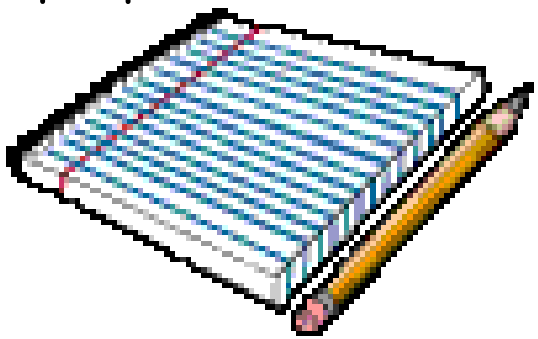
J08-11 상속

충남대학교
컴퓨터공학과



학습 내용

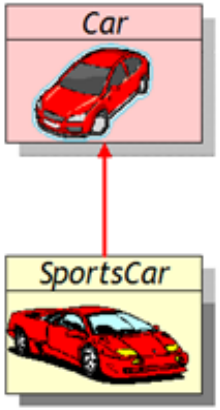
1. 상속의 개념
2. 자바에서의 상속
3. 메소드 재정의
4. 서브 클래스에서 수퍼 클래스 참조
5. 상속과 접근 제어 지시자
6. 자바 최상위 클래스 Object
7. 상속 클래스 계층 설계 및 구현

1. 상속의 개념

- 현실 세계에서의 상속

객체지향세계의 상속 ⇒ 클래스 상속

- **클래스 상속이란?**
  - 다른 클래스의 필드와 메소드를 물려받아 새로운 클래스를 정의하는 기법
  - 수퍼(super)클래스≡상위/기본(base)클래스≡부모클래스
    - 물려주는(이미 있던) 클래스
  - 서브(sub)클래스≡하위/파생(derived)클래스≡자식클래스
    - 물려받은(새로 만든) 클래스



- **상속의 필요성**
  - 객체지향 프로그래밍에서 코드 중복을 줄이는 방법
  - 이미 있는 코드를 재사용하는 방법
  - 신뢰성 있는 소프트웨어를 손쉽게 개발, 유지 보수 가능

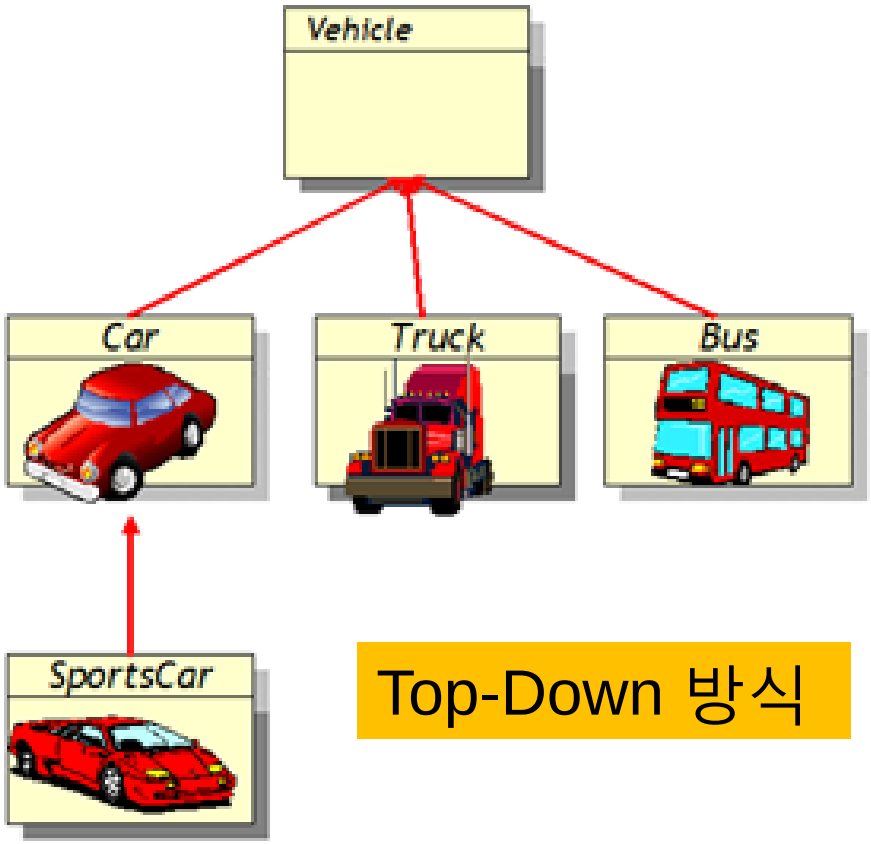
상속을 위한 기본 조건

- 상속관계에 있는 두 클래스 사이에는 IS-A 관계가 성립해야 한다.
- IS-A 관계가 성립하지 않는 경우에는 상속의 타당성을 면밀히 검토해야 한다.
- 상속의 예:

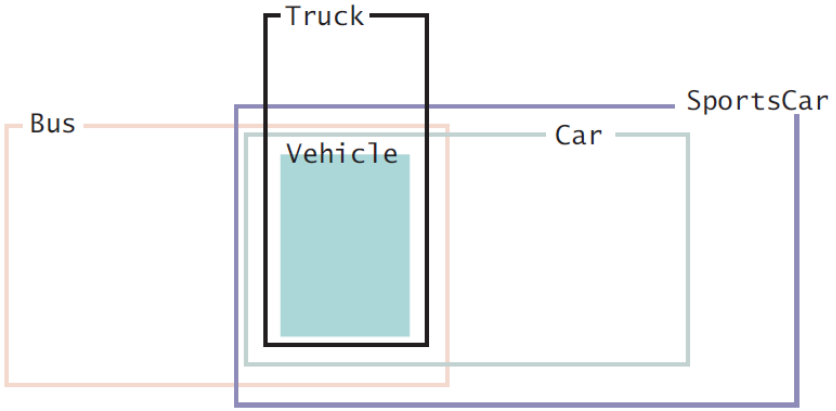
수퍼 클래스	서브 클래스
Animal	Lion, Dog,
Vehicle	Car, Bus, Truck, Boat, Motorcycle, Bicycle
Shape	Rectangle, Triangle, Circle

