

3주차(1, 2, 3, 4)

IO(Input, Output)

IO는 입출력 스트림을 의미합니다.

스트림이란, 데이터를 입출력하기 위한 방법.

프로그램에서 파일을 읽어온다든지, 콘솔에서 키보드값을 얻어오는 등의 작업을 할 수 있습니다.

자바 가상머신에서 콘솔로 값을 보낼땐 Output, 반대로 콘솔의 값을 JVM에서 읽을땐 Input

File클래스

FileEx1 클래스 정의

test.txt 문서를 만들고 안녕하세요 라고 적은 뒤에 d:/에 가져다 놓은 뒤에 코드 작성

```
public class FileEx1 {
```

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

```
        //
```

파일 객체를 생성할 경로

```
        String path = "d:/test.txt";
```

```
        //
```

준비된 경로로 File객체 생성

```
        File
```

```
        f1 = new File(path);
```

```
        if(f1.isFile()){// 생성된 객체가 파일일 경우
```

```
            System.out.println("파일의 크기:"+f1.length()+"byte");
```

```
        }
```

```
    }
```

```
}
```

-----예제1

FileEx2 클래스 정의

```
public class FileEx2 {
```

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

```
        //
```

파일객체로 쓰인 경로

```
        String path = "d:/my_study/java_study";
```

```
        //강
```

의에 사용중인 폴더 경로 지정

```
        File
```

```
        f1 = new File(path);
```

```
        if(!f1.isFile()){ //파일이 아닐경우, f1.isDirectory()
```

```
        //
```

디렉토리 안에있는 하위 요소들의 이름을 모두 가져온다.

```
        String[] names = f1.list(); //f1객체의 경로에 있는 파일 이름을
```

names배열에 저장

```
System.out.println("java_study의 하위 목록들.....");
```

```
for(int i = 0; i < names.length ; i++)
```

```
System.out.println(names[i]);
```

```
}
```

```
}
```

```
}
```

-----예제2

FileEx3 클래스 정의

```
public class FileEx3 {
```

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

```
String path = "d:/my_study/java_study";
```

File

```
f1 = new File(path);
```

/ /

폴더의 하위 요소들을 File[]로 받기

```
if(f1.isDirectory()){
```

File[]

```
ar = f1.listFiles();
```

위와 같이 하위요소들을 File형의 배열로 받으면

//

반복하면서 파일의 존재여부, 디렉토리 구별 등의 기능을

//

함께 응용할 수 있는 장점이 있다.

//

```
for(int i=0; i<ar.length; i++){
```

```
if(ar[i].isDirectory())
```

```
System.out.println(ar[i].getName());
```

```
}
```

```
}
```

```
}
```

```
}
```

-----예제3

FileEx4 클래스 정의

```
public class FileEx4 {
```

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

```
String path = "d:/test/abc";//만들어질 폴더
```

File

```
f1 = new File(path);
```

```
if(!f1.exists()){ //존재여부 확인 if(f1.exists() == false)
```

/ /

```
존재하지 않을 경우에만 물리적인 폴더 생성!!!
```

```
System.out.println("폴더를 생성합니다.");
```

```
f1.mkdirs();
```

}else{

```
System.out.println("폴더가 이미 있습니다.");
```

}

}

}

-----예제4

FileInputStream

```
public class FileInput {
```

```
public static void main(String[] args)
```

```
throws FileNotFoundException, IOException{
```

```
String path = "d:/test.txt";//위 예제에서 만든 test.txt문서
```

```
f1 = new File(path);
```

```
if(f1.exists()){ //파일이 실제 존재할 때만 수행!
```

```
파일과 연결된 입력 스트림 생성
```

```
FileInputStream fis = new FileInputStream(f1);
```

```
스트림은 더 이상 읽을 것이 없다면 -1을 반환하게 되어 있다.
```

```
즉 파일의 모든 내용을 읽어오기 위해 반복문을 수행하고
```

```
그 반복은 파일의 끝(EOF)인 -1을 만날 때까지 반복하면 된다.
```

```
code = 0;
```

```
File
```

```
/ /
```

```
/ /
```

```
/ /
```

```
/ /
```

```
i n t
```

```
while((code = fis.read()) != -1){
```

```
// 그
```

```
냥 code를 출력하면 97과 같은 실제 코드값이
```

```
// 출
```

```
력되므로 문자로 형변환하여 출력한다.
```

```
// 한
```

```
글은 깨진다.
```

```
// 한
```

```
글은 유니코드 형식인데,
```

```
// 이
```

```
것을 int 자료형에 담아 char로 형변환하여 출력하면
```

```
// 아
```

```
스키 코드값으로 변경되어 출력되기 때문.
```

```
System.out.print((char)code);
```

```

    }

    //
스트림들은 열고 사용한 후에는 반드시

    //
닫아줘야 한다.

    if(fis
!= null)

    fis.close();

}

System.out.println("프로그램 끝!");

}

