Midterm On-Your-Own Practice Questions Selected Answers and Examples

Selected answers are below in red. The file attached on this page contains some coding example solutions. Note that these show only *one possible solution*. There are many other possible coding solutions!

MidtermPracticeExampleFiles.zip (https://ccsf.instructure.com/courses/47904/files/7254627/download?wrap=1) (https://ccsf.instructure.com/courses/47904/files/7254627/download?download_frd=1)

Question 1. Bag Trace

Trace the contents of the bag (implements BagInterface) after each statement:

```
    System.out.println(nameBag.isEmpty()); // true
    nameBag.add("adam");
    nameBag.add("brian");
    nameBag.add("carl");
    nameBag.add("adam");
    nameBag.add("fred");
    nameBag.add("carl");
    nameBag.add("harry");
    nameBag.add("hank");
    System.out.println(nameBag.remove("adam")); // true
    System.out.println(nameBag.getCurrentSize()); // 7
    System.out.println(nameBag.remove("adam")); // true
    System.out.println(nameBag.remove("adam")); // false
    System.out.println(nameBag.remove("ivan")); // false
    System.out.println(nameBag.getCurrentSize()); // 6
    System.out.println(nameBag.getFrequencyOf("carl")); // 2
    System.out.println(nameBag.contains("ivan")); // false
    System.out.println(nameBag.getFrequencyOf("ivan")); // 0
    nameBag.clear();
    System.out.println(nameBag.getCurrentSize()); // 0
```

Question 2. ListInterface Trace

Trace the contents of the list (implements ListInterface) after each statement:

```
    System.out.println(nameList.isEmpty()); // true
    nameList.add("adam");
    nameList.add("brian");
    nameList.add("carl");
    nameList.add("edgar");
    nameList.add(3, "hank");
    nameList.add("lenny");
    nameList.add(1, "mark");
    System.out.println(nameList.getLength()); // 7
    System.out.println(nameList.getEntry(3)); // brian
    System.out.println(nameList.remove(2)); // adam
    System.out.println(nameList.getEntry(2)); // brian
    System.out.println(nameList.remove(1)); // mark
    System.out.println(nameList.remove(2)); // hank
    System.out.println(nameList.getLength()); // 4
    System.out.println(nameList.replace(2, "peter")); // carl
    System.out.println(nameList.getEntry(2)); // peter
    System.out.println(nameList.getEntry(1)); // brian
    System.out.println(nameList.getLength()); // 4
```

Question 3. List Trace

Trace the contents of the list (implements the Java List interface) after each statement:

```
    System.out.println(nameList.isEmpty()); // true
    nameList.add("adam");
    nameList.add("brian");
    nameList.add("carl");
    nameList.add("edgar");
    nameList.add(3, "hank");
    nameList.add("lenny");
    nameList.add(1, "mark");
    System.out.println(nameList.size()); // 7
    System.out.println(nameList.get(3)); // carl
    System.out.println(nameList.remove(2)); // brian
    System.out.println(nameList.get(2)); // carl
    System.out.println(nameList.remove(1)); // mark
    System.out.println(nameList.remove(2)); // hank
    System.out.println(nameList.size()); // 4
    System.out.println(nameList.set(2, "peter")); // edgar
    System.out.println(nameList.get(2)); // peter
    System.out.println(nameList.get(1)); // carl
```

Question 6: Tracing Nodes A

1. What is printed by the code below?
2. Explain what the mystery method does. (Do not repeat what the code does, but explain in words what it returns.) The mystery method looks for the target value in the linked chain and

prints the node *before* the target.

