

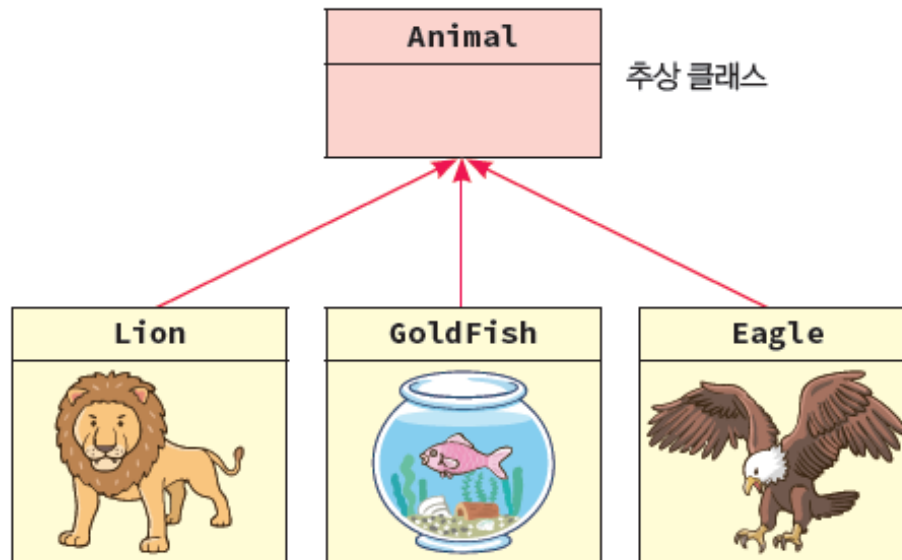
## 09. 다형성



강동기

# 추상 클래스

- abstract class
- 추상 메서드(몸체가 구현되지 않은)를 가지고 있는 클래스
  - abstract 키워드를 class 앞에 붙임
  - abstract 키워드가 붙은 메서드를 포함함
  - 추상적인 개념을 표현하는데 활용
- 상속 전용으로 사용, 객체 생성 불가
  - 일반 멤버 변수와 일반 메서드도 구현은 가능



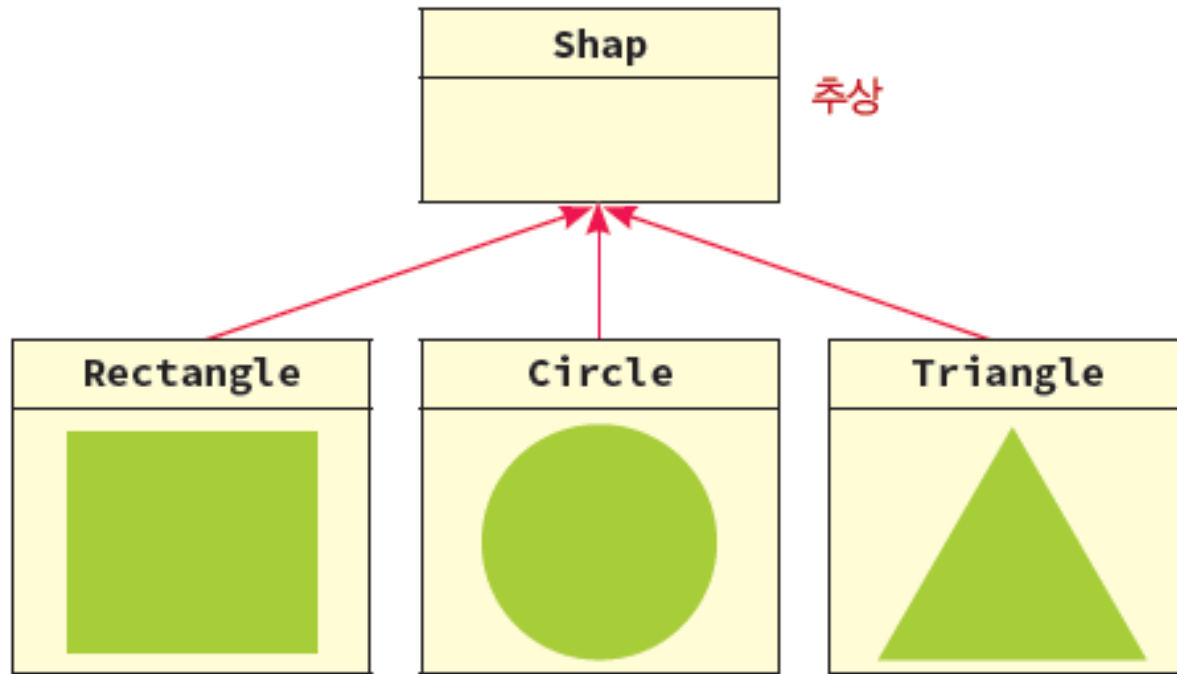
# 추상 클래스 사용

전체적인 구조



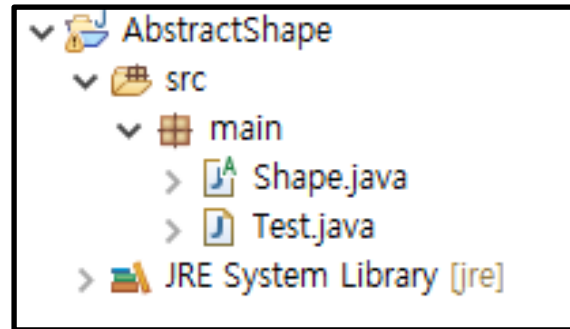
```
public abstract class Animal {  
    public abstract void move();  
    ...  
};
```

# 도형 추상 클래스 예제



# 도형 추상 클래스 예제

- 프로젝트 구조



# 도형 추상 클래스 예제

- Shape.java

```
package main;

abstract public class Shape {
    int x, y;

    public void move(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public abstract void draw();
}

class Rectangle extends Shape {
    int width, height;

    public void draw() {
        System.out.println("draw a rectangle");
    }
}

class Circle extends Shape {
    int radius;

    public void draw() {
        System.out.println("draw a circle");
    }
};
```

# 도형 추상 클래스 예제

- Test.java

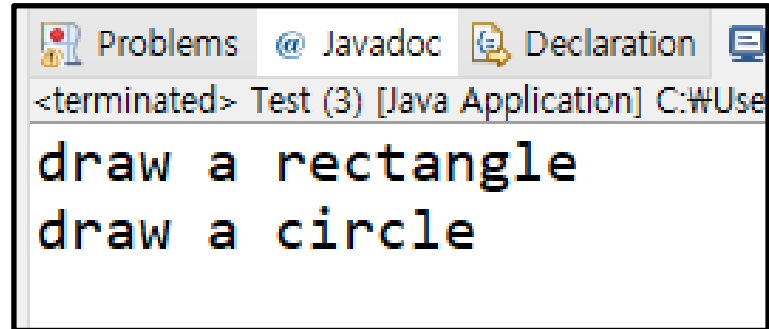
```
package main;

public class Test {
    public static void main(String args[]) {
//        Shape s = new Shape();

        Rectangle r = new Rectangle();
        Circle c = new Circle();
        r.draw();
        c.draw();
    }
}
```

