Western University
Department of Computer Science
Computer Science 1027b Midterm Exam
2 hours

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.
- For Multiple Choice questions, circle your answers on this exam paper.
- For other questions, write your answers in the spaces provided in this exam paper.
- The marks for each individual question are given.
- The interfaces for `StackADT` and `QueueADT` are at the back of the exam.
- There are also 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 are not allowed!

Mark summary

1	2	3	4	5	6	total
/20	/20	/10	/15	/15	/20	/100

Problem 1: true/false (20 marks)

Choose **one** answer for each question.

1. (1 mark) A Java Interface always contains a constructor.
   (a) true
   (b) false
2. (1 mark) A Java Class always contains a constructor.
   (a) true
   (b) false
3. (1 mark) All Java Classes are derived from the Object Class.
   (a) true
   (b) false
4. (1 mark) All the methods in a class A can see A's private variables.
   (a) true
   (b) false
5. (1 mark) Any method defined in Class A can call all public and private methods defined in A.
   (a) true
   (b) false
6. (1 mark) If class A extends class B, then all the methods in A can see B's private variables.
   (a) true
   (b) false
7. (1 mark) If class A extends class B, then all the methods in A can see B's protected variables.
   (a) true
   (b) false
8. (1 mark) The terms parent class, superclass, and subclass all mean the same thing.
   (a) true
   (b) false
9. (1 mark) A child class cannot override a parent method that is declared as final.
   (a) true
   (b) false
10. (1 mark) The push, pop, peek and isEmpty methods of the `LinkedStack` class are $O(\log(n))$ operations.
    (a) true
    (b) false

11. (1 mark) An algorithm with time complexity $O(2^n)$ runs in polynomial time.

    (a) true

    (b) false

12. (1 mark) Data encapsulation requires that instance variables be declared as private

    (a) true

    (b) false

13. (1 mark) An interface can be used only of it is implemented by another class.

    (a) true

    (b) false

14. (1 mark) Private variables in Class A can be set by methods in an unrelated Class B using a setter method defined in Class A.

    (a) true

    (b) false

15. (1 mark) Public variables in Class A can be set by methods in an unrelated Class B using a setter methods defined in Class A.

    (a) true

    (b) false

16. (1 mark) Over loaded methods distinguish themselves by the number and type of their formal parameters.

    (a) true

    (b) false

17. (1 mark) Method overriding is where a subclass replaces the implementation of one or more of its parent's methods.

    (a) true

    (b) false

18. (1 mark) Polymorphism allows a reference variable to point to objects of related types.

    (a) true

    (b) false

19. (1 mark) `dequeue` is an $O(n)$ operation for queues implemented using a circular array.

    (a) true

    (b) false

20. (1 mark) `dequeue` is an $O(n)$ operation for queues implemented using an array.

    (a) true

    (b) false

Problem 2 (20 marks)

Consider the following Java program:

```
 1 class midterm2013 {
 2
 3   private String stg;
 4
 5   // Constructor
 6   midterm2013(String stg){
 7     this.stg = stg;
 8   }
 9
10   // Getter
11   public String getValue() {
12     return(stg);
13   }
14
15   // Use a stack to sum the digits in a string
16   public void run(){
17     // Stack of characters using linked list structure
18     StackADT<Character> s = new LinkedStack<Character>();
19     String p=getValue();
20     System.out.println("\nInitial value of p=" + p);
21     System.out.println("\nPushing operation intermediate results");
22
23     while(p.length()>0)
24     {
25       s.push(p.charAt(0));
26       System.out.println(s.peek() + " pushed");
27       p=p.substring(1, p.length());
28     }
29
30     int sum=0, intValue;
31     System.out.println("\nPopping operation intermediate results");
32
33     while(s.isEmpty()==false){
34       // Convert character to string and then to its integer
35       // value, i.e. '9' --> "9" --> 9
36       intValue=Integer.parseInt(Character.toString(s.pop()));
37       sum=sum+intValue;
38       System.out.println(intValue + " is popped and the sum is: " + sum);
39     }
40
41     System.out.println("\nThe final sum of the digits in the string is: " + sum + "\n");
42   }
43 }
44
45 // Test class to execute the run method
46 class Test{
47   public static void main(String[] args){
48     midterm2013 p = new midterm2013("7304952");
```

```
49     p.run();
50   }
51 }
```

1. (2 marks) What methods for stacks are used in this program?

   Answer: pop, push, peek, isEmpty

2. (2 marks) When "java Test" is run, what does line 48 do?

   Answer: Creates an object that is an instance of class midterm2013 with actual parameter "7304952" passed to its constructor.

3. (2 marks) What is `s` in line 18?

   Answer: s is a stack of characters (implemented using a linked list)

4. (2 marks) In `p.run()`, what does line 19 do?

