University of Western Ontario
Department of Computer Science
Computer Science 1027b Midterm Exam
March 7th, 2015, NS-1, 10am-Noon, 2 hours
Sections I (Barron) and II (Locke) [Please circle one]

PRINT YOUR NAME:

PRINT YOUR STUDENT NUMBER:

DO NOT TURN THIS PAGE UNTIL INSTRUCTED TO DO SO!

Instructions

- Fill in your name and student number above immediately.
- You have **2 hours** to complete the exam.
- Part 1 of the exam consists of Multiple Choice questions. Circle your answers on this exam paper.
- Part 2 consists of questions for which you will provide written answers. Write your answers in the spaces provided in this exam paper.
- Multiple choices question are worth 1 mark, unless indicated otherwise; other than that, the marks for each individual question are given. Allow approximately 1 minute per mark on average.
- There are pages for rough work at the back of the exam. You may detach them if you wish, but hand them in with the rest of the exam paper.
- Calculators, Telephones and laptops are not allowed!

Mark summary

1	2	3	4	5	6	total
/20	/20	/15	/15	/15	/15	/100

Problem 1(20 marks)

1. ADT is an example of a data structure — true [✓ false]
2. `LinearNode`(s) can be used to implement a Java array — true [✓ false]
3. In the `LinkedStack` implementation, the `LinearNode` at the `top` of the stack always has its next node set to `null` if it exists — true [✓ false]
4. In the `LinkedQueue` implementation, the `LinearNode` at the `tail` of the queue always has its next node set to `null` if it exists — [✓ true] false
5. The Object class can have a parent class — true [✓ false]
6. If class A inherits from class B, A can access B's private attributes — true [✓ false]
7. If class A inherits from class B, B can access A's private attributes — true [✓ false]
8. The `Exception` class is a superclass of all exceptions in Java — [✓ true] false
9. When using a circular array to implement a queue, the front and rear indices are changed when dequeueing and enqueueing respectively — [✓ true] false
10. Java objects always have an implementation of the `toString()` method — [✓ true] false
11. If class A is declared as: `public class A implements C`, A must implement all methods specified by C in order to compile — [✓ true] false
12. All collections in Java are linear collections — true [✓ false]
13. Abstraction aims to integrate an interface and its implementation seamlessly — true [✓ false]
14. The Queue abstract data type is a first in, first out collection — [✓ true] false
15. The circular-array implementation for a queue outperforms the linked-linear-node implementation for all queue operations — true [✓ false]
16. Stacks are useful when implementing language parsers where you must match tags or parentheses (like HTML or Java respectively) — [✓ true] false
17. The terms overloading and overriding have very different meanings in Java — [✓ true] false
18. In a singly-linked list, inserting at the head has the same complexity as inserting at the end — true [✓ false]
19. A reference variable declared as the superclass type can reference an object of the subclass type — [✓ true] false
20. lasses A and B are the subclasses of superclass C. Because of dynamic polymorphism, reference variables of type A can also reference objects of type B — true [✓ false]

Problem 2 (20 marks)

```java
import java.util.*;

public class Midterm2015 {

    private LinkedStack<Integer> stack;

    public Midterm2015(){
        stack = new LinkedStack<Integer>();
        stack.push(new Integer(0));
    }

    public void add(String nums){
        StringTokenizer toAdd = new StringTokenizer(nums);
        while(toAdd.hasMoreTokens()){
            Integer num = Integer.parseInt(toAdd.nextToken());
            num = num + stack.peek();
            stack.push(num);
        }
    }

    public void whatDoesThisDo(){
        LinkedStack<Integer> temp = new LinkedStack<Integer>();
        while(!stack.isEmpty()){
            temp.push(stack.pop());
        }
        stack = temp;
    }

    public String toString(){
        return "this thing contains:\n" + stack.toString();
    }

    public static void main(String[] args) {
        Midterm2015 mid = new Midterm2015();
        mid.add("3 4 5 6 7");
        System.out.print("Before running, ");
        System.out.println(mid);
        mid.whatDoesThisDo();
        System.out.print("After running, ");
        System.out.println(mid);
    }
}
```

Please answer the following questions about the code above:

1. (2 %) Which methods from the StackADT are used in the above code?

   push pop peek toString isEmpty

2. (2 %) What does the method `whatDoesThisDo` do?

   Reverses the order of the elements on the stack this.stack

3. (2 %) What, if anything, is on the stack after line 34 executes?

   The stack has a single element: 0

4. (2 %) What would happen if `stack` was empty when the `add` method was invoked?

   Since the stack is empty, the peek method would cause an exception.

