600.107 Introductory Programming in Java, Summer 2015
Homework 9
Due: Thursday, July 30th at 5:59pm - note unusual time

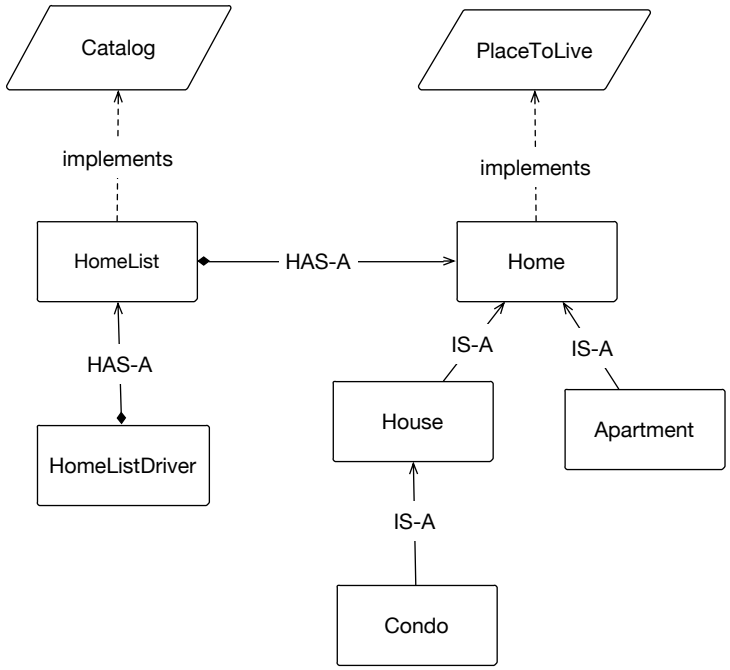
Please review the **Homework Guidelines** handout for this course before you begin.

Restriction: For this assignment, you may use arrays, but are NOT permitted to use ArrayLists.

Repetition Statements: "while", "for", or "do-while"- type statements MAY be used in this exercise, but you are NOT permitted to use `break;` or `continue;` statements except that you may use `break;` inside a case within a `switch` statement. You may NOT use `System.exit(0);` to end your program.

You may not use the `Arrays.sort()` method from the Java API; you'll instead need to include a full method definition of a sort (perhaps based on code from class).

The Task. For this final assignment, you will write a system of classes to support a list of homes in a particular town. We have carefully specified a set of classes that you must write to form this system, using inheritance and interfaces as appropriate. See the UML diagram to the right to get a sense of the relationships between files in this assignment.



Below is a list of the classes you must write and update. All instance variables must be private. You will likely need to write a few methods that are not explicitly mentioned here in order to accomplish everything. Javadoc comments are required in all files.

Class Home. Create a class to contain information about a home. This class must implement the PlaceToLive interface which we have provided for you in HW9Files.zip, available on Piazza. A Home contains data members for its monthly cost, its square footage, and its street address (e.g., 3400 N. Charles St.). In the overridden .equals method for Homes, you must check equality by comparing whether the String street addresses are equal, ignoring case. You will also need to override the toString() method so that the string representation of a Home contains its street address, cost per month, and square footage as shown in the example below. Note that the monthly cost must be displayed with two decimal places, rounded "half-up", so that, for example, 299.225 would appear as 299.23. You will likely want to use `String.format()` for this, which formats doubles in a way similar to `printf()`, but to insert them into Strings rather than as output.

```
1 Infinite Loop, 567.0 sq ft, $1967.34
```

Additionally, since PlaceToLive extends the Comparable interface, you must write a method named compareTo() for this class. In it, Homes must be compared by comparing the results returned by the costPerMonth() method. That is, your compareTo() method will consider homes which cost less per month as "less than" homes which cost more per month.

