QUEENS COLLEGE — Department of Computer Science

CSCI 313 — Practice problems on Stacks and Queues

Instructor: Alex Ryba

These are some of the solutions to practice problems. Not all problems have solutions here.

Solutions to some older problems might not make use of generics. Generics are now required in this course.

Problem 1 The generic class Queue is to be programmed with an array based implementation. A partial version of the implementation follows. The two most important methods have been omitted. The treatment of an empty Queue (as given by the constructor) is different from the one we used in class. Make sure that your methods work correctly with the choices made by the constructor.

```
public class P2<T> implements Queue<T> {
   private T data[];
   private int front, rear, size;
   public P2() { data = (T[]) new Object[100]; front = size = 0; rear = -1; }
   public int size() { return size; }
   public boolean empty() { return size == 0; }
   // methods omitted here
}
```

(a) **Identify the two missing methods**. For each give the name, parameters and return type.

Answer:

```
   public T dequeue() throws Exception
   public void enqueue(T x) throws Exception
```

(b) **Give a complete implementation** of **ONE** of the two missing Queue methods. (You can choose either of them.)

Answer:

```
   public T dequeue() throws Exception {
      if (size() <= 0) throw new Exception("Queue Empty");
      size--;
      T ans = data[front++];
      if (front == 100) front = 0;
      return ans;
   }

   public void enqueue(T x) throws Exception {
      if (size() >= 100) throw new Exception("Queue Full");
      rear++;
      if (rear == 100) rear = 0;
      data[rear] = x;
      size++;
   }
```

Problem 2 (i) Write the Java interface for the ADT Stack. (Just write an interface that specifies methods — **do not** implement the methods.)

Answer:

```
interface Stack {
   public Object pop();
   public void push(Object x);
   public int size();
   public boolean empty();
}
```

(ii) Write the Java interface for the ADT Iterator. (Just write an interface that specifies methods — **do not** implement the methods.)

Answer:

```
interface Iterator {
   public Object next();
   public boolean hasNext();
   public void remove();
}
```

(iii) Write a client function with title line:

```
public static int count(Stack s, Object x)
```

The function returns a count of the number of times the object x is found on the Stack s. When the method ends the Stack should store exactly the same data as at the beginning of the method. However your function will need to alter the Stack data as it runs.

Answer:

```
   public static int count(Stack s, Object x) {
     Stack temp = new Stack();
     int ans = 0;

     while (!s.empty()) {
        Object y = s.pop();
        if (y.equals(x)) ans++;
        temp.push(y);
     }
     while (!temp.empty()) s.push(temp.pop());
     return ans;
   }
```

(iv) Write a client function with title line:

```
public static int count(Iterator i, Object x)
```

The function returns a count of the number of times the object x is found on the Iterator i. This function can and should empty out the Iterator and should not restore it to its original state.

Answer:

```
   public static int count(Iterator i, Object x) {
     int ans = 0;
     while (i.hasNext()) {
        Object y = i.next();
        if (y.equals(x)) ans++;
     }
     return ans;
   }
```

Problem 3 Consider the following Java program that uses a standard Stack, Iterable Queue and Priority Queue as studied in class.

```
  public static void main(String args[]) throws Exception {
    Stack<Integer> s = new ArrayStack<Integer>();
    ItQueue<Integer> q = new ItQueue<Integer>();
    PriorityQueue<String> p = new HeapPriorityQueue<String>();
    for (int i = 5; i < 10; i++) {
      s.push(i);
      q.enqueue(i + 10);
    }
    System.out.println(q.dequeue());                        // line (a)
    System.out.println(s.pop());                            // line (b)
    System.out.println(q.dequeue() + q.dequeue());          // line (c)
    for (Integer x:q) {
      int y = s.pop();
      p.add("" + y);
      p.add("" + x);
      System.out.print(x);                                // line (d)
    }
    System.out.println();
    for (int i = 1; i <= 4; i++)
      System.out.print(p.removeMin());                    // line (e)
    System.out.println();
  }
```

(a) What is the ouput printed from line (a)?

Answer: 15

(b) What is the ouput printed from line (b)?

Answer: 9

(c) What is the ouput printed from line (c)?

Answer: 33

(d) What is the ouput printed from line (d)?

Answer: 1819

(e) What is the ouput printed from line (e)?

Answer: 181978

2 points per part no partial credit (except perhaps for extra spaces or lines in the middle of output).

Problem 4 Consider the following method that adds some elements from an array with at least n elements to an Iterable Stack. You should assume that a standard stack iterator begins with the element at the bottom of the stack and ends with the element at the top — this is the behavior of `java.util.Stack`.

```
public static Stack<Integer> method(int array[], int n) {
    Stack<Integer> answer = new Stack<Integer>();
    if (n == 0) return answer;
    for (int i = 0; i < n; i+=2)
       answer.push(array[i]);
    Stack<Integer> temp = method(array, n/2);
    for (Integer x:temp) answer.push(x);
    return answer;
}
```

(a) What is the output when the method is called in the following:

```
public static void main(String args[]) {
    int a[]={1, 2, 3, 4, 5, 6, 7, 8};
    Stack<Integer> s = method(a, 8);
    for (Integer x:s) System.out.print(x);
    System.out.println();
}
```

Answer: 13571311

(b) Give the recurrence relation that is satisfied by the run time $t(n)$ of the method.

Answer:

$$t(n) = t(n/2) + \Theta(n)$$

(c) Apply the Master Theorem to your answer to (b) to give a useful Θ estimate for $t(n)$.
Answer:

$$t(n) = \Theta(n)$$

Problem 5 (a) Write complete code for the standard interface Iterator.
Answer:

```
interface Iterator<T> {
   boolean hasNext();
   T next();
   void remove();
}
```

(b) Write complete code for the `class ListAdapterStack`.
Answer:

```
class ListAdapterStack<T> implements Stack<T> {
   LinkedList<T> l;
   public ListAdapterStack() { l = new LinkedList<>(); }
   public void push(T t) {l.addHead(t); }
   public T pop() {return l.removeHead();}
}
```

For the following binary tree:

```
           1
          / \
         2   3
        /     \
       4       5
      / \
     6   7
    /   / \
   8   0   9
```

(c) What is the height of the tree?
Answer: 4

(d) What is the depth of the node that stores 7?
Answer: 3

(e) What is the postorder traversal? **Answer:** 8609742531

Problem 6 The following is part of a circular array based model for a `Deque`. The method `addLast` has logic errors and does not operate correctly. Other required methods have been entirely omitted.

```
public class Deque<T> {
  private T data[];
  private int front, rear, size;
  // the next items to be added at front and rear will be placed at indices front and rear

  public Deque() {   // This is correct code.
```

```
    data = (T[]) new Object[100];
    front = size = 0; rear = 1;
  }

  public T addLast(T x) throws Exception {  // This has errors.
    ++size;
    if (size > 100) throw Exception("Deque Full");
    data[++rear] = x;
    if (rear > 100) rear = 0;
  }

  // other methods omitted

}
```

(a) Correct the code for `addLast`. Your solution should not use more more than one additional line of code and should be based on that given. It must correct any errors.

Answer:

```
  public void addLast(T x) throws Exception {
    if (size >= 100) throw new Exception("Deque Full");
    data[rear++] = x;
    if (rear >= 100) rear = 0;
    ++size;
  }
```

(b) Give a title line for each required method that has been omitted. (Credit will be given for the 3 most important methods only.)

Answer:

```
  public void addFirst(T x) throws Exception {
  public T removeFirst() throws Exception {
  public T removeLast() throws Exception {
```