# 도형 추상 클래스 예제

- Test.java 실행 결과



The screenshot shows a console window from an IDE. The title bar includes tabs for 'Problems', 'Javadoc', and 'Declaration'. The main text area displays the output of a Java application, which has terminated. The output consists of two lines: 'draw a rectangle' and 'draw a circle'.

```
<terminated> Test (3) [Java Application] C:\#Use  
draw a rectangle  
draw a circle
```



# 다형성

- polymorphism
  - 하나의 객체가 (상황에 따라) 여러 가지 클래스 타입을 가질 수 있는 것을 의미



# 다형성

- 클래스 A의 참조 변수로 클래스 B의 객체를 참조할 수 **없음**

```
class A {  
    A() {}  
}  
  
class B {  
    B() {}  
}  
  
public class Test {  
    public static void main(String args[]) {  
        A a = new B(); // 멈춰!  
    }  
}
```

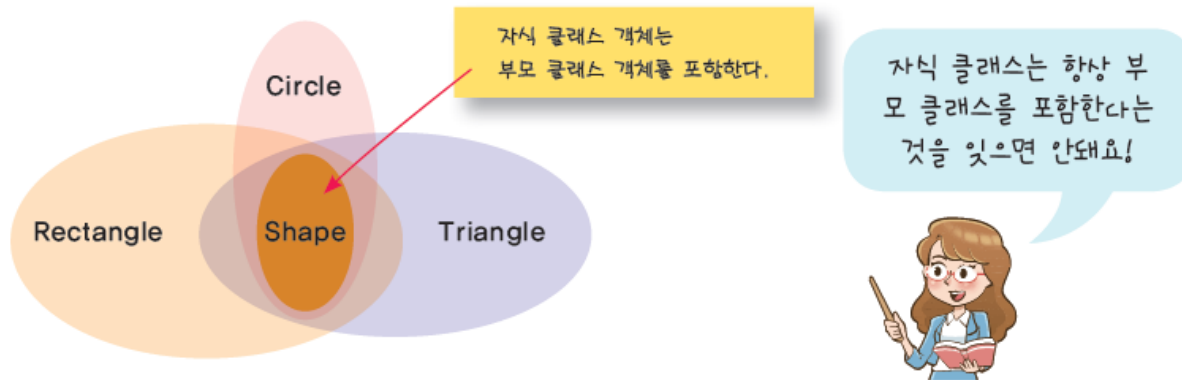
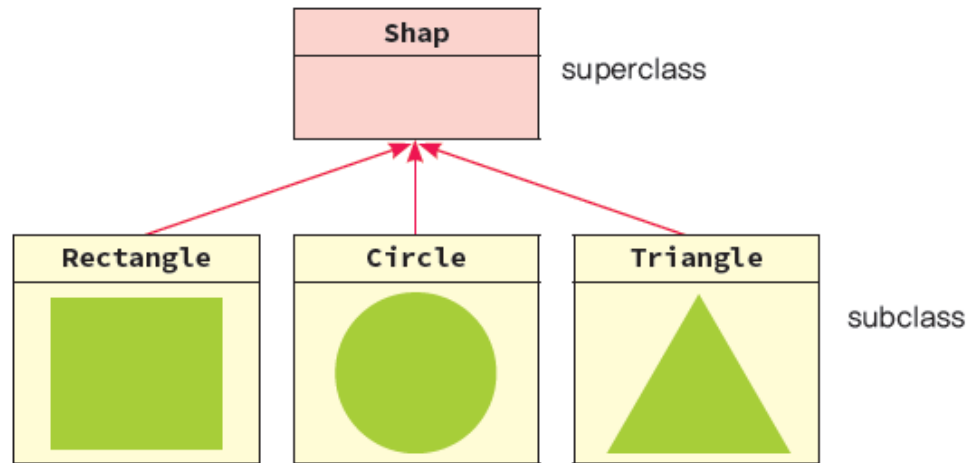
# 다형성

- 부모 클래스 **A**의 참조 변수로 자식 클래스 **B**의 객체를 참조할 수 **있음**

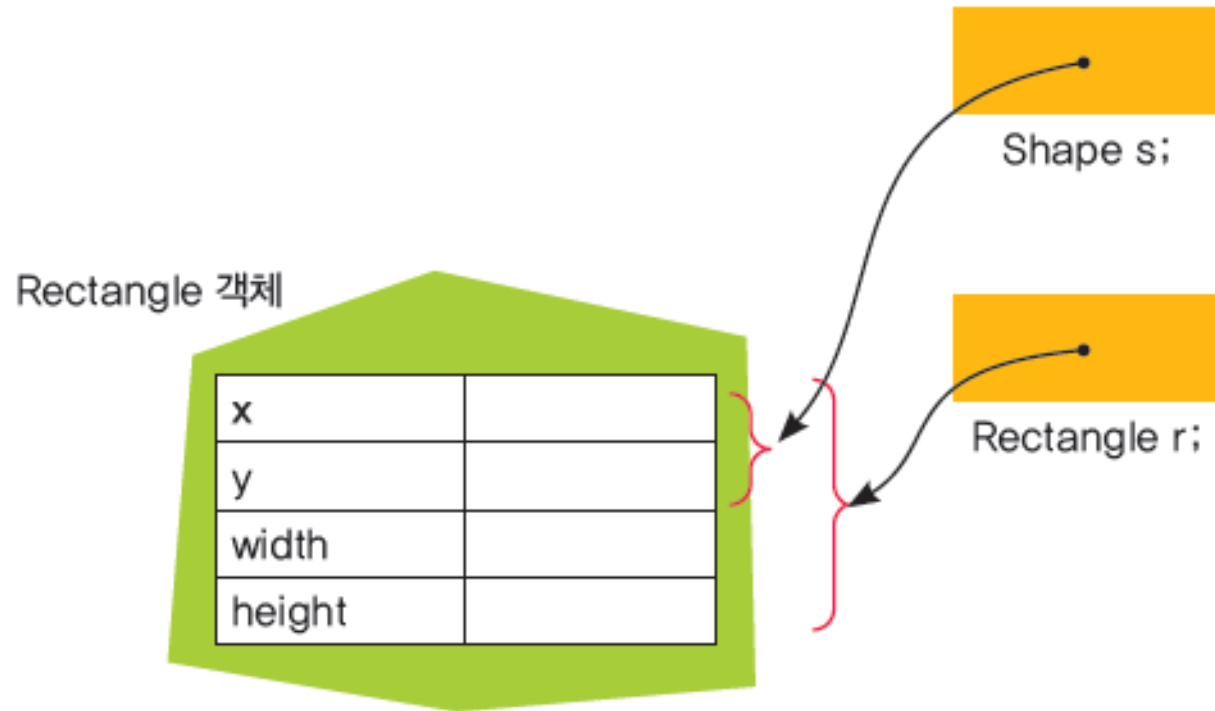
```
class A {  
    A() {}  
}  
  
class B extends A {  
    B() {}  
}  
  
public class Test {  
    public static void main(String args[]) {  
        A a = new B();    // OK!  
    }  
}
```