```

-----예제

FileInput2클래스 정의

```

public class FileInput {

    public static void main(String[] args)

    throws FileNotFoundException, IOException{

    String path = "d:/test.txt";

```



```
f1 = new File(path);

b_Read[] = new byte[100]; // 넉넉하게 100.

// 값
을 줄이면 당연히 읽어오는 내용의 일부가 잘린다.

if(f1.exists()){ //파일이 실제 존재할 때만 수행!

//

파일과 연결된 입력 스트림 생성

FileInputStream fis = new FileInputStream(f1);

//fis
경로의 txt파일 내용을 b_Read[]배열에 저장

// 값
을 저장할 byte배열, 시작위치, 끝위치

fis.read(b_Read, 0, b_Read.length);

// 위
```

의 코드로 b_Read배열에 test.txt의 내용이 저장되었다.

```
System.out.println(new String(b_Read).trim());
```

스트림들은 열고 사용한 후에는 반드시 닫아줘야 한다.

```
!= null)
```

```
fis.close();
```

```
System.out.println("프로그램 끝!");
```

```
}
```

-----예제2

자바 강의 3주차(1) 문제(회문수) 풀기!!!!

이제 본격적으로 IO를 시작할게요

IO는 크게 byte기반의 Stream과 char기반의 스트림으로 나뉩니다.

바이트Stream

(byte기반은 인,아웃풋 스트림을... char기반은 리더,라이터를 씀)

java.io패키지의 InputStream과 OutputStream이 byte구조의 스트림이다.

InputSteam에는 BufferedInputStream, ByteArrayInputStream, DataInputStream, FilterInputStream, read(), OutputStream, PushbackInputStream 등이 있다.

이중에서 많이 쓰이는 것 위주로 알아볼까요?

※ Api의 SeeAlso 참조

Byte의 InputStream(입력)먼저 알아보자

InputStream은 키보드의 입력값을 받아 화면에 출력하는 OS의 표준 입력장치와 이미 연결되어 있다는 것만 알고 넘어가자.

FileInputStream을 알아봅시다.

위에서 File클래스를 배우면서 FileInputStream했었죠?

거기서 확장된 예제라고 생각하시면 될 듯. 부담없이 확인하세요

FileInputEx클래스 정의

```
public class FileInputEx {
```

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

```
        FileInputStream fis = null;
```

```
        byte[] read[] = new byte[100];
```

```
        byte[] console[] = new byte[100];
```

```
        System.out.print("파일 경로 : ");
```

```
        System.in.read(console); //읽어올 파일의 경로를 byte배열로 받
```

byte

byte

try {

다. `//`는

```
String file = new String(console).trim();//위에서 입력한 경로를
```

```
//문자열로 변환
```

```
//Scanner scan = new Scanner(System.in);
```

```
//String file = scan.next();
```

```
// 위  
의 System.in.read(console)은 Scanner로 값을 입력받는것과
```

```
// 같  
은 결과지만
```

```
//Stream을 배우고 있기 때문에 System.in.read()를 사용했다.
```

```
fis =  
new FileInputStream(file);//FileInputStream을 통해 file경
```

```
        //로  
        를 받는 객체를 준비
```

```
        fis.read(read, 0, read.length);//read[]배열의 0번째부터 100번째
```

```
        //사  
        이에, 읽어온 txt파일의 내용을 복사
```

```
        System.out.println(new String(read).trim());//read[]를 문자열로
```

```
        //변  
        경하여 출력
```

```
    }  
    catch (Exception e) {
```

```
        e.printStackTrace();
```

```
    }

    finally { //finally는 try catch구문을 마치고 무조건 실행되는 예약어

        try {

            // 스

            트림을 닫아준다.

            if(fis

                != null)

                fis.close();

            }

            catch (IOException e) {

                //

                TODO Auto-generated catch block

                e.printStackTrace();

            }

        }

    }

}
```

```

}
}
}

