

## □ 응용 프로그래밍

- (1) **[생성형 AI활용]** 제시된 결과처럼 실행하려고 합니다. 아래의 프로그램에 존재하는 문제는 무엇인가요? 문제를 해결할 수 있는 방법과 결과를 제시하세요.

```
class ClassA {  
    public void methodOne(int i) { }  
    public void methodTwo(int i) { System.out.println("ClassA : A2"); }  
    public static void methodThree(int i) { }  
    public static void methodFour(int i) { System.out.println("ClassA : A4"); }  
}  
  
class ClassB extends ClassA {  
    public static void methodOne(int i) { }  
    public void methodtwo(int i) { System.out.println("ClassB : B2"); }  
    public void methodThree(int i) { }  
    public static void methodFour(int i) { System.out.println("ClassB :B4"); }  
}  
  
public class Test {  
    public static void main(String args[]) {  
        ClassA aa = new ClassA();  
        ClassB bb = new ClassB();  
        ClassA ab = new ClassB();  
        ClassB.methodFour(0);  
        ClassA.methodFour(0);  
        aa.methodTwo(0);  
        bb.methodTwo(0);  
        ab.methodTwo(0);  
    }  
}
```

```
ClassB :B4  
ClassA : A4  
ClassA : A2  
ClassB : B2  
ClassB : B2
```

생성형 AI 플랫폼	chatGPT
질문	<p>1.</p> <p>(1) [생성형 AI활용] 제시된 결과처럼 실행하려고 합니다. 아래의 프로그램에 존재하는 문제는 무엇인가요? 문제를 해결할 수 있는 방법과 결과를 제시하세요.</p> <pre>class ClassA {     public void methodOne(int i) { }     public void methodTwo(int i) { System.out.println("ClassA : A2"); }     public static void methodThree(int i) { }     public static void methodFour(int i) { System.out.println("ClassA : A4"); } }  class ClassB extends ClassA {     public static void methodOne(int i) { }     public void methodtwo(int i) { System.out.println("ClassB : B2"); }</pre>

	<pre>         public void methodThree(int i) { }         public static void methodFour(int i) { System.out.println("ClassB :B4"); }     }     public class Test {         public static void main(String args[]) {             ClassA aa = new ClassA();             ClassB bb = new ClassB();             ClassA ab = new ClassB();             ClassB.methodFour(0);             ClassA.methodFour(0);             aa.methodTwo(0);             bb.methodTwo(0);             ab.methodTwo(0);         }     } </pre> <p>실행결과</p> <p>ClassB : B4</p> <p>ClassA : A4</p> <p>ClassA : A2</p> <p>ClassB : B2</p> <p>ClassB : B2</p> <p>2.</p> <p>java: static methods cannot be annotated with @Override</p>
<p>답변</p>	<p>1.</p> <p>해당 코드에는 몇 가지 문제와 개선할 수 있는 부분들이 있습니다. 하나씩 살펴보겠습니다:</p> <p>### 문제점:</p> <p>1. <b>**메소드 오버라이딩과 오버로딩의 혼동**:</b></p> <ul style="list-style-type: none"> <li>- `methodOne`과 `methodThree` 메소드에서 오버라이딩과 오버로딩의 차이가 명확하지 않습니다. 예를 들어, `ClassA`의 `methodOne`은 인스턴스 메소드인데, `ClassB`에서는 `static` 메소드로 정의되었습니다. 이는 <b>**오버라이딩**</b>이 아니라 <b>**오버로딩**</b>입니다.</li> <li>- `methodThree`도 `ClassA`에서 `static`으로 정의되었는데, `ClassB`에서는 인스턴스 메소드로 정의되었습니다. 이 또한 잘못된 오버라이딩입니다. `static` 메소드에는 오버라이딩이 아닌 숨기기(하이딩)가 발생합니다.</li> </ul> <p>2. <b>**메소드 명의 일관성 문제**:</b></p> <ul style="list-style-type: none"> <li>- `ClassB`에서 `methodtwo`라는 메소드가 정의되어 있는데, `ClassA`에서 `methodTwo`라고 대소문자를 구분하여 정의된 메소드와 이름이 일치하지 않으므로 오버라이딩이 아니라 새로운 메소드가 됩니다. (Java에서는 메소드 이름 대소문자를 구분합니다)</li> </ul>

### 문제를 해결하기 위한 방법:

1. **\*\*오버라이딩을 제대로 활용\*\***:

- `methodOne`, `methodThree`의 경우, 부모 클래스 `ClassA`에서 정의된 메소드를 `ClassB`에서 올바르게 오버라이딩해야 합니다. `static` 메소드는 오버라이딩이 불가능하고, 대신 숨길 수는 있습니다. 이를 수정해야 합니다.