# 다형성

- 자식 클래스 객체는 부모 클래스 객체의 모든 내용을 포함하고 있기 때문
- 상향 형변환 (Up Casting)

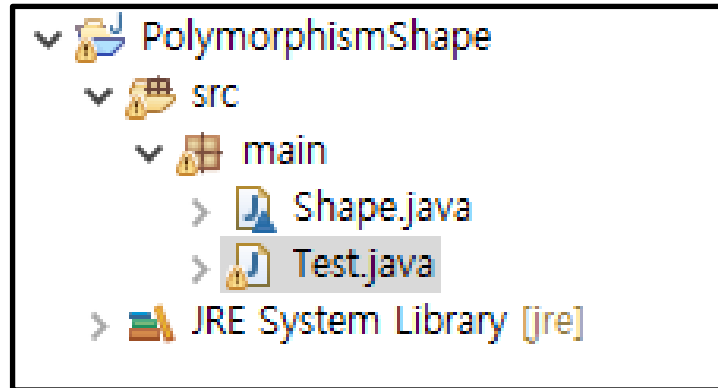


# 다형성



# 다형성

- 프로젝트 구조



# 다형성

- Shape.java

```
package main;

class Shape {
    protected int x, y;
}

class Rectangle extends Shape {
    public int width, height;
}

class Triangle extends Shape {
    public int base, height;
}

class Circle extends Shape {
    public int radius;
}
```

# 다형성

- Test.java

```
package main;

public class Test {
    public static void main(String arg[]) {
        Shape s1, s2;
        Rectangle r1, r2;

        s1 = new Shape();
        s2 = new Rectangle(); // OK

        //      r1 = new Shape(); // 예러!
        r2 = new Rectangle();
    }
}
```



# 다형성

- 수정된 Test.java

```
package main;

public class Test {
    public static void main(String arg[]) {
        Shape s1, s2;
        Rectangle r1, r2;

        s1 = new Shape();
        s2 = new Rectangle();

        // r1 = new Shape(); // 에러!
        r2 = new Rectangle();

        s1.x = 0;
        s1.y = 0;
        // s1.width = 100; // 에러!
        // s1.height = 100; // 에러!

        r2.x = 0;
        r2.y = 0;
        r2.width = 100; // OK
        r2.height = 100; // OK
    }
}
```

# 다형성과 형변환

---

- Shape s = new Rectangle();
- s 를 통하여 Rectangle 클래스의 필드와 메소드를 사용하고자 할 때는 어떻게 할까?

`((Rectangle) s).width = 100;`

# 다형성과 형변환

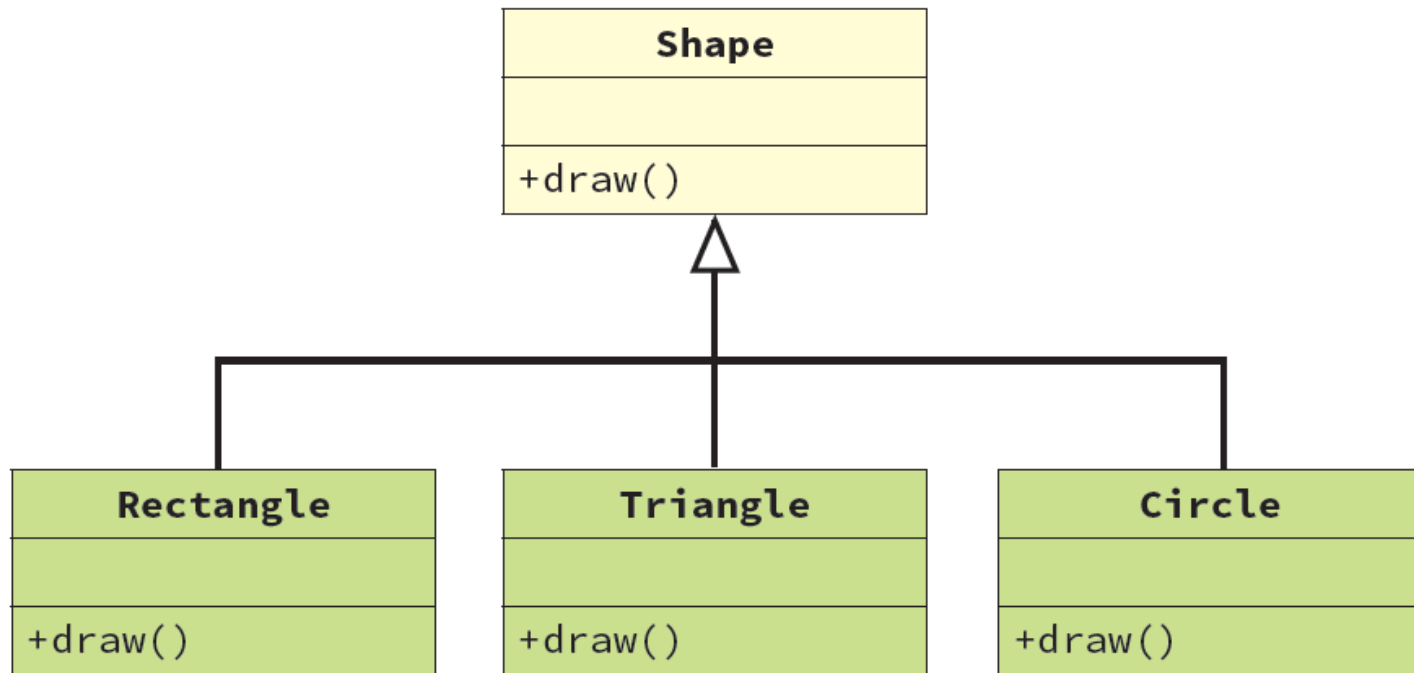
- 자식 클래스 객체로 부모 클래스 객체를 참조하면 일반적인 상황에서는 컴파일 오류가 발생
- 자식 클래스 객체라 하더라도 부모 클래스 객체를 형변환 후 참조하는 경우는 가능

```
Rectangle r;  
r = new Shape();    // NOT OK!
```

```
Rectangle r;  
Shape s;  
s = new Rectangle();  
r = (Rectangle) s;  
r->width = 100;  
r->height = 100;
```

