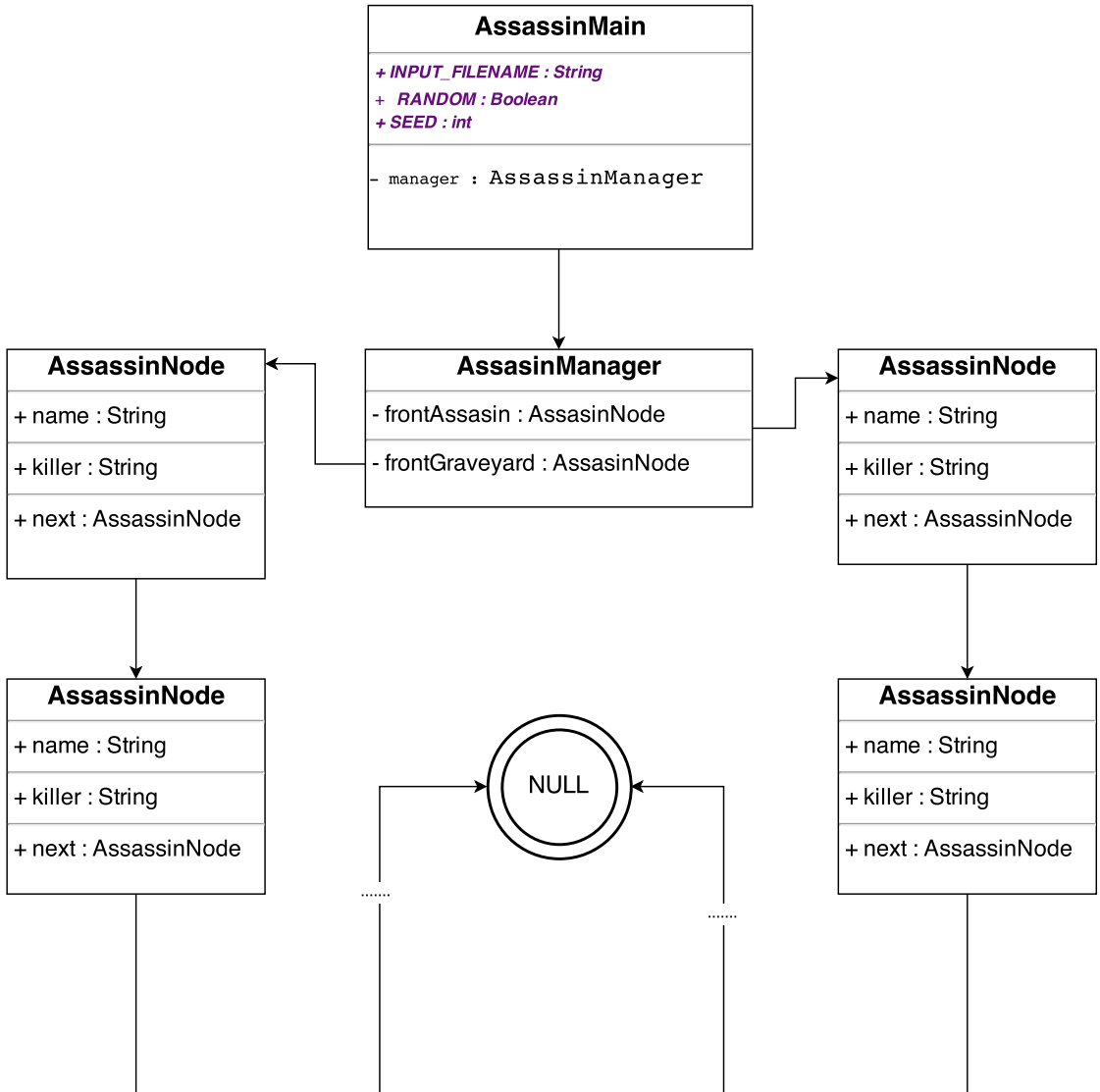


HAI H NGUYEN
SID: 844 920 234

AssassinManager
- frontAssassin : AssassinNode - frontGraveyard : AssassinNode
+ AssassinManager (name : ArrayList <String>) + printKillRing () + printGraveyard (out : PrintStream) + killRingContains (name : String) : boolean + graveyardContains (name : String) : boolean + isGameOver () : boolean + winner () : String + kill (name : String) - stalkKillStatus (villian : String, hero: String, isStalk : boolean) - getKiller (victim : String) : AssassinNode - purgatory (victim: String, killer : String)