2. **\*\*메소드 이름 일관성 유지\*\***:

- `methodTwo`의 대소문자를 일치시켜야 합니다. 즉, `ClassB`에서 `methodTwo`로 정의하고, `ClassA`에서의 메소드 이름도 대소문자가 정확히 일치하도록 수정해야 합니다.

### 수정된 코드:

```
```java
class ClassA {
    public void methodOne(int i) { System.out.println("ClassA : A1"); }
    public void methodTwo(int i) { System.out.println("ClassA : A2"); }
    public static void methodThree(int i) { System.out.println("ClassA : A3"); }
    public static void methodFour(int i) { System.out.println("ClassA : A4"); }
}

class ClassB extends ClassA {
    @Override
    public void methodOne(int i) { System.out.println("ClassB : B1"); } // 오버라이딩
    @Override
    public void methodTwo(int i) { System.out.println("ClassB : B2"); } // 오버라이딩
    @Override
    public static void methodThree(int i) { System.out.println("ClassB : B3"); } // static 메
소드 숨기기
    public static void methodFour(int i) { System.out.println("ClassB : B4"); } // static 메소
드 숨기기
}

public class Test {
    public static void main(String args[]) {
        ClassA aa = new ClassA();
        ClassB bb = new ClassB();
        ClassA ab = new ClassB();

        ClassB.methodFour(0); // ClassB : B4
        ClassA.methodFour(0); // ClassA : A4
        aa.methodTwo(0);      // ClassA : A2
        bb.methodTwo(0);      // ClassB : B2
    }
}
```

```
ab.methodTwo(0);      // ClassB : B2
    }
}
...
```

### 설명:

1. `methodOne`과 `methodTwo`는 이제 `@Override` 어노테이션을 사용하여 부모 클래스에서 정의된 메소드를 정확하게 오버라이딩하고 있습니다.
2. `methodThree`는 `static` 메소드이므로 `ClassB`에서 해당 메소드를 숨기도록 했습니다. `static` 메소드는 오버라이딩이 아니라 숨기는 방식으로 처리됩니다.
3. `methodTwo`의 이름을 대소문자 일관성에 맞게 수정하여 오버라이딩이 가능하도록 했습니다.

### 실행 결과:

```
```plaintext
ClassB : B4
ClassA : A4
ClassA : A2
ClassB : B2
ClassB : B2
...
```

이렇게 수정된 코드는 `ClassA`와 `ClassB` 간의 올바른 상속 관계와 메소드 오버라이딩을 유지하며, 기대하는 결과를 출력합니다.

2.

맞습니다, `static` 메소드에 `@Override` 어노테이션을 사용하는 것은 잘못된 사용법입니다. `@Override`는 인스턴스 메소드에만 사용되며, `static` 메소드는 오버라이딩이 아니라 **숨기기(hiding)**로 처리됩니다. 그래서 `static` 메소드는 `@Override` 어노테이션을 사용할 수 없습니다.

이를 해결하려면, `@Override` 어노테이션을 **인스턴스 메소드**에만 사용하고, `static` 메소드는 그저 이름을 동일하게 하여 **숨기기**만 해야 합니다.

수정된 코드에서 `methodThree`와 `methodFour`를 `static`으로 처리할 때, `@Override` 어노테이션을 제거해야 합니다.

### 수정된 코드:

```
```java
class ClassA {
    public void methodOne(int i) { System.out.println("ClassA : A1"); }
    public void methodTwo(int i) { System.out.println("ClassA : A2"); }
}
```

```

        public static void methodThree(int i) { System.out.println("ClassA : A3"); }
        public static void methodFour(int i) { System.out.println("ClassA : A4"); }
    }

    class ClassB extends ClassA {
        @Override
        public void methodOne(int i) { System.out.println("ClassB : B1"); } // 오버라이딩
        @Override
        public void methodTwo(int i) { System.out.println("ClassB : B2"); } // 오버라이딩
        public static void methodThree(int i) { System.out.println("ClassB : B3"); } // static 메
소드 숨기기
        public static void methodFour(int i) { System.out.println("ClassB : B4"); } // static 메소
드 숨기기
    }

    public class Test {
        public static void main(String args[]) {
            ClassA aa = new ClassA();
            ClassB bb = new ClassB();
            ClassA ab = new ClassB();

