EECS 114: Assignment 2

October 17, 2015

Due Sunday 25 Oct 2015 at 11:59 PM

1 Word Ladder

A word ladder is a word game. The player is given a start word and an end word and must come up with a "ladder" of words where each word in the ladder is no more than one letter different than the word before it and the word after it. Here are a few word ladder examples:

- style stale state slate plate prate crate craze crazy
- brave braze blaze blame flame flams flats feats fears hears heart
- stack stark stars stirs sties stied steed stead steak

For this assignment, you will write a program to find the shortest word ladder given a start word and an end word. To keep things simple, you will only be working with 5 letter words. The program will take as command line arguments the name of the dictionary file, the start word, and the end word and output to standard output the word ladder if found. The program output should match the examples above. If no word ladder can be found simply output the start and end word. You can download the dictionary of 5 letter words (`words5.dict`) we will use to grade your program from the Piazza course website. The following command must invoke your program.

% **`java WordLadder words5.dict start end`**

There are other algorithms to solve this problem including some that are much more efficient than the one we will be using. However, this assignment has been designed to give you practice combining implementations of a stack (`MyStack`), queue (`MyQueue`), and simple list (`SimpleList`), to solve a problem. In a `WordLadder` class, implement the following algorithm.

1.1 Algorithm Find Word Ladder

```
create a stack of strings
push the start word on this stack
create a queue of stacks
enqueue this stack

while the queue is not empty
   for each word in the dictionary
      if a word is 1-letter different in any pos than top string of the front stack
         if this word is the end word
            // Done! The front stack plus this word is your word ladder.
            output this word ladder
         make a copy of the front stack
         push the found word onto the copy
         enqueue the copy.
   dequeue front stack
```

2 SimpleList Class

The required `SimpleList` class methods are as follows.

- `SimpleList(SimpleList rhs)` - Constructor. Instantiates `this` list as a deep copy of `rhs`.
- `Node previous(Node curr)` - Returns the previous node for the specified node `curr`.
- `Node next(Node curr)` - Returns the next node for the specified node `curr`.
- `char getAt(int index)` - Returns the element at the specified position in this list. Index of first element in list is `0`.
- `void setAt(int index, T value)` - Replaces the element at the specified position in this list with the specified element.
- `void insertAtPos(int index, T value)` - Inserts a node with specified `value` at specified position in the list, shifting elements starting at i to the right, if needed.
- `boolean removeAt(int index)` - Removes the element at the specified position index in `this` list. Returns `true` if this list contained the specified element.
- `int size()` - Returns the number of items in the list.
- `String toString()` - Overrides the `toString()` method. Returns the formatted contents of `this` as a String. Allows for printing a `MyList` instance `llist` to standard output in following way, `System.out.println("llist contents = " + llist);`

3 MyStack Class

The required `MyStack` class methods and data members are the following.

- `final private int CAPACITY` - size of the stack's internal array
- `private int t` - index to the position of the top item in the stack
- `private T[] data` - array that holds contents of the stack
- `MyStack()` - Constructor. Initializes and allocates an empty stack.
- `public boolean isEmpty()` - Returns true if stack is empty.
- `public int size()` - Returns the size of the stack.
- `public void push(T value)` - Inserts `value` onto the top of the stack. Throws a `StackOverflowException` if the stack is full. This exception should be handled within this method.
- `public void pop()` - Removes the stack's top item. Throws a `StackUnderException` if the stack is empty. This exception should be handled within this method.
- `public T top()` - Returns the stack's top item. Throws a `StackUnderException` if the stack is empty. This exception should be handled within this method.

4 MyQueue

The `MyQueue` class should implement a Queue ADT that "has a" `SimpleList` as its underlying data structure. The following methods are required. You may add other methods if it does not violate the properties or policies for a queue. Make use of whatever `SimpleList` member methods are available to you to implement the queue's functionality.

- `public boolean isEmpty()` - Returns true if queue is empty.
- `public void push(T value)` - Inserts `value` onto the rear of the queue.
- `public void pop()` - Removes the queue's front item. Throws a `QueueUnderflowException` if the queue is empty. This exception should be handled within this method.
- `public T front()` - Returns the front item in the queue. Throws a `QueueUnderflowException` if the queue is empty. This exception should be handled within this method.

