다형성

다형성이란 프로그램 언어 각 요소들(상수, 변수, 식, 객체, 메소드 등)이
다양한 자료형(type)에 속하는 것이 허가되는 성질을 가리킨다.

- 위키피디아 -

또는 여러 형태를 받아들일 수 있는 성질, 상황에 따라 의미를 다르게 부여할 수 있는 특성 등으로 정의,

다형성이란 **하나의 타입**에 **여러 객체를 대입**할 수 있는 성질.

다형성을 활용하면 기능을 확장하거나, 객체를 변경해야 할 때,
-〉 타입 변경 없이 객체 주입만으로 수정이 일어나게 가능.

상속을 사용한다면 중복되는 코드까지 제거할 수 있으므로 더욱 객체 지향 설계와 가까워짐.

대표적으로 **오버로딩, 오버라이딩, 함수형인터페이스**

다형성

객체 지향 특징

- 추상화
- 캡슐화
- 상속
- **다형성**

객체 지향 프로그래밍

- 객체 지향 프로그래밍은 컴퓨터 프로그램을 명령어의 목록으로 보는 시각에서 벗어나 여러 개의 독립된 단위,
  즉 "**객체**"들의 **모임**으로 파악하고자 하는 것이다. 각각의 **객체**는 **메시지**를 주고받고, 데이터를 처리할 수 있다. (**협력**)
- 객체 지향 프로그래밍은 프로그램을 **유연**하고 **변경**이 용이하게 만들기 때문에 대규모 소프트웨어 개발에 많이 사용된다.

다형성

유연하고 변경에 용이?

- 레고 블럭 조립하듯이
- 키보드, 마우스 갈아끼우듯이
- 컴퓨터 부품 갈아끼우듯이
- 컴포넌트를 쉽고 유연하게 변경하면서 개발할 수 있는 방법

다형성(Polymorphism)

다형성의 실세계 비유

- 실세계와 객체 지향을 1:1로 매칭X
- 그래도 실세계의 비유로 이해하기에는 좋음
- **역할**과 **구현**으로 세상을 구분

다형성의 실세계 비유

예시

- 운전자 - 자동차
- 공연 무대
- 키보드, 마우스, 세상의 표준 인터페이스들
- 정렬 알고리즘
- 할인 정책 로직

다형성(Polymorphism)

역할과 구현을 분리
자바 언어

- 자바 언어의 다형성을 활용
  - 역할 = 인터페이스
  - 구현 = 인터페이스를 구현한 클래스, 구현 객체
- 객체를 설계할 때 **역할**과 **구현**을 명확히 분리
- 객체 설계 시 역할(인터페이스)을 먼저 부여하고, 그 역할을 수행하는 구현 객체 만들기

다형성(Polymorphism)

자바언어의 다형성

- 오버라이딩을 떠올려보자
- 오버라이딩은 자바 기본 문법
- 오버라이딩 된 메서드가 실행
- 다형성으로 인터페이스를 구현한 객체를 실행 시점에 유연하게 변경할 수 있다.
- 물론 클래스 상속 관계도 다형성, 오버라이딩 적용가능

다형성(Polymorphism)

다형성의 본질

- 인터페이스를 구현한 객체 인스턴스를 **실행 시점**에 **유연**하게 **변경**할 수 있다.
- 다형성의 본질을 이해하려면 **협력**이라는 객체사이의 관계에서 시작해야함
- **클라이언트를 변경하지 않고, 서버의 구현 기능을 유연하게 변경할 수 있다.**

다형성(Polymorphism)

역할과 구현을 분리
정리

- 실세계의 역할과 구현이라는 편리한 컨셉을 다형성을 통해 객체 세상으로 가져올 수 있음
- 유연하고, 변경이 용이
- 확장 가능한 설계
- 클라이언트에 영향을 주지 않는 변경 가능
- 인터페이스를 안정적으로 잘 설계하는 것이 중요

다형성(Polymorphism)