IS-A 관계

상속 계층 만들기

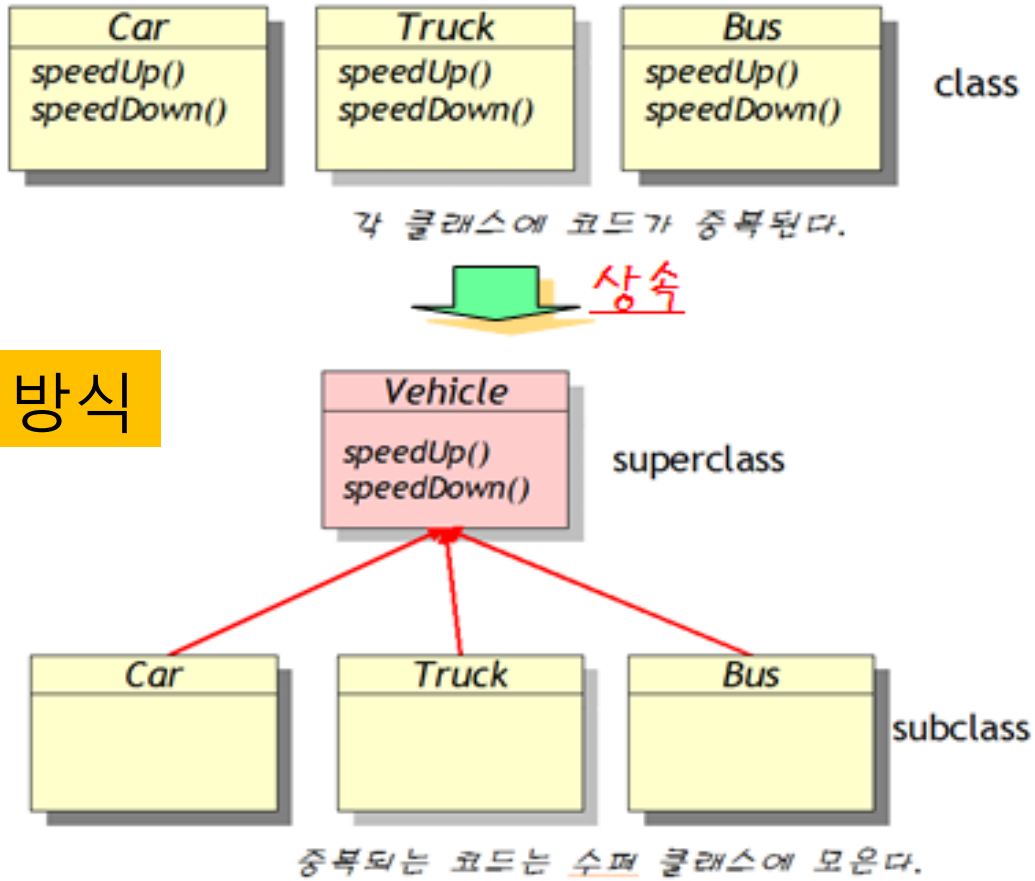


Top-Down 방식



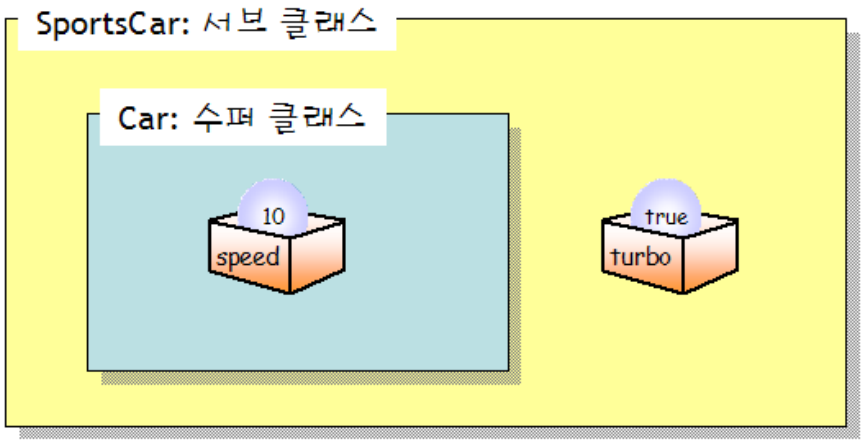
클래스들의 크기

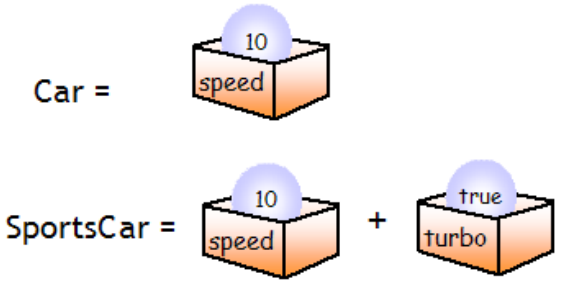
상속을 이용한 일반화

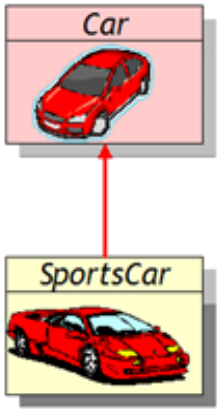

Car
speedUp()
speedDown()
Truck
speedUp()
speedDown()
Bus
speedUp()
speedDown()
class
각 클래스에 코드가 중복된다.
상속
Botton-Up 방식
Vehicle
speedUp()
speedDown()
superclass
Car
Truck
Bus
subclass
중복되는 코드는 수퍼 클래스에 모은다.

수퍼클래스와 서브클래스간의 관계

- 서브 클래스는 수퍼 클래스를 포함







상속은 “is-a” 관계이다.

- subclass is a superclass.
- SportsCar is a car. (O)
- Car is a sportsCar. (X)

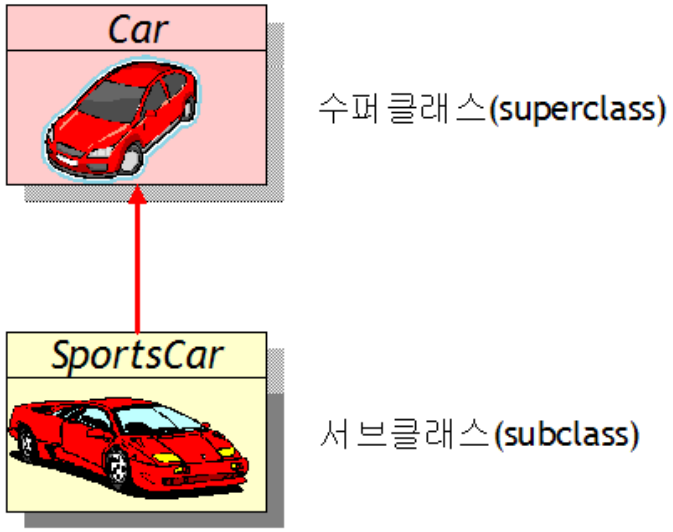
2. 자바에서의 상속

- 키워드 extends를 이용

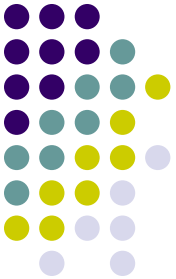
  class AA extends A { … }

```
class Car
{
        int speed;
}
class SportsCar extends Car
{
        int turbo;
}
```

상속한다는 의미



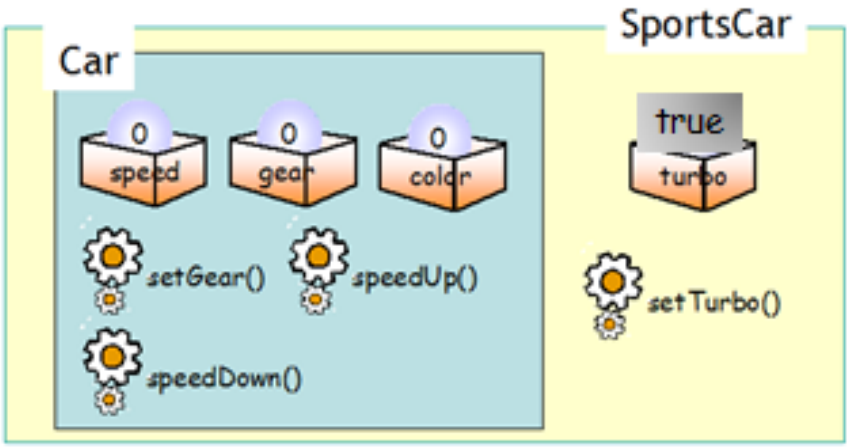
상속의 구현 예 – Car Class



```
public class Car {
        // 3개의 필드 선언
        public int speed; // 속도
        public int gear; // 주행거리
        public String color; // 색상

        // 3개의 메소드 선언
        public void setGear(int newGear) { // 기어 설정 메소드
                gear = newGear;
        }
        public void speedUp(int increment) { // 속도 증가 메소드
                speed += increment;
        }
        public void speedDown(int decrement) { // 속도 감소 메소드
                speed -= decrement;
        }
}
```

