Requirements for Individual Assignment 2 – Version 2

You'll continue with the application you started in the first assignment. This means you need a correct version of the classes Party, Poll, PollList, and TextApplication. If there are errors in any of these classes, they will have to be fixed before continuing with this application for assignment 2. There is also an updated Factory object. See below for more information on this update.

For this second iteration, each team member will create a portion of the graphical user interface (GUI) of the project by creating a view using FXML and SceneBuilder and connecting this view with the logic code created for iteration 1 by creating a controller class.

You are responsible for creating a window that allows the user to update information about all the polls and parties in the application.

Getting Started

1. Create a new JavaFX project in Eclipse.
2. Create three packages in this new project and name them: model, view, and application.
3. Put the Party, Poll, and PollList classes from assignment 1 in the model package.
4. Put the TextApplication class from assignment 1 in the application package.
5. Import the updated Factory to the model package.

You will see a few errors in the TextApplication class that need to be fixed.

- The Party, Poll, and PollList classes need to be imported since they are now in a different package than TextApplication. You can add all three at once using the statement *import model.*;*
- Every line of code that creates a new instance of the Factory class will have to call the getInstance method instead. The invocation of the constructor will have to be replaced with Factory.getInstance() instead.

Provided Classes

Factory is updated such that it implements the singleton design pattern. This means making the constructors private and creating a static variable that contains the single instance and a static method that returns the single instance.

Since the entire application now contains a single instance of the Factory class, it can be used to share information between the different components of the user interface. You'll notice that information you receive from the user and that is needed by the entire application will be maintained by the Factory class. (This does mean it isn't truly a factory anymore however.)

Collaboration and Academic Misconduct

If your team decides to continue to work together for the second team assignment, you may collaborate with your team members to complete this individual assignment. There is one place where you may collaborate with others outside your team for this iteration: the challenge that is unique to your part of the code. This unique challenge is clearly identified for each version of the individual assignment. Do not share code for the assignment directly. Instead, share 'proof of concept' code that helps you and others learn how to accomplish the task.

For all other parts of the assignment, the line between collaboration and academic misconduct is the same as Assignment 1.

You can find further information with assignment 1 and on D2L.

Files to create

EditPollView: This may be an FXML file or it may be in a .java file if you write the code in JavaFX directly.

EditPollController.java: This controls what is displayed in the view and ensures the PollList object received by the Factory is updated with the information entered by the user.

EditPollTester.java: This is a class that extends javafx.application.Application and which you can run to test your code. It should load the FXML file (or create an instance of your view class and controller directly).

Files to edit

In addition to moving classes to a different package and getting the updated version of the Factory class, change the PollList class to add a setPollName method. It should change the pollName instance variable. Also change the Party class such that it imports Color from javafx.scene.paint.Color instead of importing it from awt. This will ensure you can use the colour in the visualization in your window. Make the same update in the PartyTest class if needed.

Challenges

All challenges for this version are related to managing the information in the dropdown boxes:

- Getting the initial PollList in the *Poll to edit* dropdown box.
- Updating the *Party to update* dropdown box after a Poll was selected.
- Refreshing the dropdown boxes to contain the most recent information.

You are allowed to collaborate with classmates when figuring out how to complete these challenging requirements of the assignment. Do not share code for the assignment directly. Instead, share 'proof of concept' code that helps you and others learn how to accomplish the task.

Testing your code

When you run the EditPollTester class, the window should pop-up that allows the user to edit the data in existing poll objects. One of your teammates for assignment 2 will allow the user to enter information about the number of seats, polls, and the parties in the election. For testing purposes, use the default information that is available in the Factory class instead and use the Factory to generate a list of polls to work with.

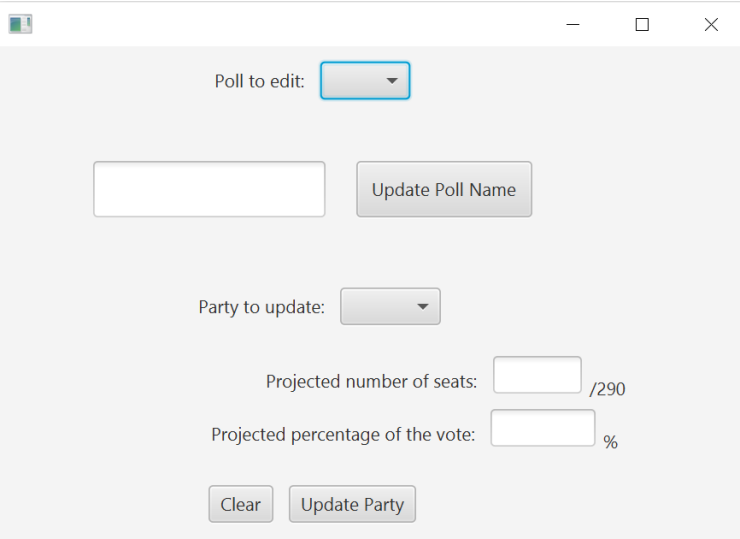
Requirements

This view allows the user to edit the information in the polls that the application is tracking. At startup (or on initialization):

- The application should ask the Factory for an empty PollList. (Get the single instance of the Factory using the Factory.getInstance() call. Then get an empty PollList by calling the *createEmptyPolls()* method.) Place the result of this call in an instance variable in the controller so all methods in the controller have access to this list of polls.
- Use the generated list of polls to populate all required information into the view.

