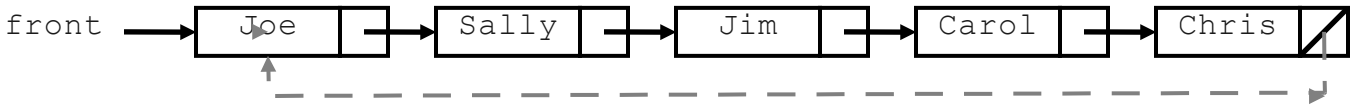
CSE 143, Winter 2011
Programming Assignment #4: Affection (40 points)
Due Thursday, February 3, 2010, 11:30 PM

This program focuses on implementing a linked list. Turn in a file named `AffectionManager.java` from the Homework section of the course web site. You will need support files `AffectionNode.java`, `AffectionMain.java`, and `names.txt` from the Homework section of the course web site; place them in the same folder as your class.

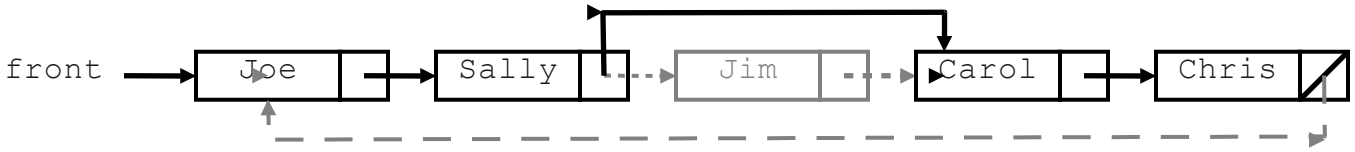
Program Description:

"Affection" is a game often played on college campuses. Each person playing has a particular target that he/she is "so into," or trying to "kiss." Generally "kissing" a person means finding them on campus in public and acting on them in some way, such as saying, "You're my baby," or giving them a nice present such as a cookie, or hugging them. One of the things that makes the game more interesting to play in real life is that initially each person knows only who they are trying to kiss; they don't know who is trying to kiss them, nor do they know whom the other people are trying to kiss.

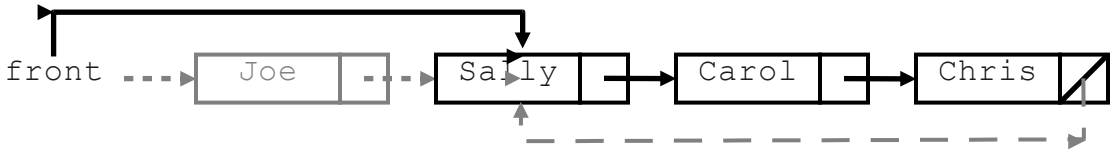
The game of Affection is played as follows: You start out with a group of people who want to play the game. For example, let's say that there are five people playing named Carol, Chris, Jim, Joe, Sally. A circular chain of kissing targets (called the "kiss ring" in this program) is randomly established. For example, we might decide Joe should try to kiss Sally, Sally should kiss Jim, Jim should kiss Carol, Carol should kiss Chris, and Chris should kiss Joe. (In the actual linked list that implements this kiss 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 "kiss ring":



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



If the first person in the kiss ring is kissed, the front of the list must adjust. If Chris kisses Joe, the list becomes:



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

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

Sample Log of Execution (user input underlined):

