Object;
        System.out.println(bool);
    }
}
```

# instanceof

```
class Animal{ }
class Cat extends Animal{ }
class Dog extends Animal{ }
class Car{ }
class CampingCar extends Car{ }
public class Test{
    public static void main(String[] args){
        Animal animalCat = new Cat();
        Animal animalDog = new Dog();
        Car car = new Car();
        Car carCampingCar = new CampingCar();
        boolean bo1 = animalCat instanceof Animal;
        boolean bo2 = animalDog instanceof Cat;
        boolean bo3 = car instanceof CampingCar;
        boolean bo4 = carCampingCar instanceof Car;
        System.out.println("animalCat이 참조하는 객체를 Animal형으로 변환 : " + bo1);
        System.out.println("animalDog이 참조하는 객체를 Cat형으로 변환 : " + bo2);
        System.out.println("car가 참조하는 객체를 CampingCar형으로 변환 : " + bo3);
        System.out.println("carCampingCar가 참조하는 객체를 Car형으로 변환 : " + bo4);
    }
}
```

# 생략된 형태

- ❖ 자바에서 생략되어 있으나 JVM 또는 컴파일러가 자동으로 인식하는 형태

생략 형태	설명
import java.lang.*;	컴파일러 자동추가
default 생성자	오버로딩 하지 않으면 추가
this	멤버 메서드의 0번째 매개변수
toString()	객체 이름 출력 시 자동 호출
extends Object	상속을 명시하지 않은 모든 클래스에 자동상속
super()	상속관계에서 하위클래스의 생성자는 첫 라인에 super() 추가

- ❖ this()와 super() 메서드

this()	자신의 또 다른 생성자 호출
super()	상위 클래스의 생성자 호출

# abstarct

## ❖ abstract(추상화)

- 클래스의 추상적인 디자인을 위해 사용한다.
- 전체 내용을 정의하는 것이 아니라 부분적으로 추상 선언
- 추상적인 내용은 사용할 때 구체적으로 기능을 명시(오버라이드)

## ❖ abstract 메서드(메서드 자체를 디자인 했다)

- 메서드의 내용부가 정의 되지 않은 형태로 모델 개념의 메서드
- 반드시 오버라이딩 되어야 사용 가능

## ❖ abstract 클래스(abstract메서드를 포함하고 있다..)

- abstract 메서드를 포함하고 있는 클래스로 다형성 표현으로 사용
- 객체를 발생시킬 수 없는 것을 제외하면 일반 클래스와 동일

# abstract

```
abstract class A{    //추상 클래스
    int a;
    int b;
    abstract public int result();    //추상 메서드
}
class B extends A{
    public int result(){    //오버라이드
        return a + b;    //구현 내용
    }
}
class C extends A{
    public int result(){    //오버라이드
        return a * b;    //구현 내용
    }
}
public class Test{
    public static void main(String[] args){
//        A ob1 = new A();    //객체 생성 불가능
        B ob2 = new B();
        C ob3 = new C();
    }
}
```

- 추상 클래스에만 추상 메서드 정의가능
- 추상 클래스는 객체생성이 불가능
- 상속 받아서 추상 메서드를 오버라이딩(재정의) 해서 사용



# interface

- ❖ abstract 클래스의 한 종류로 포함 멤버의 제약을 가짐(순수 디자인 목적)
- ❖ 다중 상속이 가능한 유일한 클래스
- ❖ 자바에서는 기본적으로 다중상속을 비허용
- ❖ 유일하게 interface만 허용
  - public static final 멤버 필드
    - static final을 명시하지 않아도 디폴트로 지정
  - public abstract 멤버 메서드
    - public abstract를 명시하지 않아도 디폴트로 지정
  - public static inner 클래스



# interface

```
interface A{           //추상 클래스
    //인터페이스 멤버 필드
    int a = 10; //자동으로 앞에 static final이 붙는다.
    public static final int b = 20; //이것이 원형이다. 상수만 선언이 가능
    //인터페이스 멤버 메서드
    int result();      //자동으로 public abstract가 붙는다.
    public abstract void disp();    //이것이 원형이다. 추상메서드만 선언가능
}

class B implements A{   //구현으로 상속 받는다.
    public int result(){    //오버라이딩 구현
        return 10;
    }
    public void disp(){ //재정의 해야한다.
        System.out.println("상속 받아 구현한 disp()메서드");
    }
}

public class Test{
    public static void main(String[] args){
//        A ob = new A(); //객체를 생성할 수 없다. 순수 추상 클래스이므로
        B ob = new B(); //구현된 클래스로 객체 생성 가능
    }
}
```

# Quiz

- ❖ 인터페이스와 추상클래스를 활용
- ❖ 다음 클래스다이어그램 구조를 표현

- 실행 결과는 다음과 같도록 Main작성 후 테스트

지구생물:곰	지구생물:독수리
폐(으)로 호흡한다.	폐(으)로 호흡한다.
머리를 움직인다.	머리를 움직인다.
앞다리 두 개를 움직인다.	날개를 움직인다.
뒷다리 두 개를 움직인다.	두 다리를 움직인다.
지상에서 움직임	날개를 퍼덕여 날 수 있음
물에 들어가도 헤엄침	작은 동물 잡아먹음
꿀을 먹음	물똥을 쌘
육식	끼야악!소리를 낸다
똥을 쌘	
우어어소리를 낸다	

