**Lesson 12 – Stacks II**

**Read Chapter 7, Sections 4-6 of the text.**

**Learning Objectives:**

* Write an array-based implementation of the ADT Stack.
* Use a stack to solve a problem.
* Describe a reference-based implementation of the ADT Stack.

**A Reference-based Implementation of ADT Stack:**

* We will use the same exception and interface that we used last lesson for the array-based implementation:

public class StackException extends RuntimeException {

public StackException(String s)

{

super(s);

}

}

public interface StackInterface {

public boolean isEmpty();

// Determines whether the stack is empty

public void popAll();

// Removes all the items from the stack

public void push(Object newItem) throws StackException;

// Adds an item to the top of a stack

public Object pop() throws StackException;

// REmvoes the top of a stack

public Object peek() throws StackException;

// Retrieves the top of a stack

}

* Let’s re-implement the Node class to support ADT Stack:

public class Node {

private Object item;

private Node next;

public Node(Object newItem)

{

item = newItem;

next = null;

}

public Node(Object newItem, Node nextNode)

{

item = newItem;

next = nextNode;

}

public Object getItem()

{

return item;

}

public Node getNext()

{

return next;

}

public void setNext(Node nextNode)

{

next = nextNode;

}

}

* Now, to implement the interface:

public class **StackReferenceBased**<E> implements Stackable<E> {

    Node<E> top;

    public StackReferenceBased() {

**this**.top = *null*;

    }

    @Override

    public boolean isEmpty() {

        return **this**.top == *null*;

    }

    @Override

    public void popAll() {

**this**.top = *null*;

    }

    @Override

    public void push(E newItem) {

        Node<E> currentTop = **this**.top;

**this**.top = new Node<E>(newItem);

**this**.top.setNext(currentTop);

    }

    @Override

    public E pop() throws StackException {

        if (**this**.isEmpty()) {

            throw new StackException("Error: stack empty!");

        } else {

            E topItem = top.getItem();

            top = top.getNext();

            return topItem;

        }

    }

    @Override

    public E peek() throws StackException {

        if (**this**.isEmpty()) {

            throw new StackException("Error: stack empty!");

        } else {

            return top.getItem();

        }

    }

    @Override

    public String toString() {

        String s = "[";

        Node<E> current = **this**.top;

        while (current != *null*) {

            s = s + current.getItem().toString();

            current = current.getNext();

            if (current != *null*)

                s = s + ", ";

        }

        s += "]";

        return s;

    }

}

* As you can see, this implementation is much simpler than the reference-based implementation for ADT List. Our operations are restricted to one end of the data structure.

**Use ADT Stack to solve a problem:**

* Let’s use our implementation to solve a common problem in computer science; check to see if opening and closing parentheses match.

**ParaChecker.java**

    private boolean paraMatch(String eq) {

        StackReferenceBased<Character> parStack = new StackReferenceBased<Character>();

        for (int i = 0; i < eq.length(); i++) {

            if (eq.charAt(i) == '(')

                parStack.push('(');

            else if (eq.charAt(i) == ')') {

                if (parStack.isEmpty())

                    return *false*;

                else

                    parStack.pop();

            }

        }

        // true if ( matched a ) and false otherwise

        return parStack.isEmpty();

    }

* Using two stacks we can evaluate equations **ParaChecker.java**

**In-class exercises using ADT Stack:**

* Write a program that, given a string, uses a stack to reverse the characters in the string.

**StringReverser.java**

    private String reverse(String text) {

        StackReferenceBased<Character> stack = new StackReferenceBased<Character>();

        for (int i = 0; i < text.length(); i++)

            stack.push(text.charAt(i));

        String r = "";

        while (!stack.isEmpty())

            r += stack.pop();

        return r;

    }

* Given a stack containing a random assortment of integers, write a program that sorts a stack using only the original stack, a second temporary stack and one other variable:

// import java.util.Random;

public static void ice2()

{

Random r = new Random();

StackReferenceBased stack = new StackReferenceBased();

for (int i = 0; i < 50; i++)

stack.push(r.nextInt(1000));

StackReferenceBased tempStack = new StackReferenceBased();

tempStack.push(stack.pop());

while (!stack.isEmpty()) {

if ((Integer)stack.peek() >= (Integer)tempStack.peek())

tempStack.push(stack.pop());

else {

int x = (Integer)stack.pop();

while (!tempStack.isEmpty() && ((Integer)tempStack.peek() > x))

stack.push(tempStack.pop());

tempStack.push(x);

}

}

while (!tempStack.isEmpty()) {

stack.push(tempStack.pop());

System.out.println(stack.peek());

}

}