file input

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

- Alan Turing

Based on slides by Mike Scott and Chand John.

Used with permission.

https://www.cs.utexas.edu/~scottm/cs312/

https://www.cs.utexas.edu/~chand/cs312/

Based on slides by Marty Stepp and Stuart Reges
from http://www.buildingjavaprograms.com/

File Input/output (I/O)

```
import java.io.File;
```

- Create a `File` object to get info about a file on your drive.
  - (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

Reading files

- To read a file, pass a `File` object to the `Scanner` Constructor (instead of `System.in`).

```
Scanner <name> = new Scanner(new File("<filename>"));
```

  - Example:

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

  - or (shorter):

```
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` , the `Scanner` will look for `H:/hw6/data/readme.txt`

Working Directory

- New programmers are often not sure what their current directory is.
- Easy to print out:

```
public static void
printWorkingDirectory() {
    System.out.println("Working Directory = " +
        System.getProperty("user.dir"));
}
```

```
Working Directory = C:\Users\scottm\Documents\312\BlueJ
```

http://docs.oracle.com/javase/tutorial/essential/environment/sysprop.html

Compiler error w/ files

```
import java.io.File;
import java.util.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 program fails to compile with the following error:

```
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 `substring` 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 a runtime 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.

Clicker 1

- Can a programmer prevent a divide by zero error from occurring in all cases?

A. no

B. yes

C. maybe

Clicker 2

- Can a programmer prevent a file not found error from occurring in all cases?

A. no

B. yes

C. maybe

The `throws` clause

- **`throws` clause**: Keywords on a method's header that state that it may generate an exception (and will not handle it).
- Syntax:

```
public static <type> <name>(...) 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 this happens. OR I am passing the buck to someone else.*

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 `weather.txt` that contains this text:

```
16.2   23.5
   19.1 7.4  22.8

18.5  -1.8 14.9

```

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

```
16.2   23.5\n19.1 7.4  22.8\n\n18.5  -1.8 14.9\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

```
16.2   23.5\n19.1 7.4  22.8\n\n18.5  -1.8 14.9\n
^
```

```
double d = input.nextDouble();     // 16.2
```

```
16.2   23.5\n19.1 7.4  22.8\n\n18.5  -1.8 14.9\n
    ^
```

```
String s = input.next();           // "23.5"
```

```
16.2   23.5\n19.1 7.4  22.8\n\n18.5  -1.8 14.9\n
           ^
```

File input question

- Recall the input file `weather.txt`:

```
16.2   23.5
   19.1 7.4  22.8

18.5  -1.8 14.9
```

- Write a program that prints the change in temperature between each pair of neighboring days.

```
16.2 to 23.5, change = 7.3
23.5 to 19.1, change = -4.4
19.1 to 7.4, change = -11.7
7.4 to 22.8, change = 15.4
22.8 to 18.5, change = -4.3
18.5 to -1.8, change = -20.3
-1.8 to 14.9, change = 16.7
```

File input answer

```
// Displays changes in temperature from data in an input file.
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Temperatures {
    public static void main(String[] args)
            throws FileNotFoundException {
        Scanner input = new Scanner(new File("weather.txt"));
        double prev = input.nextDouble();   // fencepost
        for (int i = 1; i <= 7; i++) {
            double next = input.nextDouble();
            System.out.println(prev + " to " + next +
                    ", change = " + (next - prev));
            prev = next;
        }
    }
}
```

Reading an entire file

- Suppose we want our program to work no matter how many numbers are in the file.
  - Currently, if the file has more numbers, they will not be read.
  - If the file has fewer numbers, what will happen?

A crash! Example output from a file with just 3 numbers:

