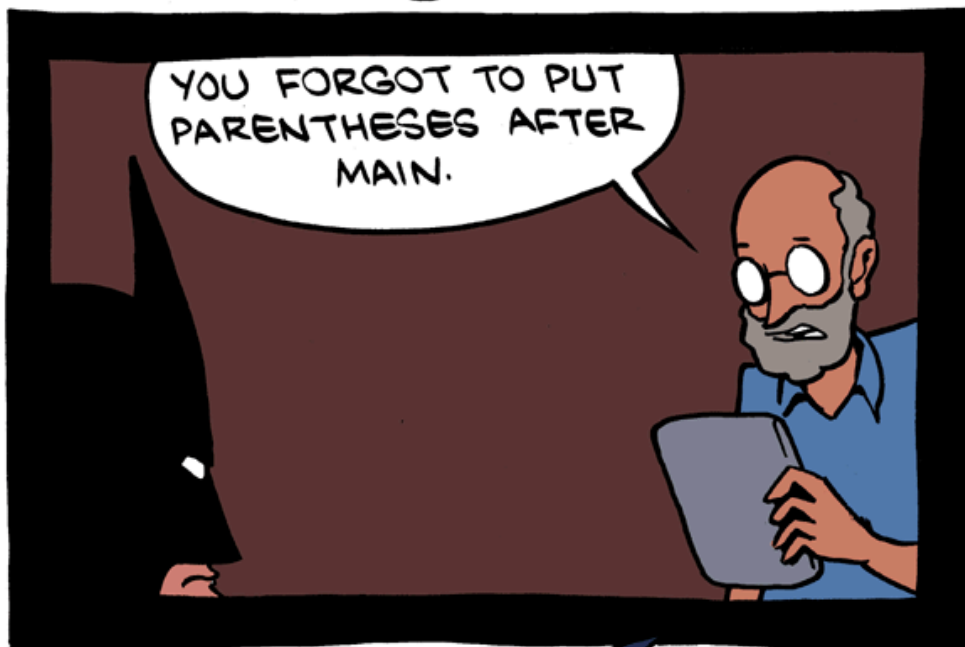# Building Java Programs

Chapter 6
Lecture 6-2: Line-Based File Input

reading: 6.3 - 6.5

Hours question

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

  ```
  123 Ross 12.5 8.2 7.6 4.0
  456 Erika 4.2 11.6 6.3 2.5 12.0
  789 Alex 16.0 12.0 8.0 20.0 7.5
  ```

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

    ```
    Ross (ID#123) worked 32.3 hours (8.075 hours/day)
    Erika (ID#456) worked 36.6 hours (7.32 hours/day
    Alex (ID#789) worked 63.5 hours (12.7 hours/day)
    ```


BEHIND EVERY GOOD TEACHER IS A GREAT TEACHER'S ASSISTANT

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.println(name + " (ID#" + id +
                    ") worked " + totalHours + " hours (" +
                    (totalHours / days) + " hours/day)");
        }
    }
}
```

Flawed output

```
Ross (ID#123) worked 488.3 hours (97.66 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"
    ```

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

```
// 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 tokens = new Scanner(line);

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

Hours question

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

```
123 Ross 12.5 8.2 7.6 4.0
456 Erika 4.2 11.6 6.3 2.5 12.0
789 Alex 16.0 12.0 8.0 20.0 7.5
```

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

```
Ross (ID#123) worked 32.3 hours (8.075 hours/day)
Erika (ID#456) worked 36.6 hours (7.32 hours/day
Alex (ID#789) worked 63.5 hours (12.7 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. "Erika"
        double sum = 0.0;
        int count = 0;
        while (lineScan.hasNextDouble()) {
            sum = sum + lineScan.nextDouble();
            count++;
        }

        double average = sum / count;
        System.out.println(name + " (ID#" + id + ") worked " +
            sum + " hours (" + average + " hours/day)");
    }
}
```

File output

reading: 6.4 - 6.5

Output to files

- **PrintStream**: An object in the `java.io` package that lets you print output to a destination such as a file.

    - Any methods you have used on `System.out` (such as `print, println`) will work on a `PrintStream`.

- Syntax:

    PrintStream **_\<name\>_** = new PrintStream(new File(**"_\<file\>_"**));

    Example:
    ```
    PrintStream output = new PrintStream(new File("out.txt"));
    output.println("Hello, file!");
    output.println("This is a second line of output.");
    ```

Details about `PrintStream`

`PrintStream` ***\<name\>*** `= new PrintStream(new File("`***\<file\>***`"));`

- If the given file does not exist, it is created.
- If the given file already exists, it is overwritten.

- The output you print appears in a file, not on the console. You will have to open the file with an editor to see it.

- Do not open the same file for both reading (`Scanner`) and writing (`PrintStream`) at the same time.
  - You will overwrite your input file with an empty file (0 bytes).

`System.out` and `PrintStream`

- The console output object, `System.out`, is a `PrintStream`.

```
PrintStream out1 = System.out;
PrintStream out2 = new PrintStream(new File("data.txt"));
out1.println("Hello, console!");   // goes to console
out2.println("Hello, file!");      // goes to file
```

- A reference to it can be stored in a `PrintStream` variable.
  - Printing to that variable causes console output to appear.

- You can pass `System.out` to a method as a `PrintStream`.
  - Allows a method to send output to the console or a file.

PrintStream question

- Modify our previous Hours program to use a `PrintStream` to send its output to the file `hours_out.txt`.

  - The program will produce no console output.
  - But the file `hours_out.txt` will be created with the text:

```
Ross (ID#123) worked 32.3 hours (8.075 hours/day)
Erika (ID#456) worked 36.6 hours (7.32 hours/day
Alex (ID#789) worked 63.5 hours (12.7 hours/day)
```

PrintStream answer

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

public class Hours2 {
    public static void main(String[] args) throws FileNotFoundException {
        Scanner input = new Scanner(new File("hours.txt"));
        PrintStream out = new PrintStream(new File("hours_out.txt"));
        while (input.hasNextLine()) {
            String line = input.nextLine();
            processEmployee(out, line);
        }
    }

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

        double average = sum / count;
        out.println(name + " (ID#" + id + ") worked " +
                    sum + " hours (" + average + " hours/day)");
    }
}
```

Prompting for a file name

- We can ask the user to tell us the file to read.
  - The filename might have spaces; use `nextLine()`, not `next()`

    ```java
    // prompt for input file name
    Scanner console = new Scanner(System.in);
    System.out.print("Type a file name to use: ");
    String filename = console.nextLine();
    Scanner input = new Scanner(new File(filename));
    ```

- `File`s have an `exists` method to test for file-not-found:

    ```java
    File file = new File("hours.txt");

    if (!file.exists()) {
        // try a second input file as a backup
        System.out.print("hours file not found!");
        file = new File("hours2.txt");
    }
    ```

File Scanner Question

- Write a program called `Spammer.java` that asks the user for an email domain and searches a file called `address_book.txt`. If an email with the domain name is found, the user is prompted whether the contact should be added to the "spam list" (User input in bold)

  ```
  Email domain to spam? @gmail.com
  Would you like to spam therealsherlock@gmail.com? yes
  ```

- The program should output the contacts that the user selected to a file named `spam_list.txt`

  ```
  Schmerlock Schmolmes <therealsherlock@gmail.com?>
  ```