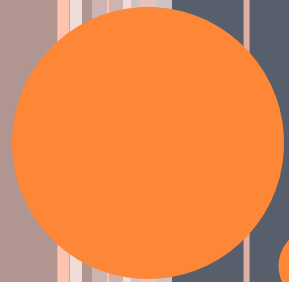




AGENDA

- IO Fundamentals
- File IO



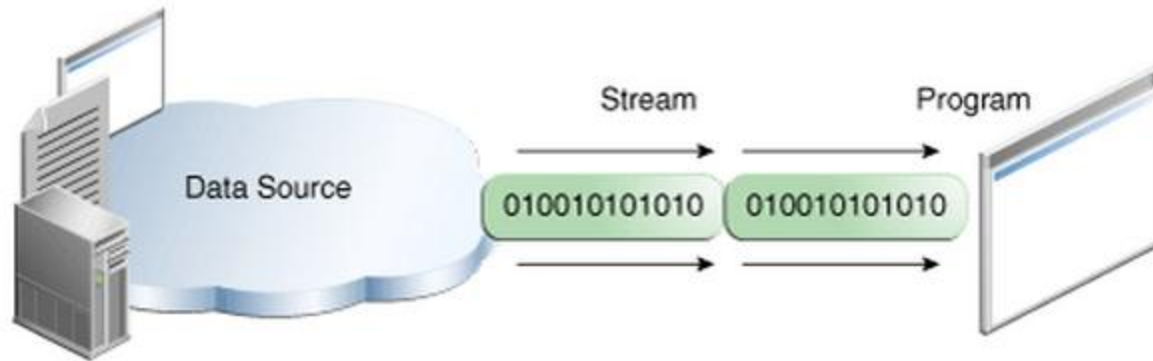


INPUT OUTPUT FUNDAMENTALS

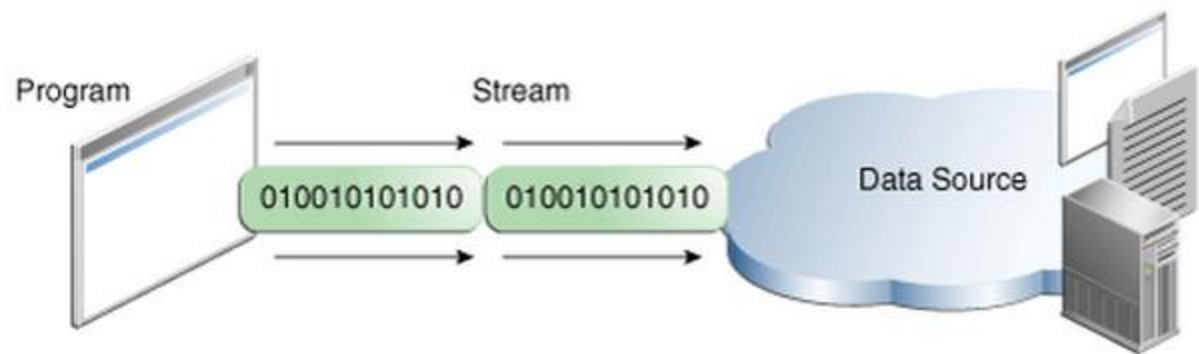
I/O FUNDAMENTALS

- A *stream* is a flow of data from a source or to a sink.
- A *source* stream initiates the flow of data, also called an input stream.
- A *sink* stream terminates the flow of data, also called an output stream.
- Sources and sinks are both *node streams*.
- Types of node streams are files, memory, and pipes between threads or processes.





Reading information into a program.



Writing information from a program.



DATA WITHIN STREAMS

- Java technology supports two types of streams: character and byte.
- Input and output of character data is handled by readers and writers.
- Input and output of byte data is handled by input streams and output streams:
 - Normally, the term *stream* refers to a byte stream.
 - The terms *reader* and *writer* refer to character streams.



