

I/O

ایجاد یک سیستم ورودی خروجی خوب یکی از وظایف مشکل برای طراح یک زبان است و این با توجه به تعداد روش های مختلف موجود مشهود است.



معرفی

□ جاوا کلاس های زیادی را برای سیستم ورودی خروجی (I/O) در نظر گرفته است که در ابتدا ممکن است کمی گیج کننده به نظر برسد.

□ در ورژن های بعد از Java 1.0 تغییر مهمی در کتابخانه I/O رخ داد و هنگامی که کتابخانه بایت گرای (byte-oriented) پایه با کتابخانه کاراکتر گرا (char oriented) تکمیل شد و کلاس های I/O برپایه unicode ایجاد گردید. کلاس های nio نیز برای بهبود عملکرد و کارایی اضافه شدند.


کلاس File


❑ در واقع "FilePath" نام بهتری برای این کلاس است. از این طریق می توان نام فایل خاص و یا نام های مجموعه ای از فایل ها را در یک دایرکتوری نمایش داد.

❑ در صورتی که مجموعه ای از فایل ها مد نظر باشد می توان از متد `list()` استفاده کرد که آرایه ای را به صورت `String` برمی گرداند.

```
File path = new File("D:\\");  
String[] list;  
list = path.list();
```

بررسی و ایجاد دایرکتوری

می توان از طریق شی File دایرکتوری جدیدی و یا یک مسیر کامل دایرکتوری را در صورتی که وجود نداشته باشد، ایجاد نمود. 

همچنین می توان به ویژگی های فایل ها نیز نگاهی انداخت (سایز، زمان آخرین تغییر، خواندن/نوشتن) 

```
private static void fileData(File f){
    System.out.println(
        "Absolute path: " + f.getAbsolutePath() +
        "\n Can read: " + f.canRead() +
        "\n Can write: " + f.canWrite() +
        "\n getName: " + f.getName() +
        "\n getParent: " + f.getParent() +
        "\n getPath: " + f.getPath() +
        "\n length: " + f.length() +
        "\n lastModified: " + f.lastModified());
    if(f.isFile())
        System.out.println("It's a file");
    else if(f.isDirectory())
        System.out.println("It's a directory");
}
```

بررسی و ایجاد دایرکتوری

```
File old = new File("C:\\Dir1"),
    rname = new File("C:\\Dir2");
old.renameTo(rname);
//Next
File f = new File("D:\\New Folder");
if(f.exists()){
    System.out.println(f + " exists");
    if(del){
        System.out.println("deleting..." + f);
        f.delete();
    } else {
        // Doesn't exist
        if(!del) {
            f.mkdirs();
            System.out.println("created " + f);
        }
    }
}
```

ورودی و خروجی

□ کلاس های کتابخانه جاوا برای i/o به دو گروه ورودی و خروجی تقسیم شده اند.

□ از طریق ارث بری، هر چیزی که از کلاس های InputStream یا Reader مشتق شده باشد دارای متدهای پایه ای به نام read () برای خواندن یک بایت یا آرایه ای از بایت ها می باشد.

□ به همین ترتیب، هر چیزی که از کلاس های OutputStream یا Writer مشتق شده باشد دارای متدهای پایه ای به نام write () برای نوشتن یک بایت یا آرایه ای از بایت ها می باشد.

انواع InputStream

□ یک شی String

□ یک فایل

□ یک pipe که مانند لوله فیزیکی کار می کند و می توان اشیا را در یک طرف آن وارد کرد و از طرف دیگر خارج نمود.

□ توالی ای از stream های دیگر، بنابراین می توان آن ها را با هم در یک جریان (stream) جمع آوری کرد.

□ منابع دیگر، مثل Internet connection

InputStream أنواع

Class	Function
ByteArrayInputStream	Allows a buffer in memory to be used as an InputStream.
StringBufferInputStream	Converts a String into an InputStream.
FileInputStream	For reading information from a file.
PipedInputStream	Produces the data that's being written to the associated PipedOutputStream. Implements the "piping" concept.
SequenceInputStream	Converts two or more InputStream objects into a single InputStream.
FilterInputStream	Abstract class that is an interface for decorators that provide useful functionality to the other InputStream classes.

انواع InputStream

```
FileInputStream fis = new FileInputStream("d:\\text.txt");
int i;
while ((i = fis.read()) != -1) {
    System.out.println(i);
}
fis.close();
```

OutputStream انواع

Class	Function
ByteArrayOutputStream	Creates a buffer in memory. All the data that you send to the stream is placed in this buffer.
FileOutputStream	For sending information to a file.
PipedOutputStream	Any information you write to this automatically ends up as input for the associated PipedInputStream . Implements the "piping" concept.
FilterOutputStream	Abstract class that is an interface for decorators that provide useful functionality to the other OutputStream classes.

