Password Management Application

Jie Dai

Today is World Password Day!

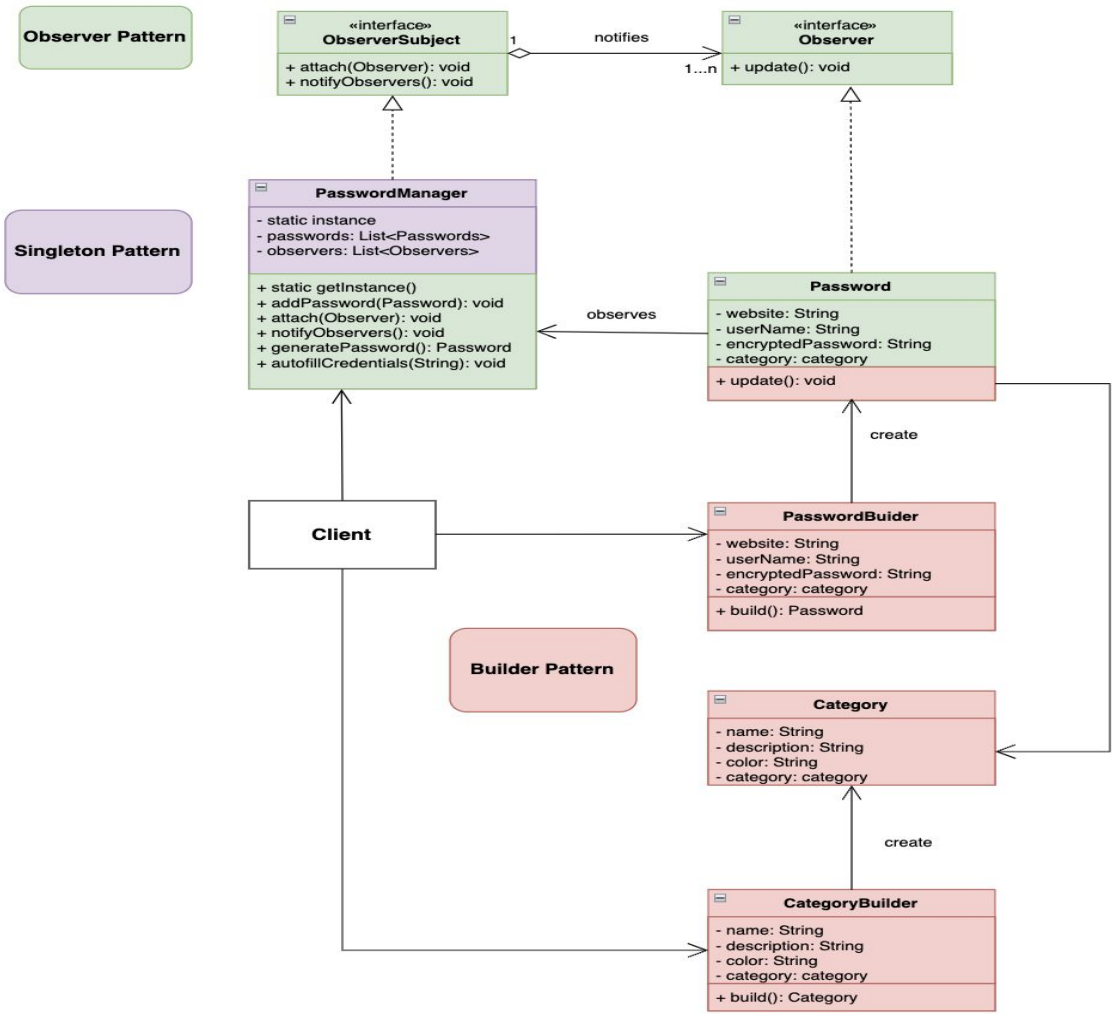
Intention

The **Password Manager** is a software application that allows users to store and manage their login credentials securely.

- Users can create, view, update, and delete login credentials for various websites
- Users can categorize their login credentials by assigning them to categories with custom names and colors
- Users can generate random passwords
- Users can autofill their login credentials when visiting websites or applications.

