The University of Western Ontario
Department of Computer Science

Computer Science 1027b Midterm Exam
With Solutions

Print your name:_____

Student number: _____

Instructions:

- **Fill in your name and student number above immediately.**
- Answer all the questions in the exam in the spaces provided.
- You have 2 hours to complete the exam.
- The exam has 8 questions on 14 pages, and is out of a possible 80 marks.
- The marks for each individual question are given. Allow approximately 1.5 minute per mark on average.
- There is a page for rough work at the end of the exam paper.
- Read the questions carefully!

DO NOT TURN THIS PAGE UNTIL DIRECTED TO DO SO.

1 (max 15)	
2 (max 5)	
3 (max 10)	
4 (max 15)	
5 (max 10)	
6 (max 10)	
7 (max 10)	
8 (max 5)	
TOTAL (max 80)	

1. (15 marks) True/False (Circle your answers) *(Answers are italicized)*

a) (1 mark) A static method within a class can be invoked without having to instantiate an object of that class.

True **False**

b) (1 mark) A Java **interface** should never contain a **constructor**.

True **False**

c) (1 mark) A Java **interface** never contains an **attribute (instance variable)** declaration.

True **False**

d) (1 mark) A new class **X** derived from a class **Y** establishes an *is-a relationship* from class **X** to class **Y**.

True **False**

e) (1 mark) All Java classesare derived, directly or indirectly from the **Object** class

True **False**

f) (1 mark) Any method in a Java class **X** can call public and private methods from **X**.

True **False**

g) (1 mark) In the Java statement

```
LinkedBag<BingoBall> b = new LinkedBag<BingoBall>( );
```

BingoBall is a generic type.

True ***False***

h) (1 mark) The reserved word **super** can be used in a class to refer to its parent class.

True **False**

i) (1 mark) The objects in a Java program are created when the program is being **executed**, as opposed to when it is compiled.

True **False**

j) (1 mark) The terms *parent class, superclass,* and *base class* all mean the same thing.

True **False**

k) (1 mark) The code in ArrayBag.java is an implementation of the BagADT interface.

True **False**

l) (1 mark) A child class cannot override a parent method that is declared as **final**.

True **False**

m) (1 mark) Executable code is generally found in **interfaces**.

True ***False***

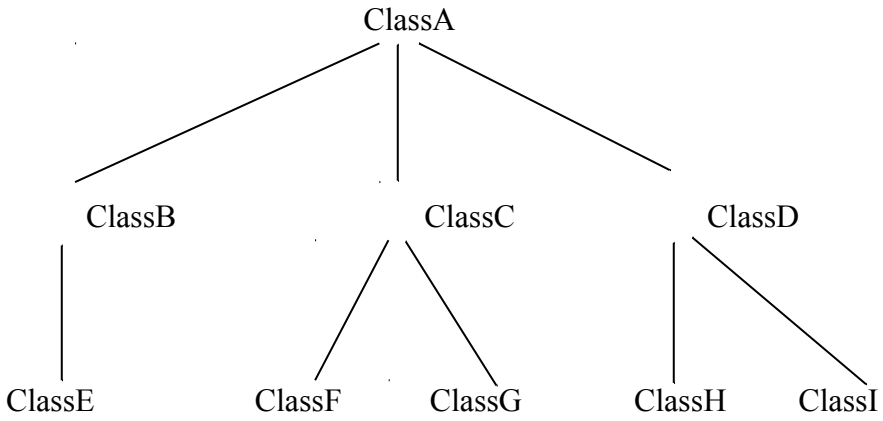
n) (1 mark) The file **BagADT.java** contains both an **interface** and an implementation of it.

True ***False***

o) (1 mark) The methods **enqueue()**, **dequeue()**, **first(), size()**, and **isEmpty()** from the class **LinkedQueue** have time complexity **O(1)**.

True **False**

2. (5 marks) The following diagram represents a hierarchy of Java classes. Use the diagram to determine whether the statements given below it are **potentially** True or False.



a) (1 mark) ClassG implements ClassC.

True ***False***

b) (1 mark) ClassF extends ClassA.

True ***False (accepted True, as could interpret extends to include"indirectly")***

c) (1 mark) Suppose that ClassD is the only class in the diagram that contains a public method **blah( )**. If a Java program attempts to invoke **blah( )** against an object from ClassH, the method in ClassD will be executed.

