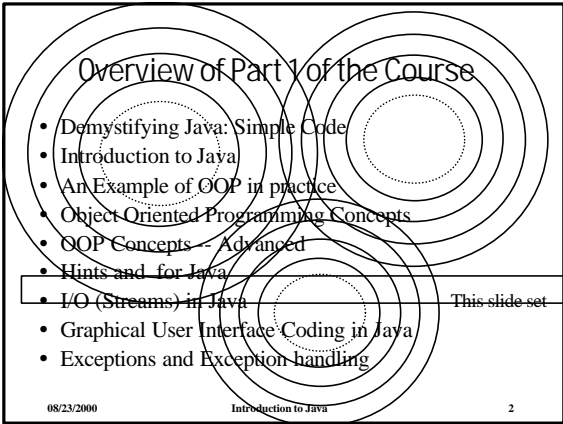


Java I/O Streams

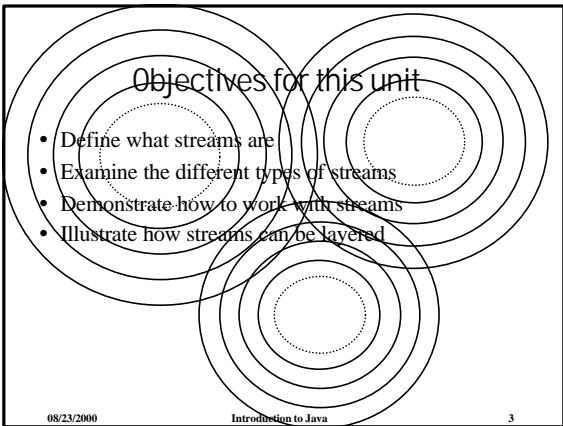
Michael B. Spring
Department of Information Science and Telecommunications
University of Pittsburgh
spring@imap.pitt.edu
http://www.sis.pitt.edu/~spring



Overview of Part 1 of the Course

- Demystifying Java: Simple Code
- Introduction to Java
- An Example of OOP in practice
- Object Oriented Programming Concepts
- OOP Concepts - Advanced
- Hints and for Java
- I/O (Streams) in Java This slide set
- Graphical User Interface Coding in Java
- Exceptions and Exception handling

08/23/2000 Introduction to Java 2



Objectives for this unit

- Define what streams are
- Examine the different types of streams
- Demonstrate how to work with streams
- Illustrate how streams can be layered

08/23/2000 Introduction to Java 3

Concepts

- Java adopts a Unix-like view of input and output that treats all I/O as streams of bytes. The semantics of the bytes are not addressed in the conceptualization of a stream
- In Unix, functionality and semantics are managed by the user or by libraries of functions. In Java, these are provided by classes.
- Just as in Unix, “streams” may be piped through sequences of tools to provide compound functionality.
- With the exception of the `RandomAccessFile` class, all of I/O in Java is unidirectional.

08/23/2000 Introduction to Java 4

The Implementation as Classes

- There are several classes which play a role in I/O in Java.
 - Files are objects of Class `File`
 - Four abstract classes organize the various I/O streams
 - `InputStream`
 - `OutputStream`
 - `Reader` (an input stream optimized for Unicode text)
 - `Writer` (an output stream optimized for Unicode text)
- There are three predefined streams that may be used without any construction
 - `System.in` - for reading from the keyboard
 - `System.out` - for writing to the screen
 - `System.err` - for writing error messages to the screen

08/23/2000 Introduction to Java 5

Subclasses of Streams

- A subclass of stream is either a sink, a source, or a filter
 - Sources include `FileInputStream`, `PipedInputStream`, etc.
 - Sinks include `FileOutputStream`, `PipedOutputStream`, etc.
 - Filters include `BufferedInputStream`, `DataInputStream`, `BufferedOutputStream`, `DataOutputStream`, etc.
- These filters, or processing streams provide additional functionality on “top” of the source and sink streams.
- It should be noted that there is a `RandomAccessFile` class that allows both reading and writing to a file.

08/23/2000 Introduction to Java 6

Data sources and sinks

- As may be deduced from the previous slides, there are different kinds of I/O streams
 - `FileInputStream` - for reading from files
 - `FileOutputStream` - for writing to files
 - `PipedInputStream` - for reading from a thread
 - `PipedOutputStream` - for writing to a thread
- There are subtle additional types for the `Readers` and `Writers` which are specialized to deal with Unicode encoded text.

