CS 106A, Lecture 10
File Processing

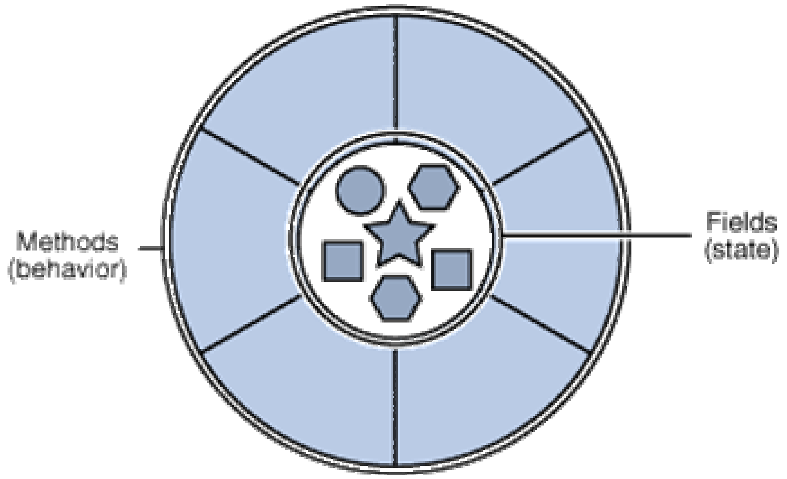
reading:

Art & Science of Java, 12.4

Objects (briefly)

- **object:** An entity that contains data and behavior.
  - *state*: data variables inside the object
  - *behavior*: methods inside the object

    - A **class** is a type/category of objects.
    - Classes of objects we have seen: `String`, `RandomGenerator`, `Karel`, ...



- Constructing (creating) an object:

  `Type name` `= new` ***`Type`***`(`***`parameters`***`);`

- Calling an method of an object:

  `object``.`***`method`***`(`***`parameters`***`);`

File objects

```
import java.io.*;   // for 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**
`f.canRead()`	returns `true` if file is able to be read
`f.delete()`	removes file from disk
`f.exists()`	returns `true` if this file exists on disk
`f.getName()`	returns file's name
`f.length()`	returns number of bytes in file
`f.renameTo(file)`	changes name of a file

File paths

- **absolute path**: specifies a drive or a top `"/"` folder

  ```
  "C:/Documents/smith/hw6/input/data.csv"
  "/home/jsmith12/Desktop/report.doc"
  ```

  – Windows can also use backslashes to separate folders; but don't.

- **relative path** *(preferred)*: does not specify any top-level folder

  ```
  "names.dat"
  "input/kinglear.txt"
  ```

  – Assumed to be relative to the *current directory*:

  ```
  File f = new File("res/readme.txt");
  ```

  If our project is in `H:/docs/hw6` ,
  Java will look for `H:/docs/hw6/`**`res/readme.txt`**

Scanner

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

- To read data from a file, construct a `Scanner` object and pass a `File` as the parameter.
  - Example:

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

  - or (shorter):

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

Scanner methods

Method	**Description**
`sc.nextLine()`	reads and returns a one-*line* `String` from the file
`sc.next()`	reads and returns a one-word `String` from the file
`sc.nextInt()`	reads and returns an `int` from the file
`sc.nextDouble()`	reads and returns a `double` from the file
`sc.hasNextLine()`	returns `true` if there are any more lines
`sc.hasNext()`	returns `true` if there are any more tokens
`sc.hasNextInt()`	returns `true` if there is a next token and it's an `int`
`sc.hasNextDouble()`	returns `true` if there is a next token and it's a `double`
`sc.close();`	should be called when done reading the file

Reading lines or tokens

```
the quick brown
 fox    jumps

over
the lazy dog
```

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

- Reading a file line-by-line:

```
while (input.hasNextLine()) {
    String line = input.nextLine();  // "the quick brown",
    println(line);                   // " fox    jumps",
}
```

- Reading a file word-by-word:
  - **token**: A unit of user input, separated by whitespace.

```
while (input.hasNext()) {
    String word = input.next();
    println(word);                   // "the", "quick",
}                                    // "brown", "fox", ...

// or hasNextInt / nextInt if tokens are numeric, etc.
```

Input cursor

- Consider a file `weather.txt` that contains this text:

```
16.2   23.2
   19.2 7.7  22.9

18.4  -1.6 14.6
```

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

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

```
16.2   23.2\n   19.2 7.7  22.9\n\n18.4  -1.6 14.6\n
^
```

- **input cursor**: The current position of the `Scanner`.
  - As you read data from the file, the cursor advances.
  - It is not possible to rewind the cursor. You must re-open the file.

Reading tokens

- Calling `nextDouble` etc. skips whitespace and reads one token.

