

Student Name: Nicolae Casian

Student ID: B00144312

Module: Object Oriented Design Patterns

## 1. Application Overview

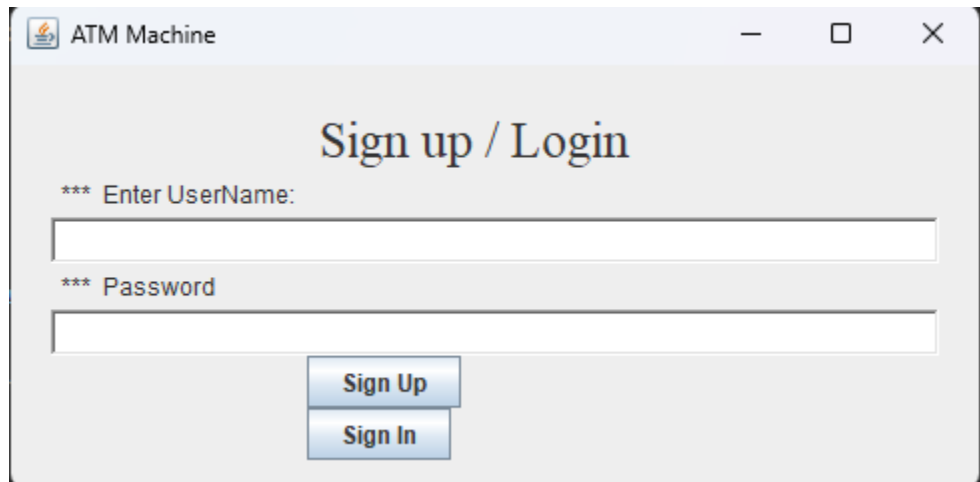
- This is a Java based ATM system that comes equipped with a full graphical comprehensive interface made with Java Swing and AWT. Users can create an account, log in, and manage their bank accounts. Some key features include depositing funds, making withdrawals and transferring money between accounts. Every account keeps a transaction log that can be viewed or exported.
- This application, uses object-oriented principles and incorporates four key design patterns. A creational, structural and behavioural pattern to ensure efficiency, maintainability and scalability.

## 2. Functional Requirements

1. Login Management: Sign-up, login, logout
2. Account/User Management: Create account, Delete Account
3. Transaction Management: deposit, withdraw, transfer
4. Transaction History: Record each transaction in a txt file

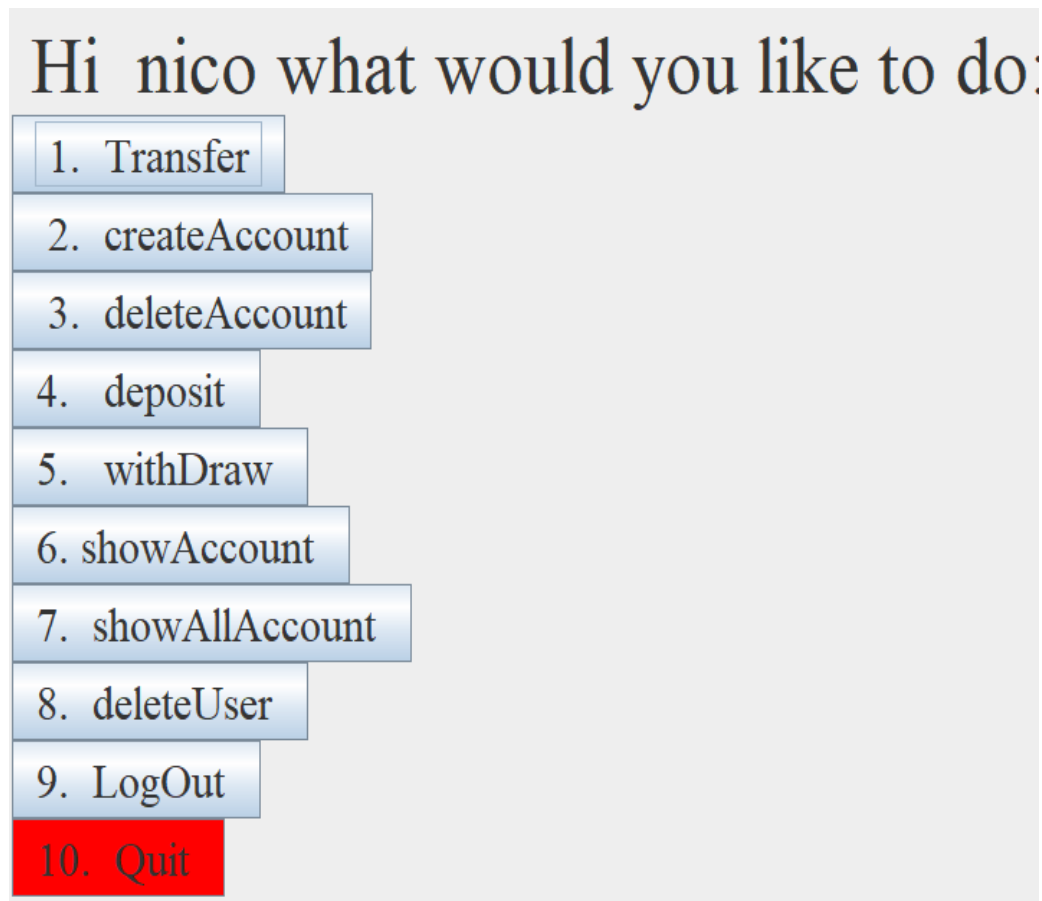
### 3. User Requirements / User Interface