            ClassB.methodFour(0); // ClassB : B4
            ClassA.methodFour(0); // ClassA : A4
            aa.methodTwo(0);      // ClassA : A2
            bb.methodTwo(0);      // ClassB : B2
            ab.methodTwo(0);      // ClassB : B2
        }
    }
    ...

```

### 주요 변경점:

- '@Override' 어노테이션을 'methodThree'와 'methodFour'의 'static' 메소드에서 제거했습니다.
- 'static' 메소드는 \*\*오버라이딩\*\*이 아니라 \*\*숨기기(hiding)\*\*가 적용되므로, '@Override'를 사용하지 않습니다.

### 실행 결과:

```

```plaintext
ClassB : B4
ClassA : A4
ClassA : A2
ClassB : B2
ClassB : B2

```

	<pre>'''</pre> <p>이제 코드가 정상적으로 실행되며, `static` 메소드 숨기기와 인스턴스 메소드 오버라이딩이 올바르게 처리됩니다.</p>
실행결과	<pre>ClassB : B4 ClassA : A4 ClassA : A2 ClassB : B2 ClassB : B2</pre>

(2) **[생성형 AI활용]** 다형성과 재정의의 활용하여 다음 조건에 따라 프로그램을 작성하고 테스트 하세요.

- Fruit class : me() 메소드는 실행문을 가지고 있지 않습니다
- Fruit 상속 받은 Apple class : me() 메소드에서 "나는 사과" 리턴
- Fruit 상속 받은 Grape class : me() 메소드에서 "나는 포도" 리턴  
save() 메소드에서 "저장할 없습니다" 리턴
- Fruit 상속 받은 Pear class : me() 메소드에서 "나는 배" 리턴
- Test class :
  - >> main() 메소드 : Fruit 타입의 배열을 생성해 Fruit, Apple, Grape, Pear 객체를 저장하고 printFruit() 호출  
Fruit [] fary = new Fruit [4];
  - >> printFruit() 메소드 : 매개변수로 전달된 객체에 대하여 me() 메소드를 호출합니다. 단, 객체가 Grape이면 "저장할 수 없습니다" 도 출력

```
나는 포도
저장할 수 없습니다
나는 사과
나는 포도
저장할 수 없습니다
나는 배
```

생성형 AI 플랫폼	chatGPT
질문	<p>(2) [생성형 AI활용] 다형성과 재정의의 활용하여 다음 조건에 따라 프로그램을 작성하고 테스트 하세요.</p> <ul style="list-style-type: none"> <li>- Fruit class : me() 메소드는 실행문을 가지고 있지 않습니다</li> <li>- Fruit 상속 받은 Apple class : me() 메소드에서 "나는 사과" 리턴</li> <li>- Fruit 상속 받은 Grape class : me() 메소드에서 "나는 포도" 리턴 save() 메소드에서 "저장할 없습니다" 리턴</li> <li>- Fruit 상속 받은 Pear class : me() 메소드에서 "나는 배" 리턴</li> <li>- Test class :           <ul style="list-style-type: none"> <li>&gt;&gt; main() 메소드 : Fruit 타입의 배열을 생성해 Fruit, Apple, Grape, Pear 객체를 저장하고 printFruit() 호출 Fruit [] fary = new Fruit [4];</li> <li>&gt;&gt; printFruit() 메소드 : 매개변수로 전달된 객체에 대하여 me() 메소드를 호출합니다. 단, 객체가 Grape이면 "저장할 수 없습니다" 도 출력</li> </ul> </li> </ul> <p>실행결과</p> <pre>나는 포도 저장할 수 없습니다 나는 사과 나는 포도</pre>

	<p>저장할 수 없습니다</p> <p>나는 배</p>
답변	<p>주어진 조건에 따라 Java 프로그램을 작성할 수 있습니다. 다형성과 메소드 재정의의 활용하여 `Fruit`, `Apple`, `Grape`, `Pear` 클래스를 구현하고, `Test` 클래스에서 객체 배열을 생성하여 각 객체에 대해 `me()` 메소드를 호출하도록 하겠습니다.</p> <p>### 단계별로 코드 구현:</p> <ol style="list-style-type: none"> <li>1. `Fruit` 클래스를 만들고 `me()` 메소드에 실행문을 넣지 않도록 합니다.</li> <li>2. `Apple`, `Grape`, `Pear` 클래스는 `Fruit` 클래스를 상속받으며, 각 클래스에서 `me()` 메소드를 재정의합니다.</li> <li>3. `Grape` 클래스에서 `save()` 메소드를 재정의하고, 해당 메소드가 호출될 경우 "저장할 수 없습니다"라는 메시지를 출력합니다.</li> <li>4. `Test` 클래스에서 `Fruit[]` 배열을 생성하고, `printFruit()` 메소드를 통해 각 객체에 대해 `me()` 메소드를 호출하도록 합니다. `Grape` 객체일 경우 "저장할 수 없습니다"도 출력합니다.</li> </ol> <p>### 전체 코드:</p> <pre> `java  // Fruit 클래스  class Fruit {      // me() 메소드는 실행문이 없습니다      public void me() { }      // save() 메소드 (기본 구현이 없으므로 필요 없을 경우 사용할 수 없음)      public String save() {          return "";     } } </pre>

```
}

}

// Apple 클래스는 Fruit 클래스를 상속받고 me() 메소드를 재정의합니다.

class Apple extends Fruit {

    @Override

    public void me() {

        System.out.println("나는 사과");

    }

}

// Grape 클래스는 Fruit 클래스를 상속받고 me()와 save() 메소드를 재정의합니다.

class Grape extends Fruit {

    @Override

    public void me() {

        System.out.println("나는 포도");

    }

    @Override

    public String save() {

        return "저장할 수 없습니다";

    }

}
```



