DUE DATE: APRIL 14TH, 2021 AT 4:59PM

Instructions:

- You must complete the "**Blanket Honesty Declaration**" checklist on the course website before you can submit any assignment.
- Only submit the **java files**. Do ***not*** submit any other files, unless otherwise instructed.
- To submit the assignment, upload the specified files to the **Assignment 4** folder on the course website.
- Assignments must follow the **programming standards** document published on UMLearn.
- After the due date and time, assignments may be submitted but will be subject to a late penalty. Please see the ROASS document published on UMLearn for the course policy for late submissions.
- If you make multiple submissions, only the **most recent version** will be marked.
- These assignments are your chance to learn the material for the exams. Code your assignments independently. We use software to compare all submitted assignments to each other, and **pursue academic dishonesty vigorously**.
- Your Java programs must compile and run upon download, without requiring any modifications.
- Automated tests will be used to grade the output of your assignment. If you do not follow precisely the guidelines described below, these automated tests can fail and you will lose marks. In particular, **make sure that your methods are spelled exactly as described below**. Also, **make sure that your code produces exactly the example outputs (including correct spacing)** shown in these guidelines.

Testing Your Code

We have provided sample test files for each phase to help you verify that your code is working to the assignment specifications (e.g., correct method headers). **These files are only *starting* points for testing your code. They do not represent a complete or systematic test suite.** Part of developing your skills as a programmer is to think through additional important test cases and to write your own code to test these cases. The test files should give you a sense of what this additional code could consist of.

Question 1: Linked Lists

Create a PersonLinkedList class to hold a linked list whose values are of type Person (use the Person.java file provided in the Assignment 4 folder). You will also have to implement a PersonNode class to represent nodes in the linked list. Your linked list must implement the methods below. For full marks, your methods should be efficient in that they should not do any unnecessary extra traversals through the list.

- A constructor with no parameters which creates an empty list.
- **`public void insertByAge(Person newItem)`** – the insert method should add the Person object to the list such hat the list is ordered according to age (youngest to oldest)
- **`public int size()`** which returns the number of elements in the linked list
- **`public boolean contains(Person person)`** which returns true if the list contains the data item and false otherwise.
- **`public Person get(int pos)`** which returns the Person object at the given position in the list (where the position of the first item in the list is 0, the position of the second item is 1, etc). If there is no such position in the list, the method should return null.
- **`public double getAverageAge()`** which computes and returns the average (mean) age of the people in the list. The method should return 0.0 if the list is empty.
- **`Public boolean equals(PersonLinkedList otherList)`** which will return true if the list contains identical Person information in the same order.
- **`public PersonLinkedList removeIfYounger(int age)`** – this method should remove all Person objects from the list who are younger than the age passed as a parameter. The removed items should then

be returned as a new ordered PersonLinkedList. You can assume the list will always be maintained in sorted order according to age.

- **`public String toString()`** – returns a String containing the string representations of the Person objects in the list, in the order they appear. Each Person object should be on a separate line, as shown in the sample output below. There should not be extra newline character after the last element.
- **`public PersonLinkedList clone()`** – returns a deep copy of the linked list, with a new set of nodes. The data should be in the same order as the data in the original list.

As a starting point, you can test your classes using the supplied test program **`TestQ1.java`**. Running this code should result in the output below. To reiterate the instructions at the top of the assignment, this is not a comprehensive test suite.

```
List1 now has 1 items:
Alice (19)
List1 now has 2 items:
Alice (19)
Jafari (27)
List1 now has 3 items:
Alice (19)
Ling (22)
Jafari (27)
List1 Contains Alice? true
List1 Contains other Alice? false
Average age in List1 22.67
At position 0: Alice (19)
At position 2: Jafari (27)
At position 4: null
List1 with younger than 23 removed:
Jafari (27)
List2 now has those items:
Alice (19)
Ling (22)
List3:
Alice (19)
Ling (22)
List2 Equals List3? true
List2 Equals List1? false
List4:
Alice (19)
Ling (22)
Jafari (27)
```

Question 2: LinkedLists and Recursion

Add the following two methods (and their helper methods) to your PersonLinkedList class. These problems must be solved ***using recursion***. There should be no loops of any kind.

- Add a **`public String reversedToString()`** that returns a String representation of the list, but in reverse order. This method should call a private recursive helper method to generate the reversed string representation.
- Add a **`public int nameFrequency(String key)`** that returns the number of Person objects in the list whose names match the key. This method should call a private helper method that traverses the list recursively.

