

Objectives

- Representing Files
- Streams
 - Byte Streams
 - Text Streams
 - Connected Streams

Oct 24, 2022

Sprenkle - CSCI209

1

1

Review

1. If your code calls a method that can throw an exception, how can you handle it?
 - (Two options)
2. What are benefits of exceptions?
3. How do we make a block of code execute regardless of whether some code threw an exception or not?

Oct 24, 2022

Sprenkle - CSCI209

2

2

Review: Benefits of Exceptions

- Force error checking/handling
 - Otherwise, won't compile
 - Does not guarantee "good" exception handling
- Ease debugging
 - Stack trace
- Separates error-handling code from "regular" code
 - Error code is in catch blocks at end
 - Descriptive messages with exceptions
- Propagate methods up call stack
 - Let whoever "cares" about error handle it
- Group and differentiate error types

Oct 24, 2022

Sprenkle - CSCI209

3

3

Exceptions Summary

- Try to prevent Runtime Exceptions
- Throw Exceptions in your code for improved error handling/robustness
- If your code calls a method that throws an exception
 - Catch the exception if you can handle it well OR
 - Throw the exception to whoever called you and let them handle it

Oct 24, 2022

Sprenkle - CSCI209

4

4

FILES

Oct 24, 2022

Sprenkle - CSCI209

5

5

java.io.File Class

- Represents a file or directory
- Provides functionality such as
 - Storage of the file on the disk
 - Determine if a particular file exists
 - When file was last modified
 - Rename file
 - Remove/delete file
 - ...

Oct 24, 2022

Sprenkle - CSCI209

6

6

Making a File Object

- Simplest constructor takes full file name (including path)

- If don't supply path, Java assumes current directory (.)

```
File myFile = new File("chicken.data");
```

- Creates a File object representing a file named "chicken.data" in the current directory

- Does not create a file with this name on disk

- Similar to Python: `myFile = open("chicken.data")`

Oct 24, 2022

Sprenkle - CSCI209

7

7

Files, Directories, and Useful Methods

- A File object can represent a file **or** a directory

- Directories are special files in most modern operating systems

- Use `isDirectory()` and/or `isFile()` for type of file File object represents

- Use `exists()` method

- Determines if a file exists on the disk

In Python, functionality are in the `os.path` module

Oct 24, 2022

Sprenkle - CSCI209

8

8

More File Constructors

- String for the path, String for filename

```
File myFile = new File("/csdept/courses/cs209/handouts",
    "chicken.data");
```

- File for directory, String for filename

```
File myDir = new File("/csdept/courses/cs209/handouts");
File myFile = new File(myDir, "chicken.data");
```

Does this “break” any of Java’s principles?

Oct 24, 2022

Sprenkle - CSCI209

9

9

“Break” any of Java’s Principles?

- Principle of Portability
 - Write and Compile Once, Run Anywhere
- Problem: file paths are OS-specific
- `java.io.File.separator`
 - OSX/Linux: /
 - Windows: \
- Takeaways:
 - Use relative paths
 - Use configuration files to set paths

Oct 24, 2022

Sprenkle - CSCI209

10

10

java.io.File Class

- 25+ methods
 - Manipulate files and directories
 - Creating and removing directories
 - Making, renaming, and deleting files
 - Information about file (size, last modified)
 - Creating temporary files
 - ...
- See online API documentation

