9. 클래스와 객체 활용

this

- **클래스 내에서 자기 자신을 가리키는 레퍼런스**
  Static 으로 선언된 메소드에서는 사용될 수 없다.

- **두 가지 목적으로 사용**
  - 자기 자신의 멤버 필드나 메소드를 명확히 지시하기 위해서 사용
  - 객체 전체를 함수의 매개변수로 전달해야 하는 경우에 사용

- **예제**

```
void a (int a) {
    this.a = a;
    b.doJob(this);
}
```

this

This()

- 생성자에서 사용되는 this()는 클래스의 다른 생성자를 호출
- 생성자가 많은 경우 매개 변수에 의해 생성자가 구분
- 생성자에서 this()는 맨 처음 부분에 나와야 한다.

예

```
public Circle() {
   this(0);
}

public Circle(int r) {
   this.r = r;
}
```

생성자 - this()

```
public class TestPoint {
    public static void main(String[] args)
   {
        Point p1 = new Point( 10, 2 );
       Point p2 = new Point();
       System.out.println
      (“(“+p1.getX()+”,”+p1.getY()+”)”);
        System.out.println(…);
    }
}
```

```
public class Point {
    private int x, y;
    public Point(int x, int y) {… }
    public Point()
    {
        x = 0;
       y = 0;
    }
    ….
}
```

```
public class TestPoint {
    public static void main(String[] args)
   {
        Point p1 = new Point( 10, 2 );
       Point p2 = new Point();
       Point p3 = new Point( 3 );
        // x = 3, y = 기본값 0

        //print p1, p2, p3
    }
}
```

```
   public Point(int x)
   {
        this.x = x;
       this.y = 0;
    }
or
    public Point(int x)
    {  this();
       this.x = x;
    }
```

생성자

❖ classname()

- 아무 생성자도 만들지 않았을 경우
  - java가 기본 생성자를 만들어 줌

- 프로그래머가 하나이상의 생성자를 만들었을 경우
  - java는 기본 생성자를 만들어 주지 않음

  → 입력 파라매터가 없는 기본 생성자를 사용하기 위해서 사용자가 직접 만들어야 함

Overloading


```
public class TestPoint {
    public static void main(String[] args)
   {
        Point p1 = new Point( 10, 2 );
        p1.set(30, 10);
      // p1.set(30);

        System.out.println
        ("("+p1.getX()+","+p1.getY()+")");}
    }
}
```

Methods:
set()
1. 두 정수(x, y)가 입력 되면 → (x,y)
2. 하나의 정수 (x)가 입력되면
   → (x, 0)

```
public class Point {
    public void set(int x, int y)
    { this.x = x;
      this.y = y;
    }
…. public void set(int x)
    {
        this.x = x; this.y = 0;
    }
}
```

오버로딩(overloading)

❖ 오버로딩

- 동일한 연산자가 자료의 타입에 따라 다른 작업을 수행하는 것
- 함수 오버로딩
  - 동일한 이름을 가지는 함수가 여러 개 존재 가능
  - 내부적으로는 다른 작업을 하더라도 의미 면에서 같다면 동일한 함수 이름을 사용
  - 한 클래스 내에서 함수 이름은 동일, 함수의 매개변수는 달라야 함
  - 함수 signature = 함수 이름 + 매개 변수 개수 + 매개 변수 타입

오버로딩 규칙

- 1. 메소드가 같은 클래스 혹은 상위 클래스에 존재해야합니다.
- 2. 메소드의 이름이 같아야 합니다.
- 3. 메소드의 파라미터 개수가 다르거나, 데이터형이 달라야 합니다.
- 4. 리턴형은 같아도 되고 달라도 됩니다.

오버로딩 예제

```
 1  class Overloading {
 2     public void say() {
…
 6     public void say(String msg) {
…
10     public void say(String msg, int n) {
…
16     public static void main(String args[]) {
17        Overloading  a = new Overloading();
18        a.say();
19        a.say("How are you ?");
20        a.say("I am fine.", 3);
```

문제 1

```
class MyOverloading{
    void methodA(){
        System.out.println("methodA()");
    }
    int methodA(){
        int i = 0;
        return i;
    }

    public static void main(String[] args){
        MyOverloading mo = new MyOverloading();
        mo.methodA();
        mo.methodA();
    }
}
```

문제 1

- 이 소스에서도 methodA() 는 두개 존재하는데, 이번에는 둘다 전달인자가 하나도 없습니다. 리턴형이 다르기는 하지만, 리턴형은 메소드를 구분하는 속성이 될 수 없기 때문에, 같은 이름의 메소드가 중복선언되었다는 의미로 컴파일 에러가 발생합니다.

