

# Assignment:-3

## By Shreyasi Reja

### Banking System

### Control Structure

#### Task 1: Conditional Statements

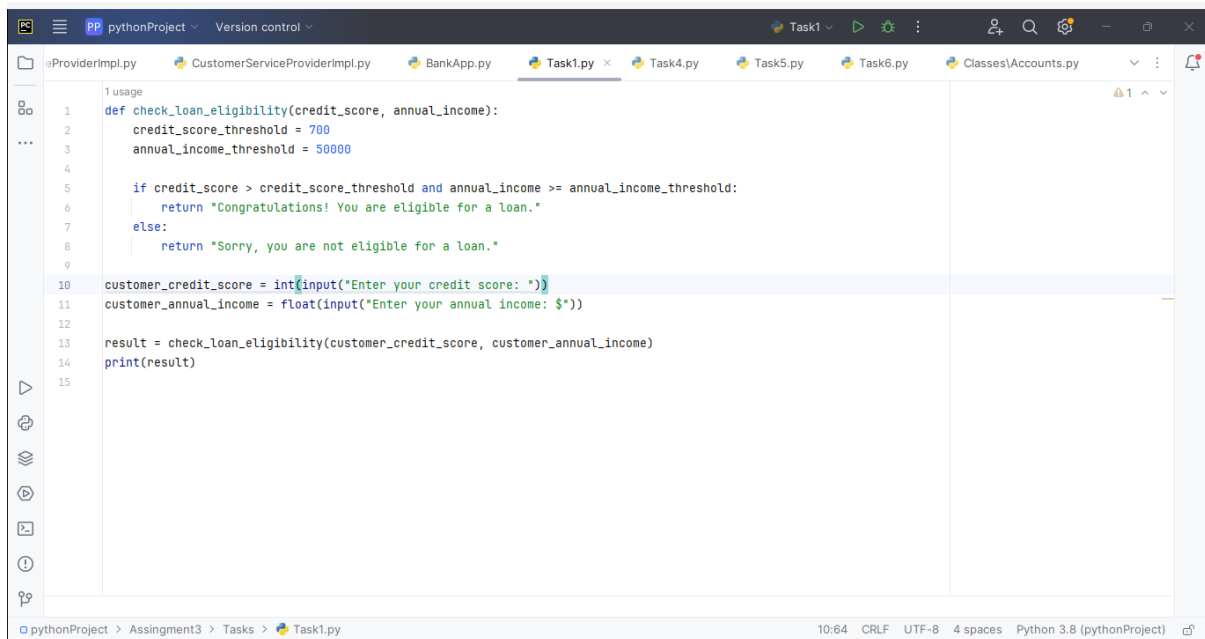
In a bank, you have been given the task is to create a program that checks if a customer is eligible for a loan based on their credit score and income. The eligibility criteria are as follows:

Credit Score must be above 700.

Annual Income must be at least \$50,000.

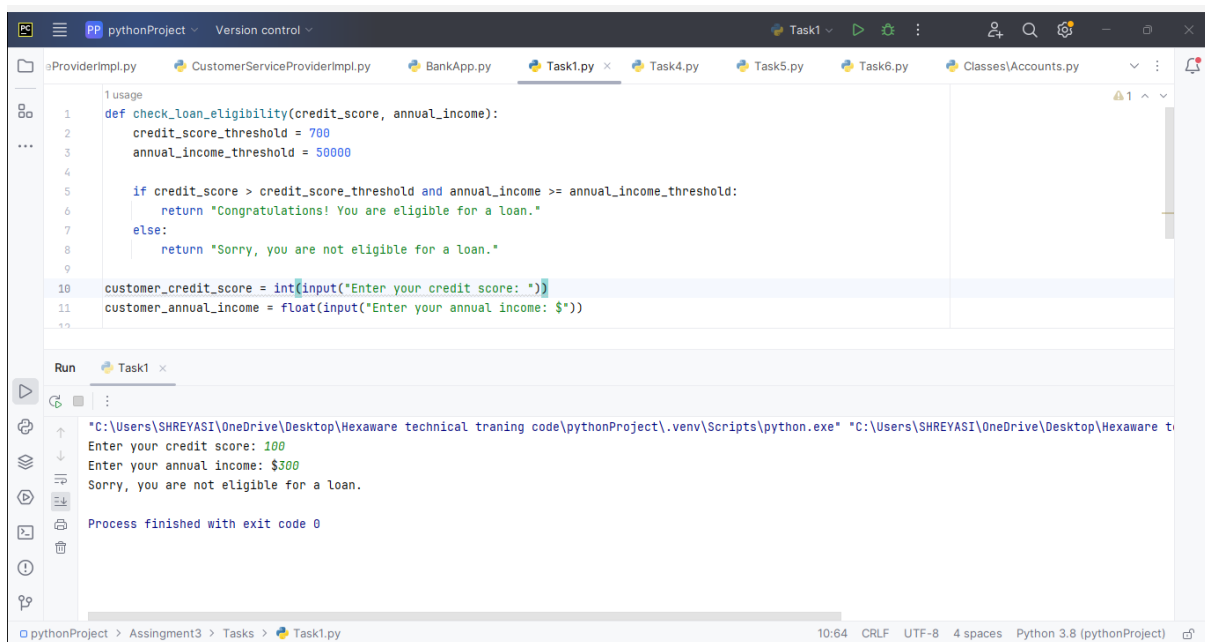
Tasks:

1. Write a program that takes the customer's credit score and annual income as input.
2. Use conditional statements (if-else) to determine if the customer is eligible for a loan.
3. Display an appropriate message based on eligibility.



```
1 usage
2 def check_loan_eligibility(credit_score, annual_income):
3     credit_score_threshold = 700
4     annual_income_threshold = 50000
5
6     if credit_score > credit_score_threshold and annual_income >= annual_income_threshold:
7         return "Congratulations! You are eligible for a loan."
8     else:
9         return "Sorry, you are not eligible for a loan."
10
11 customer_credit_score = int(input("Enter your credit score: "))
12 customer_annual_income = float(input("Enter your annual income: $"))
13
14 result = check_loan_eligibility(customer_credit_score, customer_annual_income)
15 print(result)
```

Output:-

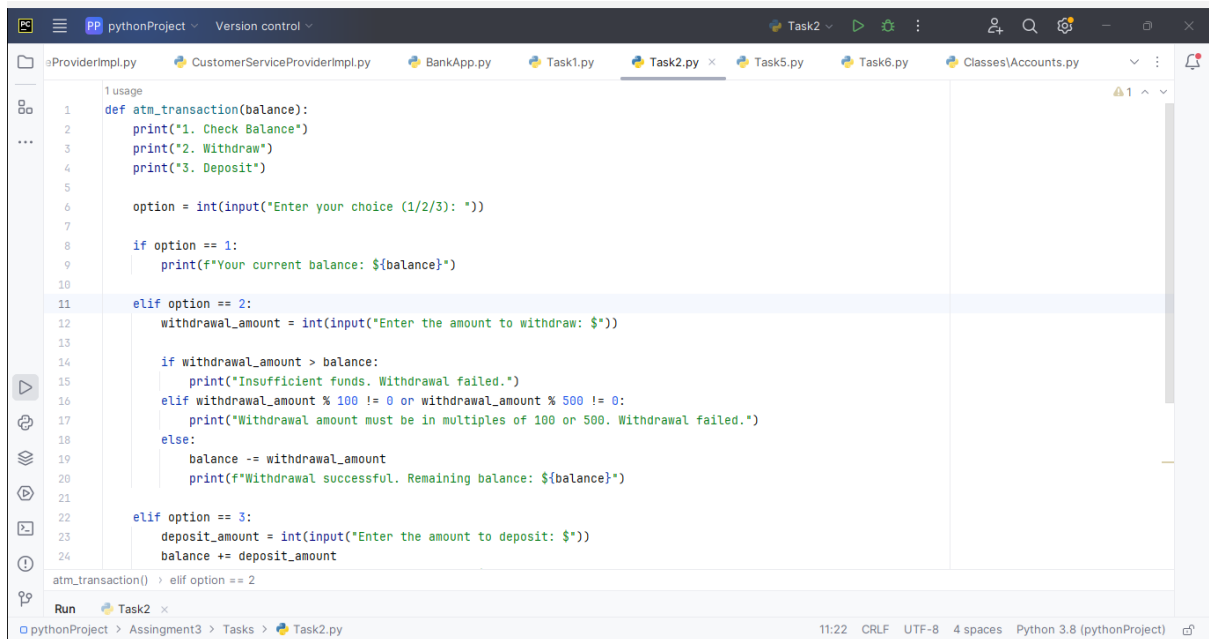


```
*C:\Users\SHREYASI\OneDrive\Desktop\Hexaware technical traning code\pythonProject\.venv\Scripts\python.exe "C:\Users\SHREYASI\OneDrive\Desktop\Hexaware t
Enter your credit score: 100
Enter your annual income: $300
Sorry, you are not eligible for a loan.
Process finished with exit code 0
```

## Task 2: Nested Conditional Statements

Create a program that simulates an ATM transaction. Display options such as "Check Balance," "Withdraw," "Deposit,". Ask the user to enter their current balance and the amount they want to withdraw or deposit. Implement checks to ensure that the withdrawal amount is not greater than the available balance and that the withdrawal

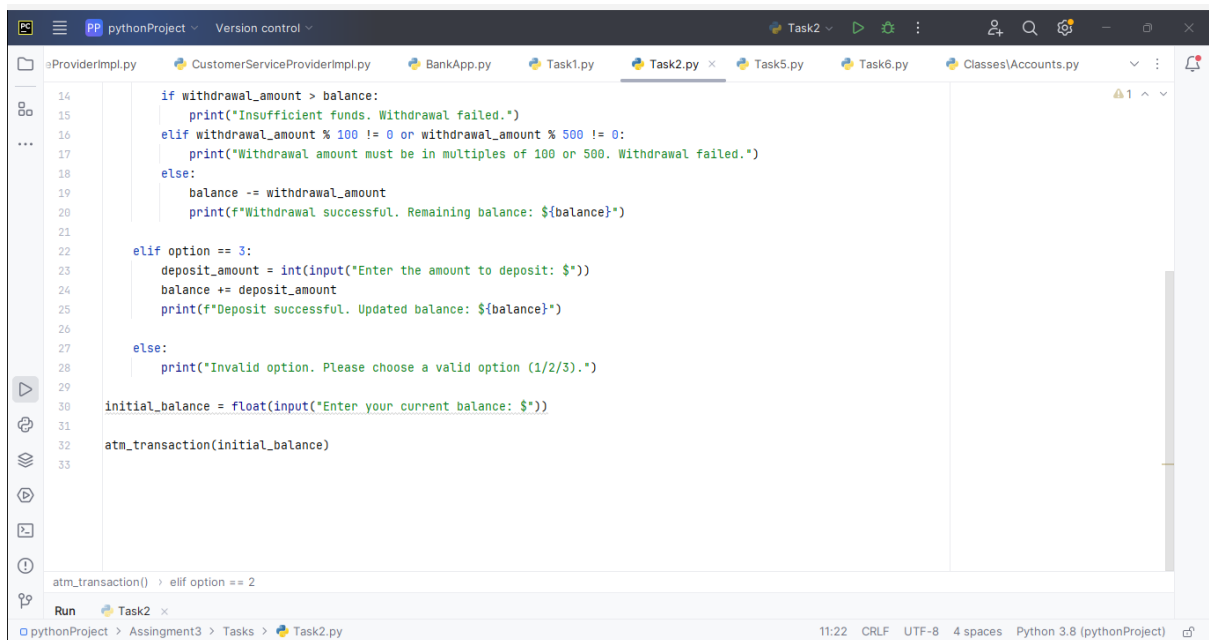
amount is in multiples of 100 or 500. Display appropriate messages for success or failure.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'pythonProject' with several files: 'ProviderImpl.py', 'CustomerServiceProviderImpl.py', 'BankApp.py', 'Task1.py', 'Task2.py', 'Task5.py', 'Task6.py', and 'Classes\Accounts.py'. The code editor shows the 'Task2.py' file with the following code:

```
1 usage
2 def atm_transaction(balance):
3     print("1. Check Balance")
4     print("2. Withdraw")
5     print("3. Deposit")
6
7     option = int(input("Enter your choice (1/2/3): "))
8
9     if option == 1:
10        print(f"Your current balance: ${balance}")
11
12    elif option == 2:
13        withdrawal_amount = int(input("Enter the amount to withdraw: $"))
14
15        if withdrawal_amount > balance:
16            print("Insufficient funds. Withdrawal failed.")
17        elif withdrawal_amount % 100 != 0 or withdrawal_amount % 500 != 0:
18            print("Withdrawal amount must be in multiples of 100 or 500. Withdrawal failed.")
19        else:
20            balance -= withdrawal_amount
21            print(f"Withdrawal successful. Remaining balance: ${balance}")
22
23    elif option == 3:
24        deposit_amount = int(input("Enter the amount to deposit: $"))
25        balance += deposit_amount
26
27    atm_transaction() > elif option == 2
```

The status bar at the bottom shows 'pythonProject > Assingment3 > Tasks > Task2.py' and '11:22 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)'.



