NUS National University of Singapore

School of Computing

# CS1020 Data Structures and Algorithms I
Lecture Note #16

## File Processing

## Objective

Input/output on files: reading input from a file and writing data to a file.

# References

Book

- Chapter 1, Section 1.8, pages 80 to 92

CS1020 website → Resources → Lectures

- http://www.comp.nus.edu.sg/~cs1020/2_resources/lectures.html

## Outline

## 0. Recapitulation

- We far we have been using the Scanner class to do interactive input.
- We have also been using the UNIX input redirection < to redirect data from a file, and output redirection > to redirect data to a file.
- < and > are UNIX features, not Java's.
- Now, we will explore how to create File objects in Java.

## 1 File Input

## 1.1 File Objects (1/2)

- The API File class represents files
  - In java.io package
  - Creating a File object does not actually create that file on your drive
- Some methods in File class:

| Method | Description |
|---|---|
| boolean canRead() | Tests whether the application can read the file |
| boolean canWrite() | Tests whether the application can modify the file |
| boolean delete() | Deletes the file or directory |
| boolean exists() | Tests whether the file or directory exists |
| String getName() | Returns the name of the file or directory |
| long length() | Returns the length (in bytes) of the file |

## 1.1 File Objects (2/2)

- Example:

```
File f = new File("myfile");

if (f.exists() && f.length() > 2048) {
   f.delete();
}
```

- Path
  - Absolute path
    - Specify a drive or start with the root (/) directory
    - Eg: `"C:/Documents/CS1020/data"`
  - Relative path
    - With respect to where the program resides
    - Eg: `"input/eels3.in"`

## 1.2 Reading a File (1/3)

- Pass a File reference when constructing a Scanner object

FileExample1.java

```
import java.util.*;
import java.io.*;
public class FileExample1 {
  public static void main(String[] args)
                               throws FileNotFoundException {

    Scanner infile = new Scanner(new File("example"));
    int sum = 0;
    while (infile.hasNextInt()) {
      sum += infile.nextInt();
    }
    System.out.println("Sum = " + sum);
}
```

File “example”:

```
2 7 -3 9 1
```

Output:

```
Sum = 16
```

## 1.2 Reading a File (2/3)

FileExample2.java

```
import java.util.*;
import java.io.*;
public class FileExample2 {
  public static void main(String[] args)
                                  throws FileNotFoundException {

    try {
      Scanner infile = new Scanner(new File("example"));
      int sum = 0;
      while (infile.hasNextInt()) {
        sum += infile.nextInt();
      }
      System.out.println("Sum = " + sum);
    }
    catch (FileNotFoundException e) {
      System.out.println("File 'example' not found!");
    }
}
```

## 1.2 Reading a File (3/3)

FileExample3.java

```java
import java.util.*;
import java.io.*;
public class FileExample2 {
  public static void main(String[] args)
                              throws FileNotFoundException {

    File f = new File("example");
    if (!f.exists()) {
      System.out.println("File 'example' does not exist!");
      System.exit(1);
    }
    Scanner infile = new Scanner(f);
    int sum = 0;
    while (infile.hasNextInt()) {
      sum += infile.nextInt();
    }
    System.out.println("Sum = " + sum);
  }
}
```

## 1.3 Input Tokens (1/3)

- Input data are broken into tokens when read.
- Scanner view all input as a stream of characters, which it processes with its input cursor
- Each call to extract the next input (next(), nextInt(), nextDouble(), etc.) advances the cursor to the end of the current token
- Tokens are separated by whitespace

## 1.3 Input Tokens (2/3)

InputTokens.java

```
import java.util.*;
import java.io.*;
public class InputTokens {
   public static void main(String[] args)
                                     throws FileNotFoundException {
      Scanner infile = new Scanner(new File("tokens"));
      int a = infile.nextInt()
      String b = infile.next();
      String c = infile.nextLine();
      double d = infile.nextDouble();
      System.out.println("a = " + a); System.out.println("b = " + b);
      System.out.println("c = " + c); System.out.println("d = " + d);
   }
}
```

File “tokens”:

(viewed on screen)

```
123 CS1020 Data Structures and Algorithms 1
456 78.9
```

(internally)

```
123 CS1020 Data Structures and Algorithms 1\n456 78.9\n
```

## 1.3 Input Tokens (3/3)

```
int a = infile.nextInt();
String b = infile.next();
String c = infile.nextLine();
double d = infile.nextDouble();

System.out.println("a = " + a);
System.out.println("b = " + b);
System.out.println("c = " + c);
System.out.println("d = " + d);
```

```
a = 123
b = CS1020
c =  Data Structures and Algorithms 1
d = 456.0
```

File "tokens":

```
123 CS1020 Data Structures and Algorithms 1\n456 78.9\n
↑
```

After `int a = infile.nextInt();`

```
123 CS1020 Data Structures and Algorithms 1\n456 78.9\n
```

After `String b = infile.next();`

```
123 CS1020 Data Structures and Algorithms 1\n456 78.9\n
```

After `String c = infile.nextLine();`

```
123 CS1020 Data Structures and Algorithms 1\n456 78.9\n
```

After `double d = infile.double();`

```
123 CS1020 Data Structures and Algorithms 1\n456 78.9\n
```

## 1.4 Tokenizing a String

- A Scanner can tokenize a string

StringTokenize.java

```
import java.util.*;
import java.io.*;
public class StringTokenize {
  public static void main(String[] args) {
    String msg = "345 students in CS1020.";
    Scanner sc = new Scanner(msg);

    int a = sc.nextInt()
    String b = sc.next();
    String c = sc.nextLine();

    System.out.println("a = " + a);
    System.out.println("b = " + b);
    System.out.println("c = " + c);
  }
}
```

## 1.5 Exercise: Runners (1/4)

- Write a program to read in the distances run by a group of runners
- Sample input file "runners_data":
  - Runner ID (type int), name (String, a single word), followed by a list of distances in km (type double)
  - You may assume that there are at least one runner and each runner has at least one distance record

```
123 Charlie 6.5 5.2 7.8 5.8 7.2 6.6 9.2 7.2
987 Alex 12.8
312 Jenny 5.7 4 6.2
509 Margaret 3.1 3.4 3.2 3.1 3.5
610 Richard 11.2 13.2 10.8 9.5 15.8 12.4
```

# 1.5 Exercise: Runners (2/4)

RunnersFlawed.java

```java
import java.util.*;
import java.io.*;
public class RunnersFlawed {
   public static void main(String[] args)
                                 throws FileNotFoundException {
      Scanner infile = new Scanner(new File("runners_data"));
      int count = 0; double totalDist = 0.0;
      while (infile.hasNext()) {
         infile.nextInt(); // read ID
         infile.next();    // read name
         while (infile.hasNextDouble()) {
            count++;
            totalDist += infile.nextDouble();
         }
      }
      System.out.printf("Total distance = %.2f\n", totalDist);
      System.out.printf("Average distance per run = %.2f\n",
                        totalDist/count);
   }
}
```

```
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:864)
        at java.util.Scanner.next(Scanner.java:1485)
        at java.util.Scanner.nextInt(Scanner.java:2117)
        at java.util.Scanner.nextInt(Scanner.java:2076)
        at RunnersFlawed.main(RunnersFlawed.java:14)
```

## 1.5 Exercise: Runners (3/4)

- What went wrong?

RunnersFlawed.java

```java
int count = 0; double totalDist = 0.0;
while (infile.hasNext()) {
   infile.nextInt(); // read ID
   infile.next();    // read name
   while (infile.hasNextDouble()) {
      count++;
      totalDist += infile.nextDouble();
   }
}
```

```
123 Charlie 6.5 5.2 7.8 5.8 7.2 6.6 9.2 7.2
987 Alex 12.8
312 Jenny 5.7 4 6.2
509 Margaret 3.1 3.4 3.2 3.1 3.5
610 Richard 11.2 13.2 10.8 9.5 15.8 12.4
```

## 1.5 Exercise: Runners (4/4)

- Solution: read line by line, then read tokens from each line.

