**Title: Banking System Algorithm**

**Objective:**

To create a banking system program that allows users to perform various banking operations such as opening an account, depositing, and withdrawing money, transferring funds between accounts, and more.

1. **Class Definitions**

-Define two classes: ‘**Bank**’ and ‘**Account**’.

-‘**Bank**’ class:

-Attributes:

-‘**Max\_Accounts**’: Maximum number of accounts allowed in the bank.

-‘**accounts**’: List to store account objects.

* Methods:

-‘\_ \_**init\_ \_( )**’: Initialize the bank with an empty list of accounts.

-‘**add\_account\_to\_bank ( )**’: Add a new account to the bank.

-‘**remove\_account\_to\_from\_bank ( )** ‘: Remove an account from the bank.

-‘**Find\_account ( )**’: Find an account by account number.

-‘**Account**’ class:

-Attributes:

-‘**account\_number**’: Unique identifier for the account.

-‘**owner\_first\_name**’: First name of the account owner.

-‘**owner\_last\_name**’: Last name of the account owner.

-‘**owner\_ssn**’: Social Security Number of account owner.

-‘**pin**’: Personal identification Number for the account.

-**balance**’: Current balance of account.

* Methods:

-‘**\_ \_init\_ \_( )** ‘: initialize an account with owner information and generate account number and

PIN.

-‘**deposit ( )** ‘: Deposit funds into the account.

-‘**withdraw ( )**’: Withdraw funds from the account.

-**is\_valid\_pin ( )**’: Check if provided PIN is valid.

-**generate\_account\_number ( )**’:Generate a random 8-digit account number.

-**generate\_pin\_( )** ‘:Generate a random 4-digit pin

**2.** **Utility classes and functions:**

-‘**Coincollector**’ class:

-**Static method**’ parse\_change ( )’: Parse change represented by coins into total cents.

-‘**BankUtility**’ class:

* Static methods:

-‘**is\_numeric ( )**’: Check if a string is numeric.

-‘**prompt\_user\_for\_string ( )** ‘: Prompt user for input string.

-‘**prompt\_user \_for\_ positive\_ number ( )**’: Prompt user for a positive number input.

-‘ **convert\_from\_dollars\_to\_cents ( )**’: Convert a dollar amount to cents.

-‘**generate\_random\_integer ( )**’: Generate a random integer within a given range.

**3.** **Bank Manager Class:**

-Define a ‘**BankManager**’ class to manage bank operations and user interactions.

* Methods:

-‘**main ( )**’: Main program loop for user interactions and operations.

-‘**open\_account ( )** ‘: Open a new bank account.

-‘**get\_account\_info\_and\_balance ( )**’: Display account information and balance.

-‘**change\_pin ( )**’: Change PIN for an account.

-‘**deposit\_money ( )**’: Deposit money into account.

-‘ **transfer\_money ( )** ‘: Transfer money between accounts.

-‘**withdraw\_money ( )** ‘: Withdraw money from an account.

-‘**atm\_withdrawal ( )**’: Make an ATM withdrawl from an account.

-‘**deposit\_change ( )**’: Deposit change into an account.

-‘**close\_account ( )**’: Close an account.

-‘**add\_monthly\_interest ( )**’: Add monthly interest to all accounts.

**4**. **Testing:**

* Define a ‘**TestBank**’ class for unit tests of banking functionalities.
* Methods to test:

-‘**test\_generate\_account\_number ( )** ‘

-‘**test\_generate\_pin ( )’**

**-‘test\_parse\_change ( )**’

**-‘test\_is\_numeric ( )**’

**-‘test\_convert\_from\_dollars\_to\_cents (** )’

**5. Execute Program:**

-Run unit tests using the ‘**run-tests ( )** ‘ function.

-Execute the main program loop using the ‘**execute\_main\_program ( )**’ function.

**6. Program Execution Flow:**

**-**Initialize a ‘**BankManager**’ object.

- Run the main program loop.

-Prompt user for choice and execute corresponding operation.

**7. Exception Handling:**

**-**Handle exceptions for invalid inputs and interrupted operations.