True **False**

d) (1 mark) Again, suppose that ClassD is the only class in the diagram that contains a public method **blah( )**. If a Java program attempts to invoke **blah( )** against an object from ClassA, the method in ClassD will be executed.

True ***False***

e) (1 mark) Suppose that we have the declaration **ClassA a;** The variable **a** can hold a reference to an object from ClassE.

True **False**

3. (10 marks) Give the number of objects that are created at *runtime* by each of the following Java code segments:

a) (2 marks) **LinkedStack<Integer> numbers;**

Answer: _____0 _____

b) (2 marks) **ArrayQueue<String> a = new ArrayQueue<String>(4);**

Answer: _____2 _____ (*there is also an array object created in the ArrayQueue constructor)*

c) (2 marks)

```
int a = 0;
for (int k = 0; k < 5; k++)
    a++;
```

Answer: _____0 _____

d) (2 marks)

```
contents = (T[]) (new Object[initialCapacity]) ;
```

Answer: _____1 _____

e) (2 marks)

```
LinkedBag<Integer> b = new LinkedBag<Integer>( );
b.add( new Integer(2));
```

Answer: _____2 _____

4. (15 marks) In each of the following situations, use **Big-O notation** to express the amount of work being done in terms of **n**.

a) (1 mark) Java code is used to print out the value of the first element in an array of **n** elements.

Answer: _____O(1) _____

b) (2 marks) Java code is used to print out the value of the bottom element in a stack containing **n** elements (assume it is a linked stack, as per the class notes).

Answer: _____O(n) _____

c) (2 marks) A Java program takes $2^n + 2n^2$ steps to complete, given an input of size n. What is its time-complexity, in big-O notation?

Answer: _____O(2^n)_____

d) (2 marks) The following Java method is executed:

```
public static void print1( int [ ] list, int n ) {
    for ( int i = 0; i < n; i++)
    System.out.println( list[i] );
}
```

Answer: _____O(n)_____

e) (2 marks) The following Java code segment is executed:

```
int k = 0;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
    k = i*j ;
    System.out.println(k);
    }
}
```

Answer: _____O(n^2)_____

f) (2 marks) The following Java code segment is executed:

```
int i = n ;
while (i >0) {
    System.out.println(i);
    i = i/2 ;
}
```

Answer: _____O(log (base 2) **n)_____**

g) (2 marks) The following Java method is executed:

```
public static void havingFun( int n ) {
    for (int i = 0; i < n; i++)
            for (int j = 0; j < n, j++)
                    for (int k = 0; k < n; k++)
                            System.out.println(i,j,k);
}
```

Answer: **_____O(n^3)_____**

h) (2 marks) The following Java method is executed:

```
public static int someSum ( int[ ][  ] table, int n ) {
    int sum = 0;
    for (int k = 0; k < n; k++)
            sum = sum + table[n-k-1][n-k-1];
    return sum;
}
```

Answer: _____O(n) _____

5. (10 marks)

The lottery called Lotto 6/49 consists of drawing 6 balls at random. The balls exist in a bag. Each ball has a unique label which is an integer between 1 and 49, inclusively.

Using the **BagADT** , write the body of the main method for the class Lotto649 in the space provided below, such that:

- it puts the appropriate balls for the lottery into a bag collection
- it randomly removes 6 balls from the bag and displays the number of each ball onto the computer screen

The **BagADT** interface is provided as a separate attachment to this exam. You may use either the **ArrayBag** or the **LinkedBag** implementation of the **BagADT**.

Assume the following class **LotteryBall** is defined. It represents a Lotto 6/49 ball:

```
public class LotteryBall {
        private int ballValue;

        public LotteryBall(int num) {
                ballValue = num;
        }

        public String toString() {
                String result = "Ball number: " ;
                result = result + ballValue;
                return result;
        }
}
```

[Put your answer on the next page]

[This page was intentionally left blank]

```
public class Lotto649 {

public static void main (String[] args) {

    // the body of the main method was required

  final int NUM_BALLS = 49;
   final int WINNERS = 6;

   ArrayBag<LotteryBall> bag =
                    new ArrayBag<LotteryBall>(NUM_BALLS);


   // or
   // ArrayBag<LotteryBall> bag =
                       new ArrayBag<LotteryBall>();
   // or
   // LinkedBag<LotteryBall> bag =
                       new LinkedBag<LotteryBall>();



   for (int k = 1; k <= NUM_BALLS; k++)
      bag.add( new LotteryBall(k) );

   for (int k = 1; k <= WINNERS; k++)
      System.out.println( bag.removeRandom().toString());

   }
```

