COMP1161 Object-Oriented Programming
Project #2 (Semester 2, 2014/15)

To be attempted by 2-3 students

SIMULATING A GAME OF FRIENDS

A society is a collection of citizens. Each citizen is a person who has a citizen number, and a location. Every citizen has a set of friends where a friend is another person whose compatibility score exceeds a personal threshold (called a friend value). The following is an illustration of compatibility scores and friend values for four citizens.

	COMPATIBILITY SCORES			
CITIZEN	John	Mary	Martha	Richard
John (friend value = 2.0)	6	2.1	1.6	1.4
Mary (friend value = 1.8)	2.1	6	1.2	1.8
Martha (friend value = 2.1)	1.6	1.2	6	3
Richard (friend value = 1.5)	1.4	1.8	3	6

A friendship can only be formed when two citizens meet. For example, if John meets Mary he will calculate their compatibility and compare it against his friend value. Based on this comparison he will add Mary to his list of friends (if she is not already in the list). Conversely, when he meets Martha he will not add her to his list of friends since her compatibility with him is below his friend value.

A mingling is an occasion when citizens meet each other in some predetermined fashion (referred to as a mode). For example, a total mingling occurs if every citizen meets every other citizen. A partial mingling is where citizens meet each other under other some rule (e.g. only citizens with odd numbered ids, etc.).

Naturally, each citizen's list can change as the result of mingling. The following is the expected list of friends for each person in the table above after a total mingling.

John's friend list:	Mary
Mary's friend list:	John, Richard
Martha's friend list:	Richard
Richard's friend list:	Martha

Mingling is a social occasion and as such can be affected by a mood. A person's friend value can change depending on the mood of a mingling. For example, there might be a rule that if the mood is good then the person's friend value is decreased by 10%, but does not go below 1.5. That is, the person is more likely to accept new friends when in a good mood. On the other hand, the rule might that if the mood is bad then the person's friend value goes up by 10% but does not ever go above 5. Thus if John mingles in a bad mood he will drop Mary as his friend since his new friend value will be 2.2 and his compatibility with Mary is only 2.1. If the mood of a mingling is neutral then each person's friend value remains unchanged.

You are required to write software that simulates a society. The software will create the society and control the simulation by causing one or more occurrences of a mingling to occur. You are given a framework that implements the basic concepts discussed above. You will extend the framework to add your own rules and other exciting things. Have fun!

Java Frameworks

This project aims to introduce several new OOP concepts. One such concept is that of a framework. Most object-oriented programming languages allow the specification and use of frameworks. A framework is a set of classes that contain methods which can be used to solve problems of a certain type. For example, there can be a framework for student registration. Such a framework will contain partially implemented classes to represent students and courses, and methods for processes such as registration. The goal of a framework is to define a process which lets developers implement certain functions based on individual requirements while the framework manages features that are typical of the problem being solved. In other words, framework defines the skeleton and developers fill in the flesh when using it. Most frameworks will have (at least):

1. Declarations of interfaces that classes are expected to implement in order to work with other classes in the framework;
2. Concrete classes that may contain fully implemented code. In some cases concrete classes are used as provided but in most cases they are extended to add, or override functionality.
3. Abstract classes that contain a mixture of fully implemented code and abstract methods. An abstract class cannot be instantiated. Instead it must be extended to create a concrete class which can then be used to create objects.

The framework for the Game of Friends is provided as a Java package. The package contains:

- The `Person` class (from Project #1) that you will extend to create the `Citizen` class.
- The `Preference` class (from Project #1)
- An abstract class named `MetaSociety` which you will extend to create a class that models a society of citizens.

PROJECT EXERCISES

Setting Up

1. Create a java project. Call it comp1161-project2
2. Download the comp1161-project2-v1-00.zip file from the course web site. The file contains a folder with the `gof` package and a text file with sample data on citizens. You are required to add more data to this file so that you have at least 20 citizens in your society.

Note: Your classes must have the statement import `gof.*` in order to use the framework.

Part 1 – Writing the Citizen class

Write the `Citizen` class as a subclass of the `Person` class. The `Citizen` class must also implement the Comparable interface so that two citizens can be compared. The rules of comparison are given below. The following must also be done for the `Citizen` class.

1. The constructor for the `Citizen` class shall accept a person's name, sex, location, and friend value (in that order). This constructor shall create an empty friend list and shall also initialize the citizen's mortal status to alive. A unique identifier shall also be created for each citizen.
2. Write a `toString` method for the `Citizen` class to return a string containing the citizen's data and a list of the names of the citizen's friends.

