Programming Languages and Techniques (CIS120)

Lecture 32

Histogram Demo

Chapter 28

Announcements

- HW7: Chat Server
  - Due *tomorrow*, Tuesday November 19th at 11:59pm
- HW9a: (see next slide)
  - Due this Friday!
- HW8: TwitterBot
  - Available very soon
  - Due: Tuesday, November 26th at 11:59pm
  - This is a *new* project (replacing SpellChecker), so please start early!
- Regrade requests for Midterm 2 due by Friday.

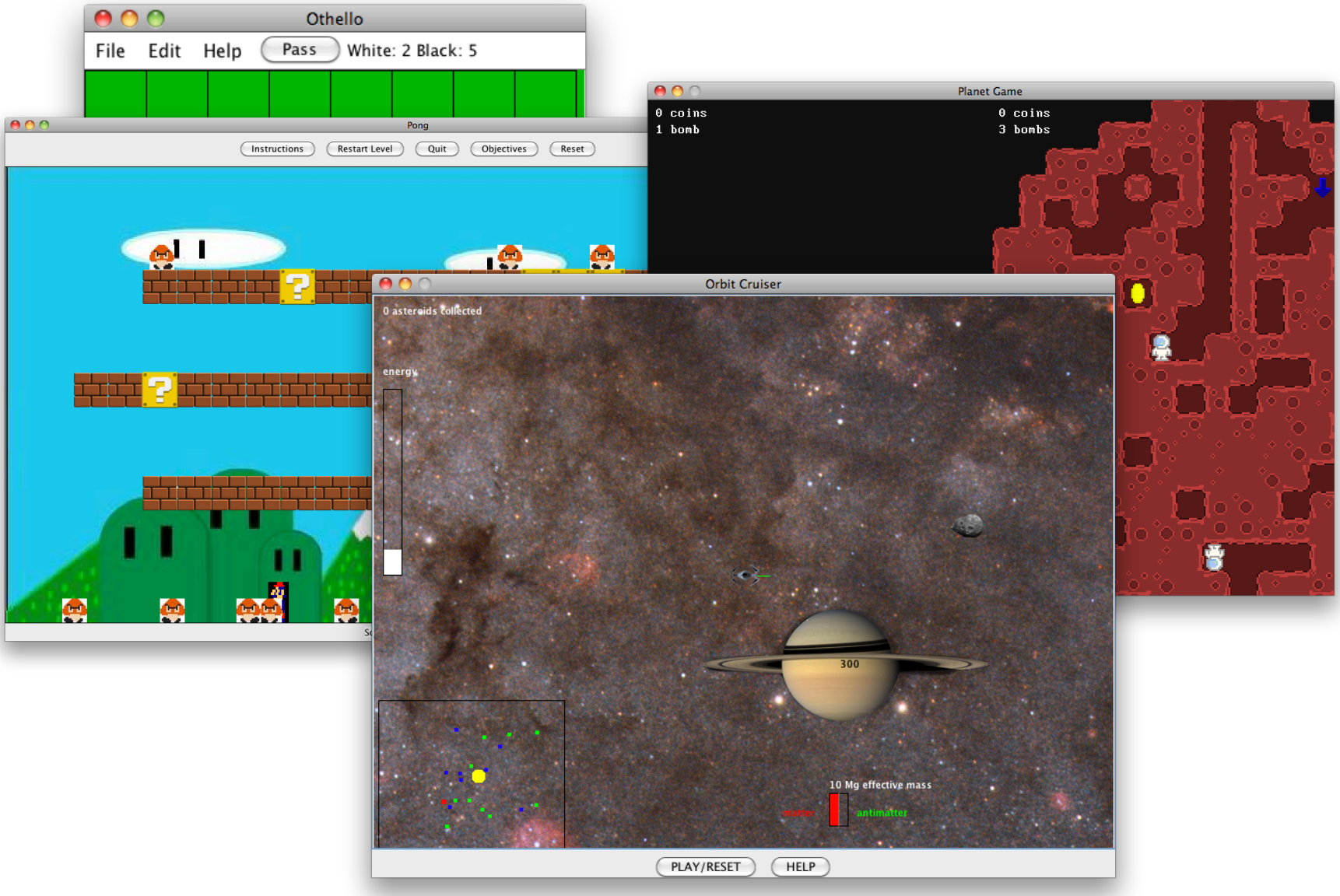
HW9: Game project

- Game Design Proposal Milestone Due: (8 points)
  Friday November 22nd at NOON = 11:59AM!!!!
  - (Should take about 1 hour)
  - Submit on GRADESCOPE
  - TAs will give you feedback over the weekend

- Final Program Due: (92 points)
  Monday, December 9th at 11:59pm
  - Submit zipfile online, submission *only* checks if your code compiles
  - Eclipse is STRONGLY recommended for this project
  - May distribute your game (after the deadline) if you do not use any of our code
- Grade based on demo with your TA during reading days
  - Grading rubric on the assignment website
  - Recommendation: don't be too ambitious.

