Building Java Programs

Chapter 6: File Processing

Lecture outline

- line-based file processing using `Scanners`
  - processing a file line by line
  - mixing line-based and token-based file processing
  - searching for a particular line record in a file
  - graphically displaying data from a file

- complex file input
  - mixing `nextLine` and token-based methods

Line-based file processing

reading: 6.3

Line-by-line processing

- A `Scanner` object has the following methods:

Method	**Description**
`nextLine()`	returns the next entire line of input
`hasNextLine()`	returns `true` if there are any more <u>lines</u> of input to read *(always true for console input)*

- The `Scanner`'s `nextLine` method reads a line of input.
  - It consumes from the input cursor's position to the next `\n` .

```
Scanner input = new Scanner(new File("<file name>"));
while (input.hasNextLine()) {
    String line = input.nextLine();
    <process this line>;
}
```

Line input example

- Given the following input data:

```
23    3.14 John Smith    "Hello world"
         45.2      19
```

- The Scanner can read the following input:

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

  - *`input.nextLine()`*

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

  - *`input.nextLine()`*

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

  - Each \n character is consumed but not returned.

File processing question

- A program that "quotes" a text file's email message:

Example input `message.txt` :	Example output:
`Please tell the students`	`> Please tell the students`
`I'll be curving the grades`	`> I'll be curving the grades`
`downward!`	`> downward!`
	`>`
`   Love, Prof. Meanie`	`>    Love, Prof. Meanie`

```
import java.io.*;     // for File
import java.util.*;   // for Scanner

public class QuoteMessage {
    public static void main(String[] args)
            throws FileNotFoundException {
        Scanner input = new Scanner(new File("message.txt"));
        while (input.hasNextLine()) {
            String line = input.nextLine();
            System.out.println("> " + line);
        }
    }
}
```

IMDb movies problem

- Consider the following Internet Movie Database (IMDb) Top-250 data from a file `imdb.txt` in this format, with rankings and votes:

```
1 9.1 196376 The Shawshank Redemption (1994)
2 8.9 93064 The Godfather: Part II (1974)
3 8.8 81507 Casablanca (1942)
```

- Write a program that prompts the user for a search phrase and displays any movies that contain that phrase.

```
Search word? part

Rank    Votes   Rating  Title
3       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-based problem, or a line-based problem?

A good start

```
// 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"));

        while (input.hasNextLine()) {
            // search for lines that match the search word
            String line = input.nextLine();
            if (line.indexOf(searchWord) >= 0) {
                System.out.println(line);
            }
        }
    }

    // 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;
    }
}
```

Flaws with our solution

- Problems with our solution:
  - It is case-sensitive.
  - It doesn't count the number of matches.
  - The output format for each line is incorrect.

- Observations:
  - We care about the line breaks (they separate movies), but we also want to break apart the tokens up to reformat each line.

  - The best solution is a hybrid approach:
    - Break the overall input into lines.
    - Break each line into tokens.

Tokenizing lines

- A `Scanner` can tokenize the contents of a `String`.

```
Scanner <name> = new Scanner(<String>);
```

- We can use String `Scanner`s to process each line of a file.

```
Scanner input = new Scanner(new File("<file name>"));
while (input.hasNextLine()) {
    String line = input.nextLine();
    Scanner lineScan = new Scanner(line);
    <process the tokens of this line>;
}
```

Line processing example

- Example: Count 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);
    int count = 0;
    while (lineScan.hasNext()) {
        String word = lineScan.next();
        count++;
    }
    System.out.println("Line has " + count + " words");
}
```

Input file `input.txt`:	Output to console:

`The quick brown fox jumps over`
`the lazy dog.`
	`Line has 6 words`
	`Line has 3 words`

IMDb revisited

- Fix our IMDB program's behavior:
  - Make it case-insensitive.
  - Make it count the matches.
  - Make it format the output correctly as shown below.
  - Break the program better into methods.

```
Search word? part

Rank    Votes   Title
3       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.
```

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);

