Java 객체지향 심화 복습 Summary

회고 6팀

팀장윤수정,

이환진, 남혜린,김도은, 이문주, 최형규, 백종현, 진솔빈

1. 상속(Inheritance, 물려받기)

- **부모의 기능을 자식이 물려받아 재사용.**

- **비유: 부모가 운전 능력을 가지고 있으면, 자식도 자동차를 운전할 수 있음.**

```
// 부모 클래스
class Vehicle {
    void move(){ System.out.println("이동"); } // 이동 기능
}

// 자식 클래스 (Vehicle을 상속받음)
class Car extends Vehicle {
    void openTrunk(){ System.out.println("트렁크 열기"); } // 자식만의 기능
}

Car c = new Car();
c.move();       // 부모 기능 사용
c.openTrunk();  // 자식 기능 사용
```

2. 추상화 + 추상클래스 (Abstraction)

1. 공통된 틀만 만들고, 자세한 내용은 자식이 구현.

2. Abstract class 는 직접 객체를 만들 수 없음

3. 비유 : "동물"이라는 설계도는 있지만, 실제로는 고양이·강아지 같은 구체적인 존재가 필요함

```
// 추상 클래스 (설계도 역할, 객체 직접 생성 불가)
abstract class Shape {
    abstract double getArea(); // 자식이 반드시 구현해야 하는 메서드
}

// 원 (구체적인 자식 클래스)
class Circle extends Shape {
    double r;
    Circle(double r){ this.r=r; }

    // 추상 메서드 구현 (구체화)
    double getArea(){ return Math.PI*r*r; }
}
```

3. 오버로딩 (Overloading, 같은 이름 다른 방법)

1. 같은 이름, 다른 매개변수.

2. 비유: “문 열어!”라고 말했을 때,

집 문일 수도, 차 문일 수도 있음.

```
// 같은 이름(print), 다른 입력(매개변수)
void print(int n){ System.out.println(n); }       // 정수 출력
void print(String s){ System.out.println(s); }    // 문자열 출력

print(10);       // 정수 출력
print("안녕");  // 문자열 출력
```

4. 오버라이딩 (Overriding, 부모 기능 다시 만들기)

- 부모 메서드를 자식이 재정의.
- 비유: “울어!”라는 기능은 같지만, 개는 멍멍, 고양이는 야옹.

```
class Animal {
    void sound(){ System.out.println("소리"); } // 기본 기능
}

class Dog extends Animal {
    @Override
    void sound(){ System.out.println("멍멍"); } // 부모 기능 고쳐씀
}

Dog d = new Dog();
d.sound(); // "멍멍"
```

5. 다형성 (Polymorphism, 여러 모습)

- 하나의 타입(부모)으로 여러 종류(자식)를 다룰 수 있음.
- 비유: “자동차”라고 부르면, 실제로는 버스일 수도, 택시일 수도 있음.
- 업캐스팅과 오버라이딩이 함께 쓰임.

```
class Animal { void sound(){ System.out.println("소리"); } }
class Dog extends Animal { @Override void sound(){ System.out.println("멍멍"); } }
class Cat extends Animal { @Override void sound(){ System.out.println("야옹"); } }

// 부모 타입으로 여러 자식 객체를 다룸
Animal a1 = new Dog();
Animal a2 = new Cat();

a1.sound(); // 실제로는 "멍멍"
a2.sound(); // 실제로는 "야옹"
```

6. 인터페이스 (Interface, 규칙서)

1. “해야 할 기능 목록”만 적어둔 설계도.

2. 특징:
   - 모든 변수 = public static final (상수)
   - 모든 메서드 = public abstract (자동)
   - JDK 8부터 default, static 메서드 가능

3. 비유: 콘센트 규격이 같으면, 다른 회사 제품도 꽂아서 사용 가능.

```
// 규칙서: 반드시 draw()를 만들어야 함
interface Drawable {
    void draw();
}

// 규칙을 따른 클래스
class Photo implements Drawable {
    public void draw(){ System.out.println("그리기"); }
}

Drawable d = new Photo();
d.draw(); // "그리기"
```

7. 캡슐화 + 접근제한자 (Encapsulation & Access Modifier)

1. 캡슐화: 데이터(필드)를 숨기고 메서드를 통해서만 접근.

2. 접근제한자:

private : 같은 클래스 안에서만

(default): 같은 패키지에서만

protected : 같은 패키지 + 자식 클래스

public : 어디서나 접근 가능

3. 비유 : 약통 뚜껑은 바로 못 열고, 의사(Setter 메서드)를 통해서만 약 꺼내기 가능.

7. 캡슐화 + 접근제한자 (Encapsulation & Access Modifier)

```
class Employee {
    private int salary; // 직접 접근 불가 (은닉)

    // setter: 값을 넣을 때 검사 가능
    public void setSalary(int s){
        if(s < 0) throw new IllegalArgumentException("급여는 음수 불가!");
        this.salary = s;
    }

    // getter: 값을 안전하게 꺼내옴
    public int getSalary(){ return salary; }
}

Employee e = new Employee();
e.setSalary(3000);        // 안전하게 값 입력
System.out.println(e.getSalary()); // 안전하게 값 출력
```

8. 업캐스팅 / 다운캐스팅 (형 변환)

1. 업캐스팅 : 자식 -> 부모 (자동, 안전)
2. 다운캐스팅 : 부모 -> 자식 (강제, 위험)
3. 비유 :

업캐스팅 : “버스는 자동차다“ -> 자동차로 불러도 문제 없음.

다운캐스팅 : “자동차를 버스라고 부르기“ -> 잘못하면 사고 남

```
class Animal { void sound(){ System.out.println("소리"); } }
class Dog extends Animal { @Override void sound(){ System.out.println("멍멍"); } }

Animal a = new Dog();   // 업캐스팅 (자동 변환)
a.sound();              // 실제로는 "멍멍"

Dog d = (Dog) a;        // 다운캐스팅 (강제 변환)
d.sound();              // "멍멍"
```

9. instanceof (진짜 타입 확인)

1. 객체가 어떤 타입인지 확인하는 연산자
2. 비유 : “이 사람이 학생이야? 물어본 뒤, 맞으면 학생 기능 사용

```
Animal a = new Dog();

if(a instanceof Dog){    // 진짜로 Dog인지 확인
    Dog d = (Dog) a;     // 안전하게 다운캐스팅
    d.sound();           // "멍멍"
}
```

✅ 한눈에 정리

- 상속: 부모 기능 물려받기 → 코드 재사용
- 추상클래스/추상화: 뼈대만 만들고 자식이 완성
- 오버로딩: 같은 이름, 다른 입력
- 오버라이딩: 부모 기능 고쳐쓰기
- 다형성: 한 타입으로 여러 객체 다루기
- 인터페이스: 규칙서(완전한 추상화)
- 캡슐화 + 접근제한자: 데이터 숨기고 보호하기
- 업/다운캐스팅 + instanceof: 타입 변환 & 안전 검사

오늘의 회고 - 남혜린

객체가 시작되고 나서부터는 진도를 따라가기가 조금 버거워졌다.
그래서 인강을 서브로 같이 들으면서, 교안에 있는 예제 코드를 카피해 직접 쳐보는 연습을 하고 있다.
언젠가는 안 보고도 코드를 짤 수 있을 만큼 익숙해지길 바라면서..!

이번 주 목표는 스터디 전에 최대한 진도를 따라잡아서 내용을 자신 있게 공유하는 것.
또 백준은 제일 쉬운 단계부터 하루에 1~2문제씩 꾸준히 풀고 있는데,
문제 풀이 자체보다는 작은 습관을 만드는 데 집중하고 있다.

오늘의 회고 - 백종현

오늘은 객체지향의 상속과 다형성에대해 더 자세히 알아보는 시간을 가졌습니다. 완벽하게는 아니지만 다른 팀원 분들이 조금식 이해해 가시는 모습을 보며 저 또한 많은 자극을 받았습니다.

또한 취직에 대한 담소를 나누었는데 형규님의 면접관 입장을 들으며 많은 생각을 할 수 있었습니다. 남은 기간 동안 다함께 멋진 개발자로성장하면 좋을것 같습니다!

오늘의 키워드

- 상속
- 추상화
- 오버로딩/오버라이딩
- 인터페이스
- 인터페이스의 모든 필드는 public static final
- 캡슐화
- 업캐스팅/다운캐스팅
- instanceof

오늘의 회고 - 윤수정

오늘은 상속, 추상클래스, 인터페이스 개념을 배웠다. 특히 추상클래스와 인터페이스를 비교하며 차이점을 이해하는 과정이 흥미로웠다.

추상클래스는 상위(부모) 클래스가 하위(자식) 클래스에게 공통 필드와 구현 메서드를 물려주면서, 반드시 구현해야 하는 추상 메서드를 정의해 주는 구조라고 이해했다. 여기서 "강제한다"는 표현은, 다른 면에서 보자면 하위 클래스가 어떤 기능을 반드시 오버라이드해야 하는지를 명확히 알려준다는 의미다. 덕분에 코드 작성 시 구현해야 할 부분이 뚜렷해져서 불필요하게 헤맬 일이 줄어든다고 느꼈다.

