Lecture - 16

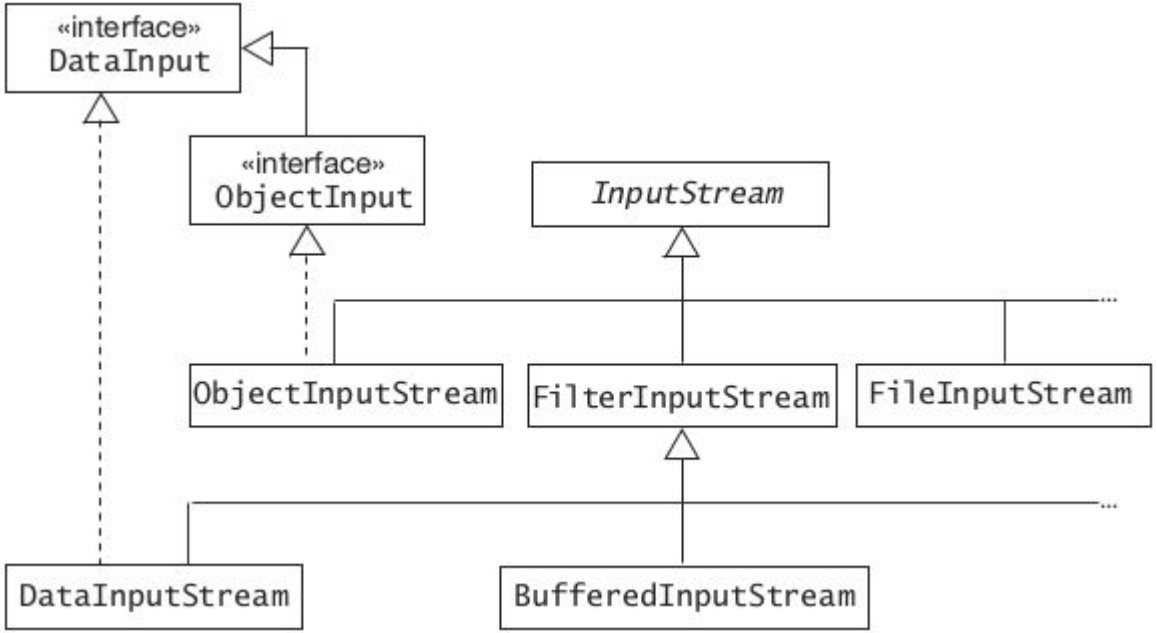
Byte Streams

Arpit Rana

7th April 2022

Partial Byte Stream Inheritance Hierarchies

The abstract classes **`InputStream`** and **`OutputStream`** are the root of the inheritance hierarchies for handling the reading and writing of bytes.



The `InputStream` Class

The subclasses, implementing different kinds of input streams, override the following methods from the **`InputStream`** class to customize the reading of bytes:

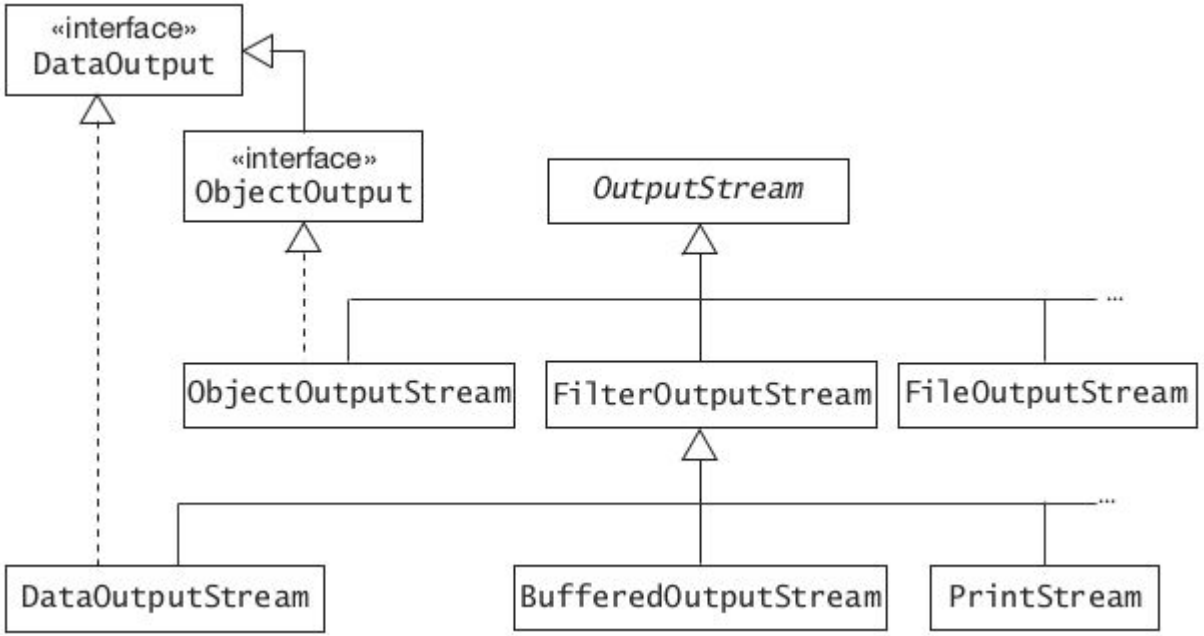
The `InputStream` class:

```
int read() throws IOException
int read(byte[] b) throws IOException
int read(byte[] b, int off, int len) throws IOException
```

