CS 1027A - Assignment 4 - Splitting The Bill

Graded

3 Days, 21 Hours Late

Student

Mohammed Ali Abdul-nabi

Total Points

20 / 20 pts

Autograder Score

15.0 / 15.0

Passed Tests

[---------- TEST 01 (UOL) ----------] (1/1)
[---------- TEST 02 (UOL) ----------] (1/1)
[---------- TEST 03 (UOL) ----------] (1/1)
[---------- TEST 04 (UOL) ----------] (1/1)
[---------- TEST 05 (UOL) ----------] (1/1)
[---------- TEST 01 (Split) ----------] (0.5/0.5)
[---------- TEST 02 (Split) ----------] (0.5/0.5)
[---------- TEST 03 (Split) ----------] (0.5/0.5)
[---------- TEST 04 (Split) ----------] (0.5/0.5)
[---------- TEST 05 (Split) ----------] (0.5/0.5)
[---------- TEST 06 (Split) ----------] (0.5/0.5)
[---------- TEST 07 (Split) ----------] (0.5/0.5)
[---------- TEST 08 (Split) ----------] (0.5/0.5)

Question 2

Code Logic

1 / 1 pt

✔ **– 0 pts** Correct - Meaningful variable names, private instance variables used

– 0.5 pts Click here to replace this description.

– 1 pt Wrong - No meaningful logic

Question 3

Code Formatting/Readability

2 / 2 pts

✔ **– 0 pts** Correct

– 0.5 pts Click here to replace this description.

– 1 pt Click here to replace this description.

– 1.5 pts Click here to replace this description.

– 2 pts No proper code formatting. Code not readable

Question 4

Comments

2 / 2 pts

✔ **– 0 pts** Correct - Comments are proper and relevant

– 0.5 pts Click here to replace this description.

– 1 pt Click here to replace this description.

– 2 pts Wrong - Comments are NOT proper and relevant or/and no comments included.

Click here to replace this description.

– 0 pts Click here to replace this description.

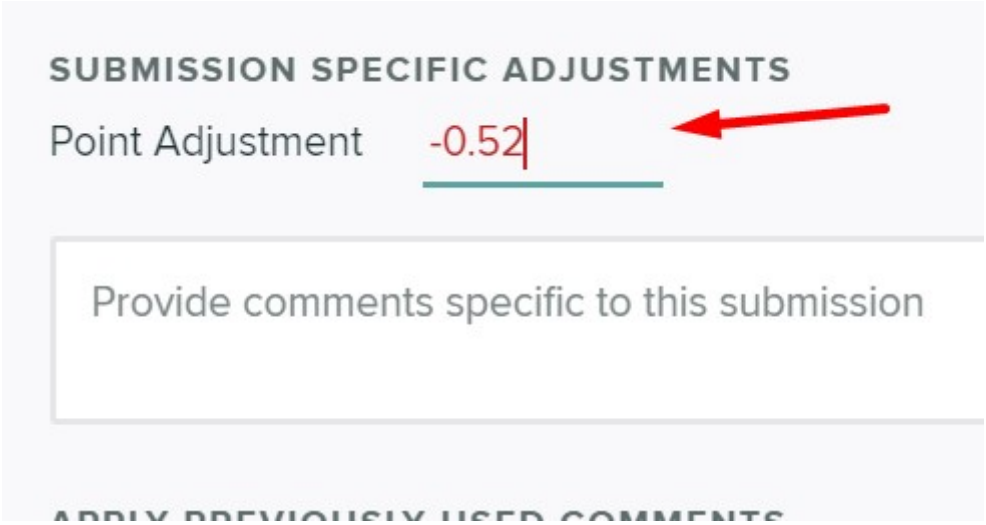
Question 5

Penalties **0** / 0 pts

5.1 ***Late Submissions* -2/day** **0** / 0 pts

– **0 pts** Click here if not late.