상속의 구현 예 – SportCar Class

```
class SportsCar extends Car {          // Car를 상속받는다.
    // 1개의 필드를 추가
    boolean turbo;

    // 1개의 메소드를 추가
    public void setTurbo(boolean newValue) { // 터보 모드 설정 메소드
            turbo = newValue;
    }
};
```



상속 클래스 사용 예

```
public class Test {
    public static void main(String[] args) {
        SportsCar  c = new SportsCar();
        c.color = "Red";                    // 수퍼 클래스 필드 접근
        c.setGear(3);           // 수퍼 클래스 메소드 호출
        c.speedUp(100);                     // 수퍼 클래스 메소드 호출
        c.speedDown(30);                    // 수퍼 클래스 메소드 호출
        c.setTurbo(true);                   // 자체 메소드 호출
    }
}
```

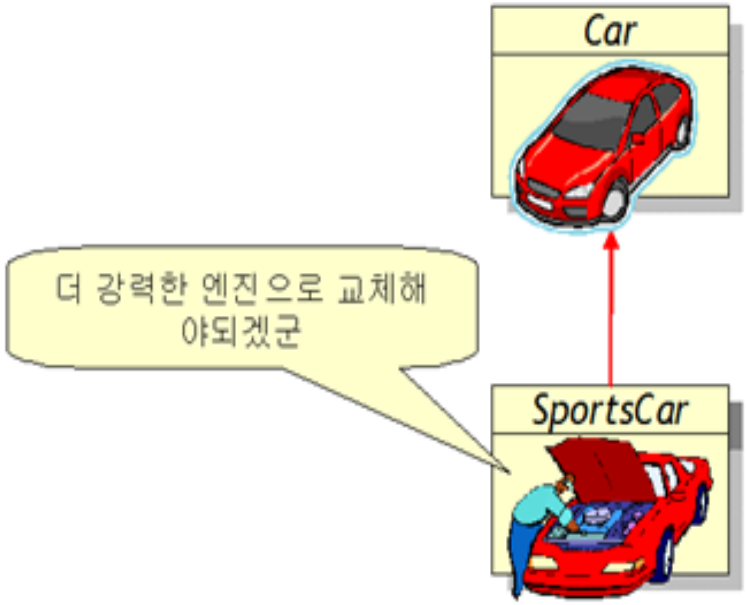
- 서브 클래스는 수퍼 클래스의 필드와 메소드를 마치 자기 것처럼 사용할 수 있다.
- 서브 클래스에서 수퍼 클래스에서 상속받은 메소드가 마음에 들지 않는다면? ⇒ 메소드 재정의(overriding)!

3. 메소드 재정의(overriding)

- 서브클래스가 필요에 따라 상속된 메소드를 재정의하는 것
- **주의사항** : 메소드 이름, 반환형, 매개변수의 개수와 데이터타입이 수퍼클래스에 있는 메소드와 일치해야 함.

```
class Car {
    ...
    public double speedUp(int upSpeed)
    {
        speed += upSpeed;
        if (speed > 120) speed = 120;
    }
}
```

```
class SportsCar extends Car {
    ...
    public double speedUp(int upSpeed)
   {
        speed += upSpeed;
        if (speed > 250) speed = 250;
    }
}
```



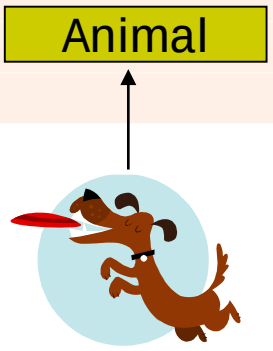
메소드 재정의 예

```
public class Animal {
    public void makeSound()
    {
        // 아직 특정한 동물이 지정되지 않았으므로 몸체는 비어 있다.
    }
};
```

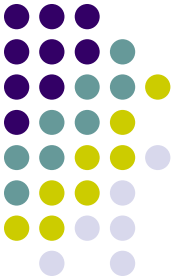
```
public class Dog extends Animal {
    public void makeSound()
    {
        System.out.println("멍멍!“);
    }
};
```

```
public class Cat extends Animal {
    public void makeSaund()
    {   System.out.println(“야옹!”);   }
}
```

@Override



메소드 재정의가 잘못된 예



```
public class Animal {
        public void makeSound()
        {
        }
};
```

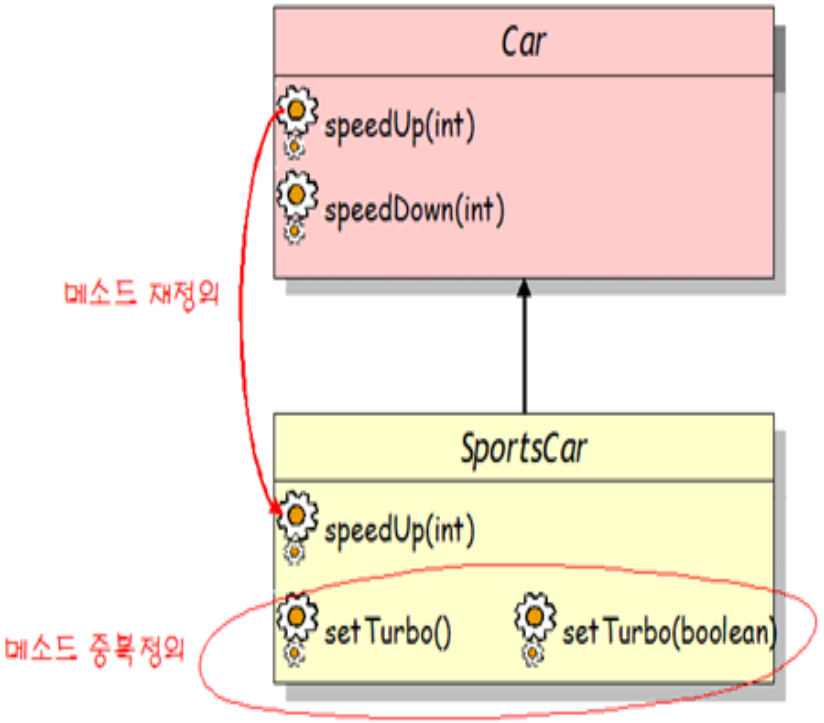
메소드 재정의가 아님(반환형이 다름)

```
public class Dog extends Animal {
        public int makeSound()
        {
        }
};
```

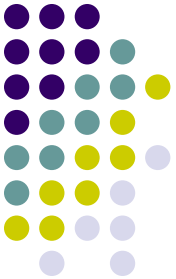
메소드 재정의 vs. 중복 정의

- 메소드 재정의(overriding)
  - 수퍼클래스로부터 상속받은 메소드를 서브클래스에서 자신의 용도에 맞게 다시 정의하는 것

- 메소드 중복정의(overloading)
  - 한 클래스 내에서 이름은 같으나, 매개변수의 개수, 타입, 순서 등이 다른 메소드를 2개이상 정의하는 것



