University of Western Ontario
Department of Computer Science
Computer Science 1027a Midterm Exam
November 1st, 2018, 3M-3250, 7pm-9pm, 2 hours
Professor John Barron

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.
• Problem 1 of the exam consists of Multiple Choice questions. Circle your answers on this exam paper.
• Problems 2-6 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, cellphones and laptops are not allowed!
• Do NOT unstaple your exam: loose pages tend to get lost.

Mark summary

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/20</td>
<td>/15</td>
<td>/15</td>
<td>/15</td>
<td>/15</td>
<td>/20</td>
<td>/100</td>
</tr>
</tbody>
</table>
Problem 1 (20 marks)

1. An ADT is an example of polymorphism True False
2. *Singly linked lists* can not be used to implement a Queue True False
3. A Queue is an example of a FILO structure True False
4. A Queue would be a better choice than a stack when converting an infix expression to a postfix expression True False
5. The Object class is a parent class of the *Person* class True False
6. If class A inherits from class B, B can access A’s protected attributes True False
7. If class B inherits from class A, B can access A’s private attributes True False
8. The class *arrayStack* can implements StackADT True False
9. The class *arrayStack* can extend StackADT True False
10. Inserting an element at the end of a linked list is always $O(1)$ True False
11. Inserting an element at the front of a linked list is always $O(n)$ True False
12. You can insert a node into a linked list in the middle and the end of the list True False
13. A stack must be implemented with an array True False
14. *Doubly linked list* is can be implemented as two *Singly linked lists* True False
15. `object1.equals(object2)` means the same thing as `object2.equals(object1)`, if `object1` and `object2` are instances of the same class True False
16. The terms “overloading” and “overriding” are synonyms in Java True False
17. At the very least, the `equals()` method is inherited from the Object class True False
18. With asymptotic complexity, $t(n) = 5\log_2 n + 3n$ is of the order $O(\log_2 n)$ True False
19. A stack typically has a top and a bottom reference/index True False
20. For a queue with a good circular array implementation, both enqueueing and dequeueing have an $O(1)$ complexity True False
Problem 2 (15 marks)

(15%) Consider the following Java program and answer the questions on the next page.

```java
public class Q2_Midterm2018 {
    private LinkedQueue<Integer> queue;
    private LinkedStack<Integer> stack;

    public Q2_Midterm2018()
    {
        queue = new LinkedQueue<Integer>();
    }

    public void addQueue(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()*stack.size());
        }
    }

    public String toString()
    {
        return "This queue contains:
        + queue.toString();
    }

    public static void main(String[] args)
    {
        Q2_Midterm2018 midterm2018 = new Q2_Midterm2018();
        midterm2018.addQueue(4);
        System.out.println(midterm2018.toString());
        midterm2018.whatDoIDo(10);
        System.out.println(midterm2018.toString());
    }
}
```
Please answer the following questions about the code above:

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

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

3. (2 %) What, if anything, is on the stack immediately after line 17 executes?

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

5. (7 %) Trace the program and write what will be printed to the screen by running java Midterm2018 here:
Problem 3 (15 marks)

This question concerns assignment 1. Below you are given the `Country.java` and `Continent.java` classes.

```java
public class Country {
    private String name;
    private int population;
    private int area;

    // Constructor class
    public Country(String name, int population, int area) {
        this.name = name;
        this.population = population;
        this.area = area;
    }

    public String getCountryName() {
        return this.name;
    }

    public int getCountryPopulation() {
        return this.population;
    }

    public int getCountryArea() {
        return this.area;
    }

    public void setCountryPopulation(int population) {
        this.population = population;
    }

    public void setCountryArea(int area) {
        this.area = area;
    }

    public String getCountryPopulationDensity() {
        return String.format("%4.2f", (double) this.population / (double) this.area));
    }

    public String toString() {
        return this.name + " has population " + this.population + " and area " + this.area;
    }
}
```

```java
public class Continent {
    private String countryName;
    private String continentName;

    // Constructor class
    public Continent(String countryName, String continentName) {
        this.countryName = countryName;
        this.continentName = continentName;
    }

    public String getContinentName() {
        return this.continentName;
    }
}
```
public String getCountryName() {
    return this.countryName;
}
public void setContinentName(String continentName) {
    this.continentName=continentName;
}
public void setCountryName(String countryName) {
    this.countryName=countryName;
}
public String toString() {
    return "The country " + this.countryName + " is on the continent " + this.continentName;
}

(15%) Assume that countryArray and continentArray arrays have been set up as on assignment 1. Fill in the Java code for the getPopulationOfContinent method on your assignment 1. Note getPopulationOfContinent is a method in the CountryContinentQuery class with private variables countryArray and continentArray that are initialized by its constructor class when an instance of it (an object) in generated in Main.java.
Your answer should fit the supplied code below.

public String getPopulationOfContinent(String continentStg) {
    int population=0;
    int index;
    boolean found;

    String s="Continent " + continentStg + " has population ";

    if(continentCt>0) {
        s=s+population+"\n";
    } else {
        s="No one lives on continent " + continentStg;
        s=s+"\n";
    }
    return s;
}
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 a `LinkedStack` of size \( n \)
   
   Answer:

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

3. (2%) An element is added to a `ArrayStack` of size \( n \), which has reached full capacity.
   
   Answer:

4. (2%) An element is added to a `ArrayStack` of size \( n \), which has **not** reached full capacity.
   
   Answer:

5. (2%) An element is added to a `LinkedStack` of size \( n \)
   
   Answer:

6. (2%) We execute the following code segment (\( \text{Math.log}(n) \) returns the \( \log_{10} \) of \( n \))

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

   Answer:

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

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

   Answer:

8. (1%) We execute the following code segment

   ```java
   for(int i=n;i<n*n;i++)
       System.out.println(i^2);
   ```

   Answer:
Problem 5 (15 marks)

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

```java
public class Q5_Midterm2018 {

    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>()

        // Insert some data
        stack1.push(3);
        stack1.push(2);
        stack1.push(9);
        topStack.push(stack1);
        stack2.push(4);
        stack2.push(1);
        topStack.push(stack2);

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

        System.out.println("Sum of all integers in all stacks on topStack: "+
                sumValue(topStack));

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

    // Compute the sum of all values of all stacks in topStack
    public static int sumValue(ArrayStack<ArrayStack<Integer>> topStack) {
        int sumval,stackSize;
        ArrayStack<Integer> stack=new ArrayStack<Integer>();
        if(topStack.isEmpty())
        {
            System.out.println("Fatal error: topStack is empty");
            System.exit(1);
        }
        sumval=0;
        while(!topStack.isEmpty()) {
            stack=topStack.pop();
            while(!stack.isEmpty())
            {
                sumval+=stack.pop();
            }
        }
        return(sumval);
    }
}
```
(15%) What is printed by the `main()` method. Assume `toString()` accesses the array elements from 0 to the top of the stack.
Problem 6 (20 marks)

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

```java
public class Q6_Midterm2018 {

    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>();
            queue1.enqueue(3);
            queue1.enqueue(2);
            queue1.enqueue(9);
            topQueue.enqueue(queue1);
            queue2.enqueue(4);
            queue2.enqueue(1);
            topQueue.enqueue(queue2);
            System.out.println("Contents of topQueue before sumValue:");
            System.out.println(topQueue.toString());
            System.out.println("Sum of all intergers on the queue of queues: " +
                         sumValue(topQueue));
            System.out.println("Contents of topQueue after sumValue:");
            System.out.println(topQueue.toString());
        }

        // Compute the sum of all values of all queuse in topQueue
        public static int sumValue(ArrayQueue<ArrayQueue<Integer>> topQueue) {
            int sumval=0;
            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);
            }
            int sizeTopQueue=topQueue.size();
            for(int i=0;i<sizeTopQueue;i++)
            {
                queue=topQueue.dequeue();
                int sizeQueue=queue.size();
                for(int j=0;j<sizeQueue;j++)
                {
                    sumval+=queue.dequeue();
                }
            }
        }
    }
```
\}
\}
return(sumval);
\}
\}

(15%) What is printed by this code?

(5%) What is the “major” flaw of the Q5_Midterm2018 and Q6_Midterm2018 classes, with respect to the stacks and queues they use?
Stacks and Queues Interfaces

```java
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 the string representation of this stack. /
     * @return the 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 the 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