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).*

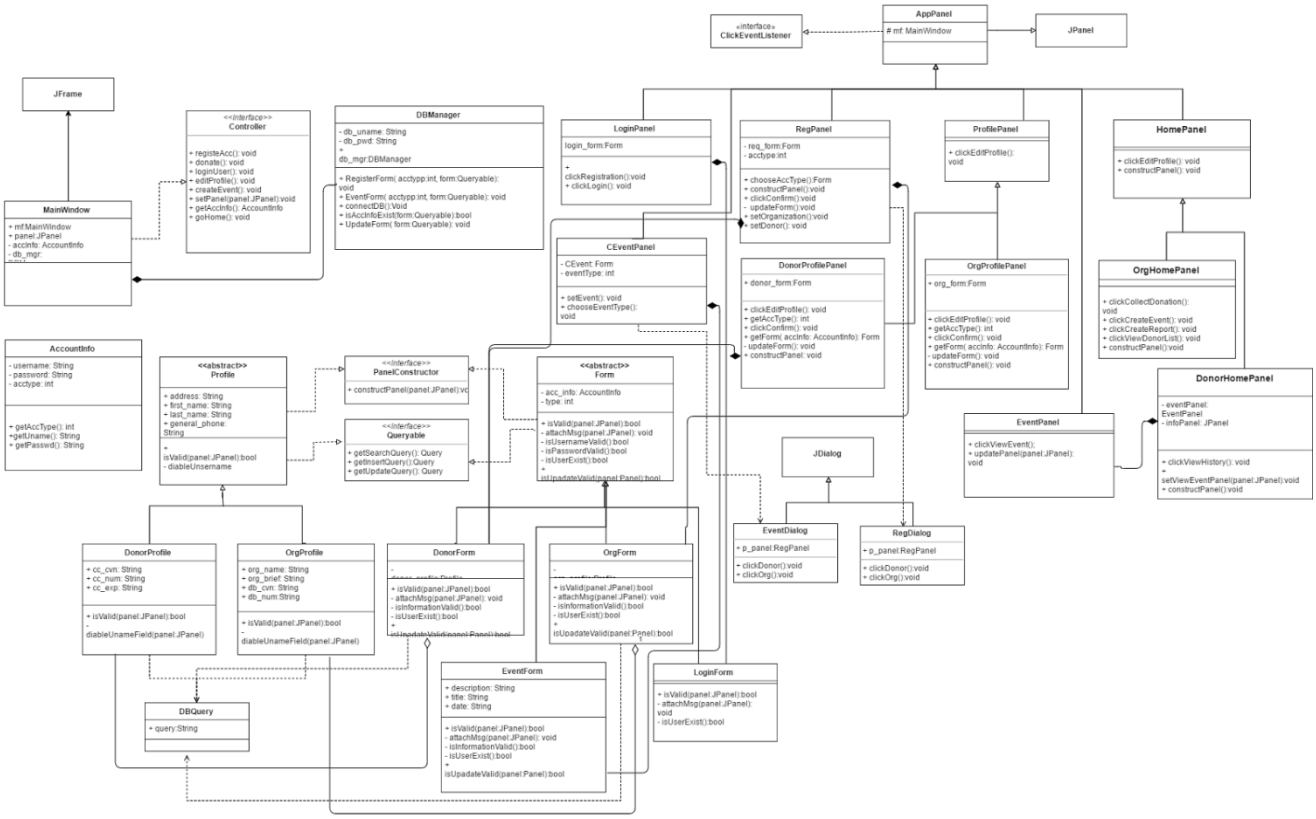
User Requirement ID	Title
(UR-01)	Donor Login
(UR-02)	Organization Login
(UR-05)	Make Donation
(UR-07)	Create Event
(UR-10)	Edit Account Information