Oct 24, 2022

Sprenkle - CSCI209

FileTest.java

11

11

A design case study

STREAMS

Oct 24, 2022

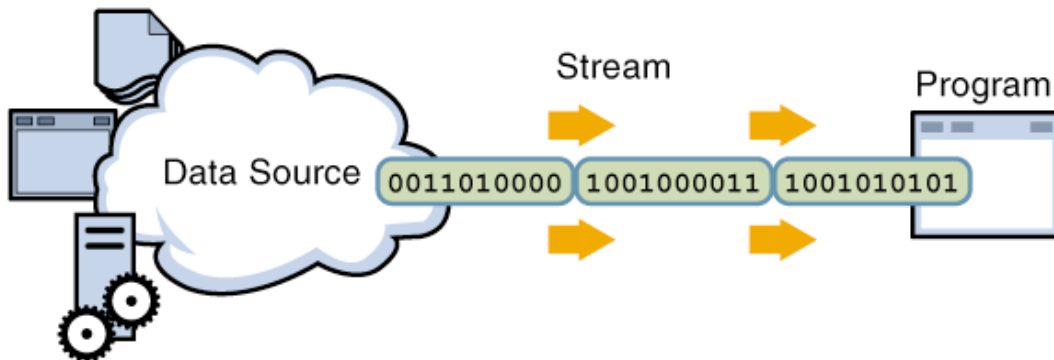
Sprenkle - CSCI209

12

12

Streams

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

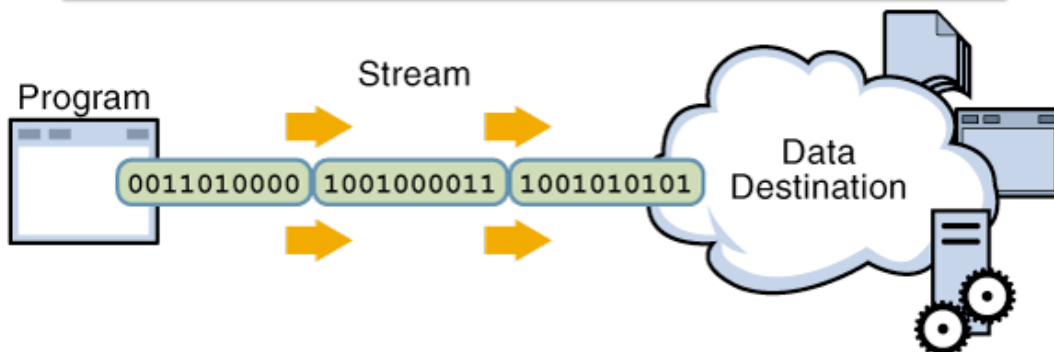


input stream: an object from which we can *read* a *sequence* of bytes
abstract class: `java.io.InputStream`

13

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 24, 2022

Sprenkle - CSCI209

14

14

Java Streams

- MANY (80+) types of Java streams
- In `java.io` package
- Why **stream** abstraction?
 - Information stored in different sources is accessed in essentially the same way
 - Example sources: file, on a web server across the network, string
 - Allows same methods to read or write data, regardless of its source
 - Create an `InputStream` or `OutputStream` of the appropriate type

Oct 24, 2022

Sprenkle - CSCI209

15

15

`java.io` Classes Overview

- Two categories of stream classes, based on datatype: Byte, Text
- Abstract base classes for **binary** data:

InputStream

OutputStream

- Abstract base classes for **text** data:

Reader

Writer

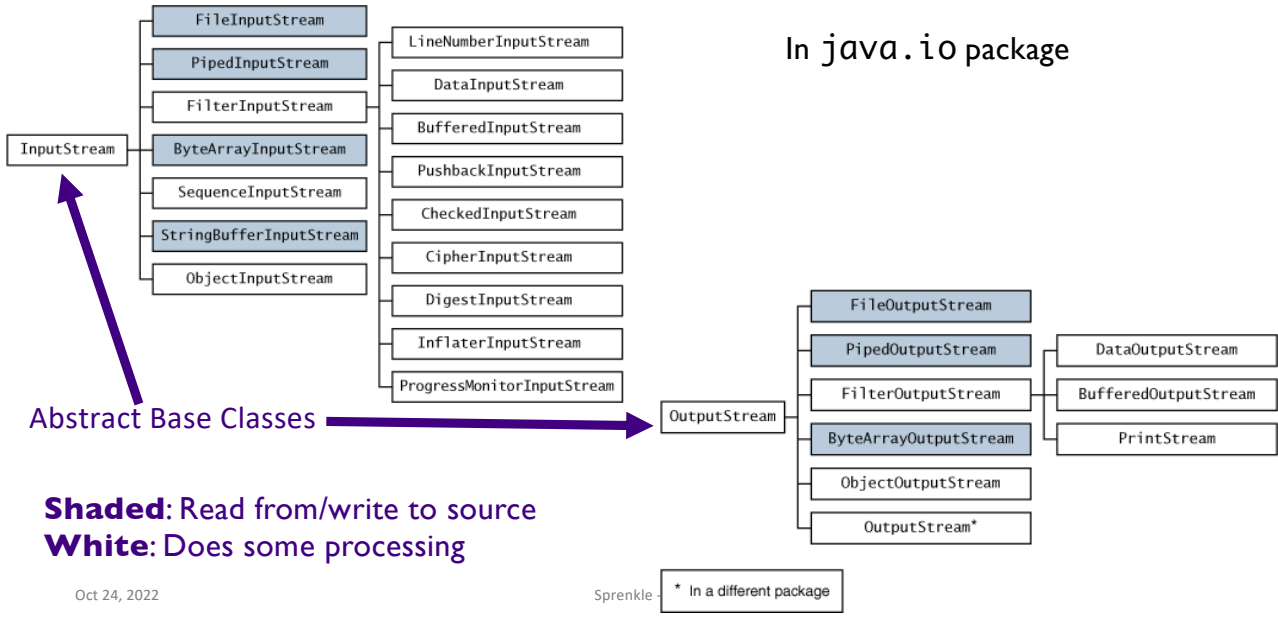
Oct 24, 2022

Sprenkle - CSCI209

16

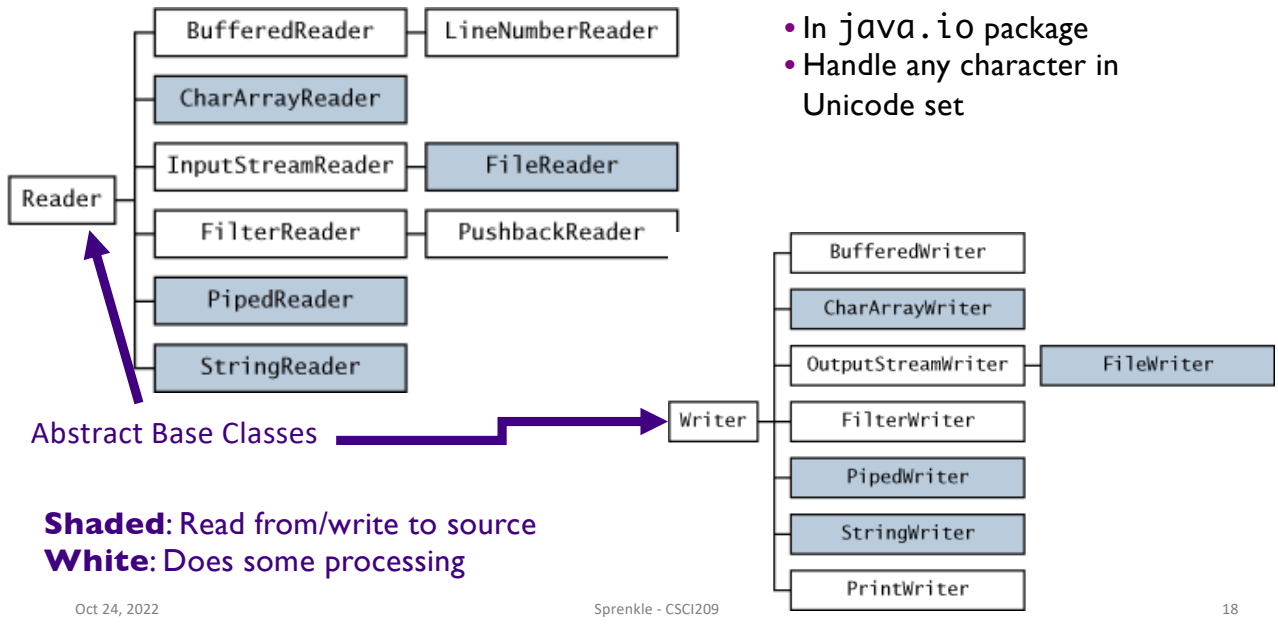
16

Byte Streams: For Binary Data



17

Character Streams: For Text



18

Console I/O: Streams!

