CSC 115

Midterm Exam:
Thursday, 13 June 2019

Name:_____RUBRIC_____(please print clearly!)

UVic ID number:_____

Signature:_____

Exam duration: 40 minutes

Instructor: Celina Berg

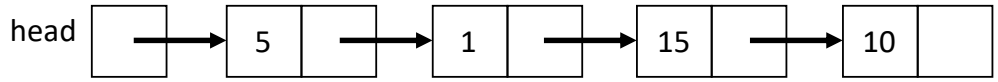
Students must check the number of pages in this examination paper before beginning to write, and report any discrepancy immediately.

- 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.

Consider the following definition contained within ***Node.java***
When answering Question 1 on the next page, base your answers on this definition.

```
public class Node {

    private   int  value;
    protected Node next;

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

    /* Parameters: nothing
     * Purpose:  get the value of this Node
     * Returns:  (int) the value
     */
    public int getValue() {
        return value;
    }

    /* Parameters: (int) value
     * Purpose:  set the value of this Node to value
     * Returns:  nothing
     */
    public void setValue(int value) {
        this.value = value;
    }

    /* Parameters: nothing
     * Purpose:  get the next of this Node
     * Returns:  (Node) the next
     */
    public Node getNext() {
        return next;
    }

    /* Parameters: (Node) next
     * Purpose:  set the next of this Node to next
     * Returns:  nothing
     */
    public void setNext(Node next) {
        this.next = next;
    }
}
```

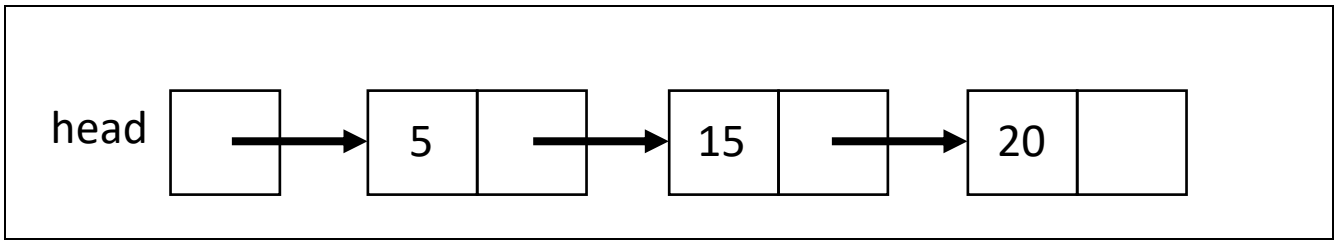
Question 1 (3 marks)
Using the Node class on the previous page and assuming the current state of head is shown in the following picture:



Draw a "boxes and arrows" representation that results after these statements have executed:

```
Node n1 = head.next;
Node n2 = n1;
n1 = n1.next;
head.next = n2.next;
n1.next = new Node(20);
```

Draw your final answer in box provided. Use the remaining page for scratch work.



Grading

- 3/3 marks for solution above
- 2/3 marks if you had one of these:
  - head ->5->1->20
  - head ->1->15->20
  - head ->5->15->20->10
- 1/3 marks if you added a node with 20 to the list

Question 2(5 marks)
The following is code for a `ItemArrayList` class that uses `Item` and `ItemList` (Pages 9-10) Answer this question as instructed in the **`//TODO`** tag below.

```
public class ItemArrayList implements ItemList {

    private static final int INIT_SZ = 2;  // initial size of data
    private int count;    // number of elements in this list
    private Item[] data;  // array to hold list elements
```

`/*TODO:`
This implementation of `find` has errors in it. Find and fix the errors by editing the code given. That is, cross out the wrong code and writing in code you want to add.
DO NOT rewrite the whole function. Marks will be deducted for "fixing" code that is correct. **`*/`**

```
    /* Parameters: (Item) item
     * Purpose:  gets the position item is found in this list
     * Returns:  (int) position is found in the list, -1 if not found
     */
    public ~~static~~ int find (Item item) {
        for ( int i = 0; i < ~~data.length~~ count; i++ ) {
            if ( ~~item == data[i]~~ item.equals(data[i])) {
                return ~~item~~ i;
            } ~~else {~~
                ~~return -1;~~
            ~~}~~
        }
        return -1;
    }// end of find
    ...
    // OTHER METHOD IMPLEMENTATIONS OMITTED INTENTIONALLY FOR SPACE
}// end of ItemArrayList class
```

Grading

- 1 mark per correction
  NOTE: removing the else and moving the `return -1` is one correction
- -1 for unnecessary/incorrect corrections

