Project Group Num. : #3
Project Group Title : Non-Profit Donation
Project Group Members : Weipeng Cao
Zhi Li
James Huynh
Ming Liew

Project Part 5: Final Report

1. *List of features that were implemented (table with ID and title).*

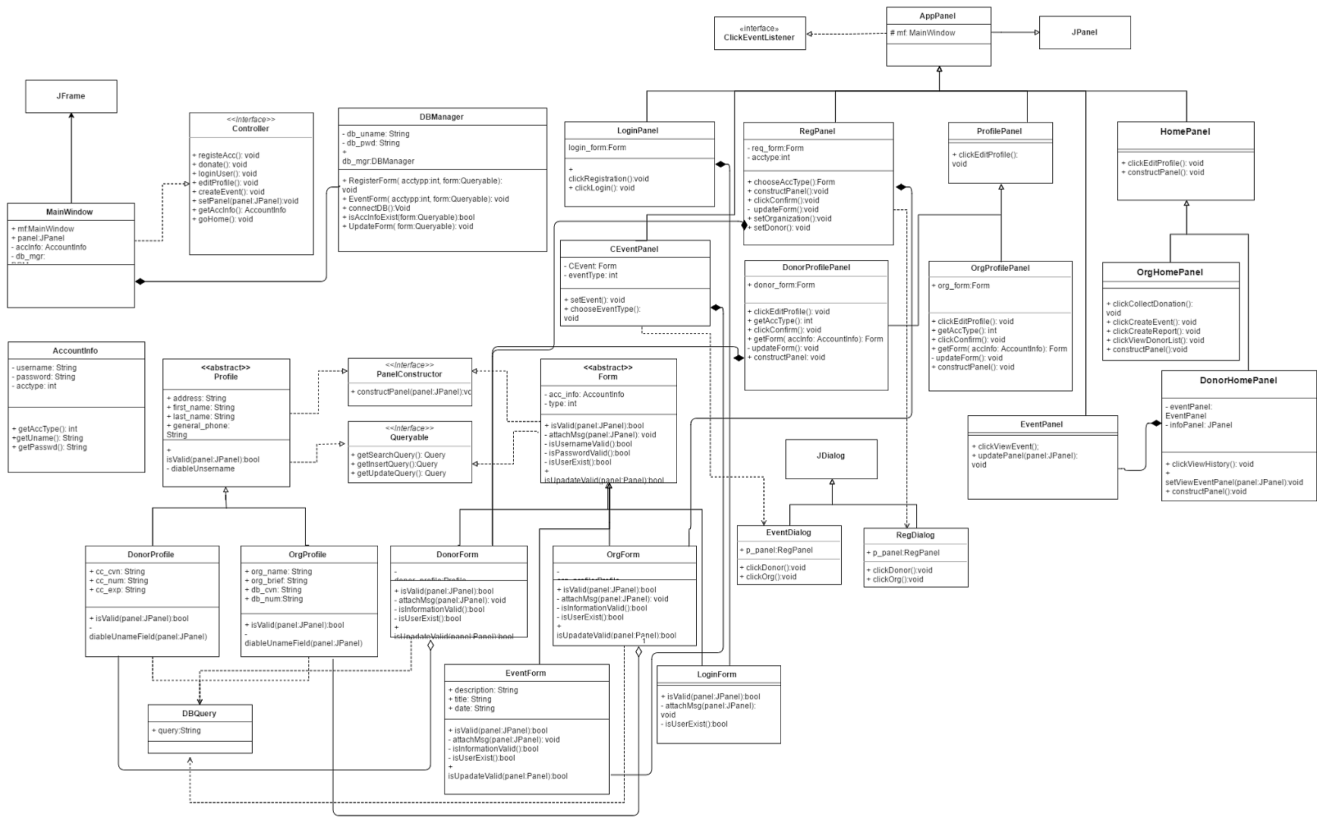
User Requirement ID	Title
(UR-01)	Donor Login
(UR-02)	Organization Login
(UR-05)	Make Donation
(UR-07)	Create Event
(UR-10)	Edit Account Information

2. *List the features were not implemented from Part 2 (table with ID and title)*

