Elegance and Classes

Designing and building a software system from scratch

- Develop a precise specification of the system
- Determine the classes, their responsibilities and collaborators
- Determine the precise protocol or interface of the classes
- Implement the classes

Example

When the Java program starts, it analyzes the specified text file. It constructs a summary report regarding each word that occurs in the file and the number of times that word occurs, sorted from most frequently occurring word to least frequently occurring.

Issues with the specification

1. What is the user interface for this program? In particular, how does the program know what text file to analyze?
   - Does it display a dialog box?
   - Does the user provide it at the command line
   - Does it expect the user to provide it as a command line argument?
2. What should happen if the user specifies the name of the file that does not exist?
3. What should be the format of the summary report? Can it be specified by the user?

Issues with the specification…

4. When counting occurrences of words, does capitalization matter? Can the user specify how to treat capitalization?
5. Can the same execution of the program be used to analyze several text files or do you need to rerun the program?
6. What is a “word”? What distinguishes a word from an arbitrary sequences of characters? Are we counting only words that appear in a dictionary? If so, which dictionary?

Issues with the specification…

7. When counting occurrences of words, are we distinguishing between singular and plural? For example, shell vs. shells? How about mouse and mice?
8. When reporting the results, what order should be used for two words that have the same frequency of occurrences?
9. Is the efficiency of the program regarding space or time an important concern?
10. Is the program going to be enhanced later to deal with input other than text files and with other forms of summary reports?

Class discussion

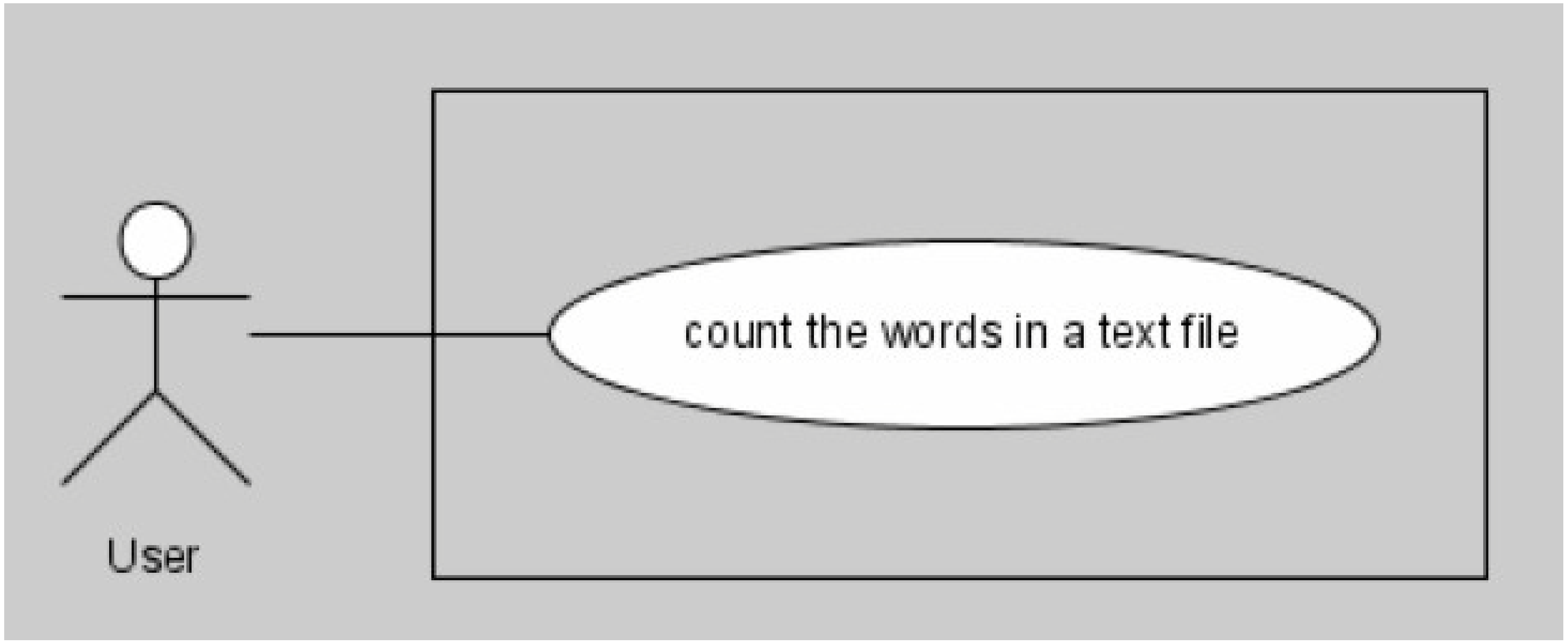
1. As a class, you agreed upon using the dialog box option.
2. Not applicable, since the user is presented with a dialog box to choose one of the available files.
3. The output is a simple table format which lists the words and their corresponding frequencies.
4. Same words in upper case, mixed case and lower case are treated as the same.
5. At the end of the program, the user is presented with an option to run the program again with a different file.