종단 클래스와 종단 메소드



- 키워드 final을 붙이면 더 이상의 클래스 상속이나 메소드 재정의가 불가능

```
final class MyClass
{
    . . . .
}
```

클래스 MyClass가 상속되는 것을 허용하지 않음

```
class YourClass
{
    final void yourFunc(int n) { . . . }
    . . . .
}
```

클래스 YourClass가 상속되는 것은 허용,
메소드 yourFunc의 재정의는 허용안함

4. 서브클래스에서 수퍼클래스 참조

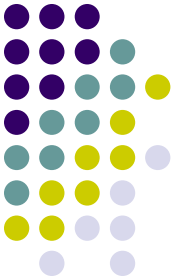
- super : 서브클래스 객체 내부의 수퍼클래스 객체 부분을 가리킴

```
class ParentClass {
    int data=100;
    public void print() {
        System.out.println("수퍼 클래스의 print() 메소드");
    }
}
public class ChildClass extends ParentClass {
    int data=200;
    public void print() { //메소드 재정의
        super.print();
        System.out.println("서브 클래스의 print() 메소드 ");
        System.out.println(data);
        System.out.println(this.data);
        System.out.println(super.data);
    }
    public static void main(String[] args) {
            ChildClass obj = new ChildClass();
            obj.print();
    }
}
```

```
수퍼 클래스의 print() 메소드
서브 클래스의 print() 메소드
200
200
100
```

수퍼클래스 객체를 가리킨다.

서브클래스 생성자에서 수퍼클래스 생성자 호출



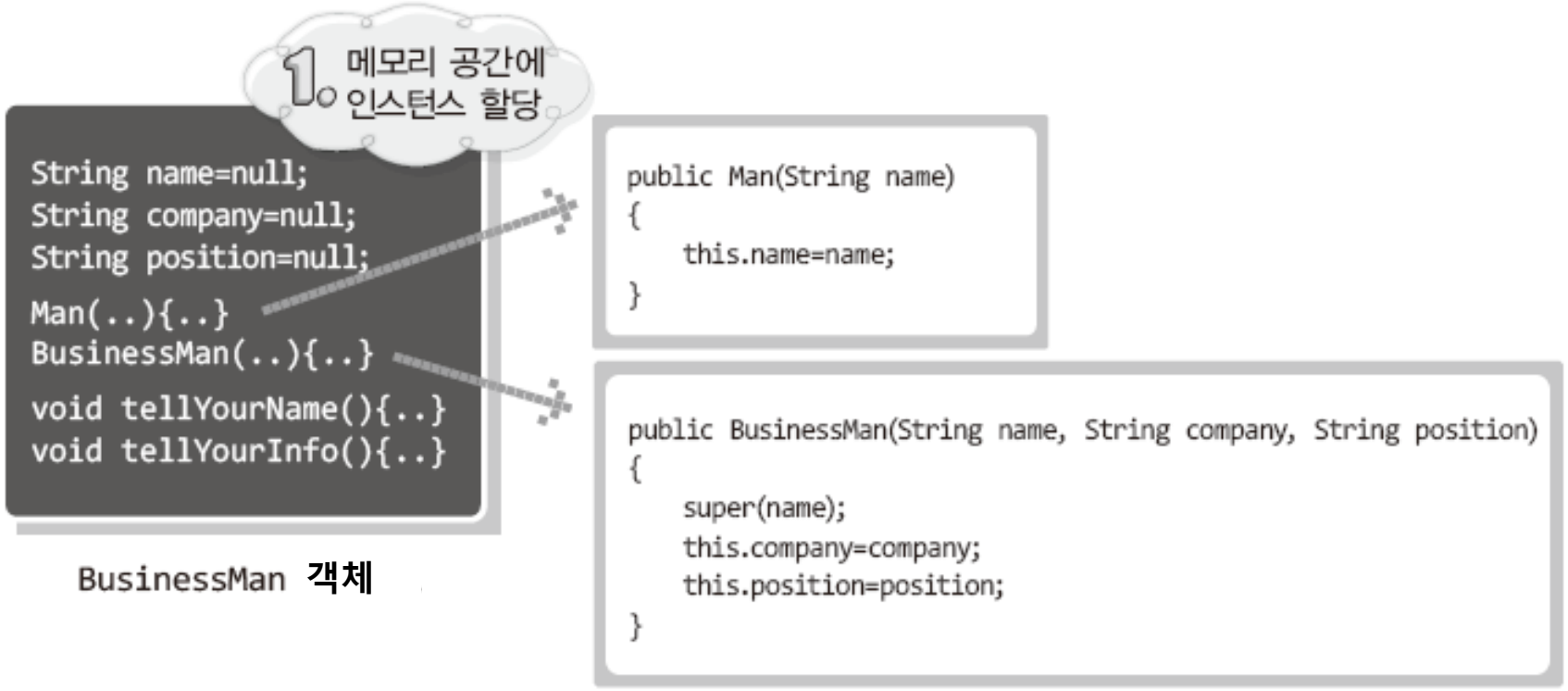
```
class Man
{
    private String name;
    public Man(String name) { this.name=name;}
    public void tellYourName() { System.out.println("My name is "+name); }
}

class BusinessMan extends Man
{
    private String company;         // 회사이름
    private String position;        // 직급
    public BusinessMan(String name, String company, String position)
    {
        super(name);          // 상위 클래스의 생성자 호출문
        this.company=company;
        this.position=position;
    }
    public void tellYourInfo()
    {
        System.out.println("My company is "+company);
        System.out.println("My position is "+position);
        tellYourName();     // Man 클래스를 상속했기 때문에 호출 가능!
    }
}
```

name을 인자로 전달받는 수퍼클래스의 생성자를 호출한다.

외부에서 호출하는 것은 BusinessMan 클래스의 생성자이니, 이 생성자가 수퍼클래스의 인스턴스 변수를 초기화 해야 한다.

