Building Java Programs

Chapter 6

Lecture 6-1: File Input with Scanner

reading: 6.1 - 6.2, 5.3

self-check: Ch. 6 #1-6

exercises: Ch. 6 #5-7

videos: Ch. 6 #1-2

Input/output (I/O)

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

- Create a `File` object to get info about a file on disk.

  (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

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, better yet:

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

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`

Compiler error w/ files

- The following program does not compile:

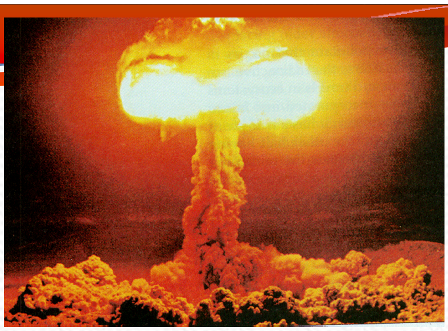
```
import java.io.*;      // for File
import java.util.*;    // for Scanner

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 following error occurs:

```
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 `charAt` 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.

- 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 it 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 `numbers.txt` that contains this text:

```
308.2
   14.9 7.4  2.8


3.9 4.7    -15.4
    2.8
```

- A `Scanner` views all input as a stream of characters:

```
308.2\n   14.9 7.4  2.8\n\n3.9 4.7    -15.4\n  2.8\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.

```
308.2\n   14.9 7.4  2.8\n\n3.9 4.7   -15.4\n  2.8\n
^
```

```
double x = input.nextDouble();    // 308.2
308.2\n   14.9 7.4  2.8\n\n3.9 4.7   -15.4\n  2.8\n
     ^
```

```
String s = input.next();          // "14.9"
308.2\n   14.9 7.4  2.8\n\n3.9 4.7   -15.4\n  2.8\n
              ^
```

File input question

- Recall the input file `numbers.txt`:

```
308.2
   14.9 7.4  2.8


3.9 4.7    -15.4
    2.8
```

- Write a program that reads the first 5 values from the file and prints them along with their sum.

```
number = 308.2
number = 14.9
number = 7.4
number = 2.8
number = 3.9
Sum = 337.2
```

File input answer

```
// Displays the first 5 numbers in the given file,
// and displays their sum at the end.

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

public class Echo {
    public static void main(String[] args)
            throws FileNotFoundException {
        Scanner input = new Scanner(new File("numbers.txt"));
        double sum = 0.0;
        for (int i = 1; i <= 5; i++) {
            double next = input.nextDouble();
            System.out.println("number = " + next);
            sum = sum + next;
        }
        System.out.printf("Sum = %.1f\n", sum);
    }
}
```

Scanner exceptions

- `InputMismatchException`
  - You read the wrong type of token (e.g. read `"hi"` as `int`).

- `NoSuchElementException`
  - You read past the end of the input.

- 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 CountTokens.sillyMethod(CountTokens.java:19)
    at CountTokens.main(CountTokens.java:6)
```

Reading an entire file

- Suppose we want our program to process the entire file.
  (It should work no matter how many values are in the file.)

```
number = 308.2
number = 14.9
number = 7.4
number = 2.8
number = 3.9
number = 4.7
number = -15.4
number = 2.8
Sum = 329.3
```

Testing for valid input

- `Scanner` methods to see what the next token will be:

Method	Description
`hasNext()`	returns `true` if there are any more tokens of input to read *(always true for console input)*
`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 do not consume input; they just give information about the next token.
  - Useful to see what input is coming, and to avoid crashes.

Using hasNext methods

- To avoid exceptions:

```
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.");
}
```

- To detect the end of a file:

```
Scanner input = new Scanner(new File("example.txt"));
while (input.hasNext()) {
    String token = input.next();    // will not crash!
    System.out.println("token: " + token);
}
```

File input question 2

- Modify the `Echo` program to process the entire file:
  (It should work no matter how many values are in the file.)

```
number = 308.2
number = 14.9
number = 7.4
number = 2.8
number = 3.9
number = 4.7
number = -15.4
number = 2.8
Sum = 329.3
```

File input answer 2

```
// Displays each number in the given file,
// and displays their sum at the end.

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

public class Echo {
    public static void main(String[] args)
            throws FileNotFoundException {
        Scanner input = new Scanner(new File("numbers.txt"));
        double sum = 0.0;
        while (input.hasNextDouble()) {
            double next = input.nextDouble();
            System.out.println("number = " + next);
            sum = sum + next;
        }
        System.out.printf("Sum = %.1f\n", sum);
    }
}
```

