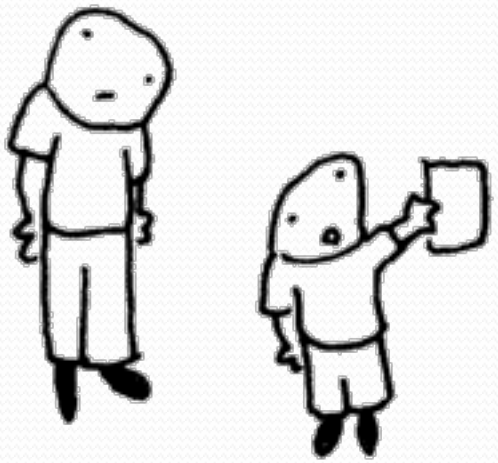
Building Java Programs

Chapter 6

Lecture 6-2: Line-Based File Input

reading: 6.3 - 6.5



Toothpaste For Dinner.com

Hours question

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

```
123 Ben 12.5 8.1 7.6 3.2
456 Greg 4.0 11.6 6.5 2.7 12
789 Victoria 8.0 8.0 8.0 8.0 7.5
```

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

```
Ben (ID#123) worked 31.4 hours (7.85 hours/day)
Greg (ID#456) worked 36.8 hours (7.36 hours/day)
Victoria (ID#789) worked 39.5 hours (7.90 hours/day)
```

Hours answer (flawed)

```
// 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

```
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"
```

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 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 Ben 12.5 8.1 7.6 3.2
456 Greg 4.0 11.6 6.5 2.7 12
789 Victoria 8.0 8.0 8.0 8.0 7.5
```

  - Recall, it should produce the following output:

```
Ben (ID#123) worked 31.4 hours (7.85 hours/day)
Greg (ID#456) worked 36.8 hours (7.36 hours/day)
Victoria (ID#789) worked 39.5 hours (7.90 hours/day)
```

Hours answer, corrected

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

IMDb movies problem

- Consider the following Internet Movie Database (IMDb) data:

```
1 9.1 196376 The Shawshank Redemption (1994)
2 9.0 139085 The Godfather: Part II (1974)
3 8.8 81507 Casablanca (1942)
```

- Write a program that displays any movies containing a phrase:

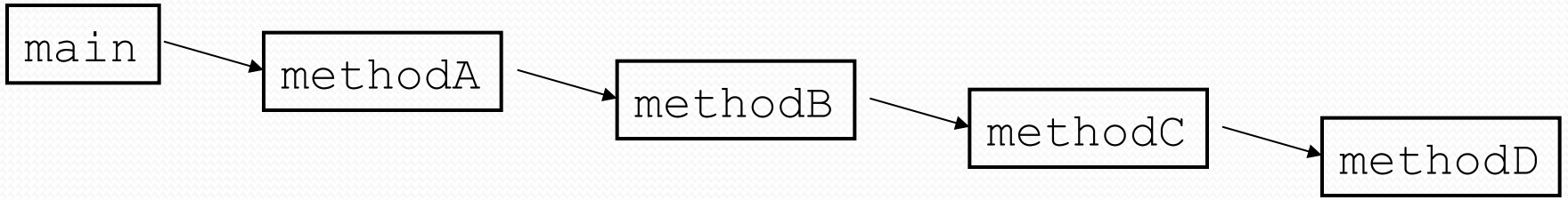
```
Search word? part

Rank    Votes   Rating  Title
2       139085  9.0     The Godfather: Part II (1974)
40      129172  8.5     The Departed (2006)
95      20401   8.2     The Apartment (1960)
192     30587   8.0     Spartacus (1960)
4 matches.
```

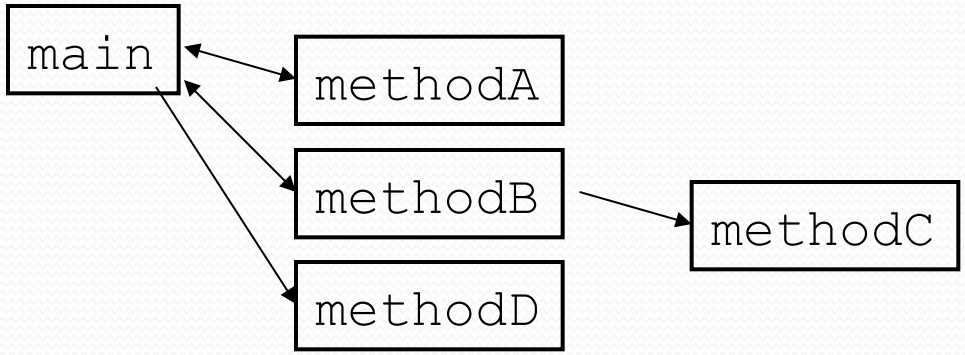
  - Is this a token or line-based problem?

