University of Western Ontario
Department of Computer Science
Computer Science 1027b Midterm Exam
March 5th, 2016, NS-1, 10am-Noon, 2 hours

Please circle one

Sections I (John Barron) and II (James Hughes)

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 explicit example of inheritance true [✓ false]
2. `LinearNode`(s) can **not** be used to implement a Queue true [✓ false]
3. A Queue is an example of a LIFO structure true [✓ false]
4. A Queue would be a good choice when evaluating a postfix expression true [✓ false]
5. The Object class is a parent class of the `toString()` method true [✓ false]
6. If class A inherits from class B, B can access A's private attributes true [✓ false]
7. Exceptions cannot use inheritance true [✓ false]
8. The `extends` keyword is the same as the `implements` keyword, but for exceptions true [✓ false]
9. Linked lists have fixed sizes true [✓ false]
10. Inserting an element at the end of a linked list is always $\mathcal{O}(n)$ true [✓ false]
11. Inserting an element into the middle of a linked list is always $\mathcal{O}(n^2)$ true [✓ false]
12. The only places you can insert into a linked list is in the middle and the end true [✓ false]
13. A stack must be implemented with an array true [✓ false]
14. *Doubly linked list* is another word for a *binary tree* true [✓ false]
15. `thing1.equals(thing2)` basically means the same thing as `thing1 == thing2` true [✓ false]
16. The terms overloading and overriding have the same meanings in Java true [✓ false]
17. At the very least, the `toString()` method is inherited from the Object class [✓ true] false
18. With asymptotic complexity, $t(n) = 5n^2 + 3n$ is of the order $\mathcal{O}(n^2)$ [✓ true] false
19. We *typically* want a front and rear reference/index for queues [✓ true] false
20. For a queue with a good linked list implementation, both enqueueing and dequeueing have an $\mathcal{O}(1)$ complexity [✓ true] false

Problem 2 (20 marks)

```java
public class Midterm2016 {

    private LinkedQueue<Integer> queue;
    private LinkedStack<Integer> stack;

    public Midterm2016(){
        queue = new LinkedQueue<Integer>();
    }

    public void add(int n){
        for(int i = 0 ; i < n ; i++){
            queue.enqueue(i);
        }
    }

    public void whatDoIDo(int n){
        stack = new LinkedStack<Integer>();
        while(!queue.isEmpty()){
            stack.push(queue.dequeue());
        }
        while(!stack.isEmpty()){
            queue.enqueue(stack.pop() * n);
        }
    }

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

    public static void main(String[] args){
        Midterm2016 mid = new Midterm2016();
        mid.add(5);
        System.out.println("Before...");
        System.out.println(mid);
        mid.whatDoIDo(10);
        System.out.println("After...");
        System.out.println(mid);
    }
}
```

Please answer the following questions about the code above:

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

   stack: push, pop, isEmpty
   queue: enqueue, dequeue, isEmpty, toString

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

   Reverses the order of the elements in the queue by using a stack and also multiplies the element by n.

3. (2 %) What, if anything, is on the `stack` immediately after line 37 executes?

   The stack is empty

4. (2 %) What is the *type* of the elements in these data structures?

   Integer *Objects*

5. (12 %) Trace the program and write what will be printed to the screen by running `java Midterm2016` here:

```
Before...
this stuff contains:
0 1 2 3 4
After...
this stuff contains:
40 30 20 10 0
```

Problem 3 (15 marks)

```
1 public class Something<E>{
2     private Something<E> anotherThing;
3     private E mine;
4
5     public Something(){
6         anotherThing = null;
7         mine = null;
8     }
9
10    public static void main(String[] args){
11        Something<Object> myStuff = new Something<Object>();
12        Something<Object> iter = myStuff;
13        try{
14            for(int i = 0 ; i < Integer.parseInt(args[0]) ; i++){
15                iter.mine = i;
16                iter.anotherThing = new Something<Object>();
17                iter = iter.anotherThing;
18            }
19        }
20        catch (ArrayIndexOutOfBoundsException e){
21            System.out.println("no args given");
22        }
23        catch (NumberFormatException e){
24            System.out.println("you didn’t give me a number");
25        }
26        catch (Exception e){
27            System.out.println("Something bad happened");
28        }
29        iter = myStuff;
30        while (iter != null){
31            System.out.println(iter.mine);
32            iter = iter.anotherThing;
33        }
34    }
35}
```

1. (1 %) What line of code could throw an exception?

   14

2. (1 %) Will Line 21 always be executed when running the above program? (Yes or No)

   No

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

   Yes

4. (2 %) What type of structure would this code be making if executed properly?

A linked structure (A forward linked structure).

5. (5 %) What would be printed to the screen if `5` is given as an argument?

```
0
1
2
3
4
null
```

6. (5 %) What would be printed to the screen if `five` is given as an argument?

```
you didn’t give me a number
null
```

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 removed from an `ArrayStack` of size n, which has reached full capacity.

   Answer: $O(1)$

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

   Answer: $O(1)$

3. (2%) We execute a method, `size`, to determine the number of elements in `ArrayStack`

   Answer: $O(1)$

4. (2%) We execute a method, `size`, to determine the number of elements in `LinkedStack`

   Answer: $O(1)$

5. (2%) An element is added to a `ArrayStack` of size n, which has reached full capacity.

