```
0) 싱글톤 관련 부분
   해당 클래스의 인스턴스가 하나만 만들어지고 어디에서든지 그 인스턴스에 접근할 수 있도록 하기 위한 패턴

/*
 * 싱글톤 디자인: 클래스를 설계하는 디자인 패턴의 하나로 하나의 인스턴스 즉 하나의 메모리를 생성해 이를 공유하고자 할때 사용하는 패턴
   하나의 메모리를 서로 공유해서 사용하므로 값 변경시 문제가 발생할 수 있는 경우는 사용하지 않는 것이 좋다.

    예]java.util패키지의 Calendar클래스

 *  방법]
 *  1.생성자의 접근 지정자를 private으로 한다
 *  2.정적 메소드로 해당 클래스의 객체를
 *    반환하도록 정의한다.
 */
//싱글톤으로 미 설계시]
class NoSingleTone{
    //[멤버변수]
    int noShareVar;
    //[멤버 메소드]
    void print() {
        System.out.println(String.format("noShareVar:%d", noShareVar));
    }
}//////////////NoSingleTone
//싱글톤으로 설계시]
class SingleTone{
    //[멤버변수]
    int shareVar;
    private static  SingleTone  single = new SingleTone();
    //[생성자]
    //1.접근지정자를 private 으로 지정
    private SingleTone() {}
    //[멤버 메소드]
    //2.정적 메소드로 해당 클래스의 객체를
    //  반환하도록 정의한다.
    public static SingleTone getInstance() {
        return single;
    }
    void print() {
        System.out.println(String.format("shareVar:%d", shareVar));
    }

}////////////////SingleTone


public class SingleToneDesign {
    public static void main(String[] args) {
        //싱글톤으로 미 설계시]
        //new할때마다 메모리가 생긴다.
        NoSingleTone no1 = new NoSingleTone();
        no1.noShareVar=100;
        no1.print();

        NoSingleTone no2 = new NoSingleTone();
        no2.noShareVar=200;
        no2.print();

        System.out.println(String.format(
                            "no1:%s,no2:%s,no1의 noShareVar:%d,no2의 noShareVar:%d",
                            no1,no2,no1.noShareVar,no2.noShareVar));

        //싱글톤으로 설계시]
        //SingleTone st1 = new SingleTone();//[x]인스턴스화 불가
        //반드시 정적메소드를 통해서 인스턴스화 한다.
        SingleTone st1 = SingleTone.getInstance();
        st1.shareVar=100;
        st1.print();
        SingleTone st2 = SingleTone.getInstance();
        st2.shareVar=200;
        st2.print();
        System.out.println(String.format(
                    "st1:%s,st2:%s,st1의 shareVar:%d,st2의 shareVar:%d",
```

```
                    st1,st2,st1.shareVar,st2.shareVar));

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

1) interface
   클래스의 설계도. 클래스는 인터페이스를 상속받을 수 있지만 인터페이스는 클래스를 상속받을 수 없다.
   자바는 단일 상속이 원칙이나 인터페이스를 이용해서 다중 상속을 구현할 수 있다.
   멤버로는 추상메소드와 상수(final 변수)로만 구성된다.
   접근지정자는 public과 default 만 가질 수 있다.
   인터페이스에 있는 추상메소드는 public과 abstract 키워드를 생략한다.

2) 다형성(polymorphism)
   '형태가 다양한', '여러 가지 형태' 의 의미.
   한 타입(클래스)의 참조 변수로 여러 타입의 객체를 참조할 수 있도록 함.
   부모 클래스 타입의 참조 변수로 자식 클래스의 인스턴스를 참조할 수 있다.
   마찬가지로 해당 인터페이스 타입의 참조변수로 클래스의 인스턴스를 참조할 수 있다.
   예)

```
class AA { //부모 클래스
 int a;
}

class BB extends AA{ //부모 클래스를 상속받은 자식 클래스
int b;
}

public class Polymor{

 public static void main(String[] args){

   AA aa = new AA();
   BB bb = new BB();

   AA aa = new BB();

   BB bb = new AA(); //컴파일 에러
   //실제 인스턴스인 AA의 멤버 개수보다 참조변수 bb가 사용할 수 있는 멤버 개수가 더 많기 때문이다.