File input question 3

- Modify the Echo 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, numbers2.txt:

```
308.2  hello
   14.9 7.4  bad stuff   2.8

3.9 4.7  oops  -15.4
:-)    2.8  @#*($&
```

File input answer 3

```
// Displays each number in the given file,
// and displays their sum at the end.
import java.io.*;     // for File
import java.util.*;   // for Scanner
public class Echo2 {
    public static void main(String[] args)
            throws FileNotFoundException {
        Scanner input = new Scanner(new File("numbers2.txt"));
        double sum = 0.0;
        while (input.hasNext()) {
            if (input.hasNextDouble()) {
                double next = input.nextDouble();
                System.out.println("number = " + next);
                sum = sum + next;
            } else {
                input.next();   // throw away the bad token
            }
        }
        System.out.printf("Sum = %.1f\n", sum);
    }
}
```

Election question

- Write a program that reads a file `poll.txt` of poll data.
  - Format: *State Obama% McCain% ElectoralVotes Pollster*

```
CT 56 31 7 Oct U. of Connecticut
NE 37 56 5 Sep Rasmussen
AZ 41 49 10 Oct Northern Arizona U.
```

- The program should print how many electoral votes each candidate leads in, and who is leading overall in the polls.

```
Obama: 214 votes
McCain: 257 votes
```

Election answer

```
// Computes leader in presidential polls, based on input file such as:
// AK 42 53 3 Oct Ivan Moore Research

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

public class Election {
    public static void main(String[] args) throws FileNotFoundException {
        Scanner input = new Scanner(new File("polls.txt"));
        int obamaVotes = 0, mccainVotes = 0;

        while (input.hasNext()) {
            if (input.hasNextInt()) {
                int obama = input.nextInt();
                int mccain = input.nextInt();
                int eVotes = input.nextInt();
                if (obama > mccain) {
                    obamaVotes = obamaVotes + eVotes;
                } else if (mccain > obama) {
                    mccainVotes = mccainVotes + eVotes;
                }
            } else {
                input.next();   // skip non-integer token
            }
        }

        System.out.println("Obama: " + obamaVotes + " votes");
        System.out.println("McCain: " + mccainVotes + " votes");
    }
}
```

Line-based file processing

reading: 6.3

self-check: #7-11

exercises: #1-4, 8-11

Hours question

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

```
123 Kim 12.5 8.1 7.6 3.2
456 Brad 4.0 11.6 6.5 2.7 12
789 Stef 8.0 8.0 8.0 8.0 7.5
```

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

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

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

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

```
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 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 the next entire line of input
`hasNextLine()`	returns `true` if there are any more lines of input to read (always true for console input)

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

```
Scanner input = new Scanner(new File("file name"));
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
  ```

Tokenizing lines of a file

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 Kim 12.5 8.1 7.6 3.2
456 Brad 4.0 11.6 6.5 2.7 12
789 Stef 8.0 8.0 8.0 8.0 7.5
```

  - Recall, it should produce the following output:

```
Kim (ID#123) worked 31.4 hours (7.85 hours/day)
Brad (ID#456) worked 36.8 hours (7.36 hours/day)
Stef (ID#789) worked 39.5 hours (7.9 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();
            Scanner lineScan = new Scanner(line);
            int id = lineScan.nextInt();           // e.g. 456
            String name = lineScan.next();         // e.g. "Brad"
            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)");
        }
    }
}
```

Hours v2 question

- Modify the `Hours` program to search for a person by ID:
  - Example:

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

  - Example:

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

Hours v2 answer 1

```
// 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");
        }
    }

    ...
```

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

Building Java Programs

Chapter 6

Lecture 6-3: Searching Files

reading: 6.3, 6.5

Recall: Line-based methods

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

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

```
Scanner input = new Scanner(new File("file name"));
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("file name"));
while (input.hasNextLine()) {
    String line = input.nextLine();
    Scanner lineScan = new Scanner(line);

    process the contents of this line...;
}
```

Hours v2 question

- Modify the `Hours` program to search for a person by ID:
  - Example:

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

  - Example:

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

Hours v2 answer 1

```
// 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");
        }
    }

    ...
```

Hours v2 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 + " worked " + hours + " hours ("
            + (hours / days) + " 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:

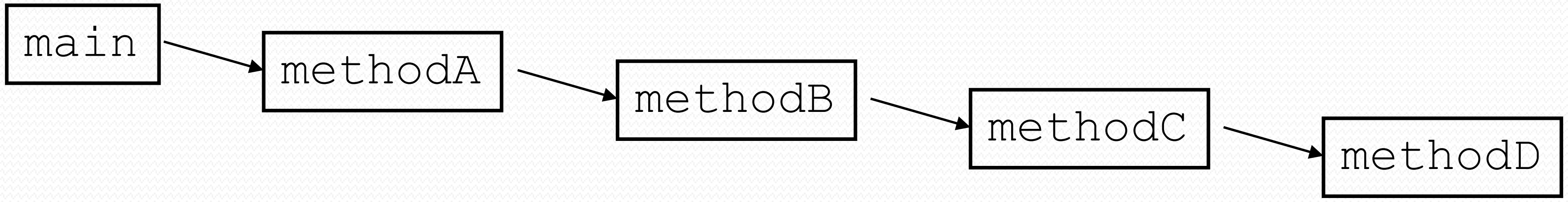
```
Search word? part

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.
```

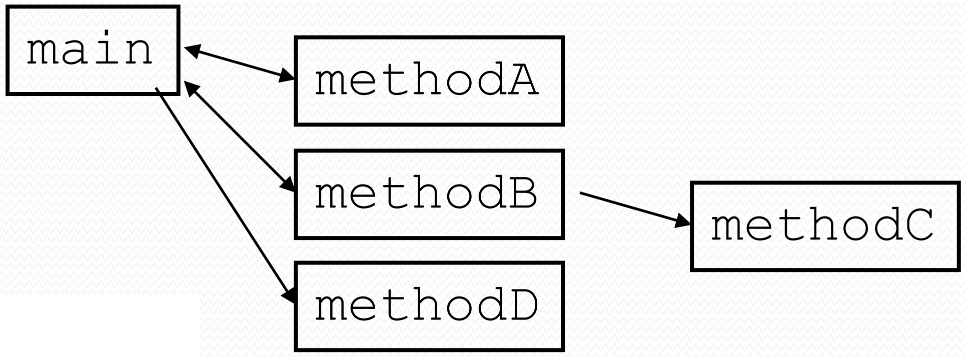
  - Is this a token or line-based problem?

"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

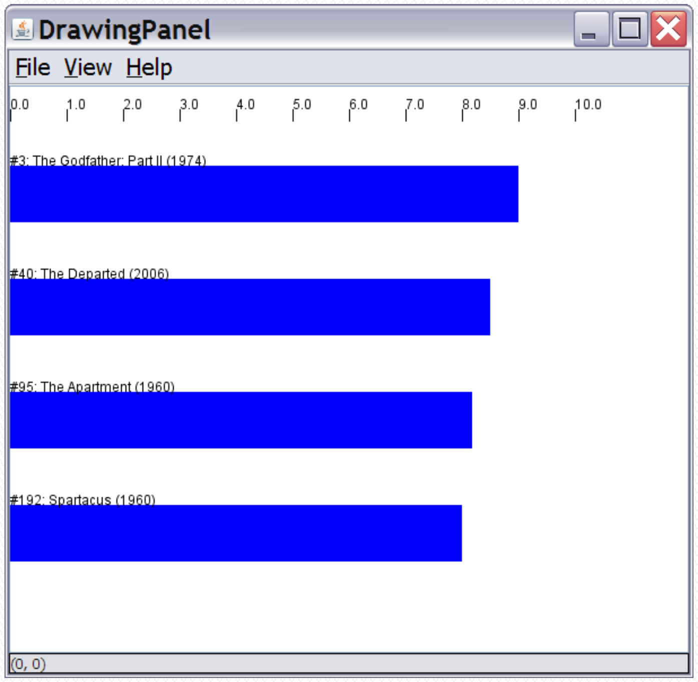
```
    ...

    // 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);
    }
}
```

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 graphics and text

- When mixing text/graphics, solve the problem in pieces.

  Do the text 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).

  Lastly, add the graphical output:
  - Draw any fixed graphics that do not depend on the file data.
  - Draw the graphics that do depend 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

```java
...
// 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);
}
}
```

Mixing tokens and lines

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

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

  - You'd think you could read `23` and `3.14` with `nextInt` 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

- Don't read both tokens and lines from 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

```
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\nSideshow Bob`
  - After `nextInt()`:
    ```
    12\nSideshow Bob
      ^
    ```
  - After `nextLine()`:
    ```
    12\nSideshow Bob
        ^
    ```