**A picture containing logo

Description automatically generated**

**Object-Oriented Design Patterns**

**Bank System Project**

**Mohamed Shasha**

**B00138205**

# Application overview

This app resembles a simple banking system where users can create different banking accounts and be able to apply debit or credit cards to each account; users will be able to transfer money between accounts.

users can deposit money and withdraw money if they are account is sufficient.

# Functional requirements

* ­­­the app should make unique instance of each bank account object and associate it with account number
* The app should keep track of account balance each time amount withdrawn or deposited into the bank account
* The app should constraints user to withdraw more than the balance in the bank account.
* The app should not allow user to transfer from the selected bank account to the same bank account Instead, the list will show allow created accounts but not the selected one
* App will constraint user form issuing more than one debit or credit card for each account
* App should notify user of amount deposited or withdrawn in a message dialog



# User requirements:

System GUI is easy to use and consistent. users will be notified with message dialog for whatever functionality of system they shall use. To enhance navigation decorator is pattern was used to color button whenever hovered on.

Errors and exceptions shall be presented to user in error message dialogs to make sure that user can avoid faults when they reuse system.

Graphical user interface, application

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# Design patterns

1. **Singleton**

Singleton pattern is used to make sure that only one instance of debit or credit card object is

Created. The CreateCard class will link the account to cards by using the static method of type Card class and will create an instance of either CreditCard or DebitCard classes depending on the type passed from the GUI by the user.

Both credit and debit card classes inherit the CardType method from card abstract class and override it depending on the type of class invoked.

Whenever a CreditCard or DebitCard object is created the instance variable will be set to true to make sure that if the app called class creation again the exception will be thrown to prevent the app from invocation the object

A screenshot of a video game

Description automatically generated

# Decorator Pattern

The decorator pattern is a structural pattern that helps in drawing a red border on menu buttons without the need to create subclasses for graphical features (red border) to be applied to certain components (menu button); so that whenever the mouse cursor enters the boundaries of a button that passed to the CoolDecorator class from the GUIAccount Class the repaint method will draw a red border around that button, and the whenever mouse cursor is out it will call the base repaint method (no red border button). The CoolDecorator class is derived from Jcomponent class that works like a container and its layout should match the layout of the panel that contains the buttons.

**Text

Description automatically generated**

# O**bserver pattern**

Observer pattern is a behavioral pattern that provides a mechanism to notify all observers objects about a subject they have registered their interest in when the event related to that object is triggered. Even though the pattern here does not have a strong use case, where it would be more useful if the app has an admin that needs to send policy or warnings updates to all users at once. In our example, both accounts saving and checking are subject classes that have methods to add observers and notify them when the status changes like deposit state and withdrawal state. The observer (Notifier abstract class) couples the observer with the subject and implements the Observer interface and overrides its update method where the subclasses of the notifier class take certain behavior based on the event passed (Anon., 2019)

**Text

Description automatically generated**

# Façade pattern

The Façade pattern is a structural pattern that provides a single-entry point to the subsystem and provides simplicity to the client and information hiding. in this project, the façade class (BankServices) encapsulates services and wraps them to facilitate users to create accounts, deposit, withdraw and transfer money without handling class instantiation or dealing with subsystem complexity where the façade class will handle user requests to the right account object. and since CheckingAccount and SavingAccount classes implement the AccountInterface, the façade class will wrap these classes under its methods to present it to the user as a simpler way to interact with the system (Anon., 2020)

**Graphical user interface

Description automatically generated**

# References

Anon., 2019. *design-patterns/observer.* [Online]   
Available at: https://refactoring.guru/design-patterns/observer  
[Accessed 24 April 2022].

Anon., 2020. *Facade Design Pattern With Example.* [Online]   
Available at: https://www.c-sharpcorner.com/article/facade-design-pattern-with-example/  
[Accessed 24 April 2022].