 }

}
```

3) mutable(가변객체) , immutable 객체(불변 객체)
   mutable객체는 객체 내의 특정 요소를 변경할 수 있는 객체
   List, ArrayList, HashMap등의 컬렉션

   immutable객체는 객체 내의 특정 요소 값을 변경할 수 없는 객체
   String, Integer, Double, Long

   보통의 immutable 객체는 객체를 복사하게 되면 객체 자체가 아닌 참조값을 복사한다.
   똑같은 값을 가진 객체가 메모리에 하나 더 생성이 되는게 아닌 그 객체를 가리키는 포인터가 하나 더 생성이 된다고 보면 된다.
   이 immutable 객체는 멀티 스레드 관점에서 보았을 때 꽤나 유용한데
   데이터가 불변객체에 저장이 되어있다면 여러 스레드에 의해서 이 객체 안의 데이터에 접근을 하게 되었을 때
   데이터가 변경되지 않는다.
   반면 mutable 객체는 위에서 설명한 참조 복사 기법으로 다루게 되면 참조값을 전달받아서
   그 참조를 통해 어디서든지 값을 변경할 수 있다.

4) IO Stream
   I/O란 Input과 Output의 약자로 입력과 출력을 의미
   입출력은 컴퓨터 내부 또는 외부의 장치와 프로그램 간의 데이터를 주고 받는 것을 말한다.
   자바에서 입출력을 수행하려면(어느 한쪽에서 다른 쪽으로 데이터를 전달하려면)
   두 대상을 연결하고 데이터를 전송할 수 있는 것이 필요한데 이것을 스트림(Stream)이라고 한다.

   스트림은 데이터를 운반하는데 사용되는 연결통로이다.
   스트림은 하천처럼 한쪽 방향으로만 흐르는 것처럼 단방향 통신만 가능하므로
   입력과 출력을 동시에 하려면 입력스트림과 출력스트림, 2개의 스트림이 필요하다.

   바이트 기반 스트림과 문자기반 스트림이 있다.
   바이트 기반 스트림
   (InputStream, OutputStream)
   (ByteArrayInputStream, ByteArrayOutputStream)
   (FileInputStream, FileOutputStream)

바이트 기반 보조 스트림
(FilterInputStream, FilterOutputStream)
(BufferedInputStream, BufferedOutputStream)
(DataInputStream, DataOutputStream)
(SequenceInputStream)
(PrintStream)

문자기반 스트림
(Reader, Writer)
(FileReader, FileWriter)
(PipedReader, PipedWriter)
(StringReader, StringWriter)

문자기반 보조스트림
(BufferedReader, BuffredWriter)
(InputStreamReader, OutputStreamWriter)

5) Java Memory 와 Java version에 따른 String pool 운영 차이점
Java6 이하에서 PermGen 영역에 String Pool을 저장해 놓았었는데 PermGen 영역은 Runtime에
확장될 수 없는 고정된 capacity를 가지고 있기 때문에 intern(객체 대신 참조를 복사하는 기법)되는 String 값이
너무 많아지면
OutOfMemoryException을 맞게 될 확률이 높았다.

그래서 Java7부터 String Pool을 PermGen영역에 저장하지 않고 heap 영역에 저장한다.
문자열 객체를 만들면 매번 메모리에 새로운 객체가 만들어진다. 이를 인턴하여 새로운 스트링 풀 String- pool 에 저장하고
그 뒤 같은 문자열이라면 풀에서 참조를 반환한다. 즉 인턴이 사용되고 있다면
2개의 객체가 같다고 판단을 하는경우는 참조가 같다고 하는 경우이다.
이것이 왜 문자열 비교를 할때 str1 ==str2 로 하면 안되는 건지 알려준다.
str1과 str2의 내부적인 문자열이 같은지를 비교하는 것이 아닌 참조값을 비교하는 것이다.
PermGen : Permanent Generation
클래스의 정의와 연관된 메타데이터를 위해 사용되는 메모리 공간