```

실행 결과

전에 만들었던 test.txt파일을 읽어오자요

파일 경로 : d:/test.txt

안녕하세요abcd

-----예제

BufferedInputStream을 알아보자

BufferedInputEx클래스 정의

Buffered스트림을 사용하는 이유는 입출력의 효율성을 향상시키기 위해서 이다.

화장실을 예로들자면, 공용화장실 같은 경우는 남녀가 모두 이용하는 공간이기 때문에 자칫 처리가 늦어질 수 있지만, 남자화장실 혹은 여자화장실로 각각의 역할을 구분해두면 쉽게 접근할 수 있는것과 비슷한 이치랄까??

위에 작성한 예제와 거의 흡사한 예제이지만, Buffered스트림을 연결해서 어떻게 구현되는지에 대한 예제를 확인해 봅시다.

```
public class BufferedInputEx {
```

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

```
//여
```

기 주석은 맨 마지막에 결과 확인 후, 작성해주자

//FileInputStream과 BufferedInputStream을 연결함으로써

```
//파
```

일을 읽을 때 버퍼링 작업을 수행하도록 한다.

```
//버
```

퍼링 작업이란, 출력할 바이트를 버퍼라고 하는 메모리 공간에 바이트

```
//배
```

열로 저장하여 한번에 출력하는 것.

퍼라는 공간은 파일을 쓰고 받기위해 마련된 기억장치의 한 부분인데, // 버퍼

출력할 자료를 버퍼에 모아두면, 입출력시에 버퍼라는 전용 공간을 // 입
// 활
용하기 때문에 출력속도 향상에 도움이 된다.

무 어려우니, Buffer스트림은 그냥 입출력 향상에 도움이 된다. 라는 정 // 너
// 도
로만 기억하고 있으면 된다.

```
FileInputStream fis = null;
```

```
BufferedInputStream bis = null;
```

```
byte // byte  
_byte[] = new byte[100];
```

```
byte // byte  
result[] = new byte[100];
```

```
try {
```

```
System.out.print("경로 입력 : ");
```

```
System.in.read(_byte);
```

```
String path = new String(_byte).trim();
```


//d:/test.txt를 불러올 예정

new FileInputStream(path);

fis =

new BufferedInputStream(fis);

bis =

bis.read(result, 0, result.length);

System.out.print(new String(result).trim());

catch (Exception e) {

}

TODO: handle exception

/ /

}finally{

try {

<pre> != null) fis.close(); != null) bis.close(); catch (Exception e2) { TODO: handle exception } } </pre>	<pre> if(fis if(bis } / / } } </pre>
---	---

-----예제

byte의 인풋을 봤으니, 이제 OutputStream을 알아보시다.

OutputEx클래스 정의

```
public class OutputEx {
```

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

```
        //PrintStream은 OutputStream의 대표적인 자식 클래스 중 하나.
```

```
        // 화
```

```
        면에 데이터를 출력하도록 하는 클래스이다.
```

```
        PrintStream ps = null;
```

```
        try {
```

```
            ps =
```

```
            System.out;
```

```
        } catch (i n t
```

```
            first = 'A';
```

```
        } catch (i n t
```

```
            second = 'B';
```

```
        ps.write(first);
```

```
ps.write(second);
```

```
//System.out.println()역시
```

```
//OutputStream의 자식인 PrintStream을 사용하고 있기때문에
```

```
를 이용해 화면에 결과를 출력할 수 있었던 것.
```

```
//이
```

```
catch (Exception e) {
```

```
}
```

```
TODO: handle exception
```

```
/ /
```

```
finally{
```

```
}
```

```
try {
```

-----예제 1

FileOutputStream에 대해 알아보시다.