The screenshot shows the same code editor with the 'Task2.py' file. The code is as follows:

```
14 if withdrawal_amount > balance:
15     print("Insufficient funds. Withdrawal failed.")
16 elif withdrawal_amount % 100 != 0 or withdrawal_amount % 500 != 0:
17     print("Withdrawal amount must be in multiples of 100 or 500. Withdrawal failed.")
18 else:
19     balance -= withdrawal_amount
20     print(f"Withdrawal successful. Remaining balance: ${balance}")
21
22 elif option == 3:
23     deposit_amount = int(input("Enter the amount to deposit: $"))
24     balance += deposit_amount
25     print(f"Deposit successful. Updated balance: ${balance}")
26
27 else:
28     print("Invalid option. Please choose a valid option (1/2/3).")
29
30 initial_balance = float(input("Enter your current balance: $"))
31
32 atm_transaction(initial_balance)
33
```

The status bar at the bottom shows 'pythonProject > Assingment3 > Tasks > Task2.py' and '11:22 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)'.

Output

The screenshot shows a Python IDE with a file explorer on the left containing files like `ProviderImpl.py`, `CustomerServiceProviderImpl.py`, `BankApp.py`, `Task1.py`, `Task2.py`, `Task5.py`, `Task6.py`, and `Classes/Accounts.py`. The main editor displays the following Python code in `Task2.py`:

```
12 withdrawal_amount = int(input("Enter the amount to withdraw: "))
13
14 if withdrawal_amount > balance:
15     print("Insufficient funds. Withdrawal failed.")
16 elif withdrawal_amount % 100 != 0 or withdrawal_amount % 500 != 0:
17     print("Withdrawal amount must be in multiples of 100 or 500. Withdrawal failed.")
18 else:
19     balance -= withdrawal_amount
20     print(f"Withdrawal successful. Remaining balance: ${balance}")
21 atm_transaction() > elif option == 2
```

Below the code, the 'Run' output for 'Task2' is shown:

```
*C:\Users\SHREYASI\OneDrive\Desktop\Hexaware technical training code\pythonProject\.venv\Scripts\python.exe* *C:\Users\SHREYASI\OneDrive\Desktop\Hexaware t
Enter your current balance: $100
1. Check Balance
2. Withdraw
3. Deposit
Enter your choice (1/2/3): 1
Your current balance: $100.0
Process finished with exit code 0
```

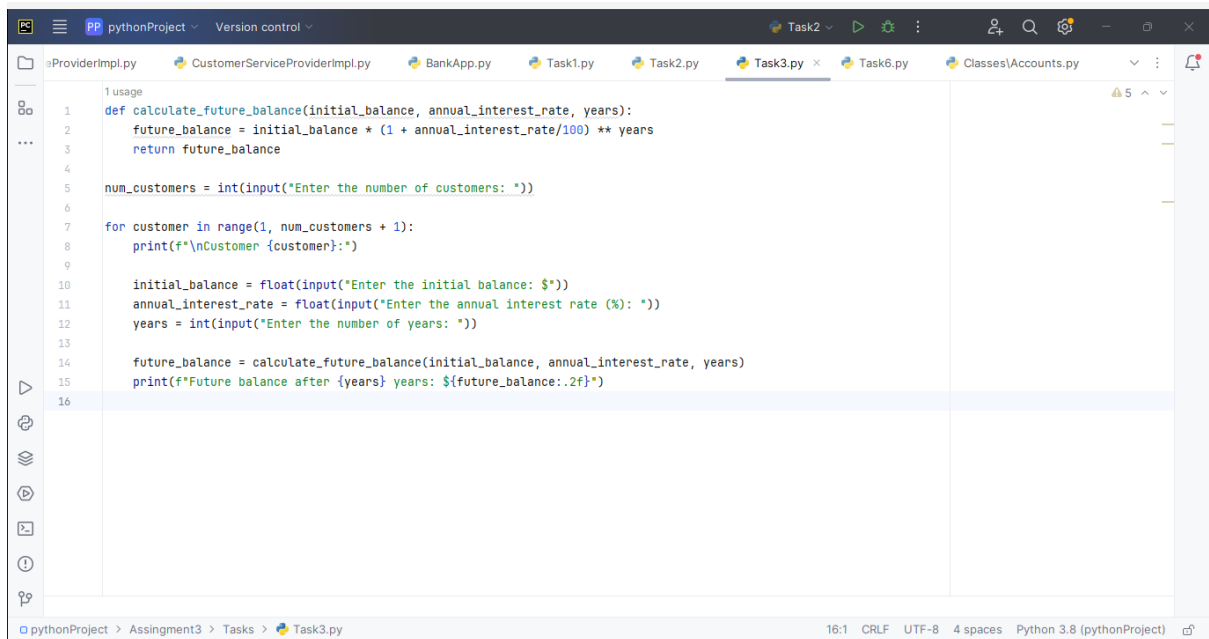
The status bar at the bottom indicates the file is `Task2.py` in the `pythonProject` directory, with a timestamp of 11:22 and encoding of CRLF, UTF-8, 4 spaces, Python 3.8.

### Task 3: Loop Structures

You are responsible for calculating compound interest on savings accounts for bank customers. You need to calculate the future balance for each customer's savings account after a certain number of years.

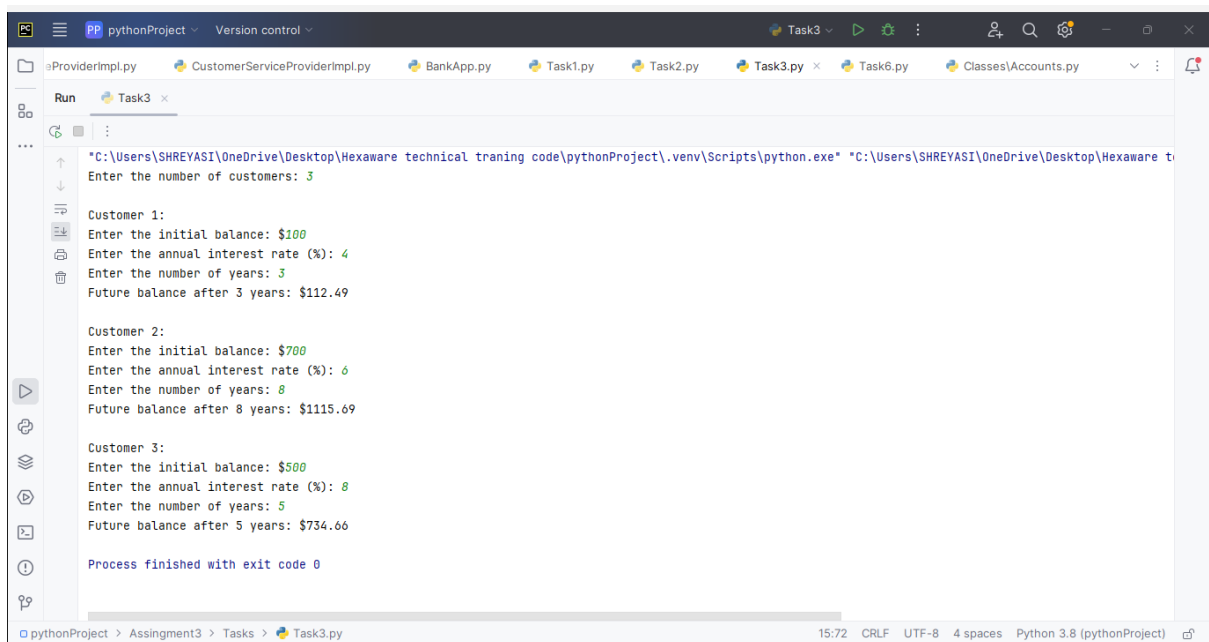
Tasks:

1. Create a program that calculates the future balance of a savings account.
2. Use a loop structure (e.g., for loop) to calculate the balance for multiple customers.
3. Prompt the user to enter the initial balance, annual interest rate, and the number of years.
4. Calculate the future balance using the formula:  $\text{future\_balance} = \text{initial\_balance} * (1 + \text{annual\_interest\_rate}/100)^{\text{years}}$ .
5. Display the future balance for each customer.



```
1 usage
2 def calculate_future_balance(initial_balance, annual_interest_rate, years):
3     future_balance = initial_balance * (1 + annual_interest_rate/100) ** years
4     return future_balance
5
6 num_customers = int(input("Enter the number of customers: "))
7
8 for customer in range(1, num_customers + 1):
9     print(f"\nCustomer {customer}:")
10
11     initial_balance = float(input("Enter the initial balance: $"))
12     annual_interest_rate = float(input("Enter the annual interest rate (%): "))
13     years = int(input("Enter the number of years: "))
14
15     future_balance = calculate_future_balance(initial_balance, annual_interest_rate, years)
16     print(f"Future balance after {years} years: ${future_balance:.2f}")
```

## Output



```
*C:\Users\SHREYASI\OneDrive\Desktop\Hexaware technical tranning code\pythonProject\.venv\Scripts\python.exe" *C:\Users\SHREYASI\OneDrive\Desktop\Hexaware t
Enter the number of customers: 3

Customer 1:
Enter the initial balance: $100
Enter the annual interest rate (%): 4
Enter the number of years: 3
Future balance after 3 years: $112.49

Customer 2:
Enter the initial balance: $700
Enter the annual interest rate (%): 6
Enter the number of years: 8
Future balance after 8 years: $1115.69

Customer 3:
Enter the initial balance: $500
Enter the annual interest rate (%): 8
Enter the number of years: 5
Future balance after 5 years: $734.66

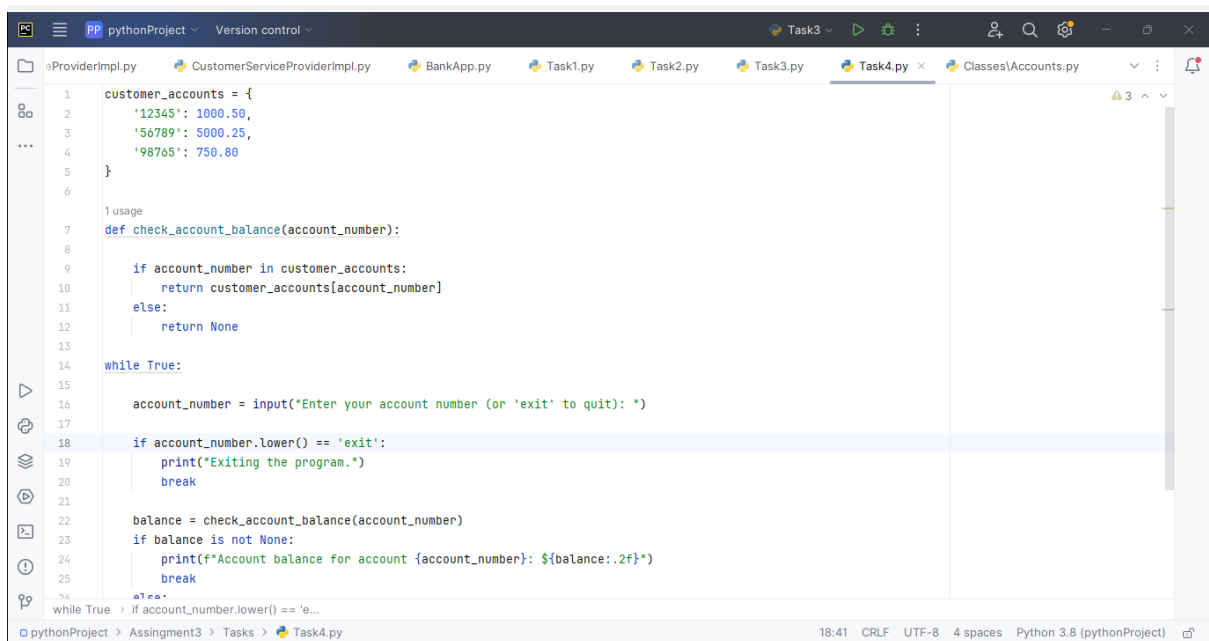
Process finished with exit code 0
```

## Task 4: Looping, Array and Data Validation

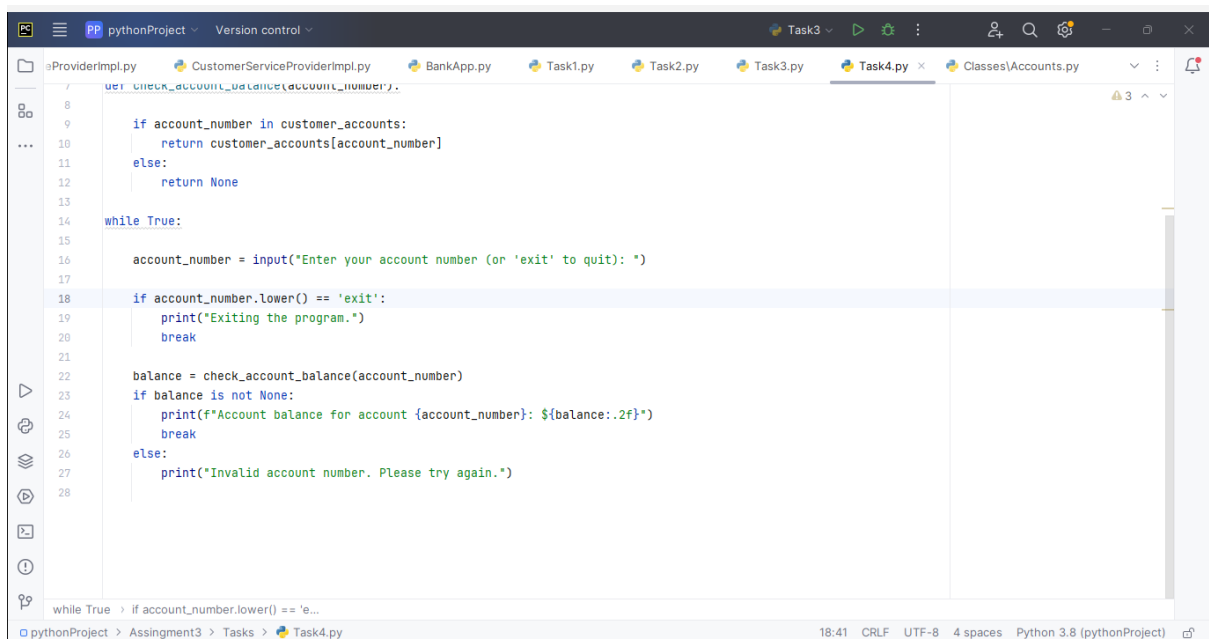
You are tasked with creating a program that allows bank customers to check their account balances. The program should handle multiple customer accounts, and the customer should be able to enter their account number, balance to check the balance.