역할과 구현을 분리

한계

- 역할(인터페이스) 자체가 변하면, 클라이언트, 서버 모두에 큰 변경이 발생한다.
- 자동차를 비행기로 변경해야한다면?
- 대본 자체가 변경된다면?
- USB 인터페이스가 변경된다면?
- 인터페이스를 안정적으로 잘 설계하는 것이 중요

다형성(Polymorphism)

스프링과 객체 지향

- 다형성이 가장 중요하다!
- 스프링은 다형성을 극대화해서 이용할 수 있게 도와준다.
- 스프링에서 이야기하는 제어의 역전(IoC), 의존관계 주입(DI)은 다형성을 활용해서
  역할과 구현을 편리하게 다룰 수 있도록 지원한다.
- 스프링을 사용하면 마치 레고 블럭 조립하듯이! 공연 무대의 배우를 선택하듯이! 구현을 편리하게 변경할 수 있다.

다형성의 구현- 오버로딩(Overloading)

오버로딩

1. 오버로딩은 여러 종류의 타입을 받아들여 결국엔 같은 기능을 하도록 만들기 위한 작업.
2. 이 역시 메소드를 동적으로 호출할 수 있으니 다형성이라고 할 수 있다.
3. 하지만 메소드를 오버로딩하는 경우 요구사항이 변경되었을 때 모든 메소드에서 수정이 수반된다.
4. 따라서 필요한 경우에만 적절히 고려하여 사용하는 것이 좋다.

오버로딩의 장점

1. 오버로딩을 통해 여러 메서드들이 하나의 이름으로 정의
2. 오류의 가능성을 많이 줄일 수 있다.
3. 메서드의 이름을 보고 기능을 유추할 수 있다.
4. 메서드의 이름 절약

```java
public void print(boolean b) { this.write(String.valueOf(b)); }

public void print(char c) { this.write(String.valueOf(c)); }

public void print(int i) { this.write(String.valueOf(i)); }

public void print(long l) { this.write(String.valueOf(l)); }

public void print(float f) { this.write(String.valueOf(f)); }

public void print(double d) { this.write(String.valueOf(d)); }

public void print( @NotNull char[] s) { this.write(s); }

public void print( @Nullable String s) { this.write(String.valueOf(s)); }

public void print( @Nullable Object obj) { this.write(String.valueOf(obj)); }

public void println() { this.newLine(); }

public void println(boolean x) {
    synchronized(this) {
        this.print(x);
        this.newLine();
    }
}
```

다형성의 구현- 오버라이딩(Overriding)

오버라이딩

1. 오버라이딩은 상위 클래스의 메서드를 하위 클래스에서 재정의하는 것.
2. 따라서 여기서는 상속의 개념이 추가된다.

   *MemberRepository를 상속 -〉 MemoryMemberRepository로 구현

```java
public interface MemberRepository {
    //4가지 기능을 만듬
    Member save(Member member); //회원이 저장소에 저장
    Optional<Member> findByID(Long id);
    //Optional는 자바8에 들어가 있는 기능
    Optional<Member> findByName(String name);
    List<Member> findAll(); //저장된 회원리스트를 전부 반환
}
```

```java
//구현체
public class MemoryMemberRepository implements MemberRepository {

    private static Map<Long, Member> store = new HashMap<>();
    private static long sequence = 0L;
    //sequence는 0,1,2키값을 생성해주는 애

    @Override
    public Member save(Member member) {
        member.setId(++sequence);
        store.put(member.getId(), member);
        return member;
    }

    @Override
    public Optional<Member> findByID(Long id) {
//        return store.get(id);
        return Optional.ofNullable(store.get(id));
        //null이 반환될 가능성이 있으면 optional 로 감싼다. 결과가 없으면 null이 반환
    }
```

오버로딩의 장점

1. 오버로딩을 통해 여러 메서드들이 하나의 이름으로 정의
2. 오류의 가능성을 많이 줄일 수 있다.
3. 메서드의 이름을 보고 기능을 유추할 수 있다.
4. 메서드의 이름 절약