이름 그대로 코드상의 내용을 파일에 쓸 수 있도록 하는 기능을 제공하는 클래스

FileOutputEx클래스 정의

```
public class FileOutputEx {
```

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

```
        FileOutputStream fos = null;
```

```
        try {
```

```
            fos =
```

```
            new FileOutputStream("d:/fileOutput.txt");
```

```
            String msg = "file OutputStream 예제";
```

```
            //msg를 byte배열로 변환한 후 fos객체를 통해 txt파일을 생성
```

```
            fos.write(msg.getBytes());
```

```
        }
```

```
    catch (Exception e) {
```

```
        e.printStackTrace();
```

```

}

finally {

}

try {

}

if(fos

!= null)

fos.close();

}

catch (Exception e2) {

//

TODO: handle exception

}

}

}

```

-----예제2

DataOutputStream과 DataInputStream을 알아보시다.

DataOutputStream은 출력 스트림에 기본자료형을 기록하기 편리하도록 메서드를 제공한다.
DataInputStream은 받아온 정보에서 기본자료형을 읽어오는 메서드를 제공한다.

DataOutputEx클래스 정의

```
public class DataOutputEx {
```

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

```
        //DataOutput스트림과 DataInput스트림을 동시에 사용하는 예제
```

```
        FileInputStream fis = null;
```

```
        FileOutputStream fos = null;
```

```
        DataInputStream dis = null;
```

```
        DataOutputStream dos = null;
```

```
        try {
```

```
            fos =
```

```
            new FileOutputStream("d:/dataOutput.txt");
```

```
            d o s
```

```
            = new DataOutputStream(fos);
```



```
dos.writeBoolean(false);
```

```
dos.writeInt(1000);
```

```
dos.writeChar('A');
```

```
dos.writeDouble(10.5);
```

```
                                                    //여  
기까지 실행하고, 해당 경로에 파일이 생성되었는지 확인
```

```
//DataOutputStream의 메서드는
```

```
//Boolean, Integer, Character, Double등 기본자료형을 포함하
```

```
//는  
부모객체를 반환하므로, 결과 확인시에 값이 깨져서 나온다는
```

```
//것  
을 알고가자.
```

```
//txt
```

파일 결과 확인 후 값을 가져오는 부분 추가

```
new FileInputStream("d:/dataOutput.txt");  
  
new DataInputStream(fis);  
  
System.out.println(dis.readBoolean());  
  
System.out.println(dis.readInt());  
  
System.out.println(dis.readChar());  
  
System.out.println(dis.readDouble());  
  
catch (Exception e) {  
  
    TODO: handle exception  
  
    finally {  
  
        try {
```

```
!= null)
```

```
if(fis
```

```
fis.close();
```

```
!= null)
```

```
if(fos
```

```
fos.close();
```

```
!= null)
```

```
if(dis
```

```
dis.close();
```

```
if(dos != null)
```

```
dos.close();
```

```

}

catch (Exception e2) {

}

//

TODO: handle exception

}

}

}

}

```

-----예제3

BufferedOutputStream에 대해 알아보시다.

앞서 진행한 FileOutputStream과 유사하지만, FileOutputStream을 BufferedOutputStream에 탑재하여 쓰기 작업에 효율성을 높여보자.

BufferedOutputEx클래스 정의

```
public class BufferedOutputEx {
```

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

```
FileOutputStream fos = null;
```

```
BufferedOutputStream bos = null;
```

```
try {
```

```
fos =
```

```
new FileOutputStream("d:/bufferedOutput.txt");
```

b o s

```
= new BufferedOutputStream(fos);
```

```
//BufferedOutputStream에 FileOutputStream을 연결
```

```
String str = "BufferedOutputStream 예제";
```

```
bos.write(str.getBytes());
```

```
//BufferedOutputStream은 flush()메서드를 호출하지 않으면
```

// 파

일에 데이터를 쓰는것(write)가 불가능하다.

```
//write()메서드는 값을 기억하고 있고 이를 물리적으로 파일에
```

// 쓰

기 위해서 flush()함수를 반드시 사용해야 합니다.

```
bos.flush();
```

```
}
```

```
catch (Exception e) {
```

```
/ /
```

```
TODO: handle exception
```

```
}finally{
```

```
try {
```

```
if(fos
```

```
!= null)
```

```
fos.close();
```

```
if(bos != null)
```

```
bos.close();
```

```

}

catch (Exception e2) {

}

//

TODO: handle exception

}

}

}

}

```

-----예제4

charStream

byte스트림이 InputStream, OutputStream과 그 하위클래스를 사용했다면,
char스트림은 Reader와 Writer의 하위클래스를 사용합니다.

byte스트림은 1바이트 단위로 입출력하기 때문에 한글의 입출력에 크게 적합한 형태를 보이지 않았지만 char스트림은 일반적으로 2바이트를 지원하기 때문에 한글(아랍어 등도)로 구성된 파일을 입출력하기에 적합하다.

charStream의 Reader!!