```
16.2 to 23.5, change = 7.3
23.5 to 19.1, change = -4.4
Exception in thread "main"
  java.util.NoSuchElementException
    at java.util.Scanner.throwFor(Scanner.java:838)
    at java.util.Scanner.next(Scanner.java:1347)
    at Temperatures.main(Temperatures.java:12)
```

`Scanner` exceptions

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

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

- 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 MyProgram.myMethodName(MyProgram.java:19)
    at MyProgram.main(MyProgram.java:6)
```

`Scanner` tests for valid input

Method	Description
`hasNext()`	returns `true` if there is a next token
`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 of the `Scanner` do not consume input; they just give information about what the next token will be.
  - Useful to see what input is coming, and to avoid crashes.
  - These methods can be used with a console `Scanner`, as well.
    - When called on the console, they sometimes pause waiting for input.

Using `hasNext` methods

- Avoiding type mismatches:

```
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.");
}
```

- Avoiding reading past the end of a file:

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

File input question 2

- Modify the temperature program to process the entire file, regardless of how many numbers it contains.
  - Example: If a ninth day's data is added, output might be:

```
16.2 to 23.5, change = 7.3
23.5 to 19.1, change = -4.4
19.1 to 7.4, change = -11.7
7.4 to 22.8, change = 15.4
22.8 to 18.5, change = -4.3
18.5 to -1.8, change = -20.3
-1.8 to 14.9, change = 16.7
14.9 to 16.1, change = 1.2
```

File input answer 2

```
// Displays changes in temperature from data in an input file.
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Temperatures {
    public static void main(String[] args)
            throws FileNotFoundException {
        Scanner input = new Scanner(new File("weather.txt"));
        double prev = input.nextDouble();   // fencepost
        while (input.hasNextDouble()) {
            double next = input.nextDouble();
            System.out.println(prev + " to " + next +
                    ", change = " + (next - prev));
            prev = next;
        }
    }
}
```

File input question 3

- Modify the temperature 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, `weather2.txt`:

```
16.2   23.5
Tuesday   19.1   Wed 7.4   THURS. TEMP: 22.8

18.5  -1.8  <-- MIKE here is my data!  --Chris
   14.9 :-)
```

  - You may assume that the file begins with a double.
  - **What if we didn't know that?**

File input answer 3

```
// Displays changes in temperature from data in an input file.
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Temperatures2 {
    public static void main(String[] args)
            throws FileNotFoundException {
        Scanner input = new Scanner(new File("weather.txt"));
        double prev = input.nextDouble();   // fencepost
        while (input.hasNext()) {
            if (input.hasNextDouble()) {
                double next = input.nextDouble();
                System.out.println(prev + " to " + next +
                        ", change = " + (next - prev));
                prev = next;
            } else {
                input.next();  // throw away unwanted token
            }
        }
    }
}
```

"File" Input

- Reading from sources other than files is not very different
- For example we can read data from a web page about as easily as reading from a file
- Example, read stock information from a web page
  - often the hardest thing is finding the web page and the format of the url
  - once you have that the code is easy

Read HTML from a Webpage

- https://www.cnbc.com/
- Read and print out the HTML from that web page
- Could easily alter to read data from any web page or via a web API
  - Service that allows a developer to read data via the web

Display HTML

```
try {
      System.out.println("Raw HTML from CNBC");
      String urlAsString = "https://www.cnbc.com/";
      URL url = new URL(urlAsString);
      Scanner sc
           = new Scanner(new InputStreamReader(url.openStream()));
      while(sc.hasNextLine()) {
          System.out.println(sc.nextLine());
      }
      sc.close();
  }
  catch(IOException e) {
      System.out.println("UH OH: " + e);
  }
```

Delayed Stock Data from Yahoo! Finance

- http://finance.yahoo.com/d/quotes.csv?s
- s are the stocks symbols we want
  AAPL = Apple, INTC = Intel, MSFT = Microsoft, AMZN = Amazon
  s=AAPL+INTC+MSFT+AMZN,
- f are the fields we want for the data
  f=ca2vyrj1s
  - c = change & percent, a2 = average volume, v = day's volume, y = dividend yield, r = PE ratio, j1 = market cap, s = symbol

