**Aaron McCarley**

**Assignment 2.2 Palindrome**

Overview: In this assignment you will write a program and upload code to Github; you also will include a document uploaded to Moodle where you answer questions and conduct analysis.

In this exercise you will write a Java program that uses a stack to test whether an input string is a palindrome. A palindrome is a string that is the same in reverse as forward - ignoring case, punctuation and spaces. For example, race car is a palindrome. So is the phrase "A man, a plan, a canal: Panama." Your palindrome doesn't have to make sense, only be a string of characters that is the same forward and backward. You don't have to worry about spaces or other delimiters.

Please follow the set of steps listed below:

1. Since we know that using a stack to accomplish this task this requires LIFO (Last In, First Out), which is them item at the top is removed first and the item at the bottom remains in the same place. When we add items to the stack it is called a Push, and when we remove those items it is called a Pop. The newest item at the top of the stack is called the Top. For the Palindrome problem, which is a word, number, phrase, of other sequence of characters that is read the same backwards or forward (White, Palindrome). Example words such as KAYAK, STATS, TENET all are Palindrome because the characters can be expressed the same backwards and forward.

Each Character expressed in the string is pushed onto the stack from top to bottom

String: KAYAK

|  |
| --- |
| K |
| A |
| Y |
| A |
| K |

Each character in the String is pooped of the stack and appended to the new String. Since it is spelled the same forwards and backwards in this case the strings are equal and is a Palindrome.

New String: KAYAK

|  |
| --- |
| K |
| A |
| Y |
| A |
| K |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| emptyStack | Push | Push | push | pop |

For my coding example I am using a StackInterface and an ArrayStack that implements the method for the array stack for the palindrome.

3. (60 Points) Now write your Java program. The program should accept a palindrome from the user, check it using a stack, and return whether the string is a palindrome or not. your program should include some output so I can see it works. Remember to include Javadocs and the formal structure from CS131.

/\*\*

\* Java API Interface List <E> An ordered collection (also known as a sequence).

\* Provides the details of each method

\* **@author** Aaron McCarley

\* **@version** 1.0

\* **@param** <T>

\*/

**public** **interface** StackInterface<T> {

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

**public** T pop();

**public** T peek();

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

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

**public** **void** capacity();

}// end Stack Interface

/\*\*

\* Provides the details of each method

\* and the stack array for a Palindrome.

\* **@author** Aaron McCarley

\* **@version** 1.0

\* **@param** <T>

\*/

**import** java.util.Arrays;

**import** java.util.EmptyStackException;

**import** java.util.LinkedList;

**import** java.util.Queue;

**import** java.util.Scanner;

**public** **class** ArrayStack<T> **implements** StackInterface<T>{

**private** **final** **int** DEFAULT\_CAPACITY = 20;

**private** T[] stack;

**private** **int** top;

/\*\* Create an Empty Stack \*/

@SuppressWarnings("unchecked")

**public** ArrayStack()

{

top = 5;

stack = (T[])(**new** Object[DEFAULT\_CAPACITY]);

}

/\*\* Specify the Capacity for the Stack \*/

@SuppressWarnings("unchecked")

**public** ArrayStack (**int** initialCapacity)

{

top = 5;

stack = (T[])(**new** Object[initialCapacity]);

}

/\*\*

\* Adds the specified element to the top of the stack. \*/

**public** **void** push (T newEntry)

{

**if** (size() == stack.length)

capacity();

stack[top] = newEntry;

top++;

}

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

{

System.***out***.print("Enter any String total: ");

@SuppressWarnings("resource")

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

String firstString = linkedList.nextLine();

Queue<Character> queue = **new** LinkedList<>();

**for** (**int** n = firstString.length()-1; n >=0; n--)

{

queue.add(firstString.charAt(n));

}

String isString = "";

//Pop all characters from stack one by one and build reverse string

**while** (!queue.isEmpty()) {

isString = isString + queue.remove();

}

**if** (firstString.equals(isString)) {

System.***out***.println("This String is a palindrome.");

} **else** {

System.***out***.println("This String is not a palindrome.");

}

}

/\*\* Removes the element at the top of the stack and returns a

\* reference to it. Throws an EmptyStackException if the stack

\* is empty. \*/

@Override

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

top--;

T result = stack[top];

stack[top] = **null**;

**return** result;

}

/\*\*

\* Reference to the element at the top of the stack.

