**Jeremy Cooley**

**ITEC 2150 – Price**

**IC 7 Practice IT problems**

* **Self-check 14.1 – stackQueueStatements**

Which of the following statements about stacks and queues is true?

A queue's remove method removes and returns the element at the front of the queue.

* **Self-check 14.13 - stackQueueUsage1**

Output: [you, are, how]

* **Self-check 14.16 - stackQueueMystery1**

[2, 6, 1] 🡪 [1,1,6,6,2,2]

[42, -3, 4, 15, 9] 🡪 [9,9,15,15,4,4,-3,-3,42,42]

[30, 20, 10, 60, 50, 40] 🡪 [40,40,50,50,60,60,10,10,20,20,30,30]

* **Exercise 14.1 – splitStack**

public Stack splitStack(Stack<Integer> s) {

ArrayList<Integer> posArray = new ArrayList<>();

ArrayList<Integer> negArray = new ArrayList<>();

while(!s.isEmpty()) {

if (s.peek() >= 0) { posArray.add(s.pop()); }

else if (s.peek() < 0) { negArray.add(s.pop()); }

}

for (Integer i = 0; i <= negArray.size() - 1; i++) { s.push(negArray.get(i)); }

for (Integer i = 0; i <= posArray.size() - 1; i++) { s.push(posArray.get(i)); }

return s;

}

* **Exercise 14.2 – stutter**

public Stack stutter(Stack<Integer> s) {

Integer n = 0;

Queue<Integer> q;

while (!s.isEmpty()) {

n = s.pop();

q.add(n);

q.add(n);

}

while (! q.isEmpty()) {

n = q.remove();

s.push(n);

}

return s;

}

* **Exercise 14.3 – copyStack**

public Stack<Integer> copyStack(Stack<Integer> s) {

Integer n = 0;

Queue<Integer> newQueue;

Stack<Integer> newStack;

while (!s.isEmpty()) {

n = s.pop();

newQueue.add(n);

}

while (! newQueue.isEmpty()) {

n = newQueue.remove();

newStack.push(n);

}

return newStack;

}

* **Exercise 14.4 – collapse**

public Stack<Integer> collapse (Stack<Integer> s) {

Integer first;

Integer second;

Integer topNumIfOdd = 0;

Stack<Integer> newStack = new Stack<Integer>();

boolean isOdd = (s.size() % 2 == 1);

if (isOdd) { topNumIfOdd = s.pop(); }

while (!s.isEmpty()) {

first = s.pop();

second = s.pop();

Integer sum = first + second;

newStack.push(sum);

}

while (!newStack.isEmpty()) { s.push(newStack.pop()); }

if (isOdd) { s.push(topNumIfOdd); }

return s;

}

* **Exercise 14.5 – equals**

public boolean equals(Stack<Integer> s1, Stack<Integer> s2) {

boolean isEqual = true;

Stack<Integer> tempStack = new Stack<Integer>();

if (s1.size() != s2.size()) { isEqual = false; return isEqual; }

while (!s1.isEmpty()) {

Integer firstInt = s1.pop();

Integer secondInt = s2.pop();

if (firstInt != secondInt) { isEqual = false; }

tempStack.push(firstInt);

tempStack.push(secondInt);

}

while (!tempStack.isEmpty()) {

s2.push(tempStack.pop());

s1.push(tempStack.pop());

}

return isEqual;

}

* **Exercise 14.8 isPalindrome**

public boolean isPalindrome(Queue<Integer> q) {

boolean palindrome = true;

ArrayList<Integer> holdingAL = new ArrayList<Integer>();

if (q.size() == 0 || q.size() == 1) { return true; }

if (q.size() == 2) {

Integer first = q.remove();

Integer second = q.remove();

palindrome = first == second;

q.add(first);

q.add(second);

}

Integer fullSize = q.size();

boolean isOdd = (fullSize%2 == 1);

Integer oddOneInTheMiddle;

Integer halfSize;

if (isOdd) {

halfSize = (fullSize - 1) / 2;

} else {

halfSize = fullSize/2;

}

for (int i = 1; i <= halfSize; i++) {

holdingAL.add(q.remove());

}

if (isOdd) {

oddOneInTheMiddle = q.remove();

holdingAL.add(oddOneInTheMiddle);

}

for (int j = 0; j <= halfSize - 1; j++) {

Integer queueInt = q.remove();

Integer arrayInt = holdingAL.get(j);

holdingAL.add(queueInt);

if ( arrayInt != queueInt ) {

palindrome = false;

}

}

for (int k = 0; k <= holdingAL.size() - 1; k++) { q.add(holdingAL.get(k)); }

return palindrome;

}

* **Exercise 14.10 – isConsecutive**

public boolean isConsecutive(Stack<Integer> s) {

    boolean consecutive = true;

    if (s.size() == 0 || s.size() == 1) { return consecutive; }

    Stack<Integer> holdingStack = new Stack<Integer>();

    Integer top;

    Integer bot;

    while (!s.isEmpty()) {

         top = s.peek();

         holdingStack.push(s.pop);

         if (s.isEmpty())  {  break;   }

         bot = s.peek();

         if (top - bot != 1)   {   consecutive = false;    }

    }

    while (!holdingStack.isEmpty())  {

   Integer stackNum = holdingStack.pop();

        s.push(stackNum);

    }

    return consecutive;

}