디지털 컨버전스 기반 UXUI Pront전문 개발자 양성과정

- 강사 이상훈
- gcccompil3r@gmail.com
- 학생 김도혜
- kimdohye0728@gmail.com

51번 복습하기

```java
private void doPayment () {

    int length = marketSellList.length;

    for (int i = 0; i < length; i++) {
        for (String element : userBuyList) {

            if (marketSellList[i].equals(element)) {

                myMoney -= marketSellListPrice[i] * userBuyListStock.get(userBuyList.indexOf(element));
                System.out.printf("찾은 물품 = %s, 가격 = %d, 수량 = %d\n",
                        element, marketSellListPrice[i],
                        userBuyListStock.get(userBuyList.indexOf(element)));



            }
        }
    }

    System.out.printf("현재 당신은 %d 원을 가지고 있습니다.\n", myMoney);

    userBuyListStock.clear();
    userBuyList.clear();
}
```

<결제코드 작성하기>

①우선, 판매리스트의 length를 입력한다.
②for문을 열어서 length만큼 반복해주고, foreach문을 만들어서 element에 구매리스트를 대입한다.
③이후 판매리스트의 인덱스와 element가 동일하다면, 소지금에서 물건값을 차감하여 대입한다.
④이 과정을 출력한다.
⑤결제완료 이후 카트를 비운다.

상속(기본개념)

```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");
    }
}
```

```java
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);
    }
}
```

이렇게 쓸 일이 있나요?

상속(예제 적용하기)

```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;
    }
}
```

```java
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(rpm, fuel, pressure, color);

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

super는 상속해준 부모를 직접 호출한다.

인터페이스개념 : 추상화

```
// 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

// 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)
```

상속은 공통 부분을 뽑아서 슈퍼 클래스로 만들고, 인터페이스는 상속에서 한걸음 더 나아가 기능만 뽑아간다.

->구글에 나온 내용인데 그게 그 말 아닌가...?

뭔진 정확히 모르겠는데 비슷한 기능을 뽑아서 사용하는 것 같다.

인터페이스(예제 적용하기)

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

①클래스를 만드는 것처럼 인터페이스를 만들어준다.
내부에 있는 메서드는 프로토타입이라고 한다.
이 메서드는 {}를 쓰지 않는다.
우리가 세부사항을 알지 않아도 되기 때문에.

```java
class Lamp {
    LampMethod lamp = new LampMethod() {
        @Override
        public void lightOn() {
            System.out.println("Lamp를 킵니다.");
        }

        @Override
        public void lightOff() {
            System.out.println("Lamp를 끕니다.");
        }
    };
```

```java
class Led {
    LampMethod led = new LampMethod() {
        @Override
        public void lightOn() {
            System.out.println("LED등을 킵니다.");
        }

        @Override
        public void lightOff() {
            System.out.println("LED등을 끕니다.");
        }
    };
}
```

```java
class StreetLamp {
    LampMethod streetLamp = new LampMethod() {
        @Override
        public void lightOn() {
            System.out.println("가로등을 킵니다.");
        }

        @Override
        public void lightOff() {
            System.out.println("가로등을 끕니다.");
        }
    };
}
```

②인터페이스를 적용할 클래스들을 세개 만들었다. 인터페이스 객체를 만든 후 오버라이드한다.
이 곳의 메서드들은 {}가 있는데, 이 부분은 직접 무언가를 만들 수 있다. **괄호 끝나면 꼭 ;를 찍어줘야 한다.**

인터페이스(예제 적용하기)

```java
public class InterfaceTest2 {
    public static void main(String[] args) {
        Lamp lamp = new Lamp();

        lamp.lamp.lightOn();
        lamp.lamp.lightOff();

        StreetLamp streetLamp = new StreetLamp();

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

        Led led = new Led();

        led.led.lightOn();
        led.led.lightOff();
    }
}
```

③메인 메서드에서는 각 클래스의 객체를 생성해 준다.
여기서는 총 3개의 클래스 객체를 만들었다.
그리고 그 밑에 메서드를 호출한다.

수고하셨습니다!