Matt Matuk

CSIT – 211

7 March 2015

Lab 3

**Programming Project PP 13.5**

-----**SPEC**-----

Design and implement an application that reads a sentence from the user and prints the sentence with the characters of each word backwards. Use a stack to reverse the characters of each word.

**SCRUM**

* Class Reverse
  + Method Main
* Class ReverseOrder
  + Variables
    - String sen
    - Stack reverse
  + Method reverseSentence()
  + Method toString()
* Class LinkedStack
* Class LinearNode
* Interface StackADT

**Class: Reverse**

1. This class contains the main method of the application.
2. This class will create a ReverseOrder object and call the reverseSentence() method to reverse the order of each word in a sentence.
3. Make sure anything that is printed looks nicely formatted.

**Import**

* Scanner

**Process**

* Variables
  + Scanner scan
  + String reverse
* Method main()
  + Create a new ReverseOrder object
  + Create a new scanner.
  + Print some intro statements for the user of the application
  + Print “Please enter a sentence to reverse the order of each word.”
  + reverse equals the return value of the method reverseSentance(scan.nextline). Pass the method sentence the user entered.
  + Print “Here is the sentence with the words reversed.”
  + Print the string reverse.
  + Print some ending statements.
  + Close the scanner

**Class: ReverseOrder**

1. This class will accept a string and return that string with the words backwards.
2. The class will use a LinkedStack to reverse the order of the words.
3. All variables are protected unless otherwise noted
4. All methods are public and void unless otherwise noted

**Import**

* Scanner
* Stack

**Process**

* Variables
  + String sen
  + Stack<String> reverse Linked stack
* Method constructor()
  + Does nothing
* Method String reverseSentance(String sentence)
  + Set sen equal to the parameter sentence
  + Create a new string stack result
  + Create a new string reverse = “”
  + Create a new scanner scan to which will can the string sen
  + While scan hasNext for the string sen
    - If the character is equal to a space
      * Pop each item in the reverse stack and append the value to the string result
    - Else
      * Add the character to the stack reverse and continue the loop until no characters are left
  + Return result

**Class: LinkedStack<T> implements StackADT<T>**

1. This Class creates a LinearNode which will be the top of a stack
2. The Nodes will point to the next node below instead of using the elements

**Import**

**Process**

* Variables
  + Int count
  + LinearNode top
* Method Constructor
  + Count = 0
  + Top = null
* Method push(T element)
  + Create a new LinearNode and pass it the element
    - Set the next node to poit to the current top
  + Top = the newly created node
  + Increase count by 1
* Method T pop()
  + Create a new generic type and set it equal to the top element
  + Set top to the current tops next element
  + Decrease count by 1
* Method T peek()
  + Return the current top element
* Method Boolean isEmpty()
  + Return true if count is 0
* Method int siz()
  + Return count

**Class: LinearNode<T>**

1. This class creates a generic element and also creates a new node for the next element
2. The class uses nodes to point to each other instead of using the actually elements
3. All variables are private
4. All method are public and void unless otherwise stated

**Import**

**Process**

* Variables
  + LinearNode<T> next
  + T element
* Method constructor()
  + Next = null
  + Element = null
* Method constructor(T elem)
  + Next = null
  + Element = elem
* Method LinearNode<T> getNext()
  + Return next
* Method setNext(LinearNode<T> node)
  + Next = node
* Method T getElement()
  + Return element
* Method setElement(T elem)
  + Element = elem

**Interface: StackADT<T>**

1. This is a interface for a StackADT
2. All methods are public

**Process**

* Methods
  + Void push(T element)
  + T pop()
  + T peek()
  + Boolean isEmpty()
  + Int size()
  + String toString()

**CODE**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Reverse.java Matt Matuk

// CSIT 211 Project 13.5 pg 546

// 1. This class contains the main method of the application.

// 2. This class will create a ReverseOrder object and call

// the reverseSentance() method to reverse the order of each

// word in a sentence.

