

Building Java Programs

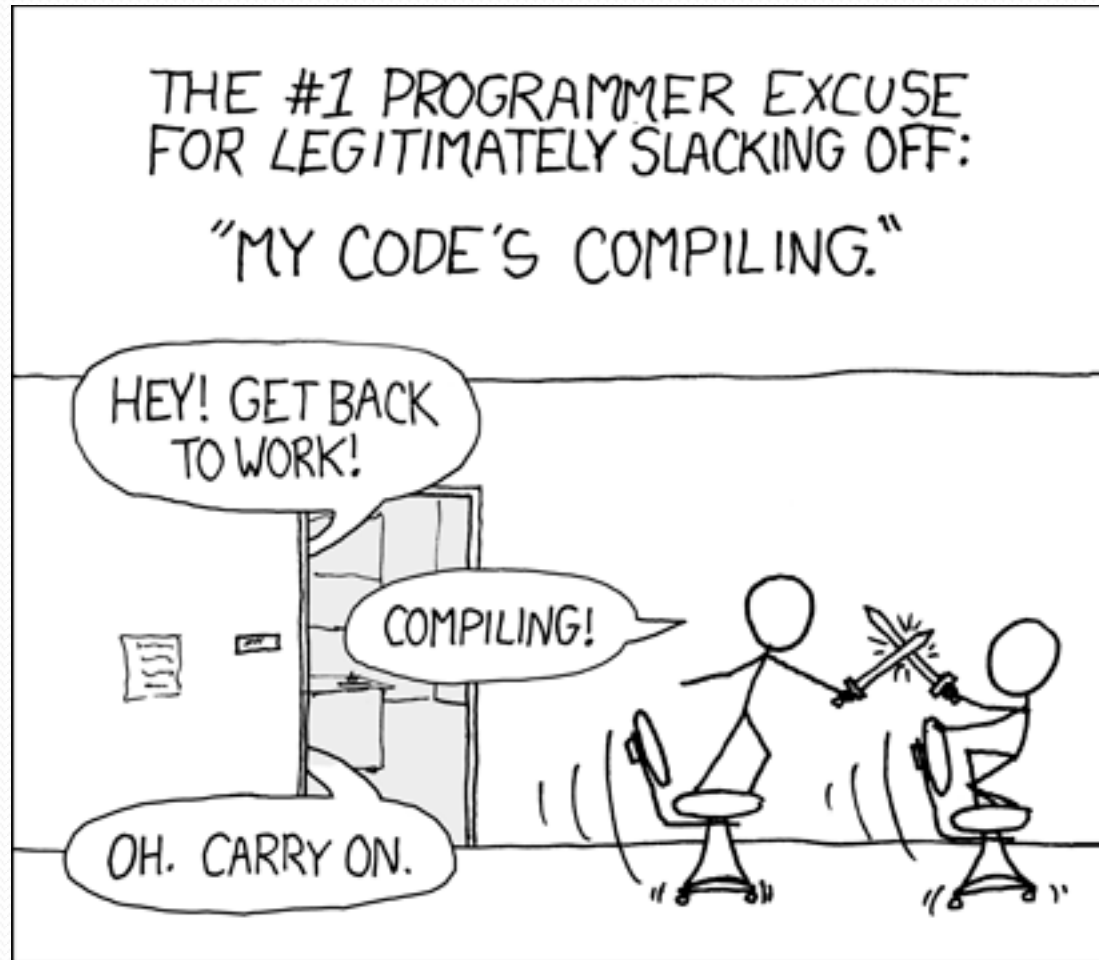
Chapter 6

Lecture 6-3: Searching Files

reading: 6.3 - 6.5

THE #1 PROGRAMMER EXCUSE
FOR LEGITIMATELY SLACKING OFF:

"MY CODE'S COMPILING."



Recall: Line-based methods

Method	Description
<code>nextLine()</code>	returns the next entire line of input
<code>hasNextLine()</code>	returns <code>true</code> 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("<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>;
}
```

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) {                 // we found them!  
            return line;  
        }  
    }  
    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  9.2  590,634  The Shawshank Redemption (1994)
2  9.2  457,173  The Godfather (1972)
3  9.0  279,059  The Godfather: Part II (1974)
```

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

Search word: part

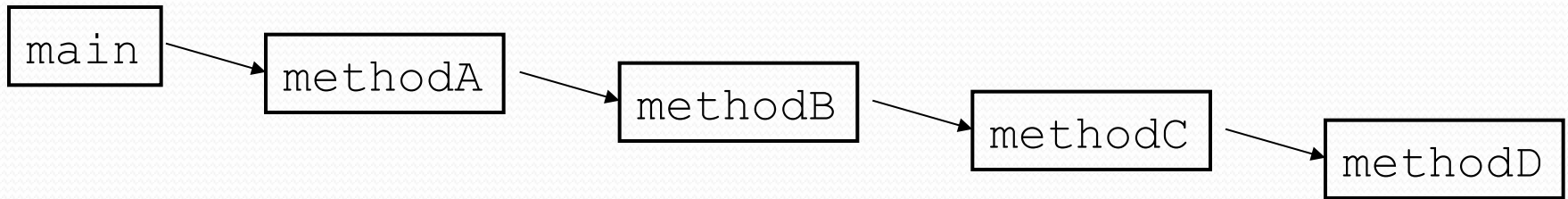
```
Rank      Votes      Rating  Title
3         279059     9.0     The Godfather: Part II (1974)
58        300640     8.4     The Departed (2006)
91        41117      8.3     The Apartment (1960)
```

3 matches.

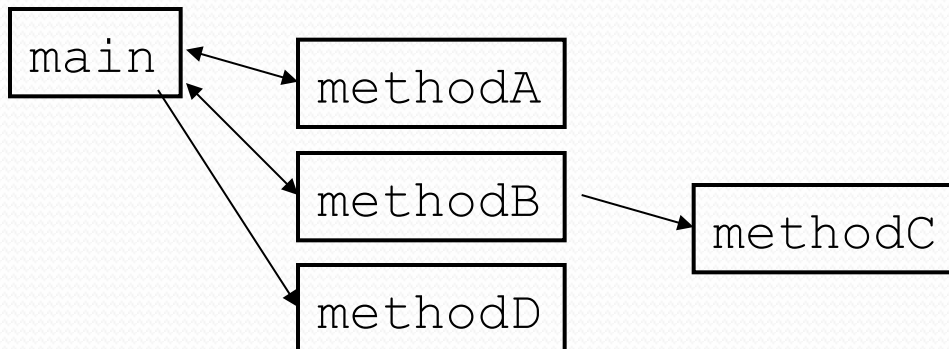
- Is this a token or line-based problem?

"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();
        Scanner lineScan = new Scanner(line);
        int rank = lineScan.nextInt();
        double rating = lineScan.nextDouble();
        int votes = lineScan.nextInt();
        String title = lineScan.nextLine(); // all the rest

        if (title.toLowerCase().indexOf(searchWord) >= 0) {
            matches++;
            System.out.println("Rank\tVotes\tRating\tTitle");
            display(line);
        }
    }
    System.out.println(matches + " matches.");
}
```

Bad IMDb "chained" code 3

...

```
// 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.");
    }

    // 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();
        Scanner lineScan = new Scanner(line);
        int rank = lineScan.nextInt();
        double rating = lineScan.nextDouble();
        int votes = lineScan.nextInt();
        String title = lineScan.nextLine(); // all the rest

        if (title.toLowerCase().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);
}
}
```

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
 ^