

# Programming Languages and Techniques (CIS120)

## Lecture 32

### Histogram Demo

#### Chapter 28

# Design Example: Histogram.java

A design exercise using java.io and the  
generic collection libraries

(SEE COURSE NOTES FOR THE FULL STORY)

# Problem Statement

Write a program that, given a filename for a text file as input, calculates the frequencies (i.e. number of occurrences) of each distinct word of the file. The program should then print the frequency distribution to the console as a sequence of “word: freq” pairs (one per line).

Histogram result:

The : 1	each : 1	line : 2	should : 1
Write : 1	file : 2	number : 1	text : 1
a : 4	filename : 1	occurrences : 1	that : 1
as : 2	for : 1	of : 4	the : 4
calculates : 1	freq : 1	one : 1	then : 1
command : 1	frequencies : 1	pairs : 1	to : 1
console : 1	frequency : 1	per : 1	word : 2
distinct : 1	given : 1	print : 1	
distribution : 1	i : 1	program : 2	
e : 1	input : 1	sequence : 1	

# Decompose the problem

- Sub-problems:
  1. How do we iterate through the text file, identifying all of the words?
  2. Once we can produce a stream of words, how do we calculate their frequency?
  3. Once we have calculated the frequencies, how do we print out the result?
- What is the interface between these components?
- Can we test them individually?

# How to produce a stream of words?

1. How do we iterate through the text file, identifying all of the words?

```
public interface Iterator<T> {  
    // returns true if the iteration has more elements  
    public boolean hasNext();  
  
    // returns the next element in the iteration  
    public T next();  
}
```

- **Key idea:** Define a class (WordScanner) that implements this interface by reading words from a text file.

# Coding: Histogram.java

WordScanner.java

Histogram.java

True or False: The following test indicates that `WordScanner` *should* raise a `NullPointerException` when called with `null`.

```
@Test
public void testNull() {
    try {
        new WordScanner(null);
    } catch (NullPointerException e) {
        return;
    }
    fail();
}
```

ANSWER: True

# Iterator – hasNext() – First Attempt?

```
@Override
public boolean hasNext() {
    boolean value = true;
    try {
        int c = r.read();
        if (c == -1) {
            value = false;
        }
    } catch (IOException io) {
        System.out.println("IO Exception happened");
    }
    return value;
}
```



```
public class WordScanner implements Iterator<String> {  
    private Reader r;  
    private int c = -1;  
    // ...  
}
```

Which combination of the following properties form a useful invariant for the WordScanner fields?

1. r is not null
  2. r is null if and only if there is no next word
- 
- A. c is 0 if there is no next word and nonzero otherwise
  - B. c is -1 if there is no next word and contains the first character of the next word otherwise

```
public class WordScanner implements Iterator<String> {  
    private Reader r;  
    private int c = -1;  
    // ...  
}
```

Which combination of the following properties form a useful invariant for the WordScanner fields?

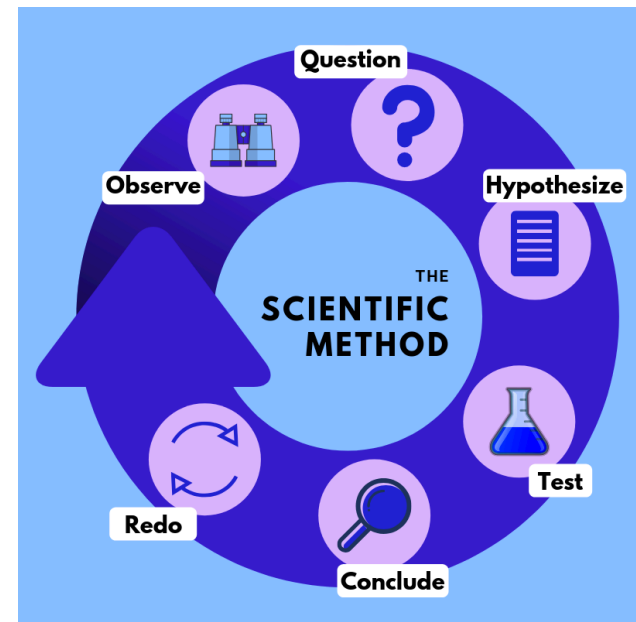
1. r is not null
  2. r is null if and only if there is no next word
- 
- A. c is 0 if there is no next word and nonzero otherwise
  - B. c is -1 if there is no next word and contains the first character of the next word otherwise

ANSWER: 1 & B

# Some Advice on Debugging

# Use the Scientific Method

1. Make an observation / ask a question
  - One of my test cases fails!
  - Which assertion? What exception? What is the stack trace?
2. Formulate a hypothesis
  - Could I have passed null as bar to foo.munge(bar)?
3. Conduct an experiment
  - Modify the program to try to confirm or refute the hypothesis.
  - *Don't* make random changes!
  - Predict the outcome of your experiment
  - Re-run test cases, or execute the program
4. Analyze the results
  - Did the modified code behave as expected?
5. Draw conclusions / Report results
  - Create a new test case (if appropriate)



# Observing Behavior

- Understand exceptions and their stack traces
  - They give you a lot of information
- If you are using Eclipse, it is worth taking a little time to learn how to use the debugger!
  - See Piazza for a Quick Start tutorial
- Simple print statements are also very effective!
  - Confirm or disprove hypothesis
  - e.g.: The code reached "HERE!" (or not)