Building Java Programs

Chapter 6: File Processing

Lecture 6-1: File input using Scanner

reading: 6.1 - 6.2, 5.3

self-check: Ch. 6 #1-6

exercises: Ch. 6 #5-7

Input/output ("I/O")

- `import java.io.*;`
- Create a `File` object to get info about a file on disk.

  (This doesn't actually create a new file on the disk.)

```
File f = new File("example.txt");
if (f.exists() && f.length() > 1000) {
    f.delete();
}
```

Method name	**Description**
`canRead()`	returns whether file is able to be read
`delete()`	removes file from disk
`exists()`	whether this file exists on disk
`getName()`	returns file's name
`length()`	returns number of bytes in file
`renameTo(`*file*`)`	changes name of file

Reading files

- To read a file, pass a `File` when constructing a `Scanner`.

```
Scanner <name> = new Scanner(new File("<file name>"));
```

Example:

```
Scanner input = new Scanner(new File("numbers.txt"));
```

or:

```
File file = new File("numbers.txt");
Scanner input = new Scanner(file);
```

File paths

- **absolute path**: specifies a drive or a top `"/"` folder
  - `"C:/Documents/smith/hw6/input/data.csv"`
  - Windows can also use backslashes to separate folders.

- **relative path**: does not specify any top-level folder
  - `"names.dat"`
  - `"input/kinglear.txt"`
  - Assumed to be relative to the *current directory*:

```
Scanner input = new Scanner(new File("data/readme.txt"));
```

If our program is in `H:/hw6`,
Scanner will look for `H:/hw6/data/readme.txt`

Compiler error w/ files

- The following program does not compile:

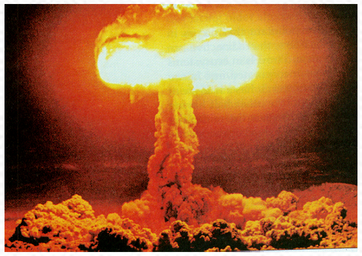
```java
import java.io.*;      // for File
import java.util.*;    // for Scanner

public class ReadFile {
    public static void main(String[] args) {
        Scanner input = new Scanner(new File("data.txt"));
        String text = input.next();
        System.out.println(text);
    }
}
```

- The following error occurs:

```
ReadFile.java:6: unreported exception java.io.FileNotFoundException;
must be caught or declared to be thrown
        Scanner input = new Scanner(new File("data.txt"));
                        ^
```

Exceptions



- **exception**: An object representing a program error.
  - Programs with invalid logic will cause exceptions:
    - dividing by 0
    - calling `charAt` on a `String` and passing too large an index
    - trying to read a file that does not exist
  - We say that a logical error "*throws*" an exception.
  - It is also possible to "*catch*" (handle or fix) an exception.

- **checked exception**: An error that must be handled by our program (otherwise it will not compile).
  - We must specify how our program will handle file I/O failures.

Throwing exceptions

- **`throws` clause**: Keywords placed on a method's header to state that it may generate an exception.
  - Like saying, *"I hereby agree that this method might throw an exception, and I accept the consequences if this happens."*

- Syntax:

```
public static <type> <name>(<params>) throws <type> {
```

  - Example:

```
public class ReadFile {
    public static void main(String[] args)
                throws FileNotFoundException {
```

Input tokens

- **token**: A unit of user input, separated by whitespace.
  - A `Scanner` splits a file's contents into tokens.

- If an input file contains the following:

```
23   3.14
  "John Smith"
```

The `Scanner` can interpret the tokens as the following types:

Token	Type(s)
`23`	`int`, `double`, `String`
`3.14`	`double`, `String`
`"John`	`String`
`Smith"`	`String`

Files and input cursor

- Consider a file `numbers.txt` that contains this text:

```
308.2
   14.9 7.4  2.8

3.9 4.7    -15.4
    2.8
```

- A `Scanner` views all input as a stream of characters:

  -
    ```
    308.2\n   14.9 7.4  2.8\n\n\n3.9 4.7 -15.4\n2.8\n
    ^
    ```

- **input cursor**: The current position of the `Scanner`.

Consuming tokens

- **consuming input**: Reading input and advancing the cursor.
  - Calling `nextInt` etc. moves the cursor past the current token.

