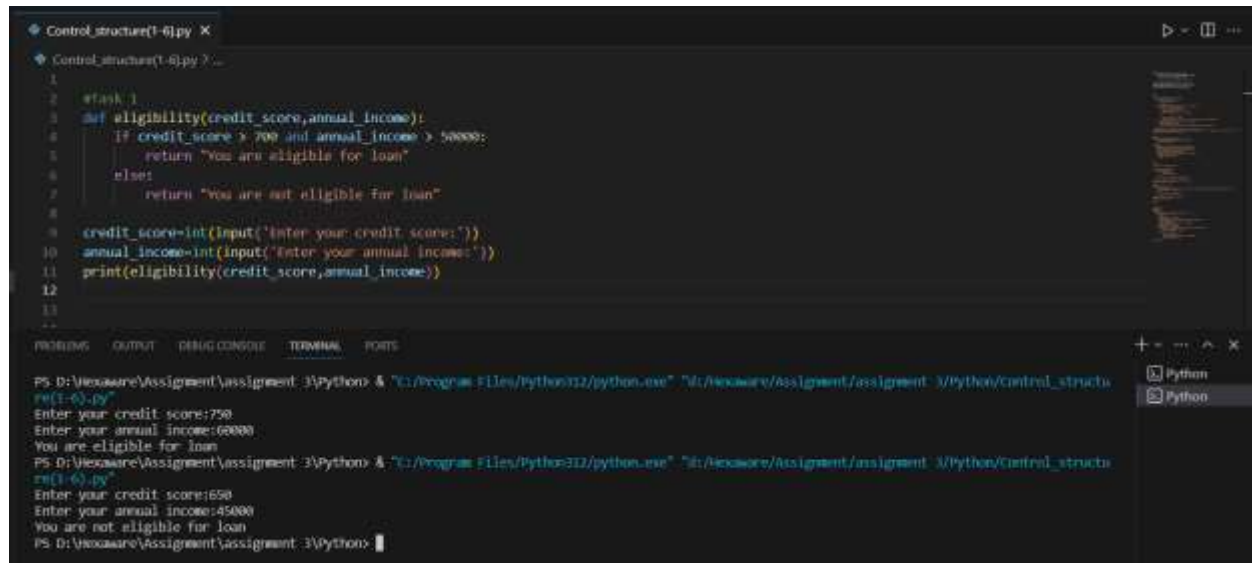


Assignment 3

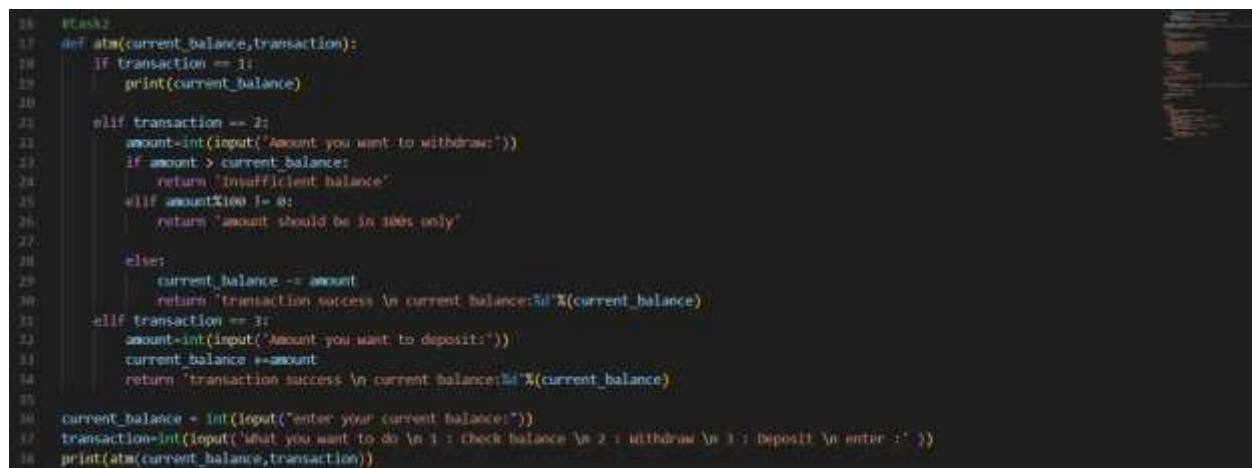
Banking System

Task1(With outputs):



```
Control_structure(1-6).py X
Control_structure(1-6).py
1
2 #Task 1
3 def eligibility(credit_score,annual_income):
4     if credit_score > 700 and annual_income > 50000:
5         return "You are eligible for loan"
6     else:
7         return "You are not eligible for loan"
8
9 credit_score=int(input("Enter your credit score:"))
10 annual_income=int(input("Enter your annual income:"))
11 print(eligibility(credit_score,annual_income))
12
13
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\Hexaware\Assignment\Assignment 3\Python & "C:\Program Files\Python312\python.exe" "D:\Hexaware\Assignment\assignment 3\python/control_structu
r(1-6).py"
Enter your credit score:750
Enter your annual income:60000
You are eligible for loan
PS D:\Hexaware\Assignment\Assignment 3\Python & "C:\Program Files\Python312\python.exe" "D:\Hexaware\Assignment\assignment 3\python/control_structu
r(1-6).py"
Enter your credit score:650
Enter your annual income:45000
You are not eligible for loan
PS D:\Hexaware\Assignment\Assignment 3\Python > |
```

Task2:



```
28 #Task2
29 def atm(current_balance,transaction):
30     if transaction == 1:
31         print(current_balance)
32
33     elif transaction == 2:
34         amount=int(input("Amount you want to withdraw:"))
35         if amount > current_balance:
36             return "Insufficient balance"
37         elif amount%100 != 0:
38             return "amount should be in 100s only"
39
40     else:
41         current_balance -= amount
42         return "transaction success \n current balance:%d"%(current_balance)
43
44     elif transaction == 3:
45         amount=int(input("Amount you want to deposit:"))
46         current_balance +=amount
47         return "transaction success \n current balance:%d"%(current_balance)
48
49 current_balance = int(input("enter your current balance:"))
50 transaction=int(input("What you want to do \n 1 : Check balance \n 2 : withdraw \n 3 : deposit \n enter :"))
51 print(atm(current_balance,transaction))
```

