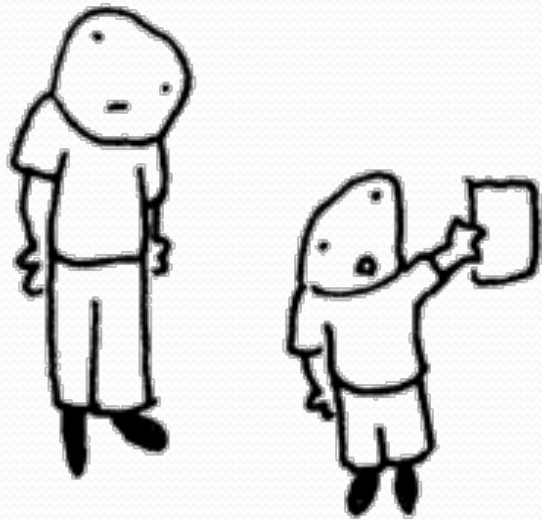# Building Java Programs

Chapter 6
Lecture 13: File Input with `Scanner`

reading: 6.1 – 6.2, 5.4

okay dad. the science fair is tomorrow. let's make up some data.

Toothpaste For Dinner.com

Input/output (I/O)

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

- 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
renameTo(*file*)	changes name of 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`

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 (shorter):

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

# Compiler error w/ files

```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 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
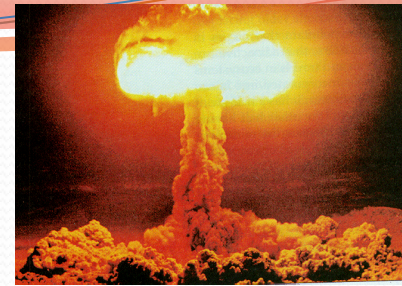
- **exception**: An object representing a runtime error.
  - dividing an integer by 0
  - calling `substring` on a `String` and passing too large an index
  - trying to read the wrong type of value from a `Scanner`
  - trying to read a file that does not exist

  - We say that a program with an 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.

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(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 this 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:

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\n   19.1 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\n    19.1 7.4  22.8\n\n18.5  -1.8 14.9\n
^
```

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

```
16.2    23.5\n    19.1 7.4  22.8\n\n18.5  -1.8 14.9\n
       ^
```

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

```
16.2    23.5\n    19.1 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

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

```java
// Displays changes in temperature from data in an input file.
import java.io.*;     // for File
import java.util.*;  // for 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  <-- Here is my data!  --Ally
   14.9 :-)
```

- You may assume that the file begins with a real number.

File input answer 3

```java
// Displays changes in temperature from data in an input file.
import java.io.*;     // for File
import java.util.*;   // for 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
            }
        }
    }
}
```

Gas prices question

- Write a program that reads a file `gasprices.txt`
  - Format: *Belgium $/gal   US $/gal   date*

  ```
  8.20      3.81  3/21/11
  8.08      3.84  3/28/11
  8.38      3.92  4/4/11
  8.62      4.03  4/11/11
  ```

- The program should print the average gas price over all data in the file for both countries:

  ```
  Belgium:
  8.38      3.92  4/4/11
  8.62      4.03  4/11/11
  ```

Gas prices solution

```java
public class GasPrices {
  public static void main(String[] args) throws
FileNotFoundException {
    Scanner s = new Scanner(new File("gasprices.txt"));
    double belgium = 0;
    double usa = 0;
    int count = 0;

    while (s.hasNext()) {
      belgium += s.nextDouble();
      usa += s.nextDouble();
      count++;
      s.next(); // skip date
    }

    System.out.println("Belgium average: " + belgium / count + "
$/gal");
    System.out.println("USA average: " + usa / count + " $/gal");
  }
}
```

Hours question

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

  ```
  123 Riley 12.5 8.1 7.6 3.2
  456 Molly 4.0 11.6 6.5 2.7 12
  789 Andrew 8.0 8.0 8.0 8.0 7.5
  ```

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

  ```
  Riley (ID#123) worked 31.4 hours (7.85 hours/day)
  Molly (ID#456) worked 36.8 hours (7.36 hours/day)
  Andrew (ID#789) worked 39.5 hours (7.90 hours/day)
  ```

Hours answer (flawed)

```java
// This solution does not work!
import java.io.*;                    // for File
import java.util.*;                  // for Scanner

public class HoursWorked {
    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.printf(
                "%s (ID#%d) worked %.1f hours (%.2f hours/day)\n",
                name, id, totalHours, totalHours / days);
        }
    }
}
```

Flawed output

```
Ben (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 next entire line of input (from cursor to \n)
hasNextLine()	returns true if there are any more lines of input to read (always true for console input)

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

  - `String line = input.nextLine();`
    `23\t3.14 John Smith\t"Hello" world``\n\t\t45.2  19\n`
    `^`

  - `String line2 = 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);

    int num = scan.nextInt();
    System.out.println(num);        // 15

    double num2 = scan.nextDouble();
    System.out.println(num2);       // 3.2

    String word = scan.next();
    System.out.println(word);       // "hello"
    ```

29

Mixing lines and tokens

Input file `input.txt`:	Output to console:
The quick brown fox jumps over the lazy dog.	Line has 6 words Line has 3 words

```java
// 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 question

- Fix the `Hours` program to read the input file properly:

```
123 Riley 12.5 8.1 7.6 3.2
456 Molly 4.0 11.6 6.5 2.7 12
789 Andrew 8.0 8.0 8.0 8.0 7.5
```

  - Recall, it should produce the following output:

```
Riley (ID#123) worked 31.4 hours (7.85 hours/day)
Molly (ID#456) worked 36.8 hours (7.36 hours/day)
Andrew (ID#789) worked 39.5 hours (7.90 hours/day)
```

Hours answer, corrected

```java
// Processes an employee input file and outputs each employee's hours.
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();
            processEmployee(line);
        }
    }

    public static void processEmployee(String line) {
        Scanner lineScan = new Scanner(line);
        int id = lineScan.nextInt();            // e.g. 456
        String name = lineScan.next();          // e.g. "Greg"
        double sum = 0.0;
        int count = 0;
        while (lineScan.hasNextDouble()) {
            sum = sum + lineScan.nextDouble();
            count++;
        }

        double average = sum / count;
        System.out.printf("%s (ID#%d) worked %.1f hours (%.2f hours/day)\n",
                name, id, sum, sum / count);    }
}
```