Team Assignment 1

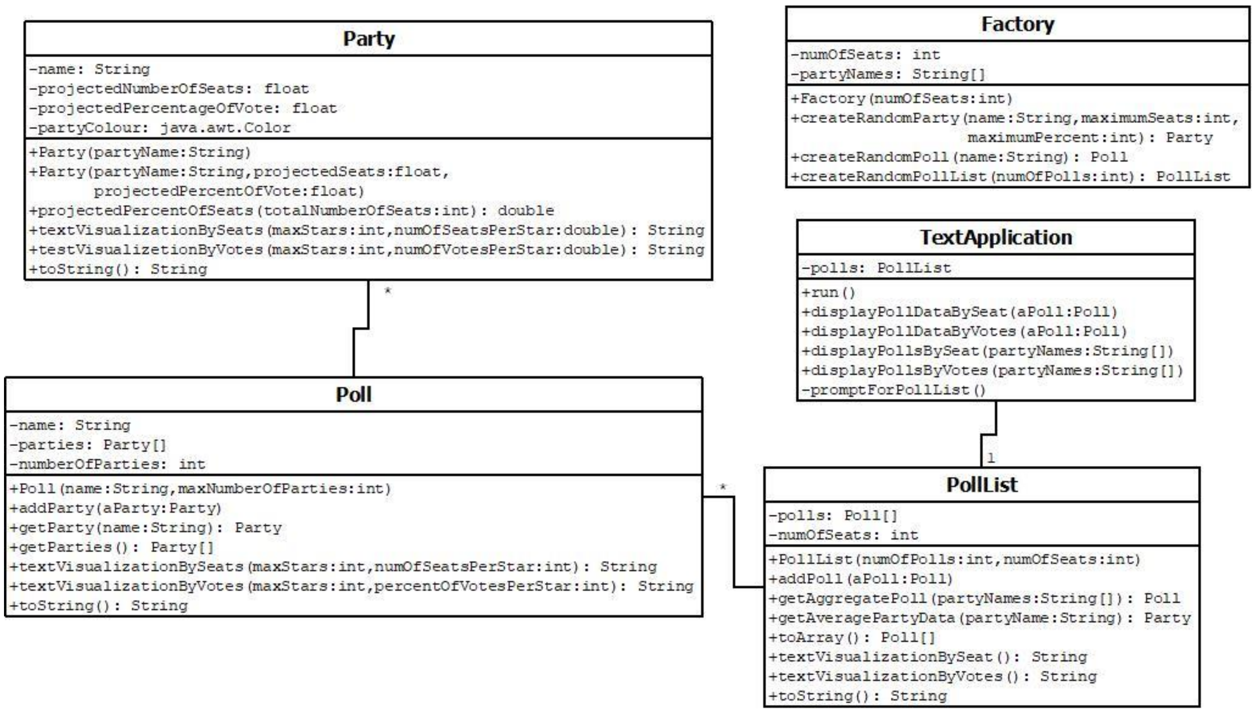
For the team assignment you'll need to combine the Party, Poll, and PollList classes that your team completed, and the Factory class provided by the instructor to create a Poll Tracker text application. In a text application, the user interacts with the application through the console: The program prompts the user for some input, and then waits until the user has entered the information.

Your team will be creating the TextApplication class.

Note that for the second individual and team assignments, the goal is to create a Poll Tracker application with a graphical user interface. You will use the same classes, but you'll add another user interface. In a graphical user interface, the application opens a window where the user interacts with the application through widgets such as buttons, text fields, sliders, etc.

Overview of Requirements

The following shows all the classes that will be included in the Poll Tracker text application. The only class that still needs to be created is the TextApplication class. It will use the Party, Poll, and PollList classes that your team created, and the Factory class provided by the instructor. Make sure to work together, each of you is an expert in a different class. Clear and frequent communication is needed to prevent code duplication.


Party
-name: String
-projectedNumberOfSeats: float
-projectedPercentageOfVote: float
-partyColour: java.awt.Color
+Party(partyName:String)
+Party(partyName:String,projectedSeats:float, projectedPercentOfVote:float)
+projectedPercentOfSeats(totalNumberOfSeats:int): double
+textVisualizationBySeats(maxStars:int,numOfSeatsPerStar:double): String
+testVisualizetionByVotes(maxStars:int,numOfVotesPerStar:double): String
+toString(): String
*
Poll
-name: String
-parties: Party[]
-numberOfParties: int
+Poll(name:String,maxNumberOfParties:int)
+addParty(aParty:Party)
+getParty(name:String): Party
+getParties(): Party[]
+textVisualizationBySeats(maxStars:int,numOfSeatsPerStar:int): String
+textVisualizationByVotes(maxStars:int,percentOfVotesPerStar:int): String
+toString(): String
*
Factory
-numOfSeats: int
-partyNames: String[]
+Factory(numOfSeats:int)
+createRandomParty(name:String,maximumSeats:int, maximumPercent:int): Party
+createRandomPoll(name:String): Poll
+createRandomPollList(numOfPolls:int): PollList
TextApplication
-polls: PollList
+run()
+displayPollDataBySeat(aPoll:Poll)
+displayPollDataByVotes(aPoll:Poll)
+displayPollsBySeat(partyNames:String[])
+displayPollsByVotes(partyNames:String[])
-promptForPollList()
1
PollList
-polls: Poll[]
-numOfSeats: int
+PollList(numOfPolls:int,numOfSeats:int)
+addPoll(aPoll:Poll)
+getAggregatePoll(partyNames:String[]): Poll
+getAveragePartyData(partyName:String): Party
+toArray(): Poll[]
+textVisualizationBySeat(): String
+textVisualizationByVotes(): String
+toString(): String