Complete URL

http://finance.yahoo.com/d/quotes.csv?s=AAPL+INTC+MSFT+AMZN&f=ca2vyrj1s

```
try {
  System.out.println("change and %, average volume, volume, " + "dividend yield,
      PE ratio, market cap, symbol");
  String urlAsString = "http://finance.yahoo.com/d/" +
      "quotes.csv?s=AAPL+INTC+MSFT+AMZN&f=ca2vyrj1s";
  URL url = new URL(urlAsString);
  Scanner sc = new Scanner(new InputStreamReader(url.openStream()));
  while(sc.hasNextLine()) {
    System.out.println(sc.nextLine());
  }
  sc.close();
} catch(IOException e) {
  System.out.println("UH OH: " + e);
}
```

Line based file input

Hours question

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

```
123 Kim 12.5 8.1 7.6 3.2
456 Eric 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)
Eric (ID#456) worked 36.8 hours (7.36 hours/day)
Stef (ID#789) worked 39.5 hours (7.9 hours/day)
```

Hours answer (flawed)

```
// This solution does not work!
import java.io.*;                    // for File
import java.util.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)");
        }
    }
}
```

```
123 Kim 12.5 8.1 7.6 3.2
456 Eric 4.0 11.6 6.5 2.7 12
```

Clicker 1

- What happens when the solution on the previous slide is run given a file with this data?

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

A. prints out correct answer
B. no output due to syntax error
C. some output then an InputMismatchException
D. some output then a NoSuchElementException
E. More than one of A - D is correct

Flawed output

```
Kim (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 Kim 12.5 8.1 7.6 3.2
456 Eric 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)
Eric (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();
            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. "Eric"
        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)");
    }
}
```

Danger with Scanner

- 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` 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?

Danger with Scanner

```
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`

File output

Output to files

- **`PrintStream`**: An object in the `java.io` package that lets you print output to a destination such as a file.
  - Any methods you have used on `System.out` (such as `print`, `println`) will work on a `PrintStream`.
- Syntax:

```
PrintStream <name>
    = new PrintStream(new File("<filename>"));
```

Example:

```
PrintStream output
          = new PrintStream(new File("out.txt"));
output.println("Hello, file!");
output.println("This is a second line of output.");
```

Details about `PrintStream`

```
PrintStream <name>
  = new PrintStream(new File("<filename>"));
```

- If the given file does not exist, it is created.
- If the given file already exists, it is **overwritten**.
- The output you print appears in a file, not on the console. You have to open the file with an editor to see it.
- Do not open the same file for both reading (`Scanner`) and writing (`PrintStream`) at the same time.
  - You will overwrite your input file with an empty file (0 bytes).

`System.out` and `PrintStream`

- The console output object, `System.out`, is a `PrintStream`.

```
PrintStream out1 = System.out;
PrintStream out2 = new PrintStream(new File("data.txt"));
out1.println("Hello, console!");   // goes to console
out2.println("Hello, file!");      // goes to file
```

  - A reference to it can be stored in a `PrintStream` variable.
    - Printing to that variable causes console output to appear.
  - You can pass `System.out` to a method as a `PrintStream`.
    - Allows a method to send output to the console or a file.

`PrintStream` question

- Modify our previous Hours program to use a `PrintStream` to send its output to the file `hours_out.txt`.
- The program will produce no console output.
- the file `hours_out.txt` will be created with the text:

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

PrintStream answer

```
// Processes an employee input file and outputs each employee's hours.
import java.io.*;    // for File
import java.util.*;  // for Scanner

public class Hours2 {
    public static void main(String[] args) throws FileNotFoundException {
        Scanner input = new Scanner(new File("hours.txt"));
        PrintStream out = new PrintStream(new File("hours_out.txt"));
        while (input.hasNextLine()) {
            String line = input.nextLine();
            processEmployee(out, line);
        }
    }

    public static void processEmployee(PrintStream out, String line) {
        Scanner lineScan = new Scanner(line);
        int id = lineScan.nextInt();          // e.g. 456
        String name = lineScan.next();        // e.g. "Eric"
        double sum = 0.0;
        int count = 0;
        while (lineScan.hasNextDouble()) {
            sum = sum + lineScan.nextDouble();
            count++;
        }