첫 번째로! FileReader를 살펴봅시다.

FileReaderEx1클래스 정의

```

public class FileReaderEx1 {

    public static void main(String[] args) {

```

```

        FileReader fr = null;

```

```

        try {

```

```

new FileReader("d:/test.txt");

readChar;

while((readChar = fr.read()) != -1){

System.out.print((char)readChar);

// 결
과를 보면 byte기반의 스트림과는 다르게 2바이트 형

// 식
으로 한글출력이 올바르게 되는것을 볼 수 있다.

}

}

catch (Exception e) {

e.printStackTrace();

}

```



```
finally {
```

```
try {
```

```
if(fr
```

```
!= null)
```

```
fr.close();
```

```
}
```

```
catch (Exception e2) {
```

```
/ /
```

```
TODO: handle exception
```

```
}
```

```
}
```

```
}
```

```
}
```

-----예제 1

문제 :

특정 경로에 test.txt파일을 만들고 파일의 내용으로 대소문자를 비롯하여 숫자나 문자를 섞어 아무 내용이나 작성한 후, Scanner를 통해 test.txt파일의 경로를 입력받아
FileReader를 통해 test.txt파일에 알파벳 대문자와 소문자가 각각 몇 개씩 있는지를 판별하는 로직을 구현해보자.

실행결과:

경로 입력 : c:/test.txt

대문자 : 5

소문자 : 5

풀이 :

InputEx클래스 생성

```
public class InputEx {
```

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

```
        System.out.print("경로 입력 : ");
```

```
        Scanner scan = new Scanner(System.in);
```

```
        String str = scan.next().trim();
```

```
        FileReader fr = null;
```

```
        try {
```

```
            fr =
```

```
new FileReader(str);
```

```
console = 0;
```

```
i n t
```

```
upper = 0;//대문자의 수를 담을 변수
```

```
i n t
```

```
lower = 0;//소문자의 수를 담을 변수
```

```
i n t
```

```
while((console = fr.read()) != -1){
```

```
if(console >= 'A' && console <= 'Z'){
```

```
upper++;
```

```
}
```

```
if(console >= 'a' && console <= 'z'){
```

```
    lower++;
```

```
}
```

```
}
```

```
System.out.println("대문자 : " + upper);
```

```
System.out.println("소문자 : " + lower);
```

```
}
```

```
catch (Exception e) {
```


BufferedReaderEx클래스 정의

```
public class BufferedReaderEx {
```

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

```
        // 일
```

단 지금까지 써왔던 test.txt의 내용을 여러줄로 수정 ㅋ

```
        FileReader fr = null;
```

```
        BufferedReader br = null;
```

```
        try {
```

```
            fr =
```

```
            new FileReader("d:/test.txt");
```

//BufferedReader로 fr이 읽어온 내용을 한줄단위로 처리한다.

```
            br =
```

```
            new BufferedReader(fr);
```

```
            String msg;
```

```
while((msg = br.readLine()) != null){
```

```
System.out.println(msg);
```

```
}
```

```
}
```

```
catch (Exception e) {
```

```
e.printStackTrace();
```

```
}
```

```
finally {
```

```
try {
```

```
if(fr
```

```
!= null)
```

```
fr.close();
```

```
!= null)
    if(br
        br.close();
    }
catch (Exception e2) {
    // 
    TODO: handle exception
}
}
```



```
f = new File(
```

```
"d:/java/Test5/src/FileReaderEx1.java");
```

```
// 이
```

전예제 java파일의 경로

```
FileInputStream fis = new FileInputStream(f);
```

```
//
```

위는 byte기반이므로 한줄 단위처리가 안된다.

```
//
```

BufferedReader가 필요하다.

```
BufferedReader br = new BufferedReader(
```

```
new
```

```
InputStreamReader(fis));
```

```
String str;
```

```
while((str = br.readLine()) != null){
```

```
System.out.println(str);
```

```
}
```

```
!= null)
```

```
if(fis
```

```
fis.close();
```

```
if(br
```

```
!= null)
```

```
br.close();
```

```
}
```

```
}
```

-----예제3

charStream의 Writer

FileWriter 빠밤!