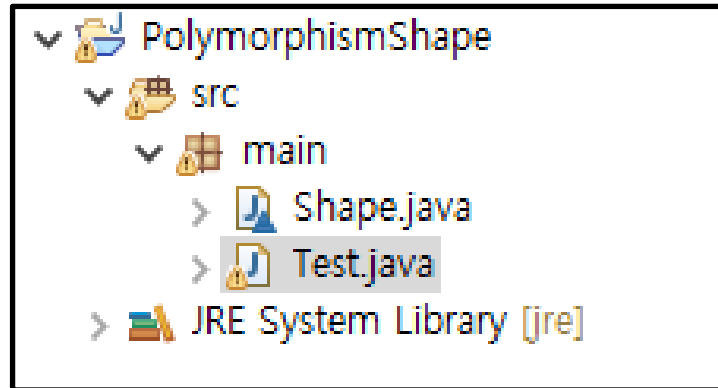
# 동적 메서드

- dynamic method
- 동일한 메서드가 동일한 메시지를 받더라도 (다형성에 의해) 각 객체의 타입에 따라서 서로 다른 동작을 하는 것



# 동적 메서드 예제

- 프로젝트 구조



# 동적 메서드 예제

- Shape.java

```
package main;

class Shape {
    protected int x, y;
    public void draw() {
        System.out.println("Shape Draw");
    }
}

class Rectangle extends Shape {
    public int width, height;
    public void draw() {
        System.out.println("Rectangle Draw");
    }
}

class Triangle extends Shape {
    public int base, height;
    public void draw() {
        System.out.println("Triangle Draw");
    }
}

class Circle extends Shape {
    public int radius;
    public void draw() {
        System.out.println("Circle Draw");
    }
}
```

# 동적 메서드 예제

- Test.java

```
package main;

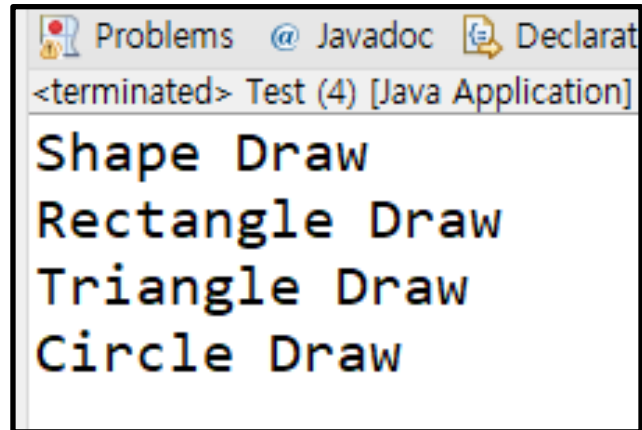
public class Test {
    public static void main(String arg[]) {
        Shape s1, s2, s3, s4;

        s1 = new Shape();
        s2 = new Rectangle();
        s3 = new Triangle();
        s4 = new Circle();

        s1.draw();
        s2.draw();
        s3.draw();
        s4.draw();
    }
}
```

# 동적 메서드 예제

- Test.java 실행 결과



The screenshot shows a Java IDE's console window. The title bar includes icons for Problems, Javadoc, and Declarations. The text in the console reads: <terminated> Test (4) [Java Application] followed by four lines of output: Shape Draw, Rectangle Draw, Triangle Draw, and Circle Draw.

```
<terminated> Test (4) [Java Application]  
Shape Draw  
Rectangle Draw  
Triangle Draw  
Circle Draw
```



# 동적 메서드 예제

- 수정된 Test.java

```
package main;

public class Test {
    private static Shape arrayOfShapes[];

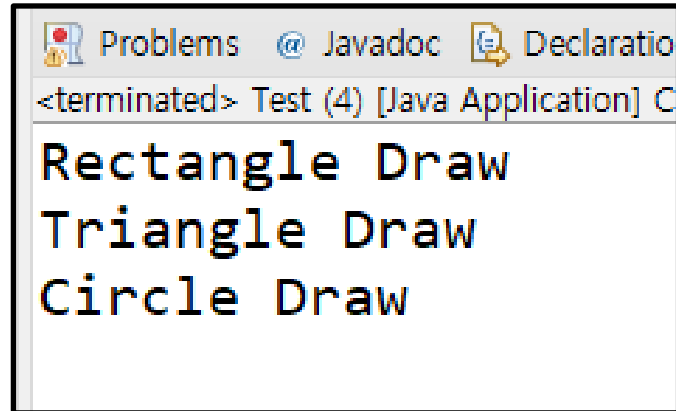
    public static void main(String arg[]) {
        init();
        drawAll();
    }

    public static void init() {
        arrayOfShapes = new Shape[3];
        arrayOfShapes[0] = new Rectangle();
        arrayOfShapes[1] = new Triangle();
        arrayOfShapes[2] = new Circle();
    }

    public static void drawAll() {
        for (int i = 0; i < arrayOfShapes.length; i++) {
            arrayOfShapes[i].draw();
        }
    }
}
```

# 동적 메서드 예제

- 수정된 Test.java 실행 결과



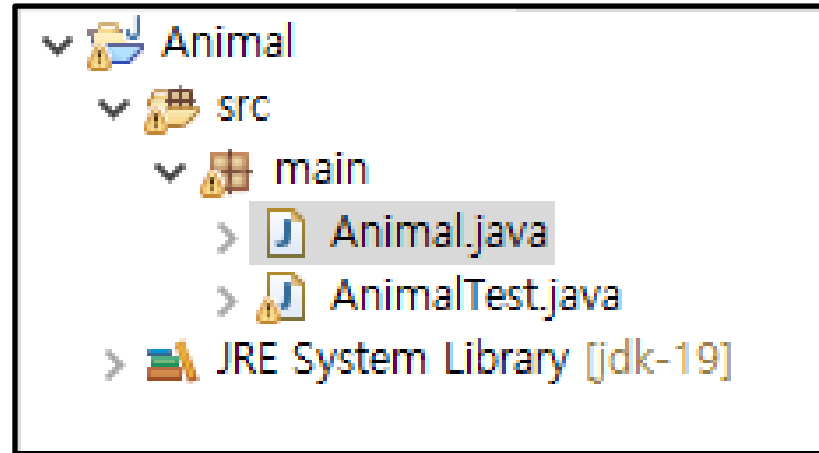
```
<terminated> Test (4) [Java Application] C
Rectangle Draw
Triangle Draw
Circle Draw
```

# 정적 메서드 오버라이딩 불가

- 정적 메서드는 오버라이딩 대신 메서드 추가의 개념으로 접근함
  - **method hiding, method shadowing** 이라고 함
  - 자식 클래스의 메서드가 부모 클래스의 메서드를 숨긴다고 표현
- 정적 메서드는 호출 시 참조 객체가 아닌 컴파일 시 결정된 클래스 타입을 고려함 (**즉 정적 메서드는 다형성이 적용되지 않음**)
- 부모 클래스의 정적 메서드를 자식 클래스에서 오버라이딩 시도 하는 경우
  - 부모 클래스 객체에서 호출되느냐
  - 자식 클래스 객체에서 호출되느냐에 따라서
- 호출되는 메서드가 달라짐

# 정적 메서드 오버라이딩 불가 예제

- 프로젝트 구조



# 정적 메서드 오버라이딩 불가 예제

- Animal.java

```
package main;

public class Animal {
    public static void eat() {
        System.out.println("Animal (static) eat()");
    }
    public void sound() {
        System.out.println("Animal (instance) sound()");
    }
}

class Cat extends Animal {
    public static void eat() {
        System.out.println("Cat (static) eat()");
    }
    public void sound() {
        System.out.println("Cat (instance) sound()");
    }
}
```

