Homework Turnin

Name:	Xuqing Wu
Account:	xw88 (xw88@uw.edu)
Student ID:	1933202
Section:	AD
Course:	CSE 143 20wi
Assignment:	a3
Receipt ID:	2fc051b2cfd582f5cea6c77c38506d7d

Turnin script completed with output:

Turnin Successful!

The following file(s) were received:

AssassinManager.java (4436 bytes, sha256: 1be1a378b153ae3d9c49354c230b6a34)

```
 1. // Xuqing Wu
 2. // 1/29/2020
 3. // CSE143
 4. // TA: Eric Fan
 5. // Assignment #3
 6. //
 7. // Class AssassinManager allows a client to manage a game
 8. // of assassin through keeping track of who is stalking whom
 9. // and the history of who killed whom
10.
11. import java.util.*;
12.
13. public class AssassinManager {
14.    private AssassinNode alive;   //stores names of people that still alive
15.    private AssassinNode dead;    //stores names of people that are killed
16.
17.    //pre: the list of name passed is not empty
18.    //(throw IllegalArgumentException if not)
19.    //post: add the names from the list into the
20.    //kill ring in the same order in which they
21.    //appear in the list
22.    public AssassinManager(List<String> names) {
23.       if(names == null) {
24.          throw new IllegalArgumentException();
25.       }
26.       for(int i = names.size() - 1; i >= 0; i--) {
27.          if(alive == null) {
28.             alive = new AssassinNode(names.get(i));
29.          }
30.          else {
31.             alive = new AssassinNode(names.get(i), alive);
32.          }
33.       }
34.    }
35.
36.    //post: print the names of the people in the kill ring
37.    //in the form <name> is stalking <name>"
38.    public void printKillRing() {
39.       AssassinNode current = alive;
40.       while(current.next != null) {
41.          System.out.println("    " + current.name + " is stalking " + current.next.name);
```

```
42.            current = current.next;
43.         }
44.         System.out.println("    " + current.name + " is stalking " + alive.name);
45.      }
46.
47.      //post: print the names of the people in the graveyard
48.      //in the form <name> was killed by <name> in reverse
49.      //kill order, produce no output if the graveyard is empty
50.      public void printGraveyard() {
51.         AssassinNode current = dead;
52.         while(current != null) {
53.            System.out.println("    " + current.name + " was killed by " + current.killer);
54.            current = current.next;
55.         }
56.      }
57.
58.      //post: return true if the given name is in the current
59.      //kill ring and return false otherwise. Ignore case in
60.      //comparing names.
61.      public boolean killRingContains(String name) {
62.         return contains(name, alive);
63.      }
64.
65.      //post: return true if the given name is in the current
66.      //graveyard and return false otherwise. Ignore case in
67.      //comparing names.
68.      public boolean graveyardContains(String name) {
69.         return contains(name, dead);
70.      }
71.
72.      //post: The method in order to reduce redundancy of the
73.      //previous two methods.
74.      private boolean contains(String name, AssassinNode in) {
75.         AssassinNode current = in;
76.         while(current != null) {
77.            if(current.name.equalsIgnoreCase(name)) {
78.               return true;
79.            }
80.            current = current.next;
81.         }
82.         return false;
83.      }
84.
85.      //post: return true if the game is over(kill ring has just
86.      //one person in it)and return false otherwise.
87.      public boolean gameOver() {
88.         return alive.next == null;
89.      }
90.
91.      //post: return the name of the winner of the game. Return
92.      //null if the game is not over.
93.      public String winner() {
94.         if(gameOver()) {
95.            return alive.name;
96.         }
97.         else {
98.            return null;
99.         }
100.     }
101.
102.     //pre: the given name is part of the current kill ring
103.     //(throw IllegalArgumentException if not)
104.     //game is not over(throw IllegalStateException if not)
105.     //post: record the killing of the person with the given name,
106.     //transferring the person from the kill ring to the graveyard.
107.     //Ignore case in comparing names.
108.     public void kill(String name) {
109.        if(!killRingContains(name)) {
110.           throw new IllegalArgumentException();
111.        }
112.        if(gameOver()) {
113.           throw new IllegalStateException();
114.        }
115.        AssassinNode current = alive;
116.        AssassinNode killed = null;
117.        if(alive.name.equalsIgnoreCase(name)) {
```

```
118.            killed = alive;
119.            while(current.next != null) {
120.               current = current.next;
121.            }
122.            killed.killer = current.name;
123.            alive = alive.next;
124.         }
125.         else {
126.            while(!current.next.name.equalsIgnoreCase(name)) {
127.               current = current.next;
128.            }
129.            killed = current.next;
130.            killed.killer = current.name;
131.            if(current.next.next == null) {
132.               current.next = null;
133.            }
134.            else {
135.               current.next = current.next.next;
136.            }
137.         }
138.         killed.next = dead;
139.         dead = killed;
140.      }
141.   }
142.
```