FileWriterEx클래스 정의

```
public class FileWriterEx {
```

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

```
FileWriter fw = null;
```

```
try {
```

```
new FileWriter("d:/fileWriterEx.txt");

String str = "나는 fileWriter예제다";

fw.write(str);

//FileOutputStream의 경우엔

//fos.write(str.getBytes());를 통해

//문자열을 바이트 단위로 쪼개야 했지만

//char기반 스트림은 문자열을 바이트 단위로 쪼개지 않고

//파일에 쓸수 있다.

catch (Exception e) {

e.printStackTrace();

finally {
```

```

try {

    if(fw

!= null)

        fw.close();

    }

    catch (Exception e2) {

        //

        TODO: handle exception

    }

}

```

-----예제 1

BufferedWriter복시다

BufferedWriterEx클래스 정의

```
public class BufferedWriterEx {
```

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

```
    FileWriter fw = null;
```

```
    BufferedWriter bw = null;
```

```
        try {
```

```
            fw =
```

```
            new FileWriter("d:/BufWriter.txt");
```

```
            bw =
```

```
            new BufferedWriter(fw);
```

```
            bw.write("BufferedWriter테스트");
```

```
            bw.newLine();//한줄 아래로(\n)내리는 함수 ↓↓
```

```
            //bw.write("\r\n"); 이걸 내부 함수를 이용해 구현한것임
```

```
bw.write("갑돌이와 갑순이는 한마을에 살았더래요" +
```

```
System.getProperty("line.separator"));
```

```
//System.getProperty("line.separator"))를 통해
```

```
// 프
```

```
로그램이 라인의 끝이라는 것을 알 수 있도록 해 준다.
```

```
// 없
```

```
어도 무방하지만, 명시하면 파일을 쓰는데 속도면에서 어느정
```

```
// 도
```

```
효율성이 높아진다.
```

```
bw.flush();//Buffer를 이용한 write에는 flush()를 통해
```

```
// 실
```

```
제로 파일에 내용을 작성해야 한다.
```

//bw.close();를 여기서 사용하면 flush()를 사용하지 않아도

//close()메서드 내부 기능에 의해 파일쓰기가 가능하지만,

//finally가 아닌 try부분에 close()를 사용하면

//close()윗라인에서 오류가 발생했을 경우

// 스

트림을 달지 못하고 종료되기 때문에 그냥 flush()를 쓰자.

}

catch (Exception e) {

e.printStackTrace();

}

finally {

try {

if(fw

```
!= null)
```

```
fw.close();
```

```
!= null)
```

```
if(bw
```

```
bw.close();
```

```
catch (Exception e2) {
```

```
}
```

```
TODO: handle exception
```

```
/ /
```

```
}
```

```
}
```

```
}
```

```
}
```

-----예제2

스트림의 마지막!!

ObjectStream

예전에 배운 `DataInput`, `Output` 스트림은 기본자료형의 입출력(`writeBoolean(false)`, `dos.readInt()` 등)을 관리했었는데, `Object` 스트림을 사용하면 사용자가 구현한 커스텀 클래스의 내용을 저장할 수 있다. 기본자료형 뿐 아니라 모든 객체의 입출력이 가능해짐
사용자가 정의한 객체를 통째로 저장할 때 자주 사용되는 스트림이다.
`Object`형으로 모든 객체를 통째로 받아 저장할 수 있기 때문에 특정 객체의 내용을 통째로 받아서 원하는 멤버를 추출해 사용할 수 있다는 장점이 있다.

//가위바위보 게임을 하고 그 결과를 파일로 저장 & 로드 해보자

문제) 일단 승, 무, 패, 유저아이디를 저장하는 클래스를 한 개 만들고
메인클래스에서 이를 이용하는 가위바위보 게임을 만들도록 하자.