Note that there are some small changes in this diagram from the one that was included in the individual assignments in the Factory and TextApplication classes.

Managing Team Tasks

Through the individual assignments, each of you was able to work independently on your own class. You will now be able to put this together into an application. Working on a single class as a team can be challenging: all code must go in a single file. We strongly recommend the following approach:

Do part one of the requirements as a team. These methods will likely be very short if you use the classes you created correctly. Input from all three team members is important to ensure the appropriate methods are called.

Once part one is completed, you should each have enough insight in the Party, Poll, and PollList classes to divide the remaining tasks amongst yourselves if you prefer to continue working on the team assignment alone. There are benefits to continuing to work together and there are benefits to dividing the work between you.

If you work on step 2 of the requirements together, all three of you will be very familiar with each class in the application and the final product won't need any additional effort to combine the work.

If you divide the tasks in step 2 of the requirements you can work in parallel and likely get the different parts completely quicker, but it will take debugging effort to add the three different components to a single class and have them work together correctly.

In either approach, the team should start working together synchronously. I strongly recommend that you use class time for the first synchronous meeting.

Grading

The team solution will be evaluation on how well the five classes are combined into a single, cohesive solution. (The five classes are the three created by individual teammates, the one developed as a team and the one provided class.) Make sure to:

- Avoid code duplication.
- Have a consistent coding style between classes. This includes how braces and parenthesis are placed, the use of white space, and consistent naming of variables.
- Have a consistent documentation style between classes.

Consider the feedback provide by your TA for guidelines related to this coding style. The same TA will mark your team submission, so make sure to ask for clarifications from your TA directly.

The solution your team submits must be the work of you and your team. Do not ask for help to complete your project from anyone other than your teammates and course instructors.

Team grading breakdown is as follows. All team members receive the same grade, unless extensive communication with all team member indicates differential grading is needed.

- TextApplication
    - (2 points) Functionality: prompts the user for input as required, uses the Factory class to generate random data, and displays requested data to the end user.
    - (2 points) Code quality.
    - (2 point) Documentation (javadoc and in-line)

- (2 points) All classes work together and the project has all the required functionality. There is no duplication of code over multiple classes. TextApplication accurately uses the other four classes to create the application.
- (2 points) All classes use the same coding conventions and have similar
    - Naming conventions
    - Spacing conventions
    - Placement of braces
    - Documentation conventions

Detailed Requirements for TextApplication Class

This class represents the entire application and pulls together all other classes. It uses the Party, Poll, PollList, and Factory classes to create an application. Make sure that you don't duplicate the code in these four classes.

The TextApplication class is an application and we should be able to run this class, in other words, it should have a main method. When the class is run, the user is prompted for information related to parties and polls, and it will display information about parties and polls.

Note that this class does not have any automated tests. To test this method, you need to run the class and ensure it is behaving as required.

Part 1

When developing this class as team start by implementing this first set of requirements as a group. (Class time is set aside for this or the team can meet outside of class time.)

The main method in this class should

1. create an instance of TextApplication using the default constructor, and then
2. call the method run to get the entire application running.

The **instance variable** *polls* of type PollList should be declared private

- *polls* of type PollList

No constructors are needed for the class.

Add the following **public methods** to the TextApplication class. (These are also noted in the class diagram above.)

- *displayPollDataBySeat* which does not return anything and takes a Poll as an argument. It prints a visualization of the data in the poll provided as an argument to allow the user to compare the parties. For example, a visualization for a poll named 'Poll1' with 6 parties may look as follows:

```
Poll1
*******      |           Liberal (40% of votes, 85.86417 seats)
*******      |           Green (35% of votes, 80.175026 seats)
***          |           PPC (9% of votes, 42.0 seats)
***          |           NDP (10% of votes, 35.877453 seats)
***          |           CPC (3% of votes, 33.0 seats)
             |           Rhinoceros (0% of votes, 0.0 seats)
```

To test this method, you can use the Factory to create a random Poll for you and use it to call this method from the main method. (This is for testing only.)