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

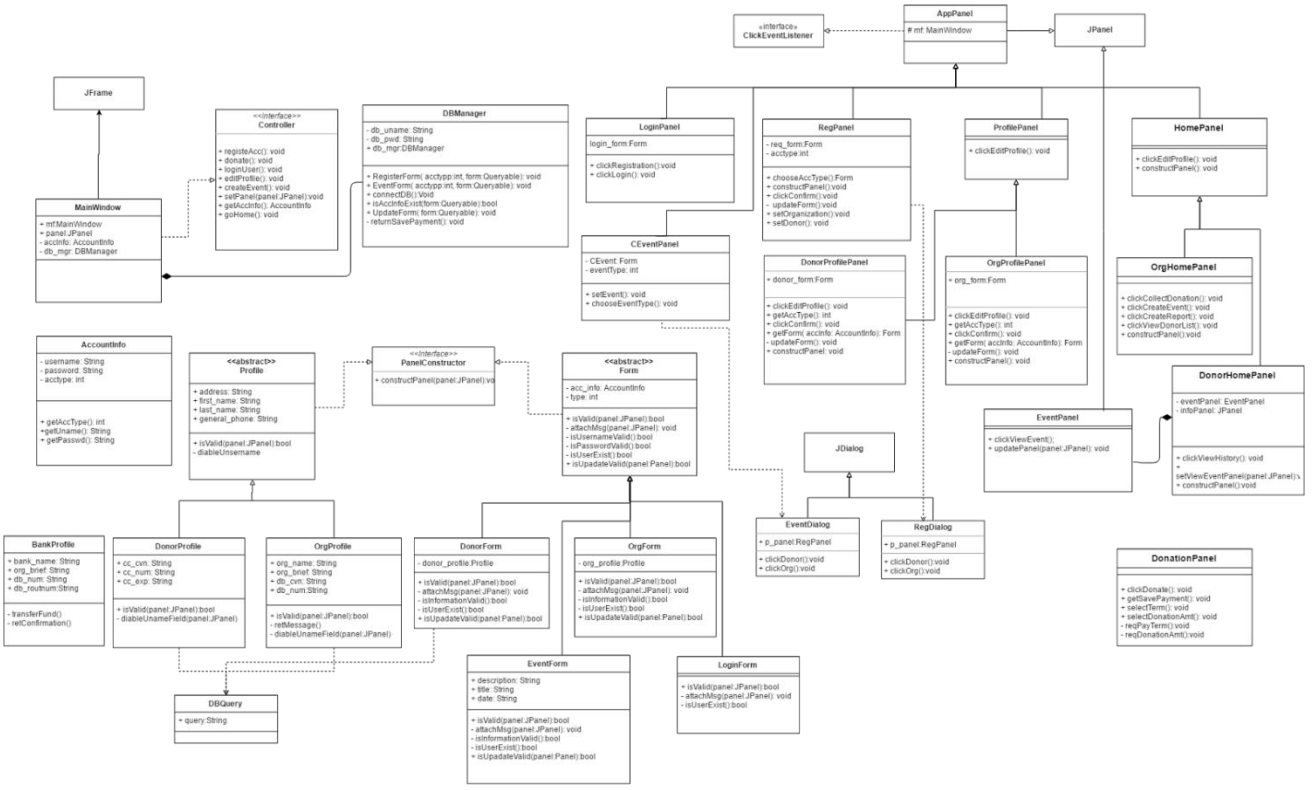
User Requirement	Title
(UR-03)	Event Search
(UR-04)	Save Payment Method
(UR-06)	View Interest
(UR-08)	View Who Donated
(UR-09)	Print Event Detail
(UR-11)	Collection Donation
(UR-12)	Message Forwarding

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

Part 2 Class Diagram:



Final Class Diagram:

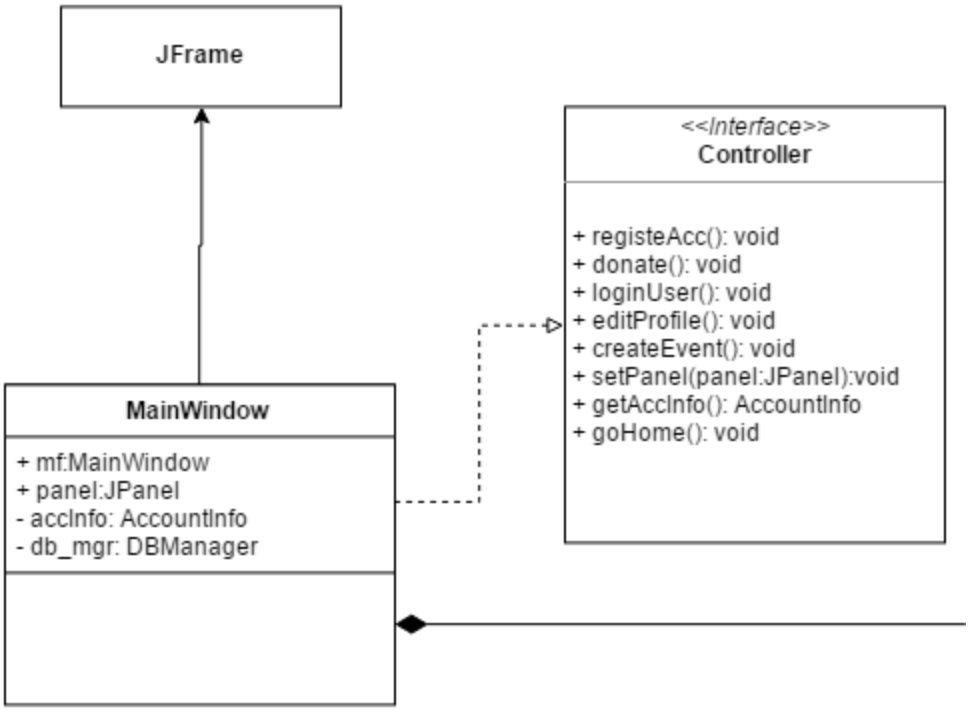


Our final class diagram didn't change much from Part 2 of the class diagram. The class diagram in Part 2 is designed based on the MVC architecture. The View Classes inherited JPanel or JDialog and the model classes inherited forms are highly decoupled. In opposite, these two types of objects are coupled 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 diagrams not only made the project more understandable to the client, if any, but also allow the team and its members to understand the scope of the project more better which in turn provide enhanced communications and better projects. The documentation and the diagrams of the UML allows everyone involved to see what the project is about and how it's going to be implemented. Each implementation, for example, the use case of registering an account can be found on the user case diagram which describes how the user should interactive with the system when registering and the activity diagram shows how the use case should be implemented. The UML diagram benefits us, as developers, in several aspects. The first is the use case documentations and the use cases; they allow the scope of the project to be much clearer for the developers. This allows the developers to have the

ability to plan ahead on how they want to handle each use cases for the project. The last aspect is that the developer can see the overall design of how the project should behave and how it should be implemented from the class diagram and sequence diagrams.

The most important benefit of design patterns is that it offers a 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 which provides the maintainability of the program. The Panels that has a form doesn't need to handle or create the sub panels. While the Form class and then later the Profile Class does have to handle 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 only needs to handle the logic of the current views and models which result in cleaner code. As more cases are needed, the relevant implementation can be inserted or added to the MainWindow Class.