This program focuses on implementing a linked list. You will turn in a file named AssassinManager.java as well as others specified below. You must get copies of these support files AssassinNode.java, AssassinMain.java, and names.txt from the course web site; place them in the same folder as your class. See Ch.15.1,15.2, 16.1,16.2 for material on linked lists.

For **all work**, **you must**:

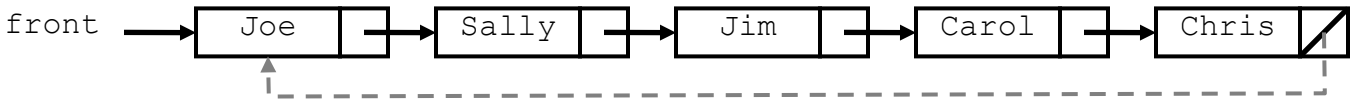
- **follow all style guidelines in Appendix I** and **F**. Do **NOT** create a java project, package or .jar
- spell all names **exactly** as they are spelled in these instructions so my tester will work on your class
- provide **both electronic and physical/paper deliverables** (See below).
- For each class you create you must provide at least these:

I. A toString method that returns a String with all (properly labeled) instance and static variables as well as current values of derived values (such as average, max, min, etc.)
II. Default constructor() with no arguments and a constructor with values for each instance variable)
III. Accessor (get) and mutator (set) methods for every instance variable
IV. Assert statements in a main that test each of the above constructors and methods.

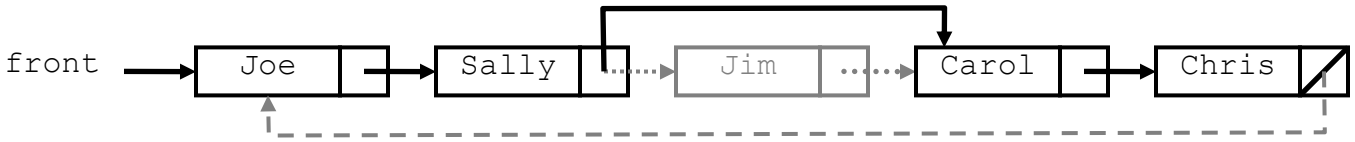
Program Description:

"Assassin" is a real-life game often played on college campuses. Each person playing has a particular target that he/she is "so into," or trying to "kill." Generally "killing" a person means finding them on campus in public and acting on them in some way, such as saying, "You're my dead," or squirting them with a squirt gun, or touching them. One of the things that make the game more interesting to play in real life is that initially each person knows only who they are trying to kill; they don't know who is trying to kill them, nor do they know whom the other people are trying to kill.

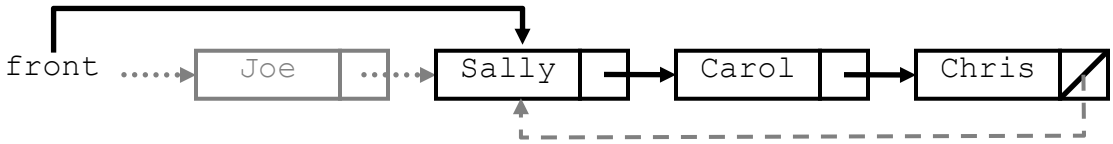
The game of Assassin is played as follows: You start out with a group of people who want to play the game. For example, let's say that there are five people playing named Carol, Chris, Jim, Joe, Sally. A circular chain of killing targets (called the "kill ring" in this program) is randomly established. For example, we might decide Joe should try to kill Sally, Sally should kill Jim, Jim should kill Carol, Carol should kill Chris, and Chris should kill Joe. (In the actual linked list that implements this kill ring, Chris's next reference would be null. But conceptually we can think of it as though the next person after Chris is Joe, the front person in the list.) Here is a picture of this "kill ring":



When someone is killed, the links need to be changed to "skip" that person. For example, suppose that Jim is killed by Sally. Sally needs a new target, so she becomes stalking Jim's target: Carol. The kill ring becomes:



If the first person in the kill ring is killed, the front of the list must adjust. If Chris killes Joe, the list becomes:



You will **write a class AssassinManager** that keeps track of who is stalking whom and the history of who killed whom. You will maintain two linked lists: a list of people currently attracted (the "kill ring") and a list of those who've been killed (the "graveyard"). As people are killed, you will move them from the kill ring to the graveyard by rearranging links between nodes. The game ends when only one node remains in the kill ring, representing the winner.

