ATM CASE STUDY

OVER VIEW

Developing / creating a software for ATM by implementing oops concepts and functionalities in Java .

Implementation of oops concepts

- Abstraction
- Encapsulation
- Polymorphism
- Interface
- Inheritance

FUNCTIONALITY

Features of the ATM machine

- At first user need to enter 5-digit account number, if the user enters wrong/invalid account number then the system asks the user to re-enter the correct account number.
- Next the user need to enter the pin number, if the pin number is correct the user can use the ATM system otherwise the user has 3 valid attempts.
- Even after 3 attempts if the credentials fail to match the user is given two options either to reset the pin or exit.
- The pin can only be reset if the user enters the correct security code (user must know the security code).
- After resetting the pin the user can re-login with the new pin
- After a successful login the user can view his account balance, deposit money, withdraw money, change pin number and one exit option through which he can logout from the ATM machine.
- The user can do these operations any number of times he/she want, finally user can exit through the exit option
- When the operating user enters the exit option his/her operations are terminated and control goes to login (to other user).
- The designed code only allows access to already existing accounts and cannot update the bank data base via an ATM due to security reasons.

OOPS CONCEPT IMPLEMENTATION AND WORKING OF IMPORTANT METHODS AND CLASSES

- Account details of user should be secured, so I made Account class as abstract class, no object can be created for the same due to security purpose.
- LIST OF CLASSES AND IMPORTANRT METHODS IN THEM:
    1. Interface execute
    2. Abstract class Account(which implements execute class)
        - getSecurity_code()
        - set_account
        - getAccount_number
        - set_pin
        - get_pin
        - check
        - check(polymorphism)
        - amtavailable
    3. Class menu(extends Account)
        - View_balance()
        - Deposit
        - Withdraw
    4. Class CashDispenser
        - DispenseCash
        - Depositslot
        - CashAvailable
    5. Class Bank-users
        - B_user
        - Check_bu
    6. Main class ATM   (contains main function)

WORKING

Interface execute

- Which implements the functions of abstract object using single object creation during run time.
- Reason for using this class is to reduce the number of objects(1).

Abstract class Account

--this class contains methods related to users login credentials and account details(balance).

- getSecurity_code() , it is a getter function to get security code.
- set_account , it is a setter function that sets account number ,pin number ,balance and security code
- getAccount_number() , it is a getter function to get account number
- set_pin() , it is a setter function that sets pin number
- get_pin() , it is a getter function to get pin number
- check(int , int) , it checks whether given input account number us present in bank data base or not
- check(int) , it checks whether given pin number input is correct for that account number or not.
- amtavailable() , this method checks the withdraw amount with balance in the user account and gives the result respectively

menu (extends Account**)**

--this class is defined for carrying out various operations on users bank balance

- View_balance() , it is a method which returns the balance of the user
- Deposit() , this method takes the deposited money(input--parameter) from the user and add it to the balance of the user in the data base
- Withdraw() , this method subtracts the withdraw money (input--paramter) from the balance of the user in the data base.

Cash Dispenser

--this class is specified for changes in amount of money in ATM system.

- DispenseCash() , this method detects the withdraw money of the user from the ATM system.
- Depositslot() , , this method adds the deposited money of the user from the ATM system.

- CashAvailable() , this method checks the withdraw amount with balance in the ATM machine and gives the result respectively

Bank-users

--this class is specified for creating data base for bank and also to check the account number of user in in banks data base using methods

- B_user() , this method contains bank data base which was created using array of execute type and objects of menu class.Using the set_account method in Account class, users details were assigned to objects that are created.
- Check_bu() , this method checks the bank user account number and returns users account index number in the array.

Main class ATM

- It has objects for the classes further it contains a menu driven program where the methods in each class are invoked .

