RIT | Golisano College of Computing and Information Sciences
School of Information

ISTE-120 - Computational Problem Solving for the Information Domain I
Homework Assignment 9 (HW09)

DELIVERABLE
Zip your files together and submit the zip file to the MyCourses Assignment folder for this homework. Homework MUST be submitted on time to receive full credit. Homework submitted to the "late" dropbox will receive a maximum grade of 80%. Homework not submitted to either dropbox will receive no credit.

PROBLEM 1 - Arrays (Note: There are two separate problems in this HW.)

Write a Java application to simulate coin tosses and compute the fraction of heads and fraction of tails. As a challenge, determine the longest run of one outcome; that is, the most consecutive heads or consecutive tails (whichever is longer).

To simulate the toss of a coin, use the **random** method in the `Math` utility class, that generates a number greater than or equal to `0` and less than `1.0`. The notation for this range is [`0 , 1.0`), where the square bracket means to include the 0 in the range and the rounded parenthesis means to exclude the 1.0 from the range. Since there are two outcomes of equal probability in the toss of a "fair" coin, divide the range of numbers between [`0 , 1.0`) into two roughly equal size sub-ranges. Assume any number less than `0.5` will be considered as a **head** while any number greater than or equal to `0.5` will be considered as a **tail**.

Program Organization Requirements:
`CoinToss.java`

- A parameterized constructor that accepts an integer parameter for the number of tosses. This value will be used to set the size of an array that will hold the coin toss results
- A loop is used to fill an array with the characters '`h`' (head) and '`t`' (tail). As described above, use a random number to determine if each "coin toss" is a head or a tail. This array initialization is accomplished in the constructor (HINT: you can keep a count of the number of **heads** and **tails** as attributes as you fill the array)
- Create methods that return the number of heads and the number of tails in the coin toss array (these are accessors)
- Create a method that returns the total number of coin tosses
- Create a method that computes the longest (consecutive) number of heads or tails in the coin tosses array. This method stores this information in two

attributes: one for the length of the longest run, and one for the outcome of that longest run (head or tail). Initialize the length to 0

To compute the longest run, keep two MORE values (not attributes, but local variables: one for the length of the *current* run of identical outcomes, and one for the *outcome associated with the run* (head or tail).

- Initialize the outcome for the *current* run to neither a head nor a tail (perhaps an 'x') and the length of the run to 0
- In a loop, compare each toss in the array to the outcome of the *current* run
    - If they match, the *current* run is now recognized as one longer (increment its length)
    - If they don't match, then the *current* run has ended. Compare the length of this run to the longest run that has been previously seen (the longest run so far)
        - If this is *current* run is longer, record its length and outcome in the two attributes (above)
    - Whether or not this is a new longest run, reset the *current* outcome to the one just examined (the mismatch) and reset the length of the current run to 1. Think about if any special processing needs to happen when the loop terminates

- Define accessor and mutator methods for the length and outcome of the longest run
- Define any additional methods as needed

`TestCoinToss.java`

- Prompt the user for the number of coin tosses
- You will repeat the run until the user enters any value less than 1

TESTING

The results should be similar in nature to those shown in the sample run window below. Because we are using a random number generator, the results will change each time the application is executed. Use Scanner and a loop to get the number of coin tosses.

```
----jGRASP exec: java TestCoinToss
Enter integer number ( >= 2) coin tosses or 0 to Exit: 29

Number of Coin Tosses = 29
Fraction of Heads = 0.586
Fraction of Tails = 0.414
Longest run is 4 heads
Enter integer number ( >= 2) coin tosses or 0 to Exit: 725

Number of Coin Tosses = 725
Fraction of Heads = 0.502
Fraction of Tails = 0.498
Longest run is 10 heads
Enter integer number ( >= 2) coin tosses or 0 to Exit: 0

End of Program

 ----jGRASP: operation complete.
```

DETAILS

1. Submit the files to the appropriate MyCourses Assignment folder
2. All calculations must be done in the CoinToss class
3. Scanner should be used for input
4. Proper data types should be used

PROBLEM 2 – ArrayLists

Write a Java application that uses an `ArrayList` to store information about new cars. You will write three classes: `NewCar, NewCarList` and `NewCarListTester.`

NewCar

The `NewCar` class is from HW6. Simply copy it from your HW6 solution and use it again here.

NewCarList

This is a class that has no constructors and a single attribute, an instantiated `ArrayList` used to store information about `NewCar`'s. For example:

```
ArrayList<NewCar> carList = new ArrayList<NewCar>();
```

`NewCarList` will have the following four methods:

- The `add()` method will accept a `NewCar` object and add it to the `ArrayList`
- The `display()` method will display, using `toString()`, the information about each object in the `ArrayList`
- The `select(double maxPrice)` method will do the same thing as `display()`, except that it will display only the cars with a final price less than or equal to the parameter `maxPrice`
- The `drop(int index)` method will delete the `NewCar` stored in the `ArrayList` slot with the given `index`

NewCarListTester

In `NewCarListTester`, the main method will instantiate five `NewCar`s. Hard code the data for each car, using the `NewCar` constructor and other methods (e.g., `calcFinalPrice`). The actual data values are in the file `HW09Data.xlsx`.