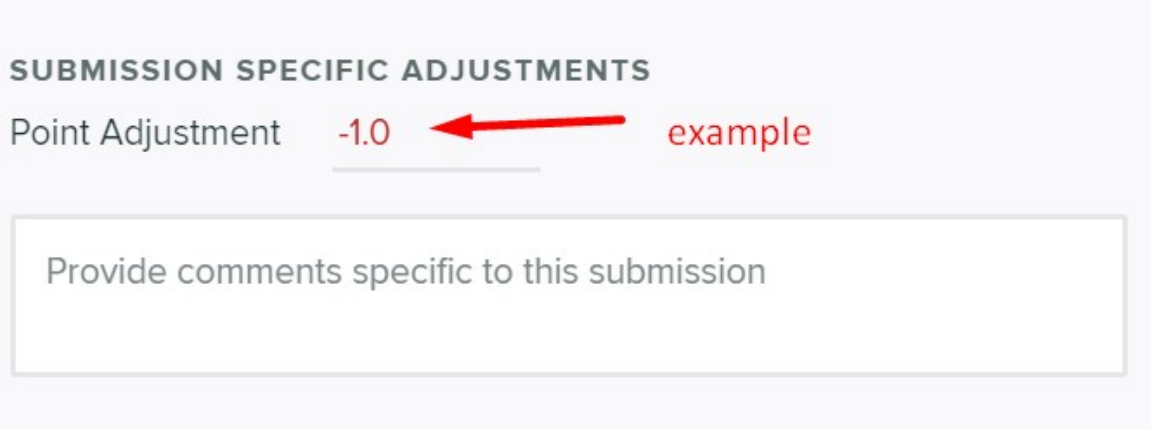
✔ – **0 pts** **@TAs: DO NOT ADD YOUR OWN RUBRICS HERE** *Please enter the deduction in the* **Point Adjustment** *field below if late penalty applies.*



5.2 **Incorrect submission (doesn't compile, package line, .class file, etc.) -5** **0** / 0 pts

✔ – **0 pts** Click here if no submission error

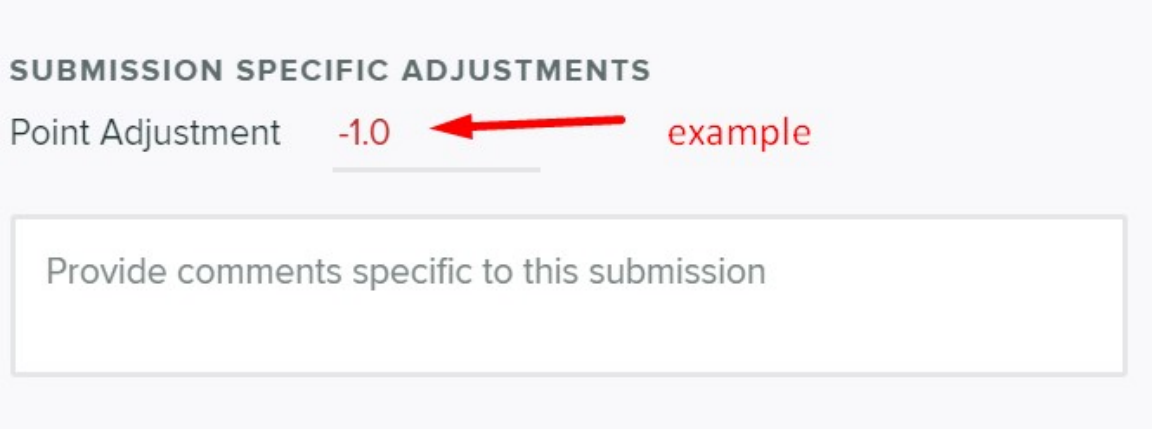
– **5 pts** **@TAs: DO NOT ADD YOUR OWN RUBRICS HERE** *Please enter the deduction in the* **Point Adjustment** *field below if submission incorrect* Example:



5.3 **Incorrect instance variables or methods -2** **0** / 0 pts

✔ **– 0 pts** Click here if no submission error

– 2 pts **@TAs: DO NOT ADD YOUR OWN RUBRICS HERE** *Please enter the deduction in the* **Point Adjustment** *field below if instance variables or methods are incorrect* Example:



Autograder Results

[---------- TEST 01 (UOL) ----------] (1/1)

[---------- TEST 02 (UOL) ----------] (1/1)

[---------- TEST 03 (UOL) ----------] (1/1)

[---------- TEST 04 (UOL) ----------] (1/1)

[---------- TEST 05 (UOL) ----------] (1/1)

[---------- TEST 01 (Split) ----------] (0.5/0.5)

[---------- TEST 02 (Split) ----------] (0.5/0.5)

[---------- TEST 03 (Split) ----------] (0.5/0.5)

[---------- TEST 04 (Split) ----------] (0.5/0.5)

[---------- TEST 05 (Split) ----------] (0.5/0.5)

