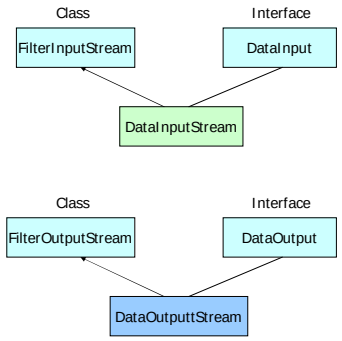
Streams and Input/Output Files Part 3

Handling Primitive Data Types

- The basic input and output streams provide read/write methods that can be used for reading/writing bytes or characters.
- To read/write the primitive data types such as integers and doubles, we can use filter classes as wrappers on the existing I/O streams to filter data to the original stream.
- The two filter classes supported for creating "data streams" for primitive data types are:
  - DataInputStream
  - DataOutputStream

Hierarchy of Data Stream Classes



Data Input Stream Creation

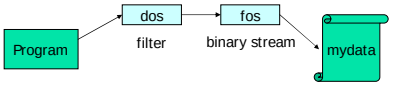
- Create Input File Stream:
  - FileInputStream fis = new FileInputStream("InFile");
- Create Input Data Stream:
  - DataInputStream dis = new DataInputStream(fis);
- The above statements wrap data input stream (dis) on file input stream (fis) and use it as a "filter".
- Methods Supported:
  - readBoolean(), readByte(), readChar(), readShort(), readInt(), readLong(), readFloat(), readDouble()
- They read data stored in file in binary format.

Data Output Stream Creation

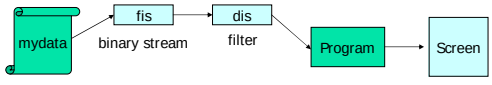
- Create Output File Stream:
  - FileOutputStream fos = new FileOutputStream("OutFile");
- Create Output Data Stream:
  - DataOutputStream dos = new DataOutputStream(fos);
- The above statements wrap data output stream (dos) on file output stream (fos) and use it as a "filter".
- Methods Supported:
  - writeBoolean(), writeByte(), writeChar(), writeShort(), writeInt(), writeLong(), writeFloat(), writeDouble()
- They write data to file in binary format.
  - How many bytes are written to file when for statements:
    - writeInt(120), writeInt(10120)

Data Streams Flow via Filter

- Write primitive data to the file using a filter.



- Read primitive data from the file using a filter.



Writing and Reading Primitive Data

```
import java.io.*;
public class ReadWriteFilter {
    public static void main(String args[]) throws IOException {
      // write primitive data in binary format to the "mydata" file
      FileOutputStream fos = new FileOutputStream("mydata");
      DataOutputStream dos = new DataOutputStream(fos);
      dos.writeInt(120);
      dos.writeDouble(375.50);
      dos.writeInt('A'+1);
      dos.writeBoolean(true);
      dos.writeChar('X');
      dos.close();
      fos.close();

      // read primitive data in binary format from the "mydata" file
      FileInputStream fis = new FileInputStream("mydata");
      DataInputStream dis = new DataInputStream(fis);
      System.out.println(dis.readInt());
      System.out.println(dis.readDouble());
      System.out.println(dis.readInt());
      System.out.println(dis.readBoolean());
      System.out.println(dis.readChar());
      dis.close();
      fis.close();
    }
}
```


Program Run and Output

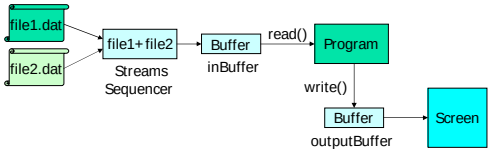
- C:\254\examples>java ReadWriteFilter
  - 120
  - 375.5
  - 66
  - true
  - X
- Display content of "mydata" file (in binary format):
  - C:\254\examples>type mydata
    - x@wx B X
- What is the size of "mydata" file (in bytes) ?
  - Size of int+double+int+boolean+char

Concatenating and Buffering Streams

- Two or more input streams can be combined into a single input stream. This process is known as logical *concatenation* of streams and is achieved using the *SequenceInputStream* class.
- A *SequenceInputStream* starts out with an ordered collection of input streams and reads from the first one until end of file is reached, whereupon it reads from the second one, and so on, until end of file is reached on the last of the contained input streams.

Sequencing and Buffering of Streams

- Buffered streams sit between the program and data source/destination and functions like a filter or support efficient I/O. Buffered can be created using BufferedInputStream and BufferedOutputStream classes.



Example Program

```
import java.io.*;
public class CombineStreams {
   public static void main(String args[]) throws IOException {
      // declare file streams
      FileInputStream file1 = new FileInputStream("file1.dat");
      FileInputStream file2 = new FileInputStream("file2.dat");
      // declare file3 to store combined streams
      SequenceInputStream file3 = null;
      // concatenate file1 and file2 streams into file3
      file3 = new SequenceInputStream(file1, file2);
      BufferedInputStream inBuffer = new BufferedInputStream(file3);
      BufferedOutputStream outBuffer = new BufferedOutputStream(System.out);
      // read and write combined streams until the end of buffers
      int ch;
      while((ch = inBuffer.read()) != -1 )
        outBuffer.write(ch);
      outBuffer.flush(); // check out the output by removing this line
      System.out.println("\nHello, This output is generated by CombineFiles.java program");
      inBuffer.close();
      outBuffer.close();
      file1.close();
      file2.close();
      file3.close();
   }
}
```


Contents of Input Files

- The file1.dat contains:
  - Hello,
  - I am C++, born in AT&T.
- The file2.dat contains:
  - Hello,
  - I am Java, born in Sun Microsystems!