Class House. This will be a subclass of the Home class. Include an instance member that indicates whether the House is paid off or not. By default, a House is not paid off. The House class must override the costPerMonth() method. The costPerMonth() for a House is defined to be 0 if the House is paid off, but if the house is not paid off, then it returns the monthly cost. The string representation of a House must contain its street address, square footage, cost per month, and whether the House is paid off or not. The monthly cost must be displayed with exactly two decimal places. An example of the required string representation is here:

```
1 Hacker Way, 617.3 sq ft, $2399.80, not paid off
```

Class Condo. This will be a subclass of House. Include an instance member that stores the amount of the monthly condo association fees. By default, a Condo is not paid off. The Condo class must override the costPerMonth() method. If it is already paid off, the costPerMonth() for a Condo is defined to be the monthly fee amount; if if it is not paid off, the costPerMonth() method for a Condo is the sum of its monthly cost and monthly fees. The String representation of a Condo must contain its street address, square footage, cost per month, and whether the Condo is paid off or not. The monthly cost must be displayed with exactly two decimal places. It should look like that of the House class.

Class Apartment. This is a subclass of Home. Include an instance member for the monthly renter's insurance. The Apartment class should override the costPerMonth() method. The costPerMonth() for an Apartment is defined to be the monthly cost plus the renter's insurance. The string representation of an Apartment should contain its street address, square footage, and cost per month. The monthly cost must be displayed with exactly two decimal places. It should look like that of the Home class.

Class HomeList. This implements the Catalog interface, which is provided for you in HW9Files.zip. This class will hold a collection of Homes and the number of homes in the collection. Some notes on the methods you have to write for this class:

- add(Home h): Allows the user to insert a Home into the HomeList. A Home cannot be added if another home in the database has the same address, ignoring case. This method returns false if the Home was not added, true otherwise. You must resize the array if there is an attempt to add a Home to a full array of homes. The start size of the array must be `START_SIZE`, which is defined for you in the Catalog interface. Each time you resize, you must double the size of the array.
- remove(double cost): Allows the user to remove all the Homes in the list whose monthly costs are greater than or equal to the specified cost. It returns the number of Homes that were removed. Your array must never contain null positions in the middle of a section of actual data; make sure your "good" data starts at index 0 within the array, and remains contiguous in the array, even after removals take place.
- display(): Displays all the homes in the array. It must report a message that there are no homes in the list when appropriate.
- setPaidOffForHousesOrCondo(Scanner kb): Takes the kb from the HomeListDriver class, and must print all the Houses or Condos (but not the Apartments) in the array and their corresponding indices in the array, as a menu of choices for the user. The output should look like:

  ```
  0) 1 Infinite Loop,  567.0 sq ft, $1967.34, not paid off
  2) 1600 Amphitheatre Parkway, 520.0 sq ft, $1656.70, not paid off
  Enter index ->
  ```

  After the user selects a House or Condo by entering its index number, the method toggles the paid-off state of the selected House or Condo. That is, if the selected House is currently listed as not paid off, this method will set it to paid off, and vice-versa. If there are no houses or condos in the list at the time this method is called, this method should just output:

  ```
  There are currently no Houses or Condos in the list.
  ```

  It should not collect any input from the user in this case.

- displayGroups(): Displays the Homes in the list grouped by type (Houses, then Apartments, then Condos). If there is no Home in a certain group, the program should output that there were no Homes in that group. See below (option 7) to see what the output should be.
- getHomeLargestSquareFootage(): This will return the Home with the largest square footage. It will return null if there are no homes in the array.
- sort(): This will sort all the Homes in the array based on costPerMonth, from lowest cost to highest cost. You are permitted to reuse sorting (or other) code provided in class, provided that you include comments in your files documenting its source. (You are not permitted to use the `Arrays.sort()` method.)

Class HomeListDriver. This is the driver class that will contain an instance of HomeList. The user will be able to perform different actions to the HomeList instance by adding, removing, sorting, and printing homes via a menu that will take input from the user. The menu will look like:

```
1) Add new House
2) Add new Apartment
3) Add new Condo
4) Print all the homes
5) Sort all homes by cost per month
6) Remove homes over a certain cost per month
7) Display homes in groups
8) Get home with largest square footage
9) Toggle whether House or Condo is paid off
10) Quit
Choice -->
```

HomeListDriver Option Descriptions.