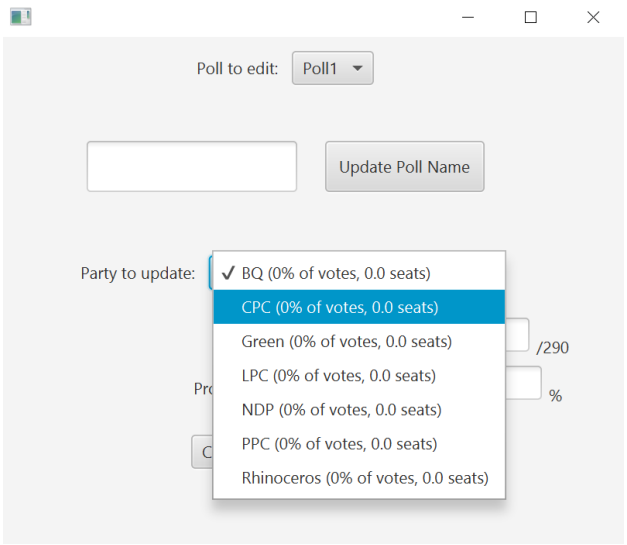
    - The list of poll names in the dropdown list for the polls.
    - The list of party names when a poll is selected from the dropdown list.
    - The accurate number of seats that are available in the election.
- Any changes the user specifies for any poll should be reflected in the PollList instance variable.

The information they can edit for a poll is the name of the poll and, for each party in the poll, the expected number of seats and expected percentage of the votes. This view may look something like the following. (Note that the /290, indicates that the election is out of 290 seats in this example. Make sure to replace this number with the number of available seats in the PollList instance variable.)



The user changes a poll name by selecting a poll in the *Poll to edit'* dropdown box, entering a new name in the textfield and then pressing the *Update Poll* Name button. (You will need to add a 'setPollName' method to the Poll class.)

The user changes party data by first selecting a *Poll to edit* then selecting a *Party to update.* This means that once the user has selected a poll in the top dropdown list, the information for all the parties in that poll should be placed in the 'Party to update' dropdown box. For example:



The user can then select any of the parties in this second dropdown box and edit the data in this party (in the selected poll.) In this example, the user could select Green and set the percent of the votes to 30% and the number of seats to 9.

After pressing the Update Party button, the Green party information should have changed in the Poll. You can test this by selecting a different poll and/or party. Then returning to the poll/party you updated to make sure these updates are still visible.

Do make sure to make all changes in PollList, Poll, and Party objects. Do not duplicate the code from these classes and do not hard code any of this. Your teammates for assignment 2 need the updated PollList, Poll, and Party objects.

Code Quality

All code submitted will be graded on documentation, legibility, and quality. See requirements for assignment 1 for additional details.

Grading

Individual grading is for your controller class and FXML file. The test JavaFX application class that is for testing purposes only and will only be used to grade functionality: We will not consider the quality of the code or the documentation of that class. Your classes should compile and it should be possible to run your test application class on its own. **Code that does not compile or can't be run is not worth any marks and will not be graded.** The code that you created will be marked as follows:

- Functionality:
  - (1 point) View has all needed components/widgets for required functionality.
  - (1 point) View and controller are linked via appropriate IDs, instance variables, naming of controller and methods that react to actions. (If programming in JavaFX directly, view and controller are in separate classes and linked appropriately through methods.)
  - (1 point) Controller ensures application correctly responds to user input.
  - (1 points) Singleton instance of Factory class state is updated appropriately to ensure data is visible throughout the application.
- Code quality:
  - (1 point) Variable names and IDs are self-documenting and code is easy to read and understand. Good use of white space and all methods fit on a single (small) screen (both height and width).
  - (1 point) Code duplication is avoided and minimized; code is well organized over methods; no unnecessary instance variables.
  - (1 point) Code is well organized and easy to maintain: nesting deeper than three levels is avoided; multiple return statements are avoided; break and continue are never used.
- Documentation:
  - (1 point) Class and all public methods are fully javadoc'd with appropriate tags and content.
  - (1 points) Good use of in-line documentation to explain blocks of code.