결과 :

아이디를 입력하세요: 1111

가위(s) | 바위(r) | 보(p) : r

당신이 이겼습니다.

1승 0무 0패

다시 하시겠습니까? y | n : n

게임을 종료합니다.

사람들이 문제를 풀면 이것을 가지고 스트림을 이용해 세이브와 로드를 알려주는걸로.

RspInfo클래스 정의

```
public class RspInfo implements Serializable{
```

```
    private int win, lose, draw;
```

```
    private String name; //유저의 ID를 저장할 변수
```

```
    public void setWin(int win) {this.win = win;}
```

```
    public void setLose(int lose) {this.lose = lose;}
```

```
    public void setDraw(int draw) {this.draw = draw;}
```

```
    public void setName(String name) {this.name = name;}
```

```

public int getWin() {return win;}

public int getLose() {return lose;}

public int getDraw() {return draw;}

public String getName() {return name;}
}

```

이 클래스는 유저의 승,무,패에 관한 기록과 아이디를 관리한다.

implements Serializable을 한 이유는 스트림으로 객체를 통째로 관리하기 위해서는 객체의 직렬화가 필수적이기 때문이다.

메모리 공간에 각각 다른 영역에 자리잡고 있던 멤버들을 한번에 쓰고 받을 수 있도록 새로운 메모리 영역에 일렬로 만들어 복사한다.

시리얼라이저블은 “내가 객체의 정보를 일렬로 만들어서 가지고 있습니다”라고 명시하고 있는 것이다.

즉, 실제로 객체를 생성하는 것이 아니라 객체의 정보만 가진채로 내용물을 복사해 가지고 있다가, Input스트림으로 값을 읽어올 때 해당 객체에 저장해뒀던 정보를 넣어준다는 것.

나중에 스트림으로 쓰고 받는 작업을 할것이기 때문에 제작단계에서 미리 Serializable을 인터페이스를 구현하였다.

RspMain클래스 정의

```

public class RspMain {

    public static void main(String[] args) {

```

// 승

점과 아이디를 저장할 RspInfo객체 생성

```

RspInfo rinfo = new RspInfo();

```

```

String id;

```

i n t

```

win = 0, lose = 0, draw = 0;

```

```
System.out.print("아이디를 입력하세요 : ");
```

```
Scanner scan = new Scanner(System.in);
```

```
scan.nextLine();
```

```
rinfo.setName(id.trim()); //입력받은 아이디를 rinfo객체에 저장
```

```
일 읽기
```

```
장 마지막에 ScoreLoader클래스 정의 후에 추가
```

```
ScoreLoader loader = new ScoreLoader(rinfo);
```

```
= loader.getInfo().getWin();
```

```
= loader.getInfo().getLose();
```

```
= loader.getInfo().getDraw();
```

```
id =
```

```
// 파
```

```
// 가
```

```
try {
```

```
w i n
```

```
l o s e
```

```
d r a w
```

```
rinfo.setWin(win);
```

```
rinfo.setDraw(draw);
```

```
rinfo.setLose(lose);
```

```
catch (Exception e) {  
    }  
}
```

```
//  
//  
// TODO Auto-generated catch block  
//  
//
```

```
e.printStackTrace();  
}  
}
```

```
System.out.println( win + "승, " + lose + "패, " + draw + "무");
```

```
while(true){
```

```
    int  
    random = new Random().nextInt(3);
```

```
    //0 :  
    가위, 1 : 바위, 2 : 보
```

```
System.out.print("가위(s) | 바위(r) | 보(p) ? : ");
```

```
String user = scan.next();
```

```
usercnt = 0;
```

```
    i n t
```

```
if(user.equalsIgnoreCase("s")){
```

```
    usercnt = 0;
```

```
        }else
```

```
if(user.equalsIgnoreCase("r")){
```

```
    usercnt = 1;
```

```
        }else
```

```
if(user.equalsIgnoreCase("p")){
```

```
    usercnt = 2;
```

```
}
```

```
// 경
```

```
우의 수 비교
```

```
if(usercnt - random == -2 || usercnt - random == 1){
```

```
System.out.println("이겼습니다."); //이긴경우
```

```
rinfo.setWin(++win);
```

```
}else
```

```
if(usercnt - random == 0){ //비긴경우
```

```
System.out.println("비겼습니다.");
```

```
rinfo.setDraw(++draw);
```

```
}else{
```

```
//진경우
```

```
System.out.println("졌습니다.");
```

```
rinfo.setLose(++lose);
```

```
}
```

```
System.out.println(rinfo.getWin() + "승, "
```

```
+
```

```
rinfo.getLose() + "패, "
```

```
+rinfo.getDraw() + "무");
```

```
System.out.print("한판 더?? y | n : ");
```

```
String select = scan.next();
```

```
if(!select.equals("y")){
```

```
break;
```

```
}
```

```
}//while()
```

```
System.out.println("게임이 종료되었습니다.");
```

```
// 일
```

```
단 구동해서 가위바위보 잘 되나 확인 해 보시구요,
```

```
// 다
```

```
시하거나 게임종료도 잘 되나 한번 보시구요
```

```
// 이
```

```
제 승률을 파일로 한번 기록해 보도록 할게요
```

```
아 래
```

```
ScoreWirter클래스 구현 마친 후에 추가할 것.
```

```
// 파
```

```
일쓰기
```

```
try {
```



```

ScoreWriter sw = new ScoreWriter(rinfo);

}

catch (Exception e) {

//

TODO Auto-generated catch block

e.printStackTrace();

}

}