1. Login/SignUp UI: Text fields for username and password and 2 sign in and signup buttons



The screenshot shows a window titled "ATM Machine" with a standard Windows-style title bar (minimize, maximize, close buttons). The main content area has a light gray background. At the top center, the text "Sign up / Login" is displayed in a large, dark blue serif font. Below this, there are two text input fields. The first field is preceded by the label "\*\*\* Enter UserName:" in a small, dark blue font. The second field is preceded by the label "\*\*\* Password" in the same font. Below the password field, there are two blue buttons with white text: "Sign Up" and "Sign In", stacked vertically.

2. Menu UI: A vertical list of buttons each having different functions



The screenshot shows a menu interface on a light gray background. At the top, the text "Hi nico what would you like to do:" is displayed in a large, dark blue serif font. Below this text is a vertical list of 10 blue buttons with white text, each containing a number and a function name. The buttons are stacked vertically, with each subsequent button shifted slightly to the right. The buttons are: "1. Transfer", "2. createAccount", "3. deleteAccount", "4. deposit", "5. withDraw", "6. showAccount", "7. showAllAccount", "8. deleteUser", "9. LogOut", and "10. Quit". The "10. Quit" button is highlighted with a red background.

3. Prompts/Dialogues: Dialogues for different function e.g Picking account type.

Creating Account

Create your Account : nico

Select your account type :

☒ Checking

☐ Saving

Amount to be added:

AccountNumber (opt):

Create Account

## 4. Design Pattern Explanation

### 1. Simple Factory pattern

- Used for creating and managing checking and savings accounts. This encapsulates the logic of creating a concrete AbstractAccount instances behind a factory method. This avoids the scattering of CheckingAccount and Savings Account around my code.
- This pattern is useful for scalability purposes if in the future I were to add a Business Account or a Fixed Deposit Account I would simply update the factory method.

### 2. Decorator Design Pattern

- The intent of the Decorator Design Pattern is to attach responsibilities to an object without affecting the other objects. I wanted to add fee and logging without creating too many subclasses. So its responsibility is for each decorator to handle fee or logging making the maintenance easier.

