📖 Java Study Skill Up Team Review

[프로그래밍 언어]

- 프로그래밍(코딩)
- 프로그래밍?
    - 사람이 컴퓨터에게 일을 시키는 것
    - But 컴퓨터는 사람의 말을 알아들을 수 없다.
    - 그렇기에 프로그래밍 언어가 필요하다.
- 언어?
    - 사람이 사용하는 언어
        - 컴퓨터 입장에서 보면 이해할 수 없는 문자 집합이다.
    - 기계어
        - 0과 1로 이루어진 이진 코드이기 때문에 사람이 이해하기는 매우 어렵다.
        - 왜 이진 코드?
            - 내부 동작과 하드웨어 구성과 관련이 있다.
            - 트랜지스터는 전기 신호를 켜고 끄는 역할을 하며, 이 두 가지 상태를 사용하여 정보를 저장하고 처리한다.
            - 즉 컴퓨터는 이러한 트랜지스터의 전기 신호를 ON(1)과 OFF(0) 상태로 사용하여 정보를 저장하고 처리하는데, 이는 컴퓨터에서 가장 정확성이 높고 효율적이라고 할 수 있다.
- 프로그래밍 언어?
    - 컴퓨터가 이해할 수 있는 언어로써 사람과 컴퓨터가 소통하기 위해 필요한 언어이다.

[컴파일러]

- 사람이 작성한 소스를 기계어로 변환시켜 주는 소프트웨어이다.

[외부 라이브러리]

- External Libraries
- System 이라는 것도 자바에서 제공하는 파일이다.
- System.out,.println("안녕");
- 인텔리J 가 선택한 JDK에 포함되어 있다. (Project Settings > Project > SDK)
- JDK 대부분은 거의 공일하게 표준 라이브러리를 제공하고 있다.

[변수 사용 이유]

- 프로그램은 작업을 처리하는 과정에서 필요에 따라 데이터를 메모리에 저장해야 한다. 이때 사용할 수 있는 게 변수다.
- 즉 변수는 값을 저장할 수 있는 메모리의 공간을 의미한다.

[리터럴과 상수]

- 리터럴 == 상수 / 같은 의미지만 프로그램에서는 다른 용어로 사용한다.
  - 상수: 값을 한 번 저장하면 변경할 수 없는 변수
  - 리터럴: 리터럴은 변수에 저장되지 않고 그 자체로 사용되는 값

[이스케이프]

- '\t' : 수평 탭
- '\n' : 줄 바꿈
- '\r' : 리턴
- '\”' : 큰 따옴표(")
- '\'' : 작은 따옴표(')
- '\' : 역슬래쉬(\)

[연산자 사용 시 리소스 절약]

```
boolean bool1 = a < b && c ++ < (d +=3);
```

- 사소하지만 연산 부하가 적은 코드를 앞에 작성함으로써 리소르를 절약할 수 있다.

[문자열 literal과 객체 인스턴스]

```
String hello1 = "Hello";
String hello2 = "Hello";
String world = "World";

// 리터럴 끼리는 비교 연산자인 '동등연산자(== )'를 사용해서 비교가 가능하다.
hello1 == hello2; // true
String hello3 = new String("Hello");
String hello4 = new String("Hello");
String hello5 = hello4; // hello5를 hello4 그대로 초기화 (같은 주소를 참조)

// 인스턴스와 비교할 때에는 equals 메서드를 사용해야 한다.
hello3 == hello4;       // false
hello3.equals(hello4);  // true
```

- 자바에서 문자열 리터럴은 String Constant Pool 이라는 특별한 영역에 저장이 된다.
- String pool은 Heap 메모리 내에 위치하지만, 일반적인 Heap 영역과는 다른 메모리 영역이다.
- 문자열 리터럴은 프로그램이 실행될 때 메모리에 한 번만 생성되며, 같은 문자열 리터럴을 참조하는 모든 부분은 동일한 객체를 공유한다.
- new String() 키워드를 사용하여 문자열을 생성하면 새로운 string 객체가 Heap 메모리에 생성된다.
- 이 경우 항상 새로운 객체가 생성되며, 문자열 내용이 같더라도 독립적인 객체가 된다.
- 정리하자면, 문자열 리터럴은 String Pool에서 관리되며 동일한 객체를 봉유한 반면 new String을 사용하면 새로운 객체가 생성되며, 동일한 문자열 내용이라도 독립적인 객체가 된다.

- overflow
    - 타입이 허용하는 최대값을 벗어나는 것
- underflow
    - 타입이 허용하는 최소값을 벗어나는 것
- NaN
    - Not a Number
- Infinity
    - 무한대

[상속]

- 상속(Inheritance[인헤러턴스])은 프로그램에서도 부모 클래스의 필드와 메서드를 자식 클래스에 물려줄 수 있는 것이다.
- 예제로 부모 클래스, 자식 클래스, 실행 클래스를 생성하였다.

[상속의 사용 이유]

- 상속은 간단하게 말하자면 이미 잘 개발된 클래스를 재사용 할 수 있다.
- 그렇기에 중복 코드를 줄여 개발 시간을 단축시켜 준다.
- 또 추가 작성해야 하는 필드와 메서드가 상속받은 클래스에 존재한다면 추가 작성할 필요가 없어 효율적이다.
- 부모 클래스를 수정하면 모든 자식 클래스에 수정되기 때문에 클래스의 수정을 최소화할 수 있다.

[다중 상속 불허용]

✋ 여러 개의 부모 클래스를 상속 받을 수 있나요?

✕ 정답은 '없다'이다.

- 다중 상속을 허용하는 언어는 존재하지만 자바에서는 다중 상속을 허용하지 않는다.
- 그 이유는 다중 상속을 받은 부모 클래스에서 동일한 명칭의 메서드가 존재할 수 있다.
- 이 경우 어떤 부모 클래스에 접근할지에 대해 모호해지기 때문에 다주 상속을 허용하지 않는다.

[생성자]

- 생성자는 객체를 초기화하여 필드나 메서드를 사용할 수 있도록 준비하는 데 사용
- 매개변수를 요구하지 않으면 컴파일 과정에서 자동으로 추가된다.

[스택(stack) 영역과 힙(heap) 영역]

```
// 기본 타입 변수
int age = 11;
double price = 100.5;

// 참조 타입 변수
```

```
String name = "비프";
String hobby = "공부";
```

- 변수를 생성하면 이 변수들은 모두 스택이라는 메모리 영역에 생성된다.
- 기본 타입 변수인 age와 price는 직접 값을 저장하고 있다.
- 참조 타입 변수인 name과 hobby는 힙 메모리 영역의 String 객체 번지를 저장하고 이 번지를 통해 String 객체를 참조한다.

[메모리 사용 영역]

- java 명령어로 JVM이 구동되면 JVM은 운영체제에서 할당받은 메모리 영역을 다음과 같이 구분해서 사용한다.

[메서드 영역]

- 바이트코드 파일을 읽은 내용이 저장되는 영역으로 클래스별 상수, 정적 필드, 메서드 코드, 생성자 코드 등이 저장된다.

[힙 영역]

- 객체가 생성되는 영역
- 객체의 번지는 메서드 영역과 스택 영역의 상수와 변수에서 참조할 수 있다.

[스택 영역]

- 메서드를 호출할 때마다 생성되는 프레임이 저장되는 영역이다.
- 메서드 호출이 끝나면 프레임은 자동 제거된다.
- 프레임 내부에는 로컬 변수 스택이 있다. 여기에서 기본 타입 타입 변수와 참조 타입 변수가 생성되고 제거된다.

[배열(Array) 타입]

- 변수는 하나의 값만 저장할 수 있다.
- 많은 양의 값을 다루기 위해 효율적인 방법은 배열을 사용하는 것이다.
- 배열은 같은 타입의 값만 관리한다. (int 배열은 int 타입의 값만 관리함, String 배열은 문자열만 관리함)
- 배열의 길이는 늘리거나 줄일 수 없다. (배열은 생성과 동시에 길이가 결정됨)

[객체]

- 정의는 '실세계에 존재하거나 생각할 수 있는, 인간이 생각하고 표현할 수 있는 모든 것'이다.
- '컴퓨터, 책상, 휴대폰 같은 사물은 물론이고, 강의나 수강 신청 같은 개념 존재하는 것 또한 객체'이다.
- 관점에 따른 개념
  - 모델링 : 객체는 명확한 의미를 담고 있는 대상 또는 개념이다.
  - 프로그래머 관점 : 객체는 `class`에서 생성된 변수이다.
  - 소프트웨어 개발 관점 : 메서드(함수)를 모아놓은 데이터 + 메서드 형태의 소프트웨어 모듈이다.
  - 객체지향 프로그래밍 관점 : 객체는 데이터와 함수를 속성(필드)과 동작(메서드) 용어로 구현한다.

```
쉽게 말해, 데이터(변수)와 데이터(변수)에 관련된 동작(함수)의 절차, 방법, 기능을 모두 포함한 개념이다.
=> 내가 서점에서 책을 사는 경우 : 속성인 '나(손님)'와 동작인 '책 구매'는 하나의 객체인 것이다.
=> 속성인 '서점 주인'과 동작인 '책 판매'도 하나의 객체라고 할 수 있다.
```

[객체 상호작용]

```java
package com.chapter.ch06.example.ex01;

public class ObjectInteraction {
  public static void main(String[] args) {
    Shop shop = new Shop();
    Person person = new Person(shop);
    String IWalk = person.walk("1km");

    System.out.println(IWalk);
  }
}

class Person {
  private Shop shop;

  Person(Shop shop) {
    this.shop = shop;
  }

  String walk(String distance) {
    return shop.rewardForWalking(distance);
  }
}

class Shop {
  String rewardForWalking(String distance) {
    return distance.equals("1km") ? "60kcal" : "";
  }
}
```

✍ 나만의 Scenario

- '나'는 Person 객체로, '상점'은 Shop 객체로 표현된다.
- '나'는 walk 메서드를 사용하여 '상점'에게 1km 만큼 걸었다고 알린다.
- '상점'은 이 정보를 기반으로 rewardForWalking 메서드를 통해 60kcal 의 보상을 제공한다.
- 이렇게 객체들은 서로 상호작용하면서 동작하는 것을 확인할 수 있다.

[객체 간 관계]

- 웹사이트를 예를 들면, 페이지당 다양한 기능과 영역을 나타내는 여러 객체로 구성되어 있다.
- 이러한 객체와의 상호 작용은 해당 필드와 메서드에 접근하거나 호출함으로써 이루어진다.

- 또 상속을 통해 객체 간의 기능을 확장하거나 재사용 할 수도 있다.

[객체 지향]

- 부품과 같은 객체들을 만들고 이 객체들을 조합해서 프로그램을 완성해 나가는 개발 기법이다.
- 즉, `class`로 생성된 객체들을 조합해서 프로그램을 완성해 나가는 개발 기법이 객체 지향이다.

[객체 지향 프로그래밍]

- 사람이 컴퓨터를 이용해서 객체들을 조합하여 프로그램을 완성해 나가는 개발 기법이다.

💎 객체 지향 프로그래밍 한 줄 설명은 정리했지만, 객체 지향 프로그래밍 핵심은 특징에 있다.

[객체 지향 프로그래밍 특징]

1. 캡슐화 : 필드와 메서드를 하나로 묶고 실제 구현 내용을 외부에 감추는 것

- 캡슐화는 내가 생성한 필드나 메서드에 해당하는 객체의 세부 구현을 숨기고 필요한 기능만을 외부에 제공함으로써 안정성과 유연성을 높이기 위한 기법이다.
  - 핵심 : **안전성**, **유연성**

> ✍ 나만의 비유 Scenario
>
> 내가 약사라고 가정하고 캡슐 알약을 제조한다고 하겠다.
> 캡슐 안에는 가루약이 존재하는데 이 가루약이 두통, 감기, 항암 등등 목적에 맞게 각각 제조되어 치료에 도움을 준다.
> 이 캡슐 알약의 제조 과정은 외부에 노출시키지 않고 필요한 환자들에게 캡슐 모양으로 제공한다.
> 환자는 필요한 알약만 약사에게서 제공받을 수 있어서 유연성이 높다고 할 수 있다.
> 또 알약이 제조되는 과정에는 환자가 관여하지 못하기 때문에 안정성이 높다고 할 수 있다.

2. 상속

- 상속은 하위 객체가 상위 객체를 상속 받아 필드와 메서드를 사용할 수 있는 기술이다.
  - 핵심 : **재사용성**, **유지 보수**

> 쉽게 이해하기
>
> 1. 재사용성
> 상위 객체의 기능을 잘 만들어 놓으면 상속 받을 하위 객체는 중복 코드 없이 이를 사용하거나 활용할 수 있다.
> 이를 통해 중복적인 코드가 필요 없어 코드의 재사용성이 향상된다.
>
> 2. 유지 보수
> 잘 구성된 상위 객체의 기능을 필요에 의해 상속 받은 하위 객체들이 많이 있을 수 있다.
> 이때 수정 작업을 하게 되어도 잘 만들어진 상위 객체의 소스만 수정하면 된다.
> 그러면 당연히 작업 시간이 단축됨과 동시에 잠재적인 오류가 줄어들어서 유지 보수에 용이하다고 할 수 있다.

3. 다형성

- 사용 방법은 동일하지만 실행 결과가 다양하게 나오는 성질을 말한다.
    - 핵심 : **유연성**, **확장성**

> 쉽게 이해하기
>
> "입력하다"라는 메서드를 가진 입력장치라는 기본 클래스(또는 인터페이스)를 상속받아 키보드와 휴대폰자판 클래스를 만들 수 있다.
> 이 두 클래스는 입력하다 메서드를 오버라이드하여 각각의 특성에 맞게 구현할 수 있다.
>
> 이처럼 다형성은 한 가지 형태의 인터페이스나 메서드를 제공하면서 다양한 구현을 할 수 있게 한다.
> 이를 통해 코드의 재사용성과 유지 보수성이 향상되며, 시스템의 확장성을 증가시키는 데 큰 도움을 준다.