---

// Pear 클래스는 Fruit 클래스를 상속받고 me() 메소드를 재정의합니다.

```
class Pear extends Fruit {
```

```
    @Override
```

```
    public void me() {
```

```
        System.out.println("나는 배");
```

```
    }
```

```
}
```

// Test 클래스

```
public class Test {
```

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

```
        // Fruit 타입의 배열을 생성하고 다양한 객체를 저장합니다.
```

```
        Fruit[] fary = new Fruit[4];
```

```
        fary[0] = new Grape();
```

```
        fary[1] = new Apple();
```

```
        fary[2] = new Grape();
```

```
        fary[3] = new Pear();
```

```
        // printFruit() 메소드를 호출하여 각 객체의 me() 메소드 실행
```

```
        printFruit(fary);
```

```
    }
```

```
    // printFruit() 메소드: 매개변수로 전달된 객체에 대해 me() 메소드를 호출
```

```
    public static void printFruit(Fruit[] fruits) {
```

---

```
for (Fruit fruit : fruits) {  
  
    fruit.me(); // me() 메소드 호출  
  
    if (fruit instanceof Grape) {  
  
        System.out.println(fruit.save()); // Grape 객체일 경우 save() 호출  
  
    }  
  
}  
  
}  
...  
}
```

### 설명:

- `Fruit` 클래스에서 `me()` 메소드는 빈 메소드로 정의되었습니다. 다른 클래스에서 이 메소드를 재정의하여 구체적인 구현을 제공합니다.

- `Apple`, `Grape`, `Pear` 클래스는 `Fruit`를 상속받고 `me()` 메소드를 재정의합니다.

- `Apple`은 "나는 사과"를 출력합니다.

- `Grape`는 "나는 포도"를 출력하고, `save()` 메소드를 재정의하여 "저장할 수 없습니다"라는 메시지를 반환합니다.

- `Pear`는 "나는 배"를 출력합니다.

- `Test` 클래스에서 `Fruit[] fary` 배열을 생성하고, `Grape`, `Apple`, `Grape`, `Pear` 객체를 배열에 저장한 후, `printFruit()` 메소드를 호출하여 각 객체의 `me()` 메소드를 실행합니다. `Grape` 객체일 경우 "저장할 수 없습니다"도 출력합니다.

### 실행 결과:

``plaintext

나는 포도

	<p>저장할 수 없습니다</p> <p>나는 사과</p> <p>나는 포도</p> <p>저장할 수 없습니다</p> <p>나는 배</p> <p>...</p> <p>위 코드는 주어진 요구사항에 맞춰 다형성과 메소드 재정의의 사용하여 구현되었습니다. `Grape` 객체가 포함되었을 때 "저장할 수 없습니다" 메시지가 출력되는 부분을 제대로 처리했습니다.</p>
실행결과	<pre>나는 포도 저장할 수 없습니다 나는 사과 나는 포도 저장할 수 없습니다 나는 배</pre>

(3) 제시된 결과처럼 실행될 수 있도록 Baby와 Dog 클래스를 작성하고 실행 결과를 제시하세요.

```
신생아 : 기어 다닙니다.
신생아 : 웅얼이를 합니다.
강아지 : 네 발로 다닙니다.
강아지 : 멍멍 소리를 냅니다
```

```
abstract class Animal {
    private String type; // 유형
    public Animal(String type) {
        this.type = type;
    }
    public String getType() {
        return type;
    }
    abstract void run();
    abstract void sound();
}

public class Test {
    public static void play(Animal ani) {
        //본인 작성
    }
}
```

```

    }

    public static void main(String[] args) {
        Animal animal = new Baby("신생아");
        play(animal);
        animal = new Dog("강아지");
        play(animal);
    }
}

