CS202 Java Object Oriented Programming
Input and Output

Chengyu Sun
California State University, Los Angeles

## Input and Output

- Console
  - Scanner
  - System.out
- Command line parameters
- File
  - Reader and Writer
  - InputStream and OutputStream
  - Scanner
- GUI

## Command Line Parameters

- Parameters of main()
  - public static void main( String args[] )
- *java Classname <arg0> <arg1> …*
- Eclipse
  - *Run –> Run … –>Arguments*

## CLP Example

- Add up a list of integers from user input

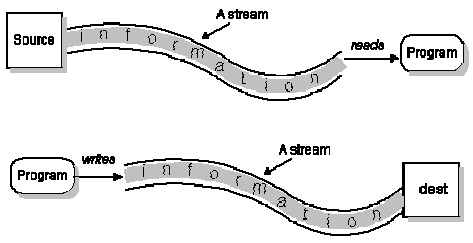
```
public class Add {

   public static void main( String args[] )
   {
      int sum = 0;
      for( int i=0 ; i < args.length ; ++i )
         sum += Integer.parseInt(args[i]);

      System.out.println(sum);
   }

} // end of class Add
```

## Understand `java.io` Package

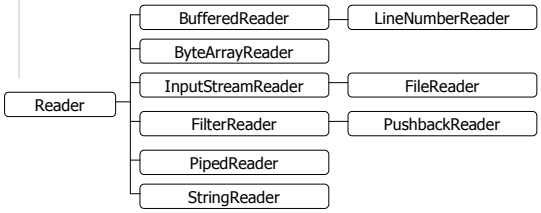



http://java.sun.com/j2se/1.5.0/docs/api/java/io/package-summary.html

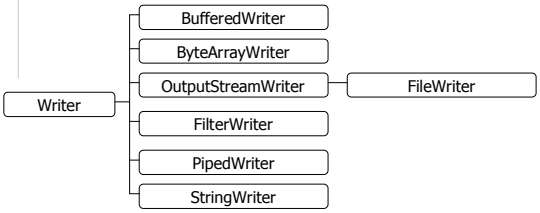
## Stream Types

- Character streams
  - Textual information
  - Handled by *Reader* and *Writer* classes
- Byte streams
  - Binary information
  - Handled by *InputStream* and *OutputStream* classes

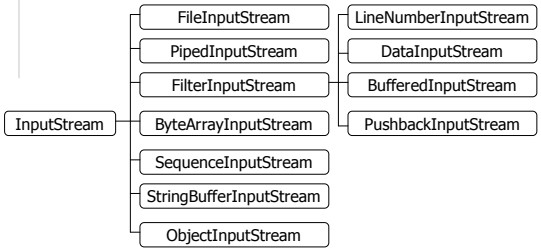
## Reader Classes



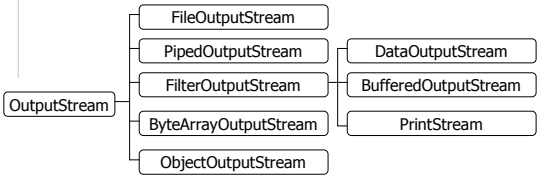

## Writer Classes




## InputStream Classes




## OutputStream Classes




## Basic Streams by Source/Destination

- Files
  - FileReader/Writer/InputStream/OutputStream
- Threads
  - PipedReader/Writer/InputStream/OutputStream
- Memory
  - ByteArrayReader/Writer/InputStream/OutputStream
  - StringReader/Writer
  - StringBufferInputStream
- General
  - InputStream, InputStreamReader
  - OutputStream, OutputStreamWriter

## Basic Stream Operations

- Basic streams only recognize *bytes* or *characters*
- Operations
  - Read/write a single byte or character
  - Read/write an array of bytes or characters
- Inconvenient

## Wrapper Streams by Function

- Data conversion
  - DataInputStream/OutputStream
- Printing
  - PrintStream
- Buffering
  - BufferedReader/Writer/InputStream/OutputStream
- Object serialization
  - ObjectInputStream/OutputStream
- Others

## Important Wrapper Streams and Operations

- DataInputStream and DataOutputStream
  - Read and write primitive types
  - readInt(), readDouble(), …
  - writeInt( int i ), writeDoube( double d ), …
- BufferedReader
  - readLine()
- BufferedWriter
  - write( String s )

## "Wrapping" Examples

```
// buffered text file read/write
BufferedReader br = new BufferedReader( new FileReader("file") );
BufferedWriter bw = new BufferedWriter( new FileWriter("file") );

// un-buffered binary file read/write
DataInputStream di = new DataInputStream( new FileInputStream("file") );
DataOutputStream do = new DataOutputStream( new FileOutputStream("file") );

// buffered binary file read/write
DataInputStream di2 = new DataInputStream( new BufferedInputStream(
                                    new FileInputStream() ) );
DataOutputStream do2 = new DataOutputStream( new BufferedOutputStream(
                                    new FileOutputStream() ) );
```

## How to Choose from Stream Classes

- Step 1: Is the data in **binary** form or **textual** form?
  - Binary: Input/OutputStream
  - Textual: Reader/Writer
- Step 2: What's the data **source** or data **destination**?
  - Files, threads, memory, general
- Step 3: How to **process** the data?
  - Primitive data types, buffering, …

## File Input Example

- Read from a file in the following format, and sum up all numbers

```
31    20    30
22    33    –1    43
23    33    44    45
79    1
-10
```

## Get The File Name

```
public static void main( String args[] )
{
   if( args.length == 0 )
   {
      System.err.println( "usage: java Sum <filename>" );
      System.exit(1);
   }

   // do something with args[0]
}
```

## Paths

- Windows
  - Absolute path
    - c:\path\to\file
  - Relative path
    - path\to\file
- Unix
  - Absolute path
    - /path/to/file
  - Relative path
    - path/to/file

- File separators – "**/**", "****", **File.separator**

## Read In Each Line

```
FileReader fr = new FileReader( filename );

// wrapping
BufferedReader br = new BufferedReader( fr );

String line;
while( (line = br.readLine()) != null )
{
   // do something with s
}
```

## Break A Line Into Tokens

```
StringTokenizer st = new StringTokenizer(line);

while( st.hasMoreTokens() )
{
   int value = Integer.parseInt( st.nextToken() );

   // add value to sum
}
```

## A Few More Things

- I/O Classes are in the java.io package
  - import java.io.*;
- StringTokenizer is in the java.util package
  - import java.util.*;
- File operations throw all kinds of exceptions
  - Catch them, or
  - Throw them
- Always remember to *close* a stream

## `File` Class

- Not directly related to I/O
- Check file status:
  - is a file or a directory
  - exist, readable, writable
  - name, path, parent
  - length

## Binary File vs. Text File

- If we can save data in either binary or text form, which one do we choose?
  - File size
  - Convenience
  - Speed
- Either way, always use buffering!

## Random Access File

- The problem with the *stream* model
- Advantages of `RandomAccessFile`
  - Deal with both binary and text files
  - Provide both read and write methods
  - `seek(long pos)`
- … but you'll probably never use it. Why?