# 정적 메서드 오버라이딩 불가 예제

- AnimalTest.java

```
package main;

public class AnimalTest {
    public static void main(String[] args) {
        Cat cat = new Cat();
        Animal animal = cat;

        Animal.eat();
        animal.eat();

        System.out.println();

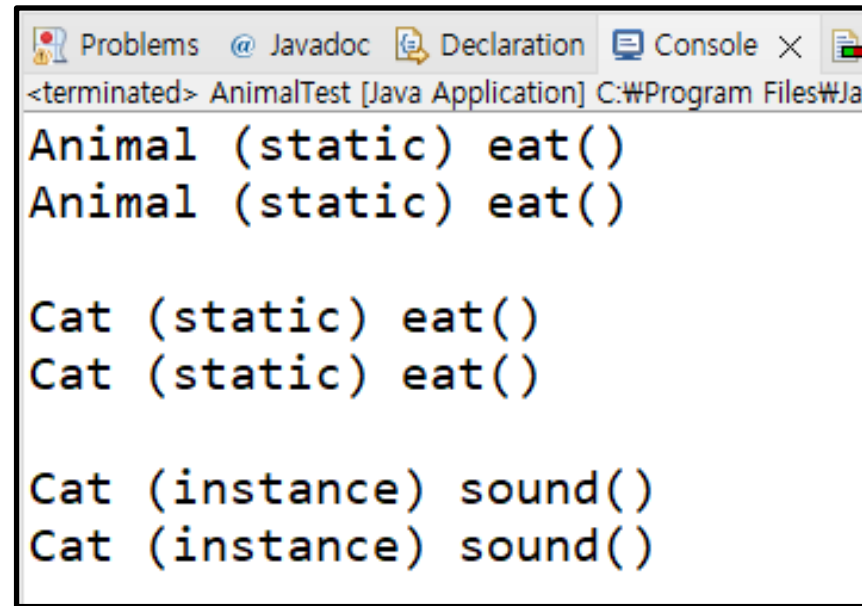
        Cat.eat();
        cat.eat();

        System.out.println();

        animal.sound();
        cat.sound();
    }
}
```

# 정적 메서드 오버라이딩 불가 예제

- AnimalTest.java 실행 결과



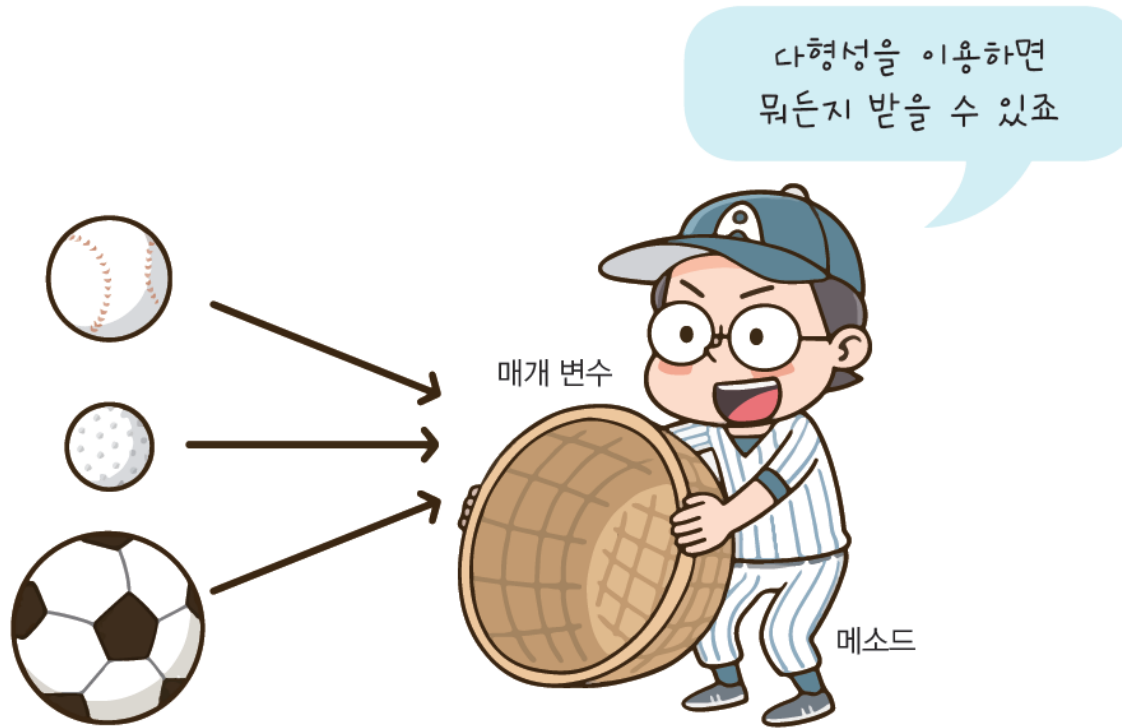
```
<terminated> AnimalTest [Java Application] C:\Program Files\Ja
Animal (static) eat()
Animal (static) eat()

Cat (static) eat()
Cat (static) eat()

Cat (instance) sound()
Cat (instance) sound()
```

# 다형성과 매개 변수

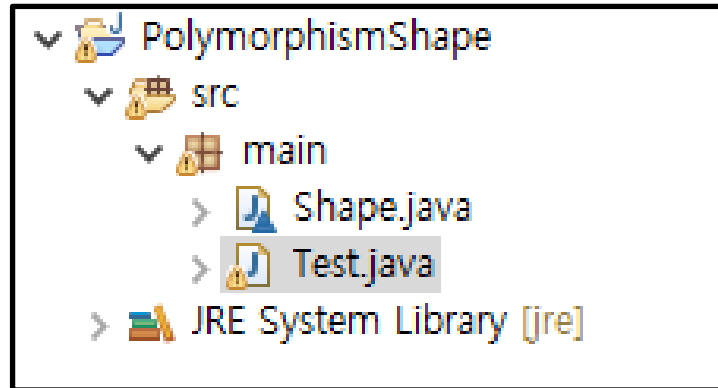
- 메소드의 매개 변수로 부모 클래스 참조 변수를 이용
  - 다형성을 이용하는 전형적인 방법





# 다형성과 매개 변수 예제

- 프로젝트 구조



# 다형성과 매개 변수 예제

- Shape.java

```
package main;

class Shape {
    protected int x, y;
    public void draw() {
        System.out.println("Shape Draw");
    }
}

class Rectangle extends Shape {
    public int width, height;
    Rectangle(){ x = 1; y = 1; }
    public void draw() {
        System.out.println("Rectangle Draw");
    }
}

class Triangle extends Shape {
    public int base, height;
    Triangle(){ x = 2; y = 2; }
    public void draw() {
        System.out.println("Triangle Draw");
    }
}

class Circle extends Shape {
    public int radius;
    Circle(){ x = 3; y = 3; }
    public void draw() {
        System.out.println("Circle Draw");
    }
}
```