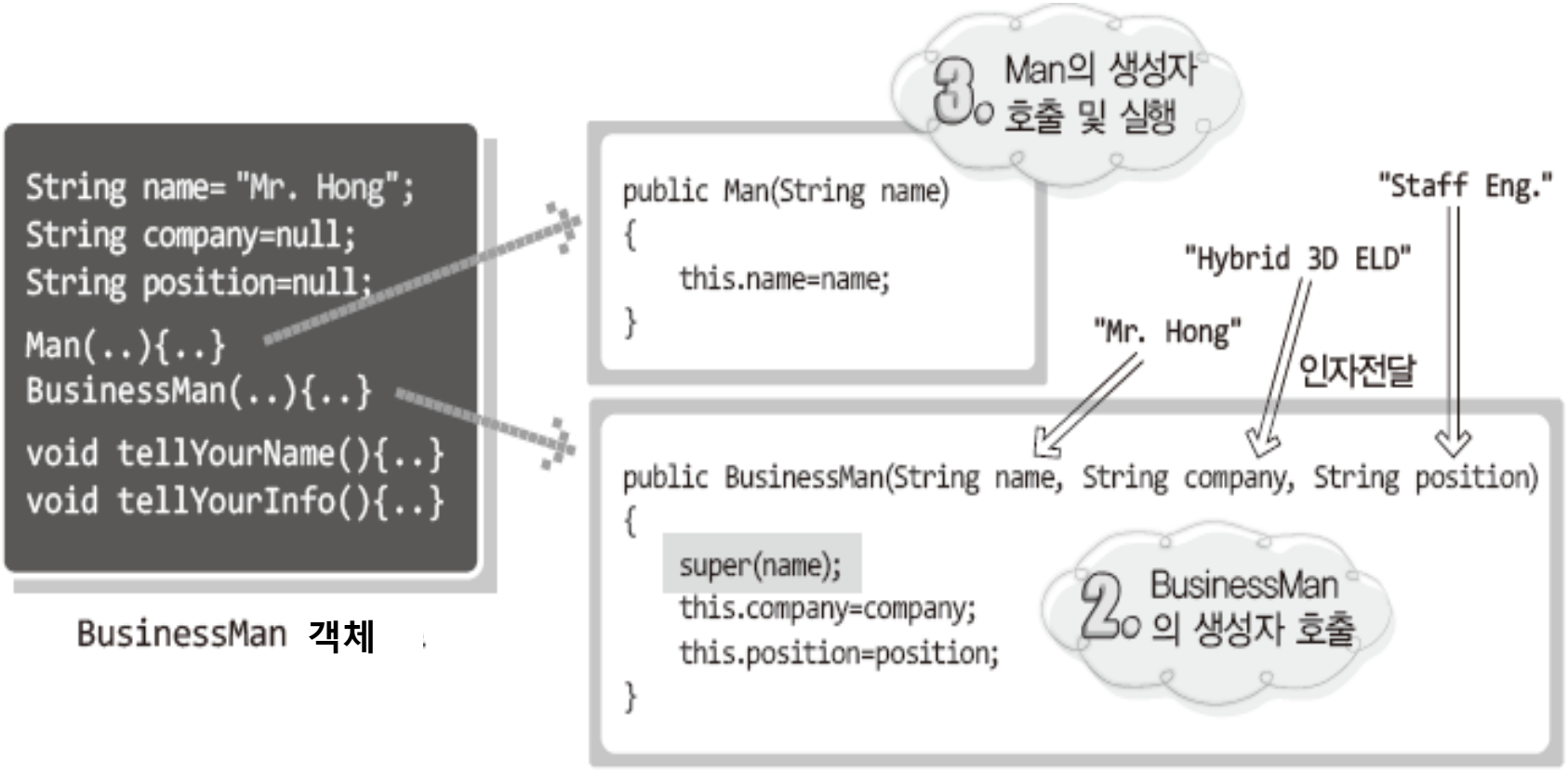
```
public static void main(String[] args) {
   BusinessMan man1
     = new BusinessMan("Mr. Hong", "Hybrid 3D ELD", "Staff Eng.");
   …
}
```

상속관계에 있는 객체 생성 과정1

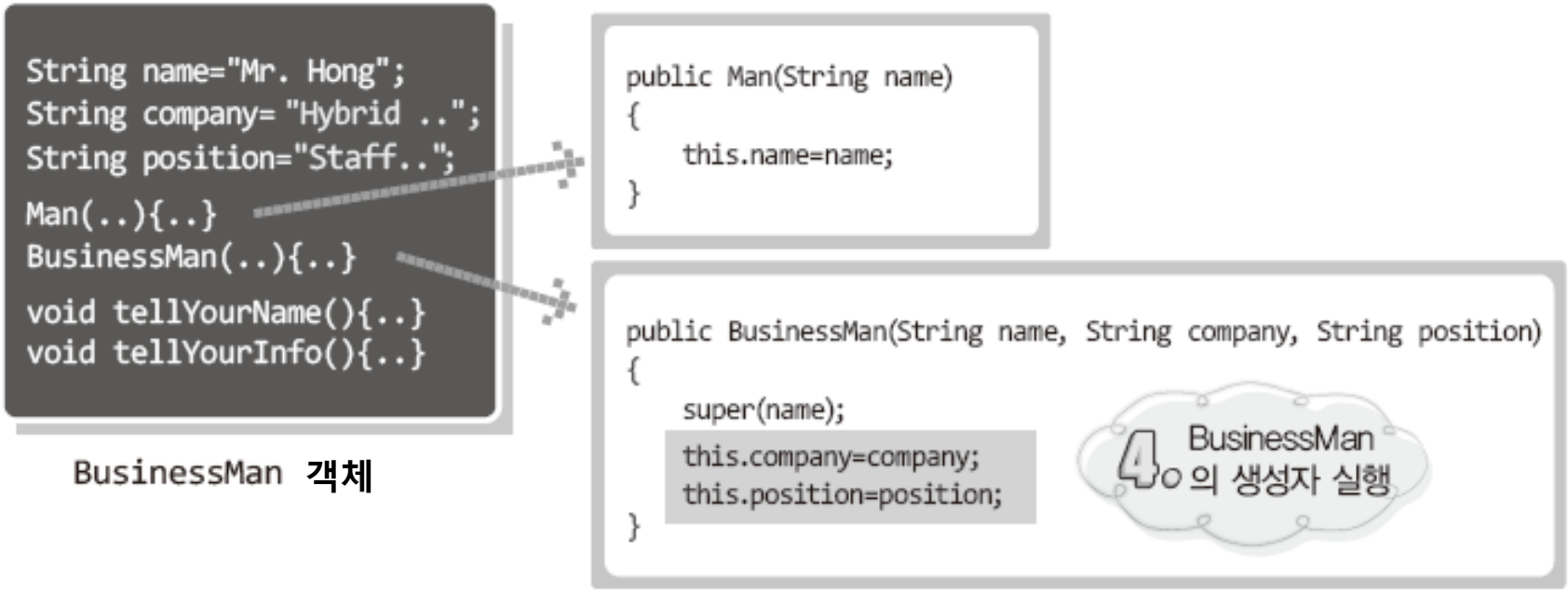



```
public static void main(String[] args) {
     BusinessMan man1
          = new BusinessMan(“Mr. Hong”, “Hybrid 3D ELD”, “Staff Eng.”);
     …
}
```

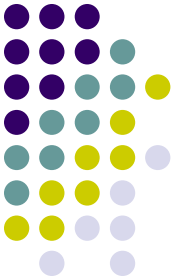
상속관계에 있는 객체 생성 과정2~3



상속관계에 있는 객체 생성 과정4



수퍼클래스 생성자는 반드시 호출됨



```
class AAA
{
    int num1;
}

class BBB extends AAA
{
    int num2;
}
```

AAA() { }

자동으로 삽입되는 디폴트 생성자의 형태

BBB() { super(); }

- 수퍼 클래스의 생성자는 반드시 호출된다!
  - 프로그래머가 직접 삽입, or
  - 컴파일러가 자동으로 삽입

```
class AAA
{
    int num1;
}

class BBB extends AAA
{
    int num2;
    BBB() { num2=0; }
}
```