문제 2

Which of the following is a legal return type of a method overloading the following method?

```
public void add(int a, int b){

}
```

a) void
b) int
c) Can be anything

문제 2

- **해설**

- **메소드 오버로딩은 리턴형과 무관합니다. 위의 메소드를 위와 같은 파라미터 수와 데이터형으로 오버로딩 할 때는 어떤 리턴형이 와도 오버로딩이 되지 않습니다.**

- **답은 c) 입니다.**

문제 3

```
public class MethodOver {
    private int x, y;
    private float z;
    public void setVar(int a, int b, float c) {
        x=a;
        y=b;
        z=c;
}
```

which two overload the setVar method?(choose two)

a)
```
void setVar(int a, int b, float c) {
    x=a;
    y=b;
    z=c;
}
```

문제 3

```
b) public void setVar(int a, float c, int b) {
    setVar(a, b, c)
}
c) public void setVar(int a, float c, int b) {
    this(a, b, c)
}
d) public void setVar(int a, float b) {
    x=a;
    y=b;
}
e) public void setVar(int ax, int by, float cz) {
    x=ax;
    y=by;
    z=cz;
}
```

문제 3

- **해설 & 답**

- **a)는 파라미터의 형과 개수가 일치하기 때문에 오버로딩이 될 수 없습니다.**
  b) 는 데이터형이 다르기 때문에 오버로딩이 될 수 있습니다.
  c)는 오버로딩 규칙에는 맞지만, 이 클래스에는 존재하지 않는 생성자를 호출하고 있습니다.
  d) 는 파라미터의 개수가 다르기 때문에 오버로딩이 될 수 있고,
  e) 처럼 파라미터 개수나 데이터형은 같은데 전달인자의 변수명만을 바꾸는 것은 의미가 없습니다.