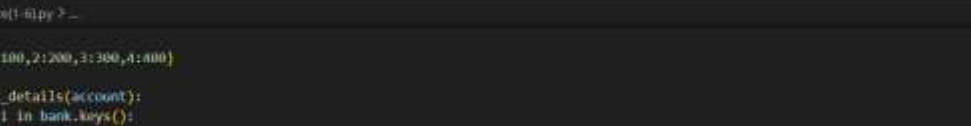
Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\Vocare\Assignment\assignment 3\Python & "C:\Program Files\Python112\python.exe" "d:\Vocare\Assignment\assignment 3\Python\Control_structure(1-6).py"
enter your current balance:1000
What you want to do
1 : Check balance
2 : Withdraw
3 : Deposit
enter :1
1000
None
PS D:\Vocare\Assignment\assignment 3\Python & "C:\Program Files\Python112\python.exe" "d:\Vocare\Assignment\assignment 3\Python\Control_structure(1-6).py"
enter your current balance:2000
What you want to do
1 : Check balance
2 : Withdraw
3 : Deposit
enter :2
Amount you want to withdraw:1000
transaction success
current balance:1000
PS D:\Vocare\Assignment\assignment 3\Python & "C:\Program Files\Python112\python.exe" "d:\Vocare\Assignment\assignment 3\Python\Control_structure(1-6).py"
enter your current balance:2000
What you want to do
1 : Check balance
2 : Withdraw
3 : Deposit
enter :3
Amount you want to deposit:1000
transaction success
current balance:3000
PS D:\Vocare\Assignment\assignment 3\Python >
```

Task3(With output):

```
Control_structure(1-6).py X
Control_structure(1-6).py 2...
41. #Task 3
42. def future(balance, interest, years):
43.     future_balance = balance * (1 + interest/100)**years
44.     print(future_balance)
45.
46. while True:
47.     balance = int(input('Enter your initial balance:'))
48.     interest = float(input('Enter your interest rate:'))
49.     years = int(input('Enter your number of years:'))
50.     print(future(balance, interest, years))
51.     ask = input('do you want to check for next customer:')
52.     if ask in ['no', 'No', 'NO', 'no']:
53.         print('Process ended')
54.         break
55.
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\Vocare\Assignment\assignment 3\Python & "C:\Program Files\Python112\python.exe" "d:\Vocare\Assignment\assignment 3\Python\Control_structure(1-6).py"
Enter your initial balance:1000
Enter your interest rate:7.5
Enter your number of years:5
1435.6293261718747
None
do you want to check for next customer:no
Process ended
PS D:\Vocare\Assignment\assignment 3\Python >
```

Task4:



```
Control_structure1-6.py X
Control_structure1-6.py
57 #Task-4
58 bank={1:100,2:200,3:300,4:400}
59
60 def bank_details(account):
61     for i in bank.keys():
62         if account == i:
63             return bank.get(i)
64     else:
65         return 'Enter correct account number'
66
67 account=int(input('Enter Bank account number:'))
68 print(bank_details(account))
69
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\hexaware\Assignment\assignment 3\Python> "C:/Program Files/Python312/python.exe" "d:/hexaware/Assignment/assignment 3/Python/Control_structure1-6.py"
Enter Bank account number:1
100
PS D:\hexaware\Assignment\assignment 3\Python> "C:/Program Files/Python312/python.exe" "d:/hexaware/Assignment/assignment 3/Python/Control_structure1-6.py"
Enter Bank account number:3
300
PS D:\hexaware\Assignment\assignment 3\Python>
```

Task5:

The image shows a code editor window with a Python file named 'Control_structure(1-6).py'. The code defines a function 'password' that checks if a password is at least 7 characters long and contains at least one digit. It prompts the user to enter a password and prints the result. The terminal window below shows the execution of the script, where the password 'rohan12345678' is entered twice, and both times it is accepted as a valid password.

```

Control_structure(1-6).py
Control_structure(1-6).py ? ...

70
71
72 #tasks
73 def password(password):
74     if len(password) > 7 and password.islower() == False and any(map(str.isdigit,password)) == True:
75         print('Valid password')
76     else:
77         print('not a valid password')
78
79 inp_pass=str(input('Enter password:'))
80 print(password(inp_pass))
81
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + - - - - -
PS D:\Hexaware\Assignment\assignment 3\Python > "C:\Program Files\Python312\python.exe" "D:\Hexaware\Assignment\assignment 3\Python\Control_structure(1-6).py"
Enter password:rohan12345678 "C:\Program Files\Python312\python.exe" "D:\Hexaware\Assignment\assignment 3\Python\Control_structure(1-6).py"
Valid password
None
PS D:\Hexaware\Assignment\assignment 3\Python > "C:\Program Files\Python312\python.exe" "D:\Hexaware\Assignment\assignment 3\Python\Control_structure(1-6).py"
Enter password:rohan12345678 "C:\Program Files\Python312\python.exe" "D:\Hexaware\Assignment\assignment 3\Python\Control_structure(1-6).py"
Valid password
None
PS D:\Hexaware\Assignment\assignment 3\Python >

```

Task6:

```
Control_structure1-61.py
Control_structure1-61.py
87 #task6
88 balance=100
89 output=[]
90 while True:
91     transaction=str(input('what you want to do:'))
92     if transaction == 'exit':
93         break
94     elif transaction == 'deposit':
95         deposit=int(input('Enter amount you want to deposit:'))
96         balance+=deposit
97         history=('amount deposited:',str(deposit))
98         output.append(history)
99     elif transaction == 'withdraw':
100         withdraw = int(input('enter amount you want to withdraw:'))
101         if withdraw > balance:
102             print('insufficient balance')
103         else:
104             balance-=withdraw
105             history=('amount withdraw:',str(withdraw))
106             output.append(history)
107 print(output)
```

PS D:\Hexaware\Assignment\Assignment 3\Python & "C:\Program Files\Python32\python.exe" "D:\Hexaware\Assignment\assignment 3\python\control_structure1-61.py"

What you want to do:deposit
Enter amount you want to deposit:1000
What you want to do:withdraw
enter amount you want to withdraw:500
What you want to do:exit
[('amount deposited:', '1000'), ('amount withdraw:', '500')]

PS D:\Hexaware\Assignment\Assignment 3\Python

Task7:

```
main.py
oops > main.py > Account > calculate_interest
1 class Customer:
2     #Customer class representing a bank customer.
3     def __init__(self, customer_id, first_name, last_name, email, phone_number, address):
4         #constructor for Customer class.
5         self.customer_id = customer_id
6         self.first_name = first_name
7         self.last_name = last_name
8         self.email = email
9         self.phone_number = phone_number
10        self.address = address
11
12
13 class Account:
14     #Account class representing a bank account.
15     def __init__(self, account_number, account_type, account_balance=0.0):
16         self.account_number = account_number
17         self.account_type = account_type
18         self.account_balance = account_balance
19
20     def deposit(self, amount):
21         self.account_balance += amount
22         print(f'Deposited {amount} into account {self.account_number} . New Balance: {self.account_balance}')
23
24     def withdraw(self, amount):
25         if amount <= self.account_balance:
26             self.account_balance -= amount
27             print(f'Withdraw {amount} from account {self.account_number}. New Balance: {self.account_balance}')
28         else:
29             print('Insufficient balance. Withdrawal failed.')
30
31     def calculate_interest(self):
32         if self.account_type == 'savings':
33             interest = 0.05 * self.account_balance
34             self.account_balance += interest
35             print(f'Interest calculated and added, New Balance: {self.account_balance}')
```


Task(8,9):

