Topic 20
more file processing

"We can only see a short distance ahead, but we can see plenty there that needs to be done."

- Alan Turing

Recall: Line-based methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextLine()</td>
<td>returns the next entire line of input</td>
</tr>
<tr>
<td>hasNextLine()</td>
<td>returns true if there are any more lines of input to read (always true for console input)</td>
</tr>
</tbody>
</table>

- nextLine consumes from the input cursor to the next \n.

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

Recall: Tokenizing lines

- A String Scanner can tokenize each line of a file.

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

clicker Question

- What is output by the following code if the input file contains

```java
public static void print(Scanner sc) {
    int total = sc.nextInt();
    total += sc.nextInt();
    System.out.println(total + " " + sc.nextLine());
}
```

A. 36  B. 36 Christmas Eve  
C. 24 Christmas Eve  
D. no output due to syntax error  
E. no output due to runtime error
**Hours v2 question**

- Modify the Hours program to search for a person by ID:
  
  - Example:
    
    Enter an ID: **456**
    
    Eric worked 36.8 hours (7.36 hours/day)

  - Example:
    
    Enter an ID: **293**
    
    ID #293 not found

**Hours v2 answer 1**

```java
// This program searches an input file of employees' hours worked
// for a particular employee and outputs their 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");
        }
    }
}
```

**Hours v2 answer 2**

```java
// 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 String
}

// 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 + " worked " + hours + " hours (" + (hours / days) + " hours/day")
}
```

**IMDb movies problem**

- Consider the following Internet Movie Database (IMDb) data:
  
  1 9.1 490,400 The Shawshank Redemption (1994)
  2 9.1 392,937 The Godfather (1972)

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

  ```java
  Search word? part
  Rank Votes Rating Title
  3 232741 9.0 The Godfather: Part II (1974)
  50 249709 8.4 The Departed (2006)
  98 34736 8.3 The Apartment (1960)
  241 48525 7.9 Spartacus (1960)
  4 matches.
  ```
Recall "Chaining"

- main should be a concise summary of your program.
  - It is generally poor style if each method calls the next without ever returning (chaining):

```
main → methodA → methodB → methodC → methodD
```

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

```
main → methodA → methodB → methodC
```

---

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();
        // case-insensitive match
        if (line.toLowerCase().indexOf(searchWord) > 0) {
            matches++;
            System.out.println("Rank\tVotes\tRating\tTitle");
        }
    }
    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);
        int matches = 0;
        if (line.length() > 0) {
            System.out.println("Rank\tVotes\tRating\tTitle");
            while (line.length() > 0) {
                matches++;
                display(line);
                line = search(input, searchWord);
            }
        System.out.println(matches + " matches.");
    }
```
Better IMDb answer 2

// 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;
}

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

Better IMDb answer 3

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

Mixing tokens and lines

- Using nextLine in conjunction with the token-based methods on the same Scanner can cause unexpected results.

  23  3.14
  Joe  "Hello world"
        45.2  19

- You'd think you could read 23 and 3.14 withnextInt and
  nextDouble, then read Joe "Hello world" with
  nextLine.

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

  - But the nextLine call produces no output! Why?

Mixing lines and tokens

- Avoid reading both tokens and lines from the same Scanner:

  23  3.14
  Joe  "Hello world"
        45.2  19

  input.nextInt();
  // 23

  23\t3.14\n  Joe\t"Hello world"
        45.2  19

  input.nextDouble();
  // 3.14

  input.nextLine();
  // "" (empty!)

  input.nextLine();
  // "Joe\t"Hello world"

  input.nextLine();
  // "Hello world"
Line-and-token example

```java
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: Sideshow Bob
is 12 years old.

Why?

- Overall input: 12
- After `nextInt()`:
  - 12
  - Sideshow Bob
- After `nextLine()`:
  - 12
  - Sideshow Bob