```

#### [프로그램 소스]

```
package week5.practice3;
```

```

abstract class Animal {
    private String type;

    public Animal(String type) {
        this.type = type;
    }

    public String getType() {
        return type;
    }

    abstract void run();
    abstract void sound();
}

```

```
package week5.practice3;
```

```

public class Baby extends Animal {
    public Baby(String type) {
        super(type);
    }

    @Override
    void run() {
        System.out.println(getType() + " : 기어 다닙니다.");
    }

    @Override
    void sound() {
        System.out.println(getType() + " : 웅얼이를 합니다.");
    }
}

```

```
package week5.practice3;
```

```

public class Dog extends Animal {
    public Dog(String type) {
        super(type);
    }

    @Override
    void run() {
        System.out.println(getType() + " : 네 발로 다닙니다.");
    }

    @Override
    void sound() {
        System.out.println(getType() + " : 멍멍 소리를 냅니다.");
    }
}

```

```
}  
}
```

```
package week5.practice3;  
  
public class Test {  
    public static void play(Animal ani) {  
        ani.run();  
        ani.sound();  
    }  
    public static void main(String[] args) {  
        play(new Baby("신생아"));  
        play(new Dog("강아지"));  
    }  
}
```

#### [실행 결과]

```
신생아 : 기어 다닙니다.  
신생아 : 웅얼이를 합니다.  
강아지 : 네 발로 다닙니다.  
강아지 : 멍멍 소리를 냅니다.
```

- (4) 슈퍼 클래스인 Shape에 도형 이름을 저장하는 필드와 도형의 면적(double getArea())을 계산하는 메소드를 제공합니다. 삼각형을 나타내는 클래스 Triangle은 Shape을 상속받아 작성합니다. 삼각형에 맞도록 면적(x=밑변의 길이, y=높이)을 계산하는 메소드를 재정의 합니다. 삼각형의 x, y는 생성자에서 저장합니다. 클래스 Circle도 Shape을 상속받아 작성합니다. 원에 맞도록 면적(x=반지름, y=3.14)을 계산하는 메소드를 재정의 합니다. 원의 x, y는 생성자에서 저장합니다. Test 클래스를 작성하여서 삼각형, 원 객체를 생성하고 다형성을 이용하여 각 객체의 정보와 면적을 출력하세요. (힌트 abstract class 사용)

```
public class Test {  
    public static void main(String args[]) {  
        Shape[] objs = new Shape[2];  
        objs[0] = new Circle("Circle", 1, 3.14);  
        objs[1] = new Triangle("Rectangle", 1, 2);  
        for( Shape obj : objs )  
            System.out.println(obj.getName() + " area = " + obj.getArea());  
    }  
}
```

#### [프로그램 소스]

```
package week5.practice4;  
  
class Circle extends Shape {  
    public Circle(String name, double x, double y) {  
        super(name);  
        this.x = x;  
        this.y = 3.14;  
    }  
    @Override  
    public double getArea() {  
        return (x*x) * y;  
    }  
}
```

```
}  
}
```

```
package week5.practice4;  
  
abstract class Shape {  
    protected String name;  
    protected double x,y;  
    public abstract double getArea();  
  
    public Shape(String name) {  
        this.name = name;  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```

```
package week5.practice4;  
  
class Triangle extends Shape {  
    public Triangle(String name, double x, double y) {  
        super(name);  
        this.x = x;  
        this.y = y;  
    }  
  
    @Override  
    public double getArea() {  
        return x*y / 2;  
    }  
}
```

```
package week5.practice4;  
  
public class Test {  
    public static void main(String args[]) {  
        Shape[] objs = new Shape[2];  
        objs[0] = new Circle("Circle",1, 3.14);  
        objs[1] = new Triangle("Rectangle",1, 2);  
        for( Shape obj : objs )  
            System.out.println(obj.getName() + " area = " + obj.getArea());  
    }  
}
```

[실행 결과]

```
Circle area = 3.14  
Rectangle area = 1.0
```

- (5) AnimalTest 클래스가 다음과 같이 동작하도록 클래스들을 완성하시오. 단 Animal 클래스는 객체 생성이 불가능 합니다. reaction()은 Animal 클래스에서 인스턴스를 확인하여 출력하고, sound()는 오버라이딩하여 사용합니다. Dog는 멍멍, Cat은 야~옹 합니다.