# 다형성과 매개 변수 예제

- Test.java (다형성이 없다면... ) 및 실행 결과

```
package main;

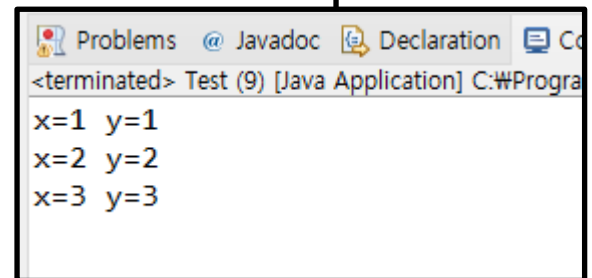
public class Test {
    public static void printLocation(Rectangle r) {
        System.out.println("x=" + r.x + " y=" + r.y);
    }

    public static void printLocation(Triangle t) {
        System.out.println("x=" + t.x + " y=" + t.y);
    }

    public static void printLocation(Circle c) {
        System.out.println("x=" + c.x + " y=" + c.y);
    }

    public static void main(String arg[]) {
        Rectangle r = new Rectangle();
        Triangle t = new Triangle();
        Circle c = new Circle();

        printLocation(r);
        printLocation(t);
        printLocation(c);
    }
}
```



<terminated> Test (9) [Java Application] C:\#Programs\Java\bin\java.exe  
x=1 y=1  
x=2 y=2  
x=3 y=3

# 다형성과 매개 변수 예제

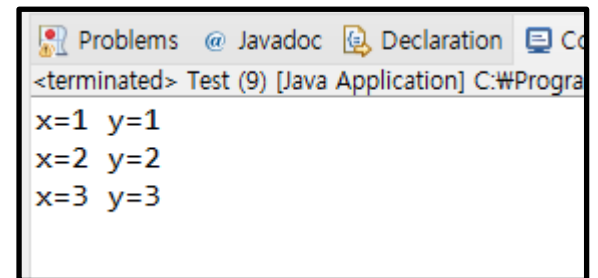
- Test.java (다형성을 활용하면... ) 및 실행 결과

```
package main;

public class Test {
    public static void printLocation(Shape s) {
        System.out.println("x=" + s.x + " y=" + s.y);
    }

    public static void main(String arg[]) {
        Rectangle r = new Rectangle();
        Triangle t = new Triangle();
        Circle c = new Circle();

        printLocation(r);
        printLocation(t);
        printLocation(c);
    }
}
```

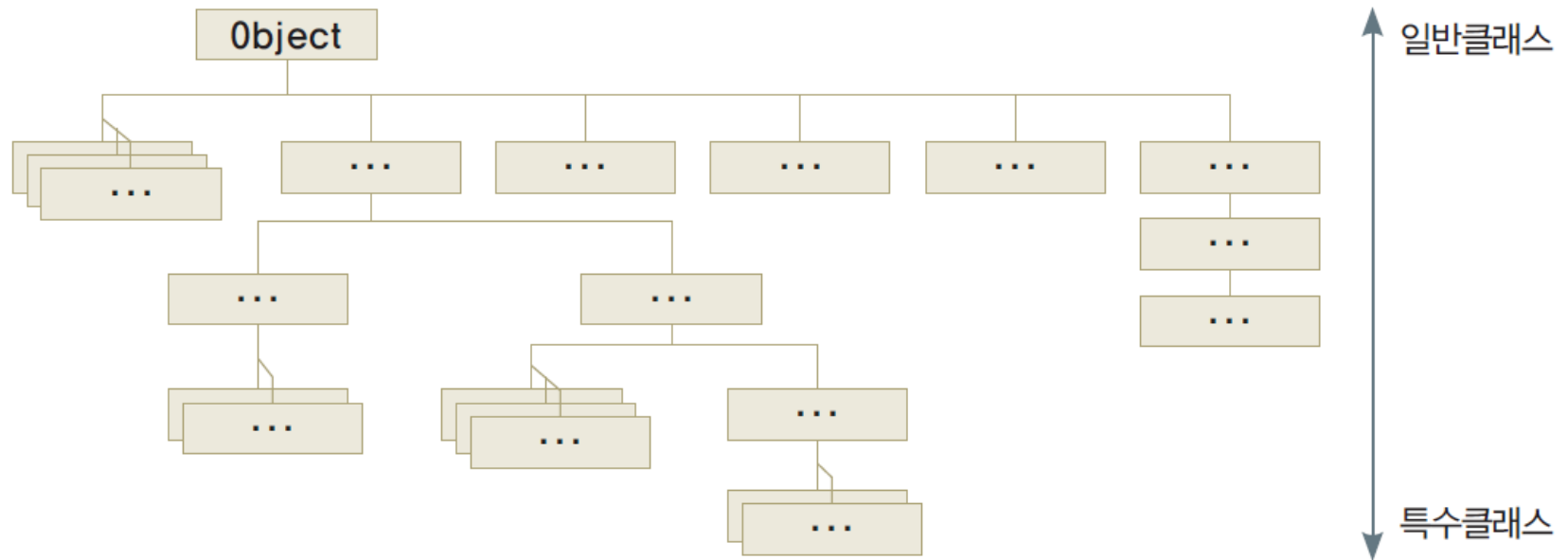


The screenshot shows a console window titled "Problems @ Javadoc Declaration Co" with the following output:

```
<terminated> Test (9) [Java Application] C:\#Progra
x=1 y=1
x=2 y=2
x=3 y=3
```

# Object 클래스

- 자바 클래스 계층 구조에서 맨 위에 위치하는 클래스
- java.lang 패키지에 포함되어 있음

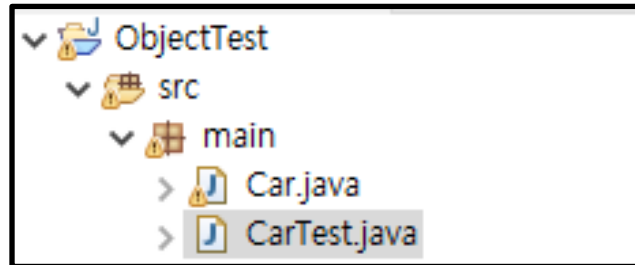


# Object 클래스의 메소드

메소드	설명
<code>Object clone( )</code>	객체 자신의 복사본을 생성하여 반환한다.
<code>boolean equals(Object obj)</code>	obj가 현재 객체와 같은지를 반환한다.
<code>void finalize()</code>	사용되지 않는 객체가 제거되기 직전에 호출된다.
<code>class getClass( )</code>	실행 시간에 객체의 클래스 정보를 반환한다.
<code>int hashCode( )</code>	객체에 대한 해쉬 코드를 반환한다.
<code>String toString( )</code>	객체를 기술하는 문자열을 반환한다.