"Chaining"

- `main` should be a concise summary of your program.
  - It is bad if each method calls the next without ever returning (we call this *chaining*):



- A better structure has `main` make most of the calls.
  - Methods must return values to `main` to be passed on later.



Bad IMDb "chained" code 1

```
// Displays IMDB's Top 250 movies that match a search string.
import java.io.*;     // for File
import java.util.*;   // for Scanner

public class Movies {
    public static void main(String[] args) throws FileNotFoundException {
        getWord();
    }

    // Asks the user for their search word and returns it.
    public static void getWord() throws FileNotFoundException {
        System.out.print("Search word: ");
        Scanner console = new Scanner(System.in);
        String searchWord = console.next();
        searchWord = searchWord.toLowerCase();
        System.out.println();

        Scanner input = new Scanner(new File("imdb.txt"));
        search(input, searchWord);
    }
    ...
```

Bad IMDb "chained" code 2

```
    ...

    // Breaks apart each line, looking for lines that match the search word.
    public static String search(Scanner input, String searchWord) {
        int matches = 0;
        while (input.hasNextLine()) {
            String line = input.nextLine();
            String lineLC = line.toLowerCase();     // case-insensitive match
            if (lineLC.indexOf(searchWord) >= 0) {
                matches++;
                System.out.println("Rank\tVotes\tRating\tTitle");
                display(line);
            }
        }

        System.out.println(matches + " matches.");
    }

    // Displays the line in the proper format on the screen.
    public static void display(String line) {
        Scanner lineScan = new Scanner(line);
        int rank = lineScan.nextInt();
        double rating = lineScan.nextDouble();
        int votes = lineScan.nextInt();
        String title = "";
        while (lineScan.hasNext()) {
            title += lineScan.next() + " ";    // the rest of the line
        }
        System.out.println(rank + "\t" + votes + "\t" + rating + "\t" + title);
    }
}
```

Better IMDb answer 1

```
// Displays IMDB's Top 250 movies that match a search string.
import java.io.*;     // for File
import java.util.*;   // for Scanner

public class Movies {
    public static void main(String[] args) throws FileNotFoundException {
        String searchWord = getWord();
        Scanner input = new Scanner(new File("imdb.txt"));
        String line = search(input, searchWord);

        if (line.length() > 0) {
            System.out.println("Rank\tVotes\tRating\tTitle");
            while (line.length() > 0) {
                display(line);
                line = search(input, searchWord);
            }
        }

        System.out.println(matches + " matches.");
    }

    // Asks the user for their search word and returns it.
    public static String getWord() {
        System.out.print("Search word: ");
        Scanner console = new Scanner(System.in);
        String searchWord = console.next();
        searchWord = searchWord.toLowerCase();
        System.out.println();
        return searchWord;
    }
    ...
```

Better IMDb answer 2

```
...

    // Breaks apart each line, looking for lines that match the search word.
    public static String search(Scanner input, String searchWord) {
        while (input.hasNextLine()) {
            String line = input.nextLine();
            String lineLC = line.toLowerCase();     // case-insensitive match
            if (lineLC.indexOf(searchWord) >= 0) {
                return line;
            }
        }
        return "";   // not found
    }

    // Displays the line in the proper format on the screen.
    public static void display(String line) {
        Scanner lineScan = new Scanner(line);
        int rank = lineScan.nextInt();
        double rating = lineScan.nextDouble();
        int votes = lineScan.nextInt();
        String title = "";
        while (lineScan.hasNext()) {
            title += lineScan.next() + " ";    // the rest of the line
        }
        System.out.println(rank + "\t" + votes + "\t" + rating + "\t" + title);
    }
}
```