다형성의 구현- 오버라이딩(Overriding)

```java
package org.opentutorials.javatutorials.polymorphism;
abstract class Calculator{
    int left, right;
    public void setOprands(int left, int right){
        this.left = left;
        this.right = right;
    }
    int _sum() {
        return this.left + this.right;
    }
    public abstract void sum();
    public abstract void avg();
    public void run(){
        sum();
        avg();
    }
}
```

추상클래스

```java
class CalculatorDecoPlus extends Calculator {
    public void sum(){
        System.out.println("+ sum :"+_sum());
    }
    public void avg(){
        System.out.println("+ avg :"+(this.left+this.right)/2);
    }
}
class CalculatorDecoMinus extends Calculator {
    public void sum(){
        System.out.println("- sum :"+_sum());
    }
    public void avg(){
        System.out.println("- avg :"+(this.left+this.right)/2);
    }
}
public class CalculatorDemo {
    public static void main(String[] args) {
        Calculator c1 = new CalculatorDecoPlus();
        c1.setOprands(10, 20);
        c1.run();

        Calculator c2 = new CalculatorDecoMinus();
        c2.setOprands(10, 20);
        c2.run();
    }
```

추상클래스

추상클래스(abstract class) - 미완성 설계도

- 클래스인데, 미완성 메서드를 포함하고 있다는 의미
  *미완성 메서드 = 추상 메서드
- 그 외적인 부분은 일반클래스와 같다.
  -> 추상클래스에도 생성자가 존재, 멤버변수와 메서드도 존재 가능
  abstract class 클래스 이름
- 추상클래스로 인스턴스를 생성할 수 없다
  = 객체를 생성할 수 없다
  -> 미완성 설계도이기 때문
- 상속을 통한 자손클래스에 의해서만 완성 가능

추상메서드(abstract method)

- 메서드는 선언부와 구현부(몸통)로 구성
- 추상메서드
- -> 선언부만 작성, 구현부는 미작성
  = 실제 수행될 내용은 작성하지 않았기 때문에 미완성 메서드
- 미완성상태로 남겨 놓는 이유
  -> 메서드의 내용이 상속받는 클래스에 따라 달라질 수 있기 때문
  -> 따라서 조상클래스에서 선언부만 작성
  -> 실제 내용은 상속받는 클래스에서 구현
- 추상메서드를 만들 때는 키워드 'abstract'를 앞에 붙여준다.
  abstract 리턴타입 메서드이름();

추상클래스? 인터페이스?

인터페이스(interface)

- 추상클래스 = 미완성 설계도
  인터페이스 = 기본 설계도
- 인터페이스도 다른 클래스를 작성하는데 도움을 주는 목적
- 일종의 추상클래스
- 몸통을 갖춘 일반 메서드 또는 멤버변수를
  구성원으로 가질 수 없다.
- 오직 추상메서드와 상수만을 멤버로 가질 수 있다.
- 다중상속(구현)이 가능
  * 인터페이스는 가능, 클래스는 불가능

추상클래스 vs 인터페이스

- 상속은 슈퍼클래스의 기능을 이용하거나 확장하기 위해 사용되고,
  다중상속의 모호성 때문에 하나만 상속 가능.
- 인터페이스는 해당 인터페이스를 구현한 객체들에 대해서
  동일한 동작을 약속하기 위해 존재.

사용의도

- 추상클래스는 IS-A "~이다".
  인터페이스는 HAS-A "~을 할 수 있는".
- 추상클래스는 상속을 받아서 그 기능을 이용, 확장시키는데 목적
  인터페이스는 함수의 껍데기만 있어서, 그 함수의 구현을 가제하기 위함.
  구현 객체의 같은 동작을 보장

공통된 기능 사용 여부

- 자바가 다중상속을 지원하지 않기 때문.
  다중상속은 여러 개의 슈퍼클래스를 두는 것을 의미