[---------- TEST 06 (Split) ----------] (0.5/0.5)

[---------- TEST 07 (Split) ----------] (0.5/0.5)

[---------- TEST 08 (Split) ----------] (0.5/0.5)

Submitted Files

BillSplitter.java Download

```java
import java.util.ArrayList;
import java.util.List;

public class BillSplitter {

    public static UniqueOrderedList<Integer> split(UniqueOrderedList<Integer> items, int target) {
        CopyableIterator<Integer> iterator = items.iterator();            //Store the list of the results
        List<Integer> result = yourSplit(iterator, target);               //Call your split to find split
        if (result != null) {                                             //If have found a valid solution.
            UniqueOrderedList<Integer> soln = new UniqueOrderedList<>();    //Convert soln to UniqueOrderedList
            for (Integer item : result) {
                soln.add(item);
            }
            return soln;                                                  //Return solution.
        } else {
            return null;                                                  //No solution found return null.
        }
    }

    private static List<Integer> yourSplit(CopyableIterator<Integer> iterator, int target) {
        if (!iterator.hasNext()) {
            if (target == 0) {
                return new ArrayList<>();              //No items left, solution found
            } else {
                return null;                           //No valid solution found
            }
        }

        int currentItem = iterator.next();             // Include the current item in the payment
        List<Integer> withCurrent = yourSplit(iterator.copy(), target - currentItem);

        if (withCurrent != null) {
            withCurrent.add(currentItem);
            return withCurrent;
        }

        List<Integer> withoutCurrent = yourSplit(iterator, target);        //Remove curr and check
        if (withoutCurrent != null) {                                      //Solution without current
            return withoutCurrent;                                         //Return list with current
        } else {
            return null;                                                   // No valid solution found, return null.
        }
    }

}
```

46

UniqueOrderedList.java

Download

```
1
2   public class UniqueOrderedList<T extends Comparable<T>> implements UniqueOrderedListADT<T>,
    SimpleIterable<T>  {
3
4       private int size;
5       private LinearNode<T> head;
6       public UniqueOrderedList() {
7           this.head = null;
8           size = 0;
9       }
10
11      public UniqueOrderedList(T[] data) {
12          this();
13          for (int i = 0; i < data.length; i++) {
14              this.add(data[i]);
15          }
16      }
17
18      public boolean contains (T element) {
19
20          LinearNode<T> curr = this.head;
21          while (curr != null  && curr.getData().compareTo(element) <= 0)  {
22              if (curr.getData().equals(element)) {
23                  return true;
24              } else {
25                  curr = curr.getNext();
26              }
27          }
28          return false;
29
30      }
31
32      public boolean add(T element) {
33
34          if (this.contains(element)) {
35              return false;
36          } else {
37
38              if (this.head == null || this.head.getData().compareTo(element) > 0) {
39                  this.head = new LinearNode<T>(element,head);
40              } else {
41                  LinearNode<T> curr = this.head;
42                  LinearNode<T> prev = null;
43
44                  while (curr != null && curr.getData().compareTo(element) < 0) {
45                      prev = curr;
```

```
46                              curr = curr.getNext();
47                          }
48
49                          prev.setNext(new LinearNode<T> (element, curr));
50                      }
51              }
52              size+=1;
53              return true;
54      }
55
56      public int size() {
57              return this.size;
58      }
59
60      public CopyableIterator<T> iterator() {       //Method to create new iterator
61              return new UOLIterator<>(head);               //UOLiterator new interator object starting at
    head.
62      }
63  }
64
```

UOLIterator.java

Download

```java
import java.util.NoSuchElementException;

public class UOLIterator<T extends Comparable<T>> implements CopyableIterator<T> {
    private LinearNode<T> current; // Pointer to the current element in the list

    public UOLIterator(LinearNode<T> startNode) {                //Constructor
        this.current = startNode;
    }

    @Override
    public boolean hasNext() {                  //Check if there are unvisted elements left.
        return current != null;
    }

    @Override
    public T next() {                           //Returns the next unvisted element in the list.
        if (!hasNext()) {
            throw new NoSuchElementException("iterator empty"); //No more elements
        }

        T data = current.getData();
        current = current.getNext(); // Move the pointer to the next element
        return data;
    }

    @Override
    public CopyableIterator<T> copy() {            //Create copy of iterator.
        return new UOLIterator<>(current);          //New iterator starting at curr.
    }
}


```