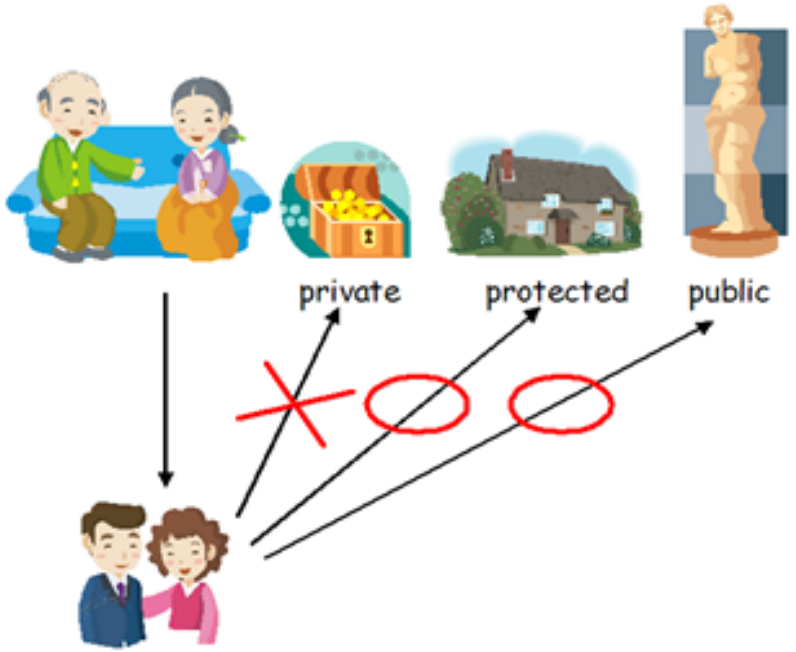
자동으로 삽입되는 수퍼 클래스의 생성자 호출문!

super();

5. 상속과 접근 제어 지시자

지시자	클래스 내부	동일 패키지	상속받은 클래스	이외의 영역
private	●	×	×	×
default	●	●	×	×
protected	●	●	●	×
public	●	●	●	●

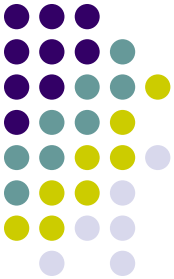
- 서브클래스에서 수퍼클래스의 메소드를 재정의할 때, 접근제어 지시자의 접근성을 축소할 수 없다.
  확대하는 것은 가능하다.



접근제어 지시자 사용 예

```
class Employee {
   public String name;
   private   int  rrn;       // 주민번호
   protected int  salary;
   protected int  getSalary() {return salary;}
   protected void setSalary(int salary){this.salary += salary;}
}

class Manager extends Employee {
   private int bonus;
   public int getSalary() {                    // 가시성 확대함 - OK
     return salary + bonus;
   }
   private void setSalary(int salary) { //  가시성 축소함 - 오류!
     super.salary = salary;
   }
   public void setRrn(int rrn) {
     super.rrn = rrn;                          // 수퍼클래스의 private에 접근불가 – 오류!
   }
}
```

private도 상속은 되나 직접 접근은 불가



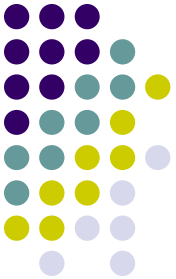
```
class Accumulator
{
    private int val;
    Accumulator(int init){ val=init; }
    protected void accumulate(int num)
    {
        if(num<0)
            return;
        val+=num;
    }
    protected int getAccVal(){return val;}
}
```

```
class SavingAccount extends Accumulator
{
    public SavingAccount(int initDep)
    {
        super(initDep);
    }
    public void saveMoney(int money)
    {
        accumulate(money);
    }
    public void showSavedMoney()
    {
        System.out.print("지금까지의 누적금액 : ");
        System.out.println(getAccVal());
    }
}
```

private 멤버도 상속이 된다! 다만, 함께 상속된 다른 메소드를 통해서 간접 접근을 해야만 한다!

```
public static void main(String[] args) {
    SavingAccount s = new SavingAccount(10000);
    s.saveMoney(10000);
    s.showSaveMoney();
}
```

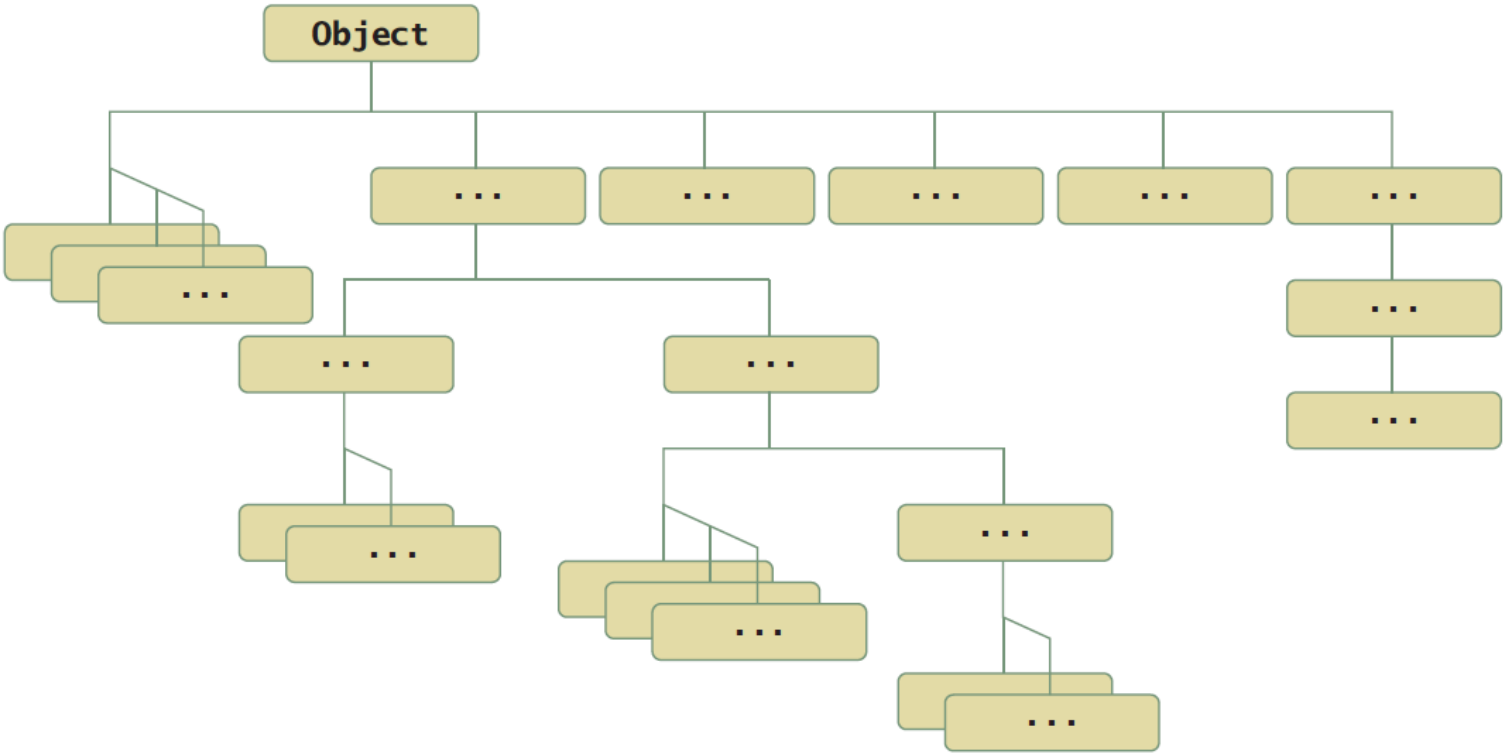
상속 관련 주요 개념 정리



- 서브클래스는 수퍼클래스를 상속
- 서브클래스는 수퍼클래스에 있는 모든 public으로 지정한 인스턴스 변수와 메소드를 상속. private으로 지정한 변수와 메소드는 상속은 되나 직접 접근은 불가
- 메소드는 재정의할 수 있지만 인스턴스 변수는 재정의 불가
- ‘A는 B다’ 테스트를 활용하여 상속 계층이 올바른지 확인!
- ‘A는 B다’ 관계는 한 방향으로만 작동
- 서브클래스에서 메소드를 오버라이드하면, 그리고 서브 클래스의 객체에 대해 그 메소드를 호출하면 오버라이드된 버전의 메소드가 호출됨
- B라는 클래스가 A라는 클래스를 상속하고 C는 B를 상속한다면 B는 A이고 C는 B이면서 또한 A가 됨