### 3. Builder Design Pattern

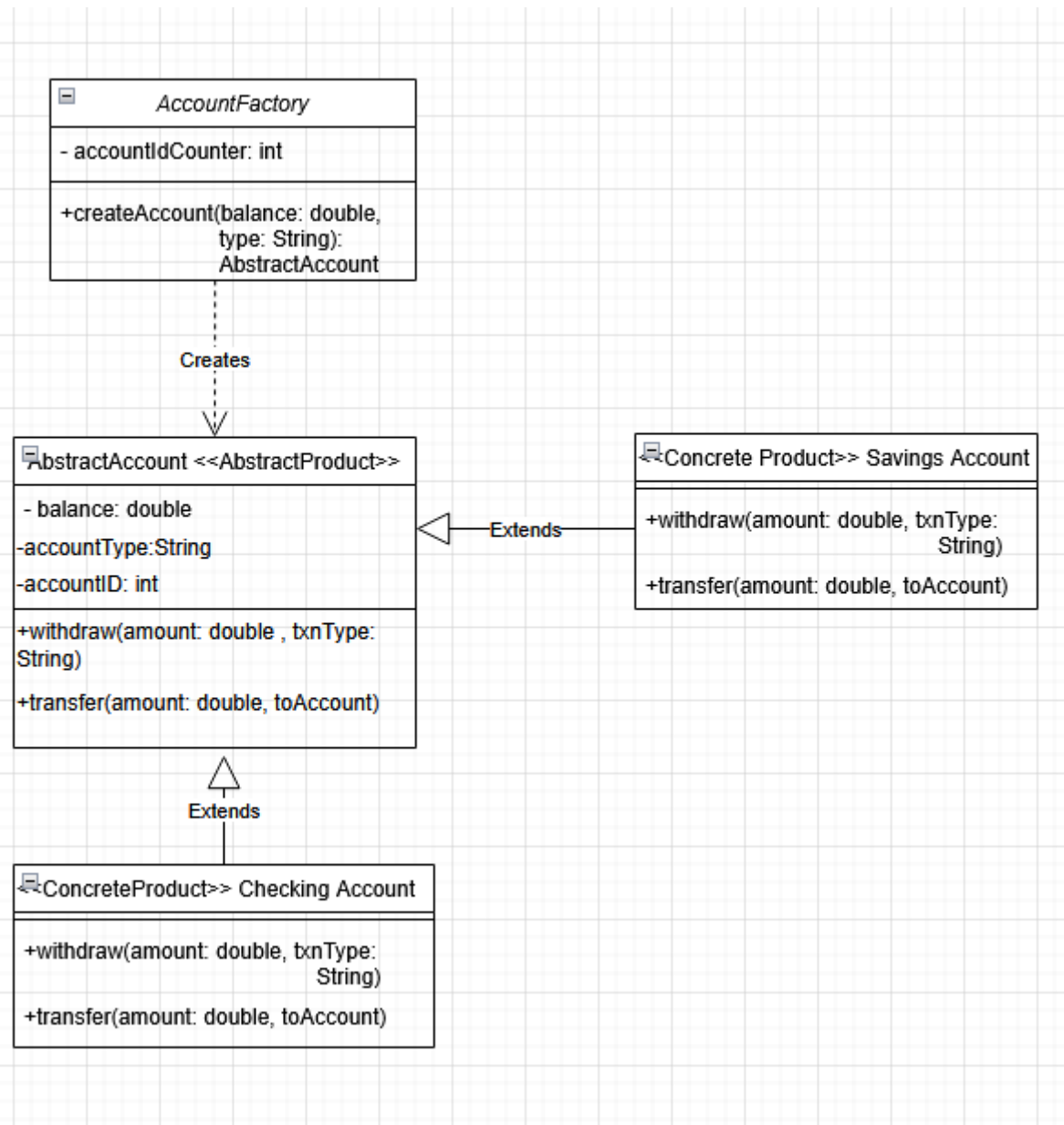
- The intent of the Builder Pattern is to separate the construction of an object from its representation so the same process can create different presentations in this case beign the sign up process. Reason for using the builder pattern is to avoid constructors with many parameters which makes it clear which fields are set.

