```java
import java.util.ArrayList;

/**
 * This object Manages AssassinNodes
 *
 * <ul>
 * <li> Name: AssassinManager.java
 * <li> Description: Assassin Manager
 * <li> Class: Java 145
 * <li> Instructor: Ken Hang
 * <li> Date: Feb 13 2015
 * </ul>
 *
 * @author  Hai H Nguyen (Bill)
 * @version Winter 2015
 */
public class AssassinManager {

    private AssassinNode frontAssassin;

    private AssassinNode frontGraveyard;

    /**
     * Constructor which Initialize the Assassin Linked list
     * @param name      A List of Assassin's Name
     */
    public AssassinManager (ArrayList<String> name){
        if(name != null && !name.isEmpty()) {
            for (int i = name.size() - 1; i >= 0; --i){
                frontAssassin = new AssassinNode(name.get(i), frontAssassin);
            }
        } else {
            throw new IllegalArgumentException("Name List is Empty or Null");
        }
    }

    /**
     * Show a report of who stalking who.
     */
    public void printKillRing() {
        if (!isGameOver()) {
            AssassinNode assassinPtr = frontAssassin;

            while (assassinPtr != null && assassinPtr.next != null) {
                stalkKillStatus(assassinPtr.name, assassinPtr.next.name, true);

                assassinPtr = assassinPtr.next;
            }

            stalkKillStatus(assassinPtr.name, frontAssassin.name, true);
        }
    }

    /**
     * Print who Stalks or Kills who
     * @param villain       Name of the Stalker or Dead man
     * @param hero          Name of the Stalked or Killer
     * @param isStalk       Is this Stalking or Killing?
     */
    private void stalkKillStatus(String villain, String hero, boolean isStalk){
        System.out.println( "  " + villain +
                ( isStalk ? (" is stalking ") : (" was killed by ")) + hero);
    }

    /**
     * Show a report of who was killed by who.
     */
    public void printGraveyard(){
```

```
        if (frontGraveyard != null) {
            AssassinNode gravePtr = frontGraveyard;

            while (gravePtr != null) {
                stalkKillStatus(gravePtr.name, gravePtr.killer, false);

                gravePtr = (gravePtr.next != null) ? gravePtr.next : null;
            }
        }
    }

    /**
     * Search for killers
     * @param name  Name of Killer
     * @return      True if found, False otherwise
     */
    public boolean killRingContains(String name){
        AssassinNode assassinPtr = frontAssassin;

        while (assassinPtr != null && assassinPtr.name != null){
            if (assassinPtr.name.equalsIgnoreCase(name)){
                return true;
            } else {
                assassinPtr = assassinPtr.next;
            }
        }

        return false;
    }

    /**
     * Search for victim in the Graveyard.
     * @param name  Name of Victim
     * @return      True if found, False otherwise
     */
    public boolean graveyardContains(String name) {
        if (frontGraveyard != null){
            AssassinNode gravePtr = frontGraveyard;

            while (gravePtr != null) {
                if (gravePtr.name.equalsIgnoreCase(name)){
                    return true;
                }else {
                    gravePtr = (gravePtr.next != null) ? gravePtr.next : null;
                }
            }

            return false;
        } else {
            return false;
        }
    }

    /**
     * @return      True if all but one is alive, False otherwise.
     */
    public boolean isGameOver(){
        return (frontAssassin.next == null);
    }

    /**
     * @return      Name of the Last man Standing
     */
    public String winner(){
        return isGameOver() ?
                frontAssassin.name : null;
    }

```

```
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 }
```