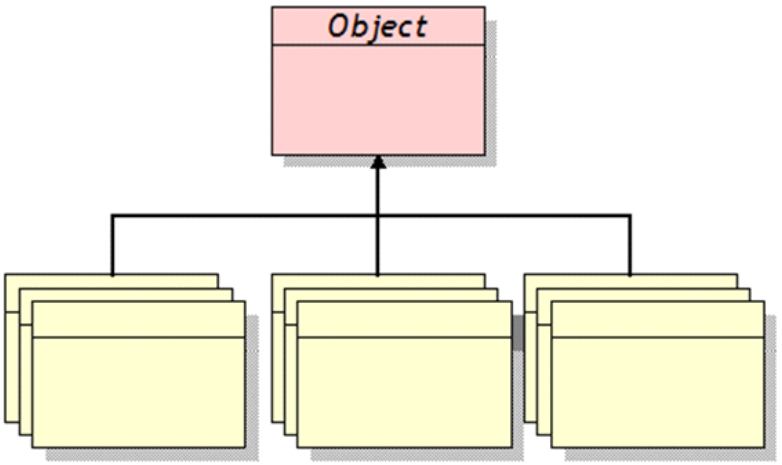
6. 자바 최상위 클래스 Object

- Object 클래스는 java.lang 패키지에 들어 있으며 자바 클래스 계층 구조에서 맨 위에 위치하는 클래스



Object 클래스

- Object 클래스는 자바 상속계층의 가장 위에 있다.
- 모든 자바 클래스는 내부적으로 object 클래스를 상속
  - 자바 클래스가 아무것도 상속하지 않으면 java.lang 패키지의 Object 클래스를 자동으로 상속한다
  - 자바의 모든 객체는 Object 클래스에 정의된 메소드 호출 가능




```
class MyClass { . . . }
```

```
class MyClass extends Object { . . . }
```

Object 클래스 주요 메소드

메소드	설명
protected Object clone()	객체 자신의 복사본을 생성하여 반환한다.
public boolean equals(Object obj)	Obj가 this 객체와 같은지를 나타낸다.
protected void finalize()	가비지 콜렉터(garbage collector)에 의하여 호출된다.
public final Class getClass()	객체의 실행 클래스를 반환한다.
public int hashCode()	객체에 대한 해쉬 코드를 반환한다.
public String toString()	객체의 문자열 표현을 반환한다.

equals() 메소드

※ 자바에서 == 연산자는 참조값 비교를 함.

- 기초 자료형의 경우에는 올바른 결과를 생성
- 객체에 대해서는 객체 참조값이 같은지를 검사 (비교되는 객체가 동일한 객체인지 검사)

- 비교 대상이 되는 객체가 동일한 객체인지 판단한다.
- 객체 간 내용 비교를 위해서는 내용을 비교할 수 있는 기능을 가진 메소드가 필요하다.
  - Object 클래스에 정의된 equals()를 재정의(override)해야 한다.

equals() 메소드 재정의 예

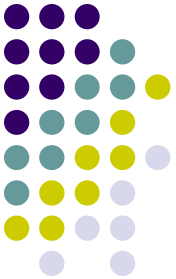
```
public class Car {
        private String model;

        public Car(String model) {
                this.model = model;
        }

        public String getModel() {
                return model;
        }

        public boolean equals(Object obj) {
                if (obj instanceof Car)
                        return model.equals(((Car) obj).getModel());
                else
                        return false;
        }
}
```

Object의 equals()를 재정의

재정의된 equals() 호출

```
public static void main(String[] args) {
                Car firstCar = new Car("HMW520");
                Car secondCar = new Car("HMW520");
                if (firstCar.equals(secondCar)) {
                        System.out.println("동일한 종류의 자동차입니다.");
                } else {
                        System.out.println("동일한 종류의 자동차가 아닙니다.");
                }
}
```

toString() 메소드



- Object 클래스의 toString() - 객체의 문자열 표현을 반환
- 객체에 대한 문자열 표현은 객체에 따라 달라져야 하므로 각 클래스에서 toString() 메소드를 재정의해야 함
- **모든 클래스가 Object 클래스를 상속하므로,**
  - Object 클래스에 정의되어 있는 toString 메소드 :

    public String toString( ) { . . . }
  - 우리가 흔히 호출하는 println 메소드는 다음과 같이 정의되어 있다.

    public void println(Object x) { . . . }
  - 때문에 모든 객체는 println(…) 메소드의 인자로 전달될 수 있다.
    - 수퍼클래스 참조변수로 서브클래스 객체를 받을 수 있다. ⇒ 다형성
  - 인자로 전달되면, 전달된 객체의 toString 메소드가 호출되고, 이 때 반환되는 문자열이 출력된다.
  - 때문에 각 클래스에서 toString 메소드는 적절한 문자열 정보를 반환하도록 재정의(override)하는 것이 좋다!

toString() 메소드 사용 예

```
class Employee {
    public String name;
    private int   rrn;// 주민번호
    public Employee(String name, int rrn) {
        this.name = name;
        this.rrn = rrn;
    }
    public String toString()
    { return name + " : " + rrn; }
}
class Manager extends Employee {
    double bonus;
    public Manager(String name, int rrn, double bonus) {
        super(name, rrn);
        this.bonus = bonus;
    }
    public String toString()
    { return super.toString() + " : " + bonus; }
}
```

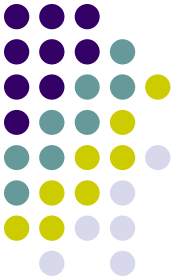
```
public static void main(String[] args)
{
    Employee e1 = new Employee("Kim", 111);
    Manager m1 = new Manager("Lee",222,0.1);
    System.out.println(e1); //e1.toString()
    System.out.println(m1); //m1.toString()
}
```

Object의 tostring()를 재정의

출력 결과

```
Kim : 111
Lee : 222 : 0.1
```

finalize() 메소드



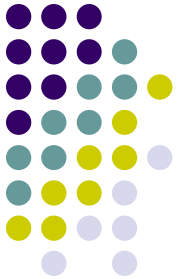
- Garbage Collector에 의해 객체가 완전히 소멸되기 직전 호출된다.
- 객체가 사용하던 자원을 반납한다.
- 모든 클래스가 재정의해서 이 메소드를 호출할 수 있다.

```
class MyName
{
    String objName;
    public MyName(String name)
    {
        objName=name;
    }
    protected void finalize() throws Throwable
    {
        super.finalize();
        System.out.println(objName+"이 소멸되었습니다.");
    }
}
```