5 Exceptions

The array-based `MyStack` class defines an insert (`push`) into a full stack or an access of (`top`) or removal from (`pop`) an empty stack to be an illegal operation. Prohibited operations for the linked list-based `MyQueue` class are an access of (`front`) or removal from (`pop`) an empty queue. To prevent this an exception should be thrown and caught within the appropriate member method for each class. You will implement your own user-defined exception classes. Derive each from `java.lang RuntimeException`. Place each exception class in a java file with the same name as the class e.g., `StackUnderflowException.java`.

An example of the required console output for an exception `throw` and `catch` is listed below. Your program's output must match this. The first line of output is produced using `System.out`. The next 3 lines are obtained by a call to `printStackTrace()` on the thrown exception object. The file line numbers, e.g., `MyStack.java:29`, are implementation dependent. For your program's classes, they will be different. Read more about `printStackTrace()` here.[1]

- `class StackUnderflowException` - Must call MyStack's `size()` method to print the current size of the stack (shown as 0 below).

```
pop() on MyStack of size == 0
StackUnderflowException: pop() on MyStack of size == 0
at MyStack.pop(MyStack.java:29)
at Main.main(Main.java:39)
```

- `class StackOverflowException` - Must use the MyStack's internal data member `CAPACITY` to print the capacity of the stack (shown as 8 below). Must call MyStack's `size()` method to print the current size of the stack (shown as 8 below).

```
push() on MyStack with CAPACITY == 8, size == 8
StackOverflowException: push() on MyStack with CAPACITY == 8, size == 8
at MyStack.push(MyStack.java:18)
at Main.main(Main.java:24)
```

- `class QueueUnderflowException` - Must call MyQueue's `size()` method to print the current size of the queue (shown as 0 below).

```
pop() on MyQueue of size == 0
QueueUnderflowException: pop() on MyQueue of size == 0
at MyQueue.pop(MyQueue.java:45)
at Main.main(Main.java:22)
```

[1] http://docs.oracle.com/javase/7/docs/api/java/lang/Throwable.html#printStackTrace()

6 Software and commands for remote login

Your Java implementation of programming assignment 2 will be tested and graded on the EECS Linux servers. Be sure you test your final submission on either `zuma.eecs.uci.edu` or `crystalcove.eecs.uci.edu`. Compile from the source on the server to ensure you are submitting code that compiles and runs. **Code that does not compile, will not be graded.** You can connect to the EECS Linux servers from virtually any computer anywhere that has internet access. What you need is a client program for *remote login*.

Use **`ssh`** as the primary way to connect to the server. **`ssh`** stands for *secure shell*, and it encrypts your network communication, so that your data cannot be understood by snoopers. For file transfers, use **`sftp`** or **`scp`**, which are secure. You could also set up an *ssh-tunnel* so that previously unencrypted communications can be encrypted. If you have a Windows machine you can download WinSCP (`https://winscp.net/eng/download.php`) which has a GUI interface for file transfer. Or use **`scp`** from the command line. Here are some examples on how to do this, `http://www.hypexr.org/linux_scp_help.php`. Depending on what computer you use, it may have a different *implementation* of **`ssh`**, but the basic function underneath are all the same.

- If you are logging in from a Windows machine, you can use **`PuTTY`** Telnet and SSH client.
  `http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`
- MacOS X already has this built-in (use Terminal or X11 to run a unix shell). Most Linux distributions also bundle **`ssh`**.
- If you are logging in from an X terminal, you can use the command below (substitute your actual EECS `username`.
  `%` **`ssh -X username@zuma.eecs.uci.edu`**
  (Note: `%` is the prompt, not part of your command) It will prompt you for your password. Note that the `-X` option allows you to run programs that open X windows on your screen.

7 Submit your work

You must turn-in a tar archive through the submission link on the EECS 114 Piazza webpage under Resources. The name of your submission must be `assn2.tgz`. No other named files will be accepted. Do not include any directories or object (e.g., *.class, executable) files. Be sure to create your tar archive with the `%`**`tar -czvf`** command. In `assn2.tgz`, there should only be the following files.

- `Main.java, WordLadder.java, MyStack.java, MyQueue.java, SimpleList.java, StackOverflowException.java, StackUnderflowException.java, QueueUnderflowException.java`

Do not forget to put a class header on every file you create or modify. Files lacking a header will not be graded. As always, re-download and test your submission. Files that are corrupted or cannot be read cannot be graded. You should **ALWAYS** turn in what you have completed thus far of the program at least 6 hours before the due date (by 6 PM). Then continue to work on your program and turn in the most current version as you get it working.