Tasks:

1. Create a Python program that simulates a bank with multiple customer accounts.
2. Use a loop (e.g., while loop) to repeatedly ask the user for their account number and balance until they enter a valid account number.
3. Validate the account number entered by the user.
4. If the account number is valid, display the account balance. If not, ask the user to try again.

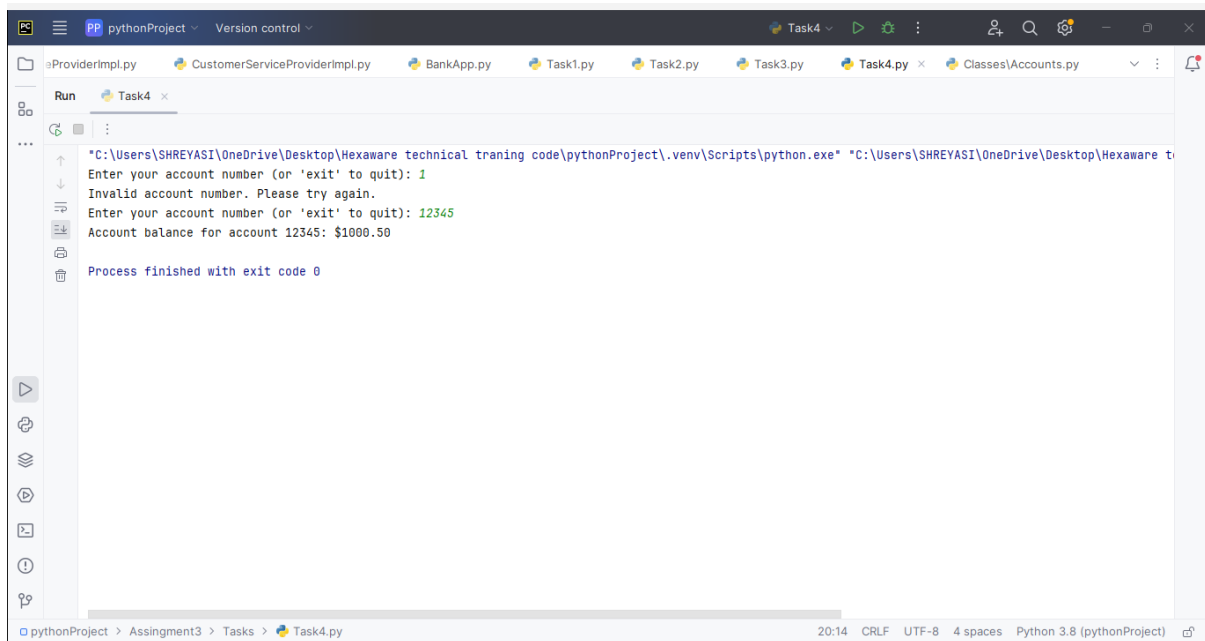


```
1 customer_accounts = {
2     '12345': 1000.50,
3     '56789': 5000.25,
4     '98765': 750.80
5 }
6
7 1 usage
8 def check_account_balance(account_number):
9     if account_number in customer_accounts:
10         return customer_accounts[account_number]
11     else:
12         return None
13
14 while True:
15
16     account_number = input("Enter your account number (or 'exit' to quit): ")
17
18     if account_number.lower() == 'exit':
19         print("Exiting the program.")
20         break
21
22     balance = check_account_balance(account_number)
23     if balance is not None:
24         print(f"Account balance for account {account_number}: ${balance:.2f}")
25         break
26     else:
27         print("Invalid account number. Please try again.")
```



```
7 def check_account_balance(account_number):
8
9     if account_number in customer_accounts:
10         return customer_accounts[account_number]
11     else:
12         return None
13
14 while True:
15
16     account_number = input("Enter your account number (or 'exit' to quit): ")
17
18     if account_number.lower() == 'exit':
19         print("Exiting the program.")
20         break
21
22     balance = check_account_balance(account_number)
23     if balance is not None:
24         print(f"Account balance for account {account_number}: ${balance:.2f}")
25         break
26     else:
27         print("Invalid account number. Please try again.")
28
```

## Output



```
*C:\Users\SHREYASI\OneDrive\Desktop\Hexaware technical training code\pythonProject\.venv\Scripts\python.exe *C:\Users\SHREYASI\OneDrive\Desktop\Hexaware t
Enter your account number (or 'exit' to quit): 1
Invalid account number. Please try again.
Enter your account number (or 'exit' to quit): 12345
Account balance for account 12345: $1000.50

Process finished with exit code 0
```

### Task 5: Password Validation

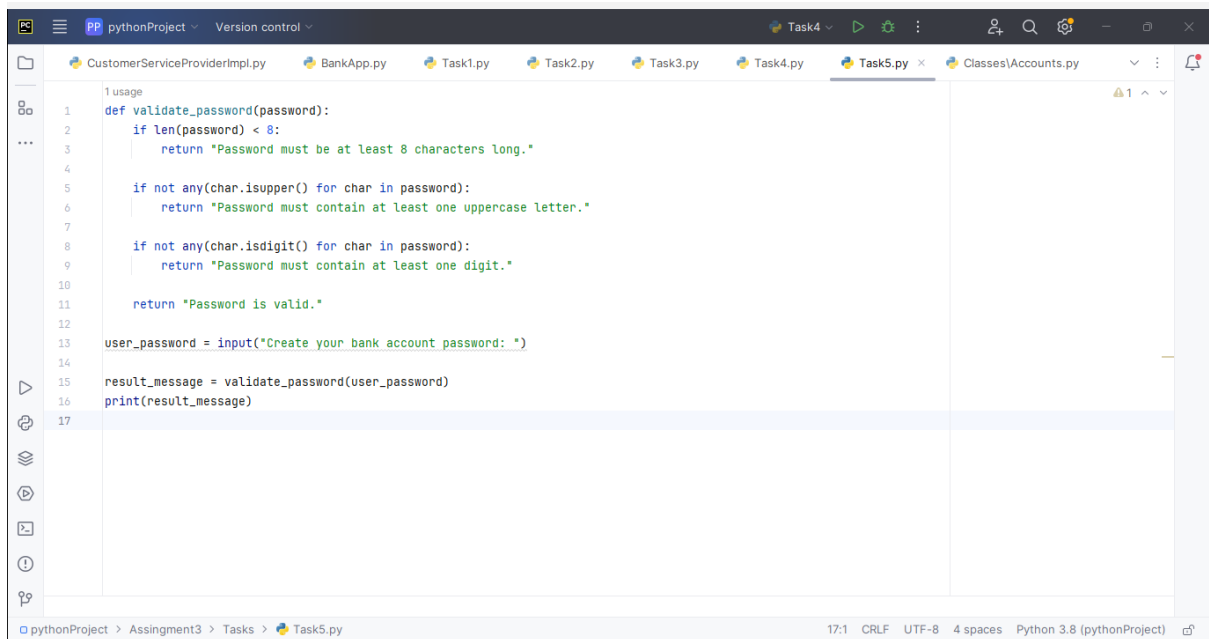
Write a program that prompts the user to create a password for their bank account. Implement if conditions to validate the password according to these rules:

The password must be at least 8 characters long.

It must contain at least one uppercase letter.

It must contain at least one digit.

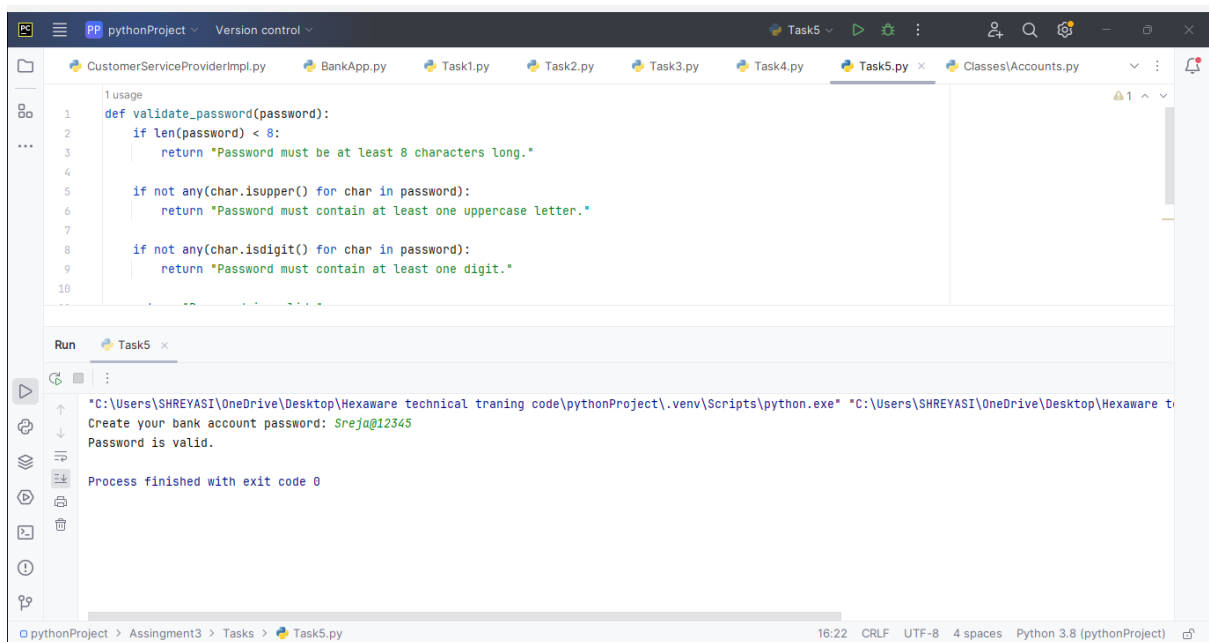
Display appropriate messages to indicate whether their password is valid or not.



```
1 usage
2 def validate_password(password):
3     if len(password) < 8:
4         return "Password must be at least 8 characters long."
5
6     if not any(char.isupper() for char in password):
7         return "Password must contain at least one uppercase letter."
8
9     if not any(char.isdigit() for char in password):
10        return "Password must contain at least one digit."
11
12    return "Password is valid."
13
14 user_password = input("Create your bank account password: ")
15
16 result_message = validate_password(user_password)
17 print(result_message)
```

pythonProject > Assingment3 > Tasks > Task5.py 17:1 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)

## Output



```
1 usage
2 def validate_password(password):
3     if len(password) < 8:
4         return "Password must be at least 8 characters long."
5
6     if not any(char.isupper() for char in password):
7         return "Password must contain at least one uppercase letter."
8
9     if not any(char.isdigit() for char in password):
10        return "Password must contain at least one digit."
11
12    return "Password is valid."
13
14 user_password = input("Create your bank account password: ")
15
16 result_message = validate_password(user_password)
17 print(result_message)
```

Run Task5

```
"C:\Users\SHREYASI\OneDrive\Desktop\Hexaware technical traning code\pythonProject\.venv\Scripts\python.exe" "C:\Users\SHREYASI\OneDrive\Desktop\Hexaware t
Create your bank account password: Sreja@12345
Password is valid.

Process finished with exit code 0
```

pythonProject > Assingment3 > Tasks > Task5.py 16:22 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)

## Task 6: Password Validation

Create a program that maintains a list of bank transactions (deposits and withdrawals) for a customer. Use a while loop to allow the user to keep adding transactions until they choose to exit. Display the transaction history upon exit using looping statements.