Class discussion…

6. Words are separated by space, comma, period, :, ;, newline and tab.
7. Singular and plural words such as (mice and mouse) are two different words.
8. Two words with the same frequency of occurrence should be listed in the lexicographical order.
9. Program should run as efficiently as possible. The efficiency of the program is important enough that only one read-through of the text file is allowed.
10. Not considered at this point.

Use cases

- A *use case* is a sequence of steps indicating how the program is to behave in a certain situation to achieve a particular goal.
- Create a use case for each of the ways the system is expected to behave.



. A use case diagram for a word frequency counter application.

Use Case - Example

- The user starts the program by typing the following on the command line of a console window:
  - *Java programName completePathNameofTheFile*
- The program checks for the specified file. If it is not found, it prints an error message and exits.
- The program traverses the file, treating each byte as a character, and treating each sequence of characters not containing any delimiters as a word., The delimiter characters are Java's default whitespace characters.

Use case Example

- The program keeps track of the number of occurrences of each word. Two words are considered equal if and only if they contain exactly the same sequence of characters.
- The program prints to the console window a list of all the words and their frequencies with words with highest frequency first and with one word and its frequency (separated by a tab) per line.
- The program exits after it prints the report.

Use Case example…

- Note that this use case has two scenarios. The successful scenario where the report is printed on the screen and the unsuccessful scenario where the file is not found.
- Also, note that this use case answers concerns 1-7 listed in slides: 4,5,6

Use Case Example…

- The following addresses concerns 8, 9, 10.
  - If two words have the same frequency, they should be reported in alphabetical order using ASCII encoding.
  - The efficiency of the program is important enough that only one read-through of the text file is allowed. The program is not allowed to copy the entire file to RAM – there may not be enough memory.

Use case Example

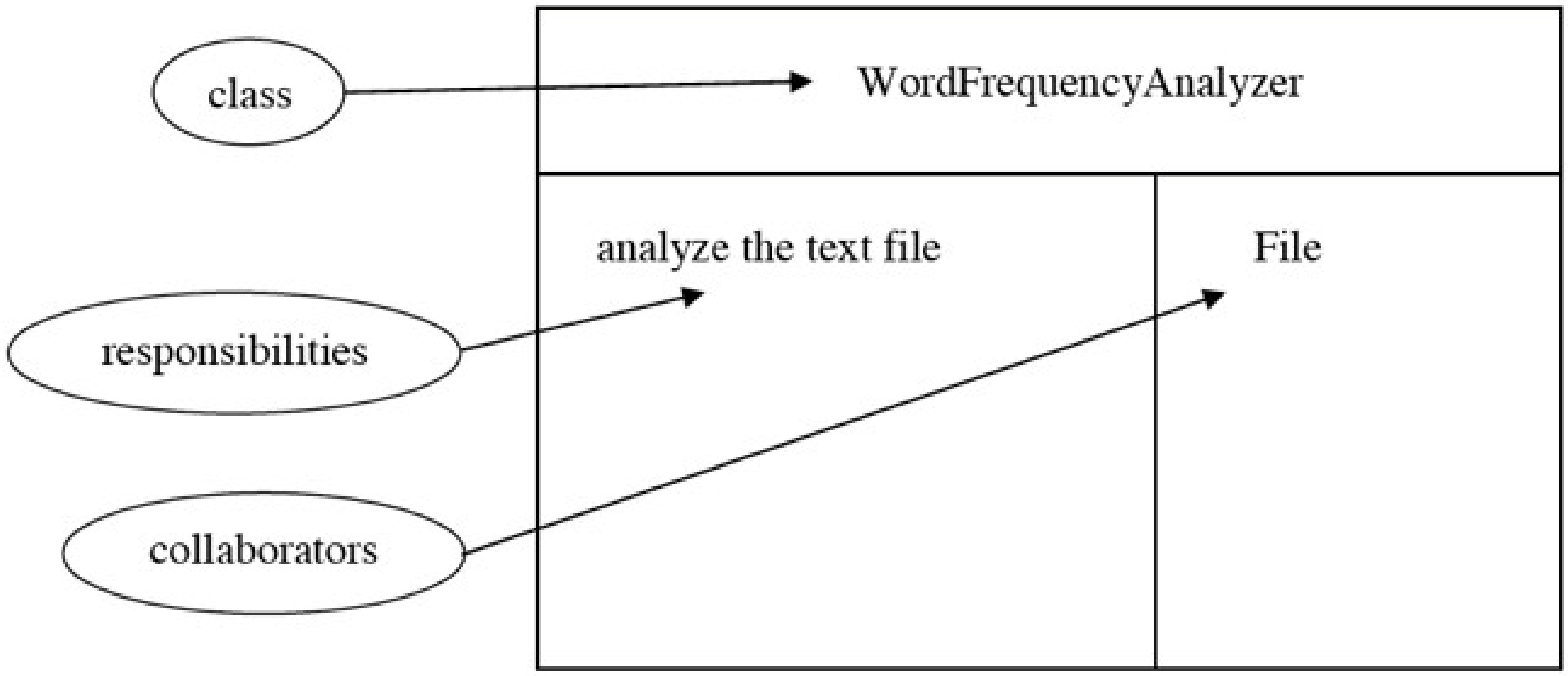
- The program is going to be enhanced later to deal with other inputs and outputs of an as-yet unknown format.