- Note that the first `read()` method reads a byte, but returns an `int` value.
- The byte read resides in the eight least significant bits of the `int` value, while the remaining bits in the `int` value are zeroed out.
- The `read()` methods return the value –1 when the end of the stream is reached.

Partial Byte Stream Inheritance Hierarchies

The abstract classes **`InputStream`** and **`OutputStream`** are the root of the inheritance hierarchies for handling the reading and writing of bytes.



The `OutputStream` Class

The subclasses, implementing different kinds of input streams, override the following methods from the **`OutputStream`** class to customize the writing of bytes:

The `OutputStream` class:

```
void write(int b) throws IOException
void write(byte[] b) throws IOException
void write(byte[] b, int off, int len) throws IOException
```

- The first `write()` method takes an `int` as argument, but truncates it down to the eight least significant bits before writing it out as a byte.

```
void close() throws IOException
void flush() throws IOException
```

Only for `OutputStream`

- A stream should be closed when no longer needed, to free system resources. Closing an output stream automatically flushes the stream.

Input and Output Streams

Selected Input Streams

`FileInputStream`	Data is read as bytes from a file. The file acting as the input stream can be specified by a `File` object, a `FileDescriptor` or a `String` file name.
`FilterInputStream`	Superclass of all input stream filters. An input filter must be chained to an underlying input stream.
`DataInputStream`	A filter that allows the binary representation of Java primitive values to be read from an underlying input stream. The underlying input stream must be specified.
`ObjectInputStream`	Allows binary representations of Java objects and Java primitive values to be read from a specified input stream.

Input and Output Streams

Selected Output Streams

`FileOutputStream`	Data is written as bytes to a file. The file acting as the output stream can be specified by a `File` object, a `FileDescriptor` or a `String` file name.
`FilterOutputStream`	Superclass of all output stream filters. An output filter must be chained to an underlying output stream.
`DataOutputStream`	A filter that allows the binary representation of Java primitive values to be written to an underlying output stream. The underlying output stream must be specified.
`ObjectOutputStream`	Allows the binary representation of Java objects and Java primitive values to be written to a specified underlying output stream.

File Streams

The classes `FileInputStream` and `FileOutputStream` define byte input and output streams that are connected to files.

- Data can only be read or written as a sequence of bytes.
- An input stream for reading bytes can be created using the following constructors:

```
FileInputStream(String name) throws FileNotFoundException
FileInputStream(File file) throws FileNotFoundException
FileInputStream(FileDescriptor fdObj)
```

- The file can be specified by its name, through a `File` object, or using a `FileDescriptor` object.
- If the file does not exist, a `FileNotFoundException` is thrown. If it exists, it is set to be read from the beginning.
- A `SecurityException` is thrown if the file does not have read access.

File Streams

An output stream for writing bytes can be created using the following constructors:

```
FileOutputStream(String name) throws FileNotFoundException
FileOutputStream(String name, boolean append) throws FileNotFoundException
FileOutputStream(File file) throws IOException
FileOutputStream(FileDescriptor fdObj)
```

- The file can be specified by its name, through a `File` object, or using a `FileDescriptor` object.
- If the file does not exist, it is created. If it exists, its contents are reset, unless the appropriate constructor is used to indicate that output should be appended to the file.
- A `SecurityException` is thrown if the file does not have write access or it cannot be created

File Streams

```
/* Copy a file.
   Command syntax: java CopyFile <from_file> <to_file>
 */
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

class CopyFile {
  public static void main(String[] args) {
    FileInputStream fromFile;
    FileOutputStream toFile;

    // Assign the files
    try {
      fromFile = new FileInputStream(args[0]);       // (1)
      toFile = new FileOutputStream(args[1]);        // (2)
    } catch(FileNotFoundException e) {
      System.err.println("File could not be copied: " + e);
      return;
    } catch(ArrayIndexOutOfBoundsException e) {
      System.err.println("Usage: CopyFile <from_file> <to_file>");
      return;
    }
```