```
16.2   23.2\n   19.2 7.7  22.9\n\n18.4  -1.6 14.6 \n
^
```

```
double d1 = input.nextDouble();   // 16.2
16.2   23.2\n   19.2 7.7  22.9\n\n18.4  -1.6 14.6 \n
    ^
```

```
double d2 = input.nextDouble();   // 23.2
16.2   23.2\n   19.2 7.7  22.9\n\n18.4  -1.6 14.6 \n
           ^
```

```
String s1 = input.next();         // "19.2"
16.2   23.2\n   19.2 7.7  22.9\n\n18.4  -1.6 14.6 \n
                    ^
```

```
String s2 = input.next();         // "7.7"
16.2   23.2\n   19.2 7.7  22.9\n\n18.4  -1.6 14.6 \n
                        ^
```

Reading lines

- When you read a line, the cursor advances past the next \n marker.

```
16.2   23.2\n   19.2 7.7  22.9\n\n18.4  -1.6 14.6 \n
^

String line = input.nextLine();  // "16.2   23.2"
16.2   23.2\n   19.2 7.7  22.9\n\n18.4  -1.6 14.6 \n
             ^

String line = input.nextLine();  // "   19.2 7.7  22.9"
16.2   23.2\n   19.2 7.7  22.9\n\n18.4  -1.6 14.6 \n
                                ^

String line = input.nextLine();  // ""   (empty)
16.2   23.2\n   19.2 7.7  22.9\n\n18.4  -1.6 14.6 \n
                                  ^

String line = input.nextLine();  // "18.4  -1.6 14.6 "
16.2   23.2\n   19.2 7.7  22.9\n\n18.4  -1.6 14.6 \n
                                                    ^
```

Compiler error

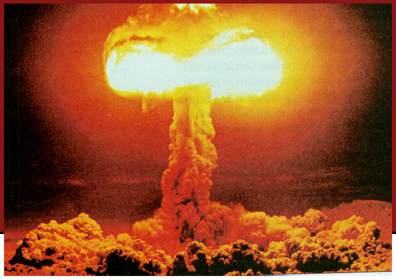
```
import acm.program.*;  // for ConsoleProgram
import java.io.*;      // for File
import java.util.*;    // for Scanner

public class ReadFile extends ConsoleProgram {
    public void run() {
        Scanner input = new Scanner(new File("data.txt"));
        while (input.hasNextLine()) {
            String line = input.nextLine();
            println(line);
        }
    }
}
```

- 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.

Try/catch

```
try {
    statements;    // code that might throw an exception
} catch (ExceptionType name) {
    statements;    // code to handle the error
}
```

- To execute code that might throw an exception, you must enclose it in a `try/catch` statement.

```
try {
    Scanner input = new Scanner(new File("data.txt"));
    ...
} catch (FileNotFoundException ex) {
    println("Error reading the file: " + ex);
}
```

Tokens exercise

Weather

- Suppose we have an input file `weather.txt` of temperatures:

```
16.2    23.2
   19.2 7.7   22.9


18.4  -1.6 14.6
```

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

```
16.2 to 23.2, change = 7.0
23.2 to 19.2, change = -4.0
19.2 to 7.7, change = -11.5
7.7 to 22.9, change = 15.2
22.9 to 18.4, change = -4.5
18.4 to -1.6, change = -20.0
-1.6 to 14.6, change = 16.2
```

Tokens solution

```
/* Displays changes in temperature from data in an input file. */
import acm.program.*;   // for ConsoleProgram
import java.io.*;       // for File
import java.util.*;     // for Scanner

public class Weather extends ConsoleProgram {
    public void run() {
        try {
            Scanner input = new Scanner(new File("weather.txt"));
            double prev = input.nextDouble();   // fencepost
            while (input.hasNextDouble()) {
                double next = input.nextDouble();
                println(prev + " to " + next
                        + ", change = " + (next - prev));
                prev = next;
            }
            input.close();
        } catch (FileNotFoundException ex) {
            println("Error reading file: " + ex);
        }
    }
}
```

Tokens exercise 2

Weather2

- Modify your **`Weather`** program to produce the same output even if it is given an input file that contains some non-numeric "junk" tokens that should be ignored:

```
16.2   23.2
   19.2 abc 7.7  hi there!  22.9

       TODO: buy pants
18.4  -1.6 14.6  :-)
```

  – You may assume that the file is non-empty and begins with a numeric token.

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):

