Topic 18
file input, tokens

"We have also obtained a glimpse of another crucial idea about languages and program design. This is the approach of stratified design, the notion that a complex system should be structured as a sequence of levels that are described using a sequence of languages. Each level is constructed by combining parts that are regarded as primitive at that level, and the parts constructed at each level are used as primitives at the next level. The language used at each level of a stratified design has primitives, means of combination, and means of abstraction appropriate to that level of detail. "

- Hal Abelson
and Gerald Sussman

Based on slides by Marty Stepp and Stuart Reges
from http://www.buildingjavaprograms.com/

File Input/output (I/O)

```
import java.io.File;
```

- Create a `File` object to get info about a file on your drive.
  - (This doesn't actually create a new file on the hard 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

Reading files

- To read a file, pass a `File` object to the `Scanner` Constructor (instead of `System.in`).

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

  - Example:

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

  - or (shorter):

```
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.

- **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` , the `Scanner` will look for `H:/hw6/data/readme.txt`

Working Directory

- New programmers are often not sure what their current directory is.
- Easy to print out:

```
public static void
printWorkingDirectory() {
    System.out.println("Working Directory = " +
        System.getProperty("user.dir"));
}
```

```
Working Directory = C:\Users\scottm\Documents\312\BlueJ
```

http://docs.oracle.com/javase/tutorial/essential/environment/sysprop.html

Compiler error w/ files

```
import java.io.File;
import java.util.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 program fails to compile with the following error:

```
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 runtime error.
  - dividing an integer by 0
  - calling `substring` 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 a runtime 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.

Clicker 1

- Can a programmer prevent a divide by zero error from occurring in all cases?

A. no

B. yes

C. maybe

Clicker 2

- Can a programmer prevent a file not found error from occurring in all cases?

A. no

B. yes

C. maybe

The `throws` clause

- **`throws` clause**: Keywords on a method's header that state that it may generate an exception (and will not handle it).
- Syntax:

```
public static <type> <name>(...) 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 this happens. OR I am passing the buck to someone else.*

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 `weather.txt` that contains this text:

```
16.2   23.5
   19.1 7.4  22.8

18.5  -1.8 14.9

```

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

```
16.2   23.5\n19.1 7.4  22.8\n\n18.5  -1.8 14.9\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

```
16.2   23.5\n19.1 7.4  22.8\n\n18.5  -1.8 14.9\n
^
```

```
double d = input.nextDouble();     // 16.2
```

```
16.2   23.5\n19.1 7.4  22.8\n\n18.5  -1.8 14.9\n
    ^
```

```
String s = input.next();           // "23.5"
```

```
16.2   23.5\n19.1 7.4  22.8\n\n18.5  -1.8 14.9\n
          ^
```

File input question

- Recall the input file `weather.txt`:

```
16.2   23.5
   19.1 7.4  22.8

18.5  -1.8 14.9
```

- Write a program that prints the change in temperature between each pair of neighboring days.

```
16.2 to 23.5, change = 7.3
23.5 to 19.1, change = -4.4
19.1 to 7.4, change = -11.7
7.4 to 22.8, change = 15.4
22.8 to 18.5, change = -4.3
18.5 to -1.8, change = -20.3
-1.8 to 14.9, change = 16.7
```

File input answer

```
// Displays changes in temperature from data in an input file.
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Temperatures {
    public static void main(String[] args)
            throws FileNotFoundException {
        Scanner input = new Scanner(new File("weather.txt"));
        double prev = input.nextDouble();   // fencepost
        for (int i = 1; i <= 7; i++) {
            double next = input.nextDouble();
            System.out.println(prev + " to " + next +
                    ", change = " + (next - prev));
            prev = next;
        }
    }
}
```

Reading an entire file

- Suppose we want our program to work no matter how many numbers are in the file.
  - Currently, if the file has more numbers, they will not be read.
  - If the file has fewer numbers, what will happen?

A crash! Example output from a file with just 3 numbers:

```
16.2 to 23.5, change = 7.3
23.5 to 19.1, change = -4.4
Exception in thread "main"
  java.util.NoSuchElementException
    at java.util.Scanner.throwFor(Scanner.java:838)
    at java.util.Scanner.next(Scanner.java:1347)
    at Temperatures.main(Temperatures.java:12)
```

`Scanner` exceptions

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

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

- 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 MyProgram.myMethodName(MyProgram.java:19)
    at MyProgram.main(MyProgram.java:6)
```

`Scanner` tests for valid input

Method	Description
`hasNext()`	returns `true` if there is a next token
`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 of the `Scanner` do not consume input; they just give information about what the next token will be.
  - Useful to see what input is coming, and to avoid crashes.
  - These methods can be used with a console `Scanner`, as well.
    - When called on the console, they sometimes pause waiting for input.

Using `hasNext` methods

- Avoiding type mismatches:

```
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.");
}
```

- Avoiding reading past the end of a file:

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

File input question 2

- Modify the temperature program to process the entire file, regardless of how many numbers it contains.
  - Example: If a ninth day's data is added, output might be:

```
16.2 to 23.5, change = 7.3
23.5 to 19.1, change = -4.4
19.1 to 7.4, change = -11.7
7.4 to 22.8, change = 15.4
22.8 to 18.5, change = -4.3
18.5 to -1.8, change = -20.3
-1.8 to 14.9, change = 16.7
14.9 to 16.1, change = 1.2
```

File input answer 2

```
// Displays changes in temperature from data in an input file.
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Temperatures {
    public static void main(String[] args)
            throws FileNotFoundException {
        Scanner input = new Scanner(new File("weather.txt"));
        double prev = input.nextDouble();   // fencepost
        while (input.hasNextDouble()) {
            double next = input.nextDouble();
            System.out.println(prev + " to " + next +
                    ", change = " + (next - prev));
            prev = next;
        }
    }
}
```

File input question 3

- Modify the temperature 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, `weather2.txt`:

```
16.2   23.5
Tuesday   19.1   Wed 7.4   THURS. TEMP: 22.8

18.5  -1.8  <-- MIKE here is my data!  --Chris
   14.9 :-)
```

  - You may assume that the file begins with a double.
  - **What if we didn't know that?**

File input answer 3

```
// Displays changes in temperature from data in an input file.
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Temperatures2 {
    public static void main(String[] args)
            throws FileNotFoundException {
        Scanner input = new Scanner(new File("weather.txt"));
        double prev = input.nextDouble();   // fencepost
        while (input.hasNext()) {
            if (input.hasNextDouble()) {
                double next = input.nextDouble();
                System.out.println(prev + " to " + next +
                        ", change = " + (next - prev));
                prev = next;
            } else {
                input.next();  // throw away unwanted token
            }
        }
    }
}
```

"File" Input

- Reading from sources other than files is not very different
- For example we can read data from a web page about as easily as reading from a file
- Example, read stock information from a web page
  - often the hardest thing is finding the web page and the format of the url
  - once you have that the code is easy

Read HTML from a Webpage

- https://www.cnbc.com/
- Read and print out the HTML from that web page
- Could easily alter to read data from any web page or via a web API
  - Service that allows a developer to read data via the web

Display HTML

```
try {
      System.out.println("Raw HTML from CNBC");
      String urlAsString = "https://www.cnbc.com/";
      URL url = new URL(urlAsString);
      Scanner sc
           = new Scanner(new InputStreamReader(url.openStream()));
      while(sc.hasNextLine()) {
          System.out.println(sc.nextLine());
      }
      sc.close();
  }
  catch(IOException e) {
      System.out.println("UH OH: " + e);
  }
```