```

main@task8.py X
oops> main@task8.py> CurrentAccount> withdraw
1 from main import Account
2 #task 8-2 subclass inherited from account class
3 class SavingsAccount(Account):
4
5     #Subclass representing a Savings Account.
6
7
8     def __init__(self, account_number, interest_rate, account_balance=0.0):
9         super().__init__(account_number, "Savings", account_balance)
10        self.interest_rate = interest_rate
11
12
13     def calculate_interest(self):
14         #calculate and add interest to the savings account balance.
15
16         interest = self.interest_rate * self.account_balance
17         self.account_balance += interest
18         print(f"Interest calculated and added. New balance: {self.account_balance}")
19
20
21 class CurrentAccount(Account):
22
23     #Subclass representing a Current Account.
24
25
26     OVERDRAFT_LIMIT = 1000.0 # Example overdraft limit
27
28
29     def __init__(self, account_number, overdraft_limit, account_balance=0.0):
30         super().__init__(account_number, "Current", account_balance)
31         self.overdraft_limit = overdraft_limit
32
33
34     def withdraw(self, amount):
35
36         #Withdraw the specified amount from the current account with overdraft limit.
37         available_balance = self.account_balance + self.overdraft_limit
38         if amount <= available_balance:
39             self.account_balance -= amount
40             print(f"Withdraw {amount} from account {self.account_number}. New balance: {self.account_balance}")
41         else:
42             print(f"Insufficient Funds. Withdrawal failed.")

```

Inputs for above methods:

```

39
40 # Example usage:
41
42 # Create a savings account
43 savings_account = SavingsAccount(account_number=2001, interest_rate=0.05)
44 savings_account.deposit(1000.0)
45 savings_account.calculate_interest()
46
47 # Create a current account
48 current_account = CurrentAccount(account_number=3001, overdraft_limit=500.0)
49 current_account.deposit(200.0)
50 current_account.withdraw(300.0)
51

```

Output:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\Hexaware\Assignment\Assignment 3\Python> & "C:/Program Files/Python312/python.exe" "d:/Hexaware/Assignment/assignment 3/Python/tops/main1(tn8k8).py"
Deposited 1000.0 into account 1001 . New balance: 1000.0
Withdraw 500.0 from account 1001. New balance: 500.0
Interest calculated and added. New balance: 522.5
Deposited 1000.0 into account 1001 . New balance: 1000.0
Withdraw 500.0 from account 1001. New balance: 500.0
Deposited 200 into account 1001 . New balance: 700.0
Withdraw 100 from account 1001. New balance: 600.0
Deposited 300.5 into account 1001 . New balance: 900.5
Withdraw 150.25 from account 1001. New balance: 750.25
Deposited 1000.0 into account 1001 . New balance: 1000.0
Interest calculated and added. New balance: 1050.0
Deposited 200.0 into account 1001 . New balance: 1200.0
Withdraw 300.0 from account 1001. New balance: 900.0
PS D:\Hexaware\Assignment\Assignment 3\Python>

```


Implementation of bank class & full operations:

```
main@bank.py •
oops > main@bank.py > Bank > perform_operations

52 class Bank:
53     """
54     Bank class representing the banking system.
55     """
56     def main(self):
57         while True:
58             print("\nBanking System Menu:")
59             print("1. Create Savings Account")
60             print("2. Create Current Account")
61             print("3. Exit")
62
63             choice = input("Enter your choice (1-3): ")
64
65             if choice == '1':
66                 # Create Savings Account
67                 account_number = int(input("Enter account number: "))
68                 interest_rate = float(input("Enter interest rate for the savings account: "))
69                 savings_account = SavingsAccount(account_number, interest_rate)
70                 self.perform_operations(savings_account)
71
72             elif choice == '2':
73                 # Create Current Account
74                 account_number = int(input("Enter account number: "))
75                 overdraft_limit = float(input("Enter overdraft limit for the current account: "))
76                 current_account = CurrentAccount(account_number, overdraft_limit)
77                 self.perform_operations(current_account)
78
79             elif choice == '3':
80                 print("Exiting the Banking System.")
81                 break
82
83             else:
84                 print("Invalid choice. Please enter a valid option.")
85
86     def perform_operations(self, account):
87
```

```
main@bank.py •
oops > main@bank.py > Bank > perform_operations

86     def perform_operations(self, account):
87
88         while True:
89             print("\nAccount Operations Menu:")
90             print("1. Deposit")
91             print("2. Withdraw")
92             print("3. Calculate Interest (for Savings Account)")
93             print("4. Exit")
94
95             operation_choice = input("Enter your choice (1-4): ")
96
97             if operation_choice == '1':
98                 amount = float(input("Enter the deposit amount: "))
99                 account.deposit(amount)
100
101             elif operation_choice == '2':
102                 amount = float(input("Enter the withdrawal amount: "))
103                 account.withdraw(amount)
104
105             elif operation_choice == '3' and isinstance(account, SavingsAccount):
106                 account.calculate_interest()
107
108             elif operation_choice == '4':
109                 print("Exiting Account Operations.")
110                 break
111
112             else:
113                 print("Invalid choice. Please enter a valid option.")
114
115
116
117 bank = Bank()
118 bank.main()
119
```

Outputs:

```

Python 3.12.0
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\Vesware\Assignment\assignment_3\Python> & "C:\Program Files\Python312\python.exe" "D:\Vesware\Assignment\assignment_3\Python\ops\main1(task0).py"
Deposited 1000.0 into account 1001 . New balance: 1000.0
Withdraw 500.0 from account 1001. New balance: 500.0
Interest calculated and added. New balance: 522.5
Deposited 1000.0 into account 1001 . New balance: 1000.0
Withdraw 500.0 from account 1001. New balance: 500.0
Deposited 200.0 into account 1001 . New balance: 700.0
Withdraw 100 from account 1001. New balance: 600.0
Deposited 300.5 into account 1001 . New balance: 900.5
Withdraw 150.25 from account 1001. New balance: 750.25

Banking System Menu:
1. Create Savings Account
2. Create Current Account
3. Exit
Enter your choice (1-3): 1
Enter account number: 2002
Enter interest rate for the savings account: 7.5

Account Operations Menu:
1. Deposit
2. Withdraw
3. Calculate Interest (for Savings Account)
4. Exit
Enter your choice (1-4): 2
Enter the withdrawal amount: 100
Insufficient balance. withdrawal failed.

Account Operations Menu:
1. Deposit
2. Withdraw
3. Calculate Interest (for Savings Account)
4. Exit
Enter your choice (1-4): 4
Exiting Account Operations.