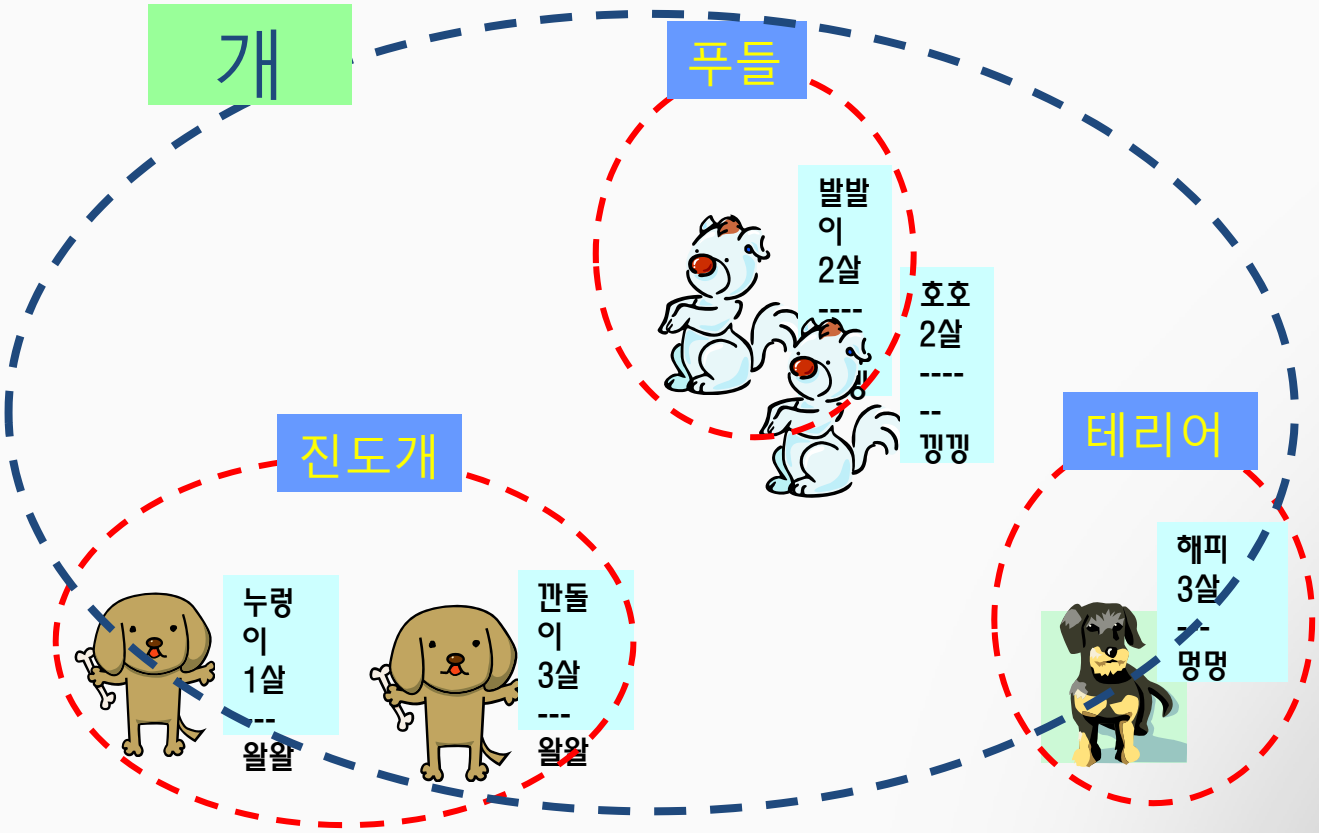
- **답은 b) 와 d)입니다.**

Transient 변수

- **Transient 변수**
  - 멤버변수가 객체의 영속적인 상태의 일부분으로 포함되지 않는다.
- **예**
  - 다음의 Point 클래스가 시스템 서비스에 의해 영속적인 메모리(하드디스크, 테이프 등)에 저장될 때 transient 부분을 제외한 x, y 부분만 저장됨을 의미

```
Class Point {
        Int x, y;
        Transient float rho, theta;
}
```

클래스 상속(inheritance)



클래스 상속(inheritance)

