1. static, final 예약어
제어자(modifier)는 클래스, 변수 또는 메소드의 선언부에 함께 사용되어 부가적인 의미를 부여한다.

1-1 static은 '클래스의', '공통적인'의 의미를 가지고 있다.
인스턴스 변수는 하나의 클래스로부터 생성되더라도 각기 다른 값을 유지하지만,
클래스 변수(static멤버변수)는 인스턴스에 관계없이 같은 값을 갖는다.

static 멤버 변수 :
- 클래스가 메모리에 로드될 때 생성.
- 모든 인스턴스에 공통적으로 사용되는 클래스 변수가 된다.
- 클래스변수는 인스턴스를 생성하지 않고도 사용 가능하다.

static 메소드 :
- 인스턴스를 생성하지 않고도 호출이 가능한 static 메소드가 된다.
- static 메소드 내에서는 인스턴스 멤버들을 직접 사용할 수 없다.

1-2 final
final은 '마지막의' 또는 '변경될 수 없는'의 의미를 가지고 있으며 거의 모든 대상에 사용될 수 있다.
변수에 사용되면 값을 변경할 수 없는 상수가 되며, 메소드에 사용하면 오버라이딩을 할 수 없게 되고
클래스에 사용되면 자신을 확장하는 자손클래스를 정의하지 못하게 된다.

final 클래스
- 변경될 수 없는 클래스. 확장될 수 없는 클래스가 된다.
- 그래서 final로 지정된 클래스는 다른 클래스의 조상이 될 수 없다.

final 메소드
- 변경될 수 없는 메소드, final로 지정된 메소드는 오버라이딩을 통해 재정의 될 수 없다.

final 멤버변수 또는 지역변수
- 변수 앞에 final이 붙으면 값을 변경할 수 없는 상수가 된다.

2. VO, DTO 데이터 전송 객체의 기능 및 관리법
VO와 DTO 둘 다 데이터를 저장하는 용도로 쓰인다.
2-1 VO(Value Object)
- DTO와 개념은 동일하나 read only 속성을 갖는다.
- 값의 변경이 없다.
- 데이터 자체로 의미있는 것을 담고 있는 객체
- 간단한 독립체를 의미하는 작은 객체
DB의 도메인정보를 클라이언트에게 전달할 때, 그 도메인 단위 정보를 VO로 구현하여 사용한다.
사용자 정보를 DB에서 가져오거나, 아니면 VIEW로부터 사용자 정보를 가져와 DB에 저장할 때 VO를 사용한다.
DB부터 VIEW까지 양방향적으로 사용 (DB-DAO-SERVICE-CONTROLLER-VIEW)

2-2 DTO(Data Transfer Object)
- 전송되는 데이터의 컨테이너
- VO와 동일하게 데이터를 저장하여 사용하도록 하는 부분에서 필요
- Layer간의 통신 용도로 오가는 객체
데이터를 오브젝트로 변환하는 객체.
뷰에서 컨트롤러 방향으로 일방향적으로 사용 CONTROLLER ← VIEW

EX >

[DTO]

- DTO a = new DTO(1);

- DTO b = new DTO(1);

일 때, a != b

[VO]

- VO a = VO(1);

- VO b = VO(1);

일 때, a == b

3. Constructor
생성자는 객체가 생성될 때(인스턴스화) 최초로 실행되는 메소드를 의미

3-1 생성자 특징
- 생성자 이름은 클래스명과 동일해야 한다
- 반환타입을 가져선 안된다.

- 생성자의 접근지정자로는 주로 public속성

3-2 생성자의 역할
- 멤버 변수를 초기화 하는 일
- 생성자를 구현하지 않았을 경우 컴파일러는 default생성자를 제공해줌
- 인자 생성자를 하나라도 구현했다면, 그 때는 컴파일러가 default생성자를 제공 해주지 않는다.
- 생성자를 다양하게 오버로딩 함으로써 다양한 초기값을 부여할 수 있다.

