

## Objectives

- Streams

Oct 3, 2011

Sprenkle - CSCI209

1

## Review

- What are the two types of *exceptions*?
- What are the two main ways to deal with exceptions?
- What are some benefits of exceptions?
- What principle of Java do files break if we're not careful?
- What abstraction do we use to "do" I/O in Java?
- What are some ways to categorize streams?
  - When would you use these streams?

Oct 3, 2011

Sprenkle - CSCI209

2

## STREAMS

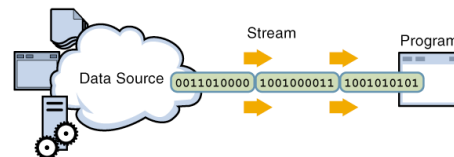
Oct 3, 2011

Sprenkle - CSCI209

3

## 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`

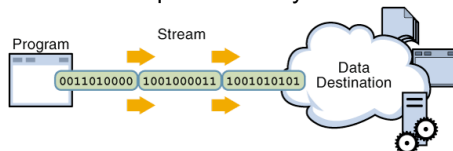
Oct 3, 2011

Sprenkle - CSCI209

4

## 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 3, 2011

Sprenkle - CSCI209

5

## Where We Left Off

- 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);
```

What was the issue we ran into when we tried to read from chicken.data?

Oct 3, 2011

Sprenkle - CSCI209

FileTest2.java 6

## More Powerful Stream Objects

### • DataInputStream

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

### • DataOutputStream

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

Oct 3, 2011

Sprenkle - CSCI209

7

## 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

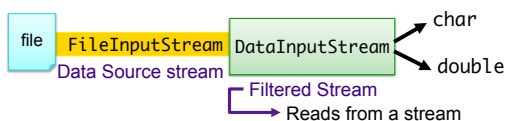
Oct 3, 2011

Sprenkle - CSCI209

8

## 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 3, 2011

Sprenkle - CSCI209

9

## Connecting Streams: From concept to implementation ...

- 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);
double num1 = din.readDouble();
  
```

"wrap" fin in din

Oct 3, 2011

Sprenkle - CSCI209

DataIODemo.java 10

## 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* contain another stream
- Add *functionality* to other stream
  - Automatically buffered IO
  - Automatic compression
  - Automatic encryption
  - Automatic conversion between objects and bytes

Oct 3, 2011

Sprenkle - CSCI209

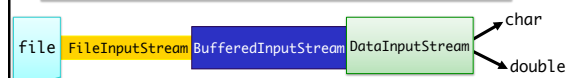
11

## 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 3, 2011

Sprenkle - CSCI209

12

## Connected Streams

Combine different types of streams  
to get functionality you want

- Creating a class for every functionality combo would result in even more classes and a lot of redundant code
- 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

Oct 3, 2011

Sprenkle - CSCI209

13

## TEXT STREAMS

Oct 3, 2011

Sprenkle - CSCI209

14

## Text Streams

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

Oct 3, 2011

Sprenkle - CSCI209

15

## 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 3, 2011

Sprenkle - CSCI209

16

## 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"), "ASCII");
```

Oct 3, 2011

Sprenkle - CSCI209

17

## Convenience Classes

- Reading and writing to text files is common
- 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"));
```

Oct 3, 2011

Sprenkle - CSCI209

18

## PrintWriter

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

Oct 3, 2011

Sprenkle - CSC1209

19

## 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 3, 2011

Sprenkle - CSC1209

20

## 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);
```

- Can make formatted output easy
  - Before 1.5, required java.util.Formatter objects to generate String passed to System.out.println()

Oct 3, 2011

Sprenkle - CSC1209

21

## 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 3, 2011

Sprenkle - CSC1209

22

## Reading Text from a Stream

- There is no PrintReader class
- Use a BufferedReader
  - 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

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

Oct 3, 2011

23

## Reading Text from a Stream

- You can also attach a BufferedReader to an InputStreamReader:

```
BufferedReader in2 = new BufferedReader(
    new InputStreamReader(System.in));
BufferedReader in3 = new BufferedReader(
    new InputStreamReader(url.openStream()));
```

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

Oct 3, 2011

Sprenkle - CSC1209

24

## java.util.Scanner

- New(er) class for handling input
  - Since Java 1.5
- Many constructors
  - Read from file, input stream, string ...
- Many methods
  - readNextXXX (int, long, line)
  - Skipping patterns, matching patterns, etc.

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

Oct 3, 2011

Sprenkle - CSCI209

25

## Using Scanners

- Use *nextXXX()* to read from it...

```
long tempLong;

// create the scanner for the console
Scanner sc = new Scanner(System.in);

// read in an integer and a string
int i = sc.nextInt();
String restOfLine = sc.nextLine();

// read in a bunch of long integers
while (sc.hasNextLong()) {
    tempLong = sc.nextLong();
}
```

Oct 3, 2011

Sprenkle - CSCI209

26

## Using Scanner

```
public static void main(String[] args) {
    // open the Scanner on the console input, System.in
    Scanner scan = new Scanner(System.in);

    System.out.print("Please enter the width of a rectangle: ");
    int width = scan.nextInt();

    System.out.print("Please enter the height of a rectangle: ");
    int length = scan.nextInt();

    System.out
        .println("The area of your square is " + length * width +
            ".");
}
```

ConsoleIODemo.java

Oct 3, 2011

Sprenkle - CSCI209

27

## Scanners

- Breaks its input into tokens using a delimiter pattern, which matches whitespace
 

What is "delimiter pattern"?  
 What is "whitespace"?
- Converts resulting tokens into values of different types using nextXXX()
- Can change token delimiter from default of whitespace
- Assumes numbers are input as decimal
  - Can specify a different radix

Scanner does not have nextString()  
 but does have a nextLine() and a next(). Why not nextString()?

## Scanners & Exceptions

- Scanners do not throw IOExceptions!
  - For a simple console program, main() does not have to deal with or throw IOExceptions
  - Required with BufferedReader/InputStreamReader combination
- Throw InputMismatchException when token doesn't match pattern for expected type
  - e.g., nextLong() called with next token "AAA"
  - RuntimeException (no catching required)

But, that doesn't make for a great user experience ...

Oct 3, 2011

Sprenkle - CSCI209

29

## RuntimeException → Programming Errors

- How can we prevent getting a RuntimeException by improving our programming?
- Look at the API for useful methods for preventing these exceptions
- Update ConsoleIODemo.java

Oct 3, 2011

Sprenkle - CSCI209

30

## Readers and Input Streams

Similar APIs for different data types

### characters

#### • Reader

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

### bytes

#### • InputStream

- `int read()`
- `int read(byte cbuf[])`
- `int read(byte cbuf[], int offset, int length)`

Writers, OutputStreams are similarly parallel

Oct 3, 2011 InvoiceUsingText.java PetSurvey.java

## Summary of Streams

Purpose	Best Choice
Writing primitive types to a binary output stream	
Getting faster access to stream	
Reading bytes from a file	
Reading text from a file	
Writing text to a file	
Getting user input from console	

Oct 3, 2011

Sprengle - CSCI209

32

## Summary of Streams

Purpose	Best Choice
Writing primitive types to a binary output stream	DataOutputStream
Getting faster access to stream	BufferedXXX
Reading bytes from a file	FileInputStream
Reading text from a file	FileReader
Writing text to a file	FileWriter or PrintWriter
Getting user input from console	Scanner

Oct 3, 2011

Sprengle - CSCI209

33

## Midterm Questions?

Oct 3, 2011

Sprengle - CSCI209

34

## TODO

- Midterm Notes
  - See midterm prep guide on class web site
  - Terminology heavy
- Assignment 7: due Friday
  - Modifying Olympic Score generator
    - Read difficulty score from console
    - Read execution scores from a file
      - Filename comes from console

Oct 3, 2011

Sprengle - CSCI209

35