(1) Adds a new House to the HomeList. Asks the user for the street address, the square footage, and monthly cost of the House. A House cannot be added to the list if another Home with the same address is in the list already, and the method must output whether or not the House was successfully added. It must follow this pattern:

```
Enter address for house -> 1 Infinite Loop
Enter square footage for house -> 567
Enter monthly cost for house -> 1967.34
House successfully added.
```

(2) Adds a new Apartment to the HomeList. Asks the user for the the address of the apartment, its square footage, its monthly cost, and the monthly renter's insurance. An Apartment cannot be added to the list if another Home with the same address is in the list already, and the method must output whether or not the Apartment was successfully added. It must follow this pattern:

```
Enter address for apartment -> 1 Hacker Way
Enter square footage for apartment -> 844.23
Enter monthly cost for apartment -> 2133.4567
Enter renter's insurance for apartment -> 387.224
Apartment successfully added.
```

(3) Adds a new Condo to the HomeList. Asks the user for for the street address of the Condo, its square footage, the monthly cost of the Condo, and the monthly fees for the Condo. A Condo cannot be added to the list if another Home with the same address is in the list already, and the method must output whether or not the Condo was successfully added. It must follow this pattern:

```
Enter address for condo -> 1 hacker way
Enter square footage for condo -> 520
Enter monthly cost for condo -> 1456.7
Enter monthly fees for condo -> 200
A home with that address is already listed.
```

(4) Outputs the String representation of each home in the HomeList. If there are no homes in the list, it should print `There are currently no homes in the list.` Otherwise, the output could look like this:

```
1 Infinite Loop, 567.0 sq ft, $1967.34,  not paid off
1 Hacker Way, 844.23 sq ft, $2520.68,
1600 Amphitheatre Parkway, 520.0 sq ft, $1656.70 not paid off
```

(5) Sorts all the homes in the list by cost per month, in ascending order. No output is produced, other than the message, `Homes sorted.`

(6) Removes all the homes in the list that are greater than or equal to a cost threshold specified by the user. It will then output the number of homes removed. The interaction for this option might look like:

```
Enter cost threshold -> 1800
Removed 4 homes.
```

(7) Displays the String representations of all of the Houses, Apartments, and Condos currently in the HomeList, grouped by type. In other words, the Houses will be shown together, the Apartments will be shown together, and the Condos will be shown together. Here is an example of output from this method:

```
-- Groups --
Houses:
1 Infinite Loop, 567.0 sq ft, $1967.34,  not paid off

Apartments:
1 Hacker Way, 844.23 sq ft, $2520.68
12 Main Street, 895.0 sq ft, $895.00

Condos:
1600 Amphitheatre Parkway, 520.0 sq ft, $1656.70, not paid off
```

If no homes of a certain type are currently in the HomeList, this method prints an appropriate message, as shown in the following example:

```
-- Groups --
Houses:
1 Infinite Loop, 567.0 sq ft, $1967.34,  not paid off

Apartments:
1 Hacker Way, 844.23 sq ft, $2520.68
12 Main Street, 895.0 sq ft, $895.00

Condos:
There are no condos currently in the list.
```

(8) Outputs the String representation of the Home with the largest square footage. If there are no homes in the list, it should output: `There are no homes currently in the list.`

(9) Toggles the paid off state for a selected House or Condo. All of the work for this choice must be done by the setPaidOffForHousesOrCondo(Scanner kb) method in the HomeList class. See the above description of this method.

(10) Ends the program.

A lengthy sample execution of the main method in this HomeListDriver class is included in HW9Files.zip.

Turn-in. Before the due date listed at the top of this file, please submit via the course Blackboard site one complete `.zip` file named `HW9-jhed.zip` (where you replace 'jhed' with your own personal JHED) containing all `.java` files required for the above tasks **EXCEPT PlaceToLive.java and and Catalog.java**, which were provided for you (we will grade your code using our own copies of those two files). Be sure that all submitted code compiles! If you were unable to complete a task, please include as part of your `.zip` submission a text file named `README` explaining anything you'd like the graders to know.