- *NO LATE SUBMISSIONS PERMITTED*

HW9: Game Project



Design Example: Histogram.java

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

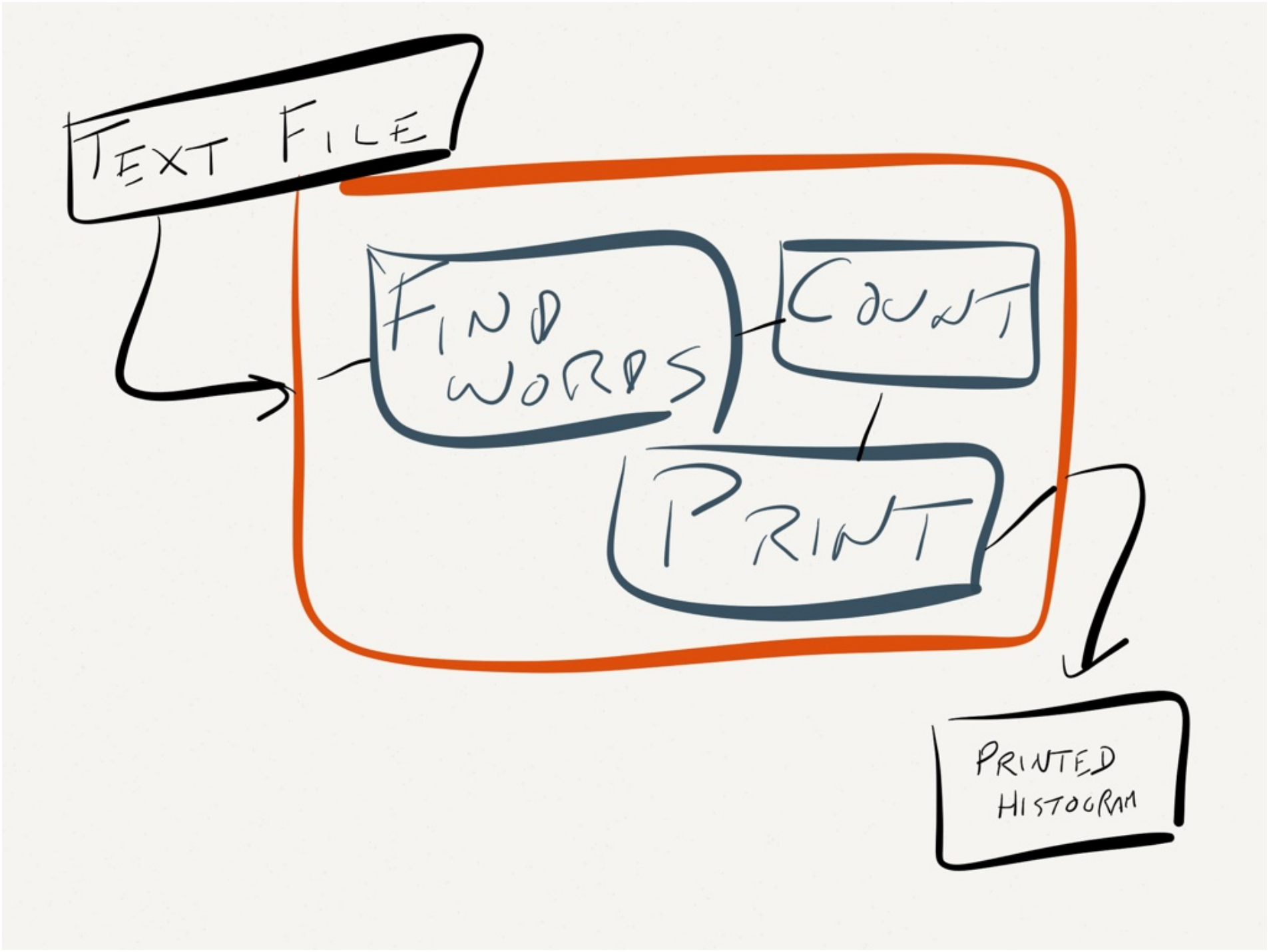
(SEE COURSE NOTES FOR THE FULL STORY)

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


TEXT FILE
FIND WORDS
COUNT
PRINT
PRINTED HISTOGRAM

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();
    // Optional: removes last element returned
    public void remove();
}
```

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

Coding: Histogram.java

WordScanner.java

Histogram.java

The The following test indicates that WordScanner should raise a NullPointerException

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

True

False

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;
}
```

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

```
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

1 & A

1 & B

2 & A

2 & B

```
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

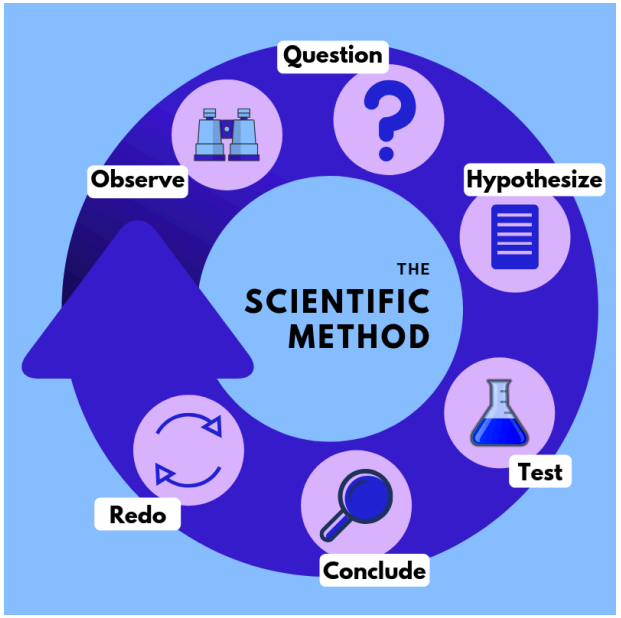
A. c is 0 if there is no next word and nonzero otherwise
B. c is -1 if there is no next word and contains the first character of the next word otherwise

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)