# (디지털컨버전스) 스마트 콘텐츠와 웹 융합 응용SW개발자 양성과정

2021년 6월 1일
[ 17일차 복습 ]


- 수강생 : 김 민 규
- 강의장 : 강 남 C
- 수강 기간 : 2021. 05. 07 ~ 2021. 12. 08
- 수강 시간 : 15:30 ~ 22:00
- 이상훈 강사님 | 이은정 취업담임

## ▶ 내용 : 상속(extends)

```
+ class A {
+     int a = 10;
+
+     void b () {
+         System.out.println("A");
+     }
+ }
+
+ // extends 키워드가 바로 상속!
+ // 상속: 말 그대로 재산을 물려 받는것이다.
+ //       클래스의 내용물들을 활용할 수 있게 된다.
+ class AA extends A {
+     int a = 20;
+
+     void b () {
+         System.out.println("AA");
+     }
+     void c () {
+         System.out.println("C");
+     }
+ }
+
+ public class ExtendsTest {
+     public static void main(String[] args) {
+         A a = new A();
+         a.b();
+         System.out.println("A a: " + a.a);
+
+         AA aa = new AA();
+         aa.b();
+         aa.c();
+         System.out.println("AA aa: " + aa.a);
+
+         // new의 대상은 AA()이며
+         // 접근 데이터는 데이터타입 A를 참조해야한다.
+         A a1 = new AA();
+         a1.b();
+         System.out.println("A a1: " + a1.a);
+     }
```

## 주요 내용

--
상속( extends)를 통해
클래스의 내용을 활용할 수 있다.
--
New를 통해 객체를 생성(호출)하게되면

A a1 = new AA();에서
New의 대상은 A가 될 것이고,
접근데이터는 A를 따라가게 된다.

즉,
접근데이터는 객체 내부에 있는
데이터들을 의미하므로
객체 내부에 있는 데이터를 따라감.

## ▶ 내용 : 상속 – super

```java
class Car {
    private float rpm;
    private float fuel;
    private float pressure;
    private String color;

    public void setRpm (float rpm) {
        this.rpm = rpm;
    }
    public float getRpm() {
        return rpm;
    }
    public float getFuel() {
        return fuel;
    }
    public void setFuel(float fuel) {
        this.fuel = fuel;
    }
    public float getPressure() {
        return pressure;
    }
    public void setPressure(float pressure) {
        this.pressure = pressure;
    }
    public String getColor() {
        return color;
    }
    public void setColor(String color) {
        this.color = color;
    }
}
```

```java
// 기존에 잘 만들어진 정보에 새로운 내용을 추가하여 작업하고자 한다.
// 내용을 변경하는것보다는 새로운 클래스에 상속을 활용하여 작업하는 것을 권장한다.
// (일전에 잠깐 언급했던 SRP 규칙 때문에 그렇다)
class SportsCar extends Car {
    private Boolean booster;

    public Boolean getBooster() {
        return booster;
    }
    public void setBooster(Boolean booster) {
        this.booster = booster;
    }

    @Override
    public String toString() {
        // super의 경우엔 상속해준 상속자를 직접 호출한다.
        return "SportsCar{" +
                "rpm=" + getRpm() +
                ", fuel=" + getFuel() +
                ", pressure=" + getPressure() +
                ", color=" + getColor() +
                ", booster=" + booster +
                '}';
    }
}

public class CarTest {
    public static void main(String[] args) {
        SportsCar sc = new SportsCar();

        sc.setRpm(100);
        sc.setFuel(2.5f);
        sc.setPressure(1.0f);
        sc.setColor("Dark Gray");
        sc.setBooster(false);

        System.out.println(sc);
    }
}
```

## 주요 내용

--

super(~~, ~~, ~~, ~~);

Super()는 무엇이 되었든 상속자인 부모를 호출하게된다.

