Given: Saturday, November 21, 2015
Due: Saturday, December 5, 2015 (before 11:59pm)

Assignments handed in after midnight will be awarded zero points but will be accepted up until 6 am without any penalties. Any assignments handed in after 6am (even 1 minute) will be awarded a grade of zero.

Points for early submission:

- 0 points = Saturday Dec 5th after midnight
- 1 point = Saturday Dec 5th before midnight
- 2 points = Friday Dec 4th before midnight.
- 3 points = Thursday Dec 3rd before midnight.
- 4 points = Wednesday Dec 2nd before midnight.

IMPORTANT NOTE:

- The *"static"* keyword can only be used for the main method and any constants created for the assignment. Use of the static key words anywhere else will result in large deductions.

The assignment is to be completed individually, in accordance with the policies and expectations in the course outline.

Objectives:

- To gain experience developing a custom class with instance variables.
- To learn about the very important concept of "encapsulation".
- To practice thinking about objects as having both *state* (via properties) and *behavior* (via public methods).
- To gain experience working with multiple objects ("instances") of a class, each with its own distinct identity and state.
- To gain experience with constructor methods.
- To gain experience with using Array Lists

Case Scenario

You will be building a basic program that will allow the user to load a file containing information about a group of Populace's (i.e. hamlets, villages, towns and cities) in a country and then presents them with a list of operations that they can do against the data.

The program must go through the following basic steps in the order shown:

1. Allows the user to enter a file name. Data will be loaded from the file which contains information about a group of populaces. The format of the file is described on this document.
   - If the user enters a file name that is blank/empty, (i.e. they just hit the enter key), the program continues to prompt for re-entry of the file name.
2. All information is loaded from the file into the program.
   - Information for each populace should be loaded into individual populace objects. This will require you to first create a Populace class. The specific requirements for this class are given in this document.
   - As you load each Populace, you will need to save the data for later processing. This will require you to use an ArrayList.
3. Present and run the text-based menu to the user until they select to quit and exit the program.
4. When the user selects quit from the menu, the program shuts down with a friendly exit message.

The Populace Class

Design a Populace class with the following:

1. The name of the class must be: Populace
2. It must have the following instance variables (data):
   - `name` , the name of the populace
   - `popSize` , the population size of the populace
   - `xCoord` , the x coordinate location of the populace on a 2D map
   - `yCoord` , the y coordinate location of the populace on a 2D map
3. a constructor: Takes in four (4) parameters used to set the instance data for:
   - **`name, popSize, xCoord, yCoords`**
4. Accessor and mutator methods for each instance variable (i.e get and set methods)
5. The following four (4) Populace methods:
   - **`public boolean isHamlet()`** – returns true if the population size is less than or equal to 500, false otherwise.
   - **`public boolean isVillage()`** – returns true if the population size is larger than a hamlet but less than or equal to 2000, false otherwise.
   - **`public boolean isTown()`** – returns true if the population size is larger than a village but less than or equal to 5000, false otherwise.
   - **`public boolean isCity()`** – returns true if the population size is larger than a Town, false otherwise.

   Note: it would be a very good idea to create constants in the Populace class to represent the upper bounds for a community type.
6. The following methods
   - **`public boolean sameName(String cmpNameTo)`** – returns `true` if the name of the current populace matches the name, `cmpNameTo`. It will return `false` otherwise. The comparison is *case insensitive.*
   - **`public int distanceTo(Populace otherPopulace)`** – calculates and returns the distance between the current populace coordinates and the populace's (`otherPopulace`) coordinates. The distance between two populaces is calculated in the following way:

     Populace 1's `(xCoord, yCoord) = (x1, y1)`

     Populace 2's `(xCoord, yCoord) = (x2, y2)`

     $$distance = \sqrt{(x2 - x1)^2 + (y1 - y2)^2}$$

File Format

For this assignment, you can assume that there are no errors in the data in the input files to the program. Therefore, no data validation is required for the data from the file. Each line in the data file will contain information about a populace.

For example, the first line in the data file `example.txt`: Mirror 468 50 100.

The line from above would translate to the following populace information:

- Name = Mirror
- Population size = 468
- xCoord = 50
- yCoord = 100

```
Mirror 468 50 100
Greendale 5748 45 200
Whoville 3222 5 10
Sunnydale 1555 30 5
```

example.txt

User interface (text-based menu)

1. The menu options are as follows:

```
1 – List all populaces
2 – Add populace
3 – Remove populace
4 – Display distance between two populaces
5 – Show stats
6 – Quit
Enter choice ( Between 1 to 6):
```

2. The menu prompts the user for their choice of operation in the menu item. The menu item choice is between 1 and 6 (both inclusive).
3. The program error checks for valid input choice from user.
    a. If an incorrect value is entered, a suitable error message is printed.
    For example,