HAI H NGUYEN
SID: 844 920 234



```

1 import java.util.ArrayList;
2
3 /**
4  * This object Manages AssassinNodes
5  *
6  * <ul>
7  * <li> Name: AssassinManager.java
8  * <li> Description: Assassin Manager
9  * <li> Class: Java 145
10 * <li> Instructor: Ken Hang
11 * <li> Date: Feb 13 2015
12 * </ul>
13 *
14 * @author Hai H Nguyen (Bill)
15 * @version Winter 2015
16 */
17 public class AssassinManager {
18
19     private AssassinNode frontAssassin;
20
21     private AssassinNode frontGraveyard;
22
23     /**
24      * Constructor which Initialize the Assassin Linked list
25      * @param name      A List of Assassin's Name
26      */
27     public AssassinManager (ArrayList<String> name){
28         if(name != null && !name.isEmpty()) {
29             for (int i = name.size() - 1; i >= 0; --i){
30                 frontAssassin = new AssassinNode(name.get(i), frontAssassin);
31             }
32         } else {
33             throw new IllegalArgumentException("Name List is Empty or Null");
34         }
35     }
36
37     /**
38      * Show a report of who stalking who.
39      */
40     public void printKillRing() {
41         if (!isGameOver()) {
42             AssassinNode assassinPtr = frontAssassin;
43
44             while (assassinPtr != null && assassinPtr.next != null) {
45                 stalkKillStatus(assassinPtr.name, assassinPtr.next.name, true);
46
47                 assassinPtr = assassinPtr.next;
48             }
49
50             stalkKillStatus(assassinPtr.name, frontAssassin.name, true);
51         }
52     }
53
54     /**
55      * Print who Stalks or Kills who
56      * @param villain      Name of the Stalker or Dead man
57      * @param hero          Name of the Stalked or Killer
58      * @param isStalk       Is this Stalking or Killing?
59      */
60     private void stalkKillStatus(String villain, String hero, boolean isStalk){
61         System.out.println( " " + villain +
62             ( isStalk ? (" is stalking ") : (" was killed by ")) + hero);
63     }
64
65     /**
66      * Show a report of who was killed by who.
67      */
68     public void printGraveyard(){

```

```

69         if (frontGraveyard != null) {
70             AssassinNode gravePtr = frontGraveyard;
71
72             while (gravePtr != null) {
73                 stalkKillStatus(gravePtr.name, gravePtr.killer, false);
74
75                 gravePtr = (gravePtr.next != null) ? gravePtr.next : null;
76             }
77         }
78     }
79
80     /**
81     * Search for killers
82     * @param name    Name of Killer
83     * @return        True if found, False otherwise
84     */
85     public boolean killRingContains(String name){
86         AssassinNode assassinPtr = frontAssassin;
87
88         while (assassinPtr != null && assassinPtr.name != null){
89             if (assassinPtr.name.equalsIgnoreCase(name)){
90                 return true;
91             } else {
92                 assassinPtr = assassinPtr.next;
93             }
94         }
95
96         return false;
97     }
98
99     /**
100    * Search for victim in the Graveyard.
101    * @param name    Name of Victim
102    * @return        True if found, False otherwise
103    */
104    public boolean graveyardContains(String name) {
105        if (frontGraveyard != null){
106            AssassinNode gravePtr = frontGraveyard;
107
108            while (gravePtr != null) {
109                if (gravePtr.name.equalsIgnoreCase(name)){
110                    return true;
111                }else {
112                    gravePtr = (gravePtr.next != null) ? gravePtr.next : null;
113                }
114            }
115
116            return false;
117        } else {
118            return false;
119        }
120    }
121
122    /**
123    * @return        True if all but one is alive, False otherwise.
124    */
125    public boolean isGameOver(){
126        return (frontAssassin.next == null);
127    }
128
129    /**
130    * @return        Name of the Last man Standing
131    */
132    public String winner(){
133        return isGameOver() ?
134            frontAssassin.name : null;
135    }
136

```

```

137  /**
138   * Search for name, and transfer it to the Graveyard
139   * If none was found or Game is over, Throw Exceptions
140   * @param name          Name of the Victim
141   */
142  public void kill (String name){
143      if (isGameOver()) {
144          throw new IllegalStateException("Game Over");
145      } else {
146          AssassinNode killerPtr = getKiller(name);
147
148          if (frontAssassin.name.equalsIgnoreCase(name)) {
149              purgatory(frontAssassin.name, killerPtr.name);
150
151              frontAssassin = frontAssassin.next;
152          } else if (killerPtr == null || killerPtr.next == null) {
153              throw new IllegalArgumentException("Nobody named " + name);
154          } else {
155              purgatory(killerPtr.next.name, killerPtr.name);
156
157              killerPtr.next = killerPtr.next.next;
158          }
159      }
160  }
161
162  /**
163   * Ignore the front guy and search for the killer
164   * @param victim          Name of the Victim
165   * @return                A pointer to the Killer
166   */
167  private AssassinNode getKiller(String victim){
168      AssassinNode ptr = frontAssassin;
169
170      while ( !(ptr == null) &&
171              !(ptr.next == null) &&
172              !(ptr.next.name.equalsIgnoreCase(victim))) {
173          ptr = ptr.next;
174      }
175
176      return ptr;
177  }
178
179  /**
180   * Moment between Life and Death
181   * @param victim          Name of The Victim
182   * @param killer          Name of The Killer
183   */
184  private void purgatory (String victim, String killer) {
185      frontGraveyard = new AssassinNode(victim, frontGraveyard);
186
187      frontGraveyard.killer = killer;
188  }
189 }

```