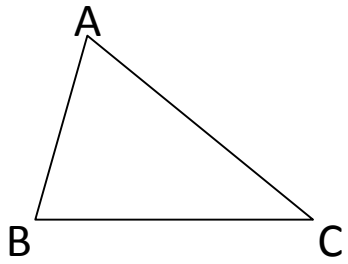
```
public static void main(String[] args)
{
    MyName obj1=new MyName("인스턴스1");
    MyName obj2=new MyName("인스턴스2");
    obj1=null;
    obj2=null;

    System.out.println("프로그램을 종료합니다.");
    // System.gc();
    // System.runFinalization();
}
```

강제로 Garbage Collection을 실행시켜 finalize 메소드를 호출시키는 방법

7. Point 클래스 계층 설계 및 구현



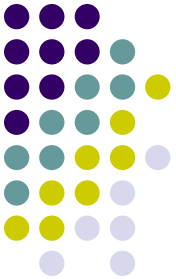
- 점 클래스 (2개)
  - Point2D
    - 2차원 좌표상의 점을 나타내는 클래스
  - NamedPoint2D
    - 이름이 붙여진 2차원 좌표상의 점을 나타내는 클래스



- Point2D 클래스와 NamedPoint2D 클래스간의 관계는?
  - IS-A 관계 성립? ⇒ 상속 적용 가능
    - NamedPoint2D ***IS-A*** Point2D.

- 클래스 간 역할 정립
  - Point2D – 수퍼클래스, NamedPoint2D - 서브클래스

클래스 설계 절차

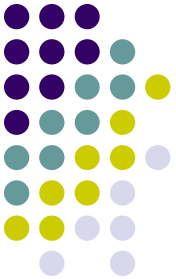


1. 클래스 이름 정하기
   - 클래스가 나타내는 대상을 클래스 이름으로 정함
   - 가능하면 구체적인 이름으로 정하는 것이 좋음

2. 데이터 필드 정하기
   - 이 클래스의 객체에 저장해야 하는 데이터와 타입을 정함
   - 필드 이름을 먼저 정하고 적합한 타입을 정함

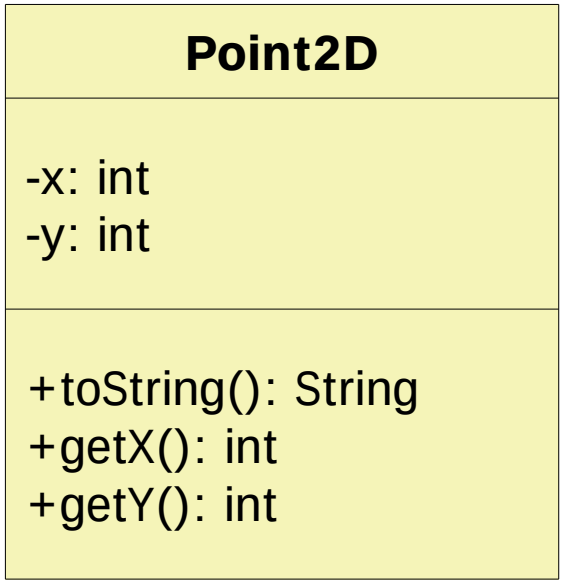
3. 메소드 정하기
   - 이 클래스의 객체에 적용할 수 있는 연산들을 생각함
   - 이 클래스를 이용하여 하려는 일에 필요한 어떤 작업이 있다면 이것을 메소드로 정함

Point2D 클래스 설계

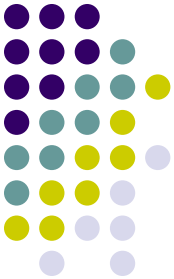


- 2차원 점 클래스 설계
  - 클래스 이름: Point2D
  - 필드 설계: x, y
  - 메소드 설계: toString(), getX(), getY()

- 클래스 다이어그램
  - 클래스 설계 내용을 담고 있음



NamedPoint2D 클래스 설계



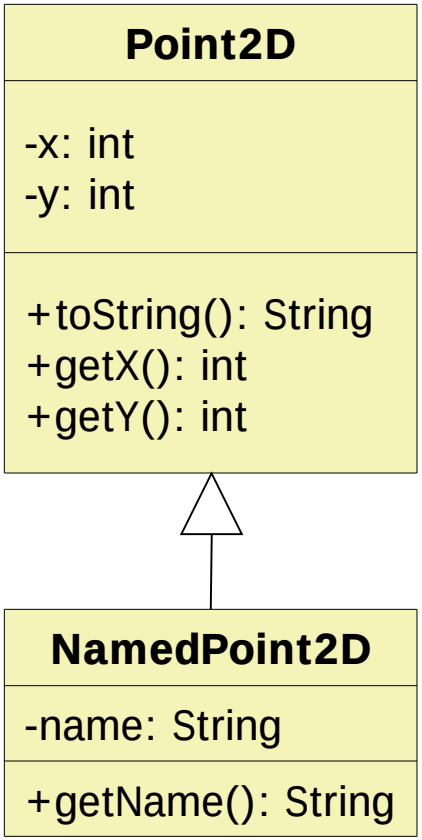
- 이름이 붙여진 점 클래스 설계
  - 클래스 이름: NamedPoint
  - 필드 설계: name, x, y
  - 메소드 설계: toString(), getX(), getY(), getName()

NamedPoint
-name: String -x: int -y: int
+toString(): String +getX(): int +getY(): int +getName(): String

클래스 상속 설계 절차

- 클래스 상속 설계 절차
  1. 수퍼클래스와 중복되는 필드 삭제
  2. 서브클래스의 오버라이딩 메소드 정의
  3. 상속 관계 수립: 하얀 삼각형 Δ 화살표

- NamedPoint2D의 상속 설계
  - 필드 x, y 삭제
  - 메소드 toString, getX, getY 삭제



클래스 상속의 구현

- 클래스 상속 프로그래밍 절차
  1. 상속 관계: 키워드 extends로 상속
  2. 필드 가시성: 꼭 필요하면 private → protected
  3. 서브클래스 생성자: 생성자 첫 줄에서 키워드 super로 수퍼클래스 생성자 호출
  4. 메소드 재정의: 서브클래스에서 다르게 구현해야 할 메소드 재지정(super 이용 가능)

- NamedPoint의 상속 프로그래밍
  - 2번은 필요 없음
  - 3번: super(x, y) 호출 필요
  - 4번: toString 메소드 재정의

Point2D 클래스 구현

```
public class Point 2D
{
    private int x, y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public String toString() {
        String buf = "(" + x + ", " + y + ")";
        return buf;
    }
    public int getX() {
        return x;
    }
    public int getY() {
        return y;
    }
}
```

메소드 toString 재정의

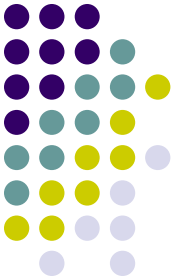
NamedPoint2D 클래스 구현

```
public class NamedPoint2D extends Point2D
{
    private String name;

    public NamedPoint(String name, int x, int y)
    {
        super(x, y);
        this.name = name;
    }
    public String toString()
    {
        String buf = name + super.toString();
        return buf;
    }
    public String getName()
    {
        return name;
    }
}
```

수퍼클래스 Point의 생성자 호출

메소드 toString 재정의

수퍼클래스 Point의 메소드 toString 호출



수고했습니다!!!