Finding the classes (first attempt)

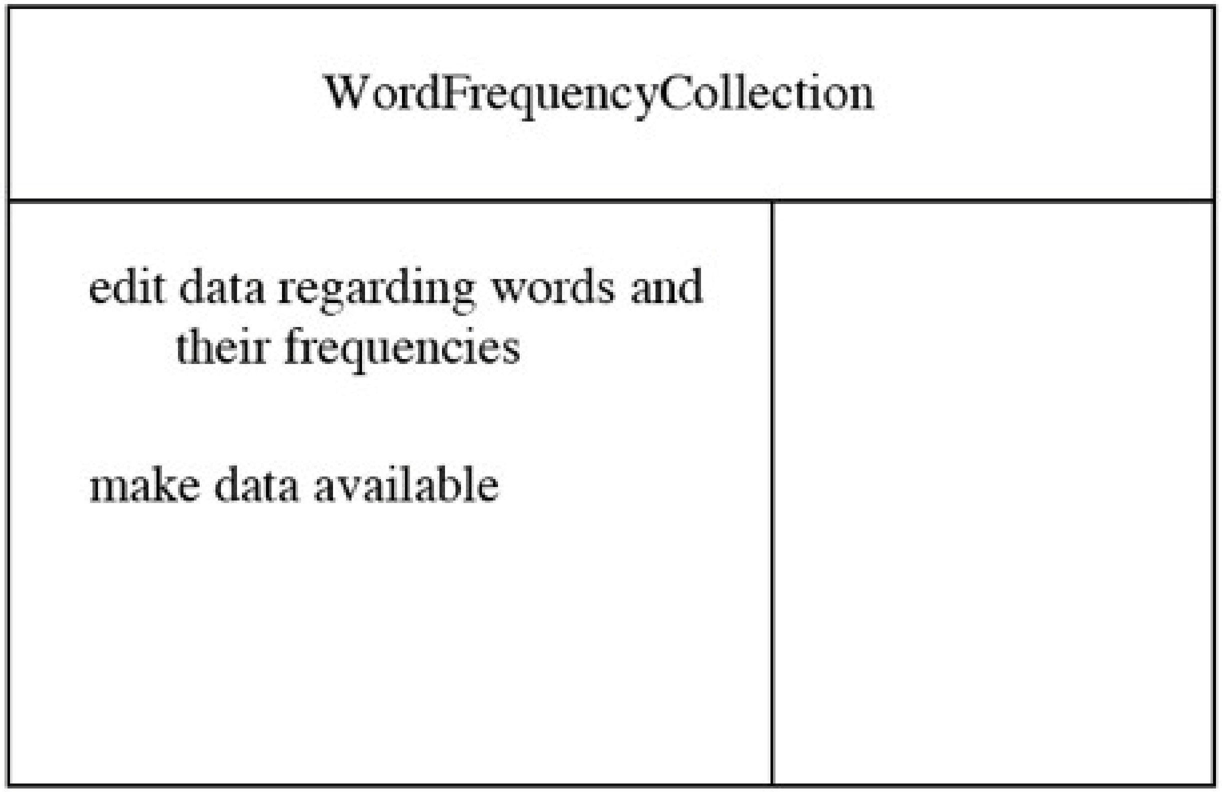
- Let each noun in the specification correspond to a class and each verb to a method of a class
- Add concepts from the application domain