RunnersCorrected.java

```
// Earlier portion omitted for brevity
Scanner infile = new Scanner(new File("runners_data"));
int count = 0; double totalDist = 0.0;
while (infile.hasNextLine()) {
   String line = infile.nextLine();
   Scanner sc = new Scanner(line);
   sc.nextInt(); // read ID
   sc.next();    // read name
   while (sc.hasNextDouble()) {
      count++;
      totalDist += sc.nextDouble();
   }
}
// Later portion omitted for brevity
```

```
Total distance = 173.40
Average distance per run = 7.54
```

## 2 File Output

## 2.1 PrintStream (1/2)

- In java.io package
- PrintStream: An object that allows you to print output to a file
  - Any methods you have used on System.out (such as println()) will work on a PrintStream

```
PrintStream name = new PrintStream(new File("filename"));
```

- Example:

```
PrintStream ps = new PrintStream(new File("greetings"));
ps.println("Hello world!");
ps.println("The quick brown fox jumps over the lazy dog.");
```

## 2.1 PrintStream (2/2)

```
PrintStream name = new PrintStream(new File("filename"));
```

- If the file does not exist, it is created.
- If the file already exists, it is overwritten.
- Note: Do NOT open the same file for reading (Scanner) and writing (PrintStream) at the same time
  - You will overwrite the input file with an empty file

Materials from Pearson

## 2.2 System.out and PrintStream

- System.out is actually a PrintStream
- A reference to it can be stored in a PrintStream variable
    - Printing to that variable causes console output to appear

```
PrintStream out1 = System.out;
PrintStream out2 = new PrintStream(new File("data.txt"));
out1.println("Hello, console!"); // goes to console
out2.println("Hello, file!");    // goes to file
```

## 2.3 Exercise: Runners (revisit)

- Modify RunnersCorrected.java to send its output to the file “running_stat”.

RunnersOutfile.java

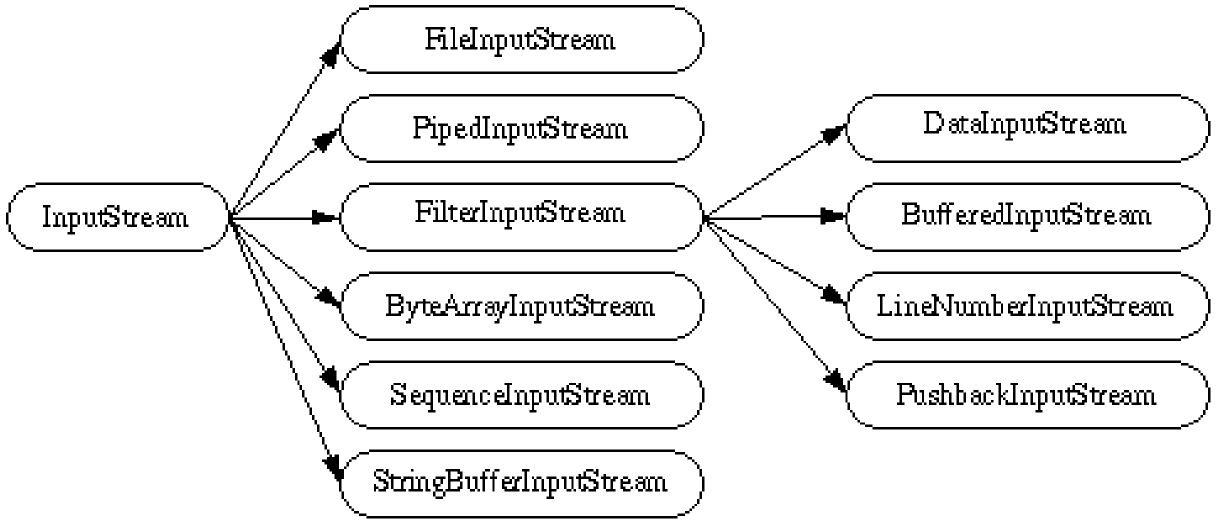
```java
import java.util.*;
import java.io.*;
public class RunnersOutfile {
   public static void main(String[] args)
                                    throws FileNotFoundException {
      Scanner infile = new Scanner(new File("runners_data"));

      // code omitted for brevity

      PrintStream outfile = new PrintStream(new File("running_stat"));
      outfile.printf("Total distance = %.2f\n", totalDist);
      outfile.printf("Average distance per run = %.2f\n",
                     totalDist/count);
      outfile.close();
   }
}
```

