CSC 115

Midterm Exam:
Thursday, 4 July 2019

Name:_____(please print clearly!)

UVic ID number:_____

Signature:_____

Exam duration: 40 minutes

Instructor: Celina Berg

Students must check the number of pages in this examination paper before beginning to write, and report any discrepancy immediately.

- We will not answer questions during the exam. If you feel there is an error or ambiguity, write your assumption and answer the question based on that assumption.
- Answer all questions on this exam paper.
- The exam is closed book. No books or notes are permitted.
  Electronic devices are not permitted.
- The marks assigned to each question and to each part of a question are printed within brackets. Partial marks are available.
- There are ten (10) pages in this document, including this cover page.
- Pages 9 and 10 contain code to be used as a reference for Questions 2 and 3. You may remove this sheet of paper from the back of the exam.
- Clearly indicate only one answer to be graded. Questions with more than one answer will be given a zero grade.
- It is strongly recommended that you read the entire exam through from beginning to end before beginning to answer the questions.
- Please have your ID card available on the desk.

Left blank for scratch work.
If you are writing an answer to a question here, clearly indicate what question you are answering both here and under the question itself.

Question 1(1 mark)

Trace the following program and circle the choice below (a,b,c,d,e,f) that represents the output.

```
public class C {
      int x;
      public C() {
            System.out.println("C constructor 1");
            this.x = 0;
      }
      public C(int x) {
            this.x = x;
            System.out.println("C constructor 2");
      }
      public void foo() {
            System.out.println("C foo:"  + x);
      }
}

public class A extends C {
      public A() {
            System.out.println("A constructor 1");
      }
      public A(int x) {
            System.out.println("A constructor 2");
      }
      public void foo() {
            System.out.println("A foo:" + x);
      }
}

public class FinalExamInheritance {
      public static void main(String[] args) {
            C two = new A(5);
            two.foo();
      }
}
```

a. A constructor 2
   A foo:5
b. C constructor 1
   A constructor 2
   A foo:5
c. C constructor 2
   A constructor 2
   A foo:5
d. C constructor 1
   A constructor 2
   A foo:0
e. C constructor 1
   A constructor 2
   C foo:0
f. None of the above

Question 2a (10 marks)
The following is code for a `LinkedList<T>` class that implements `List<T>` and uses `Node<T>` class. You can find `List<T>` and `Node<T>` on **Pages 9-10**
We have omitted the implementation of the other methods required in this class intentionally as we do not want you to implement them.
You **cannot** change or add fields to this class.

Answer this question as instructed in the **`//TODO`** tags below (there are 2 of them)

```
public class LinkedList<T> implements List<T> {

    private Node<T> head;
    private int count;

    public LinkedList() {
        head = null;
        count = 0;
    }

    /* Method Name: removeBack
     * Parameters:  none
     * Purpose:     remove element from the back of the list
     * Returns:    (T) the value of element removed
     * Throws:     OffListException if called when the list is empty
     */
    // TODO: implement this method according to the documentation above
    // Use iteration not recursion if you need to traverse the list.
    // Implementations that use recursion will receive a zero grade
```

```
    /* Method Name: searchAndUpdate
     * Parameters: (T) curVal, (T) newVal
     * Purpose:  update every occurrence of curVal in the list with newVal
     * Examples:
     *  if list is {1,2,2,3,5,2} and curVal is 2 and newVal is 10
     *     after searchAndUpdate list is {1,10,10,3,5,10}
     *  if list is {1,2,2,3,5,2} and curVal is 8 and newVal is 10
     *     after searchAndUpdate list is {1,2,2,3,5,2}
     *  if list is {s1,s2,s3,s4} and curVal is curS and newVal is newS
     *      where s1, s2, s3, s4, curS and newS are all Song objects
     *          and s1 and s3 are the same Song as curS
     *     after searchAndUpdate list is {newS,s2,newS,s4}
     * Returns:  nothing
     */
    // TODO: implement this method according to the documentation above
    // Use recursion not iteration if you need to traverse the list.
    // Implementations that use iteration will receive a zero grade
    // TIP: The method does not ask you to remove/add new elements,
    //      but asks you to update existing elements
```

```
} //end of LinkedList class
```

Question 2b (3 marks)
NOTE: Answer this question in **POINT FORM** not essay form. We expect you **not** to fill the page.

1) What is the growth rate of each method you implemented in Question 2a in Big O terms?

   a. `removeBack`

   b. `searchAndUpdate`

2) Imagine we updated the `LinkedList<T>` class to include a `tail` field which is a reference to the last `Node<T>` in the list. We would then need to reconsider the implementation of every constructor and method in this class and potentially change it to account for the addition of the `tail` field.