A client program has been written for you called AssassinMain. It reads a file of names, shuffles the names, and constructs an object of your class AssassinManager. This main program then asks the user for the names of each victim to kill until there is just one player left (at which point the game is over and the last remaining player wins). AssassinMain calls methods of the AssassinManager to carry out the tasks involved in administering the game.

Sample Log of Execution (user input underlined):

Your program must exactly reproduce the format and general behavior demonstrated in this log, although you may not exactly recreate this scenario because of the shuffling of the names that AssassinMain performs.

```
Current kill ring:
  Erica Kane is stalking Ruth Martin
  Ruth Martin is stalking Jackson Montgomery
  Jackson Montgomery is stalking Bobby Warner
  Bobby Warner is stalking Joe Martin
  Joe Martin is stalking Anita Santos
  Anita Santos is stalking Tad Martin
  Tad Martin is stalking Phoebe Wallingford
  Phoebe Wallingford is stalking Erica Kane
Current graveyard:

next target? Ruth Martin

Current kill ring:
  Erica Kane is stalking Jackson Montgomery
  Jackson Montgomery is stalking Bobby Warner
  Bobby Warner is stalking Joe Martin
  Joe Martin is stalking Anita Santos
  Anita Santos is stalking Tad Martin
  Tad Martin is stalking Phoebe Wallingford
  Phoebe Wallingford is stalking Erica Kane
Current graveyard:
  Ruth Martin was killed by Erica Kane

next target? Ruth Martin
Ruth Martin is already killed.

Current kill ring:
  Erica Kane is stalking Jackson Montgomery
  Jackson Montgomery is stalking Bobby Warner
  Bobby Warner is stalking Joe Martin
  Joe Martin is stalking Anita Santos
  Anita Santos is stalking Tad Martin
  Tad Martin is stalking Phoebe Wallingford
  Phoebe Wallingford is stalking Erica Kane
Current graveyard:
  Ruth Martin was killed by Erica Kane

next target? bobby warner

Current kill ring:
  Erica Kane is stalking Jackson Montgomery
  Jackson Montgomery is stalking Joe Martin
  Joe Martin is stalking Anita Santos
  Anita Santos is stalking Tad Martin
  Tad Martin is stalking Phoebe Wallingford
  Phoebe Wallingford is stalking Erica Kane
Current graveyard:
  Bobby Warner was killed by Jackson Montgomery
  Ruth Martin was killed by Erica Kane

next target? ERICa kaNE

Current kill ring:
  Jackson Montgomery is stalking Joe Martin
  Joe Martin is stalking Anita Santos
  Anita Santos is stalking Tad Martin
  Tad Martin is stalking Phoebe Wallingford
  Phoebe Wallingford is stalking Jackson Montgomery
Current graveyard:
  Erica Kane was killed by Phoebe Wallingford
  Bobby Warner was killed by Jackson Montgomery
  Ruth Martin was killed by Erica Kane

next target? ANITA SANTOS

(continued)

Current kill ring:
  Jackson Montgomery is stalking Joe Martin
  Joe Martin is stalking Tad Martin
  Tad Martin is stalking Phoebe Wallingford
  Phoebe Wallingford is stalking Jackson Montgomery
Current graveyard:
  Anita Santos was killed by Joe Martin
  Erica Kane was killed by Phoebe Wallingford
  Bobby Warner was killed by Jackson Montgomery
  Ruth Martin was killed by Erica Kane

next target? phoebe wallingford

Current kill ring:
  Jackson Montgomery is stalking Joe Martin
  Joe Martin is stalking Tad Martin
  Tad Martin is stalking Jackson Montgomery
Current graveyard:
  Phoebe Wallingford was killed by Tad Martin
  Anita Santos was killed by Joe Martin
  Erica Kane was killed by Phoebe Wallingford
  Bobby Warner was killed by Jackson Montgomery
  Ruth Martin was killed by Erica Kane

next target? Barack Obama
Unknown person.

Current kill ring:
  Jackson Montgomery is stalking Joe Martin
  Joe Martin is stalking Tad Martin
  Tad Martin is stalking Jackson Montgomery
Current graveyard:
  Phoebe Wallingford was killed by Tad Martin
  Anita Santos was killed by Joe Martin
  Erica Kane was killed by Phoebe Wallingford
  Bobby Warner was killed by Jackson Montgomery
  Ruth Martin was killed by Erica Kane

next target? Joe Martin

Current kill ring:
  Jackson Montgomery is stalking Tad Martin
  Tad Martin is stalking Jackson Montgomery
Current graveyard:
  Joe Martin was killed by Jackson Montgomery
  Phoebe Wallingford was killed by Tad Martin
  Anita Santos was killed by Joe Martin
  Erica Kane was killed by Phoebe Wallingford
  Bobby Warner was killed by Jackson Montgomery
  Ruth Martin was killed by Erica Kane

next target? jackson montgomery

Game was won by Tad Martin
Final graveyard is as follows:
  Jackson Montgomery was killed by Tad Martin
  Joe Martin was killed by Jackson Montgomery
  Phoebe Wallingford was killed by Tad Martin
  Anita Santos was killed by Joe Martin
  Erica Kane was killed by Phoebe Wallingford
  Bobby Warner was killed by Jackson Montgomery
  Ruth Martin was killed by Erica Kane
```