        int matches = 0;
        if (line.length() > 0) {
            System.out.println("Rank\tVotes\tRating\tTitle");
            while (line.length() > 0) {
                matches++;
                display(line, matches);
                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;
    }
    ...
```

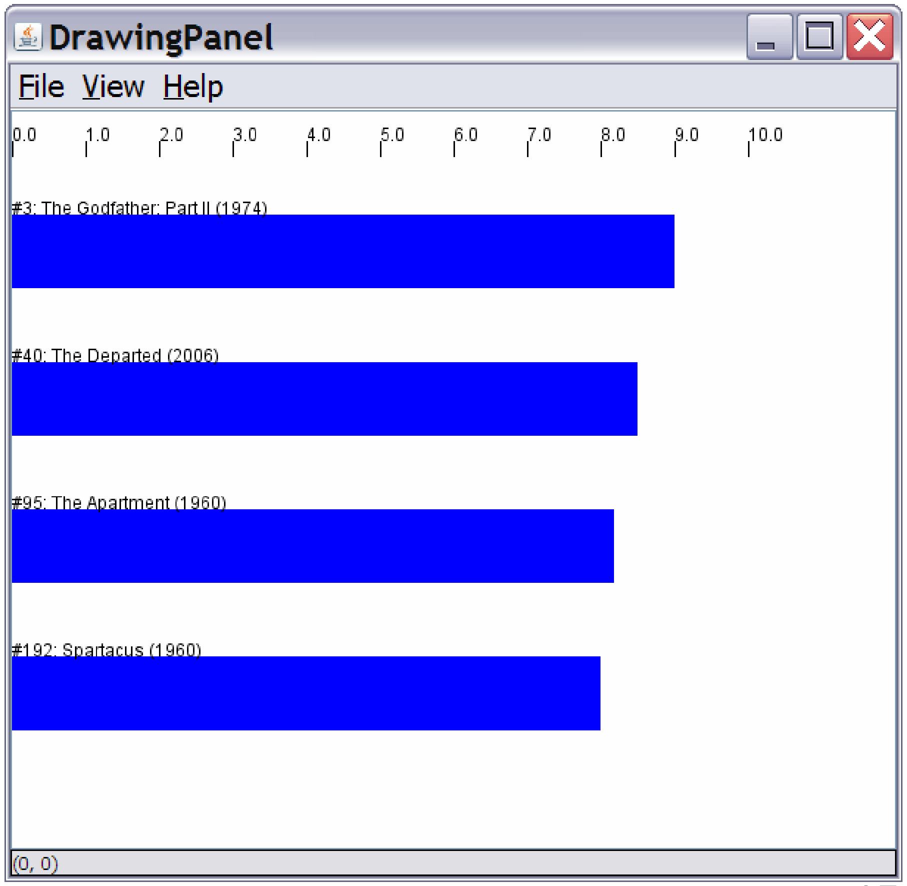
IMDb answer 2

```
    ...

    // 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, int matches) {
        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);
    }
}
```

Graphical IMDB problem

- Turn our IMDb code into a graphical program.
  - top-left 0.0 tick mark at (0, 20)
  - ticks 10px tall, 50px apart

  - first blue bar top/left corner at (0, 70)
  - bars 50px tall
  - bars 50px wide per rating point
  - bars 100px apart vertically



Mixing graphical, text output

- When solving complex file I/O problems with a mix of text and graphical output, attack the problem in pieces.

  Do the text input/output and file I/O first:

  - Display any welcome message and initial console input.
  - Open the input file and print some file data.
    (Perhaps print every line, the first token of each line, etc.)
  - Search the input file for the proper line record(s).

  Next, begin the graphical output:

  - Draw any fixed items that do not depend on the file results.
  - Draw the graphical output that depends on the search result.

Graphical IMDb answer 1

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

public class Movies2 {
    public static void main(String[] args) throws FileNotFoundException {
        String searchWord = getWord();
        Scanner input = new Scanner(new File("imdb.txt"));
        String line = search(input, searchWord);

        int matches = 0;
        if (line.length() > 0) {
            System.out.println("Rank\tVotes\tRating\tTitle");
            Graphics g = createWindow();
            while (line.length() > 0) {
                matches++;
                display(g, line, matches);
                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;
    }
    ...
```

Graphical 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(Graphics g, String line, int matches) {
    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);
    drawBar(g, matches, title, rank, rating);
}

...
```

Graphical IMDb answer 3

```
    ...

    // Creates a drawing panel and draws all fixed graphics.
    public static Graphics createWindow() {
        DrawingPanel panel = new DrawingPanel(600, 500);
        Graphics g = panel.getGraphics();

        for (int i = 0; i <= 10; i++) {      // draw tick marks
            int x = i * 50;
            g.drawLine(x, 20, x, 30);
            g.drawString(i + ".0", x, 20);
        }

        return g;
    }

    // Draws one red bar representing a movie's votes and ranking.
    public static void drawBar(Graphics g, int matches, String title,
                               int rank, double rating) {
        int y = 70 + 100 * (matches - 1);
        int w = (int) (rating * 50);
        int h = 50;

        g.setColor(Color.BLUE);   // draw the blue bar for that movie
        g.fillRect(0, y, w, h);
        g.setColor(Color.BLACK);
        g.drawString("#" + rank + ": " + title, 0, y);
    }
}
```

Another example: Hours Worked

reading: 6.2 - 6.3

Another example

- 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 Jenn 8.0 8.0 8.0 8.0 7.5
```

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

```
Susan (ID#123) worked 31.4 hours (7.85 hours/day)
Brad (ID#456) worked 36.8 hours (7.36 hours/day)
Jenn (ID#789) worked 39.5 hours (7.9 hours/day)
```

- Let's try to solve this problem token-by-token ...

