Input/output streams

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming Concepts using Java

Week 9

Input and output streams

- Input: read a sequence of bytes from some source
  - A file, an internet connection, memory ...
- Output: write a sequence of bytes to some source
  - A file, an internet connection, memory ...

Input and output streams

- Input: read a sequence of bytes from some source
  - A file, an internet connection, memory ...
- Output: write a sequence of bytes to some source
  - A file, an internet connection, memory ...
- Java refers to these as input and output streams
  - Not the same as stream objects in class `Stream`

Input and output streams

- Input: read a sequence of bytes from some source
  - A file, an internet connection, memory ...
- Output: write a sequence of bytes to some source
  - A file, an internet connection, memory ...
- Java refers to these as input and output streams
  - Not the same as stream objects in class `Stream`
- Input and output values could be of different types
  - Ultimately, input and output are raw uninterpreted bytes of data
  - Interpret as text — different Unicode encodings
  - Or as binary data — integers, floats, doubles, ...

Input and output streams

- Input: read a sequence of bytes from some source
  - A file, an internet connection, memory ...
- Output: write a sequence of bytes to some source
  - A file, an internet connection, memory ...
- Java refers to these as input and output streams
  - Not the same as stream objects in class `Stream`
- Input and output values could be of different types
  - Ultimately, input and output are raw uninterpreted bytes of data
  - Interpret as text — different Unicode encodings
  - Or as binary data — integers, floats, doubles, ...
- Use a pipeline of input/output stream transformers
  - Read raw bytes from a file, pass to a stream that reads text
  - Generate binary data, pass to a stream that writes raw bytes to a file

Reading and writing raw bytes

- Classes `InputStream` and `OutputStream`

Reading and writing raw bytes

- Classes `InputStream` and `OutputStream`
- Read one or more bytes — abstract methods are implemented by subclasses of `InputStream`

```
abstract int read();
int read(byte[] b);
byte[] readAllBytes();
// ... and more
```

Reading and writing raw bytes

- Classes `InputStream` and `OutputStream`
- Read one or more bytes — abstract methods are implemented by subclasses of `InputStream`
- Check availability before reading

```
abstract int read();
int read(byte[] b);
byte[] readAllBytes();
// ... and more
```

```
InputStream in = ....
int bytesAvailable = in.available();
if (bytesAvailable > 0)
{
   var data = new byte[bytesAvailable];
   in.read(data);
}
```

Reading and writing raw bytes

- Classes `InputStream` and `OutputStream`
- Read one or more bytes — abstract methods are implemented by subclasses of `InputStream`
- Check availability before reading
- Write bytes to output

```
abstract void write(int b);
void write(byte[] b);
// ... and more

OutputStream out = ...
byte[] values = ...;
out.write(values);
```

Reading and writing raw bytes

- Classes `InputStream` and `OutputStream`
- Read one or more bytes — abstract methods are implemented by subclasses of `InputStream`
- Check availability before reading
- Write bytes to output
- Close a stream when done — release resources

```
abstract void write(int b);
void write(byte[] b);
// ... and more

OutputStream out = ...
byte[] values = ...;
out.write(values);


in.close();
```

Reading and writing raw bytes

- Classes `InputStream` and `OutputStream`
- Read one or more bytes — abstract methods are implemented by subclasses of `InputStream`
- Check availability before reading
- Write bytes to output
- Close a stream when done — release resources
- Flush an output stream — output is buffered

```
abstract void write(int b);
void write(byte[] b);
// ... and more

OutputStream out = ...
byte[] values = ...;
out.write(values);


in.close();


out.flush();
```

Connecting a stream to an external source

- Input and output streams ultimately connect to external resources
  - A file, an internet connection, memory ...
  - We limit ourselves to files

Connecting a stream to an external source

- Input and output streams ultimately connect to external resources
  - A file, an internet connection, memory ...
  - We limit ourselves to files
- Create an input stream attached to a file

```
var in = new FileInputStream("input.class");
```

