

## Objectives

- Standard Error
- Streams
  - Byte Streams
  - Text Streams

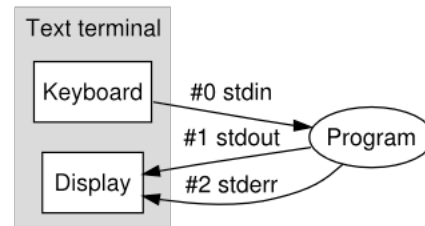
## STANDARD ERROR

## Standard Streams

- Preconnected streams

- Standard Out: `stdout`
- Standard In: `stdin`
- *Standard Error: `stderr`*

- For error messages and diagnostics
- In Java: `System.err`



Benefits of two output streams (out and err)?

Oct 5, 2016

Sprenkle - CSCI209

3

## Redirecting Output

- Recall earlier this semester

```
> java OlympicScore > score.out
```

- Redirected `stdout` to `score.out`
- `stderr` would still go to terminal

- To redirect `stderr` to file as well

```
> java OlympicScore >& score.out
```

Oct 5, 2016

Sprenkle - CSCI209

4

# STREAMS

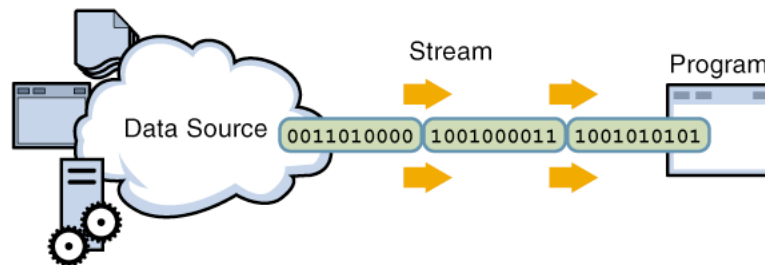
Oct 5, 2016

Sprenkle - CSCI209

5

## Streams

- Java handles input/output using *streams*, which are sequences of bytes



**input stream:** an object from which we can **read** a **sequence** of bytes

**abstract class:** `java.io.InputStream`

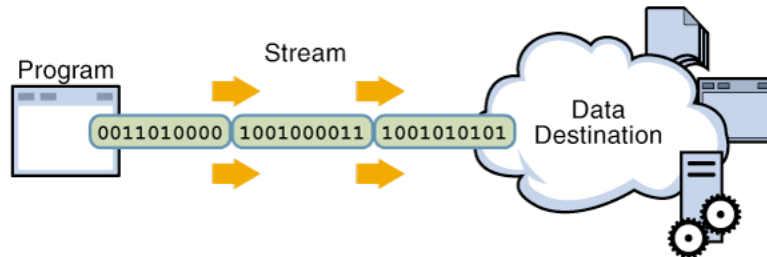
Oct 5, 2016

Sprenkle - CSCI209

6

## Streams

- Java handles input/output using **streams**, which are sequences of bytes



**output stream**: an object to which we can **write** a sequence of bytes

**abstract** class: `java.io.OutputStream`

Oct 5, 2016

Sprenkle - CSCI209

7

## java.io Classes Overview

- Two types of stream classes, based on type of data: Byte, Text

- Abstract base classes for binary data:

`InputStream`

`OutputStream`

- Abstract base classes for text data:

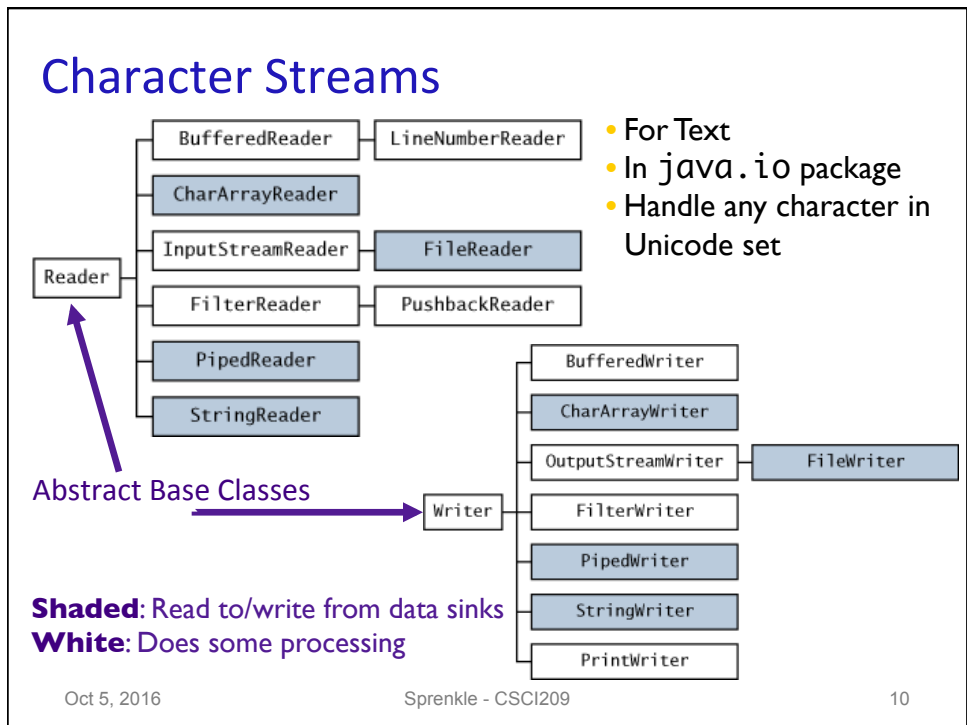
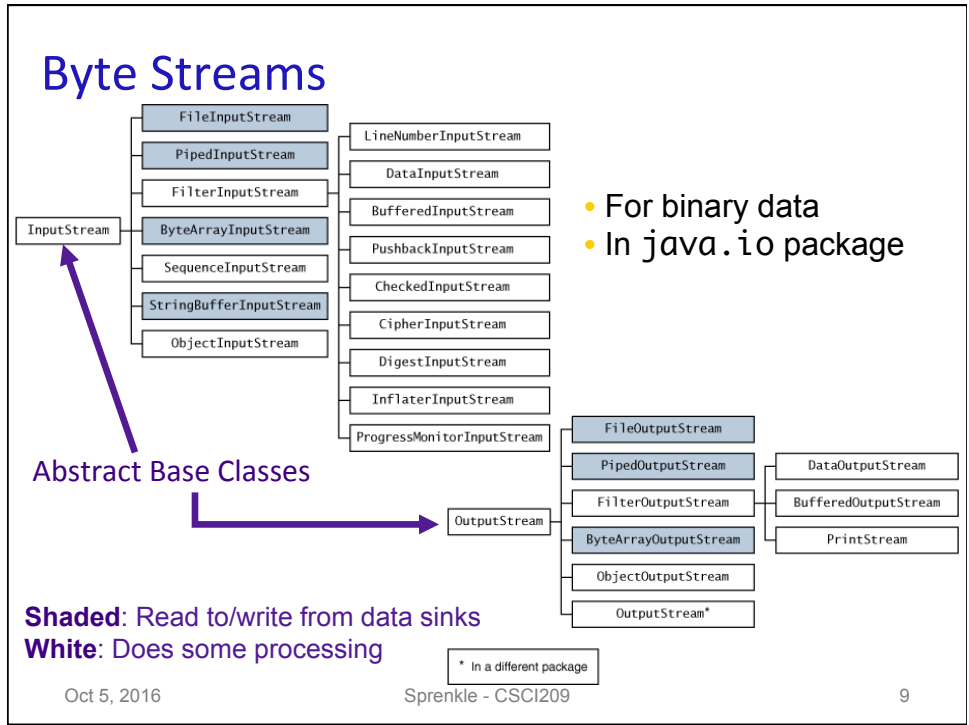
`Reader`

`Writer`

Oct 5, 2016

Sprenkle - CSCI209

8



# STREAMS

Oct 5, 2016

Sprenkle - CSCI209

11

## File Input and Output Streams

- **FileInputStream**: provides an input stream that can read from a file

- Constructor takes the name of the file:

```
FileInputStream fin = new  
    FileInputStream("chicken.data");
```

- Or, uses a **File** object ...

```
File inputFile = new File("chicken.data");  
FileInputStream fin = new FileInputStream(inputFile);
```

Oct 5, 2016

Sprenkle - CSCI209

**FileTest.java**

12

## More Powerful Stream Objects

- **DataInputStream**

- Reads Java primitive types through methods such as `readDouble()`, `readChar()`, `readBoolean()`

- **DataOutputStream**

- Writes Java primitive types with `writeDouble()`, `writeChar()`, ...

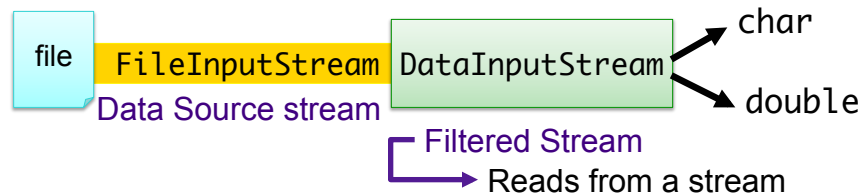
## Connected Streams

Our goal: read numbers from a file

- **FileInputStream** can read from a file but has no methods to read numeric types
- **DataInputStream** can read numeric types but has no methods to read from a file
- Java allows you to **combine** two types of streams into a **connected stream**
  - **FileInputStream** → chocolate
  - **DataInputStream** → peanut butter

## Connected Streams

- Think of a stream as a pipe
- `FileInputStream` knows how to read from a file
- `DataInputStream` knows how to read an `InputStream` into useful types
- Connect **out** end of `FileInputStream` to **in** end of `DataInputStream`...



Oct 5, 2016

Sprenkle - CSCI209

15

## Connecting Streams

- If we want to read numbers from a file
  - `FileInputStream` reads bytes from file
  - `DataInputStream` handles numeric type reading
- Connect the `DataInputStream` to the `FileInputStream`
  - `FileInputStream` gets the bytes from the file and `DataInputStream` reads them as assembled types

```
FileInputStream fin = new
    FileInputStream("chicken.data");
DataInputStream din = new
    DataInputStream(fin); "wrap" fin in din
double num1 = din.readDouble();
```

Oct 5, 2016

Sprenkle - CSCI209 `DataIODemo.java` 16



## Data Source vs. Filtered Streams

### Data Source Streams

- Communicate with a data source
  - file, byte array, network socket, or URL

### Filtered Streams

- Subclasses of `FilterInputStream` or `FilterOutputStream`
- Always contains another stream
- Adds *functionality* to other stream
  - Automatically buffered IO
  - Automatic compression
  - Automatic encryption
  - Automatic conversion between objects and bytes

Oct 5, 2016

Sprenkle - CSCI209

17

## Buffered Streams

- Use a `BufferedInputStream` object to buffer your input streams
  - A pipe in the chain that adds buffering
  - Speeds up access

```
DataInputStream din = new DataInputStream (
    new BufferedInputStream (
        new FileInputStream("chicken.data")));
```



What functionality does each stream add?

Oct 5, 2016

Sprenkle - CSCI209

18

## Connected Streams

Combine different types of streams  
to get functionality you want

- What are the tradeoffs for this design decision?

## Connected Streams

Combine different types of streams  
to get functionality you want

- Creating a class for every class would result in even more classes and a lot of redundant code
  - Consider what is required if some functionality must be updated

## Connected Streams: Output

Combine different types of streams  
to get functionality you want

- Similar for output
  - For buffered output to the file and to write types
    - Create a `FileOutputStream`
    - Attach a `BufferedOutputStream`
    - Attach a `DataOutputStream`
    - Perform typed writing using methods of the `DataOutputStream` object

## TEXT STREAMS

## Text Streams

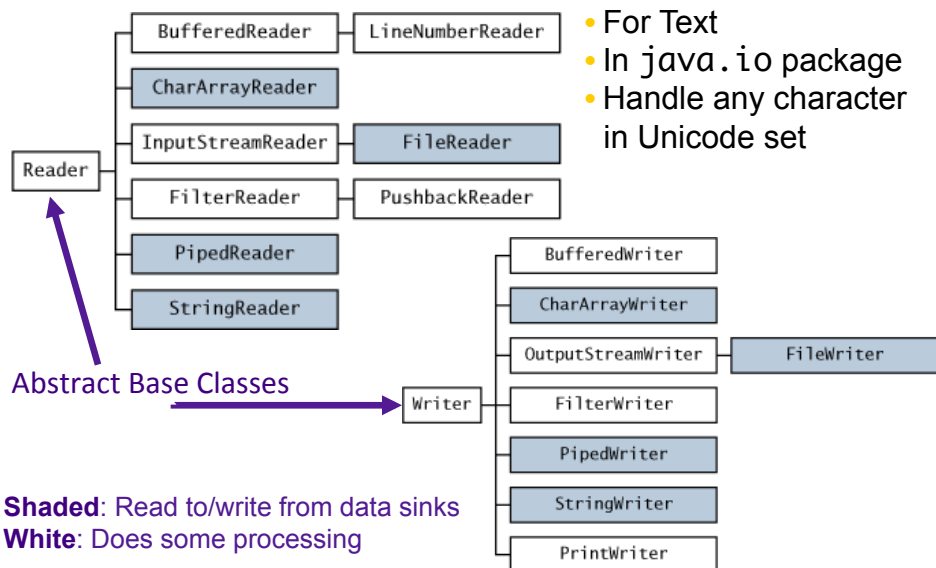
- Previous streams: operate on *binary* data, not text
- Java uses Unicode to represent characters/strings and some operating systems do not
  - Need something that converts characters from Unicode to whatever encoding the underlying operating system uses
  - Luckily, this is mostly hidden from you

Oct 5, 2016

Sprenkle - CSCI209

23

## Character Streams



Oct 5, 2016

Sprenkle - CSCI209

24

## Text Streams

- Derived from **Reader** and **Writer** classes
  - Reader and Writer generally refer to **text I/O**
- Example: Make an input reader of type **InputStreamReader** that reads from keyboard

```
InputStreamReader in = new
    InputStreamReader(System.in);
```

- **in** reads characters from keyboard and converts them into Unicode for Java

Oct 5, 2016

Sprenkle - CSCI209

25


## Text Streams and Encodings

- Attach an **InputStreamReader** to a **FileInputStream**

```
InputStreamReader in = new InputStreamReader(
    new FileInputStream("employee.data"));
```

- Assumes file has been encoded in the default encoding of underlying OS
- You can specify a different *encoding* in constructor of **InputStreamReader**...

```
InputStreamReader in = new InputStreamReader(
    new FileInputStream("employee.data"), "UTF-8");
```



Oct 5, 2016

Sprenkle - CSCI209

26

## Convenience Classes

- Reading and writing to text files is common
- **FileReader**

- Convenience class *combines* a **InputStreamReader** with a **FileInputStream**

- Similar for output of text file

```
FileWriter out = new FileWriter("output.txt");
```

is equivalent to

```
OutputStreamWriter out = new OutputStreamWriter(
    new FileOutputStream("output.txt"));
```

## PrintWriter

- Use for writing text output
  - Easiest writer to use
- Similar to a **DataOutputStream**, **PrintStream** → has methods for printing various data types
- Methods: **print**, **printf** and **println**
  - Similar to **System.out** (a **PrintStream**) to display strings

## PrintWriter Example

File to write to



```
PrintWriter out = new PrintWriter("output.txt");

String myName = "Homer Simpson";
double mySalary = 35700;

out.print(myName);
out.print(" makes ");
out.print(salary);
out.println(" per year.");
    or
out.println(myName + " makes " + salary +
            " per year.");
```

Oct 5, 2016

Sprenkle - CSCI209

29

## Review: Formatted Output

- printf or format
  - **PrintStream** new functionality since Java 1.5

```
double f1=3.14159, f2=1.45, total=9.43;
// simple formatting...
System.out.printf("%6.5f and %5.2f", f1, f2);
// getting fancy (%n = \n or \r\n)...
System.out.printf("%-6s%5.2f\n", "Tax:", total);
```

Oct 5, 2016

Sprenkle - CSCI209

30

## PrintWriters and Buffering

- PrintWriters are *always* buffered
- Option: **autoflush** mode
  - Causes any writes to be executed directly on target destination
    - In effect, defeats the purpose of buffering
  - Constructor with second parameter set to true

```
// create an autoflushing PrintWriter
PrintWriter out = new PrintWriter("output.txt",
    true);
```

Oct 5, 2016

Sprenkle - CSCI209

31

## Reading Text from a Stream

- There is no `PrintReader` class
- Use a `BufferedReader`
  - Constructor requires a `Reader` object
- Read file, line-by-line using `readLine()`
  - Reads in a line of text and returns it as a `String`
  - Returns null when no more input is available

```
BufferedReader in = new BufferedReader(
    new FileReader("inputfile.txt"));
```

```
String line;
while ((line = in.readLine()) != null) {
    // process the line
}
```

Oct

32



## Reading Text from a Stream

- You can also attach a `BufferedReader` to an `InputStreamReader`:

```
BufferedReader consoleReader= new BufferedReader(  
    new InputStreamReader(System.in));  
BufferedReader webpageReader = new BufferedReader(  
    new InputStreamReader(url.openStream()));
```

Note how easy it is to read  
from different sources

- *Used* to be the best way to read from the console