GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computing & Mathematics

CP2SD – Data Structures & Algorithms
Lab 3: Stacks

Stacks are ubiquitous data structures in computer science and are applied anytime last-in-first-out (LIFO) functionality is required. Only the top element of a stack is assessable, resulting in a limited number of methods, each with a running time of **O**(1). In this practical, we will illustrate the utility of stacks by demonstrating how they can be applied to determine if a sequence is palindromic and to check an algebraic expression for balanced parenthesis.

- Download the Zip archive ***lab3Stacks.zip*** from Moodle and extract the files into a folder on your hard disk. As the classes involved are all packaged, ensure that you create the correct directory structure and study the source code before you compile.

- Examine the class *Stack.java*. Note that the class is parameterized and contains all the essential stack methods – *push(), pop(), peek()* and *isEmpty()*. Make sure you understand how the stack operates before proceeding.

- Create a new class called ***Test.java*** with a main method. Instantiate the *PalindromeFinder* class using the syntax below.

  ```
  PalindromeFinder pf = new PalindromeFinder("NAVAN");
  System.out.println("Is a palindrome: " + pf.isPalindrome());
  ```

  The LIFO property of stacks ensures that items are removed from the stack in the reverse order in which they were pushed. Note the ***while*** loop in the *buildReverse()* method. This is a common way to access all the elements of a stack or a queue.

- As discussed in class, a stack structure can be used create a parenthesis validator. Instantiate and execute the *ParenthesesChecker* class as follows:

  ```
  ParenthesesChecker pc = new ParenthesesChecker();
  System.out.println("Balanced? " +
          pc.isBalanced("( a + b * ( c / ({ d} – e ) ) ) + (d / e )"));
  ```

Exercises

- Using the *Dictionary* class and *dictionary.txt* file from Lab 1, create a stack of the entire set of words by iterating over the string array returned by *getSortedWords()* and pushing each word onto the stack. After the stack has been created, use a ***while*** loop to transverse the stack looking for all words that contain the letters "a" and "e" as their 2nd and 4th letters respectively. What is the running time for such as search? Are there any additional properties of the stack that could be manipulated to speed up the search?
- Examine the Stack class provided in the ***java.util*** library (the source code is listed below). Pay particular attention to the design approach used.

```
package java.util;
public class Stack<E> extends Vector<E> {
	private static final long serialVersionUID = 1224463164541339165L;
	public Stack() {
		super();
	}
	public boolean empty() {
		return isEmpty();
	}
	public synchronized E peek() {
		try {
			return (E) elementData[elementCount - 1];
		} catch (IndexOutOfBoundsException e) {
			throw new EmptyStackException();
		}
	}
	public synchronized E pop() {
		if (elementCount == 0) {
			throw new EmptyStackException();
		}
		final int index = --elementCount;
		final E obj = (E) elementData[index];
		elementData[index] = null;
		modCount++;
		return obj;
	}
	public E push(E object) {
		addElement(object);
		return object;
	}
	public synchronized int search(Object o) {
		final Object[] dumpArray = elementData;
		final int size = elementCount;
		if (o != null) {
			for (int i = size - 1; i >= 0; i--) {
				if (o.equals(dumpArray[i])) {
					return size - i;
				}
			}
		} else {
			for (int i = size - 1; i >= 0; i--) {
				if (dumpArray[i] == null) {
					return size - i;
				}
			}
		}
		return -1;
	}
}
```