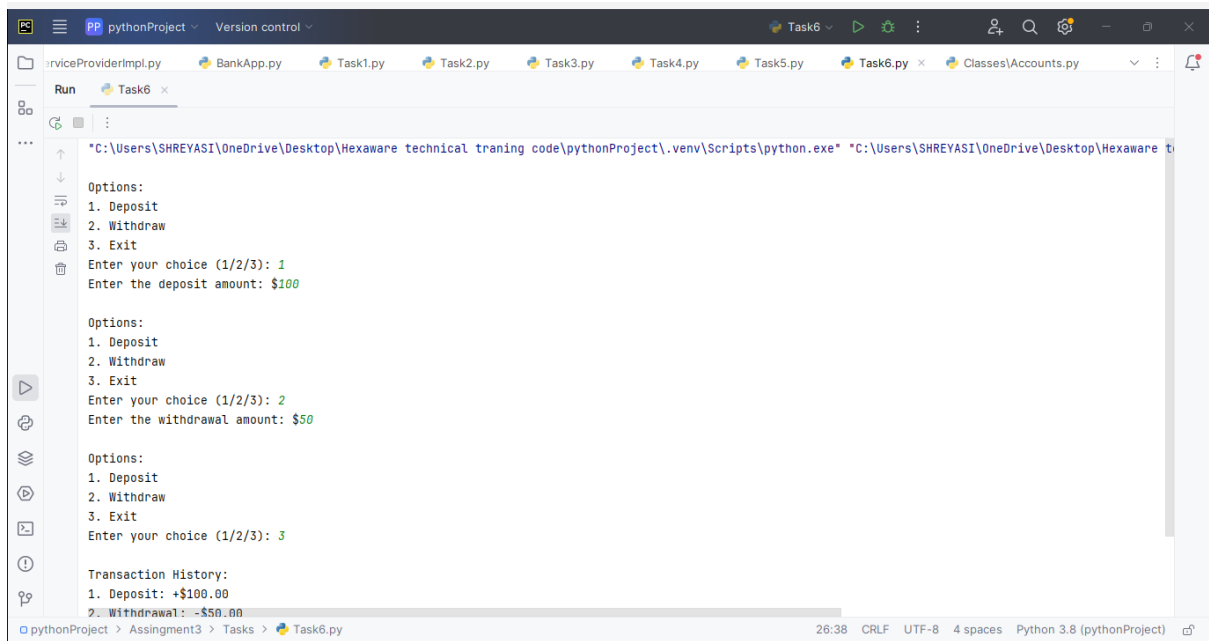
```
pythonProject Version control Task5 Task6.py Classes\Accounts.py
1 transaction_history = []
2
3 1 usage
4 def display_transaction_history():
5     print("\nTransaction History:")
6     for i, transaction in enumerate(transaction_history, start=1):
7         print(f"{i}. {transaction}")
8
9 while True:
10     print("\nOptions:")
11     print("1. Deposit")
12     print("2. Withdraw")
13     print("3. Exit")
14
15     choice = input("Enter your choice (1/2/3): ")
16
17     if choice == '1':
18         deposit_amount = float(input("Enter the deposit amount: $"))
19         transaction_history.append(f"Deposit: +${deposit_amount:.2f}")
20
21     elif choice == '2':
22         withdrawal_amount = float(input("Enter the withdrawal amount: $"))
23         transaction_history.append(f"Withdrawal: -${withdrawal_amount:.2f}")
24
25     elif choice == '3':
26         display_transaction_history()
27         print("Exiting the program.")
28         break
29
30 while True:
31     else
```

pythonProject > Assingment3 > Tasks > Task6.py 29:10 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)

```
pythonProject Version control Task5 Task6.py Classes\Accounts.py
12 print("\n Exit")
13
14 choice = input("Enter your choice (1/2/3): ")
15
16 if choice == '1':
17     deposit_amount = float(input("Enter the deposit amount: $"))
18     transaction_history.append(f"Deposit: +${deposit_amount:.2f}")
19
20 elif choice == '2':
21     withdrawal_amount = float(input("Enter the withdrawal amount: $"))
22     transaction_history.append(f"Withdrawal: -${withdrawal_amount:.2f}")
23
24 elif choice == '3':
25     display_transaction_history()
26     print("Exiting the program.")
27     break
28
29 else:
30     print("Invalid choice. Please enter a valid option.")
31
32
33
34 while True:
35     else
```

pythonProject > Assingment3 > Tasks > Task6.py 29:10 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)

Output



```
*C:\Users\SHREYASI\OneDrive\Desktop\Hexaware technical training code\pythonProject\.venv\Scripts\python.exe" *C:\Users\SHREYASI\OneDrive\Desktop\Hexaware t

Options:
1. Deposit
2. Withdraw
3. Exit
Enter your choice (1/2/3): 1
Enter the deposit amount: $100

Options:
1. Deposit
2. Withdraw
3. Exit
Enter your choice (1/2/3): 2
Enter the withdrawal amount: $50

Options:
1. Deposit
2. Withdraw
3. Exit
Enter your choice (1/2/3): 3

Transaction History:
1. Deposit: +$100.00
2. Withdrawal: -$50.00
```

## OOPS, Collections and Exception Handling

### Task 7: Class & Object

1. Create a `Customer` class with the following confidential attributes:

#### Attributes

- o Customer ID
- o First Name
- o Last Name
- o Email Address
- o Phone Number
- o Address

#### Constructor and Methods

o Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.

CUSTOMER:-

The screenshot shows a code editor with a project structure on the left and a Python file open in the center. The project structure includes a 'pythonProject' folder with subfolders like '.venv', 'Assingment3', 'Classes', 'Connection', 'Service', 'SubClassesOfAccount', 'Tasks', and 'Assingment5'. The 'Classes' folder is expanded, showing files like 'Accounts.py', 'Customers.py', and 'Transactions.py'. The 'Customers.py' file is open, showing a class definition for 'Customers' with an '.\_\_init\_\_' method and several properties. The code is as follows:

```
1 class Customers:
2     def __init__(self, CustomerID: int, FirstName: str, LastName: str, Email: str, Phone: str, Address: str):
3         self.customerID = CustomerID
4         self.firstName = FirstName
5         self.lastName = LastName
6         self.email = Email
7         self.phone = Phone
8         self.address = Address
9
10     3 usages
11     @property
12     def customerID(self):
13         return self.customerID
14
15     3 usages
16     @property
17     def firstName(self):
18         return self.firstName
19
20     3 usages
21     @property
22     def lastName(self):
23         return self.lastName
24
25     3 usages
26     @property
27     def email(self):
```

The screenshot shows a code editor with a project structure on the left and a Python file open in the center. The project structure includes a 'pythonProject' folder with subfolders like 'BankApp.py', 'Task1.py', 'Task2.py', 'Task3.py', 'Task4.py', 'Task5.py', 'Task6.py', 'Customers.py', and 'Classes\Accounts.py'. The 'Customers.py' file is open, showing a class definition for 'Customers' with an '.\_\_init\_\_' method and several properties. The code is as follows:

```
1 class Customers:
2     def __init__(self, CustomerID: int, FirstName: str, LastName: str, Email: str, Phone: str, Address: str):
3         self.customerID = CustomerID
4         self.firstName = FirstName
5         self.lastName = LastName
6         self.email = Email
7         self.phone = Phone
8         self.address = Address
9
10     3 usages
11     @property
12     def customerID(self):
13         return self.customerID
14
15     3 usages
16     @property
17     def firstName(self):
18         return self.firstName
19
20     3 usages
21     @property
22     def lastName(self):
23         return self.lastName
24
25     3 usages
26     @property
27     def email(self):
```

```
22     @property
23     def email(self):
24         return self.email
25
26     3 usages
27     @property
28     def phone(self):
29         return self.phone
30
31     3 usages
32     @property
33     def address(self):
34         return self.address
35
36     2 usages
37     @customerID.setter
38     def customerID(self, customerID):
39         self.customerID = customerID
40
41     2 usages
42     @firstName.setter
43     def firstName(self, firstName):
44         self.firstName = firstName
45
46     2 usages
47     @lastName.setter
```

```
42     @lastName.setter
43     def lastName(self, lastName):
44         self.lastName = lastName
45
46     2 usages
47     @email.setter
48     def email(self, email):
49         self.email = email
50
51     2 usages
52     @phone.setter
53     def phone(self, phone):
54         self.phone = phone
55
56     2 usages
57     @address.setter
58     def address(self, address):
59         self.address = address
60
61     def getCustomerDetails(self):
62         print("Customer Details: ")
63         print(f"ID: {self.customerID}")
64         print(f"Name: {self.firstName} {self.lastName}")
65         print(f"Email: {self.email}")
66         print(f"Phone: {self.phone}")
67         print(f"Address: {self.address}")
```

2. Create an `Account` class with the following confidential attributes:



Attributes

- o Account Number
- o Account Type (e.g., Savings, Current)
- o Account Balance

Constructor and Methods

- o Implement default constructors and overload the constructor with Account attributes,

- o Generate getter and setter, (print all information of attribute) methods for the attributes.

- o Add methods to the `Account` class to allow deposits and withdrawals.

- deposit(amount: float): Deposit the specified amount into the account.

- withdraw(amount: float): Withdraw the specified amount from the account. withdraw amount only if there is sufficient fund else display insufficient balance.

- calculate\_interest(): method for calculating interest amount for the available balance. interest rate is fixed to 4.5%

Create a Bank class to represent the banking system. Perform the following operation in main method:

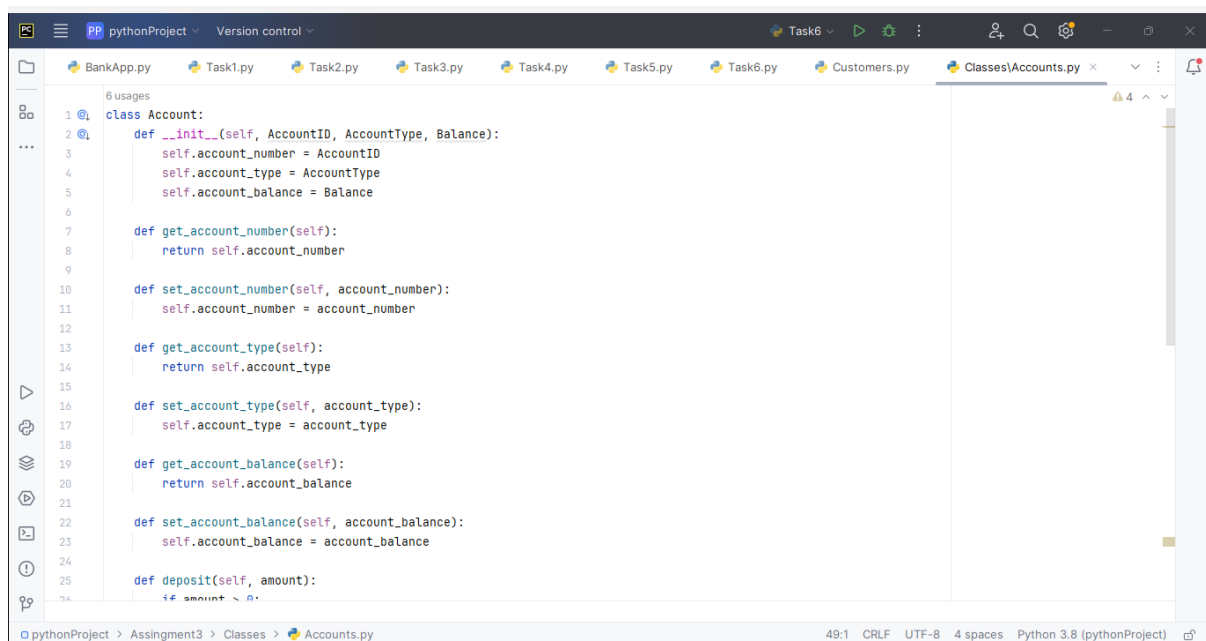
- o create object for account class by calling parameter constructor.

- o deposit(amount: float): Deposit the specified amount into the account.

- o withdraw(amount: float): Withdraw the specified amount from the account.

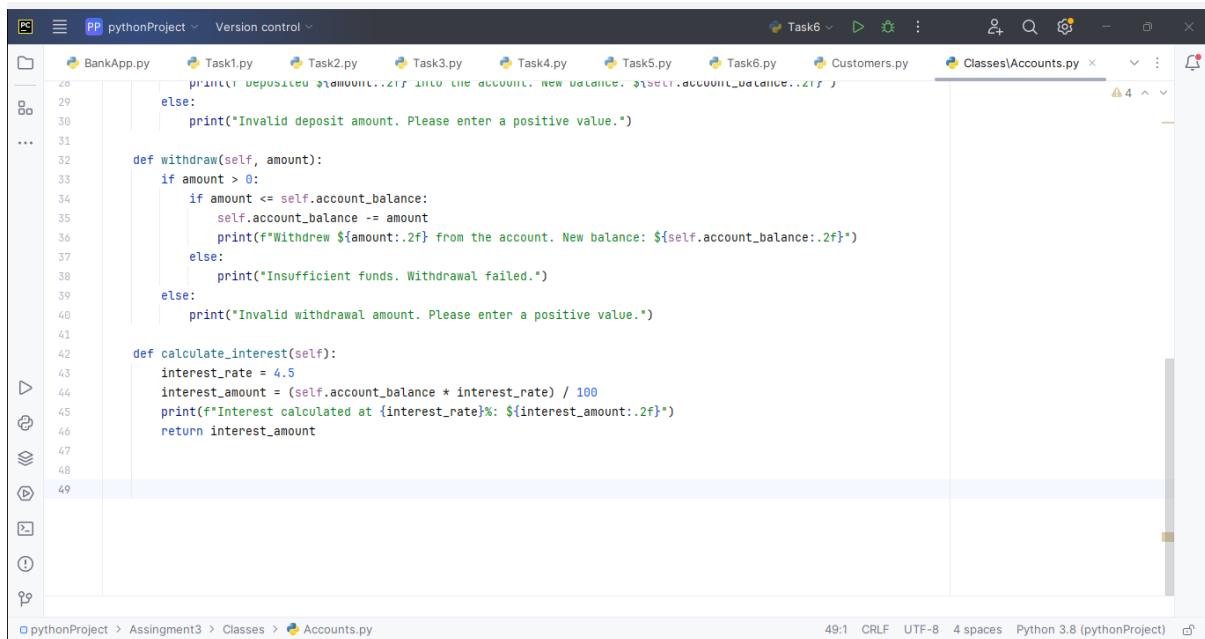
- o calculate\_interest(): Calculate and add interest to the account balance for savings accounts.