### 4. Command Design Pattern

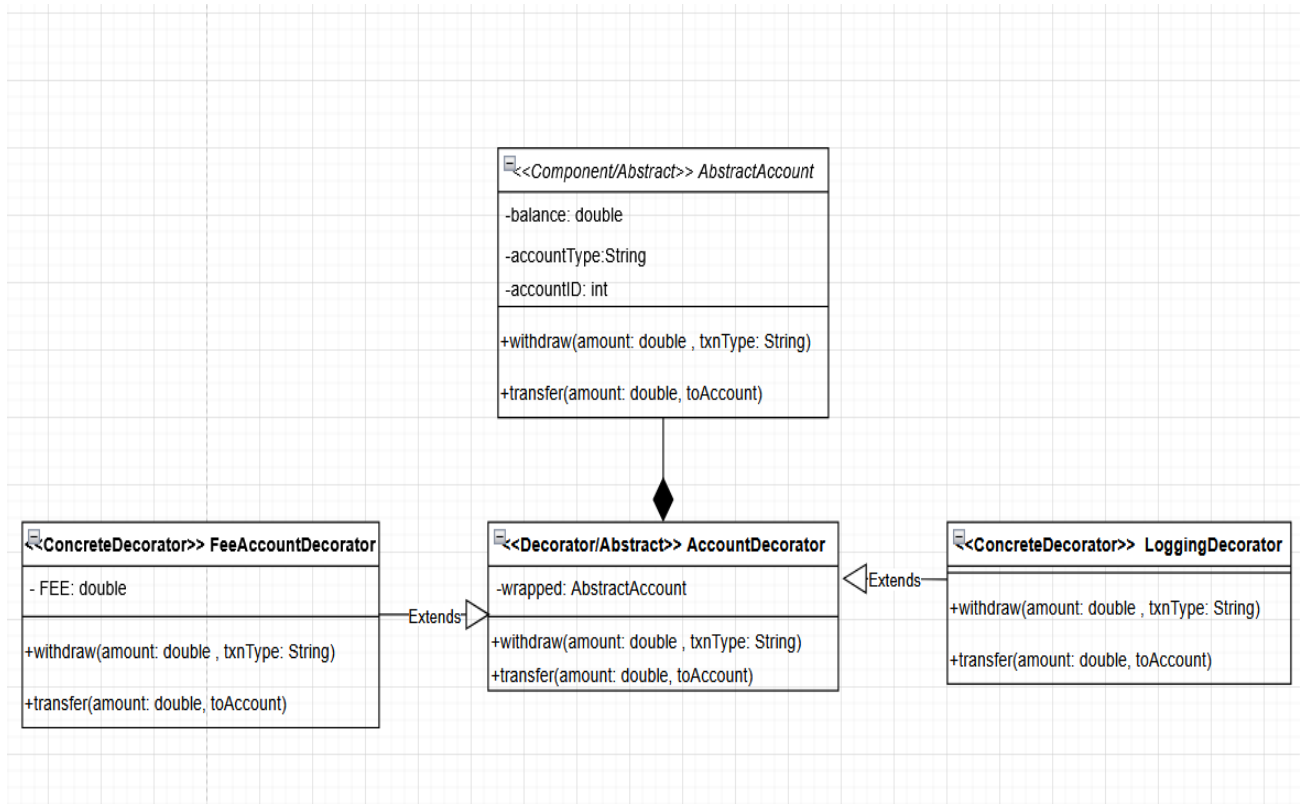
- The intent of the Command Design Pattern is to encapsulate a request as an object therefore letting you to parameterize clients with queues and requests. In the ATM GUI each action like Transfer or Deposit needs to invoke business logic. So I encapsulated core business logic to guarantee that when working with the code again the code will no break.