- Output:

- `System.out` and `System.err` are **PrintStream** objects

- Input

- `System.in` is an **InputStream** object
- Throws exceptions if errors when reading
 - Handle in `try/catch`

Oct 24, 2022

Sprenkle - CSCI209

19

19

Opening & Closing Streams

- Streams are *automatically opened* when constructed
- Close a stream by calling its `close()` method
 - Close a stream as soon as object is done with it
 - Free up system resources

Oct 24, 2022

Sprenkle - CSCI209

20

20

Reading & Writing Bytes

- Abstract parent class: **InputStream**
 - **abstract int read()**
 - reads one byte from the stream and returns it
 - Concrete child classes override **read()** to provide appropriate functionality
 - e.g., `FileInputStream`'s `read()` reads one byte from a *file*
- Similarly, **OutputStream** class has abstract **write()** to write a byte to the stream

Oct 24, 2022

Sprenkle - CSCI209

21

21

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 24, 2022

Sprenkle - CSCI209

FileTest.java

23

23

More Powerful Stream Objects

- **DataInputStream**

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

- **DataOutputStream**

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

Oct 24, 2022

Sprenkle - CSCI209

24

24

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 24, 2022

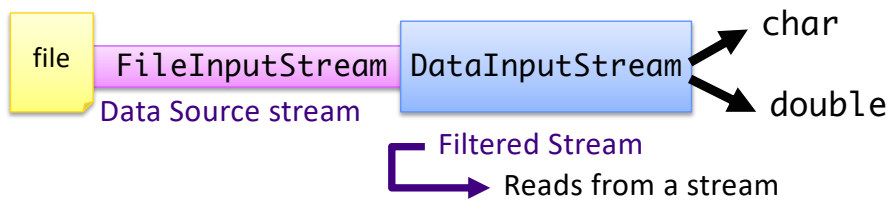
Sprenkle - CSCI209

25

25

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 24, 2022

Sprenkle - CSCI209

26

26

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

Oct 24, 2022

Sprenkle - CSCI209

DataIODemo.java

27

27

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/connects to another stream
- Adds functionality to other stream
 - Automatically buffered IO
 - Automatic compression
 - Automatic encryption
 - Automatic conversion between objects and bytes

Oct 24, 2022

Sprenkle - CSCI209

28

28

Another Filtered Stream: Buffered Streams

• `BufferedInputStream` buffers your input streams

- A pipe in the chain that adds *buffering* → speeds up access

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



Oct 24, 2022

Review: What functionality does each stream add?