// 3. Make sure anything that is printed looks nicely formatted.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**import** java.util.Scanner;

**public** **class** Reverse

{

**public** **static** **void** main(String[] args)

{

Scanner scan = **new** Scanner(System.***in***);

ReverseOrder reverOrder = **new** ReverseOrder();

String reverse;

String in;

System.***out***.println("Welcome to my application. \n"

+ "Today you will enter a sentance and then I "

+ "will print that sentance with each word "

+ "backwords.");

System.***out***.println();

System.***out***.println("Please enter a sentance to reverse "

+ "the order of each word.");

in = scan.nextLine();

reverse = reverOrder.reverseSentance(in);

System.***out***.println();

System.***out***.println("Here is the sentance with each word "

+ "backwords.");

System.***out***.println(reverse);

System.***out***.println();

System.***out***.println("Thank you for using my application "

+ "today.");

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// ReverseOrder.java Matt Matuk

// CSIT 211 Project 13.5 pg 546

// 1. This class will accept a string and return that string with

// the words backwards.

// 2. The class will use a LinkedStack to reverse the order of the

// words.

// 3. All variables are protected unless otherwise noted

// 4. All methods are public and void unless otherwise noted

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**import** java.util.LinkedList;

**import** java.util.Stack;

**public** **class** ReverseOrder

{

**protected** String sen;

**protected** LinkedStack<String> reverse;

**public** ReverseOrder()

{

}

//-------------------------------------------------------------------

// This method is where the passed string is reverse word by word.

//-------------------------------------------------------------------

**public** String reverseSentance(String sentance)

{

sen = sentance;

reverse = **new** LinkedStack<String>();

String result = "";

// loops thru each character in the string

**for** (**int** count = 0; count<sen.length(); count++)

{

// if the character equals a space, then the stack is poped

// until its empty.

**if** (sen.substring(count,count+1).equalsIgnoreCase(" "))

{

result = result + pop(reverse.size(), **true**);

}

// each character is added to the stack until a space is

// reached, then the stack is poped to the result string.

**else**

{

reverse.push(sen.substring(count,count+1));

}

}

// this is for the end of the sentance to clear the last word.

result = result + pop(reverse.size(),**false**);

**return** result;

}

//-------------------------------------------------------------------

// This method pops the stack clear by looping the size of the stack

//-------------------------------------------------------------------

**private** String pop(**int** times, **boolean** space)

{

System.***out***.println("\*\*\*\*\*\*\*\*" + reverse.toString());

String result = "";

**for** (**int** num = 0; num<times; num++)

{

result = result + reverse.pop();

}

**if** (space)

{

result = result + " ";

}

**return** result;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// LinkedStack.java Matt Matuk

// CSIT 211 Project 13.5 pg 546

// 1. Creates a linked stack by using the StackADT interface and

// the LinkedNode class. A linked stack uses references to other

// elements.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**import** java.util.EmptyStackException;

**public** **class** LinkedStack<T> **implements** StackADT<T>

{

**private** **int** count;

**private** LinearNode<T> top;

**public** LinkedStack()

{

count = 0;

top = **null**;

}

**public** **void** push(T element)

{

LinearNode<T> temp = **new** LinearNode<T>(element);

temp.setNext(top);

top = temp;

count++;

}

**public** T pop() **throws** EmptyStackException

{

**if** (isEmpty())

{

**throw** **new** EmptyStackException();

}

T result = top.getElement();

top = top.getNext();

count--;

**return** result;

}

**public** T peek()

{

**return** top.getElement();

}

**public** **boolean** isEmpty()

{

**return** (count == 0 ? **true** : **false**);

}

**public** **int** size()

{

**return** count;

}

**public** String toString()

{

String result = "[";

**boolean** done = **false**;

LinearNode<T> temp = **new** LinearNode<T>();

temp = top;

**while** (done == **false**)

{

**if** (temp.getNext() != **null**)

{

result = result + temp.getElement() + ", ";

temp = temp.getNext();

}

**else**

{

done = **true**;

result = result + temp.getElement() + "]";

}

}

**return** result;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// LinearNode.java Matt Matuk

// CSIT 211 Project 13.5 pg 546

// 1. This class creates a generic element and also creates a new

// node for the next element

// 2. The class uses nodes to point to each other instead of using

// the actually elements

// 3. All variables are private

// 4. All method are public and void unless otherwise stated

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **class** LinearNode<T>

{

**private** LinearNode<T> next;

**private** T element;

**public** LinearNode()

{

next = **null**;

element = **null**;

}

**public** LinearNode(T elem)

{

next = **null**;

element = elem;

}

**public** LinearNode<T> getNext()

{

**return** next;

}

**public** **void** setNext(LinearNode<T> node)

{

next = node;

}

**public** T getElement()

{

**return** element;

}

**public** **void** setElement(T elem)

{

element = elem;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// StackADT.java Matt Matuk