ACCOUNT:-



```
1 class Account:
2     def __init__(self, AccountID, AccountType, Balance):
3         self.account_number = AccountID
4         self.account_type = AccountType
5         self.account_balance = Balance
6
7     def get_account_number(self):
8         return self.account_number
9
10    def set_account_number(self, account_number):
11        self.account_number = account_number
12
13    def get_account_type(self):
14        return self.account_type
15
16    def set_account_type(self, account_type):
17        self.account_type = account_type
18
19    def get_account_balance(self):
20        return self.account_balance
21
22    def set_account_balance(self, account_balance):
23        self.account_balance = account_balance
24
25    def deposit(self, amount):
26        if amount > 0:
```

The screenshot shows a Python IDE with a file explorer on the left displaying a project structure: pythonProject > Assignment3 > Classes > Accounts.py. The main editor window shows the implementation of the Account class. The class has an \_\_init\_\_ method that initializes account\_number, account\_type, and account\_balance. It also includes getter and setter methods for each attribute, and deposit and set\_account\_balance methods. The deposit method includes a conditional check for a positive amount. The status bar at the bottom indicates the file encoding is UTF-8, 4 spaces for indentation, and the Python version is 3.8.



```
28 print(f'Deposited ${amount:.2f} into the account. New balance: ${self.account_balance:.2f}')
29
30 else:
31     print("Invalid deposit amount. Please enter a positive value.")
32
33 def withdraw(self, amount):
34     if amount > 0:
35         if amount <= self.account_balance:
36             self.account_balance -= amount
37             print(f'Withdrew ${amount:.2f} from the account. New balance: ${self.account_balance:.2f}')
38         else:
39             print("Insufficient funds. Withdrawal failed.")
40     else:
41         print("Invalid withdrawal amount. Please enter a positive value.")
42
43 def calculate_interest(self):
44     interest_rate = 4.5
45     interest_amount = (self.account_balance * interest_rate) / 100
46     print(f'Interest calculated at {interest_rate}%: ${interest_amount:.2f}')
47     return interest_amount
48
49
```

## Task 8: Inheritance and polymorphism

1. Overload the deposit and withdraw methods in Account class as mentioned below.

deposit(amount: float): Deposit the specified amount into the account.

withdraw(amount: float): Withdraw the specified amount from the account.  
withdraw amount only if there is sufficient fund else display insufficient balance.

deposit(amount: int): Deposit the specified amount into the account.

withdraw(amount: int): Withdraw the specified amount from the account.  
withdraw amount only if there is sufficient fund else display insufficient balance.

deposit(amount: double): Deposit the specified amount into the account.

withdraw(amount: double): Withdraw the specified amount from the account.  
withdraw amount only if there is sufficient fund else display insufficient balance.

3. Create a Bank class to represent the banking system. Perform the following operation in main method:

Display menu for user to create object for account class by calling parameter constructor. Menu should display options `SavingsAccount` and

`CurrentAccount`. user can choose any one option to create account. use switch case for implementation.

deposit(amount: float): Deposit the specified amount into the account.

withdraw(amount: float): Withdraw the specified amount from the account.

For saving account withdraw amount only if there is sufficient fund else display insufficient balance.

For Current Account withdraw limit can exceed the available balance and should not exceed the overdraft limit.

calculate\_interest(): Calculate and add interest to the account balance for savings accounts.

**ANS: USE INHERITANCE IN TASK 7 [ACCOUNTS CLASS](#).**

### Task 9: Abstraction

1. Create an abstract class BankAccount that represents a generic bank account. It should include the following attributes and methods:

Attributes:

- o Account number.

- o Customer name.

- o Balance.

Constructors:

- o Implement default constructors and overload the constructor with Account attributes, generate getter and setter, print all information of attribute methods for the attributes.

Abstract methods:

- o deposit(amount: float): Deposit the specified amount into the account.

- o withdraw(amount: float): Withdraw the specified amount from the account (implement error handling for insufficient funds).

- o calculate\_interest(): Abstract method for calculating interest.

3. Create a Bank class to represent the banking system. Perform the following operation in main method:

deposit(amount: float): Deposit the specified amount into the account.

withdraw(amount: float): Withdraw the specified amount from the account.

For saving account withdraw amount only if there is sufficient fund else display insufficient balance. For Current Account withdraw limit can exceed the available balance and should not exceed the overdraft limit.

**ANS:-USE ABSTRACTION IN TASK 7 ACCOUNTS CLASS.**

Task 11: Interface/abstract class, and Single Inheritance, static variable

1. Create a 'Customer' class as mentioned above task.
2. Create an class 'Account' that includes the following attributes. Generate account number using static variable.

Account Number (a unique identifier).

Account Type (e.g., Savings, Current)

Account Balance

Customer (the customer who owns the account)

LastAccNo

**ANS:-TASK DONE IN TASK 7 ACCOUNTS CLASS AND CUSTOMER CLASS.**

3. Create three child classes that inherit the Account class and each class must contain below mentioned attribute:

SavingsAccount: A savings account that includes an additional attribute for interest rate. Saving account should be created with minimum balance 500.

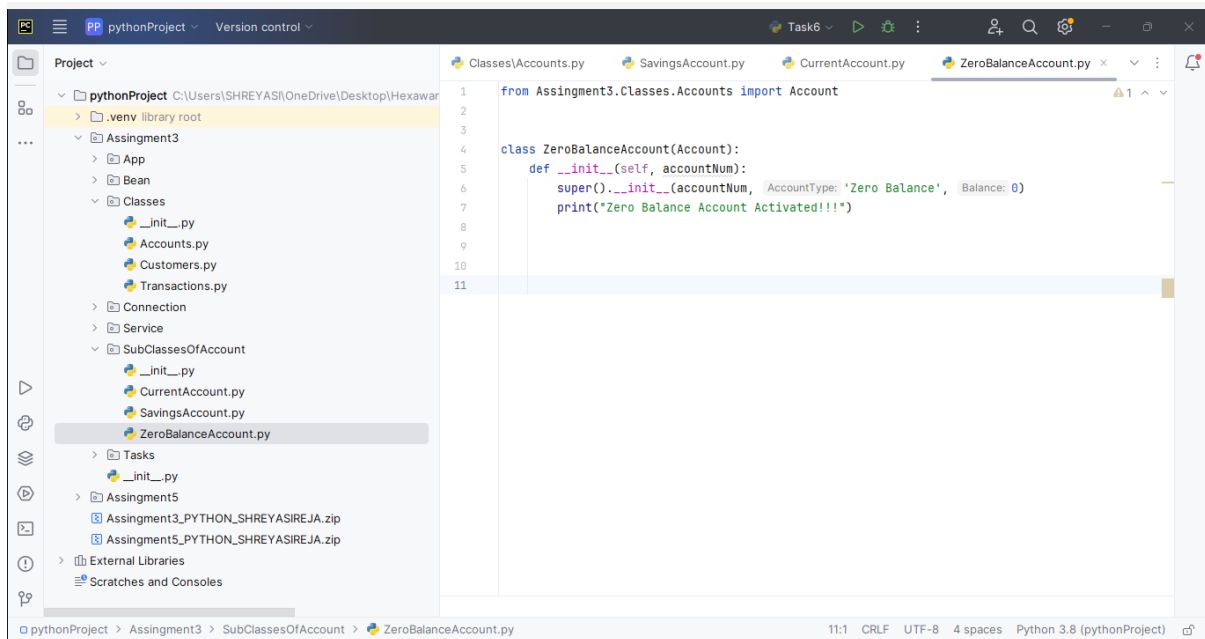


```
1 from Assignment3.Classes.Accounts import Account
2
3
4 class SavingsAccount(Account):
5     def __init__(self, accountNum):
6         super().__init__(accountNum, AccountType: 'Savings', Balance: 500)
7         self.interestRate = 4.5
8         print("Savings Account Activated!!!")
9
10    def calculateInterest(self):
11        interest = self.balance * (self.interestRate / 100)
12        self.balance += interest
13        print(f"After {interest} Interest, account balance is: {self.balance}")
14
15
16
```

**CurrentAccount:** A Current account that includes an additional attribute for overdraftLimit(credit limit). withdraw() method to allow overdraft up to a certain limit. withdraw limit can exceed the available balance and should not exceed the overdraft limit.

```
1 from Assignment3.Classes.Accounts import Account
2
3
4 2 usages
5 class CurrentAccount(Account):
6     LIMIT = 5000
7     def __init__(self, accountNum, balance):
8         super().__init__(accountNum, AccountType: 'Current', balance)
9         self.overdraftLimit = self.LIMIT
10        print("Current Account Activated!!!")
11
12
13
```

**ZeroBalanceAccount:** ZeroBalanceAccount can be created with Zero balance.



4. Create ICustomerServiceProvider interface/abstract class with following functions:

`get_account_balance(account_number: long)`: Retrieve the balance of an account given its account number. should return the current balance of account.

`deposit(account_number: long, amount: float)`: Deposit the specified amount into the account. Should return the current balance of account.

`withdraw(account_number: long, amount: float)`: Withdraw the specified amount from the account. Should return the current balance of account. A savings account should maintain a minimum balance and checking if the withdrawal violates the minimum balance rule.

`transfer(from_account_number: long, to_account_number: int, amount: float)`: Transfer money from one account to another.

`getAccountDetails(account_number: long)`: Should return the account and customer details.

```

1 from abc import ABC, abstractmethod
2
3
4 class ICustomerServiceProvider(ABC):
5     @abstractmethod
6     def get_account_balance(self, accountNum):
7         pass
8
9     @abstractmethod
10    def deposit(self, accountNum, amount):
11        pass
12
13
14    @abstractmethod
15    def withdraw(self, accountNum, amount):
16        pass
17
18    @abstractmethod
19    def transfer(self, fromAccountNum, toAccountNum, amount):
20        pass
21
22    @abstractmethod
23    def get_AccountDetails(self, accountNum):
24        pass
25
26    @abstractmethod
27    def get_Transactions(self):
28        pass

```

```

1 pass
2
3
4 @abstractmethod
5 def deposit(self, accountNum, amount):
6     pass
7
8
9 @abstractmethod
10 def withdraw(self, accountNum, amount):
11     pass
12
13
14 @abstractmethod
15 def transfer(self, fromAccountNum, toAccountNum, amount):
16     pass
17
18
19 @abstractmethod
20 def get_AccountDetails(self, accountNum):
21     pass
22
23
24 @abstractmethod
25 def get_Transactions(self):
26     pass

```

5. Create IBankServiceProvider interface/abstract class with following functions:

create\_account(Customer customer, long accNo, String accType, float balance): Create a new bank account for the given customer with the initial balance.

listAccounts():Account[] accounts: List all accounts in the bank.

calculateInterest(): the calculate\_interest() method to calculate interest based on the balance and interest rate.

```

1 from abc import ABC, abstractmethod
2
3
4 class IBankServiceProvider(ABC):
5     @abstractmethod
6     def create_account(self):
7         pass
8
9     @abstractmethod
10    def listAccounts(self):
11        pass
12
13    @abstractmethod
14    def get_AccountDetails(self, accountNum):
15        pass
16
17    @abstractmethod
18    def calculateInterest(self, inRate):
19        pass
20

```

IBankServiceProvider > get\_AccountDetails()

pythonProject > Assignment3 > Service > IBankServiceProvider.py

13:20 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)

6. Create CustomerServiceProviderImpl class which implements ICustomerServiceProvider provide all implementation methods.

```

1 from Assignment3.Service.ICustomerServiceProvider import ICustomerServiceProvider
2 from Assignment3.SubClassesOfAccount.CurrentAccount import CurrentAccount
3
4
5 class CustomerServiceProviderImpl(ICustomerServiceProvider):
6     def __init__(self, dbUtil):
7         self.dbUtil = dbUtil
8
9     def get_account_balance(self, accountNum):
10        query = "Select balance from accounts where accountID = %s"
11        value = (accountNum,)
12        result = self.dbUtil.fetchOne(query, value)
13        return result[0]
14
15    def deposit(self, accountNum, amount):
16        if amount <= 0:
17            print("Invalid Amount")
18        else:
19            if self.get_AccountDetails(accountNum) is not None:
20                balance = self.get_account_balance(accountNum)
21                newBal = float(balance) + amount
22                query = "Update accounts set balance=%s where accountID = %s"
23                values = (newBal, accountNum)
24                self.dbUtil.execute(query, values)
25

```

CustomerServiceProviderImpl > get\_account\_balance()

pythonProject > Assignment3 > Bean > CustomerServiceProviderImpl.py

9:47 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)

This screenshot shows the 'deposit' method in the 'CustomerServiceProviderImpl.py' file. The method takes 'self', 'accountNum', and 'amount' as parameters. It first checks if the amount is less than or equal to 0; if so, it prints 'Invalid Amount'. Otherwise, it checks if the account details exist. If they do, it calculates the new balance by adding the amount to the current balance, constructs an SQL query to update the account balance, and executes it. It then prints the new balance. If the account details do not exist, it raises an 'InvalidAccountIDException'.

```
15 def deposit(self, accountNum, amount):
16     if amount <= 0:
17         print("Invalid Amount")
18     else:
19         if self.get_AccountDetails(accountNum) is not None:
20             balance = self.get_account_balance(accountNum)
21             newBal = float(balance) + amount
22             query = "Update accounts set balance=%s where accountID = %s"
23             values = (newBal, accountNum)
24             self.dbUtil.executeQuery(query, values)
25             print(f"New Balance is: {newBal}")
26         else:
27             raise Exception("InvalidAccountIDException")
28
29 2 usages
30 def withdraw(self, accountNum, amount):
31     currentBal = self.get_account_balance(accountNum)
32     accountType = self.getAccountType(accountNum)
33
34     if self.get_AccountDetails(accountNum) is not None:
35         if amount <= 0:
36             return "Invalid Amount"
37         else:
38             if accountType == 'savings':
39                 if amount > currentBal:
40                     raise Exception("InsufficientFundException")
41             elif accountType == 'current':
42                 if amount > currentBal and amount - currentBal > CurrentAccount.LIMIT:
43                     raise Exception("OverdraftLimitExceededException")
44             else:
45                 query = "Update accounts set balance=balance-%s where accountID = %s"
46                 values = (amount, accountNum)
47                 self.dbUtil.executeQuery(query, values)
48                 result = self.get_account_balance(accountNum)
49                 return result
50             else:
51                 raise Exception("ZeroBalanceAccountException")
52         else:
53             raise Exception("InvalidAccountIDException")
54
55 CustomerServiceProviderImpl > get_account_balance()
```

pythonProject > Assingment3 > Bean > CustomerServiceProviderImpl.py 9:47 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)