Output

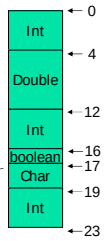
- C:\254\examples> java CombineStreams
  - Hello,
  - I am C++, born in AT&T.
  - Hello,
  - I am Java, born in Sun Microsystems!
  - Hello, This output is generated by CombineFiles.java program
- If the statement **outBuffer.flush()** is removed, the output will be:
  - Hello, This output is generated by CombineFiles.java program
  - Hello,
  - I am C++, born in AT&T.
  - Hello,
  - I am Java, born in Sun Microsystems!

Random Access Files

- So for we have discussed sequential files that are either used for storing data and accessed (read/write) them in sequence.
- In most real world applications, it is necessary to access data in non-sequential order (e.g, banking system) and append new data or update existing data.
- Java IO package supports RandomAccessFile class that allow us to create files that can be used for reading and/or writing with random access.
- The file can be open either in read mode ("r") or read-write mode ("rw") as follows:
  - myFileHandleName = new RandomAccessFile ("filename", "mode");
- The file pointer can be set to any to any location (measured in bytes) using seek() method prior to reading or writing.

Random Access Example

```
import java.io.*;
public class RandomAccess {
   public static void main(String args[]) throws IOException {
      // write primitive data in binary format to the "mydata" file
      RandomAccessFile myfile = new RandomAccessFile("rand.dat", "rw");
      myfile.writeInt(120);
      myfile.writeDouble(375.50);
      myfile.writeInt('A'+1);
      myfile.writeBoolean(true);
      myfile.writeChar('X');
      // set pointer to the beginning of file and read next two items
      myfile.seek(0);
      System.out.println(myfile.readInt());
      System.out.println(myfile.readDouble());
     // set pointer to the 4th item and read it
      myfile.seek(16);
      System.out.println(myfile.readBoolean());
      // Go to the end and "append" an integer 2003
      myfile.seek(myfile.length());
      myfile.writeInt(2003);
      // read 5th and 6th items
      myfile.seek(17);
      System.out.println(myfile.readChar());
      System.out.println(myfile.readInt());
      System.out.println("File length: "+myfile.length());
      myfile.close();
   }
}
```



Execution and Output

- C:\254\examples> java RandomAccess
  - 120
  - 375.5
  - true
  - X
  - 2003
  - File length: 23

Streams and Interactive I/O

- Real world applications are designed to support interactive and/or batch I/O operations.
- Interactive programs allow users to interact with them during their execution through I/O devices such as keyboard, mouse, display devices (text/graphical interface), media devices (microphones/speakers), etc..
  - Java provides rich functionality for developing interactive programs.
- Batch programs are those that are designed to read input data from files and produce outputs through files.

Standard I/O

- The System class contains three I/O objects (static)
  - System.in – instance of InputStream
  - System.out – instance of PrintStream
  - System.err – instance of PrintStream
- To perform keyboard input, we need use functionalities of DataInputStream and StringTokenizer classes.

Reading Integer from Standard Input

- Create buffered reader for standard input by wrapping System.in object:
  - BufferedReader dis = new BufferedReader(new InputStreamReader(System.in));
- Read a line of text from the console
  - String str = dis.readLine();
- Create Tokenens
  - StringTokenizer st;
  - st = new StringTokenizer(str);
- Convert String Token into basic integer:
  - int stdID = Integer.parseInt(st.nextToken());

Interactive IO Example

```
import java.io.*;
import java.util.*;
public class StudentRecord {
   public static void main(String args[]) throws IOException {
      // Create buffered reader for standard input
      BufferedReader dis = new BufferedReader(new InputStreamReader(System.in));
      StringTokenizer st;
      // reading data from console
      System.out.print("Enter Student ID: ");
      st = new StringTokenizer(dis.readLine());
      int stdID = Integer.parseInt(st.nextToken());
      System.out.print("Enter Student Name: ");
      String stdName = dis.readLine();
      System.out.print("Enter Student Marks: ");
      st = new StringTokenizer(dis.readLine());
      int stdMarks = Integer.parseInt(st.nextToken());
      // write to console
      System.out.println("Student details are:");
      System.out.println("ID: "+ stdID);
      System.out.println("Name: "+ stdName);
      System.out.println("Marks: "+ stdMarks);
   }
}
```


Run and Output

- C:\254\examples> java StudentRecord
  - Enter Student ID: 2002010
  - Enter Student Name: Mary Baker
  - Enter Student Marks: 85
  - Student details are:
  - ID: 2002010
  - Name: Mary Baker
  - Marks: 85

Summary

- All Java I/O classes are designed to operate with Exceptions.
- User Exceptions and your own handler with files to manger runtime errors.
- Subclasses FileReader / FileWriter support characters-based File I/O.
- FileInputStream and FileOutputStream classes support bytes-based File I/O.
- Buffered read/write operations support efficient I/O.
- DataInputStream and DataOutputStream classes support rich I/O functionality.
- RandomAccessFile supports access to any data items in files in any order.