```

<terminated> AnimalTest (Java Application) C:\Program Files\
Nabee고양이 꼬리내리고 야~옹
Anonymous강아지 꼬리올리고 멍멍
Anonymous고양이 꼬리내리고 야~옹
Anonymous강아지 꼬리올리고 멍멍
Rock강아지 꼬리올리고 멍멍
Pet들 중 Dog는 3마리

```

```

package test;

public class AnimalTest {
    public static void main(String[] args) {
        Animal[] pets = {
            new Cat("Nabee"),
            new Dog(),
            new Cat(),
            new Dog(),
            new Dog("Rock")
        };

        for (Animal ani : pets) {
            System.out.print("\n" + ani + " " + ani.reaction(ani));
            ani.sound();
        }
        System.out.print("\n\nPet들 중 Dog는 " + Dog.getCountDog() + "마리\n");
    }
}

```

## [프로그램 소스]

```

package week5.practice5;

abstract class Animal {
    // 변수 선언
    // 피드백 수정
    protected String name; // 이름

    // 생성자
    // 피드백 수정
    public Animal(String name) {
        this.name = (name != null) ? name : "Anonymous";
    }

    // 메소드
    public abstract void sound();

    // 피드백 수정
    public String reaction(Animal ani) {
        // instanceof : 객체가 특정 클래스 또는 그 하위 클래스의 인스턴스인지 확인하는 데
        // 사용된다.
        return ani instanceof Dog ? "강아지_꼬리올리고" : "고양이_꼬리내리고";
    }

    @Override
    public String toString() {
        return name;
    }
}

```

```

package week5.practice5;

public class Cat extends Animal {

    // 생성자
    // 피드백 수정
    public Cat(String name) {
        super(name);
    }

    // 메소드
    public Cat() {
        this("Anonymous");
    }
}

```

```

    }

    @Override
    public void sound() {
        System.out.print(" 야~옹");
    }
}

```

```

package week5.practice5;

class Dog extends Animal {

    // Dog 개수 변수 선언
    private static int countDog = 0;

    // 생성자
    // 피드백 수정
    public Dog(String name) {
        super(name);
        countDog++;
    }

    // 메소드
    public Dog() {
        this("Anonymous");
    }

    @Override
    public void sound() {
        System.out.print(" 멍멍");
    }

    public static int getCountDog() {
        return countDog;
    }
}

```

```

package week5.practice5;

public class AnimalTest {
    public static void main(String[] args) {
        Animal[] pets = {
            new Cat("Nabee"),
            new Dog(),
            new Cat(),
            new Dog(),
            new Dog("Rock")
        };
        // 피드백 수정
        for (Animal ani : pets) {
            System.out.print("\n" + ani + ani.reaction(ani));
            ani.sound();
        }
        System.out.print("\n\nPet 들 중 Dog 는 " + Dog.getCountDog() + "마리\n");
    }
}

```

[실행 결과]



Nabee고양이\_꼬리내리고 야~옹  
 Anonymous강아지\_꼬리올리고 멍멍  
 Anonymous고양이\_꼬리내리고 야~옹  
 Anonymous강아지\_꼬리올리고 멍멍  
 Rock강아지\_꼬리올리고 멍멍

Pet들 중 Dog는 3마리

(6) 상속과 재정의를 사용하여 제시된 조건대로 프로그램을 작성하고 테스트 하세요.

SalariedEmployee 객체
필드 : 이름, 사번, 월급 -전용 멤버
메소드: computeSalary()=> 연봉 계산(월급 * 12)하고 반환 설정자, 접근자 메소드, 생성자, toString()-객체 내용 출력
HourEmployee 객체
필드 : 이름, 사번, 시간당 임금, 일한 시간 - 전용 멤버
메소드: computeSalary()=> 임금 계산(시간당 임금 * 일한 시간)하고 반환 설정자, 접근자 메소드, 생성자, toString()-객체 내용 출력

```
class Person{ //수퍼클래스
    //공통되는 필드와 메소드를 작성
}

class SalariedEmployee{ //Person 클래스 상속
    // computeSalary()메소드 재정의
    //필요한 부분 추가
}

class HourEmployee { //Person 클래스 상속
    // 필드 추가, computeSalary()메소드 재정의
    //필요한 부분 추가
}

public class Test {
    public static void main(String[] args) {
        //프로그램 종료 전 모든 객체 정보 출력
    }
}
```

```
----- 직원 목록 출력 -----
이름 : soft, 사번 : 2345, 급여 : 70, 연봉 : 840
이름 : info, 사번 : 2345, 시간당 임금 : 60, 일한시간: 6, 금액 : 360
이름 : hallym, 사번 : 6534, 급여 : 85, 연봉 : 1020
이름 : computer, 사번 : 8546, 시간당 임금 : 55, 일한시간: 8, 금액 : 440
----- SalariedEmployee 연봉 출력 -----
이름 : soft, 연봉 : 840
이름 : hallym, 연봉 : 1020
```