This screenshot shows the 'withdraw' method in the 'CustomerServiceProviderImpl.py' file. It continues from the previous screenshot, starting with the 'else' block of the 'if self.get\_AccountDetails' check. It handles 'savings' and 'current' account types with specific balance and limit checks, raising exceptions like 'InsufficientFundException' or 'OverdraftLimitExceededException'. For other account types, it updates the balance and returns the result. If the account type is not recognized, it raises a 'ZeroBalanceAccountException'. Finally, it handles the 'InvalidAccountIDException' case.

```
36 else:
37     if accountType == 'savings':
38         if amount > currentBal:
39             raise Exception("InsufficientFundException")
40         else:
41             if float(currentBal) - amount < 500.00:
42                 raise Exception("MinimumBalanceLimitException")
43             else:
44                 query = "Update accounts set balance=balance-%s where accountID = %s"
45                 values = (amount, accountNum)
46                 self.dbUtil.executeQuery(query, values)
47                 result = self.get_account_balance(accountNum)
48                 return result
49     elif accountType == 'current':
50         if amount > currentBal and amount - currentBal > CurrentAccount.LIMIT:
51             raise Exception("OverdraftLimitExceededException")
52         else:
53             query = "Update accounts set balance=balance-%s where accountID = %s"
54             values = (amount, accountNum)
55             self.dbUtil.executeQuery(query, values)
56             result = self.get_account_balance(accountNum)
57             return result
58         else:
59             raise Exception("ZeroBalanceAccountException")
60     else:
61         raise Exception("InvalidAccountIDException")
62
63 CustomerServiceProviderImpl > get_account_balance()
```

pythonProject > Assingment3 > Bean > CustomerServiceProviderImpl.py 9:47 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)

```

58         else:
59             raise Exception("ZeroBalanceAccountException")
60         else:
61             raise Exception("InvalidAccountIDException")
62
63     1 usage
64     def transfer(self, fromAccountNum, toAccountNum, amount):
65         if self.get_AccountDetails(fromAccountNum) and self.get_AccountDetails(toAccountNum):
66             try:
67                 self.withdraw(fromAccountNum, amount)
68                 self.deposit(toAccountNum, amount)
69             except Exception as e:
70                 return f"Transfer Failed!!! Error: {e}"
71
72     6 usages
73     def get_AccountDetails(self, accountNum):
74         query = "Select * from accounts join customers on accounts.customerID = customers.customerID where accountID=%s"
75         value = (accountNum,)
76         result = self.dbUtil.fetchall(query, value)
77         return result
78
79     1 usage
80     def getTransactions(self):
81         accountNum = input("Enter your accountID: ")
82         fromDate = input("From: ")
83         toDate = input("To: ")
84
85 CustomerServiceProviderImpl > get_account_balance()
pythonProject > Assignment3 > Bean > CustomerServiceProviderImpl.py
9:47 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)

```

```

71     def get_AccountDetails(self, accountNum):
72         query = "Select * from accounts join customers on accounts.customerID = customers.customerID where accountID=%s"
73         value = (accountNum,)
74         result = self.dbUtil.fetchall(query, value)
75         return result
76
77     1 usage
78     def getTransactions(self):
79         accountNum = input("Enter your accountID: ")
80         fromDate = input("From: ")
81         toDate = input("To: ")
82         if self.get_AccountDetails(accountNum) is not None:
83             query = "Select * from transactions where transaction_date between %s and %s and accountID = %s"
84             values = (fromDate, toDate, accountNum)
85             result = self.dbUtil.fetchall(query, values)
86             return result
87         else:
88             raise Exception("InvalidAccountIDException")
89
90     2 usages
91     def getAccountType(self, accountNum):
92         value = (accountNum,)
93         result = self.dbUtil.fetchOne("Select account_type from accounts where accountID = %s", value)
94         return result[0]
95
96 CustomerServiceProviderImpl > get_account_balance()
pythonProject > Assignment3 > Bean > CustomerServiceProviderImpl.py
9:47 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)

```

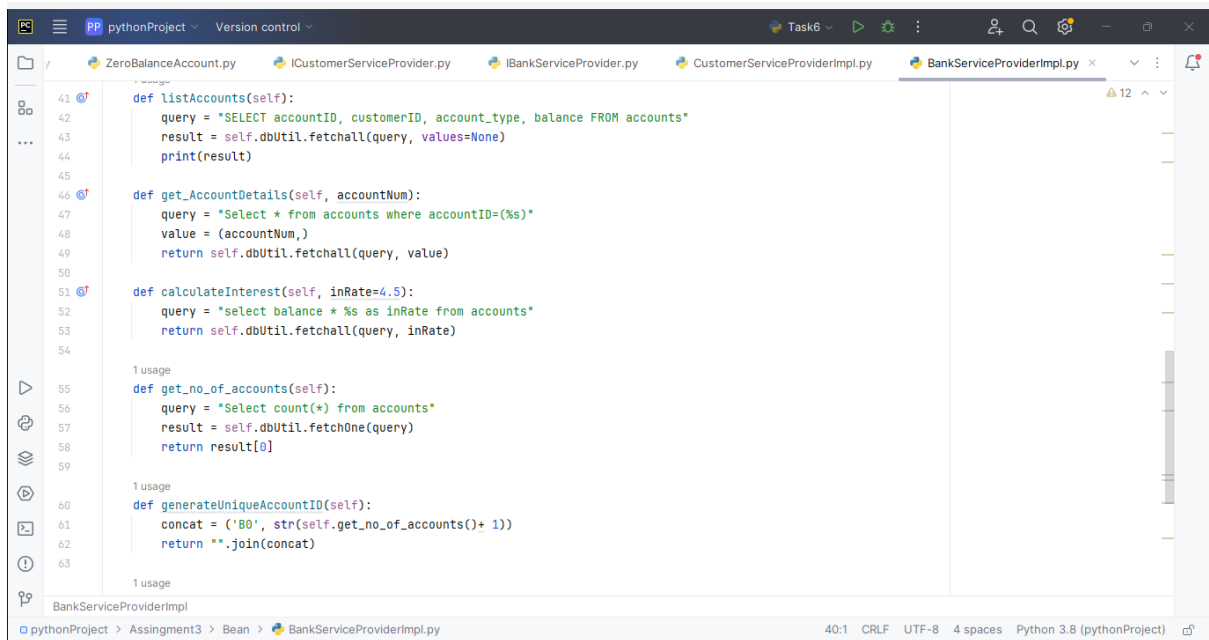
7. Create BankServiceProviderImpl class which inherits from CustomerServiceProviderImpl and implements IBankServiceProvider

Attributes

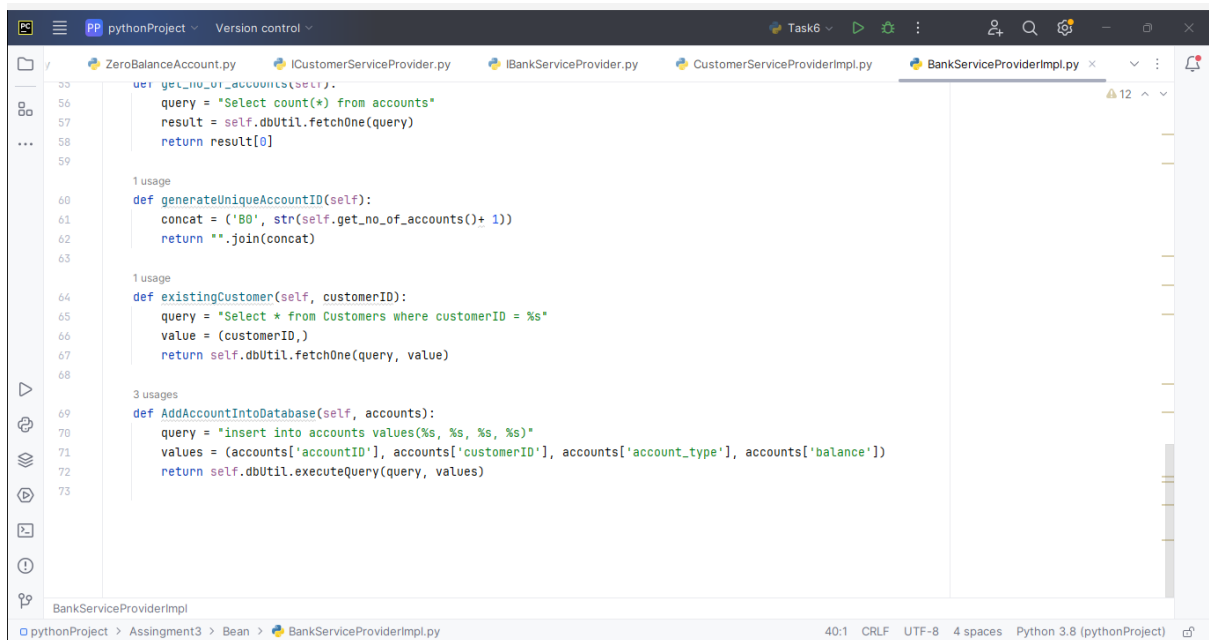
- o accountList: Array of Accounts to store any account objects.
- o branchName and branchAddress as String objects

```
pythonProject Version control Task6
ZeroBalanceAccount.py ICustomerServiceProvider.py IBankServiceProvider.py CustomerServiceProviderImpl.py BankServiceProviderImpl.py
1 from Assingment3.Service.IBankServiceProvider import IBankServiceProvider
2
3
4 2 usages
5 class BankServiceProviderImpl(IBankServiceProvider):
6     def __init__(self, dbUtil):
7         self.dbUtil = dbUtil
8
9     1 usage
10     def create_account(self):
11         customerID = input("Please enter your customerID: ")
12         if self.existingCustomer(customerID) is None:
13             raise Exception("CustomerNotRegisteredException")
14         else:
15             print("Please fill up the account details: ")
16             account = {
17                 'accountID': self.generateUniqueAccountID(),
18                 'customerID': customerID,
19             }
20             print("What type of account do you want to create?")
21             print("1. Savings Account")
22             print("2. Current Account")
23             print("3. Zero Balance Account")
24             ch = int(input("Enter your choice: "))
25
26             if ch == 1:
27                 account['account_type'] = 'savings'
```

```
pythonProject Version control Task6
ZeroBalanceAccount.py ICustomerServiceProvider.py IBankServiceProvider.py CustomerServiceProviderImpl.py BankServiceProviderImpl.py
24
25         account['account_type'] = 'savings'
26         account['balance'] = float(input("Enter the amount: "))
27         return self.AddAccountIntoDatabase(account)
28
29     elif ch == 2:
30         account['account_type'] = 'current'
31         account['balance'] = float(input("Enter the amount: "))
32         return self.AddAccountIntoDatabase(account)
33
34     elif ch == 3:
35         account['account_type'] = 'zero balance'
36         account['balance'] = 0.0
37         return self.AddAccountIntoDatabase(account)
38     else:
39         print("Invalid choice. Please enter a valid choice.")
40
41 1 usage
42     def listAccounts(self):
43         query = "SELECT accountID, customerID, account_type, balance FROM accounts"
44         result = self.dbUtil.fetchall(query, values=None)
45         print(result)
46
47     def get_AccountDetails(self, accountNum):
48         query = "Select * from accounts where accountID=(%s)"
49         value = (accountNum,)
```



```
41 def listAccounts(self):
42     query = "SELECT accountID, customerID, account_type, balance FROM accounts"
43     result = self.dbUtil.fetchall(query, values=None)
44     print(result)
45
46 def get_AccountDetails(self, accountNum):
47     query = "Select * from accounts where accountID=(%s)"
48     value = (accountNum,)
49     return self.dbUtil.fetchall(query, value)
50
51 def calculateInterest(self, inRate=4.5):
52     query = "select balance * %s as inRate from accounts"
53     return self.dbUtil.fetchall(query, inRate)
54
55 1 usage
56 def get_no_of_accounts(self):
57     query = "Select count(*) from accounts"
58     result = self.dbUtil.fetchOne(query)
59     return result[0]
60
61 1 usage
62 def generateUniqueAccountID(self):
63     concat = ('B0', str(self.get_no_of_accounts()+ 1))
64     return "".join(concat)
65
66 1 usage
```



```
56 def get_no_of_accounts(self):
57     query = "Select count(*) from accounts"
58     result = self.dbUtil.fetchOne(query)
59     return result[0]
60
61 1 usage
62 def generateUniqueAccountID(self):
63     concat = ('B0', str(self.get_no_of_accounts()+ 1))
64     return "".join(concat)
65
66 1 usage
67 def existingCustomer(self, customerID):
68     query = "Select * from Customers where customerID = %s"
69     value = (customerID,)
70     return self.dbUtil.fetchOne(query, value)
71
72 3 usages
73 def AddAccountIntoDatabase(self, accounts):
74     query = "insert into accounts values(%s, %s, %s, %s)"
75     values = (accounts['accountID'], accounts['customerID'], accounts['account_type'], accounts['balance'])
76     return self.dbUtil.executeQuery(query, values)
77
78 1 usage
```