5. (12 %) Trace the program and write what will be printed to the screen by running `java Midterm2015` here:

```
Before running, this thing contains:
25 18 12 7 3 0
After running, this thing contains:
0 3 7 12 18 25
```

Problem 3 (15 marks)

```
1. import java.io.*;
2. public class MidtermExceptions {
3.     public static void main(String[] args) throws Exception {
4.         BufferedReader keyboard = null;
5.         try{
6.             keyboard = new BufferedReader (new InputStreamReader(System.in),1);
7.             System.out.println("Please enter an integer: ");
8.             String userTyped = keyboard.readLine();
9.             int value = Integer.parseInt(userTyped);
10.            for (int i=0; i<=value; i++) {
11.                value = value + 1/(i+value);
12.            }
13.            System.out.println("The result is: " + value);
14.        } catch(NumberFormatException e){
15.            System.out.println("That is not an integer!");
16.        } catch(ArithmeticException e){
17.            System.out.println("The value entered resulted in division by zero");
18.        } catch(Exception e){
19.            System.out.println("An error occured while processing input");
20.        }
21.        finally{
22.            try{
23.                if(keyboard !=null)
24.                    keyboard.close();
25.            } catch(IOException e){
26.                System.out.println("Unable to close file");
27.            }
28.        }
29.        System.out.println("BYE!");
30.    }
31. }
```

1. (1 %) Will Line 13 always be executed when running the above program? (Yes or No)

   No

2. (1 %) Will Line 23 always be executed when running the above program? (Yes or No)

   Yes

3. (1 %) Will Line 29 always be executed when running the above program? (Yes or No)

   Yes

4. (2 %) Will the MidtermExceptions.java file still compile if the `throws exception` portion of the `main` method signature in the above program is deleted?
   (Yes or No , plus brief explanation)

   Yes, all exceptions possible are caught.

5. (5 %) What will be printed to the screen if the user enters: `0`

```
The value entered resulted in division by zero
BYE!
```

6. (5 %) What will be printed to the screen if the user enters: `2`

```
The result is: 2
BYE!
```

Problem 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.

1. (2%) An element is inserted in an `ArrayStack` of size n, which has reached full capacity.

   Answer: $O(n)$

2. (2%) An element is removed from a `LinkedQueue` of size n

   Answer: $O(1)$

3. (2%) We test whether a `LinkedQueue` of size n is empty using `size`

   Answer: $O(1)$

4. (2%) An element is removed from a `ArrayStack` of size n

   Answer: $O(1)$

5. (2%) We execute the following code segment

   ```
   for (int i = 1; i < n/2; i++)
     for (int j = i; j <= n/2; j++)
       System.out.println(i+j);
   ```

   Answer: $O(n^2)$

6. (3%) We execute the following code segment

   ```
   for (int i = 1; i < Math.log(n); i++)
     for (int j = 1; j < Math.pow(2,n); j++)
       System.out.println(i);
   ```

   `Math.pow(2,n)` computes 2^n.

   Answer: $O(log(n)2^n)$

7. (2%) We execute the following code segment

   ```
   for (int i = 1; i < n*n*n*Math.log(n); i++)
     System.out.println(i);
   ```

   Answer: $O(n^3 log(n))$

Problem 5 (15 marks)

Consider a stack of queues of integers in the following Java code:

```
public class midterm2015_questions_5 {

public static void main(String[] args) {
ArrayStack<ArrayQueue<Integer>> stack=new ArrayStack<ArrayQueue<Integer>>();
ArrayQueue<Integer> queue1=new ArrayQueue<Integer>();
ArrayQueue<Integer> queue2=new ArrayQueue<Integer>();
ArrayQueue<Integer> queue3=new ArrayQueue<Integer>();

// Insert some data
queue1.enqueue(3);
queue1.enqueue(2);
queue1.enqueue(4);
stack.push(queue1);
queue2.enqueue(1);
queue2.enqueue(6);
stack.push(queue2);
queue3.enqueue(5);
queue3.enqueue(9);
queue3.enqueue(7);
queue3.enqueue(8);
stack.push(queue3);

System.out.println("\nContents of stack:");
System.out.println(stack.toString());

System.out.println("Minimum value of all integers in all queues on the stack: " +
                    minValue(stack));
}
}
```

1. (5%) What is printed by the `toString()` method.

```
Contents of stack:
3          5
2          9
4          7
           8
1
6   or     1
           6
5
9          3
7          2
8          4
The left column was what the Java program produced (which prints the
stacks bottom to top). The right column is assuming the stacks are printed
from top to bottom). The toString method comments in the stack and queue
interfaces at the end of the exam now say the way stack traveral is done.
```