   For each of the methods you implemented in Question 2a, considering the changes you would make to account for the `tail` field (you don't need to make them!), what would the growth rate be after the change and why?

   a. `removeBack`

   b. `searchAndUpdate`

Question 3 (2 marks)

A `Stack` interface that holds `chars` and a `StackEmptyException` class is given to you on **Page 10**. The following method uses the `StackArrayBased` class (omitted intentionally) that implements the `Stack` interface to helps create and return a new String that is the reverse of the given `String` BUT…there is an error in this code. **You are to do the following two things:**

1) Find and fix the error in this code by editing the provided code.
2) Update the code wherever necessary so that it uses the `StackRefBased` (omitted intentionally) implementation of `Stack` instead of the `StackArrayBased Stack`

```
/*
 * Purpose: reverses str by pushing each char in str onto a Stack
 *  then popping each element off the Stack, adding it to result
 * Parameters: String str - the String to be reversed
 * Returns: String - the reversed result String
 */
public static String reverseString(String str) {
    Stack stk = new StackArrayBased();



    String result = "";



    for(int i=0; i<str.length(); i++) {



        stk.push(str.charAt(i));



    }



    while (!stk.isEmpty()) {



            result +=stk.pop();



    }



    return result;
}
```

END OF EXAM

Question	Value	Mark
1	1	
2a	10	
2b	3	
3	2	
Total	**16**	

You can remove this back page.
This code should be used as a reference for Questions 2 & 3.

```
public interface List<T> {

    /* Parameters: (T) val
     * Purpose:  add val to the front of the list
     * Returns:  nothing
     */
    public void addFront (T val);

    /* Parameters: nothing
     * Purpose:  get the size of the list
     * Returns:  (int) size
     */
    public int size ();

    /* Parameters: (int) position
     * Purpose:  get the value at specified position in the list
     * Returns:  (T) the value
     * Throws: OffListException if position<0 or position>=count
     */
    public T get (int position) throws OffListException;

    /* MethodName: removeBack
     * Parameters: nothing
     * Purpose:  remove element from the back of the list
     * Returns:  (T) the value of element removed
     * Throws:  OffListException if called when the list is empty
     */
    public T removeBack() throws OffListException;

    /* Parameters: (T) curVal, (T) newVal
     * Purpose:  update every occurrence of curVal in the list with newVal
     * Examples:
     *  if list is {1,2,2,3,5,2} and curVal is 2 and newVal is 10
     *     after searchAndUpdate list is {1,10,10,3,5,10}
     *
     *  if list is {1,2,2,3,5,2} and curVal is 8 and newVal is 10
     *     after searchAndUpdate list is {1,2,2,3,5,2}
     *
     *  if list is {s1, s2, s3, s4} and curVal is curS and newVal is newS
     *      where s1, s2, s3, s4, curS and newS are all Song objects
     *          and s1 and s3 are the same Song as curS
     *     after searchAndUpdate list is {newS, s2, newS, s4}
     * Returns:  nothing
     */
    public void searchAndUpdate(T curVal, T newVal);


} // end of List<T> interface


public class OffListException extends Exception {

} // end of OffListException
```

```
public class Node<T> {
    private   T value;
    protected Node<T> next;
    protected Node<T> prev;

    public Node () {
        this.value = null;
        this.next = null;
        this.prev = null;
    }

    public Node (T value) {
        this.value = value;
        this.next = null;
        this.prev = null;
    }

    public T getValue() {
        return value;
    }

    public void setValue(T value) {
        this.value = value;
    }
} // end of Node<T> class


public interface Stack {
    /* Purpose: returns the number of items currently pushed onto the Stack.
     * Returns: int - the number of items in the stack
     */
      int size();

    /* Purpose:  returns the boolean state of the Stack (empty or not empty)
     * Returns: boolean - true if stack is empty, false otherwise
     */
      boolean isEmpty();

    /* Purpose: places given element onto the top of the Stack
     * Parameters: char - element
     * Returns: nothing
     */
      void push (char element);

    /* Purpose: removes the element at the top of the Stack
     * Returns: the value of the element that was at the top of the Stack
     * Throws: StackEmptyException if called on an empty Stack
     */
      char pop() throws StackEmptyException;

    /*
     * Purpose: gets and returns the value in the element at the top of the Stack
     * Returns: char – the value of the top element
     * Throws: StackEmptyException if called on an empty Stack
     */
      char peek() throws StackEmptyException;
} // end of Stack interface

public class StackEmptyException extends Exception {

} // end of StackEmptyException
```