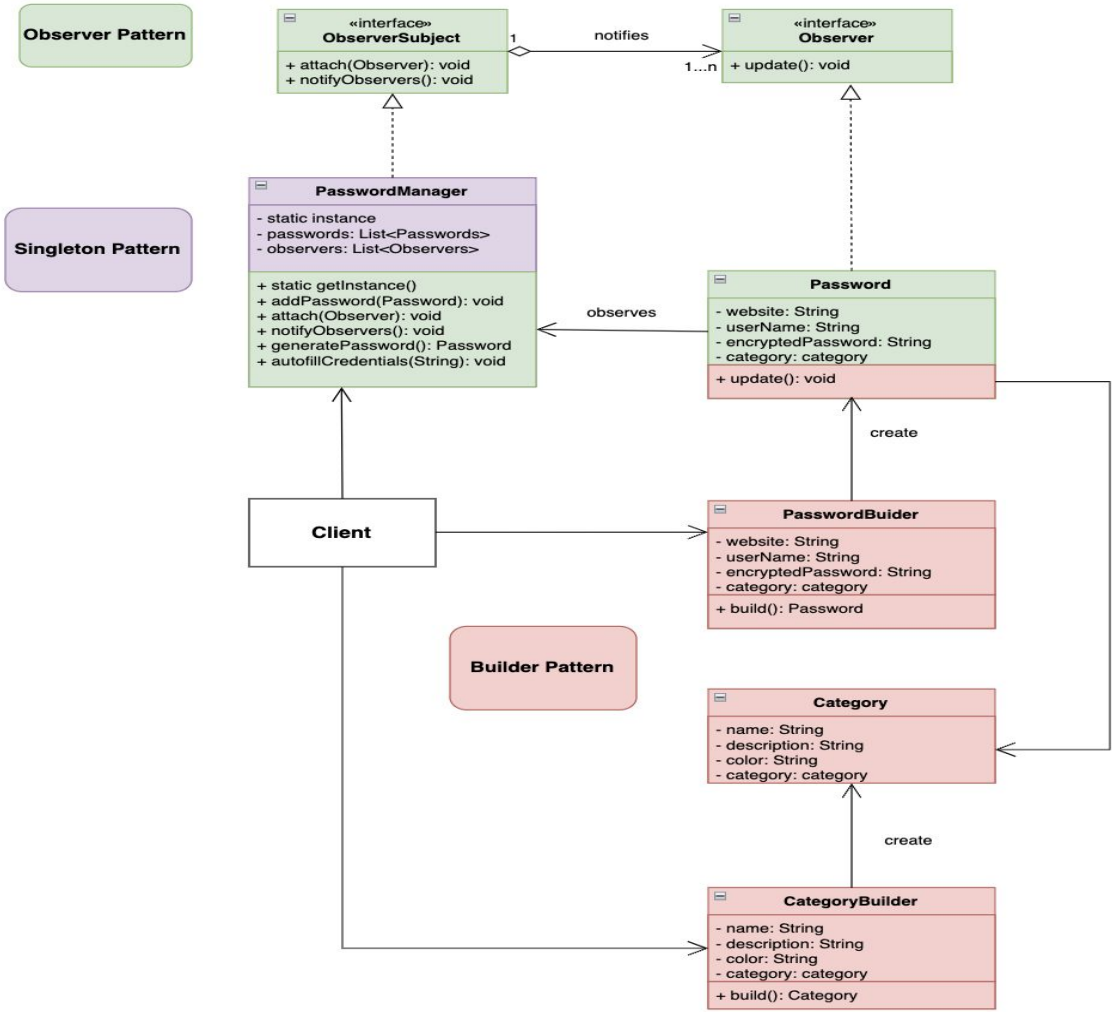
Design Patterns

- Builder Pattern
- Singleton Pattern
- Observer Pattern



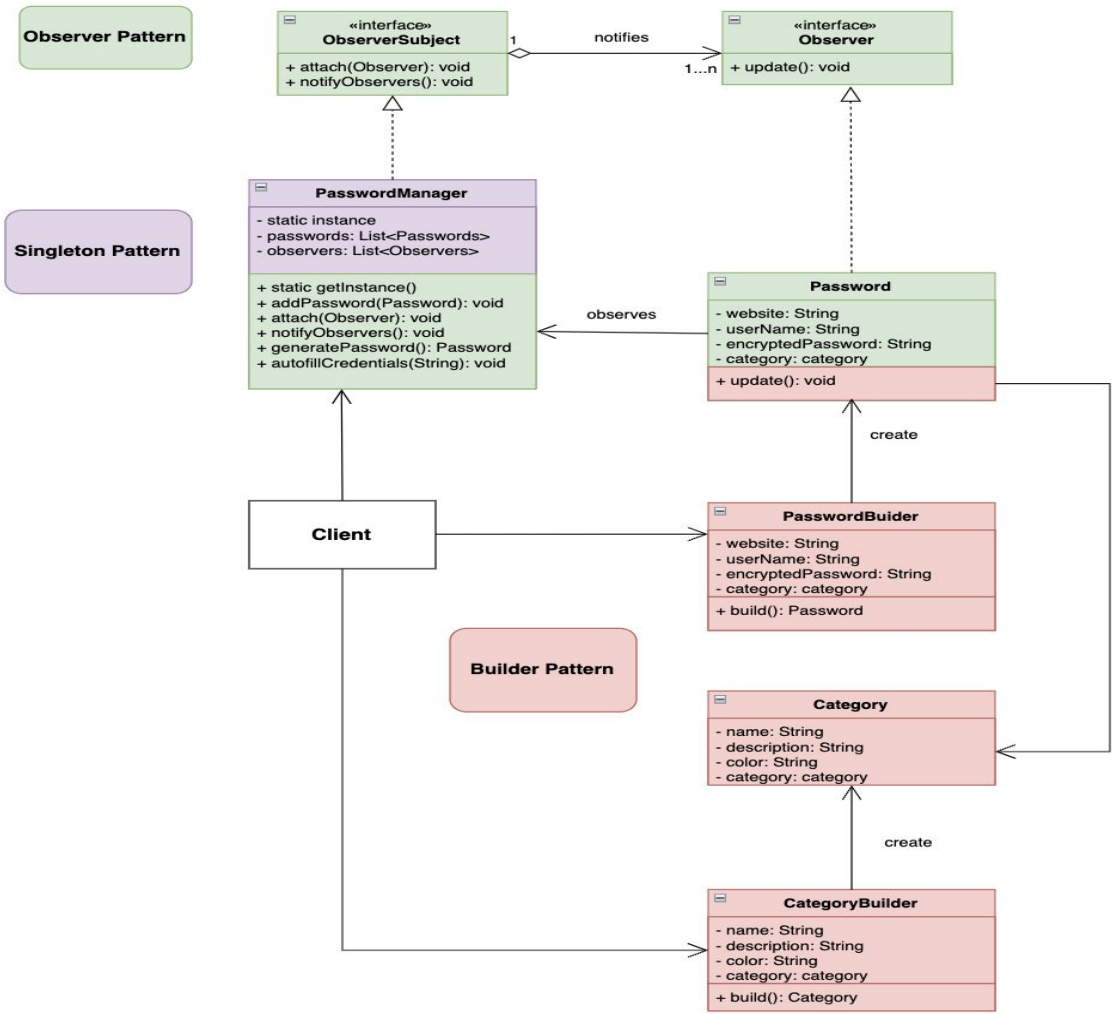
- **Builder Pattern**

The Builder pattern is used to create Password and Category objects with custom parameters, without having to specify them in the constructor.