   Answer: $O(n)$

6. (2%) An element is added to a `LinkedStack` of size n

   Answer: $O(1)$

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

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

   Answer: $O(n\log_2(n))$

8. (2%) We execute the following code segment

   ```
   for (int i = 1; i < n/3; i++)
     for (int j = 1; j < n/3; j*=3)
       System.out.println(i);
   ```

   Answer: $O(nlog_3(n))$

9. (1%) We execute the following code segment

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

   Answer: $O(n^2)$

Problem 5 (15 marks)

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

```
public class midterm2016_question_5 {

////////////////////////////////////////////////////////
// main method
////////////////////////////////////////////////////////
public static void main(String[] args) {
ArrayStack<ArrayStack<Integer>> topStack=new ArrayStack<ArrayStack<Integer>>();
ArrayStack<Integer> stack1=new ArrayStack<Integer>();
ArrayStack<Integer> stack2=new ArrayStack<Integer>();
ArrayStack<Integer> stack3=new ArrayStack<Integer>();

// Insert some data
stack1.push(3);
stack1.push(2);
topStack.push(stack1);
stack2.push(4);
stack2.push(1);
stack2.push(6);
stack2.push(5);
topStack.push(stack2);
stack3.push(9);
stack3.push(7);
stack3.push(8);
topStack.push(stack3);

System.out.println("\nContents of topStack before minValue():");
System.out.println(topStack.toString());

System.out.println("Minimum value of all integers in all stacks on the topStack: " +
                    minValue(topStack));

System.out.println("\nContents of topStack after minValue():");
System.out.println(topStack.toString());
}

}
```

1. (5%) What is printed by the `main()` method. Assume toString() accesses the array elements from 0 to the top of the stack.

```
ontents of stack before minValue():
3
2

4
1
6
5

9
7
8


Minimum value of all integers in all stacks on the topStack: 1

Contents of stack after minValue():
3
2

4
1
6
5

9
7
8
```

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,

```
////////////////////////////////////////////////////////
// Compute the min value of all stacks on the topStack
////////////////////////////////////////////////////////
public static int minValue(ArrayStack<ArrayStack<Integer>> topStack) {
int val,min,stackSize;
ArrayStack<Integer> tempStack=new ArrayStack<Integer>();
ArrayStack<Integer> stack=new ArrayStack<Integer>();
ArrayStack<ArrayStack<Integer>> tempTopStack=new ArrayStack<ArrayStack<Integer>>();

// Assume initially that the first values is the minimum
// If there is no first value the stack is empty, in
// that case quit with an error message
if(topStack.isEmpty())
{
System.out.println("Fatal error: topStack is empty");
System.exit(1);
}
min=topStack.peek().peek();

while(!topStack.isEmpty()) {
  stack=topStack.pop();
  // This code keeps stack queue intact
  // At the end tempStack contains stack in reverse order
  while(!stack.isEmpty())
       {
       val=stack.pop();
       if(val < min) min=val;
       tempStack.push(val);
       }
  // now copy tempStack back into into stack in the right order
  while(!tempStack.isEmpty())
       stack.push(tempStack.pop());
  tempTopStack.push(stack);
}
while(!tempTopStack.isEmpty())
      topStack.push(tempTopStack.pop());
return(min);
}

} // midterm2016_question_5
```

Problem 6 (15 marks)

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

```
public class midterm2016_question_6 {

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

ArrayQueue<Integer> queue1=new ArrayQueue<Integer>();
ArrayQueue<Integer> queue2=new ArrayQueue<Integer>();
ArrayQueue<Integer> queue3=new ArrayQueue<Integer>();

queue1.enqueue(3);
queue1.enqueue(2);
topQueue.enqueue(queue1);
queue2.enqueue(4);
queue2.enqueue(1);
queue2.enqueue(6);
queue2.enqueue(5);
topQueue.enqueue(queue2);
queue3.enqueue(9);
queue3.enqueue(7);
queue3.enqueue(8);
topQueue.enqueue(queue3);

System.out.println("Contents of topQueue before maxValue:");
System.out.println(topQueue.toString());

System.out.println("Maximum size of any element in the queues in topQueue: " +
                    maxValue(topQueue));

System.out.println("Contents of topQueue after maxValue:");
System.out.println(topQueue.toString());
}
```

1. (5%) What is printed by the `main()` method. Assume toString() accesses the array elements from the front (index 0) to the rear of the queue.

```
Contents of queue before maxLength:
3
2

4
1
6
5

9
7
8

Maximum size of any element on the queue of queues: 9
Contents of queue after maxLength:
3
2

4
1
6
5

9
7
8
```

2. (10%) Write the maxValue method below. Take care not to destroy the input queue structure in the method.

```
////////////////////////////////////////////////////////
// Compute the max value of all queuse in topQueue
////////////////////////////////////////////////////////
public static int maxValue(ArrayQueue<ArrayQueue<Integer>> topQueue) {
int val,max;
ArrayQueue<Integer> queue=new ArrayQueue<Integer>();

// Assume initially that the first values is the maximum
// If there is no first value the queue is empty, in
// that case quit with an error message
if(topQueue.isEmpty())
{
System.out.println("Fatal error: topQueue is empty");
System.exit(1);
}

max=topQueue.first().first();

int sizeTopQueue=topQueue.size();
for(int i=0;i<sizeTopQueue;i++)
  {
  queue=topQueue.dequeue();
  // This code keeps the queue intact
  int sizeQueue=queue.size();
  for(int j=0;j<sizeQueue;j++)
       {
       val=queue.dequeue();
       if(val > max) max=val;
       queue.enqueue(val);
       }
  topQueue.enqueue(queue);
  }
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