German University in Cairo
Computer Science Department
Prof. Dr. Slim Abdennadher

Data Structures and Algorithms, Winter Term 2007-2008
Practice Assignment 5

Discussion: 25.11.2006 - 30.11.2006

Exercise 5-1 To be discussed in the tutorial
Deque

Chapter 4 of the textbook (page 143) describes a doubly-ended queue, which is often called a deque.

In class you have seen and implemented both stack (LIFO) and queue (FIFO) abstract data types. A deque (pronounced "deck") is an abstract data type that combines what a stack can do, and what a queue can do. Thus, you are allowed to add items to either end of a deque, and inspect/remove them from either end.

Implement the Deque ADT using an array used in a circular fashion (as was done for the queue ADT).

The interface for a deque is the following:

```
public interface Deque {
   public int size();
   /**
  * Returns true if the deque is empty, false otherwise.
  */
  public boolean isEmpty();
   /**
  * Inserts an object at the beginning of the deque.
  */
   public void insertFirst(Object o);
    /**
  * Inserts an object at the end of the deque.
  */
   public void insertLast(Object o);
   /**
  * Remove the object at the begining  of the deque.
  */
   public Object removeFirst();
  /**
  * Remove the object at the end of the deque.
  */
   public Object removeLast();
   /**
  * Returns the front object of the deque.
  */
   public Object first();
   /**
  * Returns the last object of the deque.
  */
  public Object last();
}
```

Be sure to test your class implementation thoroughly. Show output demonstrating that each method of your implementation is correct.

Solution:

```
public class Deque{
    private int maxSize;
    private Object[] queArray;
    private int front;
    private int rear;
    private int nItems;

    public Deque(int s){
        maxSize=s;
        queArray=new Object[maxSize];
        front=1;  rear=0;   nItems=0;
    }

    public void insertFirst(Object j) {
        if(front == 0) front = maxSize;
        queArray[--front] = j;  nItems++;
    }

    public void insertLast(Object j)  {
     if(rear == maxSize - 1) rear = -1;
        queArray[++rear] = j;   nItems++;
    }

    public Object removeFirst() {
        Object temp=queArray[front];
        if(front == maxSize - 1) front = -1;
        ++front;
        nItems--;
        return temp;
    }

    public Object removeLast()  {
        Object temp=queArray[rear];
        if(rear == 0)   rear = maxSize;
        --rear;
        nItems--;
        return temp;
    }

    public Object peekFront()  {
        return queArray[front];
    }

    public Object peekRear() {
        return queArray[rear];
    }

    public boolean isEmpty()  {
        return (nItems==0);
    }

    public boolean isFull()  {
        return (nItems==maxSize);
    }

    public int size()  {
        return nItems;
    }
}
```

Exercise 5-2 To be discussed in the tutorial

Mirror using Queues

Write a method `mirror` that takes as a parameter a queue containing a sequence of integers and that replaces the sequence in the queue with its mirror image. The mirror image of a sequence contains the original sequence in reverse order followed by the original sequence. For example, if the original sequence is:

```
1, 8, 15, 7, 2
```

the mirror image is:

```
2, 7, 15, 8, 1, 1, 8, 15, 7, 2
```

Notice that the mirror-image has twice as many elements as the original sequence because it contains both the original sequence and the reversed sequence. Also notice that the reversed sequence comes first.

Solution:

```
class QMirror {
  final static int size = 5;

  public static queueObj mirror(queueObj original)
  {
    queueObj mirror = new queueObj(2 * size);
    queueObj rev = new queueObj(size);
    int i = 0; int max = original.size();
    while (i < max)
    {
      rev.enqueue(original.peek());
      original.enqueue(original.dequeue());
      i++;
    }
    while(!rev.isEmpty())
    {
      for (i = 0; i < rev.size()-1; i++)
        rev.enqueue(rev.dequeue());
      mirror.enqueue(rev.dequeue());
    }
    while(!original.isEmpty())
      mirror.enqueue(original.dequeue());
    return mirror;
  }

  public static void main(String[] args){
    queueObj firstQ = new queueObj(size);
    firstQ.enqueue(new Integer(1));
    firstQ.enqueue(new Integer(8));
    firstQ.enqueue(new Integer(15));
    firstQ.enqueue(new Integer(7));
    firstQ.enqueue(new Integer(2));

   queueObj mirrorQ = mirror(firstQ);
    while(!mirrorQ.isEmpty())
        System.out.print(((Integer)mirrorQ.dequeue()).intValue()+"\t");
  }
}
```

Exercise 5-3 To be discussed in the lab
Palindrome using Stacks and Queues

You are required to create a class that can detect palindromes. A palindrome is a word or sentence that spells the same forwards as backwards.
There are many ways to detect if a phrase is a palindrome. the methods that you will use in this assignment is by using a stack and a queue.This works in the following way: Pushhalf of the characters onto the stack and enqueue the second half in the queue, then compare the characters as you pop and dequeue from the stacks and queue.
Create a class called Palindrome that has a single methid called isPalindrome(String...). Inside this method create a stack and a queue, and determine if the given String is a palindrome.

