Loan Prediction and Processing System

Team: Chutney
Manan Khasgiwale and Tavishi Priyam

Final State

The final state of the loan prediction system consists of a webapp that offers login, loan prediction and loan sanction services.

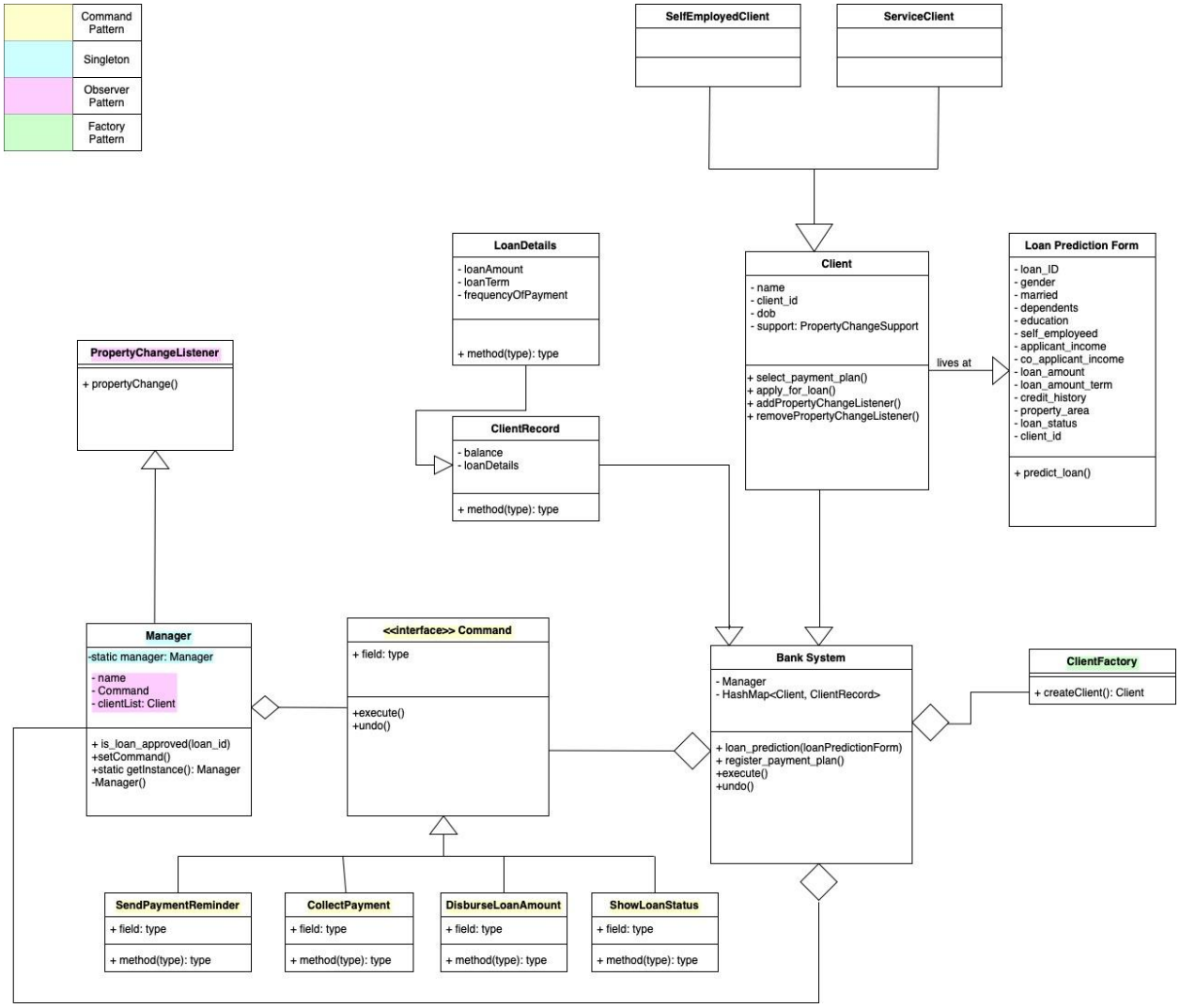
- Users fill out the loan prediction form to get a prediction.
- Loan is approved by the bank manager.
- Once approved, the loan is sanctioned.
- Client status page displays the client details and remaining balance
- Regular reminders are sent to the clients.
- Client pays off the loan through periodic payments.
- Once the loan is paid in full, the system terminates for that particular client.

There are three stakeholders in this model: Client, Bank, Bank Manager
We are implementing four object oriented design patterns: Observer, Command, Factory and Singleton.
The webapp is integrated with a postgresql database.