\* Throws EmptyStackException if the stack is empty. \*/

@Override

**public** T peek() **throws** EmptyStackException

{

**if** (isEmpty())

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

**return** stack[top-1];

}

/\*\*

\* Returns true if the stack is empty

\* false if not empty. \*/

@Override

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

{

**return** (top == 0);

}

/\*\*

\* Returns the number of elements in the stack.

\*/

@Override

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

{

**return** top;

}

@SuppressWarnings("unchecked")

**public** **void** capacity()

{

T[]length = (T[])(**new** Object[stack.length]);

**for** (**int** index= 1; index <= stack.length; index++)

length[index] = stack[index];

stack = length;

}

@Override

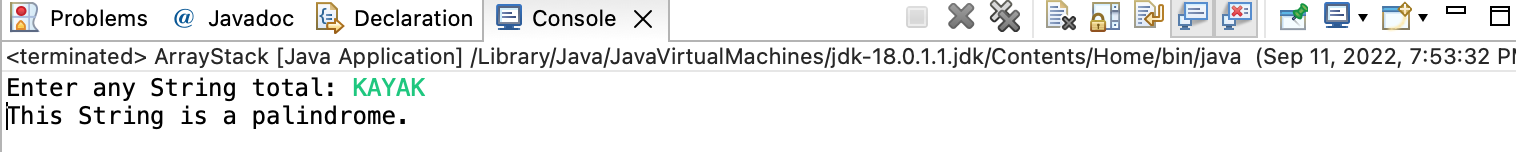
**public** String toString() {

**return** "ArrayStack [stack=" + Arrays.*toString*(stack) + ", top=" + top + "]";

}//end toString

}//end class

**Output below when you run the code.**



5. Thinking in terms of the material regarding computational complexity that we just finished covering answer the following in the document portion of your homework:

a. (5 Points) What is the basic operation in your code?

**Push to insert an element into the stack, then Pop to remove an element from the stack, and top to return the element to the stack. This is the basic operation, but I also implemented size to return the size of the stack, isEmpty that returns true if the stack is empty and false otherwise, and capacity which returns stays within the length of the stack.**

b. (5 Points) What input determines how many times the basic operation takes place?

**Input size (n) as the total number of items present in the input.**

c. (5 Points) Express the number of times the basic operation occurs in terms of n. What does n represent?

**The input size n for the basic operation to occur is O(n4).**

d. (5 Points) What is the computational complexity of the code in terms of Big O? Explain why.

**I believe it is Linenar Complexity based on the fact that it is a stack, and I am using a for loop and performing the same set of actions for every item that in the input array. It is linear in complexity so it would be O(n).**

Text

Description automatically generated

5. (10) Points) Explain how a stack data structure differs from a bag data structure. Explain how you used the stack in your code to solve the palindrome problem.

**A stack data structure differs in that it is a linear data structure and pushed/inserted from the top and can be removed only from top to bottom. A Bag Data structure allows you to have duplicates and where removing items is not removable. A stack has empty stack created initially then you can add an item, remove the most recent added item from the top (LIFO) verify if the stack is empty and the number of items still in the stack. A Bag data structure allows you to create an empty bag, add an item, check to see if bag is empty, and see what items are in the bag, but it does not meant to remove items.**

**I did go through the material to see what a Palindrome was for a stack and started using different combinations of characters that I could use to meet the requirement. ABBA, KAYAK, TENET, STATS which are all able to be used forward and backwards. I noticed a few methods that could be used in the material for implementing an interface, the ADT Stack, LinkedStack, etc. and decided to use an interface and an ArrayStack to create my code.**

**I used the interface List directions from the material and from Java API Interface List for ordered collections (Oracle, 2022). I used a stack of string objects that used generics with parameters. I did not use a node class or quene classes because what was asked for in this assignment was basic enough I did not think it was a requirement to sue more than a class and an interface.** Once the entry is put in the stack entry it outputs whether or not the the String is a Palindrome or not. If I put in something like mentioned above ABBA it will outputs it is a Palindrome, and if I put something in like dede below it will not be a palindrome.

Graphical user interface, application

Description automatically generated

Include your Word document in your Eclipse Repository in a sub-directory called Word.  Commit your project to a GitHub repository called **Palindrome\_XXX** where **XXX** are you initials.

References

1. Mary White. Palindrome Examples: Fun Forward and Backward Words.

https://examples.yourdictionary.com/palindrome-examples.html

1. Oracle Docs- Interface List<E>. https://docs.oracle.com/javase/8/docs/api/java/util/List.html

1993, 2022.