# Object 클래스 예제

- 프로젝트 구조



# Object 클래스 예제

- Car.java

```
package main;

public class Car {
    private String model;
    private int years;
    private int kMs;
}
```



# Object 클래스 예제

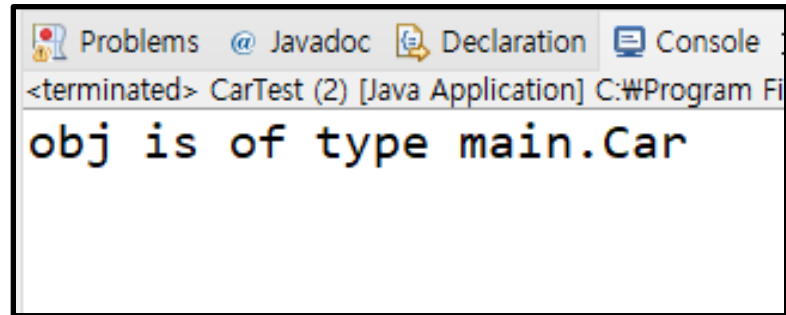
- CarTest.java

```
package main;

public class CarTest {
    public static void main(String[] args) {
        Car obj = new Car();
        System.out.println("obj is of type " + obj.getClass().getName());
    }
}
```

# Object 클래스 예제

- CarTest.java 실행 결과

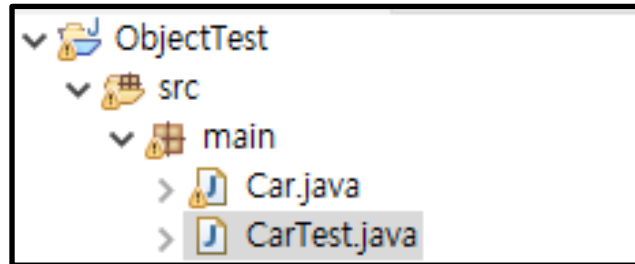


The screenshot shows a console window from an IDE. The title bar includes tabs for 'Problems', 'Javadoc', 'Declaration', and 'Console'. The console text reads: '<terminated> CarTest (2) [Java Application] C:\Program Fi' followed by the output line 'obj is of type main.Car'.

```
<terminated> CarTest (2) [Java Application] C:\Program Fi  
obj is of type main.Car
```

# instanceof 예제

- 프로젝트 구조



# instanceof 예제

- Car.java

```
package main;

public class Car {
    private String model;
    private int kMs = 70000;

    public int getkMs() { return kMs; }
}

class AirPlane {
    private String model;
    private int maxSpeed = 900;

    public int getMaxSpeed() { return maxSpeed; }
}

class Ship {
    private String model;
    private int length = 50;

    public int getLength() { return length; }
}
```

# instanceof 예제

- CarTest.java

```
package main;

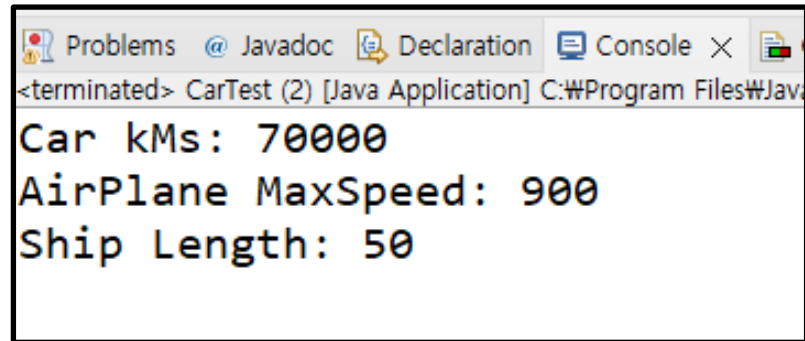
public class CarTest {
    public static void printInfo(Object obj) {
        if(obj instanceof Car)
            System.out.println("Car kMs: " + ((Car)obj).getkMs());
        if(obj instanceof AirPlane)
            System.out.println("AirPlane MaxSpeed: " + ((AirPlane)obj).getMaxSpeed());
        if(obj instanceof Ship)
            System.out.println("Ship Length: " + ((Ship)obj).getLength());
    }

    public static void main(String[] args) {
        Car obj1 = new Car();
        AirPlane obj2 = new AirPlane();
        Ship obj3 = new Ship();

        printInfo(obj1);
        printInfo(obj2);
        printInfo(obj3);
    }
}
```

# instanceof 예제

- CarTest.java 실행 결과

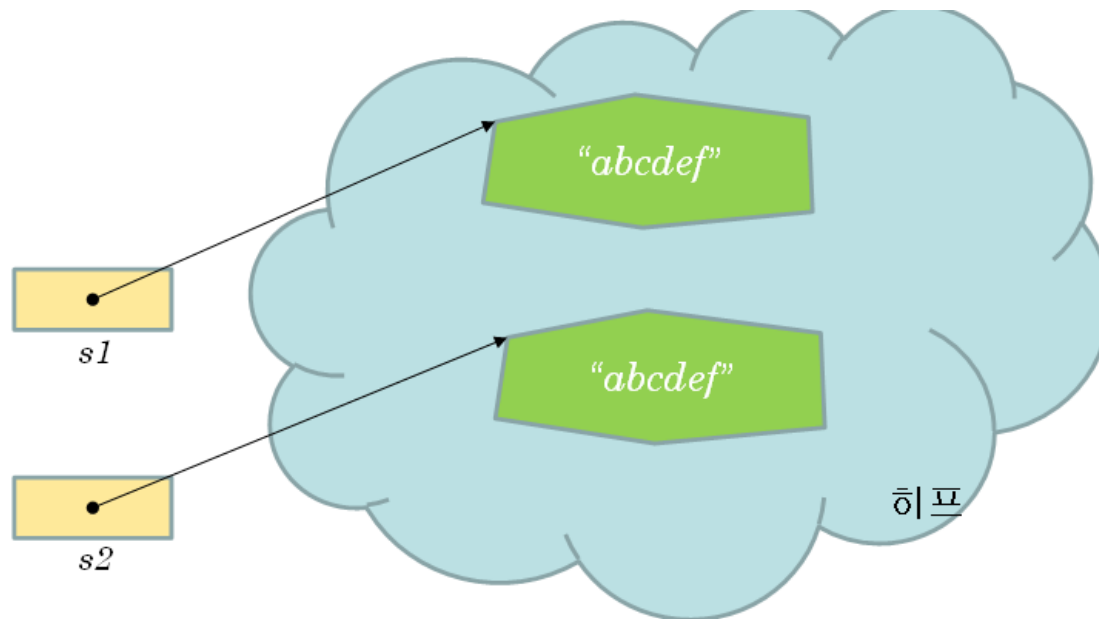


The screenshot shows a Java IDE console window with the following tabs: Problems, Javadoc, Declaration, and Console. The console output is as follows:

```
<terminated> CarTest (2) [Java Application] C:\Program Files\Java\
Car kMs: 70000
AirPlane MaxSpeed: 900
Ship Length: 50
```

# Object.equals() 메소드 오버라이딩

- Object 의 equals() 는 비교를 위해 == 사용
  - 객체 주소가 동일한지를 검사하여 true 또는 false 를 리턴
- String 객체에 대해서 내용이 같더라도 저장 주소가 다른 경우 false 를 리턴
  - String 클래스에서는 equals() 메소드 오버라이딩을 수행