6. (10 marks) The following main program uses both **LinkedStack.java** and **LinkedQueue.java** to create a stack of queues, in which **Integer** objects are enqueued:

```
public class StackOfQueues {
    public static void main(String args[]) throws Exception
    {
            final int MAX = 3 ;
            LinkedQueue<Integer> queue ;
            LinkedStack<LinkedQueue<Integer>> stack ;

            stack = new LinkedStack<LinkedQueue<Integer>>();

            for (int j = 0; j < MAX; j++) {
                    queue = new LinkedQueue<Integer>();
                    for (int k = j; k < MAX; k++) {
                            queue.enqueue(new Integer(j+k+1)) ;
                    }
                    stack.push(queue);
            }

            System.out.println("Size of stack: \n" + stack.size() + "\n");
            System.out.println("S Contents: \n" + stack.toString() + "\n");

            while (!stack.isEmpty()){
                    queue = stack.pop();
                    System.out.println("Q contents: " + queue.toString());
            }
    }
}
```

Show the output produced by running this main method.

Size of stack:
3
S Contents: *(shown with contents of each queue on the same line to conserve space)*
5
3 4
1 2 3

Q Contents: 5 *(shown with contents of each queue on the same line to conserve space)*
Q Contents: 3 4
Q Contents: 1 2 3

7. (10 marks) The following program fills a stack of queues and then uses a method called **attach()** to pop two queues from the stack and join them into a single queue, which gets pushed back on the stack. The program repeats these steps until the stack is empty and all the queues are attached into a single queue:

```
public class StackOfQueues {

    public static void main(String args[ ]) throws Exception
    {
            LinkedQueue<Integer> queue ;
            LinkedStack<LinkedQueue<Integer>> stack;

            stack = new LinkedStack<LinkedQueue<Integer>>( );

            // assume here is code to fill the stack with queues…

        while (!stack.isEmpty( )){
           queue = stack.pop( );
           if (!stack.isEmpty( )) {
             queue = queue.attach(stack.pop( )) ;
             stack.push(queue) ;
           }
        }
    }
}
```

Write the **attach()** method in such a way that it attaches the rear of the first popped queue (**this** queue) to the head of the queue passed as a parameter, to form a single queue, which is then returned by the method.

attach() is to be added to the class **LinkedQueue<T>**. Recall that the class has the following instance variables, which can be used in your solution:

```
int count;              /* the number of values currently in the queue */
LinearNode<T> front;  /* a reference to the first node in the queue */
LinearNode<T> rear;   /* a reference to the last node in the queue */
```

[Put your answer on the next page]

```
public LinkedQueue<T> attach(LinkedQueue<T> q) {
```

Note: it was not in the specifications, but it is good programming practice to NOT destroy either "this"queue or the parameter queue when the method returns a new queue. (If you do change them, that is called a "side effect" and is considered to be undesirable.) The solution shown below does not change the original queues referenced by "this" and by q.

```
    LinkedQueue<T>  joinedQ = new LinkedQueue<T> ();

    for (int i = 0; i < this.count; i ++) {         // could use size() instead of count
        {
            T element = this.dequeue();
            joinedQ.enqueue(element);
            this.enqueue(element);              // put it back in this queue
         }

    for (int i = 0; i < q.count; i ++) {
        {
            T element = q.dequeue();
            joinedQ.enqueue(element);
            q.enqueue(element);                 // put it back in the q queue
         }

   return joinedQ;

}
```

(5 marks) Consider the circular array implementation of the Queue ADT in our lecture notes. Suppose that the array is capable of holding four values. Suppose that, into an initially empty queue, we enqueue three Integer objects containing the values 3, 5, and 7, in this order. Then, two objects are dequeued, and three more Integer objects containing the values 8, 6, and 9 are enqueued. Which of the following arrays may be used to visualize how the contents are now stored in the underlying data structure? Circle the correct array.

(The correct answer is the 3rd array, the one with the numbers italicized.)

Index 0	1	2	3
7	8	6	9

6	7	8	9

6	***9***	***7***	***8***

9	7	8	6

[This page was intentionally left blank for rough work]