Your program should 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 `AffectionMain` performs.

```
Current kiss ring:
  Erica Kane is so into Ruth Martin
  Ruth Martin is so into Jackson Montgomery
  Jackson Montgomery is so into Bobby Warner
  Bobby Warner is so into Joe Martin
  Joe Martin is so into Anita Santos
  Anita Santos is so into Tad Martin
  Tad Martin is so into Phoebe Wallingford
  Phoebe Wallingford is so into Erica Kane
Current schoolyard:

next sweetheart? Ruth Martin

Current kiss ring:
  Erica Kane is so into Jackson Montgomery
  Jackson Montgomery is so into Bobby Warner
  Bobby Warner is so into Joe Martin
  Joe Martin is so into Anita Santos
  Anita Santos is so into Tad Martin
  Tad Martin is so into Phoebe Wallingford
  Phoebe Wallingford is so into Erica Kane
Current schoolyard:
  Ruth Martin was kissed by Erica Kane

next sweetheart? Ruth Martin
Ruth Martin is already kissed.

Current kiss ring:
  Erica Kane is so into Jackson Montgomery
  Jackson Montgomery is so into Bobby Warner
  Bobby Warner is so into Joe Martin
  Joe Martin is so into Anita Santos
  Anita Santos is so into Tad Martin
  Tad Martin is so into Phoebe Wallingford
  Phoebe Wallingford is so into Erica Kane
Current schoolyard:
  Ruth Martin was kissed by Erica Kane

next sweetheart? bobby warner

Current kiss ring:
  Erica Kane is so into Jackson Montgomery
  Jackson Montgomery is so into Joe Martin
  Joe Martin is so into Anita Santos
  Anita Santos is so into Tad Martin
  Tad Martin is so into Phoebe Wallingford
  Phoebe Wallingford is so into Erica Kane
Current schoolyard:
  Bobby Warner was kissed by Jackson Montgomery
  Ruth Martin was kissed by Erica Kane

next sweetheart? ERICa kaNE

Current kiss ring:
  Jackson Montgomery is so into Joe Martin
  Joe Martin is so into Anita Santos
  Anita Santos is so into Tad Martin
  Tad Martin is so into Phoebe Wallingford
  Phoebe Wallingford is so into Jackson Montgomery
Current schoolyard:
  Erica Kane was kissed by Phoebe Wallingford
  Bobby Warner was kissed by Jackson Montgomery
  Ruth Martin was kissed by Erica Kane

next sweetheart? ANITA SANTOS
```

(continued)

```
Current kiss ring:
  Jackson Montgomery is so into Joe Martin
  Joe Martin is so into Tad Martin
  Tad Martin is so into Phoebe Wallingford
  Phoebe Wallingford is so into Jackson Montgomery
Current schoolyard:
  Anita Santos was kissed by Joe Martin
  Erica Kane was kissed by Phoebe Wallingford
  Bobby Warner was kissed by Jackson Montgomery
  Ruth Martin was kissed by Erica Kane

next sweetheart? phoebe wallingford

Current kiss ring:
  Jackson Montgomery is so into Joe Martin
  Joe Martin is so into Tad Martin
  Tad Martin is so into Jackson Montgomery
Current schoolyard:
  Phoebe Wallingford was kissed by Tad Martin
  Anita Santos was kissed by Joe Martin
  Erica Kane was kissed by Phoebe Wallingford
  Bobby Warner was kissed by Jackson Montgomery
  Ruth Martin was kissed by Erica Kane

next sweetheart? Barack Obama
Unknown person.

Current kiss ring:
  Jackson Montgomery is so into Joe Martin
  Joe Martin is so into Tad Martin
  Tad Martin is so into Jackson Montgomery
Current schoolyard:
  Phoebe Wallingford was kissed by Tad Martin
  Anita Santos was kissed by Joe Martin
  Erica Kane was kissed by Phoebe Wallingford
  Bobby Warner was kissed by Jackson Montgomery
  Ruth Martin was kissed by Erica Kane

next sweetheart? Joe Martin

Current kiss ring:
  Jackson Montgomery is so into Tad Martin
  Tad Martin is so into Jackson Montgomery
Current schoolyard:
  Joe Martin was kissed by Jackson Montgomery
  Phoebe Wallingford was kissed by Tad Martin
  Anita Santos was kissed by Joe Martin
  Erica Kane was kissed by Phoebe Wallingford
  Bobby Warner was kissed by Jackson Montgomery
  Ruth Martin was kissed by Erica Kane

next sweetheart? jackson montgomery

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

Implementation Details:

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

```
public class AffectionNode {
    public String name;          // this person's name
    public String kisser;        // name of who kissed this person (null if alive)
    public AffectionNode next;   // next node in the list

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

For this assignment we are going to specify exactly what fields you can have in your `AffectionManager` class. You must have exactly the following two fields; you are not allowed to have any others:

- a reference to the front node of the kiss ring
- a reference to the front node of the schoolyard (`null` if empty)

Implementation Details (continued):

Your class should have the following public methods.

`public AffectionManager(ArrayList<String> names)`

In this constructor you should initialize a new affection manager over the given list of people. Your constructor should not save the list parameter itself as a field, nor modify the list; but instead it should build your own kiss ring of linked nodes that contains these names in the same order. For example, if the list contains `["John", "Sally", "Fred"]`, the initial kiss ring should represent that John is so into Sally who is "so into" Fred who is "so into" John (in that order). You may 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.

`public void printKissRing()`

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

```
  Joe is so into Sally
  Sally is so into Jim
  Jim is so into Carol
  Carol is so into Chris
  Chris is so into Joe
```

`public void printSchoolyard()`

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

```
  Carol was kissed by Sally
  Chris was kissed by Carol
  Jim was kissed by Sally
```

`public boolean kissRingContains(String name)`

In this method you should return `true` if the given name is in the current kiss 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"`.

