Study Guide CS265 Exam 3

Note 1: please bring this study guide to your exam.

Note 2: for Stack and Queue classes, you are only allowed to use both classes that are provided at the back and you can find both of them from the attached two java files.

1. Consider the following statements for an integer stack:

```
Stack S;
int x, y;
```

Show what is output by the following segment of code.

```
x = 5;
S.push(8);
S.push(x + 3);
y = S.peek( );
S.push(2 * y);
S.pop( );
x = S.peek( );
S.push(S.peek( ));
While(!S.empty())
    System.out.println("S = " + S.pop());
```

2. Consider the following statements for an integer queue:

```
Queue Q;
int x, y;
```

Show what is output by the following segment of code.

```
x = 5;
Q.Enqueue(8);
Q.Enqueue(x + 3);
y = Q.Front( );
Q.Enqueue(2 * y);
Q.Dequeue( );
x = Q.Front( );
Q.Enqueue(Q.Front( ));
System.out.println("Q = " + Q.toString());
```

3. Write a recursive function that calculates the summation of an array:

```
public static double sumofarray(double [ ] a, int first, int
last)
```

4. Write a non-member to delete every occurrence of a specified item from a stack, leaving the order of the remaining items unchanged(assuming the stack is an integer stack):

```
public static void RemoveItem(Stack aStack, int item)
```

5. Write a public static general function (non-member function of queue class) to alternately merge Q1's items and Q2's items to form a new merged Queue. For example, if Q1 is 100, 200, 300, 500, 600 and Q2 is -100, -200, -300, after merging, the new merged queue will be 100, -100, 200, -200, 300, -300, 500, 600.

```
public static Queue MergeQueue(Queue Q1, Queue Q2)
```

Stack.java file:

```
public class Stack {
  private int[] elements;
  private int size;
  public static final int DEFAULT_CAPACITY = 16;

  /** Construct a stack with the default capacity 16 */
  public Stack() {
    this(DEFAULT_CAPACITY);
  }

  /** Construct a stack with the specified maximum capacity */
  public Stack(int capacity) {
    elements = new int[capacity];
  }

  /** Push a new integer into the top of the stack */
  public void push(int value) {
    if (size >= elements.length) {
      int[] temp = new int[elements.length * 2];
      System.arraycopy(elements, 0, temp, 0, elements.length);
      elements = temp;
    }

    elements[size++] = value;
  }

  /** Return and remove the top element from the stack */
  public int pop() {
    return elements[--size];
  }

  /** Return the top element from the stack */
  public int peek() {
    return elements[size - 1];
  }

  /** Test whether the stack is empty */
  public boolean empty() {
    return size == 0;
  }

  /** Return the number of elements in the stack */
  public int getSize() {
    return size;
  }
}
```

Queue.java file:

```
public class Queue implements Cloneable{
   private final int DEFAULT_CAPACITY = 10;
   private int front, back, count;
   private int[] m_array;

   //-----------------------------------------------------------------
   //  Creates an empty queue using the default capacity.
   //-----------------------------------------------------------------
   public Queue()
   {
      front = back = count = 0;
      m_array = new int[DEFAULT_CAPACITY];
   }

   //-----------------------------------------------------------------
   //  Creates an empty queue using the specified capacity.
   //-----------------------------------------------------------------
   public Queue (int initialCapacity)
   {
      front = back = count = 0;
      m_array = new int[initialCapacity];
   }

   //-----------------------------------------------------------------
   //  Adds the specified element to the rear of the queue, expanding
   //  the capacity of the queue array if necessary.
   //-----------------------------------------------------------------
   public void enqueue (int element)
   {
      if (size() == m_array.length)
         expandCapacity();

      m_array[back] = element;

      back = (back+1) % m_array.length;

      count++;
   }

   //-----------------------------------------------------------------
   //  Removes the element at the front of the queue and returns a
   //  reference to it. Throws an EmptyCollectionException if the
   //  queue is empty.
   //-----------------------------------------------------------------
   public int dequeue() throws EmptyCollectionException
   {
      if (isEmpty())
         throw new EmptyCollectionException ("queue");

      int result = m_array[front];
      front = (front+1) % m_array.length;
      count--;

      return result;
   }
```

```
 //-----------------------------------------------------------------
 //  Returns a reference to the element at the front of the queue.
 //  The element is not removed from the queue.  Throws an
 //  EmptyCollectionException if the queue is empty.
 //-----------------------------------------------------------------
 public int front() throws EmptyCollectionException
 {
    if (isEmpty())
       throw new EmptyCollectionException ("queue");

    return m_array[front];
 }

 //-----------------------------------------------------------------
 //  Returns true if this queue is empty and false otherwise.
 //-----------------------------------------------------------------
 public boolean isEmpty()
 {
    return (count == 0);
 }

 //-----------------------------------------------------------------
 //  Returns the number of elements currently in this queue.
 //-----------------------------------------------------------------
 public int size()
 {
    return count;
 }


 //-----------------------------------------------------------------
 //  Returns a string representation of this queue.
 //-----------------------------------------------------------------
public String toString()
{
    String result;
    StringBuilder sb = new StringBuilder();

    for(int i = front; i != back; i = (i+1)%m_array.length)
      sb.append(String.valueOf(m_array[i])+"\n");
    result =sb.toString();

    return result;
}

 //-----------------------------------------------------------------
 //  Creates a new array to store the contents of the queue with
 //  twice the capacity of the old one.
 //-----------------------------------------------------------------
public void expandCapacity()
{
  int[] larger = new int[m_array.length *2];

  for(int scan=0; scan < count; scan++)
  {
    larger[scan] = m_array[front];
```

```
      front=(front+1) % m_array.length;
    }

    front = 0;
    back = count;
    m_array = larger;
  }

  @Override
  public Object clone()throws CloneNotSupportedException{
    //return super.clone();
    Queue queueClone = new Queue(m_array.length);
    queueClone.front = front;
    queueClone.back = back;
    queueClone.count = count;
    for(int i = front; i != back; i = (i+1)%m_array.length)
    {
        queueClone.m_array[i] = m_array[i];
    }
    return queueClone;
  }
}
```