OutputStream انواع

```
File file = new File("D:/text.txt");  
FileOutputStream fos = new FileOutputStream(file);  
fos.write("Hello".getBytes());  
fos.close();
```

خواندن از یک InputStream از طریق FilterInputStream

□ دلیل وجود کلاس های “filter” در کتابخانه I/O جاوا این است که کلاس

انتزاعی filter به عنوان کلاسی پایه برای تمام decoratorها است.

□ کلاس هایی که واسط decorator را برای کنترل InputStream یا

OutputStream خاصی فراهم می کنند، FilterInputStream و

FilterOutputStream هستند که نام های بسیار شهودی ای ندارند.

نوشتن در یک OutputStream از طریق FilterOutputStream

Class	Function
DataOutputStream	Used in concert with DataInputStream so you can write primitives (int , char , long , etc.) to a stream in a portable fashion.
PrintStream	For producing formatted output. While DataOutputStream handles the <i>storage</i> of data, PrintStream handles <i>display</i> .
BufferedOutputStream	Use this to prevent a physical write every time you send a piece of data. You're saying, "Use a buffer." You can call flush() to flush the buffer.

Writer و Reader ها

□ کلاس های `InputStream` و `OutputStream` همچنان قابلیت های ارزشمندی را در قالب `I/O` بایت گرا دارند. گرچه کلاس های `Reader` و `Writer` سازگاری با `unicode` و در واقع `I/O` بر پایه کاراکتر را فراهم می کنند.

□ مواردی وجود دارد که باید کلاس هایی با سلسله مراتب بایتی را در ترکیبی با کلاس های با سلسله مراتب کاراکتر استفاده کنیم:

`InputStreamReader : InputStream -> Reader,`

و

`OutputStreamWriter : OutputStream -> Writer.`

□ مهمترین دلیل برای سلسله مراتب `Reader` و `Writer` به جهت بین المللی کردن است.

Writer & Reader

Sources & sinks: Java 1.0 class	Corresponding Java 1.1 class
InputStream	Reader adapter: InputStreamReader
OutputStream	Writer adapter: OutputStreamWriter
FileInputStream	FileReader
FileOutputStream	FileWriter
StringBufferInputStream (deprecated) (no corresponding class)	StringReader StringWriter
ByteArrayInputStream	CharArrayReader
ByteArrayOutputStream	CharArrayWriter
PipedInputStream	PipedReader
PipedOutputStream	PipedWriter

تغییر دادن رفتار stream

Filters: Java 1.0 class	Corresponding Java 1.1 class
FilterInputStream	FilterReader
FilterOutputStream	FilterWriter (abstract class with no subclasses)
BufferedInputStream	BufferedReader (also has readLine())
BufferedOutputStream	BufferedWriter
DataInputStream	Use DataInputStream (except when you need to use readLine() , when you should use a BufferedReader)
PrintStream	PrintWriter
LineNumberInputStream (deprecated)	LineNumberReader
StreamTokenizer	StreamTokenizer (Use the constructor that takes a Reader instead)
PushbackInputStream	PushbackReader

فایل ورودی بافرشده

```
import java.io.*;
public class BufferedInputFile
{
    public static String read(String filename) throws IOException
    {
        // Reading input by lines:
        BufferedReader in = new BufferedReader(new FileReader(filename));
        String s;
        StringBuilder sb = new StringBuilder();
        while((s = in.readLine()) != null)
            sb.append(s + "\n");
        in.close();
        return sb.toString();
    }
    public static void main(String[] args) throws IOException
    {
        System.out.print(read("BufferedInputFile.java"));
    }
}
```

RandomAccessFile

RandomAccessFile در فایل هایی که شامل رکودهایی با سایز مشخص □

هستند، استفاده می شوند و بنابراین می توان از طریق seek از رکوردی به ()

. رکورد دیگر حرکت کرد و سپس رکوردها را خواند و یا تغییر داد

RandomAccessFile اساسا رفتاری متفاوت با سایر نوع های I/O دارد □

. چرا که می توان در طول یک فایل به جلو یا عقب حرکت کرد

RandomAccessFile

□ DataOutputStream همراه با متدهای getFilePointer () جهت پیدا کردن مکان فعلی در فایل، با

متد seek () جهت حرکت به مکان جدیدی در فایل و با متد length () جهت مشخص کردن

. بیشترین سائز فایل استفاده می شود

• به علاوه constructor ها نیازمند آرگومان دومی جهت مشخص کردن این است که آیا شما تنها

به صورت تصادفی read می کنید ("r") و یا اینکه عملیات read و write را انجام می دهید.

