# (디지털컨버전스)스마트 콘텐츠와 웹 융합응용SW개발자 양성과정

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 –Joongyeon Kim(김중연)
jjjr69@naver.com

## 2021년 6월 2일 수업노트

[김중연]

```java
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
      // 접근 데이터 : 객체 내부에 있는데이터
      A a1 = new AA();
      a1.b();  //AA로 생성되었기 때문에 AA객체가 출력된다
      //a1.c(); A객체에 c가 존재하지 않기에 오류가 난다 (A a1 = new AA();로 생성할 경우 양 쪽에 동일한 이름의 변수가 존재해야 되는 겁니까?)
      System.out.println("A a1: " + a1.a);  //A객체에 직접 접근하는 것이기에 A에 있는 a가 출력되는 것인가??
    }
}
```

객체안에 있는 메소드를 호출함

A데이터를 불러와 더하는 것이기에 A의 a가 출력된다 데이터타입 은A이고 AA는 객체다

상속
상위클래스에 있는 내용물을 활용할수
있게해준다

하위클래스 extends 상위클래스 라고
입력하면 상속이 된다

```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; }
}

// 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);
    }
}
```

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

    public Vehicle(float rpm, float fuel, float pressure, String color) {
        this.rpm = rpm;
        this.fuel = fuel;
        this.pressure = pressure;
        this.color = color;
    }

    @Override
    public String toString() {
        return "Vehicle{" +
                "rpm=" + rpm +
                ", fuel=" + fuel +
                ", pressure=" + pressure +
                ", color='" + color + '\'' +
                '}';
    }
}

class Airplane extends Vehicle {
    private float aileron;
    private float pitch;
    private float rudder;

    public Airplane(float rpm, float fuel, float pressure, String color,
                    float aileron, float pitch, float rudder) {
        // super()는 무엇이 되었든 상속자인 부모를 호출한다.
        // super()만 적혀 있으니 생성자를 호출하게 된다.
        super(rpm, fuel, pressure, color);  //super로 부모클래스의 변수들을 호출하여 생성자에 대입할 수 있게 한다

        this.aileron = aileron;
        this.pitch = pitch;
        this.rudder = rudder;
    }

    @Override
    public String toString() {
        return "Airplane{" +
                // super.toString()은 부모 클래스의 toString()을 호출한 것이다.
                "super.Vehicle()=" + super.toString() +
                ", aileron=" + aileron +
                ", pitch=" + pitch +
                ", rudder=" + rudder +
                '}';
    }
}

public class InheritanceWithSuperTest {
    public static void main(String[] args) {
        Vehicle v = new Vehicle( rpm: 200,  fuel: 1.2f,  pressure: 1.0f,  color: "Red");

        System.out.println(v);

        Airplane a = new Airplane(
                rpm: 1000,  fuel: 112.5f,  pressure: 12.3f,  color: "White",
                aileron: 77.3f,  pitch: 0.02f,  rudder: 33.9f);

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

super
부모클래스의 변수들을 호출할 수 있게해준다

```java
import java.math.BigInteger;

class 이건희 {
    BigInteger money;
    String company;

    public 이건희(BigInteger money, String company) {
        this.money = money;
        this.company = company;
    }

    @Override
    public String toString() {
        return "이건희{" +
                "money=" + money +
                ", company='" + company + '\'' +
                '}';
    }
}

class 이재용 extends 이건희 {
    String recentInvest;

    public 이재용(BigInteger money, String company, String recentInvest) {
        super(money, company);
        this.recentInvest = recentInvest;
    }
```

```java
@Override
public String toString() {
    return "이재용{" +
            "money=" + money +
            ", company='" + company + '\'' +
            ", recentInvest='" + recentInvest + '\'' +
            '}';
}


blic class Prob53 {
    public static void main(String[] args) {
        BigInteger bigNum = new BigInteger( val: "100000000000000");
        이건희 samsung = new 이건희(bigNum,  company: "삼성그룹");

        이재용 newSamsung = new 이재용(bigNum.multiply(BigInteger.TEN),
                            // BigInteger.TEN은 BigInteger 클래스에서 제공하는 전역에서 접근 가능한 상수다
                company: "삼성전자",  recentInvest: "바이오 사업");

        System.out.println(samsung);
        System.out.println(newSamsung);
    }
```

```java
// 인터페이스 작성법
// 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, 리턴 타입, 매서드 이름, 입력등을 기록한 형태)
interface Remocon {
    public void turnOn();
    public void turnOff();

    // 리모콘 제조사가 15만개
    /*
    public void companyATurnOn();
    public void companyBTurnOn();
    public void companyZTurnOn();
    public void companyAATurnOn();
    public void companyABurnOn();
    public void companyAZurnOn();
    public void companyZZTurnOn();
    public void companyZZZTurnOn();
     */
    // ........................
    // TurnOn() 매서드만 몇 개 ? 15만개
}

// 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
```