Connecting a stream to an external source

- Input and output streams ultimately connect to external resources
  - A file, an internet connection, memory ...
  - We limit ourselves to files
- Create an input stream attached to a file
- Create an output stream attached to a file

```
var in = new FileInputStream("input.class");
```

```
var out = new FileOutputStream("output.bin");
```

Connecting a stream to an external source

- Input and output streams ultimately connect to external resources
  - A file, an internet connection, memory ...
  - We limit ourselves to files
- Create an input stream attached to a file
- Create an output stream attached to a file
- Overwrite or append?
  - Pass a boolean second argument to the constructor

```
var in = new FileInputStream("input.class");

var out = new FileOutputStream("output.bin");

var out = new
   FileOutputStream("newoutput.bin",false);
   // Overwrite

var out = new
   FileOutputStream("sameoutput.bin",true);
   // Append
```

Reading and writing text

- Recall `Scanner` class
  - Can apply to any input stream

```
var fin = new FileInputStream("input.txt");
var scin = new Scanner(fin);

var scin = new Scanner(
      new FileInputStream("input.txt")
    );
```

Reading and writing text

- Recall `Scanner` class
  - Can apply to any input stream
- Many read methods

```
var fin = new FileInputStream("input.txt");
var scin = new Scanner(fin);

var scin = new Scanner(
      new FileInputStream("input.txt")
    );


String s = scin.nextLine(); // One line
String w = scin.next(); // One word
int i = scin.nextInt(); // Read an int
boolean b = scin.hasNext(); // Any more words?
```

Reading and writing text

- Recall `Scanner` class
  - Can apply to any input stream
- Many read methods
- To write text, use `PrintWriter` class
  - Apply to any output stream

```
var fout = new FileOutputStream("output.txt");
var pout = new PrintWriter(fout);

pout var = new PrintWriter(
      new FileOutputStream("output.txt");
    );
```

Reading and writing text

- Recall `Scanner` class
  - Can apply to any input stream
- Many read methods
- To write text, use `PrintWriter` class
  - Apply to any output stream
- Use `println()`, `print()` to write txt

```
var fout = new FileOutputStream("output.txt");
var pout = new PrintWriter(fout);

pout var = new PrintWriter(
      new FileOutputStream("output.txt");
    );


String msg = "Hello, world";
pout.println(msg);
```

Reading and writing text

- Recall `Scanner` class
  - Can apply to any input stream
- Many read methods
- To write text, use `PrintWriter` class
  - Apply to any output stream
- Use `println()`, `print()` to write txt
- Example: Copy input text file to output text file

```
var in = new Scanner(...);
var out = new PrintWriter(...);

while (in.hasNext()){
    String line = in.nextLine();
    out.println(line);
}
```

Reading and writing text

- Recall `Scanner` class
  - Can apply to any input stream
- Many read methods
- To write text, use `PrintWriter` class
  - Apply to any output stream
- Use `println()`, `print()` to write txt
- Example: Copy input text file to output text file
- Beware: input/output methods generate many different kinds of exceptions
  - Need to wrap code with `try` blocks

```
var in = new Scanner(...);
var out = new PrintWriter(...);

while (in.hasNext()){
    String line = in.nextLine();
    out.println(line);
}
```

Reading and writing binary data

- To read binary data, use `DataInputStream` class
  - Can apply to any input stream

```
var fin = new FileInputStream("input.class");
var din = new DataInputStream(fin);

var din = new DataInputStream(
      new FileInputStream("input.class")
    );
```

Reading and writing binary data

- To read binary data, use `DataInputStream` class
  - Can apply to any input stream
- Many read methods

```
var fin = new FileInputStream("input.class");
var din = new DataInputStream(fin);

var din = new DataInputStream(
      new FileInputStream("input.class")
    );
```

```
readInt, readShort, readLong
readFloat, readDouble,
readChar, readUTF
readBoolean
```