# Object.equals() 메소드 오버라이딩

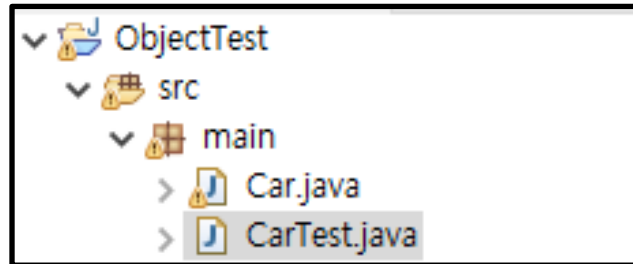
- Object 의 equals() 는 비교를 위해 == 사용
  - 객체 주소가 동일한지를 검사하여 true 또는 false 를 리턴
- String 객체에 대해서 내용이 같더라도 저장 주소가 다른 경우 false 를 리턴
  - String 클래스에서는 equals() 메소드 오버라이딩을 수행
  - String.class

```
public boolean equals(Object anObject) {  
    if (this == anObject) {  
        return true;  
    }  
    return (anObject instanceof String aString)  
        && (!COMPACT_STRINGS || this.coder == aString.coder)  
        && StringLatin1.equals(value, aString.value);  
}
```



# Object.equals() 메소드 오버라이딩 예제

- 프로젝트 구조



# Object.equals() 메소드 오버라이딩 예제

- Car.java

```
package main;

public class Car {
    private String model;
    private int kMs = 70000;

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

    public int getkMs() { return kMs; }

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

# Object.equals() 메소드 오버라이딩 예제

- CarTest.java

```
package main;

public class CarTest {
    public static void main(String[] args) {
        Car firstCar = new Car("Genesis");
        Car secondCar = new Car("Genesis");

        if (firstCar.equals(secondCar)) {
            System.out.println("They are same.");
        } else {
            System.out.println("They are not same.");
        }
    }
}
```

# IS-A 관계

---

- is-a 관계: “~은 ~이다” 와 같은 관계
- **is-a 관계**인 경우 **상속**으로 모델링
  - 자동차는 탈것이다 (Car is a Vehicle): 자동차는 탈것의 자식 클래스
  - 강아지는 동물이다 (Dog is a Animal): 강아지는 동물의 자식 클래스
- is-a 관계가 성립하지 않는다면 상속으로 모델링 하지 말 것
  - 책은 도서관이다? (Book is a Library?): 책은 도서관의 자식 클래스가 아님
  - 삼겹살은 아침식사다? (Three slices are breakfast?): 애매하다...

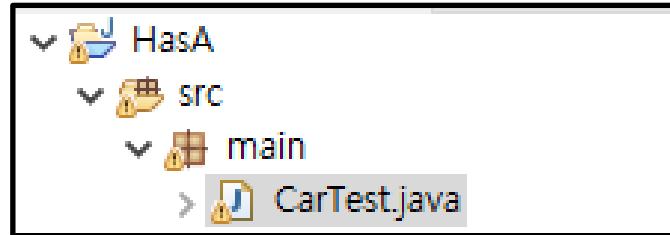
# HAS-A 관계

---

- has-a 관계: “~은 ~을 가지고 있다” 와 같은 관계
- **has-a 관계**인 경우 **내부클래스**로 모델링
  - 도서관은 책을 가지고 있다 (Library has a book)
  - 거실은 소파를 가지고 있다 (Living room has a sofa)
  - has-a 관계에서는 클래스안에 다른 클래스의 객체를 포함시킬 것

# HAS-A 관계 예제

- 프로젝트 구조



# HAS-A 관계 예제

- CarTest.java

```
package main;

public class CarTest {
    public static void main(String[] args) {
        Car car = new Car();
    }
}

class Vehicle {
    private String model;
    private String category;
}

class Engine {
    private String CC;
    private String Builder;
}

class Car extends Vehicle{
    private Engine engine;
}
```

# 종단 클래스

- final class
- **상속**을 시킬 수 없는 클래스
  - 종단 클래스가 필요한 이유는 클래스의 안정성과 일관성 유지 때문

```
public final class MyFinal {...}  
public class ThisIsWrong extends MyFinal {...}
```

```
141 public final class String  
142     implements java.io.Serializable, Comparable<String>, CharSequence,  
143                 Constable, ConstantDesc {  
144  
145     /**  
146      * The value is used for character storage.  
147      *  
148      * @implNote This field is trusted by the VM, and is a subject to  
149      * constant folding if String instance is constant. Overwriting this  
150      * field after construction will cause problems.  
151      *  
152      * Additionally, it is marked with {@link Stable} to trust the contents  
153      * of the array. No other facility in JDK provides this functionality (yet).  
154      * {@link Stable} is safe here, because value is never null.  
155      */  
156     @Stable  
157     private final byte[] value;  
158  
159     /**  
160      * The identifier of the encoding used to encode the bytes in  
161      * {@code value}. The supported values in this implementation are
```



# 종단 메소드

- final method
- **오버라이딩**을 시킬 수 없는 메소드
  - 예시로 바둑에서 게임 시작은 항상 "흑" 돌 플레이어가 먼저 해야 함

```
class Baduk {  
    enum BadukPlayer { WHITE, BLACK }  
  
    ...  
  
    final BadukPlayer getFirstPlayer() {  
        return BadukPlayer.BLACK;  
    }  
}
```

# 연습문제 1

- 페이지 14-17을 참고하여 Test.java 에러를 수정하자.

```
package main;

public class Test {
    public static void main(String arg[]) {
        Shape s1, s2;
        Rectangle r1, r2;

        s1 = new Shape();
        s2 = new Rectangle();

        //      r1 = new Shape(); // 에러!
        r2 = new Rectangle();

        s1.x = 0;
        s1.y = 0;
        //      s1.width = 100; // 에러!
        //      s1.height = 100; // 에러!

        r2.x = 0;
        r2.y = 0;
        r2.width = 100; // OK
        r2.height = 100; // OK
    }
}
```

## 연습문제 2

- 페이지 32-36을 참고하여 다형성을 활용한 메서드 파라미터 구현을 해보자.
  - (자유롭게) 부모 클래스 1개와 이를 상속하는 자식 클래스 4개를 만들자.  
(예: Dog - Beagle, Jindo, Chihuahua, ShigolJabjong)
  - 메서드 파라미터로는 부모 클래스 객체를 정의하자.
  - 메서드 인수로는 자식 클래스 객체를 넘겨주자.
  - 전달받는 자식 클래스 객체의 특성에 따라 동적 메서드 호출이 이루어지도록 하자.  
(예: Beagle 을 전달받으면 bark() 메서드에 의해 '왈왈, 다 물어뜯는다!' 출력)  
(예: ShigolJabjong 을 전달받으면 bark() 메서드에 의해 '끼잉끼잉?' 출력)



[참조] <https://blog.naver.com/miampuzzy/222211644882>

**감사합니다! XD**