```

```
Banking System Menu:
1. Create Savings Account
2. Create Current Account
3. Exit
Enter your choice (1-3): 2
Enter account number: 3002
Enter overdraft limit for the current account: 100

Account Operations Menu:
1. Deposit
2. Withdraw
3. Calculate Interest (for Savings Account)
4. Exit
Enter your choice (1-4): 1
Enter the deposit amount: 1000
Deposited 1000.0 into account 3002 . New balance: 1000.0

Account Operations Menu:
1. Deposit
2. Withdraw
3. Calculate Interest (for Savings Account)
4. Exit
Enter your choice (1-4): 4
Exiting Account Operations.

Banking System Menu:
1. Create Savings Account
2. Create Current Account
3. Exit
Enter your choice (1-3): 3
Exiting the Banking System.
PS D:\Vesware\Assignment\Assignment 3\Python>
```


Task(10,11,12,13):

```
main2(task10-13).py X
ooga > main2(task10-13).py > Customer
1 import re
2 class InsufficientFundsException(Exception):
3     pass
4
5
6 class InvalidAccountException(Exception):
7     pass
8
9
10 class OverdraftLimitExceeded(Exception):
11     pass
12
13 class Customer:
14
15     #Customer class representing a bank customer.
16
17     def __init__(self, customer_id, first_name, last_name, email, phone_number, address):
18         #constructor for customer class.
19
20         self.customer_id = customer_id
21         self.first_name = first_name
22         self.last_name = last_name
23         self.set_email(email)
24         self.set_phone_number(phone_number)
25         self.address = address
26
27     def set_email(self, email):
28         # Simple email validation using regex
29         email_pattern = re.compile(r"[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-zA-Z]{2,4}")
30         if email_pattern.match(email):
31             self.email = email
32         else:
33             raise ValueError("Invalid email address.")
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
main2(task10-13).py X
ooga > main2(task10-13).py > Bank
35     def set_phone_number(self, phone_number):
36
37         # Simple phone number validation for 10 digits
38         if re.match(r"^\d{10}$", phone_number):
39             self.phone_number = phone_number
40         else:
41             raise ValueError("Invalid phone number.")
42
43
44 class Account:
45
46     #Account class representing a bank account.
47
48     account_number_generator = 1000 # Starting account number
49
50     def __init__(self, customer, account_type, balance=0.0):
51
52         self.account_number = Account.account_number_generator + 1
53         Account.account_number_generator += 1
54         self.customer = customer
55         self.account_type = account_type
56         self.balance = balance
57
58
59 class Bank:
60
61     #Bank class representing the banking system.
62
63     accounts = [] # List to store created accounts
64
65     def create_account(self, customer, acc_type, balance):
66
67         account = Account(customer, acc_type, balance)
68         Bank.accounts.append(account)
69         return account
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
main2task10-13.py X
ops > main2task10-13.py > Bank
71 def get_account_balance(self, account_number):
72
73     for account in Bank.accounts:
74         if account.account_number == account_number:
75             return account.balance
76     raise InvalidAccountException("Account not found.")
77
78 def deposit(self, account_number, amount):
79
80     for account in Bank.accounts:
81         if account.account_number == account_number:
82             account.balance += amount
83             return account.balance
84     raise InvalidAccountException("Account not found.")
85
86 def withdraw(self, account_number, amount):
87
88     for account in Bank.accounts:
89         if account.account_number == account_number:
90             if account.balance >= amount:
91                 account.balance -= amount
92                 return account.balance
93             else:
94                 raise InsufficientFundsException("Insufficient funds.")
95     raise InvalidAccountException("Account not found.")
96
97 def transfer(self, from_account_number, to_account_number, amount):
98     #Transfer money from one account to another.
99
100     from_account = None
101     to_account = None
102
103     for account in Bank.accounts:
104         if account.account_number == from_account_number:
105             from_account = account
106         elif account.account_number == to_account_number:
107             to_account = account
```

```
main2task10-13.py X
ops > main2task10-13.py > Bank
107         to_account = account
108
109     if from_account and to_account:
110         if from_account.balance >= amount:
111             from_account.balance -= amount
112             to_account.balance += amount
113             return from_account.balance, to_account.balance
114         else:
115             raise InsufficientFundsException("Insufficient funds.")
116     else:
117         raise InvalidAccountException("One or both accounts not found.")
118
119 def get_account_details(self, account_number):
120
121     #Retrieve account and customer details for a given account number.
122     for account in Bank.accounts:
123         if account.account_number == account_number:
124             return account, account.customer
125     raise InvalidAccountException("Account not found.")
126
127
128 class BankApp:
129
130     #BankApp class to simulate the banking system.
131
132     def main(self):
133         bank = Bank()
134         while True:
135             try:
136                 print("\nBanking System Menu")
137                 print("1. Create Account")
138                 print("2. Deposit")
139                 print("3. Withdraw")
140                 print("4. Transfer")
141                 print("5. Get Account Balance")
142                 print("6. Get Account Details")
143                 print("7. Exit")
```

```
main2task10-13.py X
oops > main2task10-13.py > Bank
145 choice = input("Enter your choice (1-7): ")
146
147 if choice == '1':
148     customer_id = int(input("Enter Customer ID: "))
149     first_name = input("Enter First Name: ")
150     last_name = input("Enter Last Name: ")
151     email = input("Enter Email Address: ")
152     phone_number = input("Enter Phone Number: ")
153     address = input("Enter Address: ")
154
155     try:
156         customer = Customer(customer_id, first_name, last_name, email, phone_number, address)
157         acc_type = input("Choose Account Type (Savings/Current): ").capitalize()
158         balance = float(input("Enter Initial Balance: "))
159         bank.create_account(customer, acc_type, balance)
160         print("Account created successfully.")
161     except ValueError as e:
162         print(f"Error: {e}")
163
164 elif choice == '2':
165     account_number = int(input("Enter Account Number: "))
166     amount = float(input("Enter Deposit Amount: "))
167
168     try:
169         balance = bank.deposit(account_number, amount)
170         print(f"Deposit successful. Current balance: {balance}")
171     except ValueError as e:
172         print(f"Error: {e}")
173
174 elif choice == '3':
175     account_number = int(input("Enter Account Number: "))
176     amount = float(input("Enter Withdrawal Amount: "))
177
178     try:
179         balance = bank.withdraw(account_number, amount)
180         print(f"Withdrawal successful. Current balance: {balance}")
181     except ValueError as e:
182         print(f"Error: {e}")
```

```
main2task10-13.py X
oops > main2task10-13.py > Bank
183 elif choice == '4':
184     from_account_number = int(input("Enter Source Account Number: "))
185     to_account_number = int(input("Enter Destination Account Number: "))
186     amount = float(input("Enter Transfer Amount: "))
187
188     try:
189         from_balance, to_balance = bank.transfer(from_account_number, to_account_number, amount)
190         print(f"Transfer successful. Source balance: {from_balance}, Destination balance: {to_balance}")
191     except ValueError as e:
192         print(f"Error: {e}")
193
194 elif choice == '5':
195     account_number = int(input("Enter Account Number: "))
196
197     try:
198         balance = bank.get_account_balance(account_number)
199         print(f"Current balance: {balance}")
200     except ValueError as e:
201         print(f"Error: {e}")
202
203 elif choice == '6':
204     account_number = int(input("Enter Account Number: "))
205
206     try:
207         account, customer = bank.get_account_details(account_number)
208         print(f"Account Details:\n{account.__dict__}\nCustomer Details:\n{customer.__dict__}")
209     except ValueError as e:
210         print(f"Error: {e}")
211
212 elif choice == '7':
213     print("Exiting the Banking System.")
214     break
215
216 else:
217     print("Invalid choice. Please enter a valid option.")
218
219 except (InsufficientFundsException, InvalidAccountException, OverDraftLimitExceeded) as e:
220     print(f"Error: {e}")
221
222 except Exception as e:
```

```

210         break
211     else:
212         print("Invalid choice. Please enter a valid option.")
213
214 except (InsufficientFundsException, InvalidAccountException, OverdraftLimitExceeded) as e:
215     print(f"Error: {e}")
216
217 except Exception as e:
218     print(f"An unexpected error occurred: {e}")
219
220
221 bank_app = BankApp()
222 bank_app.main()

