

Building Java Programs

Chapter 6

Lecture 6-1: File Input with Scanner

reading: 6.1 - 6.2, 5.3

self-check: Ch. 6 #1-6

exercises: Ch. 6 #5-7

videos: Ch. 6 #1-2

Input/output (I/O)

- So far we have used only the console and drawing panels
- Also very common to read or write files
 - Similar ideas, especially for text files
- Create a `File` object to get access to a file on disk.

```
import java.io.*;
```

*(This does **not** actually create a new file on the hard disk.)*

```
File f = new File("example.txt");
```

- See textbook for useful methods in `File` class (e.g., delete or rename)
- For now, we will just create `Scanners` that use files
 - Will get an exception if the file cannot be found.

Reading files

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

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

Example:

```
File file = new File("mydata.txt");
```

```
Scanner input = new Scanner(file);
```

or, more compactly:

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

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.
- Macintosh example: /Users/smith/hw6/input/data.csv

- **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"));
```

- Easiest: Read a file in the same directory as your program with just "readme.txt".

Compiler error w/ files

- The following program does not compile:

```
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:** Something representing a runtime error.
 - dividing an integer by 0
 - calling `charAt` on a `String` and passing too large an index
 - trying to read the wrong type of value from a `Scanner`
 - trying to read a file that does not exist
- We say that a program with an error "*throws*" an exception.
- (It's possible to "*catch*" (handle) an exception, but we won't)
- **checked exception:** An error that must be handled unless we admit it isn't.
 - We must admit our method "won't work" if the file doesn't exist.

The throws clause

- **throws clause:** Keywords on a method's header to state that it (or something it calls) may generate an exception.
- **Syntax:**

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

- **Example:**

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

- Like saying, *"I hereby announce that this method might throw an exception, and I accept the consequences if it happens."*

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:

| <u>Token</u> | <u>Type(s)</u> |
|--------------|--|
| 23 | <code>int</code> , <code>double</code> , <code>String</code> |
| 3.14 | <code>double</code> , <code>String</code> |
| "John | <code>String</code> |
| Smith" | <code>String</code> |

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\n3.9 4.7 -15.4\n 2.8\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\n3.9 4.7 -15.4\n 2.8\n^
```

```
double x = input.nextDouble(); // 308.2
```

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

```
String s = input.next(); // "14.9"
```

```
308.2\n 14.9 7.4 2.8\n\n3.9 4.7 -15.4\n 2.8\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 = 337.2
```

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 = sum + next;  
        }  
        System.out.printf("Sum = %.1f\n", sum);  
    }  
}
```

File input mini-exercise

- Start with the program that reads the first 5 values from the file and prints them along with their sum. Modify it to read the first 5 *tokens* from the file and print them.

Mini-exercise - answer

```
// Displays the first 5 tokens in the given file.
```

```
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("stuff.txt"));
        for (int i = 1; i <= 5; i++) {
            String next = input.next();
            System.out.println("token = " + next);
        }
    }
}
```

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)
```

Reading an entire file

- Suppose we want our 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 = 329.3
```

- A while-loop, naturally -- but we don't know yet how to write a useful test for this situation!

Testing for valid input

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

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

- 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 = 329.3
```

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 Echo {  
    public static void main(String[] args)  
        throws FileNotFoundException {  
        Scanner input = new Scanner(new File("numbers.txt"));  
        double sum = 0.0;  
        while (input.hasNextDouble()) {  
            double next = input.nextDouble();  
            System.out.println("number = " + next);  
            sum = sum + next;  
        }  
        System.out.printf("Sum = %.1f\n", sum);  
    }  
}
```

File input question 3

- Modify the `Echo` 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.txt`:

```
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 Echo2 {  
    public static void main(String[] args)  
        throws FileNotFoundException {  
        Scanner input = new Scanner(new File("numbers2.txt"));  
        double sum = 0.0;  
        while (input.hasNext()) {  
            if (input.hasNextDouble()) {  
                double next = input.nextDouble();  
                System.out.println("number = " + next);  
                sum = sum + next;  
            } else {  
                input.next(); // throw away the bad token  
            }  
        }  
        System.out.printf("Sum = %.1f\n", sum);  
    }  
}
```

Searching for something

- A while-loop that returns in the middle of scanning a file is useful when “looking for something”
 - Bad style, and wasteful, to keep reading the rest of the file
- Example: First prime number in a file of integers

Stopping early answer, part 1

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

public class FirstPrime {
public static void main(String[] args)
    throws FileNotFoundException {
    Scanner input = new Scanner(new File("integers.txt"));
    int first = scanForPrime(input);
    if(first==0) {
        System.out.println("No primes were found.");
    } else {
        System.out.println("First prime was " + first + ".");
    }
}
```


Stopping early answer, part 2

```
public static int scanForPrime(Scanner s) {
    while (s.hasNextInt()) {
        int n = s.nextInt();
        if(isPrime(n)) {
            return n;
        }
    }
    return 0;
}
```

```
public static boolean isPrime(int n) {
    for(int i=2; i*i <= n; ++i) {
        if(n % i == 0) {
            return false;
        }
    }
    return true;
}
```