A flawed solution

```
import java.io.*;                  // for File
import java.util.*;                // for Scanner

public class HoursWorked {         // a non-working solution
    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)");
        }
    }
}
```

The flaw

- Flawed solution's output:

```
Susan (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 java.util.Scanner.nextInt(Scanner.java:2050)
        at HoursWorked.main(HoursBad.java:9)
```

  - The inner `while` loop is grabbing the next person's ID.

- Observations:
  - We need to process the individual tokens, but we also care about the line breaks (they tell us when one person is done).
  - The best solution is a hybrid approach:
    - Break the overall input into lines.
    - Break each line into tokens.

Complex lines

- Fix the program to compute employee hours worked:

```
Susan (ID#123) worked 31.4 hours (7.85 hours/day)
Brad (ID#456) worked 36.8 hours (7.36 hours/day)
Jenn (ID#789) worked 39.5 hours (7.9 hours/day)
```

- Modify the program so it searches for a person by ID:
  - Example:

```
Enter an ID: 456
Brad (ID#456) worked 36.8 hours (7.36 hours/day)
```

  - Example:

```
Enter an ID: 293
ID#293 not found
```

Complex input answer 1

```java
// This program searches an input file of employees' hours worked
// for a particular employee and outputs that employee's hours data.

import java.io.*;    // for File
import java.util.*;  // for Scanner

public class HoursWorked {
    public static void main(String[] args) throws FileNotFoundException {
        Scanner console = new Scanner(System.in);
        System.out.print("Enter an ID: ");
        int searchId = console.nextInt();       // e.g. 456

        Scanner input = new Scanner(new File("hours.txt"));
        String line = findPerson(input, searchId);
        if (line.length() > 0) {
            processLine(line);
        } else {
            System.out.println("ID#" + searchId + " was not found");
        }
    }

    ...
```

Complex input answer 2

```
    // Locates and returns the line of data about a particular person.
    public static String findPerson(Scanner input, int searchId) {
        while (input.hasNextLine()) {
            String line = input.nextLine();
            Scanner lineScan = new Scanner(line);
            int id = lineScan.nextInt();              // e.g. 456
            if (id == searchId) {
                return line;                          // we found them!
            }
        }
        return "";   // not found, so return an empty line
    }

    // Totals the hours worked by the person and outputs their info.
    public static void processLine(String line) {
        Scanner lineScan = new Scanner(line);
        int id = lineScan.nextInt();                  // e.g. 456
        String name = lineScan.next();                // e.g. "Brad"
        double hours = 0.0;
        int days = 0;
        while (lineScan.hasNextDouble()) {
            hours += lineScan.nextDouble();
            days++;
        }

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

Advanced File I/O

reading: 6.4 - 6.5

Confusion w/ nextLine

- Using `nextLine` in conjunction with the token-based methods on the same `Scanner` can cause odd results.
  - Given the following input:

```
23   3.14
Joe    "Hello world"
         45.2   19
```

  - You'd think that you could read the `23` and `3.14` with calls to `nextInt` and `nextDouble` respectively, and then read the following `Joe "Hello world"` part with `nextLine` . But:

```
System.out.println(input.nextInt());      // 23
System.out.println(input.nextDouble());   // 3.14
System.out.println(input.nextLine());     //
```

  - The nextLine call produces no output! Why is this?

Mixing line-based with tokens

- Here's what the Scanner does when you mix `nextLine` with the token-based methods on the same Scanner :

```
23   3.14
Joe    "Hello world"
             45.2  19


input.nextInt()                                 // 23
23\t3.14\nJoe\t"Hello world"\n\t\t45.2  19\n
  ^

input.nextDouble()                              // 3.14
23\t3.14\nJoe\t"Hello world"\n\t\t45.2  19\n
        ^

input.nextLine()                                // "" (empty!)
23\t3.14\nJoe\t"Hello world"\n\t\t45.2  19\n
          ^

input.nextLine()                       // "Joe\t\"Hello world\""
23\t3.14\nJoe\t"Hello world"\n\t\t45.2  19\n
                              ^
```

Line-and-token example

- Another example of the confusing behavior:

```
Scanner console = new Scanner(System.in);
System.out.print("Enter your age: ");
int age = console.nextInt();
System.out.print("Now enter your name: ");
String name = console.nextLine();
System.out.println(name + " is " + age + " years old.");
```

Log of execution (user input underlined):

```
Enter your age: 12
Now enter your name: Marty Stepp
 is 12 years old.
```

- Why?
  - User's overall input: `12\nMarty Stepp`
  - After `nextInt()`:

```
12\nMarty Stepp
  ^
```

  - After `nextLine()`:

```
12\nMarty Stepp
    ^
```