```

Outputs:

```

PS D:\Hackware\Assignment\assignment_3\Python> & "C:/Program Files/Python312/python.exe" "d:/Hackware/Assignment/assignment_3/Python/cops/main2(task3@_11).py"

Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get Account Balance
6. Get Account Details
7. Exit
Enter your choice (1-7): 1
Enter Customer ID: 1001
Enter First Name: Rohan
Enter Last Name: Chaudhari
Enter Email Address: rohan@g.com
Enter Phone Number: 1234567895
Enter Address: jaigaoon
Choose Account Type (Savings/Current): Savings
Enter Initial Balance: 10000
Account created successfully.

Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get Account Balance
6. Get Account Details
7. Exit
Enter your choice (1-7): 2
Enter Account Number: 1001
Enter Deposit Amount: 1000
Deposit successful. Current balance: 11000.0

```

```

Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get Account Balance
6. Get Account Details
7. Exit
Enter your choice (1-7): 3
Enter Account Number: 1001
Enter Withdrawal Amount: 1000
Withdrawal successful. Current balance: 10000.0

Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get Account Balance
6. Get Account Details
7. Exit
Enter your choice (1-7): 4
Enter Source Account Number: 1002
Enter Destination Account Number: 1001
Enter Transfer Amount: 100
Error: One or both accounts not found.

Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get Account Balance
6. Get Account Details
7. Exit
Enter your choice (1-7): 4
Enter Source Account Number: 1001
Enter Destination Account Number: 1002
Enter Transfer Amount: 10
Error: One or both accounts not found.

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Error: One or both accounts not found.

Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get Account Balance
6. Get Account Details
7. Exit
Enter your choice (1-7): 5
Enter Account Number: 1001
Current balance: 10000.0

Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get Account Balance
6. Get Account Details
7. Exit
Enter your choice (1-7): 6
Enter Account Number: 1001
Account Details:
{'account_number': 1001, 'customer': <__main__.Customer object at 0x0000016437A00110>, 'account_type': 'Savings', 'balance': 10000.0}
Customer Details:
{'customer_id': 1001, 'first_name': 'Rohan', 'last_name': 'Chaudhari', 'email': 'rohan@leg.com', 'phone_number': '1234567895', 'address': 'Jaipur'}

Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get Account Balance
6. Get Account Details
7. Exit
Enter your choice (1-7): 7
Exiting the Banking System.
PS D:\Hexaware\Assignment\Assignment_3\Python>

```

Task 14:

```

❖ database.py ×
oops > ❖ database.py > _
1 import mysql.connector
2 from mysql.connector import error
3 from datetime import datetime
4 class InvalidAccountException(Exception):
5     pass
6
7 class InsufficientFundException(Exception):
8     pass
9
10 class OverDraftLimitExceeded(Exception):
11     pass
12
13
14 class Customer:
15     def __init__(self, customer_id, first_name, last_name, email, phone_number, address):
16         self.customer_id = customer_id
17         self.first_name = first_name
18         self.last_name = last_name
19         self.email = email
20         self.phone_number = phone_number
21         self.address = address
22
23
24 class Account:
25     last_account_number = 0
26
27     def __init__(self, customer, account_type, balance=0.0):
28         Account.last_account_number += 1
29         self.account_number = Account.last_account_number
30         self.customer = customer
31         self.account_type = account_type
32         self.balance = balance
33
34

```



```

database.py X
oops > database.py ~
15 class Transaction:
16     def __init__(self, account, description, transaction_type, transaction_amount):
17         self.account = account
18         self.description = description
19         self.date_time = datetime.now()
20         self.transaction_type = transaction_type
21         self.transaction_amount = transaction_amount
22
23
24 class SavingsAccount(Account):
25     def __init__(self, customer, interest_rate):
26         super().__init__(customer, "Savings")
27         self.interest_rate = interest_rate
28
29
30 class CurrentAccount(Account):
31     def __init__(self, customer, overdraft_limit):
32         super().__init__(customer, "Current")
33         self.overdraft_limit = overdraft_limit
34
35
36 class ZeroBalanceAccount(Account):
37     def __init__(self, customer):
38         super().__init__(customer, "ZeroBalance")
39
40
41 class DBUtil:
42     @staticmethod
43     def get_db_conn():
44         try:
45             connection = mysql.connector.connect(
46                 host="localhost",
47                 user="root",
48                 password="root",
49                 database="hibern"
50             )
51             return connection
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70