CRC cards

- CRC stands for “Class, Responsibilities, and Collaborators”
- Use one note card for each class
- Use role playing to determine responsibilities and collaborators



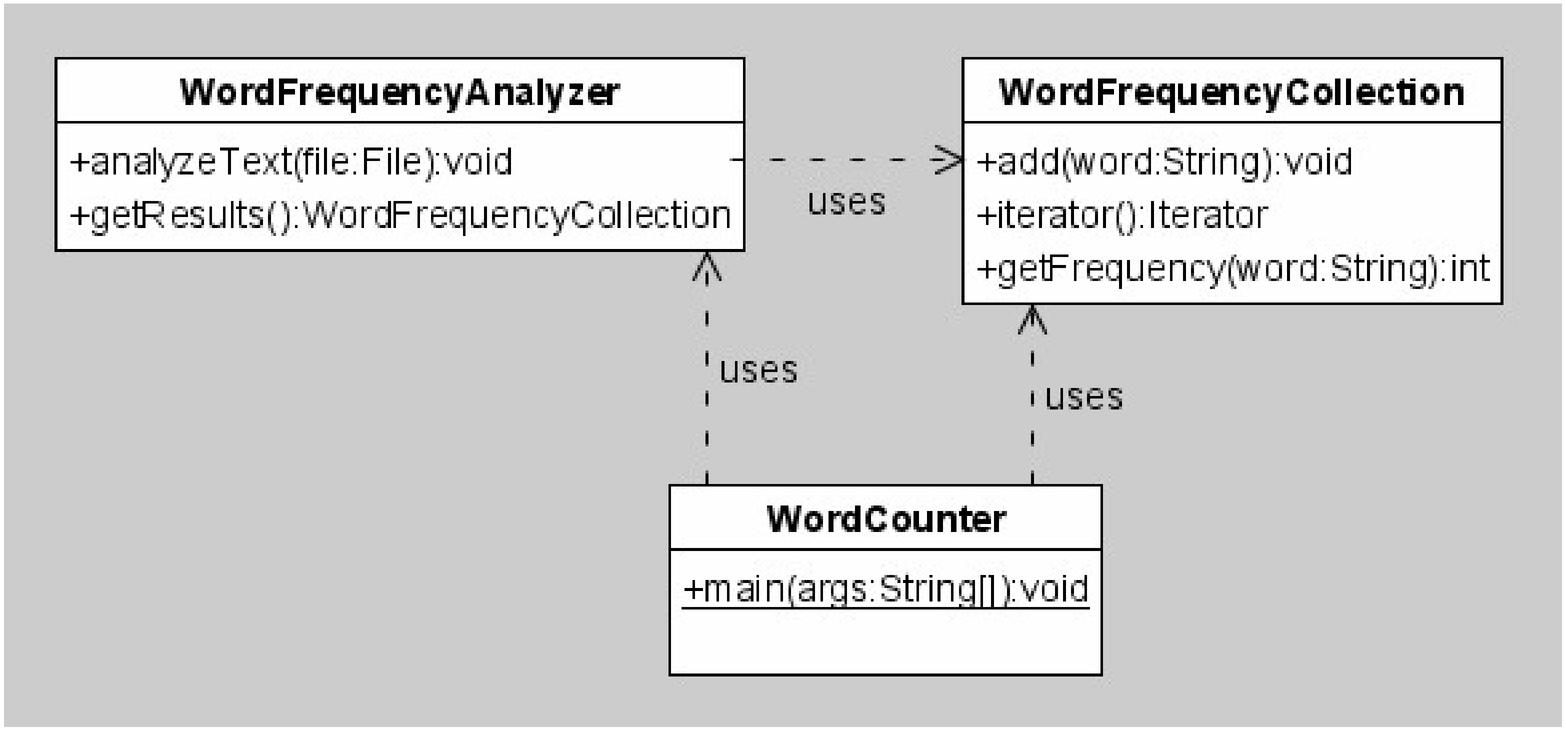
A CRC card for a WordFrequencyAnalyzer class.



. A WordFrequencyCollection CRC card.