8. Create BankApp class and perform following operation:

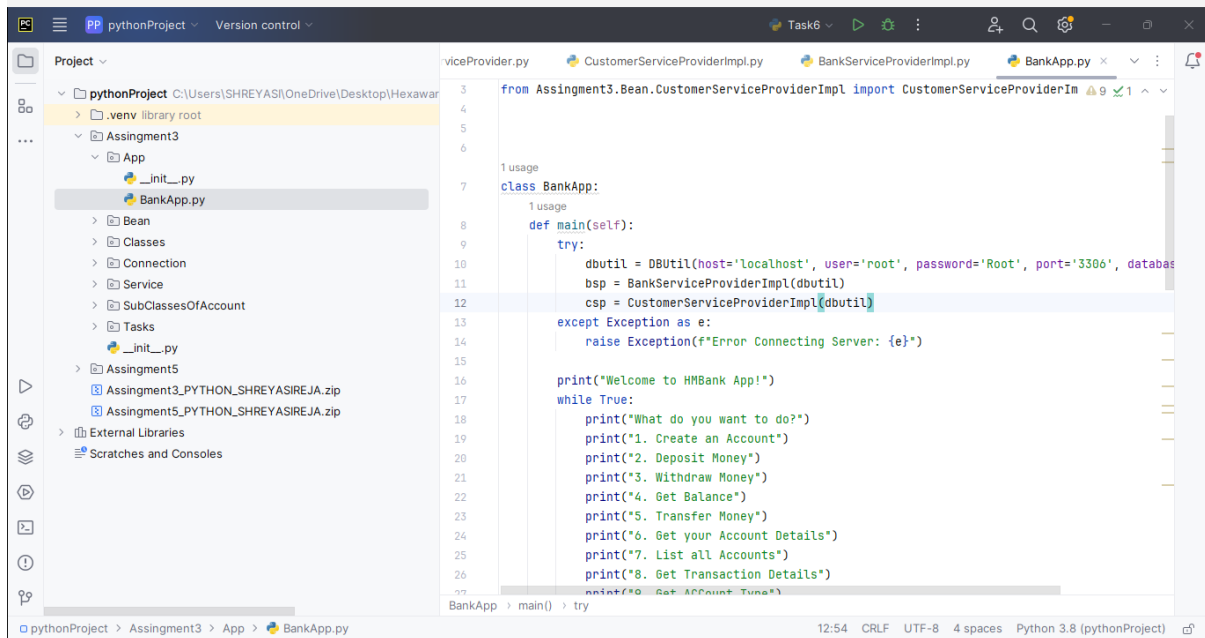
main method to simulate the banking system. Allow the user to interact with the system by entering choice from menu such as "create\_account", "deposit", "withdraw", "get\_balance", "transfer", "getAccountDetails", "ListAccounts" and "exit."

create\_account should display sub menu to choose type of accounts and repeat this operation until user exit.

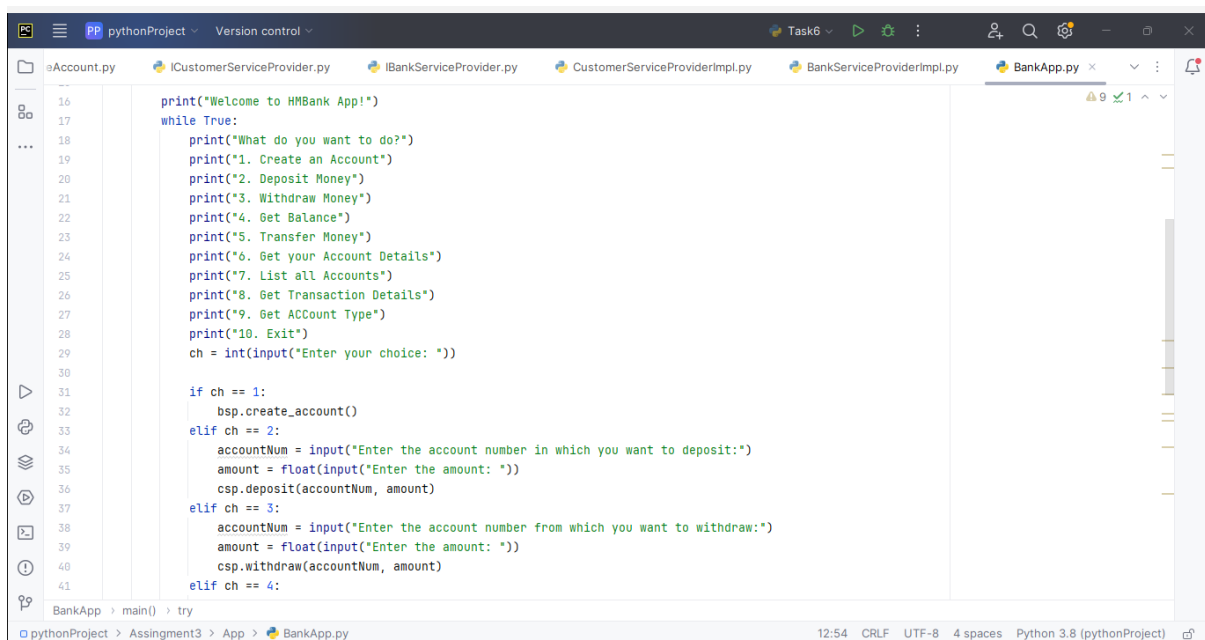


9. Place the interface/abstract class in service package and interface/abstract class implementation class, account class in bean package and Bank class in app package.

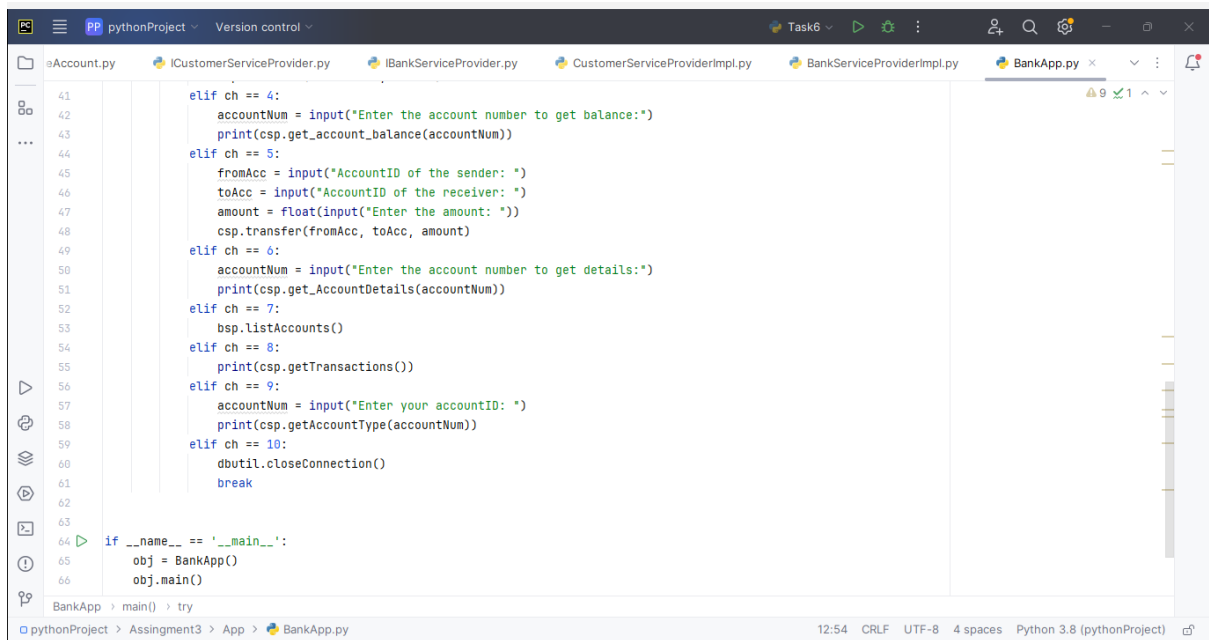
**BANK APP:-**



```
1 from Assignment3.Bean.CustomerServiceProvderImpl import CustomerServiceProvderIm
2
3
4
5
6
7 class BankApp:
8     def main(self):
9         try:
10             dbutil = DBUtil(host='localhost', user='root', password='Root', port='3306', database='bank')
11             bsp = BankServiceProviderImpl(dbutil)
12             csp = CustomerServiceProvderImpl(dbutil)
13         except Exception as e:
14             raise Exception(f"Error Connecting Server: {e}")
15
16     print("Welcome to HMBank App!")
17     while True:
18         print("What do you want to do?")
19         print("1. Create an Account")
20         print("2. Deposit Money")
21         print("3. Withdraw Money")
22         print("4. Get Balance")
23         print("5. Transfer Money")
24         print("6. Get your Account Details")
25         print("7. List all Accounts")
26         print("8. Get Transaction Details")
27         print("9. Get Account Type")
28         print("10. Exit")
29         ch = int(input("Enter your choice: "))
30
31         if ch == 1:
32             bsp.create_account()
33         elif ch == 2:
34             accountNum = input("Enter the account number in which you want to deposit:")
35             amount = float(input("Enter the amount: "))
36             csp.deposit(accountNum, amount)
37         elif ch == 3:
38             accountNum = input("Enter the account number from which you want to withdraw:")
39             amount = float(input("Enter the amount: "))
40             csp.withdraw(accountNum, amount)
41         elif ch == 4:
```

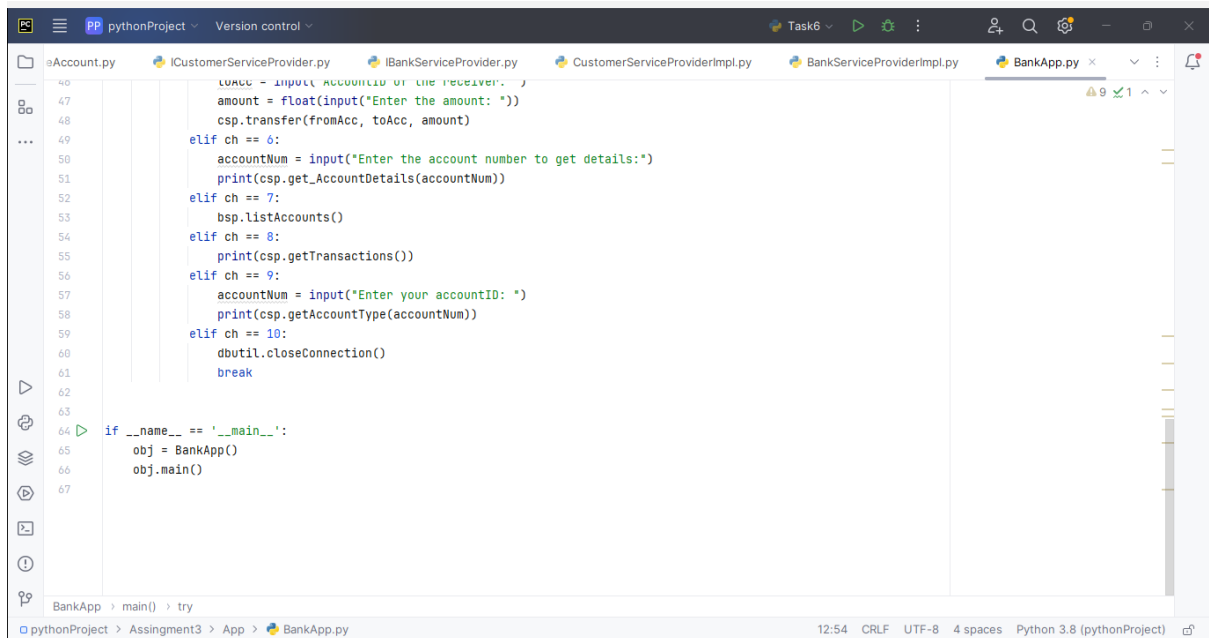


```
16     print("Welcome to HMBank App!")
17     while True:
18         print("What do you want to do?")
19         print("1. Create an Account")
20         print("2. Deposit Money")
21         print("3. Withdraw Money")
22         print("4. Get Balance")
23         print("5. Transfer Money")
24         print("6. Get your Account Details")
25         print("7. List all Accounts")
26         print("8. Get Transaction Details")
27         print("9. Get Account Type")
28         print("10. Exit")
29         ch = int(input("Enter your choice: "))
30
31         if ch == 1:
32             bsp.create_account()
33         elif ch == 2:
34             accountNum = input("Enter the account number in which you want to deposit:")
35             amount = float(input("Enter the amount: "))
36             csp.deposit(accountNum, amount)
37         elif ch == 3:
38             accountNum = input("Enter the account number from which you want to withdraw:")
39             amount = float(input("Enter the amount: "))
40             csp.withdraw(accountNum, amount)
41         elif ch == 4:
```



```
41         elif ch == 4:
42             accountNum = input("Enter the account number to get balance:")
43             print(csp.get_account_balance(accountNum))
44         elif ch == 5:
45             fromAcc = input("AccountID of the sender: ")
46             toAcc = input("AccountID of the receiver: ")
47             amount = float(input("Enter the amount: "))
48             csp.transfer(fromAcc, toAcc, amount)
49         elif ch == 6:
50             accountNum = input("Enter the account number to get details:")
51             print(csp.get_AccountDetails(accountNum))
52         elif ch == 7:
53             bsp.listAccounts()
54         elif ch == 8:
55             print(csp.getTransactions())
56         elif ch == 9:
57             accountNum = input("Enter your accountID: ")
58             print(csp.getAccountType(accountNum))
59         elif ch == 10:
60             dbutil.closeConnection()
61             break
62
63
64     if __name__ == '__main__':
65         obj = BankApp()
66         obj.main()
67
68 BankApp > main() > try
```