```
308.2\n   14.9 7.4  2.8\n\n\n3.9 4.7 -15.4\n2.8\n
^

input.nextDouble()     // 308.2
308.2\n   14.9 7.4  2.8\n\n\n3.9 4.7 -15.4\n2.8\n
     ^

input.next()           // "14.9"
308.2\n   14.9 7.4  2.8\n\n\n3.9 4.7 -15.4\n2.8\n
              ^
```

File input question

- Recall the input file `numbers.txt`:

```
308.2
   14.9 7.4  2.8


3.9 4.7    -15.4
    2.8
```

- Write a program that reads the first 5 values from the file and prints them along with their sum.

```
number = 308.2
number = 14.9
number = 7.4
number = 2.8
number = 3.9
Sum = 337.19999999999993
```

File input answer

```
// Displays the first 5 numbers in the given file,
// and displays their sum at the end.

import java.io.*;    // for File
import java.util.*;  // for Scanner

public class Echo {
    public static void main(String[] args)
            throws FileNotFoundException {
        Scanner input = new Scanner(new File("numbers.txt"));
        double sum = 0.0;
        for (int i = 1; i <= 5; i++) {
            double next = input.nextDouble();
            System.out.println("number = " + next);
            sum += next;
        }
        System.out.println("Sum = " + sum);
    }
}
```

Scanner exceptions

- `InputMismatchException`
  - You read the wrong type of token (e.g. read `"hi"` as `int`).

- `NoSuchElementException`
  - You read past the end of the input.

- Finding and fixing these exceptions:
  - Read the exception text for line numbers in your code (the first line that mentions your file; often near the bottom):

```
Exception in thread "main"
java.util.NoSuchElementException
        at java.util.Scanner.throwFor(Scanner.java:838)
        at java.util.Scanner.next(Scanner.java:1347)
        at CountTokens.sillyMethod(CountTokens.java:19)
        at CountTokens.main(CountTokens.java:6)
```

Testing for valid input

- `Scanner` methods to see what the next token will be:

Method	Description
`hasNext()`	returns `true` if there are any more tokens of input to read (always true for console input)
`hasNextInt()`	returns `true` if there is a next token and it can be read as an `int`
`hasNextDouble()`	returns `true` if there is a next token and it can be read as a `double`

- These methods do not consume input;
  they just give information about the next token.
  - Useful to see what input is coming, and to avoid crashes

Using hasNext methods

- To avoid exceptions:

```
Scanner console = new Scanner(System.in);
System.out.print("How old are you? ");
if (console.hasNextInt()) {
    int age = console.nextInt();   // will not crash!
    System.out.println("Wow, " + age + " is old!");
} else {
    System.out.println("You didn't type an integer.");
}
```

- To detect the end of a file:

```
Scanner input = new Scanner(new File("example.txt"));
while (input.hasNext()) {
    String token = input.next();   // will not crash!
    System.out.println("token: " + token);
}
```

File input question 2

- Modify the Echo program to process the entire file:

  (It should work no matter how many values are in the file.)

```
number = 308.2
number = 14.9
number = 7.4
number = 2.8
number = 3.9
number = 4.7
number = -15.4
number = 2.8
Sum = 329.29999999999995
```

File input answer 2

```
// Displays each number in the given file,
// and displays their sum at the end.

import java.io.*;      // for File
import java.util.*;    // for Scanner

public class Echo2 {
    public static void main(String[] args)
            throws FileNotFoundException {
        Scanner input = new Scanner(new File("numbers.dat"));
        double sum = 0.0;
        while (input.hasNextDouble()) {
            double next = input.nextDouble();
            System.out.println("number = " + next);
            sum += next;
        }
        System.out.println("Sum = " + sum);
    }
}
```

File input question 3

- Modify the program to handle files that contain non-numeric tokens (by skipping them).
- For example, it should produce the same output as before when given this input file, `numbers2.dat`:

```
308.2  hello
   14.9 7.4  bad stuff   2.8

3.9 4.7  oops  -15.4
:-)    2.8  @#*($&
```

File input answer 3

```
// Displays each number in the given file,
// and displays their sum at the end.
import java.io.*;       // for File
import java.util.*;     // for Scanner
public class Echo3 {
    public static void main(String[] args)
            throws FileNotFoundException {
        Scanner input = new Scanner(new File("numbers2.dat"));
        double sum = 0.0;
        while (input.hasNext()) {
            if (input.hasNextDouble()) {
                double next = input.nextDouble();
                System.out.println("number = " + next);
                sum += next;
            } else {
                input.next();   // throw away the bad token
            }
        }
        System.out.println("Sum = " + sum);
    }
}
```

Line-based file processing

reading: 6.3

self-check: #7-11

exercises: #1-4, 8-11

Hours question

- Given a file `hours.txt` with the following contents:

```
123 Susan 12.5 8.1 7.6 3.2
456 Brad 4.0 11.6 6.5 2.7 12
789 Jenn 8.0 8.0 8.0 8.0 7.5
```

  - Consider the task of computing hours worked by each person:

```
Susan (ID#123) worked 31.4 hours (7.85 hours/day)
Brad (ID#456) worked 36.8 hours (7.36 hours/day)
Jenn (ID#789) worked 39.5 hours (7.9 hours/day)
```

- Let's try to solve this problem token-by-token ...

Hours answer (flawed)

```
import java.io.*;                 // for File
import java.util.*;               // for Scanner

public class HoursWorked {        // a non-working solution
    public static void main(String[] args)
            throws FileNotFoundException {
        Scanner input = new Scanner(new File("hours.txt"));
        while (input.hasNext()) {
            // process one person
            int id = input.nextInt();
            String name = input.next();
            double totalHours = 0.0;
            int days = 0;
            while (input.hasNextDouble()) {
                totalHours += input.nextDouble();
                days++;
            }
            System.out.println(name + " (ID#" + id +
                    ") worked " + totalHours + " hours (" +
                    (totalHours / days) + " hours/day)");
        }
    }
}
```

Flawed output

```
Susan (ID#123) worked 487.4 hours (97.48 hours/day)
Exception in thread "main"
java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:840)
        at java.util.Scanner.next(Scanner.java:1461)
        at java.util.Scanner.nextInt(Scanner.java:2091)
        at HoursWorked.main(HoursBad.java:9)
```

- The inner `while` loop is grabbing the next person's ID.
- We want to process the tokens, but we also care about the line breaks (they mark the end of a person's data).

- A better solution is a hybrid approach:
  - First, break the overall input into lines.
  - Then break each line into tokens.

Line-based Scanner methods

Method	Description
`nextLine()`	returns the next entire line of input
`hasNextLine()`	returns `true` if there are any more lines of input to read (always true for console input)

- `nextLine` consumes from the input cursor to the next `\n` .

```
Scanner input = new Scanner(new File("<file name>"));
while (input.hasNextLine()) {
    String line = input.nextLine();
    <process this line>;
}
```

Consuming lines of input

```
23   3.14 John Smith     "Hello world"
              45.2          19
```

- The Scanner reads the lines as follows:

```
23\t3.14 John Smith\t"Hello world"\n\t\t45.2  19\n
^
```

  - *input.nextLine()*

```
23\t3.14 John Smith\t"Hello world"\n\t\t45.2  19\n
                                    ^
```

  - *input.nextLine()*

```
23\t3.14 John Smith\t"Hello world"\n\t\t45.2  19\n
                                                  ^
```

  - Each \n character is consumed but not returned.

Scanners on Strings

- A `Scanner` can tokenize the contents of a `String`:

  `Scanner` ***<name>*** `= new Scanner(`***<String>***`);`

  - Example:

```
String text = "15  3.2 hello   9  27.5";
Scanner scan = new Scanner(text);
System.out.println(scan.nextInt());       // 15
System.out.println(scan.nextDouble());    // 3.2
System.out.println(scan.next());          // hello
```

Tokenizing lines of a file

Input file `input.txt`:	Output to console:

`The quick brown fox jumps over`
`the lazy dog.`
	`Line has 6 words`
	`Line has 3 words`

```
// Counts the words on each line of a file
Scanner input = new Scanner(new File("input.txt"));
while (input.hasNextLine()) {
    String line = input.nextLine();
    Scanner lineScan = new Scanner(line);

    // process the contents of this line
    int count = 0;
    while (lineScan.hasNext()) {
        String word = lineScan.next();
        count++;
    }
    System.out.println("Line has " + count + " words");
}
```

Hours answer corrected

```
// Processes an employee input file and outputs
  each employee's hours data.
import java.io.*;     // for File
import java.util.*;   // for Scanner

public class Hours {
    public static void main(String[] args)
  throws FileNotFoundException {
        Scanner input = new Scanner(new
  File("hours.txt"));
        while (input.hasNextLine()) {
            String line = input.nextLine();
            Scanner lineScan = new
  Scanner(line);
            int id = lineScan.nextInt();
  // e.g. 456
            String name = lineScan.next();
  // e.g. "Brad"
```