Fundamental Stream Classes

Stream	Byte Streams	Character Streams
Source streams	InputStream	Reader
Sink streams	OutputStream	Writer



BYTE STREAMS

- Programs use *byte streams* to perform input and output of 8-bit bytes.
- All byte stream classes are descended from Input Stream and Output Stream.



THE INPUTSTREAM METHODS

- The three basic read methods are:

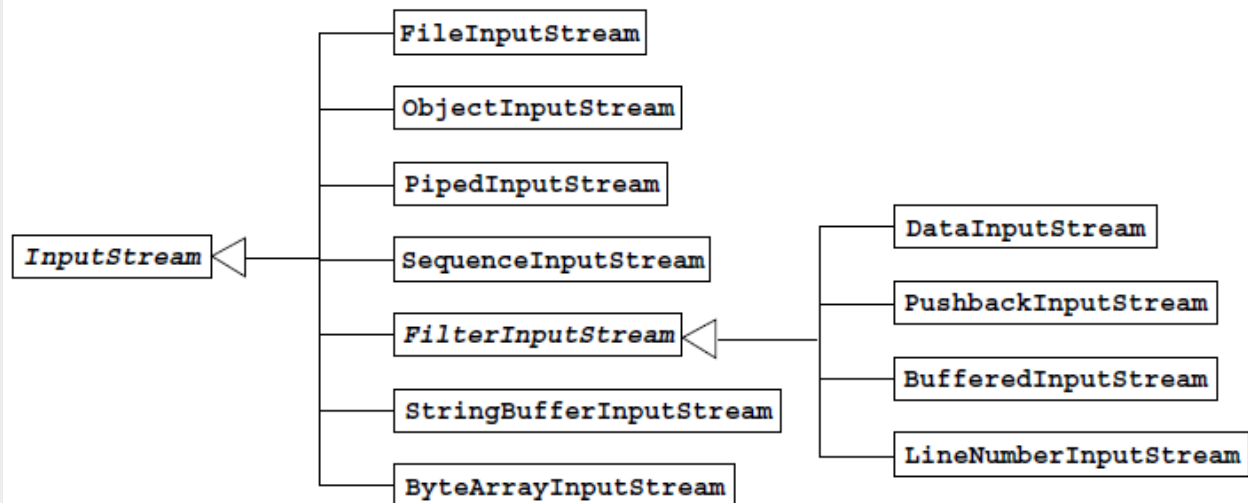
```
int read()  
int read(byte[] buffer)  
int read(byte[] buffer, int offset, int length)
```

- Other methods include:

```
void close()  
int available()  
long skip(long n)  
boolean markSupported()  
void mark(int readlimit)  
void reset()
```



The InputStream Class Hierarchy



THE OUTPUTSTREAM METHODS

- The three basic write methods are:

```
void write(int c)
```

```
void write(byte[] buffer)
```

```
void write(byte[] buffer, int offset, int length)
```