Class Name: Jindo

Variable:
- name
- age

Methods:
- bark // 왈왈

Class Name: Poodle

Variable:
- name
- age

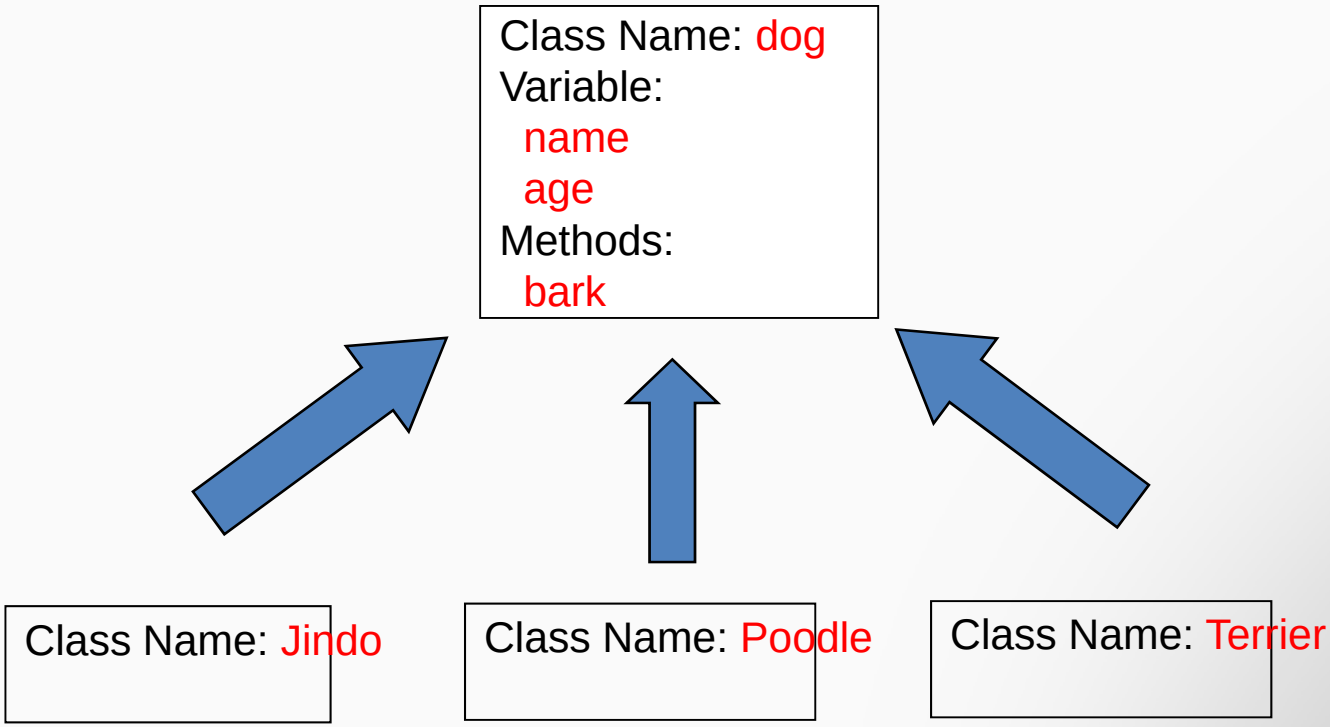
Methods:
- bark // 낑낑

Class Name: Terrier

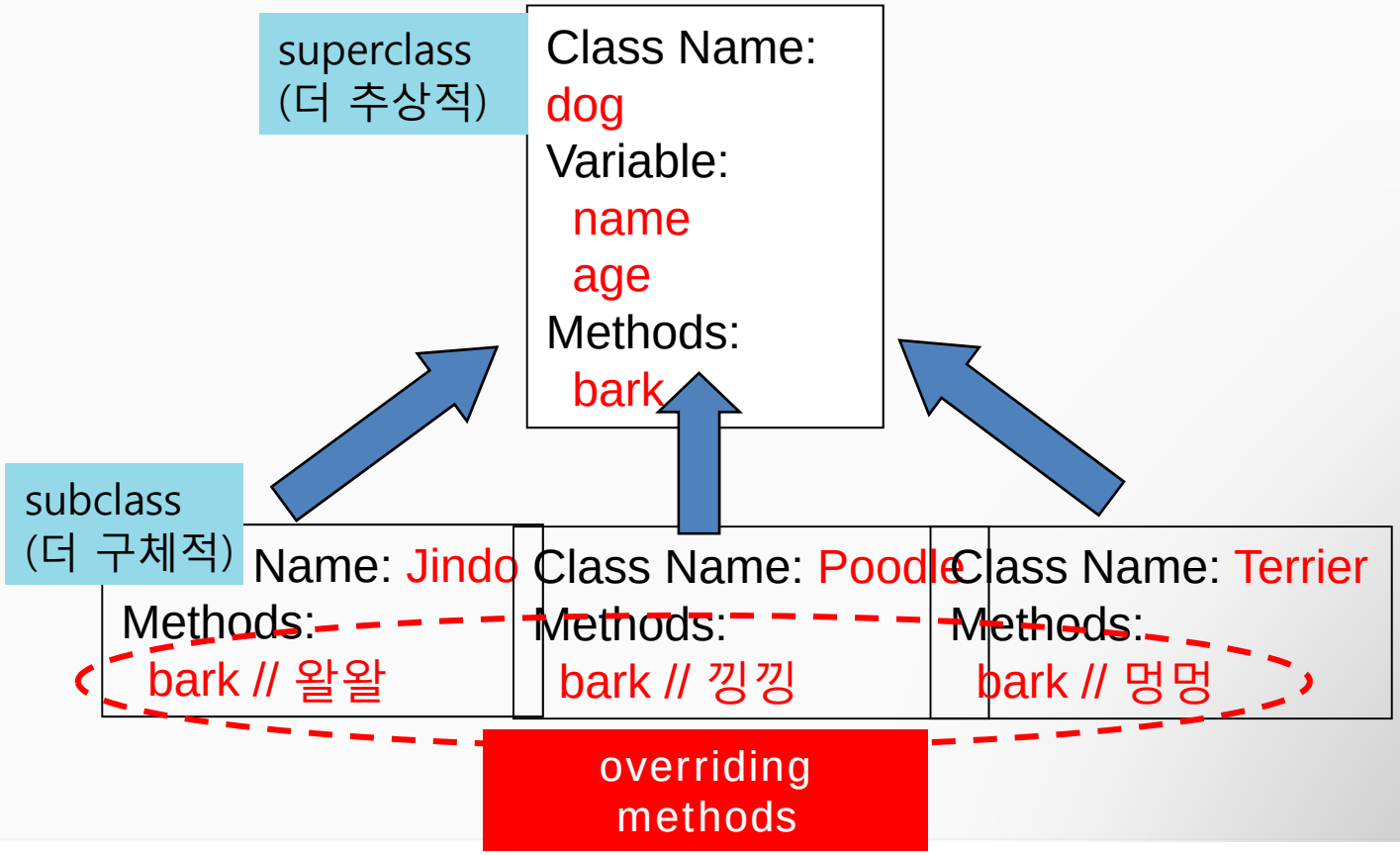
Variable:
- name
- age

Methods:
- bark // 멍멍

추상화


Class Name: dog
Variable:
name
age
Methods:
bark
Class Name: Jindo
Class Name: Poodle
Class Name: Terrier

superclass & subclass



superclass & subclass

```
public class Dog {
        private String name;
        private int age;
           public Dog()
           {
              System.out.println("Dog 생성자");
           }
        public void bark()
           {
              System.out.println("기본소리")
           }
}
```

```
public class Jindo extends Dog {

           public Jindo()
           {
              super();
              System.out.println("Jindo 생성자");
           }
        public void bark()
           {
            System.out.println("왈왈");
              super.bark();
           }
}
```

```
public class TestDog {
           public static void main(String[] args)
           {
              Dog myDog = new Dog();
              myDog.bark();
              Jindo myJin = new Jindo();
              myJin.bark();
           }
}
```

```
Dog myDog2 = myJin;
Dog myDog3 = new Jindo();
myDog2.bark()
```

superclass & subclass

```
public class Dog {
        protected String name;
        protected int age;

           public Dog()
           {
                    name="";
                    age = 0;
           }
           public Dog(String name, int age)
           {
                    this.name = name;
                    t
           }
        public void
           {

           }
}
```

```
public class TestDog {
        public static void main(String[] args)
        {
            Dog myDog = new Jindo("검둥이", 3);