Question 3a (10 marks)
The following is code for the `GroceryCart` that uses `Item`, `ItemList` and `ItemArrayList`.
Complete the sections of code marked with **`//TODO`** tags (there are **3**).
You cannot change any code we have provided you.

```
public class GroceryCart{

    private String storeName;
    private ItemList cart;

    /* Parameters: (String) storeName
     * Purpose:  initialize this GroceryCart storeName to storeName
     *           and cart to an empty ItemArrayList
     */
    public GroceryCart (String storeName) {
       // TODO 1 – complete the implementation according to documentation

          this.storeName = storeName;
          cart = new ItemArrayList();


    }



    /* Method name: addItem
     * Parameters: (Item) item
     * Purpose:  add item to cart
     * Returns:  nothing
     */
    // DO NOT IMPLEMENT, assume it will behave according to documentation
```

Grading
- 1 mark per line

```
    /* Method name: removeItem
     * Parameters: (Item) item
     * Purpose:  remove given item from cart if it is in there
     * Returns:  nothing
     */
    // TODO 2 – implement the method removeItem according to documentation

  public void removeItem(Item item) {
          int index = cart.find(item);
          if (index != -1)
              cart.remove(index);

  }
```

Grading

- 1 mark signature (not static, return void, Item parameter type)
- 1 mark call to cart.find
- 1 mark for condition on call to cart.remove
- 1 mark for call to cart.remove

```
    /* Method name: total
     * Parameters: none
     * Purpose: calculates the total price of all Items in this cart
     * Returns:  (double) - the total price
     */
    // TODO 3 – implement the method total according to documentation


  public double total() {
          double sum = 0;
          for (int i=0; i<cart.size(); i++)
              sum += cart.get(i).getPrice();

          return sum;
  }
```

Grading

- ½ mark signature (not static, return double, no parameters)
- 1 mark loop that accumulates a sum and returns it
- ½ mark for using cart.size() in loop
- 1 mark for calling getPrice() (accept data[i].getPrice() even though not correct)
- 1 mark for calling cart.get(i) to get the item

```
} //end of GroceryCart class
```

Question 3b (2 marks)

We have decided to change our `GroceryCart` implementation from using an array-based list (`ItemArrayList`) to using a reference-based list (`ItemLinkedList`).

Assuming that I have written a `ItemLinkedList` class that implements the `ItemList` interface, I want to know what changes will I have to make to the `GroceryCart` class.

1) Clearly list which part(s) of the `GroceryCart` class will have to change.
2) Write the details of the change(s) below. Marks will be deducted for incorrect or unnecessary changes.

Change to the constructor in GroceryCart

```
cart = new ItemLinkedList();
```

Grading

- 2 marks if you
  - indicated both where to make the change **and** what change to make
  - did not suggest extra changes unnecessary changes

END OF EXAM

Question	Value	Mark
1	3	
2	5	
3a	10	
3b	2	
Total	**20**	

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

```
public interface ItemList {

    /* Parameters: (Item) item
     * Purpose:  add item to this list
     * Returns:  nothing
     */
    void add (Item item);

    /* Parameters: (int) position
     * Purpose:  remove Item from given position in this list
     * Precondition:  0 <= position < l.size()
     * Returns:  nothing
     */
    void remove (int position);

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

    /* Parameters: (Item) item
     * Purpose:  gets the position item is found in this list
     * Returns:  (int) position item is found in the list, -1 if not found
     */
    int find (Item item);

    /* Parameters: (int) position
     * Purpose:  gets the item at given position in this list
     * Precondition:  0 <= position < l.size()
     * Returns:  (Item) the item at position
     */
    Item get (int position);

}// ends ItemList interface
```

```
public class Item {
    private String  barcode;
    private String  name;
    private double  price;

    /* Parameters: (String) barcode, (String) name, (double) price
     * Purpose:  initialize fields of this Item
     *     to barcode, name, price
     */
    public Item (String barcode, String name, double price) {
        this.barcode = barcode;
        this.name    = name;
        this.price   = price;
    }
    /* Parameters: nothing
     * Purpose:  get the barcode of this Item
     * Returns:  (String) the barcode
     */
    public String getBarcode() {
        return barcode;
    }
    /* Parameters: (String) barcode
     * Purpose:  set the barcode of this Item to barcode
     * Returns:  nothing
     */
    public void setBarcode(String barcode) {
        this.barcode = barcode;
    }
    /* Parameters: nothing
     * Purpose:  get the name of this Item
     * Returns:  (String) the name
     */
    public String getName() {
        return name;
    }
    /* Parameters: (String) name
     * Purpose:  set the name of this Item to name
     * Returns:  nothing
     */
    public void setName(String name) {
        this.name = name;
    }
    /* Parameters: nothing
     * Purpose:  get the price of this Item
     * Returns:  (double) the price
     */
    public double getPrice() {
        return price;
    }
    /* Parameters: (double) price
     * Purpose:  set the price of this Item to price
     * Returns:  nothing
     */
    public void setPrice(double price) {
        this.price = price;
    }
    /* Parameters: (Item) other
     * Purpose:  determines whether this Item is the same as other Item
     * Returns:  (boolean) - true if equal, false otherwise
     */
    public boolean equals(Item other) {
        return  this.barcode.equals(other.barcode) && this.name.equals(other.name);
    }
}// end of Item class
```