File Streams

```
    // Copy bytes
    try {                                                    // (3)
      while (true) {
        int i = fromFile.read();
        if(i == -1) break;              // check end of file
        toFile.write(i);
      }
    } catch(IOException e) {
      System.err.println("Error reading/writing.");
    }

    // Close the files
    try {                                                    // (4)
      fromFile.close();
      toFile.close();
    } catch(IOException e) {
      System.err.println("Error closing file.");
    }
  }
}
```

Filter Streams

A filter is a high-level stream that provides additional functionality to an underlying stream to which it is chained.

- The data from the underlying stream is manipulated in some way by the filter.
- The `FilterInputStream` and `FilterOutputStream` classes, together with their subclasses, define input and output filter streams.
- The subclasses `BufferedInputStream` and `BufferedOutputStream` implement filters that buffer input from and output to the underlying stream, respectively.
- The subclasses `DataInputStream` and `DataOutputStream` implement filters that allow binary representation of Java primitive values to be read and written, respectively, to and from an underlying stream.

Reading and Writing Binary Values

The **`java.io`** package contains the two interfaces `DataInput` and `DataOutput`.

- The filter streams `DataOutputStream` and `DataInputStream` implement these interfaces respectively.
- These filter streams allow reading and writing of binary representations of Java primitive values (`boolean`, `char`, `byte`, `short`, `int`, `long`, `float`, `double`).
- The following constructors can be used to set up filters for reading and writing Java primitive values, respectively, from an underlying stream:

```
DataInputStream(InputStream in)
DataOutputStream(OutputStream out)
```

Reading and Writing Binary Values

Type	Methods in the *DataInput* Interface	Methods in the *DataOutput* interface
boolean	readBoolean()	writeBoolean(boolean b)
char	readChar()	writeChar(int c)
byte	readByte()	writeByte(int b)
short	readShort()	writeShort(int s)
int	readInt()	writeInt(int i)
long	readLong()	writeLong(long l)
float	readFloat()	writeFloat(float f)
double	readDouble()	writeDouble(double d)
String	readLine()	writeChars(String str)
String	readUTF()	writeUTF(String str)

- The methods `readUTF()` and `writeUTF()` also read and write characters, but use the UTF-8 character encoding.

Writing Binary Values to a File

To write the binary representation of Java primitive values to a binary file, the following procedure can be used,

1. Create a FileOutputStream:

   ```
   FileOutputStream outputFile = new FileOutputStream("primitives.data");
   ```

2. Create a DataOutputStream which is chained to the FileOutputStream:

   ```
   DataOutputStream outputStream = new DataOutputStream(outputFile);
   ```

3. Write Java primitive values using relevant writeX() methods:

   ```
   outputStream.writeBoolean(true);
   outputStream.writeChar('A');                    // int written as Unicode char
   ```

4. Close the filter stream, which also closes the underlying stream:

   ```
   outputStream.close();
   ```

Reading Binary Values From a File

To read the binary representation of Java primitive values from a binary file the following procedure can be used.

1. Create a FileInputStream:

   ```
   FileInputStream inputFile = new FileInputStream("primitives.data");
   ```

