CS 220 Software Design II	Spring 2022

In-class Exercises 07

University of Wisconsin - La Crosse *Date: March 29*

1. Given the `SinglyLinkedList` class that appears after Question 2, write the instance method `splitEvenNodes` described below (this method will be located in the `SinglyLinkedList` class). You may choose to use other methods that are typically part of a linked list class (e.g., various `remove` methods), but you might find it easier to implement without them.

```
/**
 * Splits and returns a new list containing only the even nodes from the
 * original list. The original list thus ends up containing only the odd
 * nodes.
 *
 * @return A new list containing only the even nodes removed
 */
```

Solution:

```java
public SinglyLinkedList<E> splitEvenNodes() {
    SinglyLinkedList<E> toReturn = new SinglyLinkedList<>();

    SingleListNode curNode = firstNode;
    SingleListNode endOfToReturn = toReturn.firstNode;
    while(curNode != null && curNode.nextNode != null) {
        endOfToReturn.nextNode = curNode.nextNode;
        curNode.nextNode = curNode.nextNode.nextNode;
        endOfToReturn = endOfToReturn.nextNode;
        endOfToReturn.nextNode = null;
        curNode = curNode.nextNode;
        --size;
        ++endOfToReturn.size;
    }

    return toReturn;
}
```

2. Implement a new inner class iterator `OddIterator`. This is an unusual iterator, in that it will only return the values at the odd indices in the list (usually an iterator returns every value). Your class should contain a constructor, implementations of the `next()` and `hasNext()` methods, and required global attributes.

Solution:

```
private class OddIterator implements Iterator<E> {

    private SingleListNode curNode;

    public OddIterator() {
        curNode = firstNode;
    }

    public boolean hasNext() {
        return curNode.nextNode != null && curNode.nextNode.nextNode != null;
    }

    public E next() {
        if(!hasNext()) {
            throw new NoSuchElementException();
        }

        curNode = curNode.nextNode.nextNode;
        return curNode.value;

}
```

Reference Classes

```
public class SinglyLinkedList<E> {
    private int size;
    private SingleListNode firstNode;

    public SinglyLinkedList() {
        // this list uses sentinel nodes
        firstNode = new SingleListNode(null);
        size = 0;
    }

    private class SingleListNode {
        private E data;
        private SingleListNode nextNode;

        public SingleListNode(E i) { ... }
    }
}
```

3. Conceptually, what advantage does using an iterator give us over using the `get` method for linked lists in order to retrieve all the elements of a linked list?

Solution: Nodes should only be used internally be the list, not externally by the programmer.

4. What advantage does using a sentinel node give us over not using a sentinel node?

Solution: Sentinel nodes produce cleaner code, since they remove the need for one case for handling the beginning of the list, and another case for the end of the list.

5. Implement the `public void clear` method for the `SinglyLinkedList` class on the previous page given the use of sentinel nodes.

```
/**
 * Clears the data from the list.
 * @return void
 */
```

Solution:

```
public void clear() { // 0.5 pt for signature
    firstNode.nextNode = null; // 1.5 pts for clearing the list
    size = 0; // 1 pt for resetting size to 0
}
```