// CSIT 211 Project 13.5 pg 546

// 1. This is a interface for a StackADT

// 2. All methods are public

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **interface** StackADT<T>

{

**public** **void** push(T element);

**public** T pop();

**public** T peek();

**public** **boolean** isEmpty();

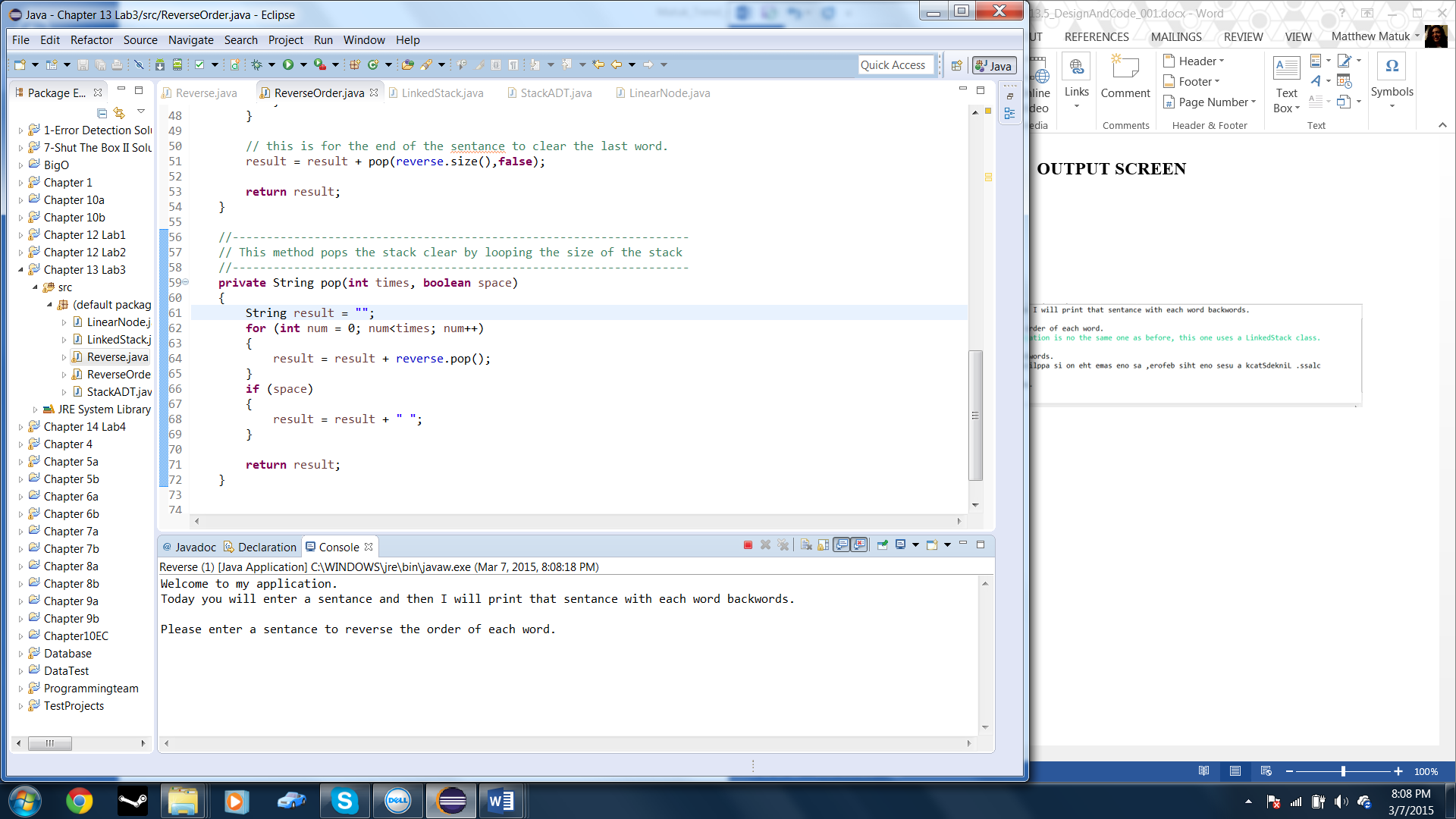
**public** **int** size();

**public** String toString();

}

**OUTPUT SCREEN**

* Starting program.



* After entering a sentence.

