# CSC 115: Fundamentals of Programming I
# Spring 2019
# Instructor: Celina Berg
Midterm Exam: Friday, 15 March 2019

Name : _____ (please print clearly)

UVic ID number : _____

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

- This is an open-book exam.
- All answers are to be written on this exam paper.
- There are 12 printed pages in this exam paper, including this cover page.
- No electronic devices are permitted; cellphones / smartphones must be turned off.
- We recommend you read the entire exam from beginning to end before starting on your answers.

Given the difficulty in reaching all students sitting for this midterm exam, the invigilators cannot answer student questions or queries. If you are unsure of the meaning of a question, then please write down the assumptions made while writing your answer.

| Question | Maximum | Score |
|---|---|---|
| 1 | 6 | |
| 2 | 5 | |
| 3 | 10 | |
| 4 | 15 | |
| *Total* | *36* | |

(This page intentionally left blank.)

**Question 1: (6 marks)**
Consider the following definition contained within ***Node.java*** (with line numbers provided as reference).

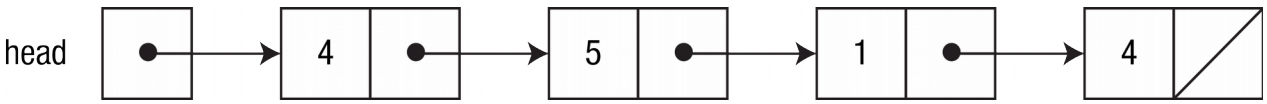
```
 1  class Node {
 2          public int value;
 3          public Node next;
 4
 5          public Node() {
 6              value = 0;
 7              next = null;
 8          }
 9
10          public int getValue() {
11              return value;
12          }
13
14          public void setValue (int v) {
15              value = v;
16          }
17
18          public Node getNext() {
19              return next;
20          }
21
22          public void setNext(Node n) {
23              next = n;
24          }
25   }
```

When answering the questions on the next page (i.e., page 3), base your answers on this definition. You may define additional ***Node*** variables as needed.

a) Draw the “boxes and arrows” representation that results from the following statements. [3 marks]

```
Node head = new Node(4);
Node nxt = new Node(1);
head.next = nxt;
Node n = nxt;
nxt = new Node(5);
n.next = nxt;
n = nxt;
nxt = new Node(4);
n.next = nxt;
```

(b) Assume that the current state of head is as shown in the following picture:




Draw the “boxes and arrows” representation that results after the following statements have executed: [3 marks]

```
head = head.next;
n = head.next;
n.next = null;
```

**Question 2 [5 marks]**

Given this class:

```
class MTException extends Exception {
}
```

What is the output of the following program?

```
class MT2Q2 {
     public static void bar (int x) throws MTException {
          System.out.println("Q");
          if (x > 2) {
               throw new MTException();
          }
          System.out.println("S");
     }
     public static void foo (int x) throws MTException {
          System.out.println("T");
          bar(x);
          System.out.println("A");
     }
     public static void main (String[] args) {
          try {
               System.out.println("D");
               foo(8);
               System.out.println("F");
          } catch (MTException e) {
               System.out.println("Z");
          }
          System.out.println("P");
     }
}
```

**Question 3 [10 marks]**

Consider these operations for an ***IntegerStack:***:

- ***void push(int value)***: Places ***value*** onto the top of the stack.
- ***int pop()***: Removes the value at the top of the stack, and returns that value to the caller of ***pop***.
- ***int peek()***: Returns the value at the top of the stack to the caller of ***peek***, but does not change the stack's contents.
- ***boolean isEmpty()***: Returns ***true*** if the stack contains no items, ***false*** otherwise.

Write a method named ***filterAndReverseItems*** that accepts two parameters (***IntegerStack stack, int threshold***). The method should return a new stack without the values in `stack` that are less than `threshold`, and in the opposite order as ***stack.***

For example, if ***someStack*** = {41, 59, 2, 65, 3} (where the left-most integer in the curly braces is the top of the stack), then a call of ***filterAndReverseItems(someStack,41)*** will return a new stack with contents {65, 59, 41}.

If ***someStack*** = {} then a call of ***filterAndReverseItems(someStack,10)*** will return {}.

Your code for `filterAndReverseItems` is allowed to destroy `someStack`. That is, `someStack` can be {} when your method returns.

Your code for ***filterAndReverseItems*** must be independent of the stack's implementation (that is, the list may only be used with ***IntegerStack*** ADT operations `push`, `pop`, `peek` and `isEmpty`).

**The only *IntegerStack* operations you may use are the operations listed. You are not permitted to make assumptions about the way *IntegerStack* is implemented.**

Your method must work for stack values other than those given in the example. Write only the code for the method (i.e., do not write a whole program). You may use the following page (i.e., page 7) for your answer.

(This page available for your answer to question 3.)

**Question 4 [15 marks]**

For this question, consider the code on pages 11 and 12 of this exam. (You may detach that last page from this exam.)

(a) [5 marks] Complete the method ***addXToAll*** as described in the comments below.

```
/*
* PURPOSE:
* Add x to all values in the list.
*
* PRE-CONDITIONS:
* None.
*
* EXAMPLES:
*
* if l is {1,2,3,4} and l.addXToAll(3) is called,
* then l is {4,5,6,7}
*
* if l is {-1,4,-5} and l.addXToAll(5) is called,
* then l is {4,9,0}.
*
* if l is {} and l.addXToAll(3) is called,
* then l is {}
*/
public void addXToAll (int x) {
```

(b) [10 marks] Complete the method ***duplicateNegative*** as described in the comments below. A negative value is one that is less than 0.

```
/*
* PURPOSE:
* Duplicate every element with a value that is negative
*
* EXAMPLES:
*
* if l is {-32,26,58,-23} and l.duplicateNegative() is called,
* then l is {-32,-32,26,58,-23,-23}
*
* if l is {-31,26,59} and l.duplicateNegative() is called,
* then l is {-31,-31,26,59}
*
* if l is {26} and l.duplicateNegative() is called,
* then l is {26}
*
* if l is {} and l.duplicateNegative() is called,
* then l is {}
*/
public void duplicateNegative() {
```

(This page available for your answer to question 4 b.)

***The following code must be used for reference when completing question 4.***

***Nothing you write on pages 11 and 12 will be graded.***
***You may detach this page from your exam.***

```
class Node {
	public int value;
	public Node next;
	public Node prev;

	public Node() {
		this(0,null,null);
	}

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

	public int getValue() {
		return value;
	}

	public void setValue (int v) {
		value = v;
	}

	public Node getNext() {
		return next;
	}

	public void setNext(Node n) {
		next = n;
	}

	public Node getPrev() {
		return prev;
	}

	public void setPrev(Node n) {
		prev = n;
	}

}
```

```
class LinkedList {
	private	Node head;
	private	Node tail;
	private	int  count;

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

	public void addFront(int value) {
		// Assume this works as expected.
		// The list is always in a valid state
	}

	public void addXToAll () {
		// implement this in Question 4 a
	}

	public void duplicateNegative() {
		// implement this in Question 4 b
	}

}
```