Reading and writing binary data

- To read binary data, use `DataInputStream` class
  - Can apply to any input stream
- Many read methods
- To write binary data, use `DataOutputStream` class
  - Apply to any output stream

```
var fout = new FileOutputStream("output.bin");
var dout = new DataOutputStream(fout);

var dout = new DataOutputStream(
      new FileOutputStream("output.bin")
    );
```

Reading and writing binary data

- To read binary data, use `DataInputStream` class
  - Can apply to any input stream
- Many read methods
- To write binary data, use `DataOutputStream` class
  - Apply to any output stream
- Many write methods

```
var fout = new FileOutputStream("output.bin");
var dout = new DataOutputStream(fout);

var dout = new DataOutputStream(
      new FileOutputStream("output.bin")
    );
```

```
writeInt, writeShort, writeLong
writeFloat, writeDouble
writeChar, writeUTF
writeBoolean
writeChars
writeByte
```

Reading and writing binary data

- To read binary data, use `DataInputStream` class
  - Can apply to any input stream
- Many read methods
- To write binary data, use `DataOutputStream` class
  - Apply to any output stream
- Many write methods
- Example: Copy input binary file to output binary file
  - Again, be careful to catch exceptions

```
var in = new DataInputStream(...);
var out = new DataOutputStream(...);

int bytesAvailable = in.available();
while (bytesAvailable > 0){
    var data = new byte[bytesAvailable];
    in.read(data);
    out.write(data);
    bytesAvailable = in.available();
}
```

Other features

- Buffering an input stream
  - Reads blocks of data
  - More efficient

```
var din = new DataInputStream(
   new BufferedInputStream(
      new FileInputStream("grades.dat")
   )
);
```

Other features

- Buffering an input stream
  - Reads blocks of data
  - More efficient
- Speculative reads
  - Examine the first element
  - Return to stream if necessary

```
var din = new DataInputStream(
   new BufferedInputStream(
      new FileInputStream("grades.dat")
   )
);
```

```
var pbin = new PushbackInputStream(
   new BufferedInputStream(
      new FileInputStream("grades.dat")));

int b = pbin.read();

if (b != '<') pbin.unread(b);
```

Other features

- Buffering an input stream
  - Reads blocks of data
  - More efficient
- Speculative reads
  - Examine the first element
  - Return to stream if necessary
- Streams are specialized
  - `PushBackStream` can only `read()` and `unread()`
  - Feed to a `DataInputStream` to read meaningful data

```
var pbin = new PushbackInputStream(
      new BufferedInputStream(
         new FileInputStream("grades.dat")));

var din = new DataInputStream(pbin);
```

Other features

- Buffering an input stream
  - Reads blocks of data
  - More efficient
- Speculative reads
  - Examine the first element
  - Return to stream if necessary
- Streams are specialized
  - `PushBackStream` can only `read()` and `unread()`
  - Feed to a `DataInputStream` to read meaningful data

```
var pbin = new PushbackInputStream(
      new BufferedInputStream(
         new FileInputStream("grades.dat")));

var din = new DataInputStream(pbin);
```

- Java has a whole zoo of streams for different tasks
  - Random access files, zipped data, ...

Other features

- Buffering an input stream
  - Reads blocks of data
  - More efficient
- Speculative reads
  - Examine the first element
  - Return to stream if necessary
- Streams are specialized
  - `PushBackStream` can only `read()` and `unread()`
  - Feed to a `DataInputStream` to read meaningful data

```
var pbin = new PushbackInputStream(
      new BufferedInputStream(
         new FileInputStream("grades.dat")));

var din = new DataInputStream(pbin);
```

- Java has a whole zoo of streams for different tasks
  - Random access files, zipped data, ...
- Chain together streams in a pipeline
  - Read binary data from a zipped file

  `FileInputStream` → `ZipInputStream` → `DataInputStream`

Summary

- Java's approach to input/output is to separate out concerns
- Chain together different types of input/output streams
  - Connect an external source as input or output
  - Read and write raw bytes
  - Interpret raw bytes as text
  - Interpret raw bytes as data
  - Buffering, speculative read, random access files, zipped data, ...
- Chaining together streams appears tedious, but adds flexibility