("rw")

I/O استاندارد

□ در پیروی از مدل استاندارد I/O جاوا دارای `System.out`، `System.in` و `System.err` می باشد.

□ معمولا ورودی را به صورت یک خط در هر زمان با استفاده از `readLine()` می خوانیم.

```
BufferedReader stdin = new BufferedReader(  
    new InputStreamReader(System.in));  
String s;  
while((s = stdin.readLine()) != null && s.length() != 0)  
    System.out.println(s);
```

تغییر System.out به یک PrintWriter

تغییر System.out به یک PrintWriter □

```
PrintWriter out = new PrintWriter(System.out, true);  
out.println("Hello, world");
```

دومین آرگومان true می گردد تا آزاد سازی حافظه به صورت خودکار انجام گیرد، در غیر این صورت ممکن است نتوانید خروجی را ببینید.

هدایت دوباره ی استاندارد I/O: □

کلاس System در جاوا امکان هدایت input، output و error جریان های ورودی/خروجی را با استفاده از فراخوانی های متد static ساده را فراهم می کند:

```
setIn(InputStream)  
setOut(PrintStream)  
setErr(PrintStream)
```

کنترل فرآیند

□ اغلب نیاز به اجرای برنامه های دیگر سیستم عامل از داخل جاوا را داریم و باید ورودی و خروجی

از چنین برنامه هایی را کنترل کنیم. کتابخانه جاوا کلاس هایی را برای انجام چنین عملیاتی فراهم

می کند:

```
Process process = new ProcessBuilder(command.split(" ")).start();
BufferedReader results = new BufferedReader(new
    InputStreamReader(process.getInputStream()));
String s;
while((s = results.readLine()) != null)
    System.out.println(s);
```

فشرده‌گی

Compression class	Function
CheckedInputStream	GetChecksum() produces checksum for any InputStream (not just decompression).
CheckedOutputStream	GetChecksum() produces checksum for any OutputStream (not just compression).
DeflaterOutputStream	Base class for compression classes.
ZipOutputStream	A DeflaterOutputStream that compresses data into the Zip file format.
GZIPOutputStream	A DeflaterOutputStream that compresses data into the GZIP file format.
InflaterInputStream	Base class for decompression classes.
ZipInputStream	An InflaterInputStream that decompresses data that has been stored in the Zip file format.
GZIPInputStream	An InflaterInputStream that decompresses data that has been stored in the GZIP file format.

Object serialization

□ از طریق object serialization در جاوا می توان هر object را که واسط serializable را پیاده سازی می

کند، به صورت توالی ای از بایت ها تبدیل کرد به شیوه ای که بعدا بتوان از آن object اصلی را ایجاد کرد.

1. سریال کردن یک object تا زمانی که واسط serializable را پیاده سازی می کند، کار نسبتا ساده ای است.

2. برای سریال کردن یک object ترتیبی از شی OutputStream را ایجاد کنید و سپس آن را در داخل شی

ObjectOutputStream قرار دهید. اکنون تنها نیاز به فراخوانی writeObject () را دارید.

3. برای معکوس کردن روند، یک InputStream را در داخل ObjectInputStream قرار دهید و سپس

readObject () را فراخوانی کنید.

Object serialization

```
Worm w = new Worm(6, 'a'); //Implements Serializable
print("w = " + w);
ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream("worm.out"));
out.writeObject("Worm storage\n");
out.writeObject(w);
out.close();
```

```
ObjectInputStream in2 = new ObjectInputStream(new
ByteArrayInputStream(bout.toByteArray()));
String s = (String)in2.readObject();
Worm w3 = (Worm)in2.readObject();
```

کلید واژه transient

□ در صورتی که با یک شی کار می کنید، این در حالی است که همه عملیات های سریال

کردن به صورت خودکار رخ می دهند. برای کنترل این کار می توان عملیات سریال کردن را

به صورت فیلد به فیلد بر پایه کلید واژه transient غیر فعال کرد.

```
public class Logon implements Serializable
{
    private Date date = new Date();
    private String username;
    private transient String password;
    public Logon(String name, String pwd)
    {
        username = name;
        password = pwd;
    }
}
```