```

ScoreWriter클래스 정의

객체를 파일로 저장하기 위한 준비

```
public class ScoreWriter {
```

// 편

의상 생성자에 정의합니다.

```
public ScoreWriter(RspInfo info){
```

// 'RspScore 폴더' 안에 '유저아이디 폴더' 안에 'UserInfo.sav'파일 생성

// 을

위한 경로설정

```
String path =
```

```
"d:/RspScore/" + info.getName().trim() + "/UserInfo.sav";
```

File

```
dir = new File("d:/RspScore");//RspScore폴더를 만들기위한 객체생성
```

```
if(!dir.exists()) //      d:/에 RspScore폴더가 없을 경우
```

```
dir.mkdir(); // RspScore폴더 생성
```

File

```
dir2 = new File(dir, info.getName().trim());
```

```
if(!dir2.exists())
```

```
dir2.mkdir();//위와 마찬가지로 유저이름으로 된 폴더를 생성
```

// 파

일쓰기

try {

```
ObjectOutputStream oos =
```

```
new ObjectOutputStream(new FileOutputStream(path));
```

```
oos.writeObject( info );//RspInfo객체를 파일에 쓴다.
```

```
if(oos != null)
```

```
oos.close();
```

```
System.out.println("기록저장");
```

```
}
```

```
catch (Exception e) {
```

```
//
```

```
TODO Auto-generated catch block
```

```
e.printStackTrace();
```

```
System.out.println("기록저장 실패");
```

```
}
```

```
}  
  
}
```

ScoreLoader클래스 정의

```
public class ScoreLoader {
```

```
    private RspInfo info;
```

```
// 겯
```

터 작성

```
    public RspInfo getInfo() {
```

```
        return info;
```

```
}
```

```
// 생
```

성자에 정의

```
    public ScoreLoader(RspInfo info){
```

```
        this.info = info;
```

```
        String path = "d:/RspScore/"
```

```
+
```

```
        info.getName().trim() + "/UserInfo.sav";
```

```
File
```

```
        f1 = new File(path);
```

```
if(f1.exists()){ //파일이 실제 존재할 때만 수행!
```

```
try {
```

```
ObjectInputStream ois =
```

```
new ObjectInputStream(new FileInputStream(path));
```

```
this.info = (RspInfo)ois.readObject();//Object객체가 값
```

```
//을
```

```
읽어왔기 때문에 캐스팅 후에 객체에 대입
```

```
if(ois
```

```
!= null)
```

```
ois.close();
```

```
System.out.println("로드성공");
```

```
catch (Exception e) {
```

```
}
```

```
TODO Auto-generated catch block
```

```
/ /
```

```
e.printStackTrace();
```

```
System.out.println("로드실패");
```

```
}
```

```
}else{
```

```
System.out.println("새로운 아이디 생성");
```

```
}
```

```
}
```

```
}
```

```
//메인클래스에 파일 읽어오는 부분 추가하고 잘 되나 구동해보고 마무리
```