클래스,인스턴스,생성자 활용

클래스,인스턴스,생성자를 활용한 공급가액에 따른 부가가치세율과 총합 구하기

자바를 활용한 공급가액에 따른 부가가치세율과 총합을 구하는 알고리즘을 만들어보았다.
여기서 **valueOfSupply는 공급가액**이고 **vatRate는 부가세를 뜻합니다**.

code1

이 코드는 클래스를 추가로 만들어 메소드만으로 값을 받아오고 보내며 값을 출력하는 코드이다.

1. Accounting 클래스 안에 있는 메소드로 값을 AccountingApp1 클래스로 받아옵니다.

2.위에 클래스에서 변수를 불러와 Accounting.valueOfSupply 에 값을 입력하고 Accounting 클래스에서 getVAT 메소드를 불러와 공급가액에 따른 부가가치세율을 구해주고

3. Accounting 클래스에서 getTotal 메소드를 불러와 총합 계산을 해줍니다.

```
class Accounting{
    public static double valueOfSupply;
    public static double vatRate = 0.1;
    public static double getVAT(){
        return valueOfSupply * vatRate;
    }
    public static double getTotal(){
        return valueOfSupply + getVAT();

    }

}
```

```
public class AccountingApp1 {
    public static void main(String[] args) {

        Accounting.valueOfSupply=10000.0;
        System.out.println("value of supply : "+ Accounting.valueOfSupply);
        Accounting.valueOfSupply=20000.0;
        System.out.println("value of supply : "+ Accounting.valueOfSupply);

        Accounting.valueOfSupply=10000.0;
        System.out.println("VAT : "+ Accounting.getVAT());
        Accounting.valueOfSupply=20000.0;
        System.out.println("VAT : "+ Accounting.getVAT());

        Accounting.valueOfSupply=10000.0;
        System.out.println("Total : "+ Accounting.getTotal());

        Accounting.valueOfSupply=20000.0;
        System.out.println("Total: " + Accounting.getTotal());

    }

}
```

code2

두 번째 코드는
1.vatRate를 제외한 나머지 메소드와 변수의 static을 제거해준다
2.두 개의 인스턴스 a1 과 a2를 만들었습니다.
3.a1, a2의 valueOfSupply의 값은 a1=10000,a2=20000으로 초기화 시킵니다.
4.이러한 방법으로 인스턴스를 만들게되면 공급가액의 수가 늘어나더라도 valueOfSupply의 값을 바꿀 필요 없이 새로운 인스턴스를 생성하여 그 인스턴스의 valueOfSupply에 공급가액을 넣으면 되므로 코드의 길이는 줄어들게 됩니다.

```
class Accounting2{
    public static double valueOfSupply;
    public static double vatRate = 0.1;
    public static double getVAT(){
        return valueOfSupply * vatRate;
    }
    public static double getTotal(){
        return valueOfSupply + getVAT();

    }

}
```

```
public class AccountingApp2 {
    public static void main(String[] args) {
        Accounting a1 = new Accounting();
        a1.valueOfSupply=10000.0;

        Accounting a2 = new Accounting();
        a2.valueOfSupply=20000.0;


        System.out.println("value of supply : "+ a1.valueOfSupply);
        System.out.println("value of supply : "+ a2.valueOfSupply);

        System.out.println("VAT : "+ a1.getVAT());
        System.out.println("VAT : "+ a2.getVAT());

        System.out.println("Total : "+ a1.getTotal());
        System.out.println("Total: " + a2.getTotal());

    }

}
```

code3

생성자라는 명령어를 사용한다. 생성자란 객체가 생성되는 시점 부터 초기값을 생성해주는

명령어이다.

1.이 코드는 class에 생성자 메소드를 만들어 인스턴스를 생성할 때 인수를 받아 class의 valueOfSupply에 바로 초기화 시킵니다.

2.this.valueOfSupply는 Accounting3 클래스의 필드에 있는 valueOfSupply를 뜻하고 valueOfSupply는 생성자 메소드에 있는 매개변수를 의미합니다.

3.이 세 번째 코드가 가장 단정한 코드입니다.

```
class Accounting3{
    public  double valueOfSupply;
    public static double vatRate = 0.1;
```

```
    public Accounting3(double valueOfSuppl){
        this.valueOfSupply = valueOfSuppl;
    }
    public double getVAT(){
        return valueOfSupply * vatRate;
    }
    public  double getTotal(){
        return valueOfSupply + getVAT();

    }

}



public class AccountingApp3 {
    public static void main(String[] args) {
        Accounting3 a1 = new Accounting3(10000.0);

        Accounting3 a2 = new Accounting3(20000.0);



        System.out.println("value of supply : "+ a1.valueOfSupply);
        System.out.println("value of supply : "+ a2.valueOfSupply);

        System.out.println("VAT : "+ a1.getVAT());
        System.out.println("VAT : "+ a2.getVAT());

        System.out.println("Total : "+ a1.getTotal());
        System.out.println("Total: " + a2.getTotal());

    }

}
```

세 개의 코드 실행결과는 같습니다