CS 101 Fall 2005 Final Exam Name: ______________________________ Email ID: ________

This exam is open text book but closed-notes, closed-calculator, closed-neighbor, etc. Unlike midterm exams, you have a full 3 hours to work on this exam. **This final exam is optional; you may choose not to have this exam be graded.** If you do turn in the exam, we will grade it and include this grade in the calculation for your final letter grade in the class. If you decide not to turn the exam in, please tear it up and turn in the pieces. Please sign the honor pledge here:

____

____

____

____

Page 1	____ / 5
Page 2	none
Page 3	____ / 15
Page 4	____ / 20
Page 5	____ / 15
Page 6	____ / 20
Page 7	____ / 25
Page 8	none
Total	**____ / 100**

Note: When an integer type is required use ***`int`****, when a floating-point type is required use* ***`double`****.*

1. (1 point) What section are you in?

____ CS 101-E ____ CS 101-2 (lab 7-8:30 PM Thu)

____ CS 101-3 (lab 12-1:30 PM Fri) ____ CS 101-4 (lab 2-3:30 PM Fri)

2. (4 points) What is your overall impression of the class so far? Check only one in *each* column.

___ Too slow ___ Too easy

___ Too fast ___ Too hard

___ Just right ___ Just right

___ / 5

The following questions ask you to code up a class to implement a *queue*. A queue, like the *stack* class we have discussed, is similar to an array or a vector in that it stores a list of elements. Unlike a stack, where new elements are always added to and removed from the same end of the stack, the elements in a queue are added to one end of the queue and removed from the other end. A queue is sometimes called a *FIFO* for “First In, First Out”, since the first element in a series placed in a queue will be the first element removed from it (as opposed to a stack, or LIFO, where the last element of a series placed in a queue will be the first element removed). Think of a line at a grocery store – people enter the line (queue) at one end, and leave the line (queue) at the other (at the cash register). We will say that elements are added to the *head* of the queue (called *enqueing* the element) and removed from the *tail* (called *dequeueing*).

In the class that you implement, you will store the elements of the queue—integers in this case—in an array (***not*** a Vector). For full credit, your implementation should use only a single array and should not create or copy additional arrays e.g. in the enqueue() or dequeue() methods. You may assume that the queue will never be required to store more than 1000 elements at once and size your array accordingly. Note, however, that the queue should be able to enqueue over 1000 items, as long as enough items are dequeued along the way that the queue never contains more than 1000 items all at once. Hint: you will probably want instance variables called “head” and “tail” to indicate where in the array elements will be added and removed.

The elements of the queue will form a range in the array between the head and the tail positions. Note that this range may not be contiguous: if many elements are enqueued and dequeued, the head may need to “wrap around” to the beginning of the array (so that the elements of the queue go from array[tail] to array[array.length-1] and from array[0] to array[head]). For example, if you enqueue 1000 elements (so your queue is full), then dequeue 500, your queue will hold the remaining 500 elements in the last 500 positions of the array (positions 500 to 999). You can then enqueue 100 more elements. Thus, your queue will hold 600 elements: the first 500 in positions 500 to 999, the last 100 in positions 0 to 99. For full credit your code must correctly account for this “wrap-around” behavior; however, you will still receive substantial partial credit if your Queue class works correctly but does not support “wrap around”.

You will provide code for the following:

- Any necessary instance variables.
- **`Queue()`**: The default constructor.
- **`void enqueue(int item)`**: Add the specified integer to the head of the queue.
- **`int dequeue()`**: Remove the integer from the tail of the queue and return it.
- **`int peek()`**: Return the integer at the tail of the queue without removing it.
- **`int size()`**: Return the number of elements currently stored in the queue.
- **`String toString()`**: Return a string suitable for printing the elements currently stored in the queue.
- **`boolean containsElement(int item)`**: Search for the specified integer among the elements of the queue.

To emphasize that you are writing a single functional class, we have written this up in the form of skeleton code for you to complete. The different code sections equate to exam questions, and are explained further in the comments along with their point values. Your complete answer to these questions should be a functioning Java class. Note that some of the method prototypes (i.e., return type & list of parameters) are incomplete and need to be filled out! Don’t forget to count your braces, parentheses, etc.

We also include the full code for a **`main()`** method to illustrate the use of the Queue class.

```
public class Queue {

    //          Question 3: 10 points
    // instance variables: give the code necessary to declare the instance variables
    // needed by this class, using the information given in the introduction. The
    // instance variables may be initialized here or in the constructor below.
```

```
    //          Question 4: 5 points
    // Provide the code for the definition of the default constructor. You may
    // initialize your instance variables here or when they are defined above.
```

___ / 15

```
//          Question 5: 10 points
// enqueue(): Add the specified integer to the head of the queue
//
// You may assume that enqueue() will never be called if it would cause
// the queue to contain more than 1000 elements

public void enqueue (int value) {
```

```
//          Question 6: 10 points
// dequeue(): Remove the integer from the tail of the queue and return it
//
// You may assume that dequeue() will never be called if the queue is empty

public int dequeue() {
```

___ / 20

```
//          Question 7: 5 points
// peek(): Return the integer at the tail of the queue without removing it
//
// You may assume that peek() will never be called if the queue is empty.
// Note that for the remaining questions you must also complete the prototype!

      peek () {
```

```
//          Question 8: 10 points
// size(): Return the number of elements currently stored in the queue

      size () {
```

___ / 15

```
//          Question 9: 20 points
// containsElement(): Return true if the queue contains the specified integer
//
// If the integer isn’t present or if the queue is empty you should return false

      containsElement (int element) {
```

___ / 20

```
//          Question 10: 25 points
// toString(): Return a String with the elements currently stored in the queue.
//
// The String should be of the form, "Queue[a, b, c]" where a,b,c are elements
// in the queue. The first element (a in this example) should be at the tail
// of the queue and the final element (c in this case) should be at the head.
// Of course there may be more or less than the three elements in this example.
// Notice that elements are followed by a comma and a space, except the last.

      toString() {
```

___ / 25

```
    // The below main() method is simply to give you an example of how the
    // Queue class might be used and how it should behave.  You do not need
    // to modify, add, or otherwise do anything with the below code.

    public static void main (String args[]) {

          // Create a new Queue
          Queue queue = new Queue();

          // Enqueue integers 5 through 10

          for ( int i = 5; i <= 10; i++ )
              queue.enqueue(i);

          // Print the queue:
          // should print "Queue[5, 6, 7, 8, 9, 10]"

          System.out.println (queue);

          // Print the value at the tail of the queue:
          //  should print "peek() returned: 5"

          System.out.println ("peek() returned: " + queue.peek());

          // Try a search: should print "containsElement(6) returned: true"

          System.out.println ("containsElement(6) returned: "
                              + queue.containsElement(6));

          // Test dequeue(): should print "dequeue() returned: 5"

          int ret = queue.dequeue();
          System.out.println ("dequeue() returned: " + ret);

          // Print the queue again: should print "Queue[6, 7, 8, 9, 10]"
          System.out.println (queue);

          // Print the value at the tail of the queue:
          //  should print "peek() returned: 6"
          System.out.println ("peek() returned: " + queue.peek());

          // dequeue 2 more values, enqueue the integer -1 in between

          queue.dequeue();
          queue.enqueue(-1);
          queue.dequeue();

          // Print the queue: should print "Queue[8, 9, 10, -1]"

          System.out.println (queue);

          // Try a search: should print "containsElement(6) returned: false"

          System.out.println ("containsElement(6) returned: "
                              + queue.containsElement(6));
    }
}
```