```

```

database.py X
oops > database.py ~
70         return connection
71
72     except Exception as e:
73         print(f"Error: {e}")
74         return None
75
76 class BankRepositoryImpl():
77     def create_account(self, customer, acc_type, balance):
78         try:
79             conn = DBUtil.get_db_conn()
80             cursor = conn.cursor()
81
82             # Insert customer details
83             cursor.execute("INSERT INTO customers (customer_id, first_name, last_name, email, phone_number, address) "
84                             "VALUES (%s, %s, %s, %s, %s, %s)",
85                             (customer.customer_id, customer.first_name, customer.last_name, customer.email,
86                             customer.phone_number, customer.address))
87             conn.commit()
88
89             # Insert account details
90             cursor.execute("INSERT INTO accounts (account_number, account_type, balance, customer_id) "
91                             "VALUES (%s, %s, %s, %s)",
92                             (Account.last_account_number, acc_type, balance, customer.customer_id))
93             conn.commit()
94
95             return Account.last_account_number
96
97     except Error as e:
98         print(f"Error: {e}")
99         return None
100
101     finally:
102         if conn.is_connected():
103             cursor.close()
104             conn.close()
105

```



```
database.py X
oops > database.py > ...
106 def list_accounts(self):
107     try:
108         conn = DBUtil.get_db_conn()
109         cursor = conn.cursor()
110
111         # Retrieve all accounts
112         cursor.execute("SELECT * FROM accounts")
113         accounts = cursor.fetchall()
114
115         return [self.map_account(row) for row in accounts]
116
117     except Error as e:
118         print(f"Error: {e}")
119         return None
120
121     finally:
122         if conn.is_connected():
123             cursor.close()
124             conn.close()
125
126 def calculate_interest(self):
127     try:
128         conn = DBUtil.get_db_conn()
129         cursor = conn.cursor()
130
131         # Calculate interest for savings accounts
132         cursor.execute("UPDATE accounts SET balance = balance * (1 + interest_rate / 100) "
133                        "WHERE account_type = 'Savings'")
134         conn.commit()
135
136     except Error as e:
137         print(f"Error: {e}")
138
139     finally:
140         if conn.is_connected():
141             cursor.close()
142             conn.close()
```

```
database.py X
oops > database.py > ...
143 def get_account_balance(self, account_number):
144     try:
145         conn = DBUtil.get_db_conn()
146         cursor = conn.cursor()
147
148         # Retrieve account balance
149         cursor.execute("SELECT balance FROM accounts WHERE account_number = %s", (account_number,))
150         balance = cursor.fetchone()
151
152         if balance:
153             return balance[0]
154         else:
155             return "Account number is not present"
156
157     except Error as e:
158         print(f"Error: {e}")
159         return None
160
161     finally:
162         if conn.is_connected():
163             cursor.close()
164             conn.close()
165
166 def deposit(self, account_number, amount):
167     try:
168         conn = DBUtil.get_db_conn()
169         cursor = conn.cursor()
170
171         # Update account balance for deposit
172         cursor.execute("UPDATE accounts SET balance = balance + %s WHERE account_number = %s", (amount, account_number))
173         conn.commit()
174
175         # Record the transaction
176         self.record_transaction(account_number, "Deposit", "Deposit", amount)
177
178         return self.get_account_balance(account_number)
179
180
```

```
database.py X
oops > database.py > ...
181         except Error as e:
182             print(f"Error: {e}")
183             return None
184
185         finally:
186             if conn.is_connected():
187                 cursor.close()
188                 conn.close()
189
190     def withdraw(self, account_number, amount):
191         try:
192             conn = DBUtil.get_db_conn()
193             cursor = conn.cursor()
194
195             # Retrieve current account balance
196             cursor.execute("SELECT balance, account_type FROM accounts WHERE account_number = %s", (account_number,))
197             result = cursor.fetchone()
198
199             if result:
200                 balance, account_type = result
201                 if account_type == "Savings" and (balance - amount) < 500:
202                     raise InsufficientFundException("Withdrawal violates minimum balance rule.")
203
204                 if account_type == "Current" and (balance - amount) < 0:
205                     raise OverdraftLimitExceeded("Withdrawal exceeds available balance and overdraft limit.")
206
207                 # Update account balance for withdrawal
208                 cursor.execute("UPDATE accounts SET balance = balance - %s WHERE account_number = %s",
209                               (amount, account_number))
210                 conn.commit()
211
212                 # Record the transaction
213                 self.record_transaction(account_number, "Withdrawal", "Withdrawal", amount)
214
215                 return self.get_account_balance(account_number)
216             else:
217                 raise InvalidAccountException("Account not found.")
```

```
database.py X
oops > database.py > ...
218         raise InvalidAccountException("Account not found.")
219
220     except (InsufficientFundException, OverdraftLimitExceeded) as e:
221         raise e
222
223     except Error as e:
224         print(f"Error: {e}")
225         return None
226
227     finally:
228         if conn.is_connected():
229             cursor.close()
230             conn.close()
231
232     def transfer(self, from_account_number, to_account_number, amount):
233         try:
234             conn = DBUtil.get_db_conn()
235             cursor = conn.cursor()
236
237             # Check if both accounts exist
238             if not self.account_exists(from_account_number) or not self.account_exists(to_account_number):
239                 raise InvalidAccountException("One or both accounts not found.")
240
241             # Check if the source account has sufficient funds
242             from_balance = self.get_account_balance(from_account_number)
243             if from_balance < amount:
244                 raise InsufficientFundException("Insufficient funds for the transfer.")
245
246             # Update source account balance for withdrawal
247             cursor.execute("UPDATE accounts SET balance = balance - %s WHERE account_number = %s",
248                             (amount, from_account_number))
249             conn.commit()
250
251             # Update destination account balance for deposit
252             cursor.execute("UPDATE accounts SET balance = balance + %s WHERE account_number = %s",
253                             (amount, to_account_number))
254             conn.commit()
```

```

database.py X
oogn > database.py > ..
253         (amount, to_account_number))
254     conn.commit()
255
256     # Record the transactions
257     self.record_transaction(from_account_number, "transfer", "transfer to " + str(to_account_number), amount)
258     self.record_transaction(to_account_number, "transfer", "transfer from " + str(from_account_number), amount)
259
260     return self.get_account_balance(from_account_number), self.get_account_balance(to_account_number)
261
262 except (InsufficientFundException, InvalidAccountException) as e:
263     raise e
264
265 except Error as e:
266     print("Error: (e)")
267     return None
268
269 finally:
270     if conn.is_connected():
271         cursor.close()
272         conn.close()
273
274 def get_account_details(self, account_number):
275     try:
276         conn = DBUtil.get_db_conn()
277         cursor = conn.cursor()
278
279         # Retrieve account details
280         cursor.execute("SELECT * FROM accounts INNER JOIN customers "
281                        "ON accounts.customer_id = customers.customer_id "
282                        "WHERE account_number = %s", (account_number,))
283         result = cursor.fetchone()
284
285         if result:
286             account_details = {
287                 "account number": result[0],
288                 "account type": result[1],
289                 "balance": result[2],