```

package week5.practice6;

class Person {
    // 변수 선언
    private String name;        // 이름
    private String employeeId;  // 사번

    // 생성자
    public Person(String name, String employeeId) {
        this.name = name;
        this.employeeId = employeeId;
    }

    // 설정자 및 접근자
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmployeeId() {
        return employeeId;
    }

    public void setEmployeeId(String employeeId) {
        this.employeeId = employeeId;
    }

    // 피드백 수정
    public int computeSalary() {
        return 0; // 메소드 타입이 void가 아닌 경우 반환 값이 존재해야한다.
    }

    // toString()
    @Override
    public String toString() {
        return "이름 : " + name + ", 사번 : " + employeeId;
    }
}

```

```

package week5.practice6;

public class SalariedEmployee extends Person {
    // 변수 선언
    private int salary; // 월급

    // 생성자
    public SalariedEmployee(String name, String employeeId, int salary) {
        super(name, employeeId);
        this.salary = salary;
    }

    // 설정자 및 접근자
    public int getSalary() {
        return salary;
    }

    public void setSalary(int salary) {
        this.salary = salary;
    }

    // 메소드
    @Override // 피드백 수정

```

```

public int computeSalary() {
    return salary * 12; // 연봉
}

// toString()
@Override
public String toString() {
    return super.toString() + ", 급여 : " + salary + ", 연봉 : " + computeSalary();
}
}

```

```

package week5.practice6;

public class HourEmployee extends Person {
    // 변수 선언
    private int hourlySalary;
    private int workedHours;

    // 생성자
    public HourEmployee(String name, String employeeId, int hourlySalary, int
workedHours) {
        super(name, employeeId);
        this.hourlySalary = hourlySalary;
        this.workedHours = workedHours;
    }

    // 설정자 및 접근자
    public int getHourlySalary() {
        return hourlySalary * workedHours; // 금액 계산
    }

    public void setHourlySalary(int hourlySalary) {
        this.hourlySalary = hourlySalary;
    }

    public int getWorkedHours() {
        return workedHours;
    }

    public void setWorkedHours(int workedHours) {
        this.workedHours = workedHours;
    }

    @Override // 피드백 수정
    public int computeSalary() {
        return hourlySalary * workedHours; // 임금 계산
    }

    // toString()
    @Override
    public String toString() {
        return super.toString() + ", 시간당 임금: " + hourlySalary + ", 일한시간: " +
workedHours + ", 금액: " + computeSalary();
    }
}

```

```

package week5.practice6;

public class Test {
    public static void main(String[] args) {

        SalariedEmployee emp1 = new SalariedEmployee("soft", "2345", 70);
        SalariedEmployee emp2 = new SalariedEmployee("hallym", "6534", 85);

        HourEmployee emp3 = new HourEmployee("info", "2345", 60, 6);
    }
}

```

```

HourEmployee emp4 = new HourEmployee("computer", "8546", 55, 8);

System.out.println("----- 직원 목록 출력 -----");
System.out.println(emp1);
System.out.println(emp3);
System.out.println(emp2);
System.out.println(emp4);

System.out.println("----- SalariedEmployee 연봉 출력 -----");
--");
    System.out.println("이름 : " + emp1.getName() + ", 연봉 : " +
emp1.computeSalary());
    System.out.println("이름 : " + emp2.getName() + ", 연봉 : " +
emp2.computeSalary());
}
}

```

#### [실행 결과]