There is no separate file to submit for Question 2. Testing your code with TestQ2.java should produce the following output.

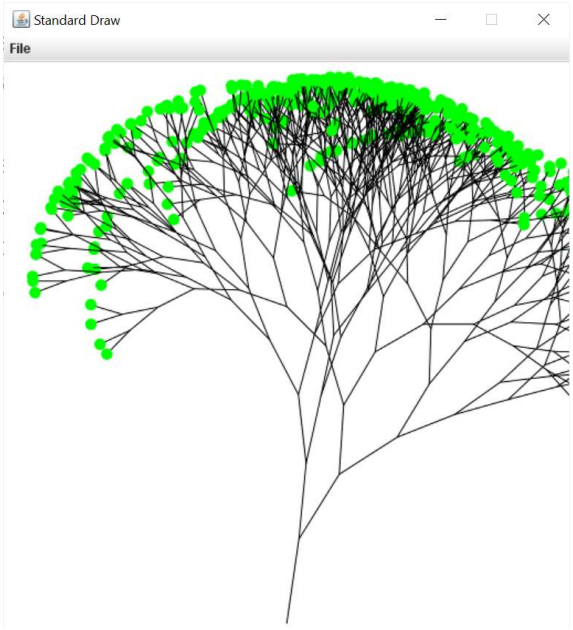
```
List1 forwards
```

```
Alice (19)
Ling (22)
Jafari (27)
List1 reversed
Jafari (27)
Ling (22)
Alice (19)
List2 forwards
Jafari (27)
List2 reversed
Jafari (27)
List3 forwards

List3 reversed

Frequency of Alice: 2
Frequency of Happy: 0
Frequency of Jafari in an empty list 0
```

Question 3: Recursive graphics (Fractal Trees)



Write a program, using recursion, that draws a tree in the StdDraw window as shown in the example above. This tree is a shape called a fractal. Fractals are shapes that have a recursive structure. In this example, every line has two other lines sprouting from it, at two different angles. Then, each of those lines has two smaller lines sprouting from it, and this process continues recursively until you reach the leaves.

Download **`TestQ3.java`**, and **`StdDraw.java`**. Define a class **`FractalTree.java`** that has one method:

`public static void drawTree(double startX, double startY, double length, double theta, int depth)`.

`drawTree` should draw a tree that has **`depth`** levels of branches. The main branch should be a line starting at StdDraw coordinate (**startX** , **startY**). This line should have the given **length** and be drawn at an angle given by **theta** radians. If your knowledge of trigonometry is rusty, that means that the other end of the line will be at StdDraw coordinate (**`startX+length*Math.sin(theta)`**, **`startY+length*Math.cos(theta)`**). The StdDraw command to draw a line from(x1,y1) to (x2,y2) is **`StdDraw.line(x1,y1,x2,y2)`**. Then there should be two smaller trees, with **`depth-1`**

levels, that branch out from the end of this line. One should go a random amount to the left, and one should go a random amount to the right. (In the example above, random amounts from 0 to Math.PI/6 were used. Try different values to see what kinds of trees you get.) The branches should become shorter as you move up the tree. (In the example above, each branch is 10% shorter than the one below. Try your own values.) At the end of the smallest lines (i.e., at the top of the tree) draw small green circles to represent the leaves, using **`StdDraw.filledCircle(x,y,radius)`**.You can chose the radius for your leaves. To control colours, use **`StdDraw.setPenColor(Color)`**, where **`Color`** is a class defined in **`java.awt.Color`** (look up its description online).

Hand in

Submit your three Java files (**`PersonLinkedList.java`**,**`PersonNode.java`**,**`FractalTree.java`**). ***Do not submit .class or .java~ files!*** You do ***not*** need to submit the **`TestQN.java`**, **Person.java** or **StdDraw.java** files. If you did not complete all three questions of the assignment, use the Comments field in UMLearn when you hand in the assignment to tell the marker which questions were completed, so that only the appropriate tests will be run. For example, if you say that you completed Questions 1-2, then the marker will compile your files and then run the automated tests for questions 1-2 only. If the appropriate test files do not compile and run, you will lose ***all*** of the marks for the test runs. The marker will ***not*** try to run anything else, and will ***not*** edit your files in any way. ***Make sure none of your files specify a package at the top!***