1. 상속 (Inheritance)

✅ 개념

- 상속은 기존 클래스를 기반으로 새로운 클래스를 만드는 것.
- 부모 클래스(상위 클래스)의 속성과 기능(메서드)을 자식 클래스(하위 클래스)가 물려받는 것

✅ 주요 키워드

- `extends`
  - 부모 클래스를 상속받을 때 사용
- `super`
  - 자식 클래스에서 부모 클래스의 멤버(필드, 메소드)를 참조할 때 사용
  - 부모 클래스의 생성자를 호출할 때에도 사용
- `this`
  - 자기 자신의 멤버(필드, 메소드)를 참조할 때 사용
  - 자기 자신의 다른 생성자를 호출할 때 사용

✅ 기본문법

```
class Parent {
    // 부모 클래스의 필드와 메소드
}
```

```
class Child extends Parent {
    // 자식 클래스는 부모의 기능을 물려받음
}
```

✅ 예시

```
// 예시 1
class Animal {
    void sound() {
        System.out.println("동물이 소리를 냅니다.");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("멍멍!");
    }
}



// 예시 2
class Vehicle {
 String brand;

 void start() {
 System.out.println("차량이 출발합니다.");
 }
}

class Car extends Vehicle {
 int doors;
```

```
    void openDoor() {
    System.out.println("문을 엽니다.");
    }
}
```

- 코드 재사용이 가능해서 중복을 줄일 수 있음
- 부모 클래스의 기능을 자식 클래스가 물려받아 코드를 재사용하고 확장할 수 있는 개념
- 공통된 기능은 부모 클래스에, 특수한 기능은 자식 클래스에 넣을 수 있음
- 객체지향의 핵심 기법으로 코드의 재사용성을 높여줄 수 있음
- 상속을 통해 클래스간의 계층적 구조 설계 가능

---

2. 다형성(Polymorphism)

✅ 개념

- 하나의 객체가 여러 가지 형태를 가질 수 있는 성질.
- 자식 클래스의 객체를 부모 클래스 타입으로 참조 가능.
- 메소드 오버라이딩을 통해 실행 시 다른 결과를 나타냄.

✅ 주요 키워드

- 업캐스팅(Upcasting)
    - 기본 자료형의 업캐스팅은 작은 자료형을 큰 자료형으로 변환하는 것
    - 객체의 업캐스팅은 자식 객체를 부모 타입으로 변환하는 것
    - 자동으로 변환됨 (묵시적 형변환)

    - `Animal animal = new Dog();` // 자동 업캐스팅
- 다운캐스팅(Downcasting)
    - 기본 자료형의 업캐스팅은 큰 자료형을 작은 자료형으로 변환하는 것
    - 객체의 다운캐스팅은 부모 타입을 자식 타입으로 변환하는 것
    - 명시적 형변환 필요
    - `Dog dog = (Dog) animal;` // 명시적 다운캐스팅
- `instanceof`
    - 객체의 타입 확인하는 연산자
    - 다형성을 활용한 코드에서 사용 가능

```
Animal animal = new Dog();

if (animal instanceof Dog) {
 // Dog 타입에 맞는 처리
 Dog dog = (Dog) animal;

} else if (animal instanceof Cat) {
 // Cat 타입에 맞는 처리
 Cat cat = (Cat) animal;
}
```

✅ 예시

```


class Animal {
    void sound() {
        System.out.println("동물이 소리를 냅니다.");
    }
}

class Dog extends Animal {
```

```
    @Override
    void sound() {
        System.out.println("멍멍!");
    }
}

class Cat extends Animal {
    @Override
    void sound() {
        System.out.println("야옹!");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal a1 = new Dog();
        Animal a2 = new Cat();

        a1.sound(); // 멍멍!
        a2.sound(); // 야옹!
    }
}
```

- 여러 자식 클래스를 하나의 부모 타입으로 다룰 수 있어서 유연한 코드 구성 가능
- 메서드를 오버라이딩(재정의)해서 자식 클래스마다 다르게 동작시킬 수 있음 (유지보수 및 재사용성 향상)

---

3. 추상 클래스 (Abstract Class)