`public boolean schoolyardContains(String name)`

In this method you should return `true` if the given name is in the current schoolyard 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"`.

`public boolean isGameOver()`

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

`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.

`public void kiss(String name)`

In this method you should record the kissing of the person with the given name, transferring the person from the kiss ring to the front of the schoolyard. This operation should not change the relative order of the kiss ring (i.e., the links of who is "into" whom should stay the same other than the person who is being kissed/removed). Ignore case in comparing names. A node remembers who kissed the person in its `kisser` 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 kiss ring (if both of these conditions are true, the `IllegalStateException` takes precedence).

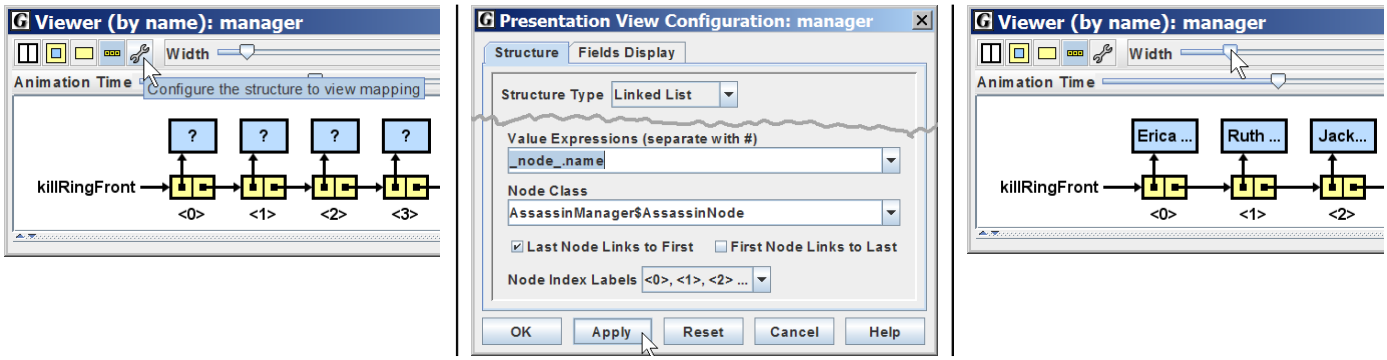
The `kiss` method is the hardest one, so write it last. Use the jGRASP debugger and `println` statements liberally to debug problems in your code. You will likely have a lot of `NullPointerException` errors, infinite loops, etc. and will have a very hard time tracking them down unless you are comfortable 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:

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:

- You must use our `AffectionNode` class for your lists. You are not allowed to modify it.
- You may not construct any arrays, `ArrayList`s, `LinkedList`s, stacks, queues, sets, maps, or other data structures; you must use linked nodes. You may not modify the list of `String`s passed to your constructor.
- If there are *N* names in the list of `String`s passed to your constructor, you should create exactly *N* new `AffectionNode` objects in your constructor. As people are kissed, you have to move their node from the kiss ring to the schoolyard by changing references, without creating any new node objects.

Your constructor will create the initial kiss 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 `AffectionNode` (like `current` from lecture) as you like. `AffectionNode` variables are not node objects and therefore don't count against the limit of *n* nodes.

You should write some of your own testing code. `AffectionMain` 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. You may share your testing code with others on the message board so long as the code does not give away the details of your program solution.

A word of caution: Some students try to store the kiss 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; we strongly suggest that you 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; it is easy to end up with infinite loops or other bugs. There is no need to use a circular list, because you can always get back to the front via the fields of your `AffectionManager`. If you feel strongly that you want to use a circular list, you are allowed to do so, but it is likely to make the program harder to write.

Style Guidelines and Grading:

Part of your grade will come from appropriately utilizing linked lists and nodes as described previously. Redundancy is another major grading focus; some methods are very similar, and you should avoid repeated logic as much as possible.

You should follow good general style guidelines such as: making fields `private` and avoiding unnecessary fields; appropriately using control structures like loops and `if`/`else` statements; properly using indentation, good variable names and proper types; and not having any lines of code longer than 100 characters. Do not use recursion on this program.

Comment your code descriptively in your own words at the top of your class, each method, and on complex sections of your code. Comments should explain each method's behavior, parameters, return, pre/post-conditions, and exceptions. For reference, our solution is around 130 lines long (60 "substantive" lines).