인터페이스는 기능의 설계서 역할을 하는 구조로 생각했다. 추상 메서드(필요 시 static, default 메서드 포함)를 정의해 두고, implements한 클래스들이 그 기능의 구체적인 내용을 채우도록 하는 방식이다.
두 개념 모두 코드 중복을 줄이고 다형성을 실현할 수 있어, 런타임 시 실제 생성된 객체의 타입에 맞는 오버라이드된 메서드가 실행되는 구조가 무척 유연하고 효율적이라고 느꼈다.

또한 회고 시간에 팀원들과 학습 진도를 나누고 미래 취업 방향성에 대해 이야기 하며, 나 스스로 실력을 꾸준히 키우고 팀 안에서 협업 능력을 쌓아가는 것이 정말 중요하다는 생각이 들었다. 앞으로 더 공부하고, 로직 설계와 구현에 익숙해지도록 노력해야겠다. 아자!

오늘의 회고 - 최형규

객체 심화에 오면서 "와, 큰일났다."싶었지만
개념이해하거나 암기만 하고 넘어갔던 부분들이
마치 흩어진 조각이 하나하나 맞춰져가는 느낌이 들었다.
물론 코드를 직접 작성하는데까지는 시일이 좀 걸리겠지만
너무 조급해하지 않고 하나씩 차근차근 해내자

문주님이 새롭게 합류하면서 또다른 시각에서 말씀하시는 부분이 좋았고,
배운 것을 회고시간 때 다시 듣고 하는 과정이 노는듯하면서도 재밌게 다시 복습할 수 있었다.
무엇보다 이번 주 자바 기초 박살내자!!

그리고 오늘 아침 러닝도 다녀왔는데, 가볍게 다녀오니 공부하는데는 더 좋았다.
그리고 팀 개개인 모두 특별한 강점을 지닌 육각형 개발자로 성장하길 기대한다.

오늘의 키워드
- 추상클래스 vs 인터페이스
- 오버로딩 vs 오버라이딩
- instanceof
- 캡슐화

오늘의 회고 - 이환진

오늘은 하루종일 정신을 놓고 듣고있었다.
수업내용은 기억이날라고하면 다형성부터 오브젝트 시작했는데 정확한 개념을 이해를 못했다
강사님도 정말 잘 설명하시지만 나는 아직 제대로 갈피잡지못해서 질문하려면 어디서부터 질문해야할지 잘몰라서 질문을못했다.

앞으로 내가 해야하는일은 복습을 꾸준히해야만하는것이다. 문법이 중요하지않다고하지만 결국엔 코딩은 기본기부터 잡아야 꼬을 피울수있기때문에 기본기를 굳히고 강의에 대한 내용을 이해해야겠다. 앞으로는 잠을 11시에자서 아침 6시에 일어나서 공부하는습관을 시작해야겠다.

오늘의 회고 – 진솔빈

오늘은 클래스랑 상속 부분을 다시 공부했다. 부모 클래스에 있는 필드랑 메서드를 자식이 물려받는다는 게 핵심인데, 막상 정리하다 보니 단순한 개념 같아도 은근히 디테일이 많았다. 예를 들어, 부모가 가진 private 멤버는 자식이 바로 쓸 수 없다는 점이라든지.

또 추상 클래스 같은 경우는 자식이 상속을 통해서만 확장할 수 있다는 것도 다시 한 번 짚었다. 상속은 하나의 부모만 가능하다는 단일 상속 원칙도 다시 떠올렸는데, 인터페이스는 여러 개를 붙일 수 있어서 '자격증' 같다는 표현이 참 와 닿는다. 인터페이스는 결국 "이 기능은 반드시 구현해야 한다"는 계약 같은 거라, 구조를 짤 때 유용하게 쓰일 수 있겠다 싶었다.

이론만 읽으면 단순해 보이는데, 막상 코드로 구현하다 보면 어디서 막히는 구석이 생긴다. 그래도 오늘은 정리하면서 조금 더 개념이 깔끔해진 것 같다.

오늘의 회고 - 이문주

객체 지향의 중심이 다형성이라는 것을 새롭게 알게 되었다.
앞으로 수업을 이론 흐름 파악 정도로 활용하고 중간에 설명이 부족하거나 이해가 안되는 부분은
무조건 필기를 해서 구글링을 통해 더 깊게 학습하는 시간을 가질 것이다.

그리고 나머지 시간은 그 전 내용 누적 복습 및 문제 풀이를 하면 도 좋을 것 같다