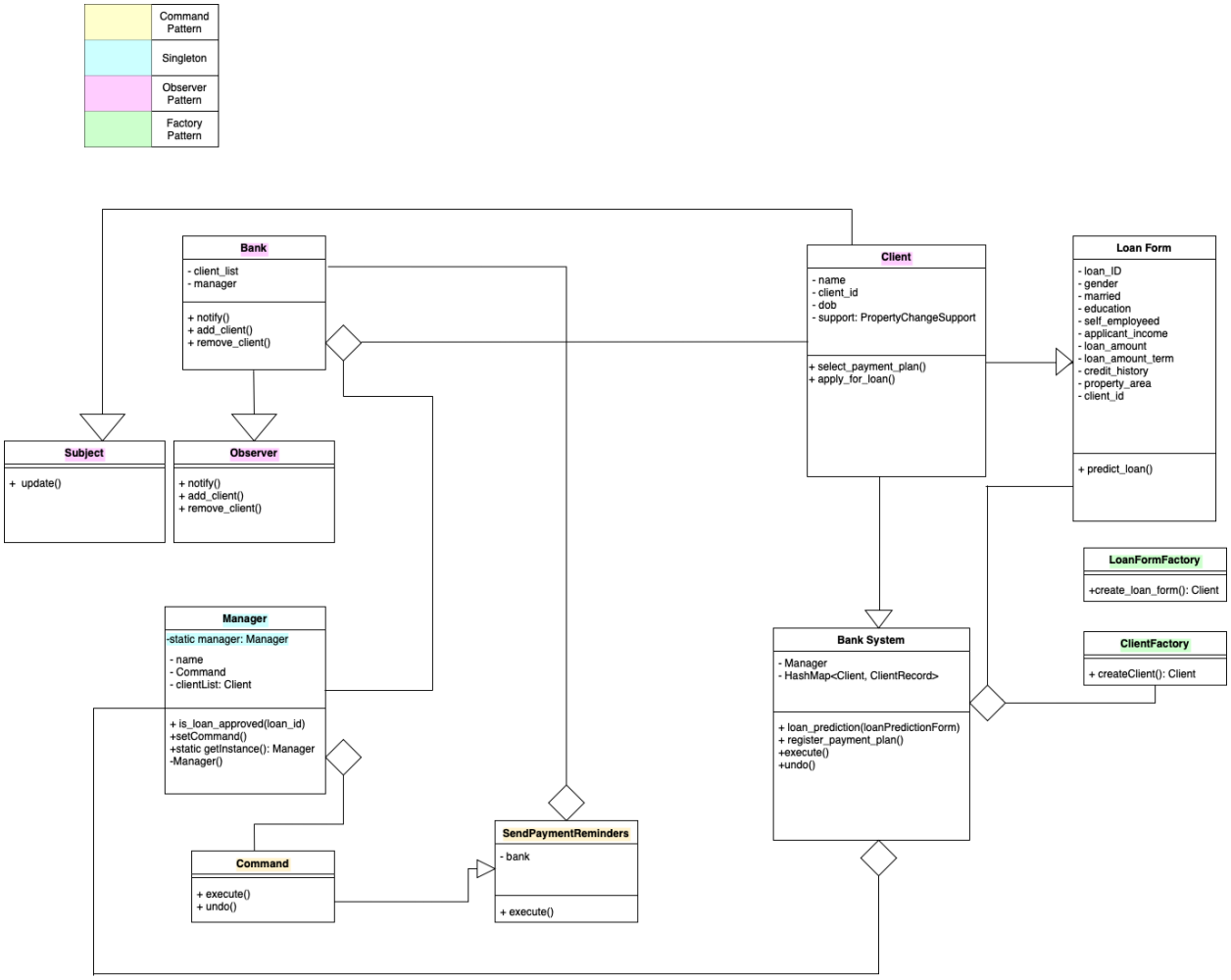
Final Class Diagram & Comparison

After the project 5, we have made the following updates to the class diagram:

- Implementation of command pattern
- Removing PropertyChangeListener and adding an observer and subject class.
- Adding a predictor class to run the prediction model on client data.
- Removing service and self employed clients
- Adding LoanFormFactory
- Created bank class
- Implemented flask app routing



UML Class Diagram - Project 5



UML Class Diagram - Project 7

Third-Party code vs. Original code statement

Most of the code is written by us. We used external help from tutorials to create the react app, integrate it with flask and the postgresql database.
In addition to this, we referred to a couple of tutorials explaining the implementation of the design patterns in Python.

- https://reactjs.org/tutorial/tutorial.html
- https://towardsdatascience.com/build-deploy-a-react-flask-app-47a89a5d17d9
- https://blog.miguelgrinberg.com/post/how-to-create-a-react--flask-project
- https://refactoring.guru/design-patterns/command/python/example
- https://python-gino.org/docs/en/master/tutorials/tutorial.html#
- https://www.tutorialspoint.com/postgresql/index.htm
- https://gist.github.com/Kartones/dd3ff5ec5ea238d4c546
- https://www.postgresqltutorial.com/psql-commands/
- https://www.geeksforgeeks.org/get-post-requests-using-python/
- https://stackoverflow.com/questions/18860098/on-a-browser-sessionstorage-in-safaris-private-browsing-does-not-work-the-same/18861484
- https://www.reddit.com/r/flask/comments/jvr1oq/how_to_redirect_to_a_different_website_in/
- https://www.tutorialspoint.com/flask/flask_sessions.htm
- https://www.youtube.com/watch?v=j942wKiXFu8&list=PL4cUxeGkcC9gZD-Tvwfod2gaISzfRiP9d&index=1&ab_channel=TheNetNinja

Statement on the OOAD process

Some design elements that impacted our development process positively were

- Class Diagram
  Having drawn a structured class diagram before starting the development process provided clarity for the entire process. We kept referring to this diagram to guide us through the coding.
  Despite having made some changes, we have mostly remained true to our initial design for the project.

- Activity Diagram
  In addition to the class diagram, the other UML diagrams such as the activity and the sequence diagrams also helped in structuring our code the way we wanted it. The UML representations helped us develop an OO thinking process.

- OO Concepts
  We have made use of key OO concepts such as abstraction, encapsulation, polymorphism and inheritance. It helped a lot with making the code clean, modular, easy to understand/reuse/modify.

Some issues we faced during the development process were

- We had previously only seen and implemented the design patterns in Java. Translating these to Python and discovering the features Python offers to facilitate the implementation of the design patterns was intriguing yet challenging.

- We are working with a react frontend and python backend with a postgresql database. Integrating these and querying data seemed to be a little challenging.

- Implementing the command pattern concurrently with the flask app running also proved to be a little difficult.