Super.toSTring()은 부모 클래스의 toSTring()을 호출하는 것

--

▶ 내용 : interface 추상화

**//작성방법**
1. 일단 interface를 적는다.
2. 인터페이스명(일종의 클래스 같은 것이라고 보면 됨)을 적는다.
3. 인터페이스 내부에는 매서드 프로토타입을 작성한다.
   (프로토타입이 뭘까요? 매서드의 접근 제한자, 리턴타입, 매서드 이름, 입력등을 기록한 형태)

---

**// 추상화란 무엇인가????**
객체 〈〈〈=== 대표적인 추상화의 예
객체 〈〈〈=== 현 시점에서 우리는 무엇을 생각하는가?
new, 메모리에 올라간 데이터들 혹은 정보들 ...
단어가 어떤 함축된 의미를 포함해버렸음(우리는 알게 모르게 사용하고 있었고)
객체란 단어만 보고도 이것이 어떻게 어떻게 형성되었는지 등이 이미 뇌리에 스치고 있음

---

OOP(객체지향)에서 제일 중요시 여기는 것이 바로 추상화다.
현재까지의 내용을 토대로 추상화란 궁극적으로 무엇을 추구하는것인가?

---

복잡하고 어렵고 토나오는것은 우리가 해줄게(자바 라이브러리 개발자 진영 및 스프링 프레임워크 개발 진영)
라이브러리 사용자들은 편하게 API 사용해서 개발만 하세요 ~
이런입력 ---〉 Black Box(블랙 박스) ---〉 요런 출력이 나와요
++ 굳이 모든 내용을 알지 않더라도 기능을 알 수 있는것
ex) sout() --〉 우리는 단순히 출력한다라는 기능만을 인지하고 사용하고 있음.

++ 킨다의 개념은 여러개가 있지만, 하나로 그것을 사용하겠다. 이것이 인터페이스를 쓰는 이유

▶ 내용 : interface 추상화

**추상화 Abstraction**
기본 자료형으로는 저장할 수 없어서 새로운 자료형(객체)이 필요할 때 그것의 설계도(클래스)를 만드는 것
※ abstract(추상 클래스)를 쓰지 않아도, Class를 만드는 것 자체를 추상화라고 함

---

컴퓨터 관점에서 생각해보면, **추상화란 데이터나 프로세스 등을 의미가 비슷한 개념이나 표현으로 정의해 나가는 과정**이면서 **동시에 각 개별 개체의 구현에 대한 상세함은 감추는 것,** 이것이 추상화

예를 들어, 프로그래밍 언어에서 **for, while, foreach 등은 반복을 추상화한 것들**.
실제로 이것들은 CPU의 이동 명령을 통해서 구현되겠지만, 이 구현으로부터 **'반복'이라는 개념을 뽑아내서 for, while 등으로 추상화한 것**.
비슷하게 OS는 그래픽 기능을 위한 API를 제공하는데, 이 API는 서로 다른 그래픽 카드들의 공통된 기능을 추상화한 결과물

---

간단히 말해서 추상화는 **객체의 관련 속성 만 '표시'하고 불필요한 세부 정보는 숨긴것.**
즉, 사용자에게 구현 세부 정보를 숨기고 **최종 사용자에게 기능 만 노출하고**
**사용자는 '어떻게 하는가'보다는 '그것이하는 일'만 알 수 있다**

---

**OOP의 추상화 유형**
**# 1) 데이터 추상화**
데이터 추상화에서 우리는 대부분 복잡한 데이터 유형을 만들고 구현을 숨 깁니다. 구현의 세부 사항으로 이동하지 않고 이러한 **데이터 유형을 조작하는 작업 만 노출**합니다.
이 접근 방식의 한 가지 장점은 사용자에게 **노출되는 동작을 변경하지 않고 언제든지 구현을 변경할 수 있다**는 것입니다.
**# 2) 제어 추상화**
제어 추상화는 **애플리케이션의 일부인 모든 제어문을 수집하고 이를 하나의 단위로 노출**합니다. 이 기능은이 제어 장치를 사용하여 작업 기능을 수행해야 할 때 사용됩니다.
제어 추상화는 구조화 된 프로그래밍의 기본 단위를 형성하며 제어 추상화를 사용하여 **복잡한 프레임 워크에 간단한 기능을 정의** 할 수 있습니다.