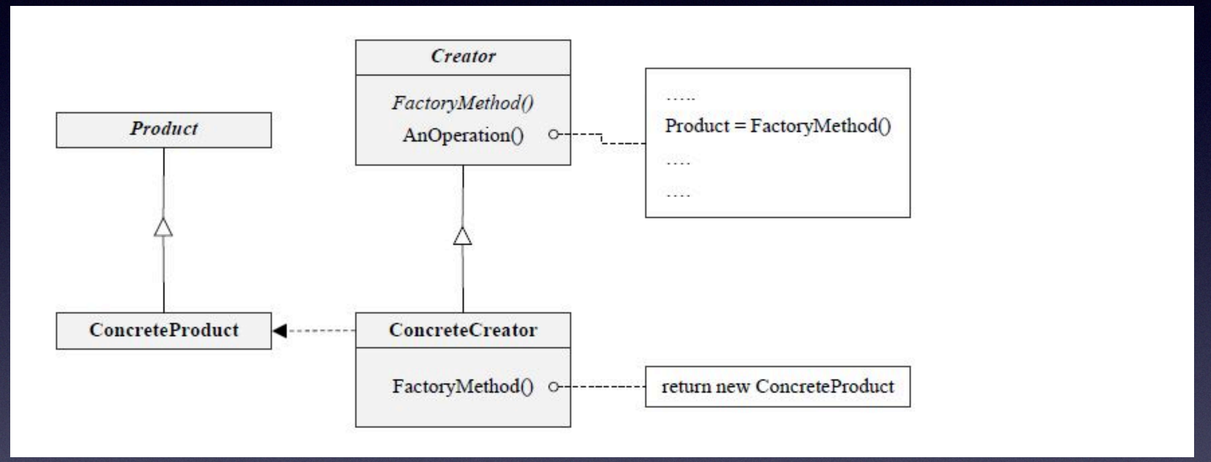
객체 지향 프로그래밍
with 자바

다음 코드를 상속과 Factory pattern 을 이용하여 리펙토링 하세요

```
public  class Car {

    public static final String SONATA = "Sonata";
    public static final String GRANDEUR = "Grandeur";
    public static final String GENESIS = "Genesis";

    String productName;

    public Car(String productName) {
        this.productName = productName;
    }

    public String toString() {
        return productName;
    }
}

public class CarTest {

    public static void main(String[] args) {

        CarTest test = new CarTest();
        Car car = test.produceCar("Sonata");

        System.out.println(car);
    }

    public Car produceCar(String name) {

        Car car = null;

        if( name.equalsIgnoreCase(Car.SONATA)) {
            car = new Car(Car.SONATA);
        }
        else if( name.equalsIgnoreCase(Car.GRANDEUR)) {
            car = new Car(Car.GRANDEUR);
        }
        else if( name.equalsIgnoreCase(Car.GENESIS)) {
            car = new Car(Car.GENESIS);
        }
        else {
            car = new Car("noname");
        }

        return car;
    }
}
```



아래의 코드가 항상 true가 되도록 Singleton Pattern 으로 구현하세요

```
public class SingletonTest {

  public static void main(String[] args) {

    Singleton instanceA = Singleton.getInstance();
    Singleton instanceB = Singleton.getInstance();

    System.out.println(instanceA == instanceB);
  }

}


#static
```

다음 객체를 구현하는데 Decorator pattern을 활용해보세요

- 커피를 만듭니다.

  커피 원두는 여러 종류가 있습니다. 이 원두를 활용하여 만드는 커피는 아메리카노, 라떼, 모카커피, 휘핑크림이 올라간 모카커피 등이 있습니다.

  원두의 종류가 케냐와 에티오피아가 있다고 할 때 다음과 같이 여러 종류의 커피를 만들 수 있는 클래스 구조를 디자인 해보세요

  커피에 첨가되는 장식자(Decorator)들은 다양하게 첨가되거나 바뀔 수 있습니다.

```
KenyaAmericano
KenyaAmericano Adding Milk
KenyaAmericano Adding Milk Adding Mocha Syrup
EtiopiaAmericano Adding Milk Adding Mocha Syrup Adding WhippedCream
```

java I/O Stream