

File Input and Output

TOPICS

- File Input
- Exception Handling
- File Output



File class in Java

- Programmers refer to input/output as "I/O".
- The File class represents files as objects.
- The class is defined in the java.io package.
- Creating a File object allows you to get information about a file on the disk.
- Creating a File object does NOT create a new file on your disk.

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



Files

Some methods in the File class:

Method name	Description
canRead()	returns whether file can be read
delete()	removes file from disk
exists()	whether this file exists on disk
getName()	returns name of file
length()	returns number of characters in file
renameTo(filename)	changes name of file



Scanner reminder

- The Scanner class reads input and processes strings and numbers from the user.
- When constructor is called with System.in, the character stream is input typed to the console.
- Instantiate Scanner by passing the input character stream to the constructor:

Scanner scan = new Scanner(System.in);

Scanner reminder

- Common methods called on Scanner:
 - Read a line
 - String str = scan.nextLine();
 - Read a string (separated by whitespace)
 - String str = scan.next();
 - Read an integer
 - int ival = scan.nextInt();
 - Read a double
 - double dval = scan.nextDouble();

Opening a file for reading

- To read a file, pass a File object as a parameter when constructing a Scanner

 String variable
- Scanner for a file:
 - Scanner <name> = new Scanner(new File(<filename>));
- Fyample:
- Scanner scan= new Scanner(new File("numbers.txt"));
- or:

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



File names and paths

- relative path: does not specify any top-level folder, so the path is relative to the current directory:
 - "names.dat"
 - "code/Example.java"
- absolute path: The complete pathname to a file starting at the root directory /:
 - □ In Linux: "/users/cs160/programs/Example.java"
 - □ In Windows: "C:/Documents/cs160/programs/data.csv"



File names and paths

 When you construct a File object with a relative path, Java assumes it is relative to the current directory.

```
Scanner scan =
```

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

- □ If our program is in ~/w
 - ~/workspace/P4
- □ Scanner will look for ~/workspace/P4/data/input.txt

8



Compiler error with files

Question: Why will the following program NOT compile?

Answer: Because of Java exception handling!



Compiler error with files

Here is the compilation error that is produced:

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

- The problem has to do with error reporting.
- What to do when a file cannot be opened?
- File may not exist, or may be protected.
- Options: exit program, return error, or throw exception
- Exceptions are the normal error mechanism in Java.

10



Exceptions



- exception: An object that represents a program error.
 - Programs with invalid logic will cause exceptions.
 - Examples:
 - dividing by zero
 - calling charAt on a String with an out of range index
 - trying to read a file that does not exist
 - We say that a logical error results in an exception being thrown.
 - □ It is also possible to *catch* (handle) an exception.

11



Checked exceptions

- checked exception: An error that must be handled by our program (otherwise it will not compile)
 - We must specify what our program will do to handle any potential file I/O failures.
 - We must either:
 - declare that our program will handle ("catch") the exception, or
 - state that we choose not to handle the exception (and we accept that the program will crash if an exception occurs)



Throwing Exceptions

- throws clause: Keywords placed on a method's header to state that it may generate an exception.
- It's like a waiver of liability:
 - "I hereby agree that this method might throw an exception, and I accept the consequences (crashing) if this happens."
 - General syntax:

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

□ When doing file open, we throw IOException.

13

0,1

Handling Exceptions

When doing file I/O, we use IOException.

```
public static void main(String[] args) {
   try {
     File file = new File("input.txt);
     Scanner scan = new Scanner(file);
     String firstLine = scan.nextLine();
     ...
} catch (IOException e) {
     System.out.println("Unable to open input.txt");
     System.exit(-1);
}
```



Fixing the compiler error

- Throwing an exception or handling the exception both resolve the compiler error.
- Throwing Exceptions: User will see program terminate with exception, that's not very friendly.
- Handling Exceptions: User gets a clear indication of problem with error message, that's much better.
- We will handle exceptions when reading and writing files in programming assignments.

15



Using Scanner to read file data

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
```

16



Consuming tokens

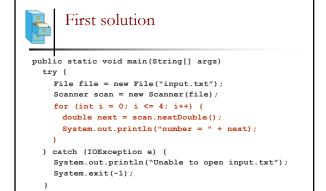
Each call to next, nextInt, nextDouble, etc. advances the position of the scanner to the end of the current token, skipping over any whitespace:

17



First problem

 Write code that reads the first 5 double values from a file and prints.





Second problem

How would we modify the program to read all the file?

20



Second solution

```
public static void main(String[] args)
try {
   File file = new File("input.txt");
   Scanner scan = new Scanner(file);
   while (scan.hasNextDouble() {
      double next = scan.nextDouble();
      System.out.println("number = " + next);
   }
} catch (IOException e) {
   System.out.println("Unable to open input.txt");
   System.exit(-1);
}
```



Refining the problem

- Modify the program again to handle files that also contain non-numeric tokens.
 - □ The program should skip any such tokens.
- For example, it should produce the same output as before when given this input file:

```
308.2 hello
14.9 7.4 bad stuff 2.8

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

22

Refining the program

```
while (scan.hasNext()) {
   if (scan.hasNextDouble()) {
      double next = scan.nextDouble();
      System.out.println("number = " + next);
   } else {
      // consume the bad token
      scan.next();
   }
}
```

Reading input line-by-line

- Given the following input data: 23 3.14 John Smith "Hello world" 45.2 19
- The Scanner can read it line-by-line:

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

scan.nextLine()

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

scan.nextLine()

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

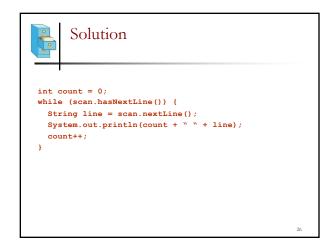
■ The \n character is consumed but not returned.



File processing question

 Write a program that reads a text file and adds line numbers at the beginning of each line

25





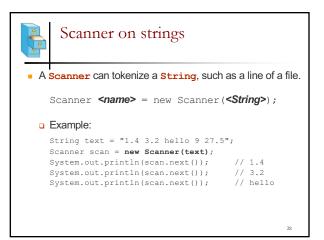
Problem

Given a file 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 Jennifer 8.0 8.0 8.0 8.0 7.5
```

- Consider the task of computing hours worked by each person
- Approach:
 - Break the input into lines.
 - Break each line into tokens.

27





Tokenize an entire file

We can use string Scanner (s) to tokenize each line of

```
Example

Example: Count the tokens on each line of a file.

Scanner scan = new Scanner(new File("input.txt"));
while (scan.hasNextLine()) {

String line = scan.nextLine();

Scanner lineScan = new Scanner(line);
int count = 0;
while (lineScan.hasNext()) {

String token = lineScan.next();
count++;
}
System.out.println("Line has "+count+" tokens");
}

Input file input.txti
23 3.14 John Smith "Hello world"

45.2 19

Quitout to console:
Line has 6 tokens
Line has 2 tokens

30
```



Opening a file for writing

Same story as reading, we must handle exceptions:

```
public static void main(String[] args) {
   try {
     File file = new File("output.txt");
     PrintWriter output = new PrintWriter(file);
     output.println("Integer number: " + 987654);
     ...
} catch (IOException e) {
     System.out.println("Unable to write output.txt");
     System.exit(-1);
}
```



File output

- You can output all the same things as you would with System.out.println:
- Discussion so far has been limited to text files.

```
output.println("Double: " + fmt.format(123.456));
output.println("Integer: " + 987654);
output.println("String: " + "Hello There");
```

- Binary files store data as numbers, not characters.
- Binary files are not human readable, but more efficient.