    ```
    Enter choice ( Between 1 to 6): 7
    Invalid choice entered. Enter choice again!
    Enter choice ( Between 1 to 6):
    ```

4. The program executes the corresponding action according to user choice.
    a. Unless the user chooses to quit (menu option 6), the menu is re-displayed.
    b. If the user chooses to quit (menu option 6), the program ends.
5. The menu options as it should function are described below:
    a. List all populaces
        - This will list all populaces in the array list, including the name, population size and the coordinates (x, y). For example,

        ```
        (1) Name=Mirror | PopSize=468 | Location=(50,100)
        (2) Name=Greendale | PopSize=5748 | Location=(45,200)
        (3) Name=Whoville | PopSize=3222 | Location=(5,10)
        (4) Name=Sunnydale | PopSize=1555 | Location=(30,5)
        ```

    b. Add populace
        - The user can add a new populace. The system prompts the user to enter the information pertaining to the populace.

        ```
        Enter populace name: Redmond
        Enter population size: 10000
        Enter x coordinate: 15
        Enter y coordinate: 30
        ```

        - Input validation is processed as described below:
            - Populace name
                - Check if the name already exists in the list of populaces. This will require you to search through all the populaces already in the system and check that the new name does not match any of the populace names. This search must be case insensitive.
            - Population size
                - Error check to make sure the size entered is a positive value
            - x coordinate – no error check required
            - y coordinate – no error check required

c. Remove populace

- The program prompts the user for the name of the populace to be removed from the list.
  `Enter populace name to be removed: Redmond`
- The program searches all the populaces in the list and removes the first populace it finds with a matching name.
- If the entered populace name does not exist in the list, the program indicates that with appropriate message that nothing was removed.

d. Display distance between two populaces

- The program prompts the user for the name of two populaces, `firstPopulaceName` and `secondPopulaceName`. Ideally, the program proceeds to compute the distance between them and prints out the distance as:
  `The distance between the two populaces entered is: 100.13.`
- If either name does not exist in the list, the program displays an appropriate message:
  "`One or both is/are unknown populace(s), nothing can been done!`"
- If both populace names are the same, the program displays an appropriate message:
  "`You have entered the same name for both populaces`".

Note: Write a method that searches the array list for a given populace and returns the index when found. If no matching populace is found, it returns a value of -1.

e. Show stats

- The program generates and shows the current stats based upon the populaces in the list. For example,

```
Total population size:       51193
Total # hamlets:             5
Total # villages:            2
Total # towns:               4
Total # cities:              5
The largest populace, and its population in current list:      MoneyVille, 9009
The smallest populace, and its population in the current list: Holden,381
```

The program lists the total population size of the populaces in the list, the total number of hamlets, the total number of villages, the total number of towns, the total number of cities.

The populace with the largest population size is also printed along with its population.

  - If two or more populaces have the same population as the largest, the first one found based upon the order in the array list is printed.

The populace with the smallest population size is also printed along with its population.

  - If two or more populaces have the same population as the smallest, the first one found based upon the order in the array list is printed.

f. Quit

- The program exits the text-based menu. The program prints out an exit message "`Thank you for using the program!!`" and the program ends.

You should try to match your output to the same as shown below:

example.txt

```
Total pop size: 10993
         # Hamlets: 1
        # Villages: 1
            # Town: 1
            # City: 1
 Largest Populace: (      Greendale, 5748)
Smallest Populace: (         Mirror,  468)
```

Empty.txt

```
    Total pop size: 0
         # Hamlets: 0
        # Villages: 0
            # Town: 0
            # City: 0
 Largest Populace: (           NONE,    0)
Smallest Populace: (           NONE,    0)
```

One.txt

```
Total pop size: 5748
         # Hamlets: 0
        # Villages: 0
            # Town: 0
            # City: 1
 Largest Populace: (      Greendale, 5748)
Smallest Populace: (      Greendale, 5748)
```

Large.txt

```
    Total pop size: 51193
         # Hamlets: 5
        # Villages: 2
            # Town: 4
            # City: 5
 Largest Populace: (     MoneyVille, 9009)
Smallest Populace: (         Holden,  381)
```

Submission Instructions

Before attempting the steps below, ensure your solution compiles **without errors**, ensure all existing tests pass, and ensure your program works as expected.

Code that does not compile and can't be tested will be heavily penalized.

Important reminder: This assignment is to be completed *individually*. Students are strictly cautioned against submitting work they didn't do themselves.

Please follow submission instructions precisely.

1. Rename your BlueJ project folder (if you haven't already) as:
   `<LastName>_<FirstName>_Asg5`
2. Submit the entire BlueJ project folder (with all its contents) to the "submit" folder. If you are submitting from a non-university computer (e.g. a home PC), you can access the submit folder via `secure.mtroyal.ca`, into which you can upload a compressed (e.g. ".zip") version of your main folder.