✅ 개념

- 추상화란 객체의 공통된 특징이나 속성을 뽑아내는 것 (복잡한 것을 단순화함)
- 추상 클래스는 인스턴스화 불가능하며, 자식 클래스가 상속받아 구체적인 구현을 해야 함
- 추상 클래스는 객체를 직접 만들 수는 없고, 공통된 틀만 정의해 놓은 클래스 => 그 안에 있는 추상 메서드(내용 없는 메서드)는 자식 클래스에서 반드시 오버라이딩해야 함

✅ 주요 키워드

- `abstract`
  - 추상 클래스는 abstract 키워드를 사용해 만들 수 있음
  - 추상 클래스 내부에는 생성자, 필드, 일반 메소드, 추상 메소드가 포함될 수 있음

```
abstract class 클래스명 {
 // 필드

 // 생성자

 // 일반 메소드

 // 추상 메소드
 abstract void 메소드명();
}
```

✅ 기본문법

```
abstract class 클래스명 {
```

```
    abstract void 메소드명(); // 추상 메소드
    void 일반메소드() {
        // 구현 가능
    }
}
```

✅ 예시

```
// 예시 1

abstract class Animal {
 // 공통 특성 추출 소리내기는 동물마다 다름
 abstract void makeSound();

 void walk() { } // 걷는 것은 동물마다 비슷

 void run() { } // 뛰는 것도 비슷
}

class Dog extends Animal {
 void makeSound() {
 System.out.println("멍멍!");
 }
}

class Cat extends Animal {
 void makeSound() {
 System.out.println("야옹!");
 }
}


// 예시 2
```

```java
abstract class Shape {
 // 추상 메소드 선언
 abstract void draw();
 abstract double getArea();

 // 일반 메소드
 void printInfo() {
 System.out.println("도형입니다.");
 }
}

// 자식 클래스에서 구현
class Circle extends Shape {
 double radius;

 @Override
 void draw() {
 System.out.println("원을 그립니다.");
 }

 @Override
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

- new 연산자로 객체를 생성할 수 없음
- `abstract` 키워드 사용
- 추상 메소드는 바디(몸통)가 없음
- 일반 메소드와 필드도 가질 수 있음
- 자식 클래스는 반드시 추상 메소드를 오버라이딩해야 함

- 추상화를 적용하면 공통 특성 추출해서 자식 클래스에서 각기 구현 가능 => 유지보수 용이 및 복잡도 감소
- 추상 클래스를 상속 받은 자식 클래스가 필수로 구현해야 하는 추상 메소드 (메소드 바디 없음)
- 공통 동작은 추상 클래스에서 직접 구현
- 자식 클래스마다 세부 구현이 달라질 수 있다면 추상 메소드로 선언

🔹 상속 (Inheritance)

- **정의**: 기존 클래스의 필드와 메소드를 자식 클래스가 물려받는 것
- **장점**: 코드 재사용, 유지보수성 향상
- **주의**: 다중 상속은 자바에서 지원하지 않음 (단, 인터페이스는 다중 구현 가능)

🔹 다형성 (Polymorphism)

- **정의**: 부모 타입의 참조변수로 여러 자식 객체를 다룰 수 있는 능력
- **장점**:
  - 유연한 코드 구조
  - 새로운 클래스가 추가되어도 기존 코드를 변경하지 않아도 됨
- **구현 방법**:
  - 메소드 오버라이딩
  - 업캐스팅 (자식 → 부모) 사용

🔹 추상 클래스 (Abstract Class)

- **정의**: 공통된 특성을 가지지만 자체로는 객체화할 수 없는 클래스
- **용도**: 자식 클래스가 공통 메소드를 재정의하도록 강제할 때 사용
- **특징**:
  - 객체 생성 불가
  - 일반 메소드 + 추상 메소드 포함 가능
  - 자식 클래스는 반드시 추상 메소드 구현 필요

개념	설명
상속	부모 클래스의 속성과 기능을 자식 클래스가 물려받는 것
다형성	하나의 객체가 여러 형태로 동작하는 것 (ex. 오버라이딩)
추상 클래스	공통된 구조를 잡아주는 틀, 직접 객체 생성은 안 되고 자식 클래스에서 구현해야 함