Assertion Review Exercise 1

CS 210

```
public static int mystery(int a) {
    int b = 0;
    int c = 0;

    // Point A

    while (a != 0) {
        // Point B

       c = a % 10;

      if (c % 2 == 0) {
          b++;
      } else  {
          b = 0;
          // Point C
       }

       a = a / 10;
       // Point D

    }

    // Point E
    return b;
}
```

Which of the following assertions are true at which point(s) in the code? Choose ALWAYS, NEVER, or SOMETIMES.

	a != 0	c % 2 == 0	b > 0
Point A	SOMETIMES	ALWAYS	NEVER
Point B	ALWAYS	SOMETIMES	SOMETIMES
Point C	ALWAYS	NEVER	NEVER
Point D	SOMETIMES	SOMETIMES	SOMETIMES
Point E	NEVER	SOMETIMES	SOMETIMES

Assertion Review Exercise 2

CS 210

```
public static int mystery(int n) {
    int x = 2;

    // Point A
    while (x < n) {
        // Point B

      if (n % x == 0) {
          n = n / x;
          x = 2;
          // Point C
      } else  {
          x++;
          // Point D
        }

    }
    // Point E
    return n;
}
```

Which of the following assertions are true at which point(s) in the code? Choose ALWAYS, NEVER, or SOMETIMES.

	x > 2	x < n	n % x == 0
Point A	NEVER	SOMETIMES	SOMETIMES
Point B	SOMETIMES	ALWAYS	SOMETIMES
Point C	NEVER	SOMETIMES	SOMETIMES
Point D	ALWAYS	SOMETIMES	SOMETIMES
Point E	SOMETIMES	NEVER	SOMETIMES

Building Java Programs

A Back to Basics Approach

CS 210

CHAPTER 6

FILE PROCESSING

Please download the PPT, and use Slide Show for a better viewing experience

Winnie Li

Topics will be covered...

CS 210

- File Reading Basics
- Token-Based Processing
- Line-Based Processing
- File Output

File reading basics

Input/output (I/O)

CS 210

- What we have seen for the previous chapters:

  from keyboard (input) & console (output)
- But practically, I/O can be found from a lot of other sources:

  i.e., web, network, port, disk, or even the most common one file.

Input/output (I/O)


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

- 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
`renameTo(`*file*`)`	changes name of file

Reading files

CS 210

- It is nice that Java uses the same object to read files as it does to read the keyboard. It's simpler and easier to learn. Some languages (C, Python, etc.) don't do this.
- 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 (shorter):

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

Compiler error w/ files

CS 210

```
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 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 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.
  - `FileNotFoundException` is a checked exception

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(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 this happens."*

File paths

CS 210

- **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/mydata.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`

Token-based processing

Input tokens

CS 210

- **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`

Even though we think of 23 as being an `int`, but it can be any of the three types: 23, 23.0, or "23".

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

CS 210

- **consuming input**: Reading input and advancing the cursor.
  - Calling `next()` etc. moves the cursor past the current token.

```
16.2   23.5\n\t19.1 7.4  22.8\n\n18.5  -1.8 14.9\n
^
```

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

```
16.2   23.5\n\t19.1 7.4  22.8\n\n18.5  -1.8 14.9\n
    ^
```

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

```
16.2   23.5\n\t19.1 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
```

8 temperatures in the file, but 7 lines of output. It's a fencepost problem in disguise.

File input answer


```
// Displays changes in temperature from data in an input
   file.

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

CS 210

- `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

CS 210

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

CS 210

- 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

CS 210

```
// Displays changes in temperature from data in an input
   file.
import java.io.*;     // for File
import java.util.*;   // for 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  <-- Marty here is my data!  --Kim
   14.9 :-)
```

  - You may assume that the file begins with a real number.

File input answer 3


```
// Displays changes in temperature from data in an input
   file.

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

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

CS 210

- 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`
    `    ^`

Line-based processing

CS 210

Hours question

- Given a file `hours.txt` 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 Jennifer 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)
Jennifer (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.*;                     // 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 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("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.
    - On Windows, you will see `\r\n` – both are 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

CS 210

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 Susan 12.5 8.1 7.6 3.2
456 Brad 4.0 11.6 6.5 2.7 12
789 Jennifer 8.0 8.0 8.0 8.0 7.5
```

  - Recall, it should produce the following output:

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

Hours answer, corrected 1

CS 210

```
// 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 answer, corrected 2 (better!)

CS 210

```
// 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. "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)");
    }
}
```

common bugs here:
- not understanding the difference between a String and a Scanner.
i.e., try to call `line.nextInt()` or similar.
- calling a method on the wrong Scanner.
e.g. forget to change input to lineScan.

File output

CS 210

Output to files

CS 210

- **`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("file name"));
```

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("file name"));
```

- 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 will 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).

common `PrintStream` bug:
- declaring it in a method that gets called many times. This causes the file to be re-opened and wipes the past contents. So only the last line shows up in the file.

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.
  - But the file `hours_out.txt` will be created with the text:

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

PrintStream answer 1


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

PrintStream answer 2 (better!)


```
// 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. "Brad"
        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));
```

- `Files` 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");
}
```

The End

CS 210

CHAPTER 6

FILE PROCESSING

Winnie Li

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");
    }
}
```