Homework Turnin

Name:	Akshit K Patel
Email:	akshit@uw.edu
Student ID:	1561387
Section:	DC
Course:	CSE 143 16au
Assignment:	a3
Receipt ID:	e06caf3c072806c5233dd483be1c6491

Replacing prior submission from Thu 2016/10/20 10:47pm.

Turnin Successful!

The following file(s) were received:

AssassinManager.java (9586 bytes)

```java
/**
 * @author Akshit Patel
 * @Date 10/17/2016
 * CSE 143D DC
 * TA: Melissa Medsker
 * HW #3 AssassinManager
 */
import java.util.List;// List

/**
 * This class manages the assassination of people for an Assassin game. It
 * provides useful methods to know who is killing whom and who is being stalked,
 * also handles killing.
 */
public class AssassinManager {

    private AssassinNode killFront;// Stalking
    private AssassinNode deadFront;// Dead or killed.

    /**
     * This constructor initializes a new assassin manager over the given list
     * of people.
     *
     * @param names List of people to be managed for one complete game.
     * @throws IllegalArgumentException if the List is empty or null.
     *
     * post: The order of people managed is same as in the List.
     */
    public AssassinManager(List<String> names) {
        if (names == null || names.isEmpty()) {
            throw new IllegalArgumentException();
        }
        // initialize the killRing
        this.killFront = new AssassinNode(names.get(0));
        this.deadFront = null;
        AssassinNode curr = this.killFront;// reference
        // get the names from the list in order.
        for (int i = 1; i < names.size(); i++) {
            // add the person to killring.
            curr.next = new AssassinNode(names.get(i));
            curr.next.killer = curr.name;// set the killer.
            curr = curr.next;
            // NOTE: killer of first person handled by the kill method.
        }
    }

    /**
     * This method prints the names of the people in the killring, one per line,
     * indented by four spaces, as X is stalking Y. If the game is over, then
     * prints X won the game!
     */
    public void printKillRing() {
        // when only one person left: The game is won!
        if (this.killFront.next == null) {
            System.out.println("    " + this.killFront.name + " won the game!");
        } else {
            AssassinNode curr = this.killFront;
            // print the stalking list.
            while (curr != null) {
                System.out.print("    " + curr.name + " is stalking ");
                if (curr.next != null) {
                    System.out.println(curr.next.name);
                    curr = curr.next;
                } else {
                    // last person stalking the first person.
                    System.out.println(this.killFront.name);
                    curr = null;
                }
            }
        }
    }

    /**
     * This method prints the names of the people in the graveyard, one per
     * line, with each line indented by four spaces, as X was killed by Y. (most
     * recently killed first, then next more recently killed, and so on). No
     * output if graveyard is empty.
     *
     */
    public void printGraveyard() {
        AssassinNode curr = this.deadFront;
        while (curr != null) {
            System.out.println(
                    "    " + curr.name + " was killed by " + curr.killer);
            curr = curr.next;
        }
    }

    /**
     * This method helps to know if a person is in the killring. It will ignore
     * case in comparing names.
     *
     * @param name The name of person to check if in the killring.
     * @return true if the given name is in the current killring and false
     * otherwise.
     */
    public boolean killRingContains(String name) {
        AssassinNode curr = this.killFront;
        return this.contains(name, curr);
    }

    /**
     * This method helps to know if a person is in the graveyard. It will ignore
     * case in comparing names.
     *
     * @param name The name of person to check if in the graveyard.
     * @return true if the given name is in the graveyard and false otherwise.
     */
    public boolean graveyardContains(String name) {
        AssassinNode curr = this.deadFront;
        return this.contains(name, curr);
    }

    /**
     * This method helps to know if a person is in the killring or graveyard. It
     * will ignore case in comparing names.
     *
     * @param name The name of person to check if in the graveyard.
     * @param current The reference for killring or graveyard.
     * @return true if the given name is in the killring or graveyard and false
     * otherwise.
     */
    private boolean contains(String name, AssassinNode current) {
        while (current != null) {
            // if name is found return true.
            if (current.name.equalsIgnoreCase(name)) {
                return true;
            }
            current = current.next; // update to next name if not found.
        }
        return false;// false if person not found.
    }

    /**
     * This method helps to know if the game has finished.
     *
     * @return true if game over or false otherwise.
     */
    public boolean isGameOver() {
        return this.killFront.next == null;
    }

    /**
     * This method helps to get the name of winner in the game played.
     *
     * @return String name of the winner if game over else null.
     */
    public String winner() {
        if (this.isGameOver()) {
            return this.killFront.name;// winner.
        }
        return null;
    }

    /**
     * This method records the assassination of the person with the given name,
     * transferring the person from the killring to the front of the graveyard.
     * This operation does not change the relative order of the killring (i.e.
     * the links of who is killing whom stays the same other than the person who
     * is being killed). This method ignores case in comparing names.
     *
     * @param name The name of person to be killed from the killring.
     * @throws IllegalStateException if the game is finished.
     * @throws IllegalArgumentException if the person is not in killring.
     *
     * NOTE: IllegalStateException takes precedence if both conditions i.e. game
     * is finished & person not found in killring are true.
     */
    public void kill(String name) {
        if (this.isGameOver()) {
            throw new IllegalStateException();
        }
        if (!this.killRingContains(name)) {
            throw new IllegalArgumentException();
        }
        AssassinNode curr = this.killFront;
        AssassinNode prev = curr;// previous person reference.
        // if the first person in the killRing is to be killed update the
        // killring reference and get/set the killer.
        if (curr.name.equalsIgnoreCase(name)) {
            this.killFront = curr.next;// go the next alive.
            // get the killer
            while (prev.next != null) {
                prev = prev.next;
            }
            curr.killer = prev.name;// set killer name.
            this.graveYardUpdate(curr);// add to graveyard.
        } else {
            while (curr != null) {
                // if name found, send the person to graveyard, update the
                // killer of the next person alive
                if (curr.name.equalsIgnoreCase(name)) {
                    String previousName = prev.name;
                    // if the person killed is last in killring then
                    // update the killer of the first.
                    if (curr.next == null) {
                        this.killFront.killer = previousName;
                    } else {
                        // update the killer of the next person to the one
                        // before in killring.
                        curr.next.killer = previousName;
                    }
                    prev.next = curr.next; // kill the person from killRing
                    this.graveYardUpdate(curr);// add to graveyard.
                    curr = null;

                } else {
                    // goto next if not found.
                    prev = curr;
                    curr = curr.next;
                }
            }
        }
    }

    /**
     * This method updates the graveyard by adding the dead and maintaining the
     * order in which the person was killed i.e. how recently were they killed.
     *
     * @param current reference to the person killed in killring.
     */
    private void graveYardUpdate(AssassinNode current) {
        current.next = this.deadFront; // add person to graveyard.
        this.deadFront = current;// update graveyard reference.
    }

    //////// DO NOT MODIFY AssassinNode.  You will lose points if you do. ////////
    /**
     * Each AssassinNode object represents a single node in a linked list
     * for a game of Assassin.
     */
    private static class AssassinNode {
        public final String name;  // this person's name
        public String killer;      // name of who killed this person (null if alive)
        public AssassinNode next;  // next node in the list (null if none)

        /**
         * Constructs a new node to store the given name and no next node.
         */
        public AssassinNode(String name) {
            this(name, null);
        }

        /**
         * Constructs a new node to store the given name and a reference
         * to the given next node.
         */
        public AssassinNode(String name, AssassinNode next) {
            this.name = name;
            this.killer = null;
            this.next = next;
        }
    }
}
```