            System.out.println(myDog.getAge());
        }
}
```

```
public class Jindo extends Dog {

        public Jindo()
        {
                name = "";
                age= 0;
        }
        public Jindo(String name, int age)
        {
                super.name = name;
                super.age = age;
        }

                                rk()

                                ystem.out.println("왈왈")

                                Age()
                                rn age;

}
```

JAVA에서의 상속

- **extends 예약어를 사용하여 직속 상위 클래스를 확장**
  - 부모가 가지는 모든 멤버변수와 함수를 상속 받음
  - 부모의 생성자는 상속 받지 않음
  - private 멤버 변수는 상속은 받으나 바로 접근을 할 수 없음
  - private 함수는 자식 클래스에서 호출 할 수 없으므로 상속 되지 않는다고 볼 수 있음
- **자바는 단일 상속**
  - C++과 달리 다중 상속 지원하지 않음
- **모든 자바 클래스는 Object의 하위 클래스**
  - 명시적으로 상속 받지 않아도 자동적으로 Object 상속
- **객체를 생성하면 객체에 Object 클래스를 포함한 모든 상위 클래스가 포함**

상속의 표현

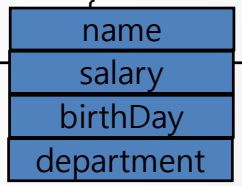
❖ **extends 예약어를 사용**

```
[modifiers] class ClassName
                        [ extends SuperClassName ]
            [ implements InterfaceName, . . . ]
{
    classBody  // 멤버변수와 함수
}
```

```
public class Employee {
        private String name;
        private double salary;
        private Date birthDay;

        public Employee () { }
        public String getInformation() { … }
        public double getSalary() { … }
}
```

```
public class Manager  extends Employee {
        private String department;

        public Manger() { …
        }

        public String getDepartment() {
        }
…
}
```

new Manager()



Manger의 object

상속에서의 생성자

- **자식의 생성자에서는 반드시 부모의 생성자를 호출**
  - super() 사용
  - 매개변수를 이용해서 부모의 여러 생성자 중 의미에 맞는 생성자 호출

- **반드시 자식의 생성자 첫 라인에서 부모의 생성자를 호출해야 됨**

- **명시적으로 부모의 생성자 호출 하지 않으면 부모의 디폴트 생성자 호출 됨**
  - 부모의 생성자 호출이 없으면 super()가 자식의 생성자에 있는 것과 같음