OOPS CONCEPTS:

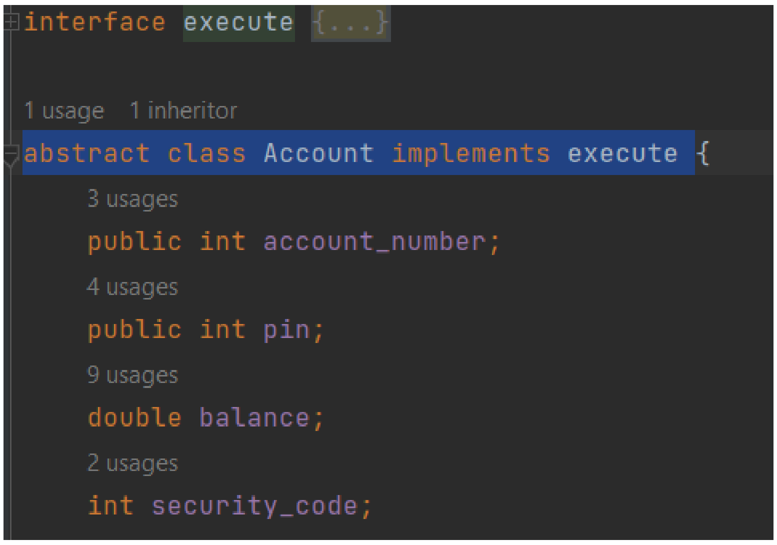
Polymorphism:

- Same method name is used to perform two different functionalities by varying the parameters.
- Example: check(int , int) , it checks whether given input account number us present in bank data base or not.

check(int) , it checks whether given pin number input is correct for that account number or not.

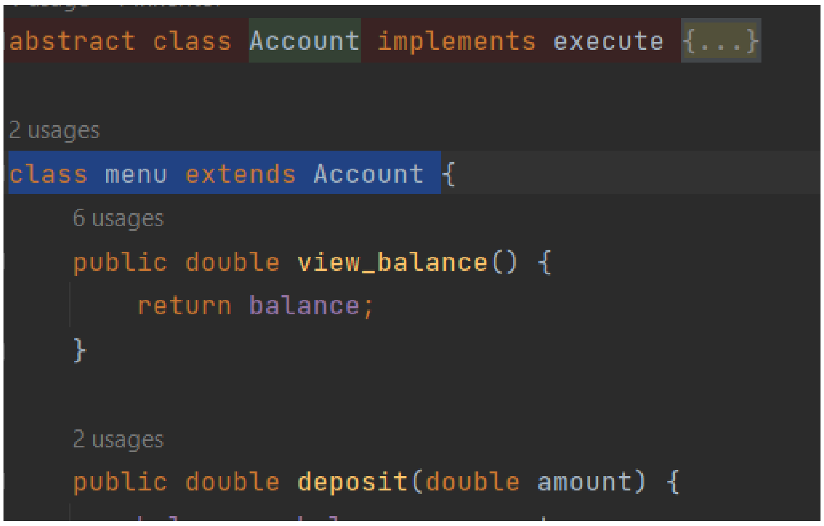
Inheritance:

- The idea behind inheritance is that you can create new classes that are built upon existing classes.
- Here in the present code , Account class implements execute class and the menu class extends Account class.



```
interface execute {...}

1 usage   1 inheritor
abstract class Account implements execute {
    3 usages
    public int account_number;
    4 usages
    public int pin;
    9 usages
    double balance;
    2 usages
    int security_code;
```