```
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:838)
        at java.util.Scanner.next(Scanner.java:1347)
        at MyProgram.readFile(MyProgram.java:39)
        at MyProgram.run(MyProgram.java:15)
```

Tokens solution 2

```
public class Weather2 extends ConsoleProgram {
    public void run() {
        try {
            Scanner input = new Scanner(new File("weather2.txt"));
            double prev = input.nextDouble();    // fencepost
            while (input.hasNext()) {
                if (input.hasNextDouble()) {
                    double next = input.nextDouble();
                    println(prev + " to " + next
                            + ", change = " + (next - prev));
                    prev = next;
                } else {
                    input.next();    // throw away junk token
                }
            }
            input.close();
        } catch (FileNotFoundException fnfe) {
            println("Error reading file: " + fnfe);
        }
    }
}
```

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();
    println(num);                         // 15
    double num2 = scan.nextDouble();
    println(num2);                        // 3.2
    String word = scan.next();
    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()) {
    Scanner tokens = new Scanner(input.nextLine());
    // process the contents of this line
    int count = 0;
    while (tokens.hasNext()) {
        String word = tokens.next();
        count++;
    }
    println("Line has " + count + " words");
}
...
```

Prompting for file name

```
// prompt for a file name in the res/ folder
String filename = readLine("Input file name? ");
File inputFile = new File("res", filename);
```

- To ensure that the file exists, you may want to re-prompt:

```
// re-prompt for a file name in the res/ folder
while (!inputFile.exists()) {
    filename = readLine("Not found. Try again: ");
    inputFile = new File("res", filename);
}
```

- Or the method **`promptUserForFile`** handles all of this:

```
// re-prompt for a file name in the res/ folder
String filename = promptUserForFile("Input? ", "res");
File inputFile = new File(filename);
```

Election question

- Write a program **`Election`** that reads a file of poll data.

  Format: *State Candidate1% Candidate2% 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.
  - If they tie in a given region, don't give anybody those votes.

```
Input file? polls.txt
Candidate 1: 325 votes
Candidate 2: 183 votes
```

Election solution

```
String filename = promptUserForFile("Input file? ",
  "res");
Scanner input = new Scanner(new File(filename));
int totalVotes1 = 0;
int totalVotes2 = 0;
while (input.hasNextLine()) {          // "CT 56 31
  7 Oct U. of Conn"
    Scanner tokens = new Scanner(input.nextLine());
    tokens.next();                     // skip state
  abbreviation
    int votes1 = tokens.nextInt();
    int votes2 = tokens.nextInt();
    int eVotes = tokens.nextInt();
    if (votes1 > votes2) {
        totalVotes1 += eVotes;
    } else if (votes2 > votes1) {
        totalVotes2 += eVotes;
```

Hours question

- Given a file `hours.txt` of payroll information about 106A SLs:

```
123 Amy 12.5 8.5 7.25 3.25
456 Miles 4.0 11.6 6.5 12.2 2.7
802 Jessie 1.5
647 Vilde 8.0 3.5 6.5
```

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

```
Amy (ID#123) worked 31.5 hours (7.875/day)
Miles (ID#456) worked 37.0 hours (7.4/day)
Jessie (ID#802) worked 1.5 hours (1.5/day)
Vilde (ID#647) worked 18.0 hours (6.0/day)
```

Hours solution (flawed)

```
public void run() {   // this code does not quite work!
    try {
        Scanner input = new Scanner(new File("res/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++;
            }
            println(name + " (ID#" + id +
                    ") worked " + totalHours + " hours (" +
                    (totalHours / days) + " hours/day)");
        }
        input.close();
    } catch (FileNotFoundException fnfe) {
        println("Error reading file: " + fnfe);
    }
}
```

Flawed output

```
Amy (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.run(HoursWorked.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.

Hours solution

```
public void run() {
    try {
        Scanner input = new Scanner(new File("res/hours.txt"));
        while (input.hasNextLine()) {   // process one person
            Scanner tokens = new Scanner(input.nextLine());
            int id = tokens.nextInt();
            String name = tokens.next();
            double totalHours = 0.0;
            int days = 0;
            while (tokens.hasNextDouble()) {
                totalHours += tokens.nextDouble();
                days++;
            }
            double avg = totalHours / days;
            println(name + " (ID#" + id + ") worked " + totalHours
                    + " hours (" + avg + "/day)");
        }
    } catch (FileNotFoundException fnfe) {
        println("Error reading file: " + fnfe);
    }
}
```