```

----- 직원 목록 출력 -----
이름 : soft, 사번 : 2345, 급여 : 70, 연봉 : 840
이름 : info, 사번 : 2345, 시간당 임금: 60, 일한시간: 6, 금액: 360
이름 : hallym, 사번 : 6534, 급여 : 85, 연봉 : 1020
이름 : computer, 사번 : 8546, 시간당 임금: 55, 일한시간: 8, 금액: 440
----- SalariedEmployee 연봉 출력 -----
이름 : soft, 연봉 : 840
이름 : hallym, 연봉 : 1020

```

- (7) 생성형 AI가 제시하는 다형성을 활용하는 문제를 프로그램하고 결과를 제시 하세요. 문제 해결을 위한 코드는 답변에서 제외 되어야 합니다

생성형 AI 플랫폼	chatGPT
질문	(7) 생성형 AI가 제시하는 다형성을 활용하는 문제를 프로그램하고 결과를 제시 하세요. 문제 해결을 위한 코드는 답변에서 제외 되어야 합니다
답변	<p>다형성을 활용하는 다른 주제의 예시로 <b>***결제 시스템***</b>을 생각해볼 수 있습니다. 이 문제는 여러 결제 방법을 구현하고, 동일한 인터페이스를 통해 다양한 결제 방식을 처리할 수 있게 하는 방법입니다.</p> <p>### 문제: <b>***온라인 쇼핑몰 결제 시스템***</b></p> <p>온라인 쇼핑몰에서는 다양한 결제 방법이 있습니다. 예를 들어, <b>**신용카드**</b>, <b>**페이팔**</b>, <b>**구글 페이**</b> 등 여러 방법으로 결제를 할 수 있습니다. 이때 다형성을 활용하여 각 결제 방법을 동일한 방식으로 처리할 수 있게 만드는 문제입니다.</p> <p>### 요구사항:</p> <ol style="list-style-type: none"> <li><b>**결제 인터페이스**</b>: <ul style="list-style-type: none"> <li>- 모든 결제 방식은 `결제`라는 공통 메서드를 가져야 한다.</li> <li>- 결제 방식별로 `결제` 메서드의 구현이 다르다.</li> </ul> </li> <li><b>**결제 방식별 행동**</b>: <ul style="list-style-type: none"> <li>- `신용카드`: 신용카드 정보를 입력받아 결제 처리가 이루어진다.</li> </ul> </li> </ol>

- `페이팔`: 페이팔 계정을 통해 결제 처리가 이루어진다.
- `구글 페이`: 구글 계정을 통해 결제 처리가 이루어진다.

### 3. **\*\*프로그램 흐름\*\***:

- 사용자가 선택한 결제 방법에 맞춰 결제를 처리한다.
- 결제 방식에 따라 `결제` 메서드가 다르게 동작해야 한다.

### ### 결과:

프로그램을 실행했을 때, 사용자가 결제 방법을 선택하면 각 결제 방식에 맞춰 결제가 진행됩니다. 예를 들어:

- **\*\*신용카드\*\***로 결제 시 "신용카드 결제 완료" 메시지가 출력된다.
- **\*\*페이팔\*\***로 결제 시 "페이팔 결제 완료" 메시지가 출력된다.
- **\*\*구글 페이\*\***로 결제 시 "구글 페이 결제 완료" 메시지가 출력된다.

이와 같이 다형성을 활용하여 동일한 `결제` 인터페이스로 다양한 결제 방식을 처리할 수 있습니다.

각 결제 방식은 공통된 인터페이스를 구현하면서도, 실제 처리 내용은 다르게 구현되어야 합니다.

#### [프로그램 소스]

```
package week5.practice7;

// 결제 추상 클래스
public abstract class Payment {
    // 결제 추상 메소드
    public abstract void processPayment(int amount);
}
```

```
package week5.practice7;

// 신용카드 결제 방식
public class CreditCardPayment extends Payment {
    // 메소드 재정의
    @Override
    public void processPayment(int amount) {
        System.out.println("신용카드 결제 완료: " + amount + "원");
    }
}
```

```
package week5.practice7;

// 페이팔 결제 방식
public class PaypalPayment extends Payment {
    // 메소드 재정의
    @Override
    public void processPayment(int amount) {
        System.out.println("페이팔 결제 완료: " + amount + "원");
    }
}
```

```
package week5.practice7;

// 구글 결제 방식
public class GooglePayPayment extends Payment {
    // 메소드 재정의
    @Override
    public void processPayment(int amount) {
        System.out.println("구글 페이 결제 완료: " + amount + "원");
    }
}
```

```
}  
}
```

```
package week5.practice7;  
  
// 결제 과정 클래스  
public class PaymentProcessor {  
    // 변수 선언  
    private Payment paymentMethod; // 결제 처리 방식  
  
    // 생성자  
    public PaymentProcessor(Payment paymentMethod) {  
        this.paymentMethod = paymentMethod;  
    }  
  
    // 메소드  
    public void process(int amount) {  
        paymentMethod.processPayment(amount);  
    }  
}
```

```
package week5.practice7;  
  
public class Main {  
    // 피드백 수정  
    public static void main(String[] args) {  
        // 객체 배열  
        // CreditCardPayment:신용카드 결제, GooglePayPayment:구글결제, PaypalPayment :  
        //페이팔 결제  
        Payment[] payments = {new CreditCardPayment(), new PaypalPayment(), new  
        GooglePayPayment()};  
  
        // 가격 배열  
        int[] amount = {10000, 5000, 2000};  
  
        // 배열 삽입  
        for(int i = 0; i < payments.length; i++) {  
            payments[i].processPayment(amount[i]);  
        }  
    }  
}
```

[실행 결과]

신용카드 결제 완료: 10000원

페이팔 결제 완료: 5000원

구글 페이 결제 완료: 2000원