Implementation Details:

You must use the node class AssassinNode (provided on the course website) which has the following implementation:

```
public class AssassinNode {
   public 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

   public AssassinNode(String name) { ... }
   public AssassinNode(String name, AssassinNode next) { ... }
}
```

This assignment specifies exactly what fields you can/must have in your AssassinManager class. You must have exactly the following two fields; you are not allowed to have any others:

1. a reference to the front node of the kill ring & 2. a reference to the front node of the graveyard (null if empty)

Implementation Details (continued): Your class should have the following public methods.

public AssassinManager(List<String> names)

In this constructor you should initialize a new Assassin manager over the given list of people. Your constructor must **not** save the List<String> itself as a field, nor modify the list; but instead it should build your own kill ring of linked nodes that contains these names in the same order. For example, if the list contains ["John", "Sally", "Fred"], the initial kill ring should represent that John is stalking Sally who is stalking Fred who is stalking John (in that order). You may generally assume that the names are non-empty, non-null strings and that there are no duplicates.

You should throw an IllegalArgumentException if the list is null or has a size of 0.

s

public void printKillRing()

In this method you should print the names of the people in the kill ring, one per line, indented by two spaces, as "**name** is stalking **name**". The behavior is unspecified if the game is over. For the names on the first page, the initial output is:

```
  Joe is stalking Sally
  Sally is stalking Jim
  Jim is stalking Carol
  Carol is stalking Chris
  Chris is stalking Joe
```

s

public void printGraveyard()

In this method you should print the names of the people in the graveyard, one per line, with each line indented by two spaces, with output of the form "**name** was killed by **name**". It should print the names in the opposite of the order in which they were killed (most recently killed first, then next more recently killed, and so on). It should produce no output if the graveyard is empty. From the previous names, if Jim is killed, then Chris, then Carol, the output is:

```
  Carol was killed by Sally
  Chris was killed by Carol
  Jim was killed by Sally
```

s

public boolean killRingContains(String name)

In this method you should return true if the given name is in the current kill ring and false otherwise. It should ignore case in comparing names; for example, if passed "salLY", it should match a node with a name of "Sally".

s

public boolean graveyardContains(String name)

In this method you should return true if the given name is in the current graveyard and false otherwise. It should ignore case in comparing names; for example, if passed "CaRoL", it should match a node with a name of "Carol".

s

public boolean isGameOver()

In this method you should return true if the game is over (i.e., if the kill ring has just one person) and false otherwise.

s

public String winner()

In this method you should return the name of the winner of the game, or null if the game is not over.

s

public void kill(String name)

In this method you should record the killing of the person with the given name, transferring the person from the kill ring to the front of the graveyard. This operation should not change the relative order of the kill ring (i.e., the links of who is "into" whom should stay the same other than the person who is being killed/removed). Ignore case in comparing names. A node remembers who killed the person in its killer field. It is your code's responsibility to set that field's value.

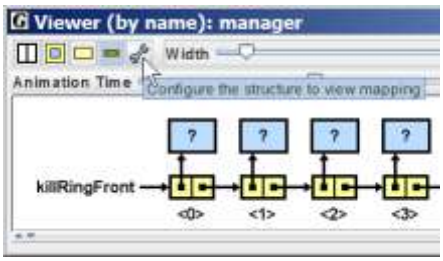
You should throw an IllegalStateException if the game is over, or an IllegalArgumentException if the given name is not part of the kill ring (if both of these conditions are true, the IllegalStateException takes precedence).

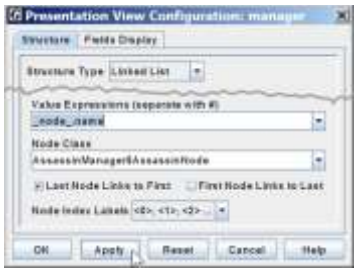
The kill method is the hardest one, so write it last. Use the jGRASP debugger and println statements liberally to debug problems in your code. You will likely have a lot of NullPointerException errors, infinite loops, etc. and will have a very hard time tracking them down unless you are comfortable with debugging techniques and jGRASP. For full credit, every method's runtime should be at worst O(*N*), where *N* is the number of people in your linked lists.

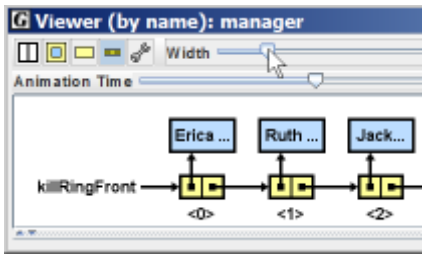
jGRASP's Debugger (jGRASP gets updated. These notes may be outdated. See internet):

In jGRASP's debugger you can use a structure viewer to see what your list looks like by dragging one of your fields from the debug window outside the window. By default the viewer won't show you the name in each node (it'll show a "?" mark instead). Fix this by clicking the wrench icon, then in the "Value Expressions" box, type: **_node_.name**

Click OK, and you should see the names in the nodes. You can also drag the Width scrollbar to see the names better.







Other Constraints and Tips:

This is meant to be an exercise in linked list manipulation. As a result, **you must adhere to the following rules**:

1. You must use our AssassinNode class for your lists. You are **not** allowed to modify it.
2. You may **not** construct any arrays, ArrayLists, LinkedLists, stacks, queues, sets, maps, or other data structures;
3. you **must** use linked nodes.
4. You must **not** modify the list of Strings passed to your constructor.
5. If there are *N* names in the list of Strings passed to your constructor, it must create exactly *N* new AssassinNode objects.
6. As people are killed, you **must** move their node from the kill ring to the graveyard by changing references, without creating any new node objects.

Your constructor will create the initial kill ring of nodes, and then your class may not create any more nodes for the rest of the program. You can declare as many local variables of type AssassinNode (like current from lecture) as you like. AssassinNode variables are not node objects and therefore do not count against the limit of *n* nodes.

You should write some of your own testing code. AssassinMain requires every method to be written in order to compile, and it never generates any of the exceptions you have to handle, so it is not exhaustive.

A word of caution: Some students try to store the kill ring in a "circular" linked list, with the list's final element storing a next reference back to the front element. But we discourage you from implementing the program in this way; you should follow the normal convention of having null in the next field of the last node. Most novices find it difficult to work with a circular list; and end up with infinite loops or other bugs. There is no need to use a circular list, you can get back to the front via the fields of your AssassinManager.

Do not use recursion on this program. For reference, a solution is around 130 lines long.

You must submit all physical and electronic Deliverables. See Canvas for due date.

Electronic:

1. Copies of **all source code and data files** used by your code
2. You **must include** thorough **test plan** that fully exercises all required features showing input and expected results in the rtf file containing your test runs. You must do overall, system/problem testing **in addition to the unit tests done with asserts in your class file**(s).
   - 2.1. Show input to use and expected output **that you calculated** (a spreadsheet helps for this).
   - 2.2. List at least one test for **each possible type of successful action** your code could take and least one test for **each possible type of error action** your code could take.
   - 2.3. Remember to test empty lines/files for expected inputs, non-numeric data and incomplete numeric input
   - 2.4. Include explanations of any discrepancies or reasons why the test could fail
3. Copies of at least 3 test runs results; you may Include more in your rtf file,
4. The **.html and other files generated by processing Javadoc comments (select the Doc button** in JGrasp).
5. Compress all of the items above, together in **one .zip** file. (Not .rar nor jar)
6. Include your name and the program number in the file name, for example: EdHyde_p42.zip.
7. Submit this single **zip** file (with code, data, test plan, test runs, javdoc output) on **Canvas**

Physical: Staple all paper pages **in the following order** (print on both sides if possible to save paper).

1. All pages of **these instructions**, with your **names clearly printed** on it, **as the top sheet**.
2. A printed copy of **Java_Style_Grading.pdf** from Canvas Files (that you should use to check your code as will I).