## 5. Use cases and UML diagrams

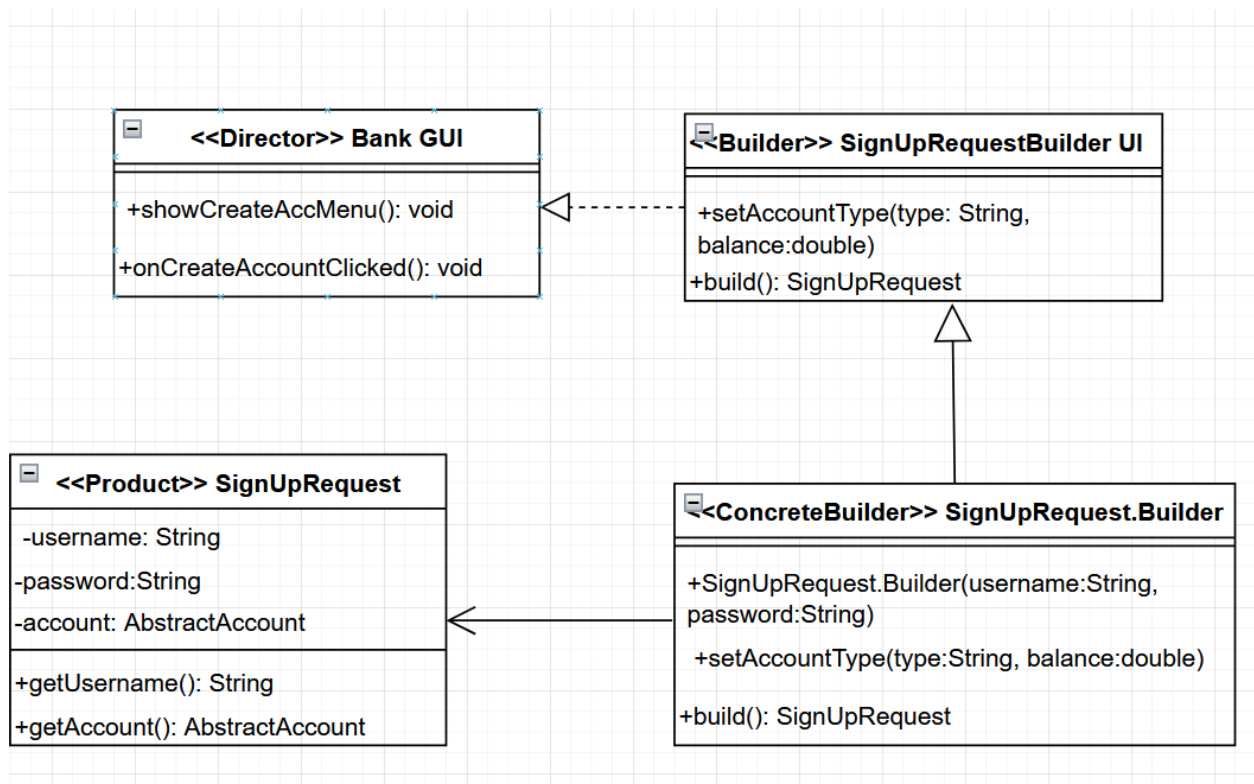
### Simple Factory pattern UML Diagram



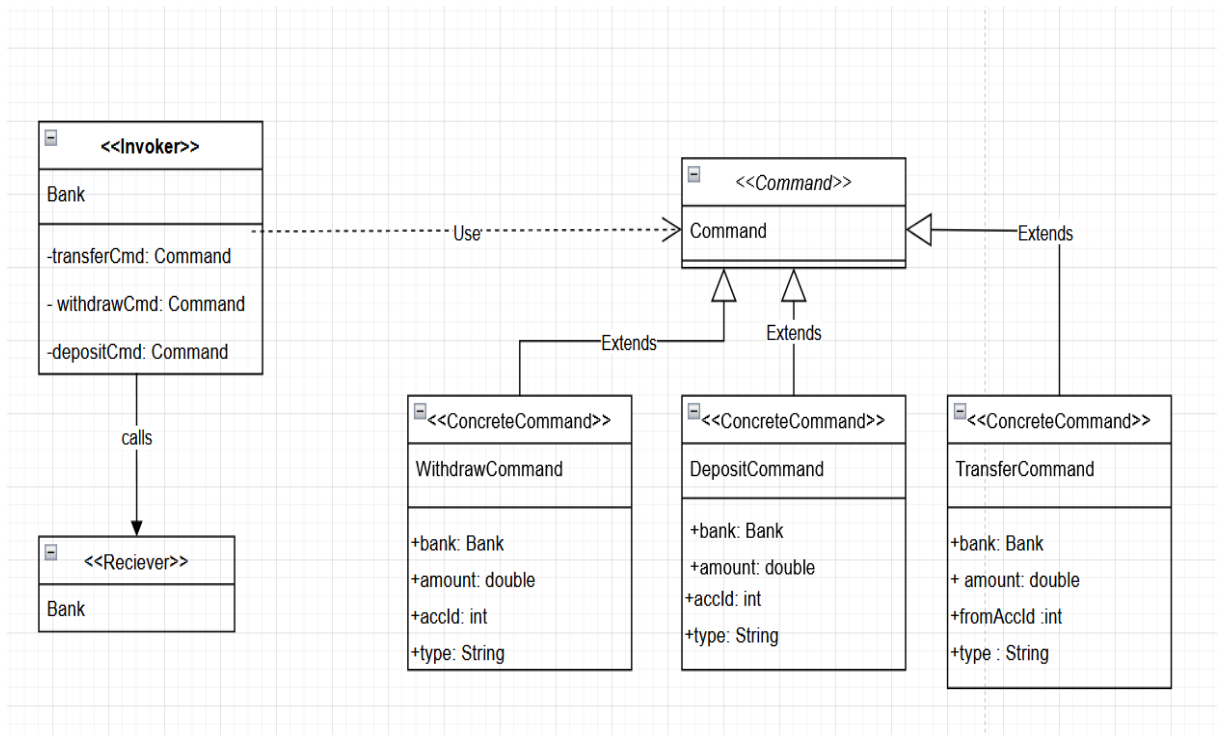
# Decorator Design Pattern



## Builder Design Pattern



# Command Design Pattern





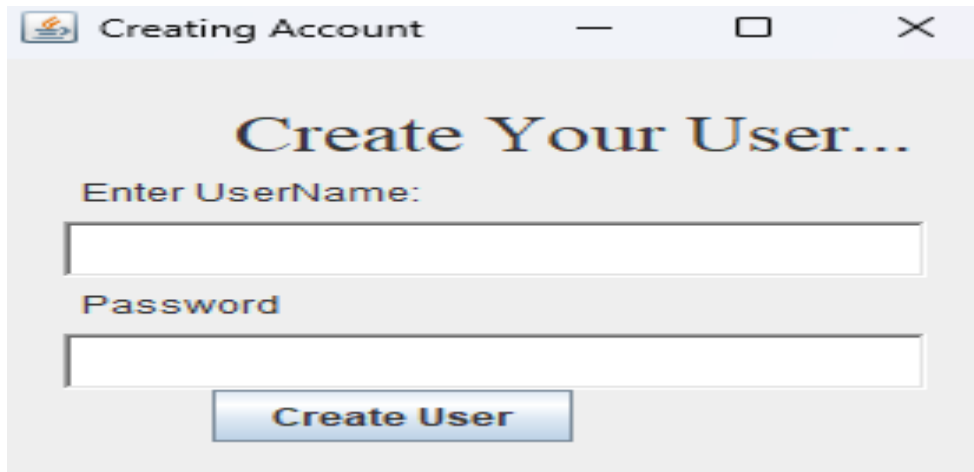
## 6. Application Execution

- Logging/Sign Up Menu



A screenshot of a Java Swing window titled "ATM Machine". The window has a light gray background and a title bar with standard Windows controls. The main content area displays the text "Sign up / Login" in a large, bold, serif font. Below this, there are two labels: "\*\*\* Enter UserName:" and "\*\*\* Password". Each label is followed by a white text input field with a thin gray border. At the bottom right of the form, there are two blue buttons with white text: "Sign Up" and "Sign In", stacked vertically.

- SignUp Menu



A screenshot of a Java Swing window titled "Creating Account". The window has a light gray background and a title bar with standard Windows controls. The main content area displays the text "Create Your User..." in a large, bold, serif font. Below this, there are two labels: "Enter UserName:" and "Password". Each label is followed by a white text input field with a thin gray border. At the bottom center of the form, there is a single blue button with white text: "Create User".

- Creating an Account UI

Creating Account

Create your Account : 1

Select your account type :

☒ Checking

☐ Saving

Amount to be added:

AccountNumber (opt):

Create Account

## Main Menu

