[**May use one side of an** 8.5×11 **inch sheet of paper**] Show all of your work clearly in the space provided or on the additional page at the end of the exam. If the additional page is used, clearly identify to which exam question it is related. Be sure to **read each problem carefully**. Note that the exam is double sided.

1. (15 points) Describe the difference between an **interface** and an **abstract class**. Give one example one example of when it is appropriate to use an interface (instead of an abstract class) and one example of when it is appropriate to make use of an abstract class (instead of an interface).

2. (10 points) For arbitrarily long data structures, it makes more sense to implement the queue interface using a LinkedList instead of an ArrayList as the fundamental data structure on which the implementation is built. Explain why this is the case.

3. A palindrome is a phrase that reads the same forward and backward. Here are some examples:

- radar
- Straw? No, too stupid a fad. I put soot on warts.

Notice that punctuation, capitalization, and spacing are ignored when determining whether a phrase is a palindrome or not.

(a) (20 points) Use **both** the ArrayPureStack<E> and the CircularPureQueue<E> classes developed in lecture to implement the following method:

```
// Return true if the phrase is a palindrome, otherwise return false.
public static boolean isPalindrome(char[] phrase)
```

You may assume the CircularPureQueue<E> class has a buffer that is large enough to handle all phrases passed to isPalindrome.

(b) (10 points) Use big-oh notation to describe the overall worst case time complexity for your algorithm. Be sure to explain your reasoning.

4. (15 points) Consider the following method:

```
public static double findKthLargest(double[] nums, int k)
{
  int N = nums.length;
  if(k<0 && N>=k)
  {
    throw new Exception("Get a grip");
  }
  double largest = nums[0];
  int index = 0;
  for(int i=0; i<k; ++i)
  {
    for(int j=1; j<N-i; ++j)
    {
      if(nums[j]>largest)
      {
        largest = nums[j];
        index = j;
      }
    }
    swap(nums[N-i], nums[index]); // swaps to values
  }
  return largest;
}
```

Using big-oh notation, describe the overall worst case time complexity for the function. Be sure to explain your reasoning and state any assumptions that you make. Hint: Use N and k to characterize the input size.

5. (15 points) Consider the following method:

```
public static double findKthLargest(DoublyLinkedList<Double> nums, int k)
{
  int N = nums.size();
  if(k<0 && N>=k)
  {
    throw new Exception("Get a grip");
  }
  double largest = nums.getFirst();
  int index = 0;
  for(int i=0; i<k; ++i)
  {
    for(int j=1; j<N-i; ++j)
    {
      if(nums.get(j)>largest)
      {
        largest = nums.get(j);
        index = j;
      }
    }
    swap(nums, N-i, index);
  }
  return largest;
}
```

Assume that swap swaps the $(N-i)^{th}$ and $index^{th}$ values in nums and runs in $O(N)$ time. Using big-oh notation, describe the overall worst case time complexity for the function. Be sure to explain your reasoning and state any additional assumptions that you make.

6. (15 points) Consider the partially defined DoublyLinkedList<E> class with the specified inner Node<E> class.

```
public class DoublyLinkedList<E> implements List<E>
{
  protected Node<E> front = null;

  protected class Node<E>
  {
    protected E element;
    protected Node<E> previous;
    protected Node<E> next;

    Node(E obj){
      element = obj;
      next = null;
      previous = null;
    }
  }

  // ...

  /**
   * Returns the index in this list of the first occurrence of the specified
   * element, or -1 if this list does not contain this element.
   */
  public int indexOf(Object o)
  {
    // [show your implementation of this method below and/or on the next page.]
  }

  // ...
}
```

Implement the indexOf method for this class.

6. cont. . .