클래스 객체화(인스턴스화/heap영역에 메모리 생성)
- 클래스명 변수(인스턴스변수) = new 생성자();
- new의 의미:heap영역에 새롭게 메모리 생성

this()
-자기 자신의 생성자를 의미
-항상 생성자안에서만 사용 가능
-생성자 안에서도 맨 첫번째 문장에 와야한다.
-멤버변수만큼 인자를 가진 인자 생성자를 호출하기 위해서 주로 사용
(멤버변수보다 인자가 적은 생성자 안에서)

this 인스턴스 자신을 가리키는 참조변수, 인스턴스의 주소가 저장되어 있다.
모든 인스턴스 메소드에 지역변수로 숨겨진 채로 존재한다.

```
class Point{

    //[멤버 변수]
    private int x,y;
    //[기본 생성자]
    public Point() {
        this(1,1);//[o]
        System.out.println("기본 생성자");
        //this(1,1);[x]항상 첫번째 문장이어야한다
    }
    //[인자 생성자]
    public Point(int y) {
        this(1,y);
        System.out.println("인자 생성자:y");
    }
    public Point(int x,int y) {
        //멤버 변수 초기화
        this.x= x;
        this.y=y;
        System.out.println("인자 생성자:x,y");
    }

    //[멤버 메소드]
    void print() {
        //this(1,1);//[x]생성자에서만 호출 가능
        System.out.println(String.format("X좌표:%d,Y좌표:%d",x,y));
    }
}
public class Constructor03 {

    public static void main(String[] args) {
        //[기본 생성자로 객체 생성]
        Point point1 = new Point();
        point1.print();
        //[인자 생성자로 객체 생성]
        Point point2 = new Point(10);
        point2.print();
        Point point3 = new Point(10,20);
        point3.print();

    }////////////////main

}//////////////////class
```

4. Singleton관리 기법
싱글톤 디자인: 클래스를 설계하는 디자인 패턴의하나로 하나의 인스턴스 즉 하나의 메모리를 생성해 이를 공유하고자 할때 사용하는 패턴
즉 하나의 메모리를 서로 공유해서 사용하기 때문에 값 변경시 문제가 발생할 수 있는 경우는 사용하지 않는 것이 좋다.

- 고정된 메모리 영역을 얻으면서 한 번의 new로 인스턴스를 사용하기 때문에 메모리 낭비를 방지할 수 있다.
- 싱글톤으로 만들어진 클래스의 인스턴스는 전역 인스턴스이기 때문에 다른 클래스의 인스턴스들이 데이터를 공유하기 쉽다.
- 공통된 객체를 여러개 생성해서 사용해야하는 상황에서 많이 사용.

예시)
Normal.class

```
package singleton_pattern;

public class Normal {
    public Normal() {
        System.out.println("Normal Instance Created..");
    }
}
```

Singleton.class

```
package singleton_pattern;

public class Singleton {
    private static Singleton singleton = new Singleton();
    private Singleton() {
        System.out.println("Singleton Instance Created..");
    }
    public static Singleton getInstance() {
        return singleton;
    }
}
```

Main.class

```
public class Main {
    public static void main(String[] args) {
        Normal normal1 = new Normal();
        Normal normal2 = new Normal();
        Singleton singleton1 = Singleton.getInstance();
        Singleton singleton2 = Singleton.getInstance();
    }
}
```

콘솔 결과

```
Normal Instance Created..

Normal Instance Created..

Singleton Instance Created..
```

5. 상속의 기본, 사용법, 사용 이유
상속 : 기존의 클래스를 재사용하여 새로운 클래스를 작성하는 것.

-단일 상속만 지원
-IS A 관계 성립해야 한다. 자식 IS A 부모
-extends 키워드 사용
-private접근 지정자가 붙은 부모의 멤버는 상속은 받으나 접근 불가.

```
접근지정자 class Child extends Parent{

}
```

상속의 장점 :
- 적은 양의 코드로 새로운 클래스를 작성할 수 있다.
- 코드를 공통적으로 관리하여 코드의 추가 및 변경이 매우 용이하다.
- 코드의 재사용성을 높이고 프로그램의 생산성과 유지 보수에 기여