For each car, create (i.e. instantiate) a `NewCar` object using a parameterized constructor for year, make and model (see HW06) and pass this object to the `add` method in `NewCarList` that uses that object as a parameter so it can be added to the `ArrayList`.

Included in `NewCar` is an attribute, `finalPrice`, with an appropriate accessor titled `getFinalPrice()` accessor. `NewCar` also will provide the following methods:

```
calcFinalPrice(double stickerprice, double discount,
               double salesTaxRate)
calcZeroPctMonPmt(int numMonths)
getCarName()
getCarAbbrev()
toString()
```

Create a `NewCarListTester` class that will:

1. Add five cars to the list (run `calcFinalPrice` for each). Again,the data input is hard-coded; i.e. there is no need to use Scanner input
2. Display all the cars in the list
3. Display all the cars under a certain final price (inclusive) (use the select method and be sure to test a boundary condition)
4. Add three more cars (run `calcFinalPrice` for each) and display the list
5. Drop the third item in the list (What is the index?)
6. Display the cars again

DETAILS

1. Submit your files to the MyCourses Assignment folder
2. Carry out all calculations in the within the `NewCar` and `NewCarList` classes
3. Use appropriate data types
4. Loop counts should not be hard coded (flexible number of cars in the list)

Output should resemble the following:

```
----jGRASP exec: java NewCarListTester
*** List of cars
You want to purchase a "2020 Kia Rio" Abbreviation: "20KR"
Final price: 17485.2
You want to purchase a "2017 Kia Rio" Abbreviation: "17KR"
Final price: 11869.2
You want to purchase a "2014 Honda Civic" Abbreviation: "14HC"
Final price: 14677.2
You want to purchase a "2020 Honda Civic" Abbreviation: "20HC"
Final price: 22415.4
You want to purchase a "2010 Chevrolet Cobalt" Abbreviation: "10CC"
Final price: 7554.6

*** List of cars under $14500
You want to purchase a "2017 Kia Rio" Abbreviation: "17KR"
Final price: 11869.2
You want to purchase a "2010 Chevrolet Cobalt" Abbreviation: "10CC"
Final price: 7554.6

*** Add three more cars and list
You want to purchase a "2020 Kia Rio" Abbreviation: "20KR"
Final price: 17485.2
You want to purchase a "2017 Kia Rio" Abbreviation: "17KR"
Final price: 11869.2
You want to purchase a "2014 Honda Civic" Abbreviation: "14HC"
Final price: 14677.2
You want to purchase a "2020 Honda Civic" Abbreviation: "20HC"
Final price: 22415.4
You want to purchase a "2010 Chevrolet Cobalt" Abbreviation: "10CC"
Final price: 7554.6
You want to purchase a "2010 Honda Accord" Abbreviation: "10HA"
Final price: 11869.2
```

```
You want to purchase a "2019 Subaru Forester" Abbreviation: "19SF"
Final price: 24397.2
You want to purchase a "2020 Chevrolet Malibu" Abbreviation: "20CM"
Final price: 22314.96

*** Delete second item in the list
You want to purchase a "2020 Kia Rio" Abbreviation: "20KR"
Final price: 17485.2
You want to purchase a "2014 Honda Civic" Abbreviation: "14HC"
Final price: 14677.2
You want to purchase a "2020 Honda Civic" Abbreviation: "20HC"
Final price: 22415.4
You want to purchase a "2010 Chevrolet Cobalt" Abbreviation: "10CC"
Final price: 7554.6
You want to purchase a "2010 Honda Accord" Abbreviation: "10HA"
Final price: 11869.2
You want to purchase a "2019 Subaru Forester" Abbreviation: "19SF"
Final price: 24397.2
You want to purchase a "2020 Chevrolet Malibu" Abbreviation: "20CM"
Final price: 22314.96

 ----jGRASP: operation complete.
```

RIT | Golisano College of Computing and Information Sciences
School of Information

Name: ____

Homework 9 Grade Sheet

Note: No functionality points if the program does not compile properly.

Required Functionality of Problem#1: Coin Toss	**Pt. Value**	**Pts. Earned**
- Program produces the correct results	**15**	
- Required features in the Java code	**25**	
- Correct declaration of array of char		
- Correct use of a random number to determine head or tail		
- Correct loop to store 'h' and 't' into array		
- Correct loop to count heads and tails		
- Format of output matches sample		
- Correctly determines longest run of heads or tails	**10**	
Required Functionality: Problem #2: NewCarList		
- Correct modification to the toString class in NewCar class		
• Final Price added to returned String	**2**	
• Comes on a new line	**2**	
• final price stored as an attribute	**2**	
- Required features in the NewCarList class		
- ArrayList declared correctly	**5**	
- add method is correctly added	**5**	
- display method is correctly added	**5**	
- select method is correctly added	**5**	
- drop method is correctly added	**5**	
- Loops do not use hard coded limits	**5**	
- Required features in the NewCarListTester class		
- Five cars are added using NreCarList	**2**	
- All cars are displayed using NewCarList	**2**	
- Some cars are selected using NewCarList	**2**	
- Three more cars are added	**2**	
- the third car is deleted	**2**	
- The cars are listed again.	**2**	
- Output is correct	**2**	
Total Points	**100**	

Additional Comments: