하위클래스의 동작방식을 숨기고 상위클래스로 하여금 하위클래스를 동작시키는 클래스 설계방법의 장점

하위클래스의 동작 메서드를 숨기고 상위클래스가 하위클래스를 조작할 수 있도록 클래스를 설계하는 방법은 여러 가지 이점을 제공합니다:

1.**캡슐화:** 상위클래스가 하위클래스의 구현 세부 사항을 숨김으로써 캡슐화를 달성합니다. 상위클래스는 각 하위클래스가 작업을 어떻게 수행하는 지의 특정 세부 사항을 알 필요가 없으므로 모듈화 되고 유지 관리 가능한 디자인이 가능해집니다.

1.추상화: 상위클래스는 추상 메서드 또는 인터페이스를 통해 하위클래스와 상호 작용합니다. 이 추상화를 통해 상위클래스는 다른 하위클래스를 균일하게 처리할 수 있으며 코드의 재사용성과 유연성을 높일 수 있습니다.

2.다형성: 상위클래스가 추상 메서드 또는 인터페이스를 통해 하위클래스와 상호 작용함으로써 다형성을 구현합니다. 하위클래스들은 이러한 메서드들에 자신들의 구현을 제공할 수 있으며, 상위클래스는 컴파일 시간에 특정 하위클래스 유형을 알 필요 없이 이러한 구현을 사용할 수 있습니다.

3.느슨한 결합: 상위클래스는 하위클래스와의 계약을 정의하는 추상 메서드 또는 인터페이스에만 의존합니다. 이 느슨한 결합은 하위클래스의 변경이 상위클래스에 영향을 미치지 않도록 보장하며, 유지 관리와 향후 확장이 용이 해집니다.

4.유연성과 확장성: 상위클래스와 호환되는 새로운 하위클래스를 추가하는 것은 간단합니다. 기존 상위클래스를 수정하지 않고도 새로운 하위클래스를 생성할 수 있으며, 이는 클래스가 확장에는 열려 있지만 수정에는 닫혀 있어야 한다는 개방-폐쇄 원칙을 따릅니다.

5.코드 재사용성: 상위클래스는 자신의 계약을 따르는 모든 하위클래스와 작동할 수 있도록 합니다. 이로써 같은 상위클래스 로직을 여러 하위클래스에 걸쳐 재사용하는 것이 쉬워집니다.

6.클라이언트 코드 간소화: 상위클래스와 상호 작용하는 클라이언트 코드는 하위클래스의 구체적인 세부 사항을 다룰 필요가 없습니다. 이로 인해 클라이언트 코드가 단순화되며, 직접적으로 하위클래스의 동작을 조작하는 것에 따른 오류를 줄일 수 있습니다.

7.계층 관리: 상위클래스가 하위클래스를 조작함으로써 클래스 계층 구조를 더 효율적으로 관리할 수 있습니다. 상위클래스는 여러 하위클래스들 간에 공통 동작과 특성을 정의합니다.

요약하면, 하위클래스의 동작 메서드를 숨기고 상위클래스가 하위클래스를 조작할 수 있도록 클래스를 설계하는 것은 캡슐화, 추상화, 다형성, 느슨한 결합, 유연성, 코드 재사용성 등의 이점을 제공합니다. 이를 통해 유지 관리 가능하고 확장 가능한 코드베이스를 구축하며, 클래스 계층 구조에 대한 향후 개선과 수정을 용이하게 합니다.

설계순서

1.추상 클래스 또는 인터페이스 생성: 상위클래스로 추상 클래스 또는 인터페이스를 정의합니다. 이 추상 클래스 또는 인터페이스는 하위클래스들이 따라야 할 공통 동작 또는 계약을 선언하는 역할을 합니다. 이 추상 클래스 또는 인터페이스는 하위클래스들의 청사진 역할을 합니다.

2.상위클래스에서 공통 동작 구현: 추상 클래스에서 공통 동작 또는 기본 동작을 구현하거나 인터페이스에서 기본 메서드 구현을 제공합니다. 이 공통 동작은 상위클래스를 상속받는 모든 하위클래스들이 공유할 수 있습니다.

3.추상 메서드 선언: 추상 클래스 또는 인터페이스에서 추상 메서드를 선언합니다. 이 추상 메서드들은 하위클래스들이 반드시 구현해야 하는 동작을 나타냅니다. 추상 클래스는 이러한 메서드에 대한 구체적인 구현을 제공하지 않으며, 하위클래스들이 자신들의 특정 구현을 제공해야 합니다.

4.하위클래스 구현: 상위클래스를 확장하는 각 하위클래스는 상위클래스에서 선언한 추상 메서드들에 대한 구체적인 구현을 제공해야 합니다. 이를 통해 하위클래스들은 상위클래스가 정의한 계약을 이행합니다.

이러한 단계를 따라, 상위클래스는 하위클래스와의 상호 작용을 위한 공통 인터페이스를 제공하며, 하위클래스들은 자신들의 구현 세부 사항을 외부 세계로부터 캡슐화 합니다.

간단한 예시를 통해 이 개념을 설명하겠습니다:

이 예시에서 Shape 클래스는 상위클래스 역할을 하며 calculateArea() 메서드를 추상 메서드로 선언합니다.
Circle과 Rectangle 클래스는 Shape 클래스를 상속받고 calculateArea() 메서드를 자신들의 구체적인 구현으로 제공합니다.

ShapeMain 클래스에서는 상위클래스 참조 (Shape)를 사용하여 하위클래스들 (Circle과 Rectangle)과 상호 작용하며, 각 하위클래스의 구현 세부 사항을 알 필요 없이 동적인 동작을 구현할 수 있습니다. 이는 실행 시간에 실제 하위클래스 인스턴스를 기반으로 유연하고 동적인 동작을 가능하게 합니다.

추상클래스 실습 예제

```
// 스텝 1: 추상 클래스 또는 인터페이스 생성 (상위클래스)
public abstract class Shape {
    // 스텝 2: 공통 동작 구현 (선택 사항)
    public void display() {
        System.out.println("도형을 표시합니다.");
    }

    // 스텝 3: 추상 메서드 선언 (하위클래스들이 반드시 구현)
    public abstract double calculateArea();
}

// 스텝 4: 하위클래스 구현
public class Circle extends Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    // 원의 넓이를 계산하는 추상 메서드 구현
    @Override
    public double calculateArea() {
        return Math.PI * radius * radius;
    }
}
```

```
public class Rectangle extends Shape {
    private double width;
    private double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    // 사각형의 넓이를 계산하는 추상 메서드 구현
    @Override
    public double calculateArea() {
        return width * height;
    }
}

public class ShapeMain {
    public static void main(String[] args) {
        // 상위클래스 참조를 사용하여 하위클래스들과 상호 작용
        Shape circle = new Circle(5.0);
        Shape rectangle = new Rectangle(4.0, 6.0);

        circle.display();
        System.out.println("원의 넓이: " + circle.calculateArea());

        rectangle.display();
        System.out.println("사각형의 넓이: " + rectangle.calculateArea());
    }
}
```

```
도형을 표시합니다.
원의 넓이: 78.53981633974483
도형을 표시합니다.
사각형의 넓이: 24.0
```