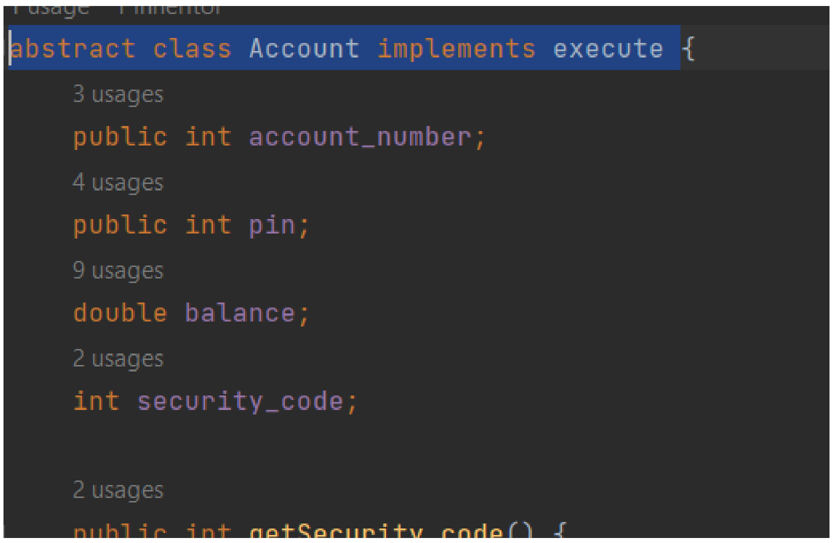
```
abstract class Account implements execute {...}

2 usages
class menu extends Account {
    6 usages
    public double view_balance() {
        return balance;
    }

    2 usages
    public double deposit(double amount) {
```

Abstraction:

- Account class is made abstract
- the account details of the user are present in this Account class ,these details should be secure so by using abstract class , object cannot be created and that class cannot be accessed directly.
- Example:
  Abstract Account class.

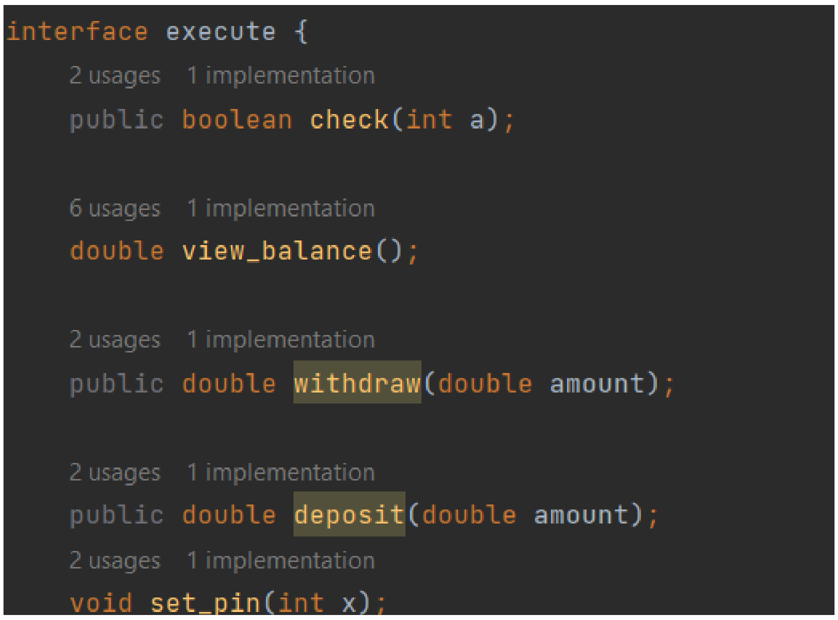


```
1 usage   1 inheritor
abstract class Account implements execute {
    3 usages
    public int account_number;
    4 usages
    public int pin;
    9 usages
    double balance;
    2 usages
    int security_code;

    2 usages
    public int getSecurity_code() {
```

Interface:

- Interface execute class contains mostly all methods of the ATM project
- By using this interface class one object is created to access respective classes related to the user functionalities.
- It mostly helps to reduce the number of objects that need to be created.



```
interface execute {
    2 usages   1 implementation
    public boolean check(int a);

    6 usages   1 implementation
    double view_balance();

    2 usages   1 implementation
    public double withdraw(double amount);

    2 usages   1 implementation
    public double deposit(double amount);
    2 usages   1 implementation
    void set_pin(int x);
```

<u>Encapsulation:</u>

- Objects have been created for each and every classes .
- Each class have different operations and functionalities(methods).
- These functionalities are encapsulated in classes
- Example:
  - Class Account
  - Class menu
  - Class Cash-Dispenser
  - Class bank-user