예제 1

```
public class Employee {
    private String name;
    private double salary;
    private String birthDay;

    public Employee () {
        name = "몰라";
        salary = 0;
        birthDay = "몰라";
    }
    public String getName() { return name; }
    public double getSAlary() { return salary; }
    public String getBirthDay() { return birthDay; }

    public String toString() {
        return name + "\t" + salary + "\t" + birthDay;
    }
}
```

```
public class Manager  extends Employee {
    private String department;

    public Manger() {
        super();
        department = "몰라";
    }

    public String toString() {
       return getName() + "\t" + getSalary() + "\t" +
                getBirthDay() + "\t" + department;
    }
}
```

```
public static void main(String[] arg) {
    Manger m1 = new Manger();
    System.out.println(m1);
}
```

output

```
몰라    0.0    몰라    몰라
```

예제2

```
public class Employee {
    private String name;
    private double salary;
    private String birthDay;

    public Employee () {
        name = "몰라";
        salary = 0;
        birthDay = "몰라";
    }
    public String getName() { return name; }
    public double getSAlary() { return salary; }
    public String getBirthDay() { return birthDay; }

    public String toString() {
        return name + "\t" + salary + "\t" + birthDay;
    }
}
```

```
public class Manager  extends Employee {
    private String department;

    public Manger() {
        //super(); //default constructor 자동 호출!!
        department = "몰라";
    }

    public String toString() {
       return getName() + "\t" + getSalary() + "\t" +
                getBirthDay() + "\t" + department;
    }
}
```

```
public static void main(String[] arg) {
    Manger m1 = new Manger();
    System.out.println(m1);
}
```

output

```
몰라    0.0    몰라    몰라
```

예제3

```
public class Employee {
    private String name = “”;
    private double salary;
    private String birthDay;
    public Employee () {
        name = “몰라”;
        salary = 0;
        birthDay = “몰라”;
    }
    public Employee (String n, double s, String b) {
        name = n;
        salary = s;
        birthDay = b;
    }
    public String getName() { return name; }
    public double getSAlary() { return salary; }
    public String getBirthDay() { return birthDay; }
    public String toString() {
        return name + “\t” + salary + “\t” + birthDay;
    }
}
```

```
public class Manager  extends Employee {
    private String department;

    public Manger() {
        super();
        department = “몰라”;
    }

    public Manger(String n, double s, String b,
String d) {
        super(n, s, b); // 매개 변수로 결정
        department = “몰라”;
    }

    public String toString() {
      return getName() + “\t” + getSalary() + “\t”
+
                getBirthDay()       + “\t” +
department;
```

```
public static void main(String[] arg) {
    Manger m1 = new Manger();
    System.out.println(m1);
    Manger m2 = new Manger(“길동”, 10000, “19
    System.out.println(m2);
}
```

output

```
몰라        0.0         몰라        몰라
길동        10000.0     1986-1-1    무학
```

예제 4

```
public class Employee {
    private String name = “”;
    private double salary;
    private String birthDay;
    public Employee () {
        name = “몰라”;
        salary = 0;
        birthDay = “몰라”;
    }
    public Employee (String n, double s, String b)
{
        name = n;
        salary = s;
        birthDay = b;
    }
    public String toString() {
       return name + “\t” + salary + “\t” + birthDay;
    }
}
```

```
public class Manager  extends Employee {
    private String department;

    public Manger() {
        department = “몰라”;
        super(); // compile error
    }

    public Manger(String n, double s, String b,
String d) {
          department = “몰라”;
          super(n, s, b); // compile error
    }

    public String toString() {
      return getName() + “\t” + getSalary() + “\t” +
                getBirthDay()       + “\t” +
department; }
}
```

super

super()

- 부모의 생성자를 호출
- 매개변수에 따라서 부모의 다른 생성자 호출 가능
- 반드시 자식의 생성자의 첫 번째 라인에서 사용
- 다른 함수에서는 사용 불가

super

- 부모 클래스로부터 물려 받은 멤버 변수나 함수를 가르킴
- overriding 된 데이터 멤버나 함수를 이용 시 사용
- this 와 마찬가지로 static 함수에선 사용 불가

“Learn by studying examples”
“Learn by hand programming”