2. (10%) Write the `minValue` method below. Take care not to destroy the input `stack` in the method. You can assume there are no empty stacks or queues initially,

```
public static int minValue(ArrayStack<ArrayQueue<Integer>> stack) {
int val,min=0,queueSize;
ArrayQueue<Integer> queue;
ArrayStack<ArrayQueue<Integer>> tempStack=
        new ArrayStack<ArrayQueue<Integer>>();

// Assume initially that the first values is the minimum
if(!stack.isEmpty()) min=stack.peek().first();
else
{
System.out.println("Fatal error: stack is empty");
System.exit(1);
}

while(!stack.isEmpty()) {
  queue=stack.pop();
  // This code keeps each queue intact
  queueSize=queue.size();
  for(int i=0;i<queueSize;i++) {
       val=queue.dequeue();
       if(val < min) min=val;
       queue.enqueue(val);
       }
  // At the end tempStack contains stack in reverse order
  tempStack.push(queue);
  }
// Now copy tempStack back into into stack in the right order
while(!tempStack.isEmpty())
   stack.push(tempStack.pop());
return(min);
}
```

Problem 6 (15 marks)

Consider a queue of stacks of integers in the following Java code:

```
public class midterm2015_questions_6 {

public static void main(String[] args) {
ArrayQueue<ArrayStack<Integer>> queue= new ArrayQueue<ArrayStack<Integer>>();

ArrayStack<Integer> stack1=new ArrayStack<Integer>();
ArrayStack<Integer> stack2=new ArrayStack<Integer>();
ArrayStack<Integer> stack3=new ArrayStack<Integer>();

stack1.push(3);
stack1.push(2);
stack1.push(4);
queue.enqueue(stack1);
stack2.push(1);
stack2.push(6);
queue.enqueue(stack2);
stack3.push(5);
stack3.push(9);
stack3.push(7);
stack3.push(8);
queue.enqueue(stack3);

System.out.println("Contents of queue:");
System.out.println(queue.toString());

System.out.println("Maximum size of any stack on the queue: " +
                    maxLength(queue));
}
}
```

1. (5%) What is printed by the `toString()` method.

```
Contents of queue:
3         4
2         2
4         3

1         6
6  or     1

5         8
9         7
7         9
8         5
The left column was what the Java program produced (which prints the stacks
bottom to top). The right column is assuming the stacks are printed from top
to bottom). The toString method comments in the stack and queue interfaces
at the end of the exam now say the way stack traveral is done.
```

2. (10%) Write the `maxLength` method below. Take care not to destroy the input `queue` in the method. You can assume there are no empty stacks or queues initially,

```
public static int maxLength(ArrayQueue<ArrayStack<Integer>> queue) {
int max=0,queueSize;
ArrayStack<Integer> stack;

queueSize=queue.size();
if(queueSize==0)
{
System.out.println("Fatal error: queue is empty");
System.exit(1);
}

for(int i=0;i<queueSize;i++) {
  stack=queue.dequeue();
  if(stack.size()>max) max=stack.size();
  queue.enqueue(stack);
  }
return(max);
}
```

Stacks and Queues Interfaces

```
public interface StackADT<T>{
  /**  Adds one element to the top of this stack.
   *   @param element element to be pushed onto stack  */
  public void push (T element);

  /**  Removes and returns the top element from this stack.
   *   @return T element removed from the top of the stack */
  public T pop();

  /**  Returns without removing the top element of this stack.
   *   @return T element on top of the stack */
  public T peek();

  /**  Returns true if this stack contains no elements.
   *   @return boolean whether or not this stack is empty */
  public boolean isEmpty();

  /**  Returns the number of elements in this stack.
   *   @return int number of elements in this stack */
  public int size();

  /**  Returns a string representation of this stack.
   *   @return String representation of this stack
   *   Stack elements are printed from the bottom to
   *   the top of the stack and the stack is undestroyed
   */
  public String toString();
}
```

```
public interface QueueADT<T>{
   /**
    * Adds one element to the rear of this queue.
    * @param element  the element to be added to the rear of this queue  */
   public void enqueue (T element);

   /**
    * Removes and returns the element at the front of this queue.
    * @return  the element at the front of this queue */
   public T dequeue();

   /**
    * Returns without removing the element at the front of this queue.
    * @return  the first element in this queue */
   public T first();

   /**
    * Returns true if this queue contains no elements.
    * @return  true if this queue is empty */
   public boolean isEmpty();

   /**
    * Returns the number of elements in this queue.
    * @return  the integer representation of the size of this queue */
   public int size();

   /**
    * Returns a string representation of this queue
    * @return  the string representation of this queue
    * Queue elements are printed from first to last
    * The queue is not destroyed
    */
Public String toString();
```

Rough work 1/4

Rough work 2/4

Rough work 3/4

Rough work 4/4