   Answer: getValue is a getter method that return private variable stg's value to p

5. (12 marks) Hand trace the program

   Answer:

```
Initial value of p=7304952

Pushing operation intermediate results
7 pushed
3 pushed
0 pushed
4 pushed
9 pushed
5 pushed
2 pushed

Popping operation intermediate results
2 is popped and the sum is: 2
5 is popped and the sum is: 7
9 is popped and the sum is: 16
4 is popped and the sum is: 20
0 is popped and the sum is: 20
3 is popped and the sum is: 23
7 is popped and the sum is: 30

The final sum of the digits in the string is: 30
```

Problem 3 (10 marks)

Consider the following Java code:

```
int sum=0;
for(int i=-2;i<n;i++)
   sum=sum+i;
System.out.println("sum=" + sum);
```

It might be useful for you to know that $\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$ in answering the questions below.

1. (4 marks) What is the value of `sum` for $n = 8$?

   Answer: 25

2. (3 marks) What is the value of `sum` for $n = -8$?

   Answer: 0

3. (3 marks) What is the value of `sum` for $n = 0$?

   Answer: -3

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 marks) An element is inserted in a `LinkedStack` of size n

   Answer: $O(1)$

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

   Answer: $O(1)$

3. (2 marks) An element is inserted in a `LinkedQueue` of size n

   Answer: $O(1)$

4. (2 marks) We execute the following code segment

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

   Answer: $O(n^2)$

5. (3 marks) We execute the following code segment

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

   Answer: $O(n^2)$

6. (2 mark) We fill an array `t` of size n using the rule `t[0]=1` and `t[i+1]=2*t[i]` for all other indices.

   Answer: $O(n)$

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

```
for (int i = 1; i < (n*n+23*n+12); i++)
  System.out.printlnt(i);
```

   Answer: $O(n^2)$

Problem 5 (15 marks)

Rewrite the `toString` method of the `LinkedStack` class. The top of the stack should come first. Remember that the `LinkedStack` class looks like this:

```
public class LinkedStack<T> implements StackADT<T>{
  /** indicates number of elements stored */
  private int count;
  /** pointer to top of stack */
  private LinearNode<T> top;
  ..
}
```

and that `LinearNode` is as follows:

```
public class LinearNode<T>{
  private LinearNode<T> next;
  private T element;

  public LinearNode(){
    next = null;
    element = null;
  }
  public LinearNode (T elem){
    next = null;
    element = elem;
  }
  public LinearNode<T> getNext(){
    return next;
  }
  public void setNext (LinearNode<T> node){
    next = node;
  }
  public T getElement(){
    return element;
  }
  public void setElement (T elem){
    element = elem;
  }
}
```

Answer:

```
  public String toString(){
    String s = "";
    LinearNode<T> tmp = top;
    while(tmp != null){
      s += tmp.getElement() + "\n";
      tmp = tmp.getNext();
    }
    return s;
  }
```

Problem 6 (20 marks)

There is a long queue of customers in front of the new PappleStore. Some customers have coupons, some do not. The manager asks you to reorganize the queue so that all customers with coupons come before all customers without coupons. The order among customers with coupons should not change, and similarly for customers without coupons. For instance, suppose that the queue is (a, b, c, d, e), where a came first and e came last, and suppose that b, c, d have coupons and a, e do not. The queue after reorganization should be (b, c, d, a, e).

1. Suppose that you have a class `Customer` that contains a method `public boolean hasCoupon()`. Consider the following method:

```
public static void mystery(LinkedQueue<Customer> line){
  int sz = line.size();
  for (int i = 0; i < sz; i++){
    Customer c = line.dequeue();
    System.out.print(c.hasCoupon() + " ");
    line.enqueue(c);
  }
}
```

(5 marks) If called upon the queue (a, b, c, d, e) of the previous example, what do you see?
Answer: `false`, `true`, `true`, `true`, `false`

(5 marks) What happened to the queue after execution of this method?
Answer: It did not change

2. (10 marks) Write a static method

```
public static LinkedQueue<Customer> reorganize(LinkedQueue<Customer> line)
```

that returns a new queue in the required order. You may only use the methods from the `QueueADT` interface, and the constructor of `LinkedQueue`. Hint: use the same kind of loop as in the mystery method above, twice.

```
public static LinkedQueue<Customer> reorganize(LinkedQueue<Customer> line){
  LinkedQueue<Customer> line2 = new LinkedQueue<Customer>();
  int sz = line.size();

  for (int i = 0; i < sz; i++){
    Customer c = line.dequeue();
    if (!c.hasCoupon())
      line2.enqueue(c);
    else
      line.enqueue(c);
  }

  int sz2 = line2.size();

  for (int i = 0; i < sz2; i++){
    Customer c = line2.dequeue();
    line.enqueue(c);
    line2.enqueue(c);
  }
  return line2;
}
```

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 */
  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 */
Public String toString();
```

Rough work 1/4

Rough work 2/4

Rough work 3/4

Rough work 4/4