29

29

Connected Streams: Similar for Output

- Example: 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

Combine different types of streams
to get functionality you want

Oct 24, 2022

Sprenkle - CSCI209

30

30

TEXT STREAMS

Oct 24, 2022

Sprenkle - CSCI209

31

31

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

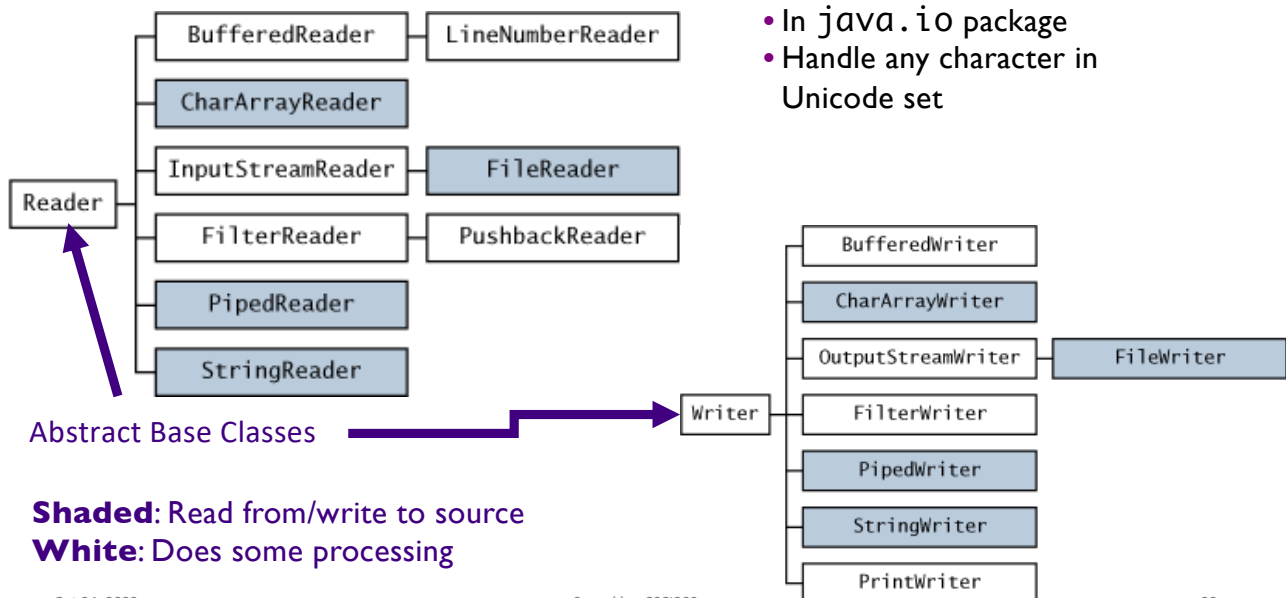
Oct 24, 2022

Sprenkle - CSCI209

32

32

Character Streams: For Text



Oct 24, 2022

Sprenkle - CSCI209

33

33

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 24, 2022

Sprenkle - CSCI209

34


34

Text Streams and Encodings

- Attach an **InputStreamReader** to a **FileInputStream**
 - Assumes file has been encoded in the default encoding of underlying OS
- Can specify a different *encoding* in constructor of **InputStreamReader**

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

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



Oct 24, 2022

35

35

Convenience Classes: Common Combinations

- Reading and writing to text files is common
- **FileReader**
 - Convenience class *combines* a `InputStreamReader` with a `FileInputStream`
- Similar for output to text file

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

is equivalent to

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

Oct 24, 2022

Sprenkle - CSCI209

36

36

PrintWriter

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

Oct 24, 2022

Sprenkle - CSCI209

37

37

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 24, 2022

Sprenkle - CSCI209

38

38

Review: Formatted Output

- printf or format

```
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 24, 2022

Sprenkle - CSCI209

39

39

Reading Text from a Stream: BufferedReader

- There is no PrintReader class
- Constructor requires a Reader object

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

- 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 24, 2022

41

41

Reading Text from a Stream

- You can 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

Oct 24, 2022

Sprenkle - CSCI209

42

42

Scanners

- Scanners do not throw `IOExceptions`!
 - For a simple console program, `main()` does not have to deal with or throw `IOExceptions`
 - Handling those [checked] exceptions is required with `BufferedReader/InputStreamReader` combination
- Throws `InputMismatchException` when token doesn't match pattern for expected type
 - e.g., `nextLong()` called with next token "AAA"
 - No catching required

Meaning it is what type of exception?
How do you prevent errors in Scanner?

Oct 24, 2022

43

43

Scanners

- Scanners do not throw `IOExceptions`!
 - For a simple console program, `main()` does not have to deal with or throw `IOExceptions`
 - Handling those [checked] exceptions is 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)

How do you prevent such errors?

Oct 24, 2022

Sprent

44

44

Preventing Scanner Runtime Exceptions

- Methods to check before reading, e.g. hasNextLong()
- Example code excerpt