08/23/2000 Introduction to Java 7

Filters and Processing streams

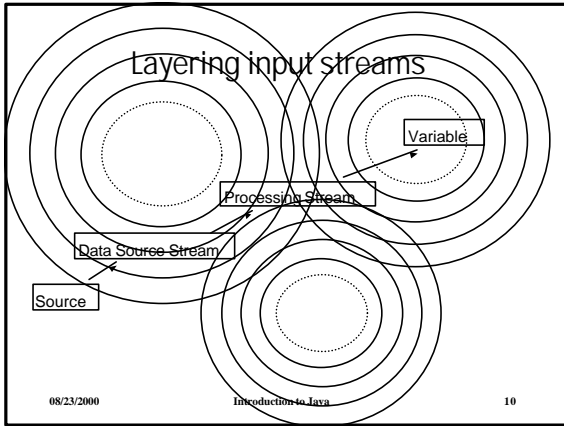
- `DataInputStream` - processes the primitive Java types from an input stream allowing them to be assigned to variables.
- `DataOutputStream` - processes the primitive Java types to an output stream
- `BufferedInputStream` - optimizes I/O from a stream by using a buffer
- `BufferedOutputStream` - optimizes I/O to a stream by using a buffer
- `PrintStream` - converts various primitive data types to text before placing them in the stream
- `PushbackInputStream` - allows look ahead input such as might be required for tokenizing or parsing

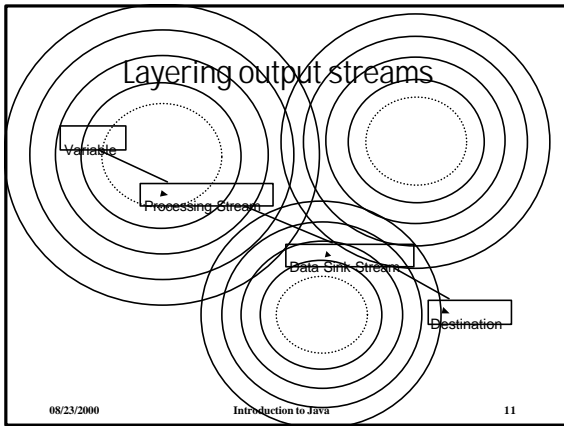
08/23/2000 Introduction to Java 8

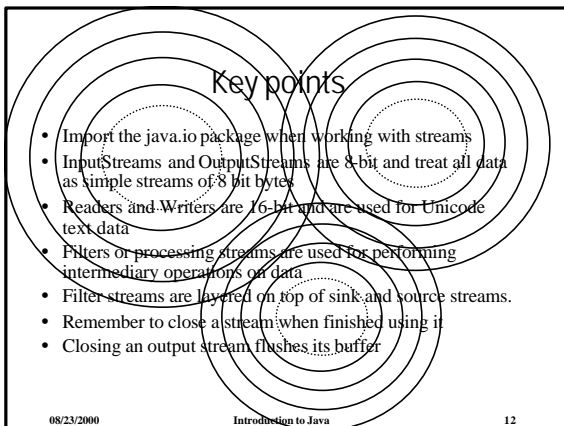
The basic procedures for accessing a stream

- Open the stream by calling its constructor (Java streams do not use an `open()` method)
- Read or write to the stream using the methods associated with that particular stream (when layering streams, use the methods of the outermost stream)
- Close the stream with the `close()` method

08/23/2000 Introduction to Java 9







Streams and Exceptions

- Java has a method of handling errors that encloses code subject to failures or exceptions in “try/catch” blocks.
- IO tends to be subject to a large number of exceptions (file not found, file locked, IO errors, etc.)
- Care needs to be taken in writing Java code that involves streams to anticipate and handle the various exceptions that might occur.

08/23/2000 Introduction to Java 13

Example

```
String name = new String("John Smith");
int age = 23;
double gpa = 3.756;
try{
    File F1 = new File("binary.dat");
    File F2 = new File("text.txt");
    FileOutputStream OFS1 = new FileOutputStream(F1);
    FileOutputStream OFS2 = new FileOutputStream(F2);
    DataOutputStream ODS = new DataOutputStream(OFS1);
    PrintStream OPS = new PrintStream(OFS2);
    ODS.writeChars(name);
    ODS.writeInt(age);
    ODS.writeDouble(gpa);
    OPS.print(name);
    OPS.print(age);
    OPS.print(gpa);}
}
```

08/23/2000 Introduction to Java 14

Results of the Program

- Both streams started with the following data:


```
String name = new String("John Smith");
int age = 23;
double gpa = 3.756;
```
- The DataOutputStream took the information provided and produced a file with the following contents:
 - ?J?o?h?n? ?S?2?3?7?5?6? @ ?2?3?5?
- The PrintStream took the same information and produced a file with the following contents:
 - John Smith233.756
- If read back in with the appropriate filters the original data correctly formed would be returned.

08/23/2000 Introduction to Java 15