3. Override the `addPreference` method so that no person can have more than 5 likes or 5 dislikes. The new version should simply not add any more items if the list to be affected already has 5 items.

4. Create the method `setMood` that accepts a single parameter of type integer and adjusts the citizen's friend value depending on the parameter value as described in the preamble.

5. Create the method `meet` that accepts a parameter of type `Object`. The parameter is another `Citizen` object. The method should cause the two citizens to meet and should add the parameter object to the receiving citizen's friend list if the appropriate conditions are met, as described in the preamble. This method should also reduce the friends list where necessary.

6. Two citizens are compared by their last names (lexicographic ordering). If the two last names are the same then their first names are compared.

7. Finally, you should write getter and setter methods for a citizen's id, mortal status, and location.

Part 2- Writing the `Society` Class.
`MetaSociety` is an abstract class. It defines some mandatory features of a simulation. For example, a simulation takes place over a number of periods. You can interpret a period to mean a year, a day, an hour, a week in a semester, or anything that fits into the story line of your simulation. **You are not allowed to modify the code in this class.**

In order to define your society you will extend `MetaSociety` class to create the concrete class `Society`. The concrete class shall have:

1. A constructor that accepts the name of the society (a string), the starting period (an integer) of the society, and the name of a text file (a string) as it parameters. The constructor shall open the file, read the data therein, create citizen objects from the input data and shall add each new object to the list of citizens in the society.

2. The method `mingle` that will implement an algorithm by which citizens meet each other. This method shall accept two parameters being an integer representing the mood of a mingling, and a second integer representing the mode of mingling. A positive value for the mood attribute indicates a good mood while a negative value indicates a bad mood, and 0 is used if the mood is neutral. The mode parameter can be used as you wish to define different types of mingling. For example, you may decide to use a particular value to restrict meetings to only those persons in the same location, etc.

3. The method `update` that effects some sort of change to each citizen. You are free to decide what rules to apply here in order to make the simulation interesting. For example, you can decide that persons who have no friends shall be sent to a specific location (e.g. solitary confinement ☺) and will only be reintegrated with the rest of the society after some defined number of ticks. Of course, this will mean that your `Citizen` class must have methods that are used to count how many periods a citizen spends in a location.

Part 3 – Building a Text User Interface

One of the principles of good software design is "separation of concerns". For example, it is desirable to separate code that allows a user to interact with an application from the code that implements the logic of the application. Most of the programs you have written up to this point have violated this principle. In many cases your driver class has been responsible for user interface tasks as well as for managing the logic of the application. This project will demonstrate building a class whose responsibility will be communicating with the user via a text menu and sending messages to the class that carries out the logic. The class, named `GameofFriendsTxtUI` shall have the following methods:

1. A constructor that accepts a `Society` object as its parameter. This object will be passed to the constructor by a driver program. The constructor will store the `Society` object as an attribute of the `GameFriendsTxtUI` object.

2. A method named `start` that can be invoked to start the user interface. You might use this method to display a menu from which the user selects a menu option. The command chosen will cause invocations of the methods of the `Society` object. Where necessary, results received from the `Society` object will be displayed for the user.

3. The menu must include (at least) options to:
    a. Cause one tick of the simulation followed by displaying the society;
    b. Cause a specified number of ticks (uninterrupted) followed by displaying the society;

Part 4 – The Driver Class

If you have not already been so informed, the `main` method is a static member of a class and should really only be used to create the objects that comprise an application, introduce them to each other, and start the processing. The following example is used to illustrate this approach.

```
public class ASmallDriverProgram {

// A lean main method. The way it should be
     public static void main(String[] args)
     {
      Society game = new Society("LIVING IN UTOPIA", 2014,"utopia.txt");
      GameOfFriendsTxtUI ui = new GameOfFriendsTxtUI(game);
      // Now run the game via the text user interface.
      ui.run();

     }
}
```

The importance of *separation of concerns* should be clear from this example. It should be easy to see how the user interface for the application could be changed to a Graphical User Interface without affecting how the game works. Think about it!

SUBMISSION INSTRUCTIONS

Please submit an executable jar file with Project #2 and Project #3 (remember to include source code) by midnight on Friday April 10, 2015 then attend a demonstration session during the week starting the 12th of April for grading.