- *displayPollsBySeat* which does not return anything and takes an array of Strings as an argument. It should display all the Poll objects that are in the PollList instance variable *polls* followed by the aggregate of the polls in the PollList instance variable. (Use the visualize method in PollList for all the Poll objects in *polls* and the visualize method in Poll for the aggregate poll. The PollList class has the method to get the aggregate poll. This method needs the list of party names.) For example, it may display the following when the application has 3 polls and 6 parties.

```
Poll0
*************|*           Liberal (56% of votes, 154.11794 seats)
*******      |            CPC (33% of votes, 81.92139 seats)
***          |            Green (5% of votes, 40.0 seats)
             |            NDP (4% of votes, 1.0 seats)
             |            Rhinoceros (0% of votes, 1.0 seats)
             |            PPC (0% of votes, 0.0 seats)

Poll1
*******      |            Liberal (40% of votes, 85.86417 seats)
*******      |            Green (35% of votes, 80.175026 seats)
***          |            PPC (9% of votes, 42.0 seats)
***          |            NDP (10% of votes, 35.877453 seats)
***          |            CPC (3% of votes, 33.0 seats)
             |            Rhinoceros (0% of votes, 0.0 seats)

Poll2
*************|********    CPC (88% of votes, 236.83212 seats)
**           |            Green (8% of votes, 25.0 seats)
*            |            Liberal (3% of votes, 14.0 seats)
             |            NDP (0% of votes, 2.0 seats)
             |            PPC (0% of votes, 0.0 seats)
             |            Rhinoceros (0% of votes, 0.0 seats)

Aggregate
**********   |            CPC (41% of votes, 117.25117 seats)
*******      |            Liberal (33% of votes, 84.6607 seats)
****         |            Green (16% of votes, 48.391674 seats)
*            |            PPC (3% of votes, 14.0 seats)
*            |            NDP (4% of votes, 12.959151 seats)
             |            Rhinoceros (0% of votes, 0.33333334 seats)
```

To test this, you can initialize the *polls* instance variable using the Factory to get a random list of polls.

- *run* which does not return anything and does not take any arguments. It should run the entire application. It will prompt the user for information and then uses the visualization methods you just created to display the polls created.

The application should first prompt the user for the following information:
  - The number of seats available in the election.
  - The parties that are running in the election.
  - The number of polls to track

Then ask the user if they want to enter the data for all the polls or if they want you to generate a random set of polls.   For now, only respond to the request to generate a random set of polls.

Make sure to call the appropriate method in the Factory class to generate a set of random polls. Again, do not duplicate the code from the Factory class. Just use the result returned by the Factory class whether the result is correct or not.

Once the polls are generated, ask the user which visualization they want, allowing the user to quit any time. Users can visualization the aggregate poll or all polls in the application.

An interaction with the user may look as follows:

```
Welcome to the poll tracker
How many seats are available in the election? 280
Which parties are in the election (provide names, comma separated):
CPC,Green,Liberal,NDP,PPC,Rhinoceros
How many polls do you want to track with this application? 3
Would you like me to create a random set of polls? yes

Options: all (show result of all polls), aggregate (show aggregate result), quit (end application)
Choose an option: aggregate

Aggregate
**********    |             CPC (41% of votes, 117.25117 seats)
*******       |             Liberal (33% of votes, 84.6607 seats)
****          |             Green (16% of votes, 48.391674 seats)
*             |             PPC (3% of votes, 14.0 seats)
*             |             NDP (4% of votes, 12.959151 seats)
              |             Rhinoceros (0% of votes, 0.33333334 seats)

Options: all (show result of all polls), aggregate (show aggregate result), quit (end application)
Choose an option: all
```

You are not required to manage or check for invalid user input. You will do this through the graphical user interface you'll create for the next iteration.

Step 2

At this point, decide if you want to divide the remaining tasks between the three team members so you can continue development in parallel or if you want to continue developing as a group. The complete the following remaining tasks:

1. Add functionality that prompts the user for all information for the poll list. This should be placed in the method *promptForPollList.*
    a. Prompt for each poll name.
    b. For each poll:
        i. Prompt the user for the expected number of seats for each party.
        ii. Prompt the user for the expected percentage of the vote for each party.
    c. Make sure that each of these polls is placed in the poll list.
2. Give the user the option to visualize the data by seats or by votes.

3. Add code that implements the display by votes. (The display by seats has already been implemented in step 1.)

If you do decide the divide the work between the three team members, do note that the 3 points are not a good way to divide the work. Carefully consider the work required for each and divide the work accordingly.

You may add any private methods that support the public methods that you wish. (You will likely have to break the run into multiple methods to ensure the amount of code in run is reasonable.) If a method gets very long and no longer fits on a screen (including a laptop screen using a reasonable font) you should create such helper methods that are declared private.