Solution:

Exercise 5-4 To be discussed in the lab
Shift the Zeros

Write a static method shiftZeroes which takes an instance of the Queue class. The method should take all the zeroes in a queue and place them at the back. For instance, if we have the following instance of Queue called queue with the sequence

```
[5, 0, 1, 4, 3, 0, 0, 6, 1, 0]
```

we could call `shiftZeroes(queue)`, and the sequence would be

```
[5, 1, 4, 3, 6, 1, 0, 0, 0, 0]
```

after the call. **Note** that the order of the non-zero elements stays the same.

The only objects you may use to solve this problem are instances of the Stack and Queue classes. Assume the classes have the usual methods (Queue has `enqueue`, `dequeue`, `peek`, and `isEmpty`; Stack has `pop`, `peek`, `push`, and `isEmpty`).

Solution:

```
//Using the simplest Queue ( Queue of integers )

public class ShiftZeros
{
    public static void shiftZeroes(Queue mainQ)
    {
        int max=mainQ.size();
        Queue numbers=new Queue(max);
        Queue zeroes=new Queue(max);

        while(!mainQ.isEmpty())
        {
            if(mainQ.peekFront()==0)
                zeroes.enqueue(mainQ.dequeue());
            else
                numbers.enqueue(mainQ.dequeue());
        }

        while(!numbers.isEmpty())
            mainQ.enqueue(numbers.dequeue());

        while(!zeroes.isEmpty())
            mainQ.enqueue(zeroes.dequeue());

    }
```

```
    public static void displayQueue(Queue qu)
    {
        while(!qu.isEmpty())
            System.out.print(qu.dequeue()+" ");
    }

    public static void main(String[] args)
    {
        Queue qu=new Queue(10);

        qu.enqueue(5);
        qu.enqueue(0);
        qu.enqueue(1);
        qu.enqueue(4);
        qu.enqueue(3);
        qu.enqueue(0);
        qu.enqueue(0);
        qu.enqueue(6);
        qu.enqueue(1);
        qu.enqueue(0);

        shiftZeroes(qu);
        displayQueue(qu);
    }
}
```

Exercise 5-5 To be discussed in the lab

Anagrams

An anagram is a word formed by reordering the letters of another word. For example, the word `post` is an anagram of `stop`. Your task is to implement a method that takes as input two strings and returns either `true` meaning that they are anagrams or `false` meaning that they are not. You should use the following technique to obtain your results:

- for each of the two input strings enqueue all the characters forming the string to an initially empty queue,
- now that you have obtained two queues, each representing a string, iterate through one of the queues by dequeuing a character from its front and enqueuing it again at the rear, and repeating repeating this process as many times as needed,
- if the front character of both queues is the same, dequeue the front character of each of the queues,
- in case you have iterated through the whole queue and the front characters were never equal then you have determined that the strings are not anagrams,
- finally if you end up with two empty queues, then you have determined that the strings are anagrams.

Use the following header for your method: `static boolean anagrams(String a, String b)`

Solution:

```
class Anagrams
{
   static boolean anagrams(String a, String b)
   {
      int lenA = a.length();
      int lenB = b.length();
      if(lenA != lenB)
         return false;
```

```
        ArrayQueue q1 = new ArrayQueue(lenA);
        ArrayQueue q2 = new ArrayQueue(lenB);
        for(int i = 0; i < lenA; i++)
        {
           q1.enqueue(new Character(a.charAt(i)));
           q2.enqueue(new Character(b.charAt(i)));
        }
        while(!q1.isEmpty())
        {
           Character c1 = (Character) q1.peek();
           Character head = (Character) q2.dequeue();
           Character c2 = head;
           while(!c1.equals(c2))
           {
              q2.enqueue(c2);
              c2 = (Character) q2.dequeue();
              if(c2 == head)
                 return false;
           }
           q1.dequeue();
        }
        return true;
     }

     public static void main(String args [])
     {
        System.out.println(anagrams("stop", "post"));
        System.out.println(anagrams("stop", "spot"));
        System.out.println(anagrams("abcd", "dbad"));
     }
  }
```