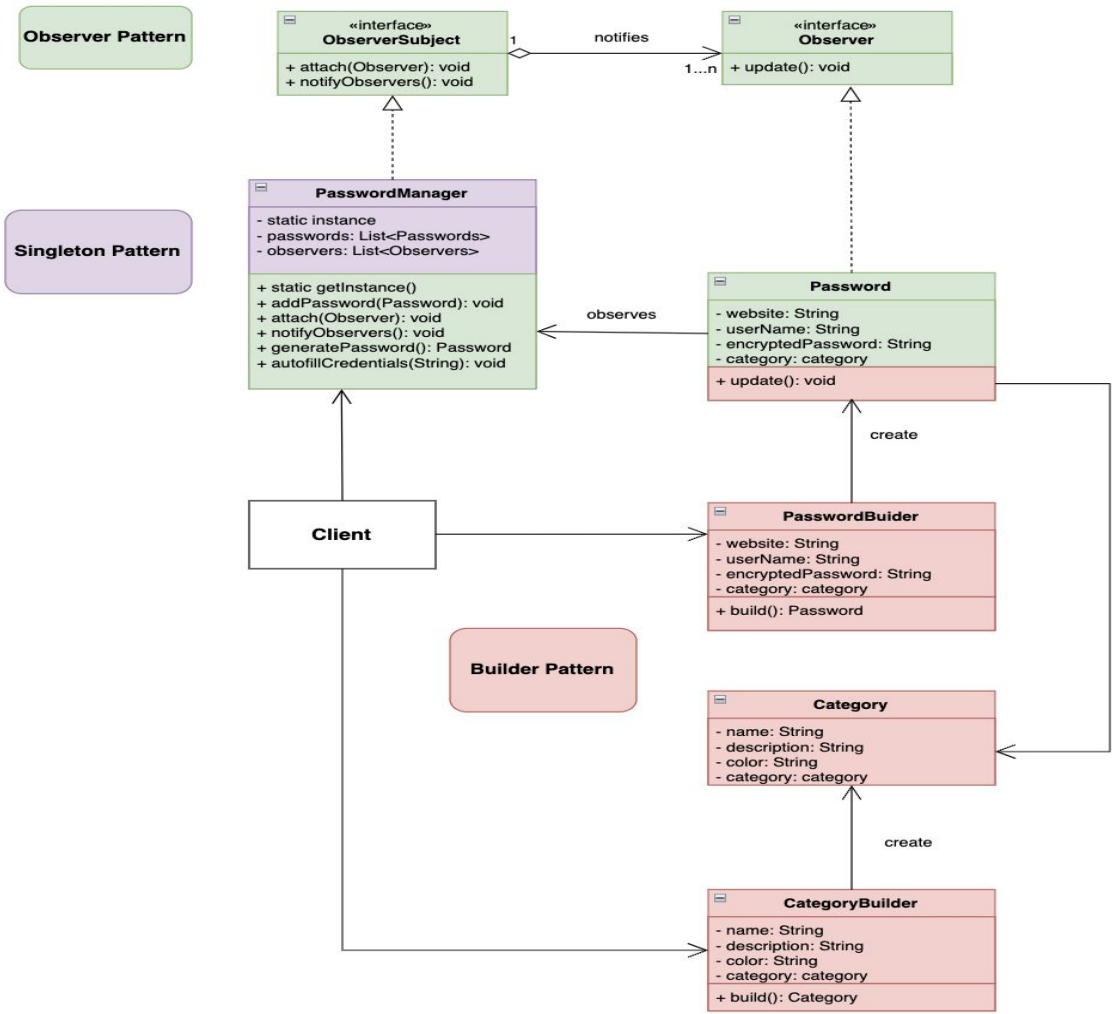
- **Singleton Pattern**

The Singleton pattern is used to ensure that there is only one instance of the PasswordManager class in the application. The PasswordManager class is the central class that manages the storage and retrieval of passwords.



- **Observer Pattern**

The Observer pattern is used to notify Password objects when the PasswordManager changes. When the PasswordManager changes, it notifies all Password observers, which can then update themselves accordingly.



Demo

Pros and Cons

Builder pattern: By separating the object creation logic from the object itself, the Builder pattern makes it easier to modify or extend the object creation process without affecting the object's functionality.

Singleton pattern: By using the Singleton pattern, we can ensure that the PasswordManager is easily accessible from anywhere in the application, without having to worry about creating multiple instances or managing shared state. The Singleton pattern can make the code harder to test, since the PasswordManager cannot be easily replaced or mocked in unit tests. Additionally, if multiple threads access the PasswordManager simultaneously, there may be performance or concurrency issues that need to be addressed.

Observer pattern: The Observer pattern allows for the Password objects to be updated automatically when the PasswordManager changes. This can help to keep the Password objects up-to-date with the latest changes in the PasswordManager, and can also help to reduce the amount of code required for manual updates. The Observer pattern can introduce additional complexity to the code, since the Password objects need to register themselves as observers and implement the update() method. Additionally, if the number of observers is very large, the performance of the application may be impacted.

Thank you for your time and attention!

May the 4th be with you!