```

```

database.py X
oogn > database.py > ..
290             "account type": result[1],
291             "balance": result[2],
292             "customer_id": result[3],
293             "first_name": result[5],
294             "last_name": result[6],
295             "email": result[7],
296             "phone_number": result[8],
297             "address": result[9]
298         }
299         return account_details
300
301     else:
302         raise InvalidAccountException("Account not found.")
303
304 except InvalidAccountException as e:
305     raise e
306
307 except Error as e:
308     print("Error: (e)")
309     return None
310
311 finally:
312     if conn.is_connected():
313         cursor.close()
314         conn.close()
315
316 def get_transactions(self, account_number, from_date, to_date):
317     try:
318         conn = DBUtil.get_db_conn()
319         cursor = conn.cursor()
320
321         # Retrieve transactions within the date range
322         cursor.execute("SELECT * FROM transactions WHERE account_number = %s "
323                        "AND date_time BETWEEN %s AND %s",
324                        (account_number, from_date, to_date))
325         transactions = cursor.fetchall()
326

```

```
database.py X
oops > database.py > ...
324         return [self.map_transaction(row) for row in transactions]
325
326     except Error as e:
327         print(f"Error: {e}")
328         return None
329
330     finally:
331         if conn.is_connected():
332             cursor.close()
333             conn.close()
334
335     def record_transaction(self, account_number, description, transaction_type, transaction_amount):
336     try:
337         conn = DBUtil.get_db_conn()
338         cursor = conn.cursor()
339
340         # Record the transaction
341         cursor.execute("INSERT INTO transactions (account_number, description, date_time, transaction_type, transaction_amount) "
342                        "VALUES (%s, %s, %s, %s, %s)",
343                        (account_number, description, datetime.now(), transaction_type, transaction_amount))
344         conn.commit()
345
346     except Error as e:
347         print(f"Error: {e}")
348
349     finally:
350         if conn.is_connected():
351             cursor.close()
352             conn.close()
353
354     @staticmethod
355     def map_account(row):
356         return {
357             "account_number": row[0],
358             "account_type": row[1],
359             "balance": row[2],
360
361 database.py X
oops > database.py > ...
356     def map_account(row):
357         return {
358             "account_number": row[0],
359             "account_type": row[1],
360             "balance": row[2],
361             "customer_id": row[3]
362         }
363
364     @staticmethod
365     def map_transaction(row):
366         return {
367             "account_number": row[0],
368             "description": row[1],
369             "date_time": row[2],
370             "transaction_type": row[3],
371             "transaction_amount": row[4]
372         }
373
374     def account_exists(self, account_number):
375     try:
376         conn = DBUtil.get_db_conn()
377         cursor = conn.cursor()
378
379         # Check if the account exists
380         cursor.execute("SELECT * FROM accounts WHERE account_number = %s", (account_number,))
381         return cursor.fetchone() is not None
382
383     except Error as e:
384         print(f"Error: {e}")
385         return False
386
387     finally:
388         if conn.is_connected():
389             cursor.close()
390             conn.close()
391
```

```
database.py X
oops > database.py > ...
356     def map_account(row):
357         return {
358             "account_number": row[0],
359             "account_type": row[1],
360             "balance": row[2],
361             "customer_id": row[3]
362         }
363
364     @staticmethod
365     def map_transaction(row):
366         return {
367             "account_number": row[0],
368             "description": row[1],
369             "date_time": row[2],
370             "transaction_type": row[3],
371             "transaction_amount": row[4]
372         }
373
374     def account_exists(self, account_number):
375     try:
376         conn = DBUtil.get_db_conn()
377         cursor = conn.cursor()
378
379         # Check if the account exists
380         cursor.execute("SELECT * FROM accounts WHERE account_number = %s", (account_number,))
381         return cursor.fetchone() is not None
382
383     except Error as e:
384         print(f"Error: {e}")
385         return False
386
387     finally:
388         if conn.is_connected():
389             cursor.close()
390             conn.close()
391
```

Inputs :

```

193
194
195 # Example usage:
196
197 # Create an instance of BankRepositoryImpl and test the methods
198 bank_repository = BankRepositoryImpl()
199
200 # Create a customer
201 customer = Customer(1, "John", "Doe", "john.doe@example.com", "1234567890", "123 Main St")
202 customer1 = Customer(2, "John1", "Doe1", "john1.doe@example.com", "1234567890", "12345 Main St")
203
204 # Create a SavingsAccount
205 savings_account_number = bank_repository.create_account(customer, "Savings", 1000.0)
206 print("Savings Account Number:", savings_account_number)
207
208 # Create a CurrentAccount
209 current_account_number = bank_repository.create_account(customer1, "Current", 2000.0)
210 print("Current Account Number:", current_account_number)
211
212 # Perform transactions
213 bank_repository.deposit(1, 500)
214 bank_repository.withdraw(1, 100)
215 bank_repository.transfer(savings_account_number, current_account_number, 300.0)
216
217 # Get account details
218 savings_account_details = bank_repository.get_account_details(1)
219 current_account_details = bank_repository.get_account_details(1)
220
221 print("Savings Account Details:", savings_account_details)
222 print("Current Account Details:", current_account_details)
223
224 # Get account balance
225 savings_balance = bank_repository.get_account_balance(1)
226 current_balance = bank_repository.get_account_balance(1)
227
228 print("Savings Account Balance:", savings_balance)
229 print("Current Account Balance:", current_balance)
230
231
232 # List all accounts
233 accounts_list = bank_repository.list_accounts()
234 print("All Accounts:", accounts_list)
235
236 # Calculate interest for savings accounts
237 bank_repository.calculate_interest()
238
239 # Get transactions
240 transactions = bank_repository.get_transactions(1, datetime(2023, 1, 1), datetime.now())
241 print("Savings Account Transactions:", transactions)
242

```

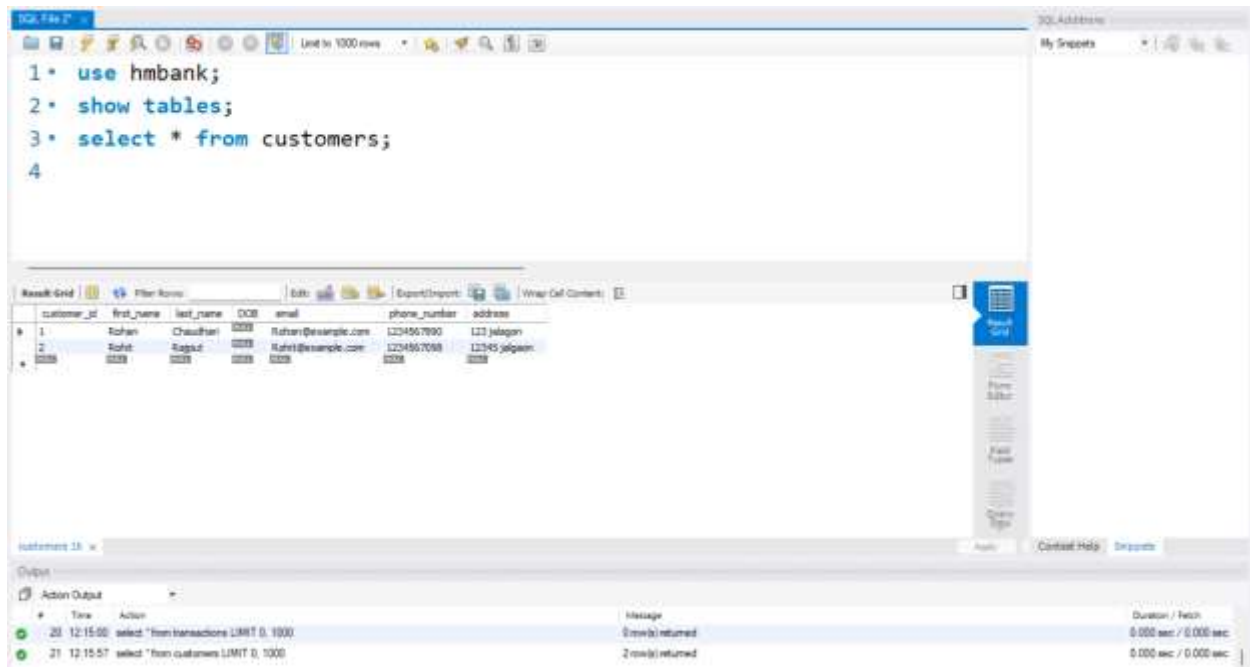
outputs:

```

PS D:\Hexaware\Assignment\assignment 3\python & "C:/Program Files/Python312/python.com" "D:\Hexaware\Assignment\assignment 3\python\ops/database.py"
Savings Account Number: 0
Current Account Number: 0
Error: 1054 (42S22): unknown column 'account_number' in 'Field list'
Error: 1054 (42S22): unknown column 'account_number' in 'Field list'
Savings Account Details: {'account_number': 1, 'account_type': 1, 'balance': 'Savings', 'customer_id': Decimal('1000.00'), 'first_name': 'Rohan', 'last_name': 'Chaudhari', 'email': None, 'phone_number': 'Ruharg@example.com', 'address': '1234567890'}
Current Account Details: {'account_number': 1, 'account_type': 1, 'balance': 'Savings', 'customer_id': Decimal('1000.00'), 'first_name': 'Rohan', 'last_name': 'Chaudhari', 'email': None, 'phone_number': 'Ruharg@example.com', 'address': '1234567890'}
Savings Account Balance: 1000.00
All Accounts: [{'account_number': 1, 'account_type': 1, 'balance': 'Savings', 'customer_id': Decimal('1000.00')}, {'account_number': 2, 'account_type': 2, 'balance': 'Current', 'customer_id': Decimal('2000.00')}]

```


Database Changes:



The screenshot shows the SQL File Editor interface. The SQL script in the editor is:

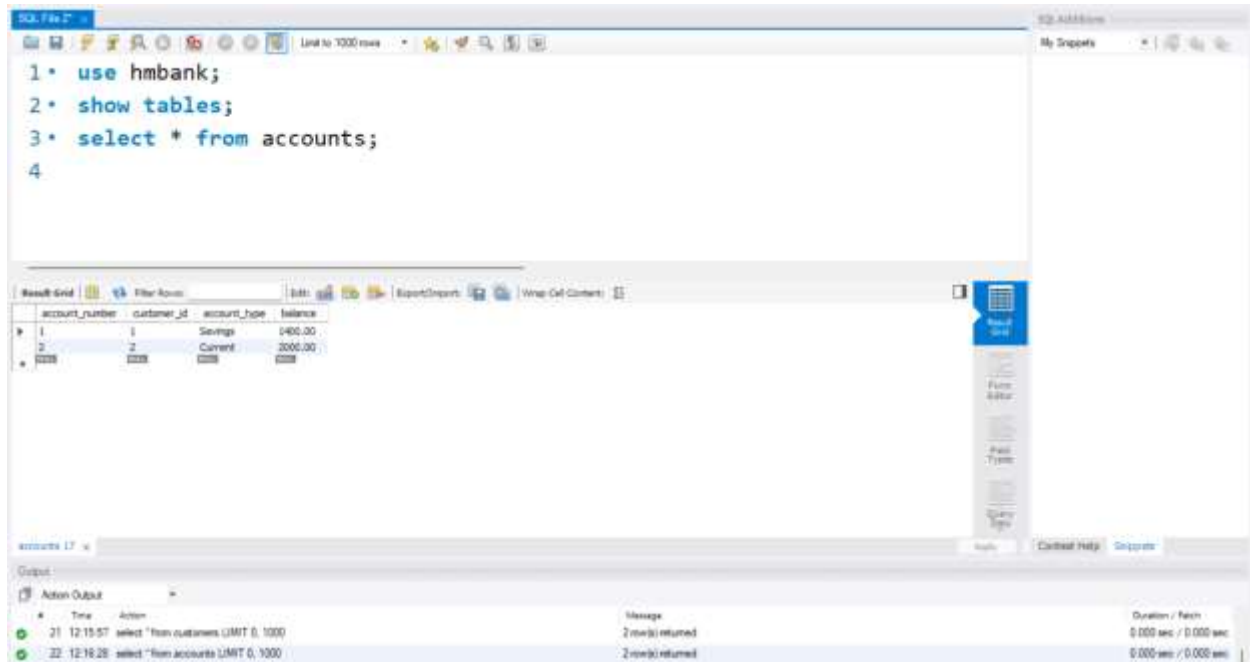
```
1 • use hmbank;  
2 • show tables;  
3 • select * from customers;  
4
```

The results grid displays the output of the SQL script:

customer_id	first_name	last_name	DOB	email	phone_number	address
1	Rohan	Chaudhri	2008	Rohan@example.com	1234567890	123 Jyegon
2	Rohit	Raghu	2008	Rohit@example.com	1234567890	12345 Jyegon

The output pane shows the execution details:

Time	Action	Message	Duration / Fetch
20 12:15:00	select * from transactions LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
21 12:15:57	select * from customers LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec



The screenshot shows the SQL File Editor interface. The SQL script in the editor is:

```
1 • use hmbank;  
2 • show tables;  
3 • select * from accounts;  
4
```

The results grid displays the output of the SQL script:

account_number	customer_id	account_type	balance
1	1	Savings	1400.00
2	2	Current	2000.00

The output pane shows the execution details:

Time	Action	Message	Duration / Fetch
21 12:15:57	select * from customers LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
22 12:16:26	select * from accounts LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec