Objectives

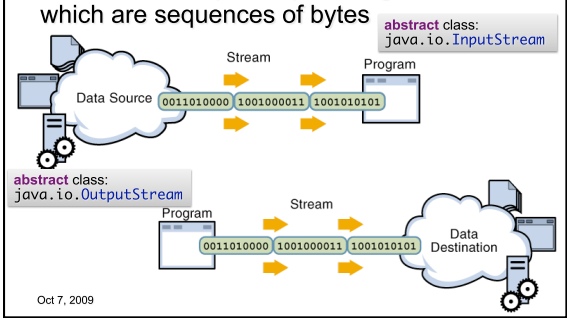
- Streams
- Comparing Java and Python

Review

- What principle of Java do files break if we're not careful?
- Why does Java have so many stream classes?
  - Why doesn't it have more?
- What are some ways to categorize streams?
  - When would you use these streams?

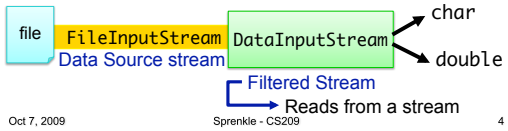
Review: Streams

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



Review: 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
- Combine their functionality by connecting **out** end of `FileInputStream` to **in** end of `DataInputStream`



Review: Data Source vs. Filtered Streams 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

Review: 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);
double num1 = din.readDouble();
```

"wrap" `fin` in `din`

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

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
- Scanners are for Java 1.5 and up only

Scanners & Exceptions

- Do not throw `IOExceptions`!
  - For a simple console program, `main()` does not have to deal with or throw `IOExceptions`
  - Required with `BufferedReader/ InputStreamReader` combination
- Throws `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 …

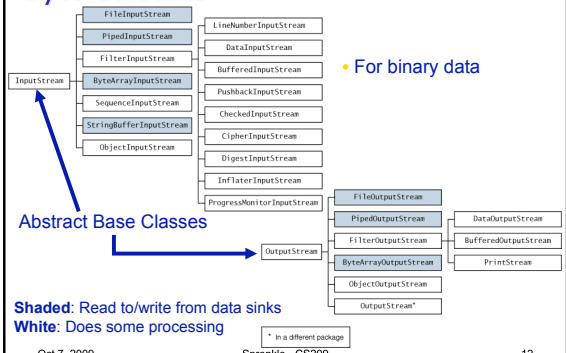
RuntimeException → Programming Errors

- 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`

java.io Classes Overview

- Two types of stream classes
  - Based on datatype: Byte, Text

Byte Streams

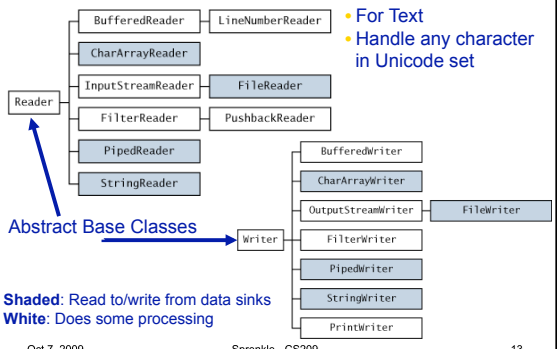


- For binary data

Abstract Base Classes

Shaded: Read to/write from data sinks
White: Does some processing

Character Streams



- For Text
- Handle any character in Unicode set

Shaded: Read to/write from data sinks
White: Does some processing

Readers and Input Streams

Similar APIs for different data types

characters	bytes
Reader	InputStream
int read()	int read()
int read(char cbuf[])	int read(byte cbuf[])
int read(char cbuf[], int offset, int length)	int read(byte cbuf[], int offset, int length)

Writers, OutputStreams are similarly parallel

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	

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

PARSING FILES

Parsing Files

- Use programs to automate tasks
- Often have large amounts of data in files
- Java provides classes to make parsing easier

Use String class

```
String[] split(String regex)
```

\\s means whitespace

```
String test = "this is a test";
String[] result = test.split("\\s");
for (int x=0; x<result.length; x++)
    System.out.println(result[x]);
```

Output:
```
this
is
a
test
```

We could also use " " to split on a space

Regular Expressions in Java

Predefined character classes

Class	Meaning
.	Any character (may or may not match line terminators)
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z_0-9]
\W	A non-word character: [^\w]

Regular expressions are very powerful!

StreamTokenizer

- Tokenize an incoming character stream
- *Table-driven lexical analyzer*
  - Every possible input character has a significance
  - Scanner uses the significance of the current character to decide what to do
- Compiler terminology!
- May be useful to parse files
  - E.g., handling comments like /* */ or //

Midterm Questions?

Midterm Notes

- See midterm prep guide on class web site
- Terminology heavy

Reminder: Assignment 7 due on Monday.