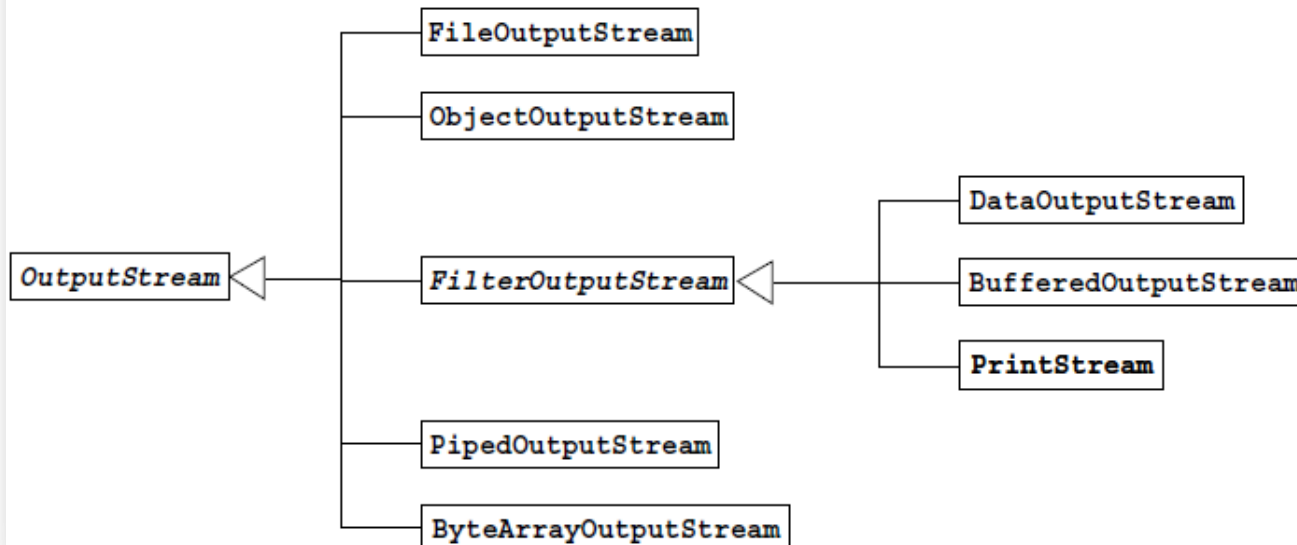
- Other methods include:

```
void close()
```

```
void flush()
```



The OutputStream Class Hierarchy



CHARACTER STREAM

- The Java platform stores character values using Unicode conventions.
- Character stream I/O automatically translates this internal format to and from the local character set.
- All character stream classes are descended from Reader and Writer



READER

- The three basic read methods are:

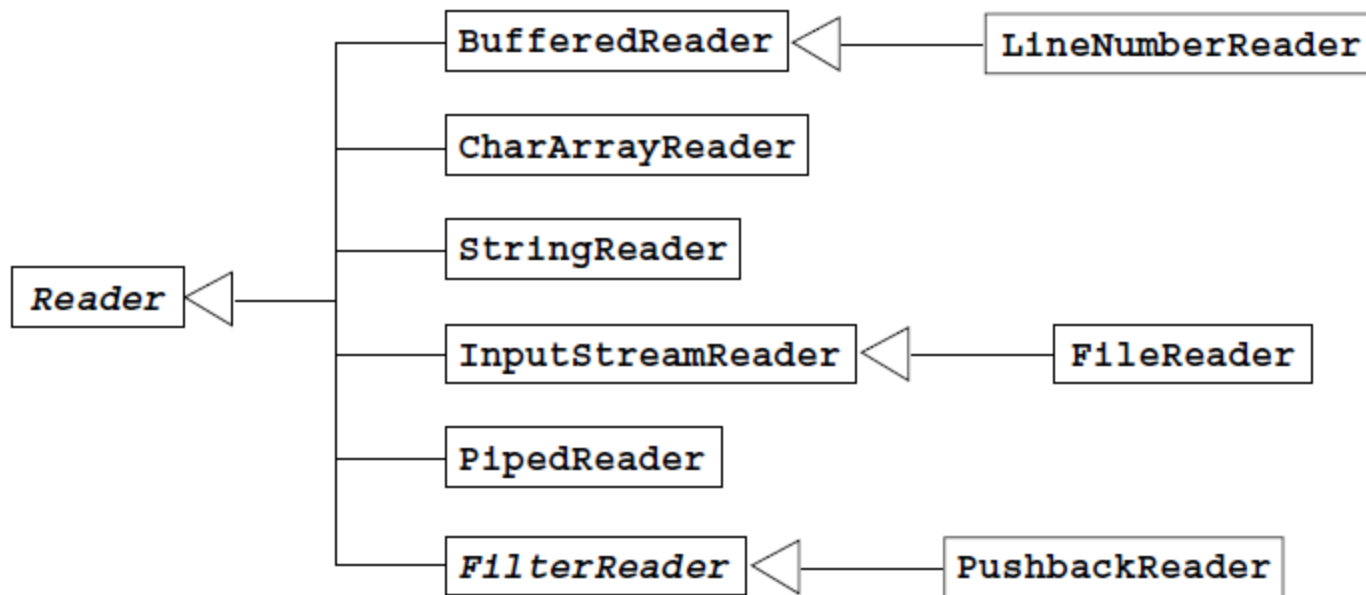
```
int read()  
int read(char[] cbuf)  
int read(char[] cbuf, int offset, int length)
```

- Other methods include:

```
void close()  
boolean ready()  
long skip(long n)  
boolean markSupported()  
void mark(int readAheadLimit)  
void reset()
```



The Reader Class Hierarchy



WRITER

- The basic write methods are:

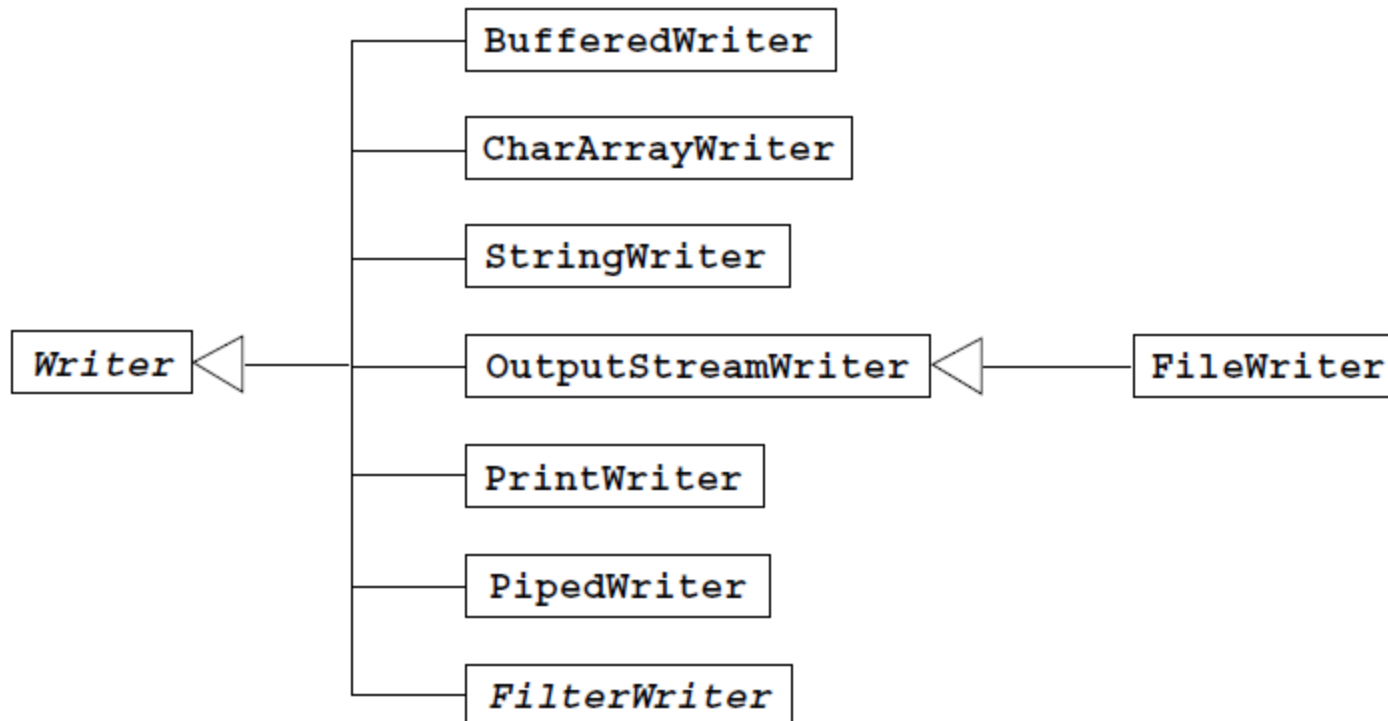
```
void write(int c)
void write(char[] cbuf)
void write(char[] cbuf, int offset, int length)
void write(String string)
void write(String string, int offset, int length)
```

- Other methods include:

```
void close()
void flush()
```



The Writer Class Hierarchy



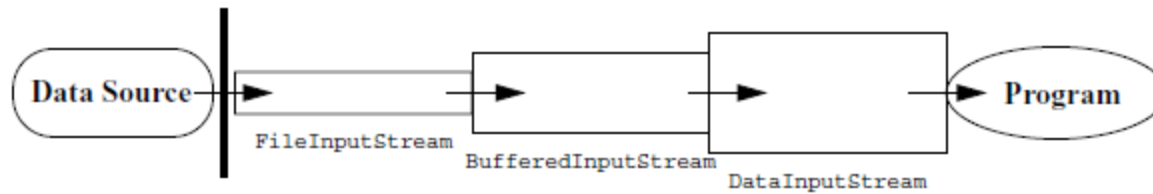
NODE STREAMS

Type	Character Streams	Byte Streams
File	FileReader FileWriter	FileInputStream FileOutputStream
Memory: array	CharArrayReader CharArrayWriter	ByteArrayInputStream ByteArrayOutputStream
Memory: string	StringReader StringWriter	N/A
Pipe	PipedReader PipedWriter	PipedInputStream PipedOutputStream

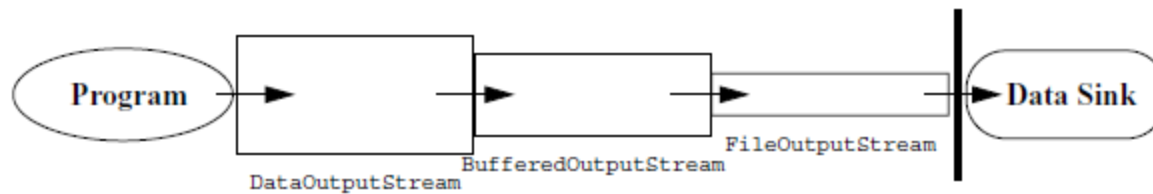


I/O STREAM CHAINING

Input Stream Chain



Output Stream Chain



PROCESSING STREAMS

Type	Character Streams	Byte Streams
Buffering	BufferedReader BufferedWriter	BufferedInputStream BufferedOutputStream
Filtering	<i>FilterReader</i> <i>FilterWriter</i>	<i>FilterInputStream</i> <i>FilterOutputStream</i>
Converting between bytes and character	InputStreamReader OutputStreamWriter	
Performing object serialization		ObjectInputStream ObjectOutputStream
Performing data conversion		DataInputStream DataOutputStream
Counting	LineNumberReader	LineNumberInputStream
Peeking ahead	PushbackReader	PushbackInputStream
Printing	PrintWriter	PrintStream



CONSOLE I/O

- The variable `System.out` enables you to write to *standard output*.

`System.out` is an object of type `PrintStream`.

- The variable `System.in` enables you to read from *standard input*.

`System.in` is an object of type `InputStream`.

- The variable `System.err` enables you to write to *standard error*.

`System.err` is an object of type `PrintStream`.



READING FROM STANDARD INPUT

```
import java.io.*;

public class KeyboardInput {
    public static void main (String args[]) {
        String s;
        // Create a buffered reader to read
        // each line from the keyboard.
        InputStreamReader ir
            = new InputStreamReader(System.in);
        BufferedReader in = new BufferedReader(ir);

        System.out.println("Unix: Type ctrl-d to exit." +
                           "\nWindows: Type ctrl-z to exit");

        try {
            // Read each input line and echo it to the screen.
            s = in.readLine();
            while ( s != null ) {
                System.out.println("Read: " + s);
                s = in.readLine();
            }

            // Close the buffered reader.
            in.close();
        } catch (IOException e) { // Catch any IO exceptions.
            e.printStackTrace();
        }
    }
}
```



SIMPLE FORMATTED INPUT

- The Scanner class provides a formatted input function.
- A Scanner class can be used with console input streams as well as file or network streams.
- You can read console input as follows:

```
import java.io.*;
import java.util.Scanner;
public class ScanTest {
    public static void main(String [] args) {
        Scanner s = new Scanner(System.in);
        String param = s.next();
        System.out.println("the param 1" + param);
        int value = s.nextInt();
        System.out.println("second param" + value);
        s.close();
    }
}
```



FILES AND FILE I/O

- The java.io package enables you to do the following:
 - Create File objects
 - Manipulate File objects
 - Read and write to file streams



CREATING A NEW FILE OBJECT

- The File class provides several utilities:
 - `File myFile;`
 - `myFile = new File("myfile.txt");`
 - `myFile = new File("MyDocs", "myfile.txt");`
- Directories are treated like files in the Java programming language.

```
File myDir = new File("MyDocs");  
myFile = new File(myDir, "myfile.txt");
```



THE FILE TESTS AND UTILITIES

○ File information:

- String getName()
- String getPath()
- String getAbsolutePath()
- String getParent()
- long lastModified()
- long length()

○ File tests:

- boolean exists()
- boolean canWrite()
- boolean canRead()
- boolean isFile()
- boolean isDirectory()
- boolean isAbsolute();
- boolean isHidden();

○ File modification:

- boolean renameTo(File newName)
- boolean delete()

○ Directory utilities:

- boolean mkdir()
- String[] list()



FILE STREAM I/O

- For file input:
 - Use the `FileReader` class to read characters.
 - Use the `BufferedReader` class to use the `readLine` method.
- For file output:
 - Use the `FileWriter` class to write characters.
 - Use the `PrintWriter` class to use the `print` and `println` methods.



FILE INPUT EXAMPLE

```
import java.io.*;
public class ReadFile {
    public static void main (String[] args) {
        // Create file
        File file = new File(args[0]);

        try {
            // Create a buffered reader
            // to read each line from a file.
            BufferedReader in
                = new BufferedReader(new FileReader(file));
            String s;

            // Read each line from the file and echo it to the screen.
            s = in.readLine();
            while ( s != null ) {
                System.out.println("Read: " + s);
                s = in.readLine();
            }
            // Close the buffered reader
            in.close();

        } catch (FileNotFoundException e1) {
            // If this file does not exist
            System.err.println("File not found: " + file);

        } catch (IOException e2) {
            // Catch any other IO exceptions.
            e2.printStackTrace();
        }
    }
}
```



FILE OUTPUT EXAMPLE

```
import java.io.*;

public class WriteFile {
    public static void main (String[] args) {
        // Create file
        File file = new File(args[0]);

        try {
            // Create a buffered reader to read each line from standard in.
            InputStreamReader isr
                = new InputStreamReader(System.in);
            BufferedReader in
                = new BufferedReader(isr);
            // Create a print writer on this file.
            PrintWriter out
                = new PrintWriter(new FileWriter(file));
            String s;
            System.out.print("Enter file text.  ");
            System.out.println("[Type ctrl-d to stop.]");

            // Read each input line and echo it to the screen.
            while ((s = in.readLine()) != null) {
                out.println(s);
            }

            // Close the buffered reader and the file print writer.
            in.close();
            out.close();

        } catch (IOException e) {
            // Catch any IO exceptions.
            e.printStackTrace();
        }
    }
}
```