WordCounter	
report an error if there is no file with a given name	File
create a File and a WordFrequencyAnalyzer	WordFrequency-Analyzer
initiate the analysis	WordFrequency-Collection
get and print the result	

A WordCounter CRC card.


WordFrequencyAnalyzer
+analyzeText(file:File):void
+getResults():WordFrequencyCollection
WordFrequencyCollection
+add(word:String):void
+iterator():Iterator
+getFrequency(word:String):int
WordCounter
+main(args:String[]):void
uses
uses
uses

The word frequency analyzer classes.

Guideline

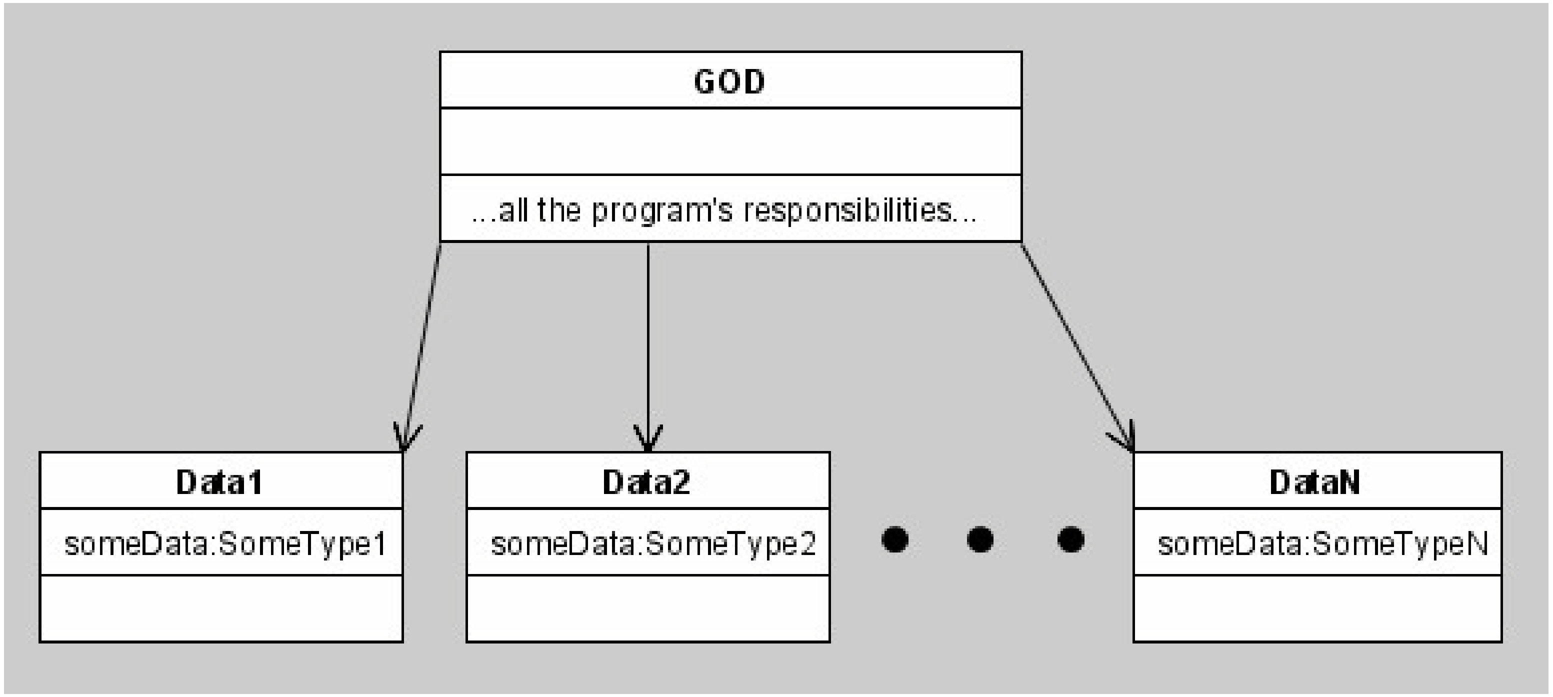
When working in the early high-level design phase, keep the discussion at a high level. In particular, avoid wasting time on implementation details and low-level data representations.

Maximize cohesion of classes

- A class should model one concept
- All its methods should be related to and appropriate for that concept
- Promotes understanding and reusability of the class
- Example: The Java String class

Guideline

Every class should be responsible for doing one thing only and doing it well.



An ill-formed God class with all the responsibilities and slave (data-only) classes.