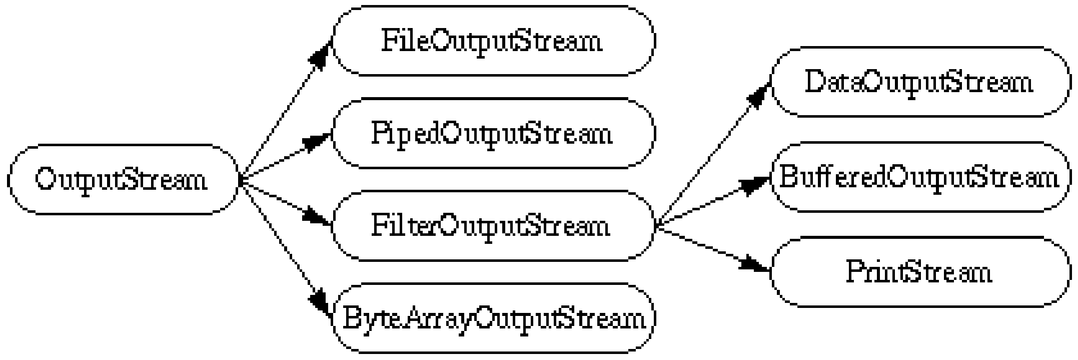
# 3 Input and Output Streams

## 3.1 InputStream and OutputStream (1/2)

- InputStream and OutputStream are abstractions of the different ways to input and output data
  - That is, it doesn’t matter if the stream is a file, a web page, a video, etc.
  - All that matters is that you receive information from the stream or send information into the stream.
  - InputStream is an abstract superclass that provides a minimal programming interface and a partial implementation of input streams. It defines methods for reading bytes, arrays of bytes, etc.
  - OutputStream is an abstract superclass that provides a minimal programming interface and a partial implementation of output streams. It defines methods for writing bytes or arrays of bytes to the stream.

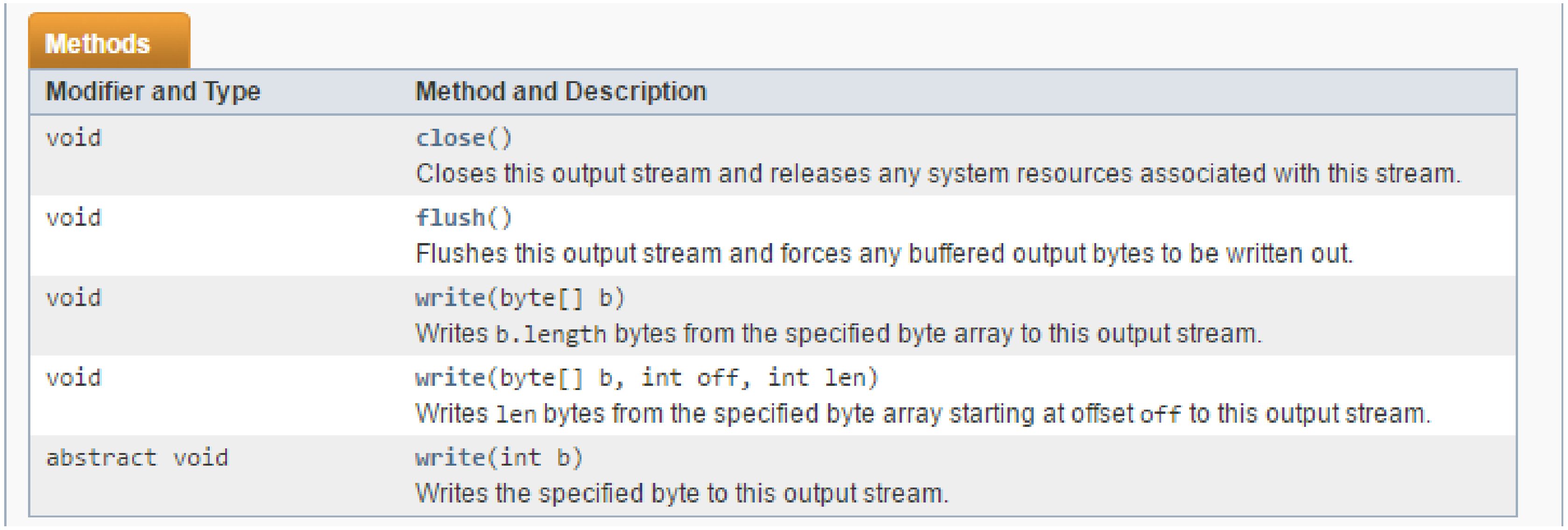
## 3.1 InputStream and OutputStream (2/2)





## 3.2 Example: Using OutputStream

- We will use some of the methods in OutputStream below:



Methods

| Modifier and Type | Method and Description |
|---|---|
| `void` | `close()`<br>Closes this output stream and releases any system resources associated with this stream. |
| `void` | `flush()`<br>Flushes this output stream and forces any buffered output bytes to be written out. |
| `void` | `write(byte[] b)`<br>Writes `b.length` bytes from the specified byte array to this output stream. |
| `void` | `write(byte[] b, int off, int len)`<br>Writes `len` bytes from the specified byte array starting at offset `off` to this output stream. |
| `abstract void` | `write(int b)`<br>Writes the specified byte to this output stream. |

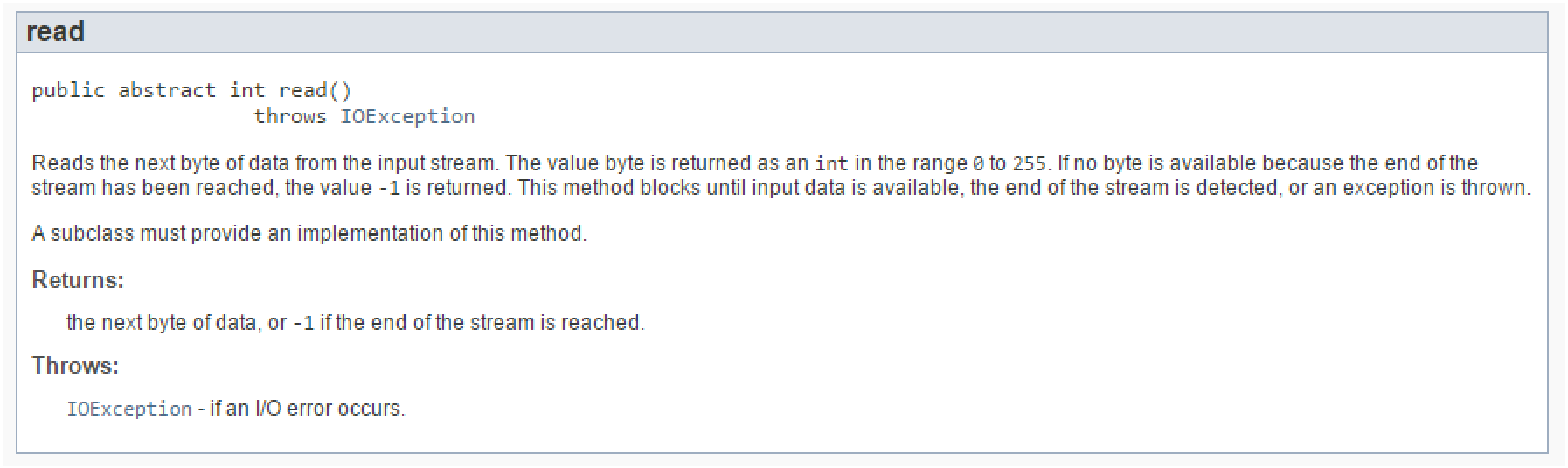
## 3.2 Example: Using OutputStream

TestOutputStream.java

```java
import java.io.*;
public class TestOutputStream {

   public static void main(String[] args) throws IOException {
      String msg = new String("Hello world!");
      OutputStream out = new FileOutputStream("msg_file");

      byte[] bytes = msg.getBytes();

      out.write(bytes);
      out.write(bytes[1]);
      out.write(10);  // ASCII value of newline
      out.write(bytes, 3, 5);
      out.close();
   }
}
```

```
javac TestOutputStream.java
java TestOutputStream
cat msg_file
Hello world!e
lo wo
```

## 3.2 Example: Using InputStream



**read**

```
public abstract int read()
                  throws IOException
```

Reads the next byte of data from the input stream. The value byte is returned as an `int` in the range `0` to `255`. If no byte is available because the end of the stream has been reached, the value `-1` is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

A subclass must provide an implementation of this method.

**Returns:**

the next byte of data, or `-1` if the end of the stream is reached.

**Throws:**

`IOException` - if an I/O error occurs.

## 3.2 Example: Using InputStream

TestInputStream.java

```
import java.io.*;
public class TestInputStream {

   public static void main(String[] args) throws IOException {
      InputStream in = new FileInputStream("msg_file");
      int value;

      while ((value = in.read()) != -1) {
         System.out.print((char)value);
      }
      System.out.println();
      in.close();
   }
}
```

```
javac TestInputStream.java
java TestInputStream
Hello world!e
lo wo
```

End of file