자바의 상속, 다형성, 추상 클래스에 대한 이해

1. 서론

오늘날 소프트웨어 개발은 점차 규모가 커지고 복잡해지고 있다. 이에 따라 코드의 재사용성, 유지보수 용이성, 확장성이 중요한 요소로 부각되고 있으며, 객체지향 프로그래밍은 이러한 문제를 해결하기 위해 등장한 대표적인 개발 패러다임이다. 자바(Java)는 객체지향 프로그래밍을 기반으로 설계된 대표적인 프로그래밍 언어이며, 이를 통해 보다 체계적이고 유연한 프로그램 개발이 가능하다.

그 중에서도 **상속**, **다형성**, **추상 클래스**는 객체지향의 가장 기본적이면서도 강력한 기능으로, 코드의 품질을 높이는 데 크게 기여한다. 본 보고서에서는 이 세 가지 개념을 설명하고, 이해를 돕기 위한 간단한 예시 코드를 함께 제시하고자 한다.

---

2. 상속 (Inheritance)

2.1 개념 설명

상속이란 기존에 정의된 클래스의 속성과 기능을 새로운 클래스가 물려받아 사용하는 개념을 말한다. 이를 통해 개발자는 공통 기능을 매번 새로 작성하지 않고도 재사용할 수 있으며, 필요한 기능만을 추가하거나 변경함으로써 효율적인 코드 작성이 가능하다.

2.2 실생활 비유

자동차를 예로 들어 설명해 보겠다. 모든 자동차는 달리고 멈추는 공통된 기능을 가지고 있다. 이를 `Car`라는 이름의 부모 클래스로 정의해 두고, 전기자동차나 가솔린자동차와 같은 구체적인 자동차를 자식 클래스로 만들어 추가적인 기능만 구현하면 된다.

2.3 예시 코드

```
// 부모 클래스
class Car {
    void run() {
        System.out.println("자동차가 달립니다.");
    }

    void stop() {
        System.out.println("자동차가 멈춥니다.");
    }
```

```
}

// 자식 클래스
class ElectricCar extends Car {
    void charge() {
        System.out.println("전기차를 충전합니다.");
    }
}

// 사용 예시
public class Main {
    public static void main(String[] args) {
        ElectricCar myCar = new ElectricCar();
        myCar.run();     // 상속받은 메서드 호출
        myCar.stop();    // 상속받은 메서드 호출
        myCar.charge();  // 자식 클래스의 메서드 호출
    }
}
```

위의 예시와 같이 `ElectricCar` 클래스는 별도의 코드 작성 없이 `Car` 클래스의 `run()`, `stop()` 메서드를 사용할 수 있으며, 전기차만의 기능인 `charge()` 메서드를 추가로 정의할 수 있다.

---

3. 다형성 (Polymorphism)

3.1 개념 설명

다형성이란 하나의 부모 타입으로 여러 자식 객체를 다룰 수 있는 특성을 의미한다. 이를 통해 코드의 유연성과 확장성을 높일 수 있으며, 다양한 객체를 일관된 방식으로 처리할 수 있다.

3.2 실생활 비유

게임 속 몬스터를 예로 들어보자. 고블린, 오크, 드래곤 등 여러 종류의 몬스터가 존재하지만, 이들은 모두 "몬스터"라는 공통된 특성을 가지고 있다. 이들을 각각 따로 관리하는 것이 아니라 "몬스터"라는 하나의 타입으로 통합해 처리할 수 있다면, 코드가 훨씬 간결해지고 유지보수가 쉬워질 것이다.

3.3 예시 코드

```
// 부모 클래스
class Monster {
    void attack() {
        System.out.println("몬스터가 공격합니다.");
    }
}

// 자식 클래스
class Goblin extends Monster {
    @Override
    void attack() {
        System.out.println("고블린이 날렵하게 공격합니다.");
    }
}

class Orc extends Monster {
    @Override
    void attack() {
        System.out.println("오크가 힘껏 공격합니다.");
    }
}

// 사용 예시
public class Main {
    public static void main(String[] args) {
        Monster monster1 = new Goblin();
        Monster monster2 = new Orc();

        monster1.attack();  // 고블린이 날렵하게 공격합니다.
        monster2.attack();  // 오크가 힘껏 공격합니다.
    }
}
```

위 코드처럼 부모 타입인 `Monster`로 자식 객체를 다룰 수 있으며, 각각의 객체가 자신만의 방식으로 동작하는 것을 확인할 수 있다.

---

4. 추상 클래스 (Abstract Class)

4.1 개념 설명

추상 클래스는 **공통된 틀을 정의하지만 직접 사용하지 않고**, 반드시 이를 상속하는 자식 클래스에서 구체적인 동작을 완성하도록 강제하는 설계도 역할을 한다. 추상 클래스는 `abstract` 키워드를 사용하여 선언하며, 하나 이상의 **추상 메서드**를 가질 수 있다.

4.2 실생활 비유

몬스터는 모두 공격과 방어를 해야 한다는 규칙이 있지만, 구체적인 공격 방식은 몬스터마다 다를 수 있다. 이처럼 "반드시 만들어야 할 기능"을 규칙으로 정의하고, 자식 클래스가 이를 구현하도록 강제하는 것이 추상 클래스의 역할이다.

4.3 예시 코드

```
// 추상 클래스
abstract class Monster {
    abstract void attack();  // 반드시 자식 클래스가 구현해야 함
    abstract void defend();  // 반드시 자식 클래스가 구현해야 함
}

// 자식 클래스
class Dragon extends Monster {
    @Override
    void attack() {
        System.out.println("드래곤이 불을 뿜습니다.");
    }

    @Override
    void defend() {
        System.out.println("드래곤이 날개로 방어합니다.");
    }
}

// 사용 예시
public class Main {
    public static void main(String[] args) {
        Monster dragon = new Dragon();
        dragon.attack();  // 드래곤이 불을 뿜습니다.
        dragon.defend();  // 드래곤이 날개로 방어합니다.
```

```
    }
}
```

위 코드에서 `Monster` 클래스는 직접 사용할 수 없으며, 반드시 자식 클래스에서 `attack()`과 `defend()` 메서드를 구현해야만 정상적으로 사용할 수 있다. 이를 통해 일관된 규칙을 유지하면서도 각자의 특성에 맞는 동작을 정의할 수 있다.

---

5. 결론

본 보고서에서는 자바의 대표적인 객체지향 개념인 **상속**, **다형성**, **추상 클래스**에 대해 살펴보았다.

상속을 통해 **코드의 재사용성**을 높일 수 있으며,

다형성을 통해 **여러 객체를 하나의 타입으로 일괄 처리**할 수 있고,

추상 클래스를 통해 **일관된 설계 규칙을 유지**하면서도 **자유로운 확장**이 가능함을 확인하였다.

이러한 개념들은 처음 접할 때는 다소 어려워 보일 수 있으나, 실제 프로그램을 작성하고 실행해보면서 점차 익숙해질 수 있다. 따라서 개념 학습에 그치지 말고, 작은 예제부터 차근차근 구현해 보는 연습을 권장한다. 이를 통해 향후 더 복잡한 설계나 고급 개념도 무리 없이 이해하고 적용할 수 있는 기반이 마련될 것이다.