인터페이스(interface)
사용자가 자세한 기능을 몰라도 일단 되게하는 것이다

쉽게 생각해서 내가 사용하는 쿠팡 윈터치구매에도 수많은 프로그래머의 에너지드링크가 소비되었다는 것을 알 수 있다

```
// KKK사의 컴퓨터를 킨다.
// GH사의 라디오를 킨다.
// A사의 리모콘을 킨다.     ========> 킨다(뭔진 모르겠지만)
// B사의 리모콘을 킨다.
// ................
// Z사의 리모콘을 킨다.

// 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 ~
// 이런 입력 ----> Black Box(블랙 박스) ---> 요런 출력이 나와요

// sout() ===> System.out.println()

class AbstractTest {
    Remocon rc = new Remocon() {  //인터페이스 객체를 생성하면 자동으로 @오버라이드가 출력된다
        @Override
        public void turnOn() {
            // 여기에 필요한 기능은 필요한 사람이 알아서 만드세요 ~
            System.out.println("나는 RC 자동차용 리모콘이야! RF 송수신기가 지금 활성화되었어!");
        }
        @Override
        public void turnOff() { System.out.println("이제 헤어질 시간이야! RF 송수신기 신호 출력을 차단할게!"); }
    };

    Remocon radio = new Remocon() {
```

```
    @Override
    public void turnOn() { System.out.println("나는 라디오야! 지금부터 주파수 채널 매칭을 시작할게!"); }

    @Override
    public void turnOff() { System.out.println("이젠 안녕! 주파수 채널 매칭을 끊을게!"); }
};

public void testMethod () {
    Remocon tv = new Remocon() { //오버라이드의 생성조건은 데이터타입이 인터페이스이면 나오는 것 같다.
        @Override
        public void turnOn() { System.out.println("나는 TV야! AM/FM 신호를 수신할게! 이제부터 방송을 보자!"); }

        @Override
        public void turnOff() { System.out.println("AM/FM 신호를 차단할게! 내일 또 보자!"); }
    };

    tv.turnOn();  //tv라는 이름의 새로운 리모콘 인터페이스 객체를 만들어서 호출했다. 메소드 내부에 tv로 인터페이스 객체를 만들었으니 호출된다
    radio.turnOff();

    Remocon airCon = new Remocon() {
        @Override
        public void turnOn() {
            System.out.println("냉방 시작!");
        }

        @Override
        public void turnOff() {
            System.out.println("냉방 종료!");
        }
    };
```

```
        };
        airCon.turnOn();

    }

    public void testMethod2 () {
        rc.turnOn();
        radio.turnOff();

    }
}

public class InterfaceTest {
    public static void main(String[] args) {
        AbstractTest at = new AbstractTest();

        at.testMethod();
        at.testMethod2();

    }
}
```

```java
interface Light{
    public void lightOn();
    public void lightOff();
}

class StreetLamp {
    Light streetlamp = new Light() {
        @Override
        public void lightOn() {
            System.out.println("가로등 켜짐");
        }

        @Override
        public void lightOff() {
            System.out.println("가로등 꺼짐");

        }
    };
}

class Lamp{
    Light lamp = new Light() {
        @Override
        public void lightOn() {
            System.out.println("램프 켜짐!");

        }

        @Override
        public void lightOff() {
            System.out.println("램프 꺼짐");
```

```
        }
    };
}

class Led{
    Light led = new Light() {
        @Override
        public void lightOn() {
            System.out.println("led 켜짐!");

        }

        @Override
        public void lightOff() {
            System.out.println("led 꺼짐");

        }
    };
}

public class Prob54 {
    public static void main(String[] args) {
        Lamp lamp = new Lamp();

        lamp.lamp.lightOn();//StreetLamp 클래스내부에  Light인터페이스 객체를 만듬으로써 클래스 내부에 있는 정보를 불러올 수 있다.
        lamp.lamp.lightOff();

        StreetLamp streetLamp = new StreetLamp();

        streetLamp.streetlamp.lightOn();
        streetLamp.streetlamp.lightOff();

        Led led = new Led();

        led.led.lightOn();
        led.led.lightOff();


    }
}
```

이 lightOn은 인터페이스에 생성된 lightOn을 불러오는 것이다