```
Scanner sc = new Scanner(System.in);
System.out.print("Enter a long: ");
while( ! sc.hasNextLong() ) {
    System.out.println("Oops, that's not a long.");
    sc.nextLine(); // read in what they (incorrectly) entered
    System.out.print("Enter a long: ");
}
long myLong = sc.nextLong();
System.out.println("You entered " + myLong);
sc.close();
```

Oct 2

45

Summary: Streams

- Abstraction: *streams* – sequences of data
- Two categories of classes based on type of data they handle
 - Bytes: InputStream OutputStream
 - Text: Reader Writer
- Two categories of classes based on their source
 - Data Source (primary source)
 - Filtered (another stream)

Oct 24, 2022

Sprenkle - CSCI209

47

47

Summary: Using Streams

- Can combine streams to get the custom functionality you want
 - Convenience classes for some common combinations
- Development decisions: What do I want this stream to do?
 - What kind of data is it dealing with?
 - What filtering/functionality do I want?
- Select the streams that provide that functionality and connect them (or use convenience class)

Oct 24, 2022

Sprenkle - CSCI209

48

48

Discussion: Stream Design Decisions

- Java's Streams
 - Combine different types of streams to get functionality you want
 - Provide convenience classes for common functionality

What are the tradeoffs for this design decision?

- What would the alternatives be?
- Consider if you maintained the Java libraries
- Consider as a user of those Java libraries

Oct 24, 2022

Sprenkle - CSCI209

49

49

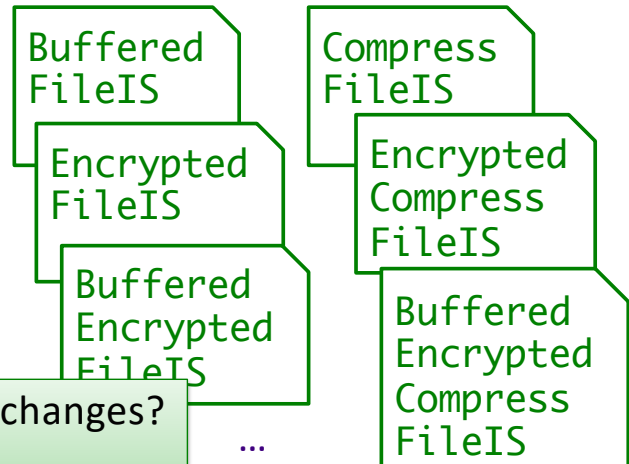
Discussion: Stream Design Decisions

Current Design:



Alternative Design:

Those classes + all the combinations



What happens when functionality changes?
New functionality added?

50

Discussion: Stream Design Decisions

Combine different types of streams
to get functionality you want

- Alternative: Creating a class for every combination would result in even more classes and a lot of redundant code
 - Consider what is required if some functionality must be updated
 - Tricky for user to pull together various streams BUT also would be hard to find the class you want that has the right combination of functionality

Oct 24, 2022

Sprenkle - CSCI209

51

51

Extra Credit Opportunity

Office of the Dean presents 2022 Nobel Symposium

The Nobel Prize in Physics : **Quantum Information Science**

Speaker: Tom Marcais



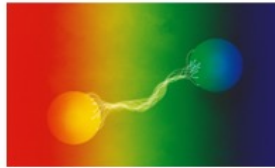
Dr. Nobel Laureate @ Nobel Prize Laureate
Alan Aspect
Prize share: 1/3



Dr. Nobel Laureate @ Nobel Prize Laureate
John F. Clauser
Prize share: 1/3



Dr. Nobel Laureate @ Nobel Prize Laureate
Anton Zeilinger
Prize share: 1/3



"for experiments with entangled photons, establishing the violation of Bell inequalities and pioneering quantum information science."

Wednesday, October 26, 12:15-1:15
Harte Center for Teaching and Learning (Leyburn 128)

Refreshments provided

Oct 24, 2022

Post summary on Canvas discussion forum

52

Assignment 5

- Practicing with Eclipse
- Inheritance, Collections
- Due Friday

Oct 24, 2022

Sprenkle - CSCI209

53

53