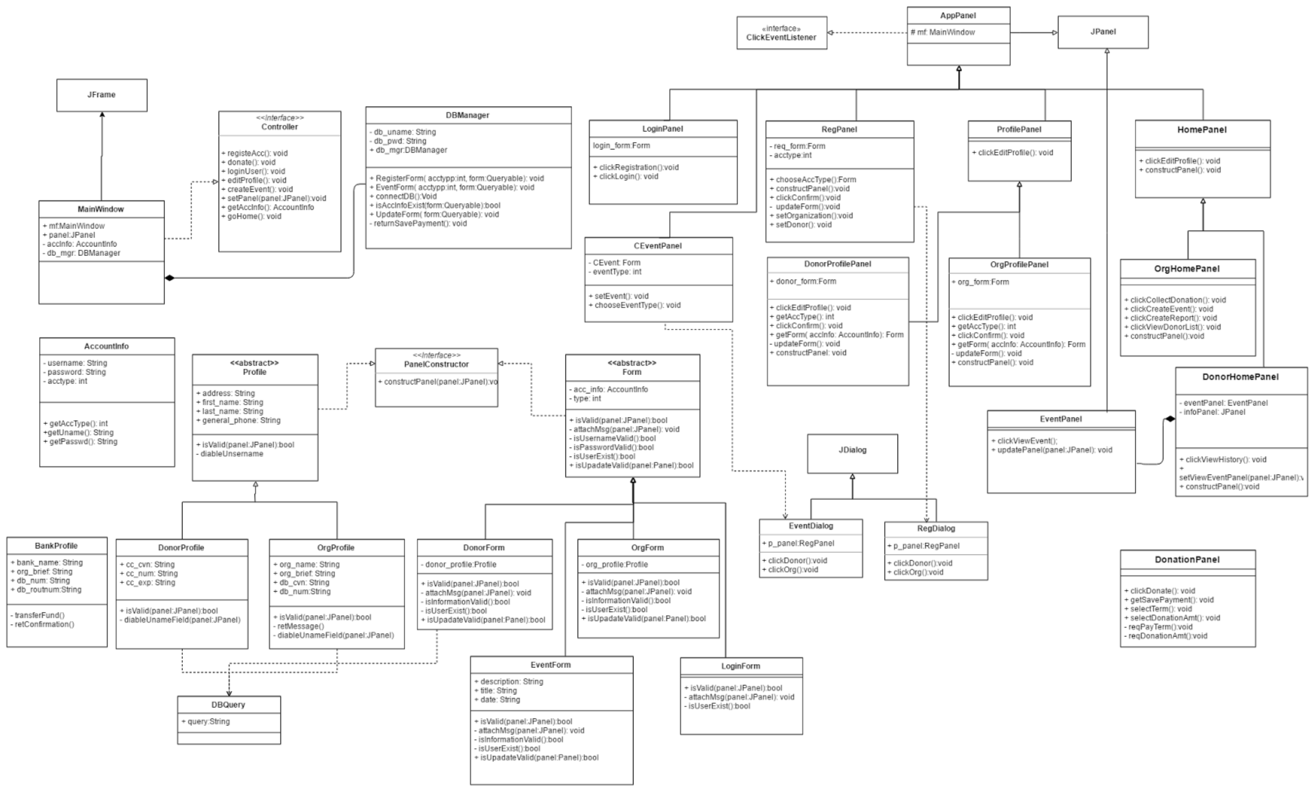
User Requirement	Title
(UR-03)	Event Search
(UR-04)	Save Payment Method
(UR-06)	View Interest
(UR-08)	View Who Donated
(UR-09)	Print Event Detail
(UR-11)	Collection Donation
(UR-12)	Message Forwarding

3. *Show your Part 2 class diagram and your final class diagram. What changes? Why? If it did not change much, then discuss how doing the design up front helped in the development.*
   https://drive.google.com/open?id=0BxRL6onnzGo9WWtxRmtJcU5HV3c

<u>Part 2 Class Diagram:</u>



Final Class Diagram:

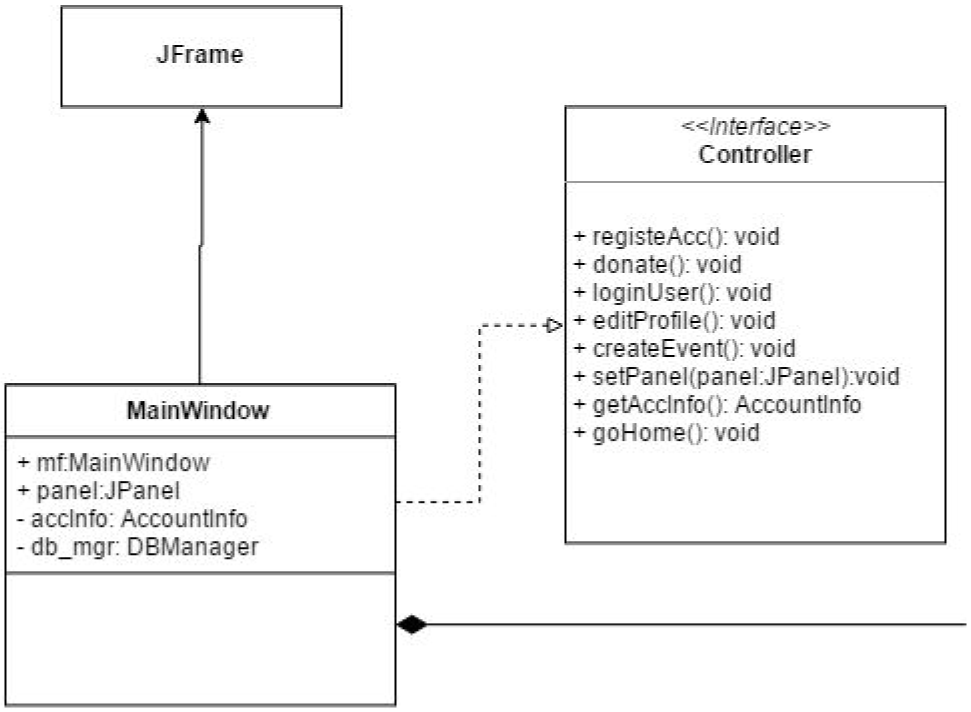


Our final class diagram didn't change much from Part 2 of the class diagram. The class diagram in Part 2 is designed based on the MVC architecture. The View Classes inherited JPanel or JDialog and the model classes inherited forms are highly decoupled. In opposite, these two types of objects are coupled again due to the data binding so that the data can be passed to the relevant forms for further processing. In addition, due to the complexity of the fancy GUI design more intermediate classes, not shown in the final class diagram, are added to make the code cleaner. Finally, more methods were implemented into most of the classes because some functions or features were not fully considered during planning.

4. *Did you make use of any design patterns in the implementation of your final prototype? If so, how? Show the classes from your class diagram that implement each design pattern (each design pattern as a separate image in the .pdf). If not, where could you make use of design patterns in your system? Show a class diagram of how you could implement each design pattern and compare how it would change from your current class diagram.*

Yes, we used some of the design patterns in the implementation of our final prototype. They are listed in the table below:

Class	Design Patterns
MainWindow	Singleton, Strategy, State
Form	Strategy, Chain of Responsibilities
Profile	Chain of Responsibilities
AppPanel	Strategy, Composite(inherited from JPanel)
FocusAdapter	Adapter



Example Class: MainWindow

- Singleton
    - The MainWindow is an attribute in MainWindow itself
    - Cycle is live as long as the application runs
- Strategy
    - The content panels shown on the frame are created and attached to the frame in the fixed sequence of methods
- State
    - The MainWindow provide the methods to construct the next view when a relevant operation is done in current view. The implementation of the methods in this class is dominated to control what the next view of the application should be.

5. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

Designing the project with UML doesn't only make the project more understandable easily to the client (if any) but also lets the team members or team to understand the scope of the project more easily to enhance the communication. The documentation and the diagrams present the everyone involved to the project the ideas; each implementation, for example the use case of registering an account, can be found on the user case diagram and documented what should be complete with activity diagram and the details in the use case document. The UML benefits us, as developers, in several aspects. With the use case documentation and the use cases the scope of the project is more clear so that we can bring up more ideas and the potential implementations for achieving the scopes; that is, we had a rough plan for the individual use case. The last aspect is that we were able to view the overall from the sequence diagrams and the class diagram, e.g. interaction between the individual works.

The most important benefit from design patterns is that it offers us some sufficient pattern to design the classes to make our application more maintainable. The chain of responsibilities design pattern is used for Form and Profile Classes in our program and provide the maintainability of the program. The Panels that has a form doesn't need to care about how to create the sub panels, but the Form class then Profile Class taking care the sub panel creation. The State pattern, with which the MainWindow class is designed, provides the flexibility to the program. The controller on each MVC just need to take the logics of current views and model, having cleaner code. As more side effects are needed, the relevant implementation can be insert/add to the MainWindow Class.