3. Is there anything wrong with how this method is implemented? Will it ever crash? The method skips looking at the first node and will crash on an empty list.

```
list: 4 -> 6 -> 10 -> 12

Node nodeA = list.firstNode.next.next;
Node nodeB = list.firstNode.next;
Node nodeC = list.firstNode.next.next.next;
Node nodeD = nodeC.next;

System.out.println(nodeA.data); // 10
System.out.println(nodeB.data); // 6
System.out.println(nodeC.data); // 12
System.out.println(nodeD.data); // will throw an exception
System.out.println(mystery(list.firstNode, 9)); // prints false
System.out.println(mystery(list.firstNode, 10)); // prints 6 and true

public boolean mystery(Node firstNode, int target) {
   Node currentNode = firstNode;
   if(currentNode.next==null) {
      return false;
   } else {
      Node tmpNode = currentNode;
      currentNode = currentNode.next;
      while(currentNode!=null) {
         if(currentNode.data==target) {
            System.out.println(tmpNode.data);
           return true;
         } else {
            tmpNode = currentNode;
            currentNode = currentNode.next;
         }
      }
      return false;
   }
}
```

Question 7: Tracing Nodes B

What is printed by the pseudocode below?

```
Node firstNode = new Node(3);
firstNode.next = new Node(4);
firstNode.next.next = new Node(6);
firstNode.next.next.next new Node(8);
Node currentNode = firstNode;

print currentNode.data // 3
```

```
    print the chain headed by firstNode // 3 -> 4 -> 6 -> 8

    currentNode = currentNode.next
    print currentNode.data // 4
    print the chain headed by firstNode // 3 -> 4 -> 6 -> 8

    currentNode.data = 7
    print currentNode.data // 7
    print the chain headed by firstNode // 3 -> 7 -> 6 -> 8

    currentNode.next = currentNode.next.next;
    print currentNode.data // 7
    print the chain headed by firstNode // 3 -> 7 -> 8

    firstNode = firstNode.next;
    print currentNode.data // 7
    print the chain headed by firstNode // 7 -> 8
```

Question 10: Comparison

a. When should you use == vs. equals? Use == to compare primitives (ints and chars) and to check for aliases. Use == to check for null. Use equals to compare for logical equivalence. Unless checking for null, typically use equals method to compare objects.

Note that technically == should not be used to compare double or float primitives; because of precision, you should instead compare differences with a threshold (Math.abs(d1-d2)<0.0001, for example).

b. Are there restrictions on what kind of objects can use equals? All objects can use equals because it is inherited from the Object class.

c. When should you use compareTo vs < or >? < and > are for primitives only. compareTo is for objects, but only if the class implements the Comparable interface.

d. Are there restrictions on what kind of objects can use compareTo? The class must implement the Comparable interface.

e. What values are returned from compareTo? A negative if *this* is < parameter, a positive if *this* is > the parameter, a 0 otherwise.

Question 11: Calculating Complexity

c. What are the Big-Os of the following algorithms?

Algorithm 1: // O(n)

```
        statement1;
```

```
if(condition1) {
   statement2;
} else {
   for(int i=0; i<n; i++) {
      statement3;
   }
   statement4;
}
```

Algorithm 2: // O(n^2)

```
i=0;
while(i<n) {
   for(int j=i; j<n; j++) {
      statement1;
   }
   statement2;
   i++;
}
```

Algorithm 3: // O(n^2)

```
for(int i=0; i<n; i++) {
   for(int j=0; j<n; j++) {
      if(condition1) {
         for(int k=0; k<10; k++) {
           statement1;
         }
      } else {
         statement2;
      }
   }
}
```

Algorithm 4: // O(n^2)

```
for(int i=0; i<=n; i++) {
   for(int j=0; j<=n; j++) {
      if(j%2==0) {
         statement1
      }
   }
}
```

d. What are the Big-Os of the following algorithms?

Algortihm A: // O(n^2)

```
        for(int i=0; i<n; i++)
           add i to the beginning of an array-based list
```

Algortihm B: // O(n)

```
        for(int i=0; i<n; i++)
           add i to the end of an array-based list
```

Algortihm C: // O(n)

```
        for(int i=0; i<n; i++)
           add i to the beginning of a linked list with only a head pointer
```

Algortihm D: // $O(n^2)$

```
        for(int i=0; i<n; i++)
           add i to the end of a linked list with only a head pointer
```