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

Prompting for a file name

- We can ask the user to tell us the file to read.
  - The filename might have spaces; use `nextLine()`, not `next()`

```
// prompt for input file name
Scanner console = new Scanner(System.in);
System.out.print("Type a file name to use: ");
String filename = console.nextLine();
Scanner input = new Scanner(new File(filename));
```

- `File`s have an `exists` method to test for file-not-found:

```
File file = new File("hours.txt");
if (!file.exists()) {
    // try a second input file as a backup
    System.out.print("hours file not found!");
    file = new File("hours2.txt");
}
```

More File input practice!

Clicker 1

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

```
12  24
Christmas Eve
```

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

```
12 24\nChristmas Eve
```

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

```
// 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 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:

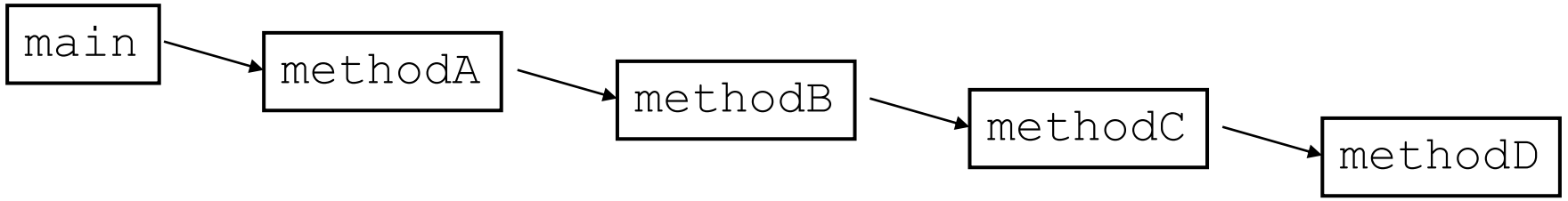
```
1  9.1      490,400  The Shawshank Redemption  (1994)
2  9.1      392,937  The Godfather (1972)
3  9.0      232,741  The Godfather: Part II (1974)
```

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

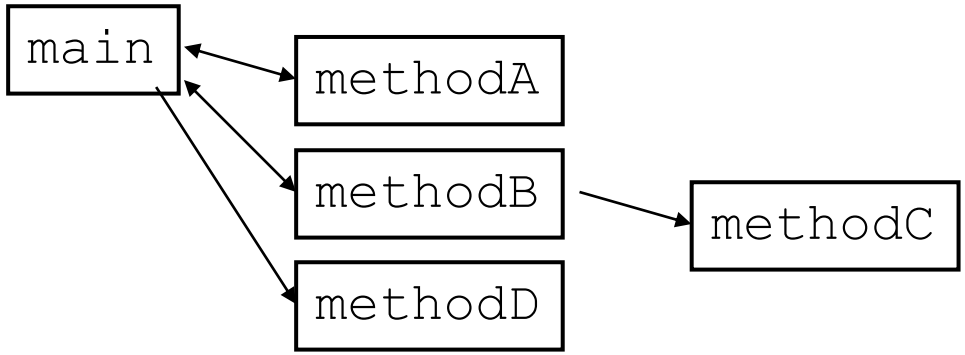
```
Search word? part

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*):



- 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);
        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 searchWor
    while (input.hasNextLine()) {
        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 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()) {
        // the rest of the line
        title += lineScan.next() + " ";
    }
    System.out.println(rank + "\t" + votes
                       + "\t" + rating + "\t" + title);
}
```