2. Create a DataInputStream which is chained to the FileInputStream:

   ```
   DataInputStream inputStream = new DataInputStream(inputFile);
   ```

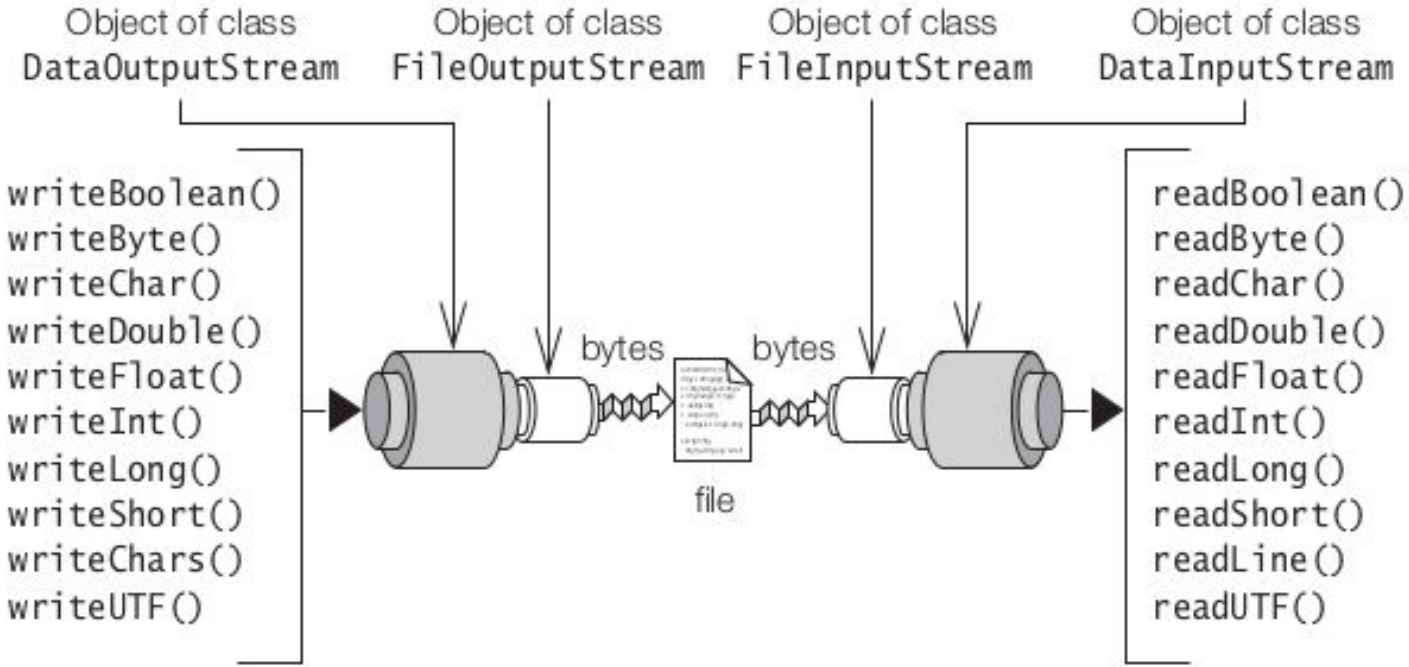
3. Read the (exact number of) Java primitive values in the same order they were written out, using relevant readX() methods:

   ```
   boolean v = inputStream.readBoolean();
   char c = inputStream.readChar();
   ```

4. Close the filter stream, which also closes the underlying stream:

   ```
   inputStream.close();
   ```

Reading & Writing Binary Values From/to a File



```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class BinaryValuesIO {
  public static void main(String[] args) throws IOException {
    // Create a FileOutputStream.
    FileOutputStream outputFile = new FileOutputStream("primitives.data");

    // Create a DataOutputStream which is chained to the FileOutputStream.
    DataOutputStream outputStream = new DataOutputStream(outputFile);

    // Write Java primitive values in binary representation:
    outputStream.writeBoolean(true);
    outputStream.writeChar('A');                // int written as Unicode char
    outputStream.writeByte(Byte.MAX_VALUE);     // int written as 8-bits byte
    outputStream.writeShort(Short.MIN_VALUE);   // int written as 16-bits short
    outputStream.writeInt(Integer.MAX_VALUE);
    outputStream.writeLong(Long.MIN_VALUE);
    outputStream.writeFloat(Float.MAX_VALUE);
    outputStream.writeDouble(Math.PI);

    // Close the output stream, which also closes the underlying stream.
    outputStream.flush();
    outputStream.close();
```

```
// Create a FileInputStream.
FileInputStream inputFile = new FileInputStream("primitives.data");

// Create a DataInputStream which is chained to the FileInputStream.
DataInputStream inputStream = new DataInputStream(inputFile);

// Read the binary representation of Java primitive values
// in the same order they were written out:
boolean v = inputStream.readBoolean();
char    c = inputStream.readChar();
byte    b = inputStream.readByte();
short   s = inputStream.readShort();
int     i = inputStream.readInt();
long    l = inputStream.readLong();
float   f = inputStream.readFloat();
double  d = inputStream.readDouble();

// Check for end of stream:
try {
  int value = inputStream.readByte();
  System.out.println("More input: " + value);
} catch (EOFException eofe) {
  System.out.println("End of stream");
} finally {
  // Close the input stream, which also closes the underlying stream.
  inputStream.close();
}
```

```
        // Write the values read to the standard input stream:
        System.out.println("Values read:");
        System.out.println(v);
        System.out.println(c);
        System.out.println(b);
        System.out.println(s);
        System.out.println(i);
        System.out.println(l);
        System.out.println(f);
        System.out.println(d);
    }
}
```

Output from the program:

```
End of stream
Values read:
true
A
127
-32768
2147483647
-9223372036854775808
3.4028235E38
3.141592653589793
```

Next lecture -
Character Streams