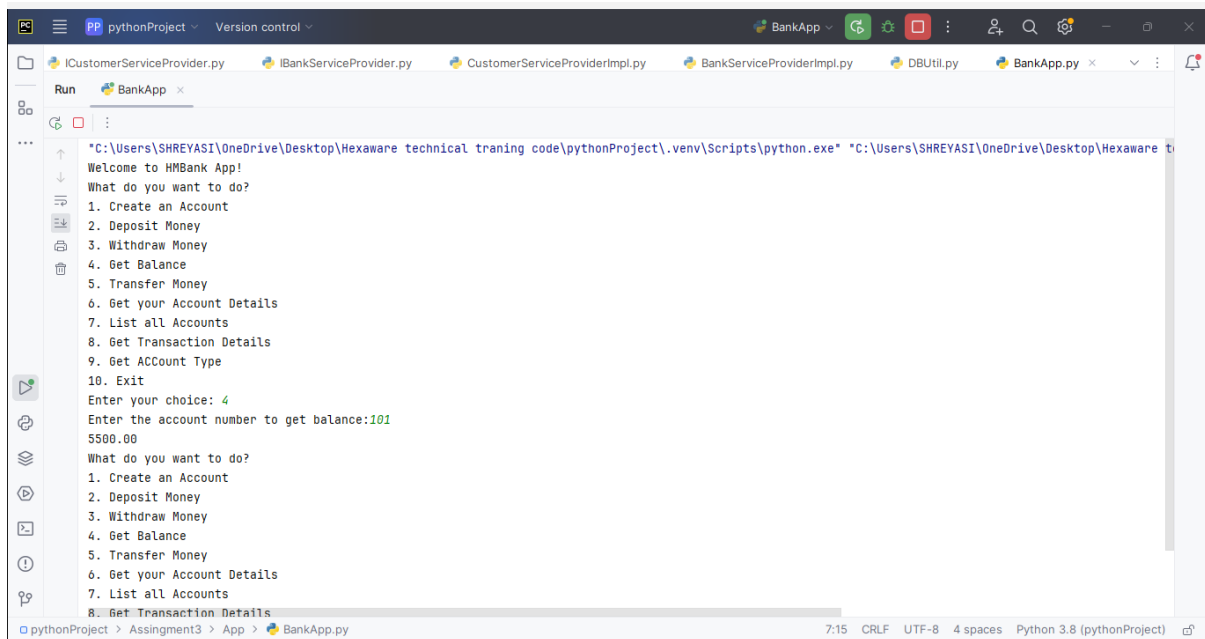
pythonProject > Assingment3 > App > BankApp.py 12:54 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)



```
46         toAcc = input("AccountID of the receiver: ")
47         amount = float(input("Enter the amount: "))
48         csp.transfer(fromAcc, toAcc, amount)
49     elif ch == 6:
50         accountNum = input("Enter the account number to get details:")
51         print(csp.get_AccountDetails(accountNum))
52     elif ch == 7:
53         bsp.listAccounts()
54     elif ch == 8:
55         print(csp.getTransactions())
56     elif ch == 9:
57         accountNum = input("Enter your accountID: ")
58         print(csp.getAccountType(accountNum))
59     elif ch == 10:
60         dbutil.closeConnection()
61         break
62
63
64     if __name__ == '__main__':
65         obj = BankApp()
66         obj.main()
67
68 BankApp > main() > try
```

pythonProject > Assingment3 > App > BankApp.py 12:54 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)

OUTPUT:-



```
*C:\Users\SHREYASI\OneDrive\Desktop\Hexaware technical training code\pythonProject\.venv\Scripts\python.exe" *C:\Users\SHREYASI\OneDrive\Desktop\Hexaware t
Welcome to HMBank App!
What do you want to do?
1. Create an Account
2. Deposit Money
3. Withdraw Money
4. Get Balance
5. Transfer Money
6. Get your Account Details
7. List all Accounts
8. Get Transaction Details
9. Get Account Type
10. Exit
Enter your choice: 4
Enter the account number to get balance:101
5500.00
What do you want to do?
1. Create an Account
2. Deposit Money
3. Withdraw Money
4. Get Balance
5. Transfer Money
6. Get your Account Details
7. List all Accounts
8. Get Transaction Details
```

## Task 12: Exception Handling

throw the exception whenever needed and Handle in main method,

1. `InsufficientFundException` throw this exception when user try to withdraw amount or transfer amount to another account and the account runs out of money in the account.
2. `InvalidAccountException` throw this exception when user entered the invalid account number when tries to transfer amount, get account details classes.
3. `OverDraftLimitExcededException` thow this exception when current account customer try to with draw amount from the current account.
4. `NullPointerException` handle in main method.

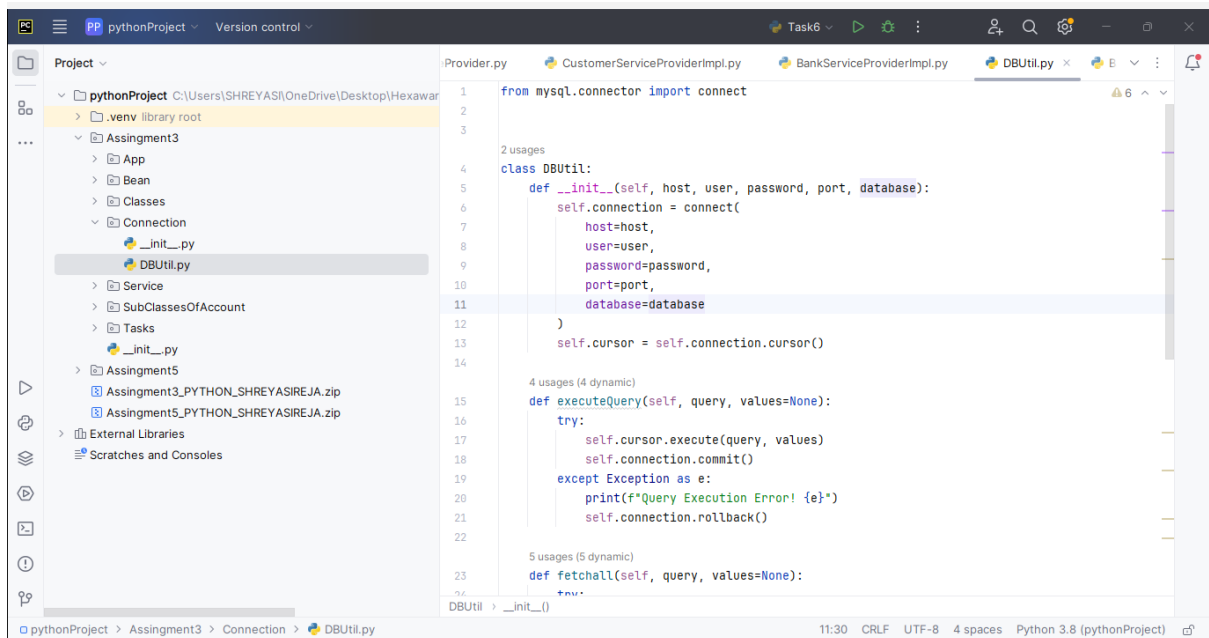
Throw these exceptions from the methods in `HMBank` class. Make necessary changes to accommodate these exception in the source code. Handle all these exceptions from the main program.

**ANS:-USE EXCEPTION HANDLING IN TASK 11 SUBTASK 8 [BANK APP](#).**

## Task 14: Database Connectivity.

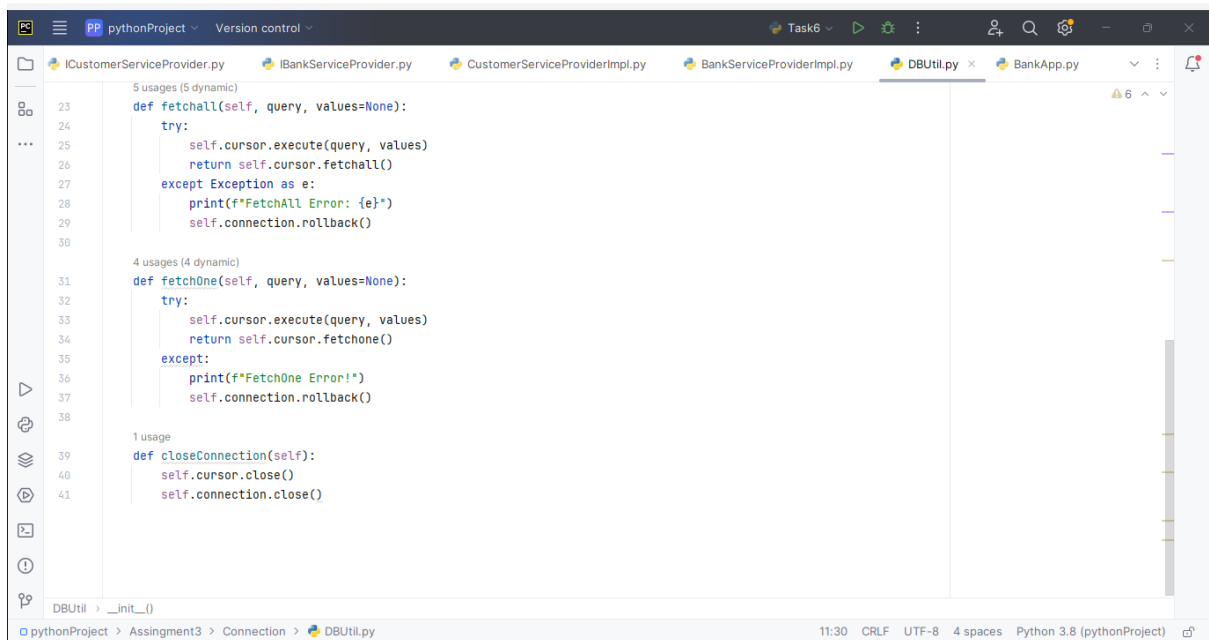
11. Create `DBUtil` class and add the following method.

`static getDBConn():Connection` Establish a connection to the database and return Connection reference



The screenshot shows the VS Code editor with the 'pythonProject' workspace. The file explorer on the left shows the project structure, including 'Assingment3' and 'Connection' folders. The 'DBUtil.py' file is open in the editor, showing the following code:

```
1 from mysql.connector import connect
2
3
4 class DBUtil:
5     def __init__(self, host, user, password, port, database):
6         self.connection = connect(
7             host=host,
8             user=user,
9             password=password,
10            port=port,
11            database=database
12        )
13        self.cursor = self.connection.cursor()
14
15     def executeQuery(self, query, values=None):
16         try:
17             self.cursor.execute(query, values)
18             self.connection.commit()
19         except Exception as e:
20             print(f"Query Execution Error! {e}")
21             self.connection.rollback()
22
23     def fetchall(self, query, values=None):
24         try:
25             self.cursor.execute(query, values)
26             return self.cursor.fetchall()
27         except Exception as e:
28             print(f"FetchAll Error: {e}")
29             self.connection.rollback()
30
31     def fetchOne(self, query, values=None):
32         try:
33             self.cursor.execute(query, values)
34             return self.cursor.fetchone()
35         except:
36             print(f"FetchOne Error!")
37             self.connection.rollback()
38
39     def closeConnection(self):
40         self.cursor.close()
41         self.connection.close()
```



The screenshot shows the VS Code editor with the 'pythonProject' workspace. The file explorer on the left shows the project structure, including 'Assingment3' and 'Connection' folders. The 'BankApp.py' file is open in the editor, showing the following code:

```
1 from DBUtil import DBUtil
2
3 def main():
4     host = input("Enter Host: ")
5     user = input("Enter User: ")
6     password = input("Enter Password: ")
7     port = input("Enter Port: ")
8     database = input("Enter Database: ")
9
10    dbutil = DBUtil(host, user, password, port, database)
11
12    while True:
13        print("\nBanking System Menu")
14        print("1. create_account")
15        print("2. deposit")
16        print("3. withdraw")
17        print("4. get_balance")
18        print("5. transfer")
19        print("6. getAccountDetails")
20        print("7. ListAccounts")
21        print("8. getTransactions")
22        print("9. exit")
23
24        choice = input("Enter your choice: ")
25
26        if choice == "1":
27            create_account(dbutil)
28        elif choice == "2":
29            deposit(dbutil)
30        elif choice == "3":
31            withdraw(dbutil)
32        elif choice == "4":
33            get_balance(dbutil)
34        elif choice == "5":
35            transfer(dbutil)
36        elif choice == "6":
37            getAccountDetails(dbutil)
38        elif choice == "7":
39            ListAccounts(dbutil)
40        elif choice == "8":
41            getTransactions(dbutil)
42        elif choice == "9":
43            exit()
44
45    dbutil.closeConnection()
46
47 if __name__ == "__main__":
48     main()
```

## 12. Create BankApp class and perform following operation:

main method to simulate the banking system. Allow the user to interact with the system by entering choice from menu such as "create\_account", "deposit", "withdraw", "get\_balance", "transfer", "getAccountDetails", "ListAccounts", "getTransactions" and "exit."

create\_account should display sub menu to choose type of accounts and repeat this operation until user exit.

13. Place the interface/abstract class in service package and interface/abstract class implementation class, account class in bean package and Bank class in app package.

14. Should throw appropriate exception as mentioned in above task along with handle SQLException.

ANS:-USE IN TASK 11 SUB TASK 8 [BANK APP.](#)