Assignment 3
Banking System

Task1(With outputs):

```python
#task 1
def eligibility(credit_score,annual_income):
    if credit_score > 700 and annual_income > 50000:
        return "You are eligible for loan"
    else:
        return "You are not eligible for loan"

credit_score=int(input('Enter your credit score:'))
annual_income=int(input('Enter your annual income:'))
print(eligibility(credit_score,annual_income))
```

```
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 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_structure(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:

```python
#task2
def atm(current_balance,transaction):
    if transaction == 1:
        print(current_balance)

    elif transaction == 2:
        amount=int(input('Amount you want to withdraw:'))
        if amount > current_balance:
            return 'Insufficient balance'
        elif amount%100 != 0:
            return 'amount should be in 100s only'

        else:
            current_balance -= amount
            return 'transaction success \n current balance:%d'%(current_balance)
    elif transaction == 3:
        amount=int(input('Amount you want to deposit:'))
        current_balance +=amount
        return 'transaction success \n current balance:%d'%(current_balance)

current_balance = int(input("enter your current balance:"))
transaction=int(input('what you want to do \n 1 : check balance \n 2 : withdraw \n 3 : Deposit \n enter :' ))
print(atm(current_balance,transaction))
```

Output:

```
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 your current balance:1000
What you want to do
 1 : Check balance
 2 : Withdraw
 3 : Deposit
 enter :1
1000
None
PS D:\Hexaware\Assignment\assignment 3\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 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:\Hexaware\Assignment\assignment 3\Python> & "C:/Program Files/Python312/python.exe" "d:/Hexaware/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:\Hexaware\Assignment\assignment 3\Python>
```

Task3(With output):

```
Control_structure(1-6).py ×
Control_structure(1-6).py > ...
# Task 3
def future(balance,intrest,years):
    future_balance=balance*(1+intrest/100)**years
    print(future_balance)

while True:
    balance=int(input('Enter your initial balance:'))
    intrest=float(input('Enter your intrset rate:'))
    years=int(input('Enter your number of years:'))
    print(future(balance,intrest,years))
    ask=input('do you want to check for next customer:')
    if ask in ['NO','No','no','no']:
        print('Process ended')
        break

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 your initial balance:1000
Enter your intrset rate:7.5
Enter your number of years:5
1435.629326171874 7
None
do you want to check for next customer:No
Process ended
PS D:\Hexaware\Assignment\assignment 3\Python>
```

Task4:

```python
#task-4
bank={1:100,2:200,3:300,4:400}

def bank_details(account):
    for i in bank.keys():
        if account == i:
            return bank.get(i)
    else:
        return 'Enter Correct account number'

account=int(input('Enter Bank account number:'))
print(bank_details(account))
```

```
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 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_structure(1-6).py"
Enter Bank account number:3
300
PS D:\Hexaware\Assignment\assignment 3\Python>
```

Task5:

```python
#task5
def password(password):
    if len(password) > 7 and password.islower() == False and any(map(str.isdigit,password)) == True:
        print('Valid password')
    else:
        print('not a valid password')

inp_pass=str(input('Enter password:'))
print(password(inp_pass))
```

```
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:Rohan@1234567% "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:rohan1234567% "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:

```python
#task6
balance=100
output=[]
while True:
    transaction=str(input('what you want to do:'))
    if transaction == 'exit':
        break
    elif transaction == 'deposit':
        deposit=int(input('Enter amount you want to deposit:'))
        balance+=deposit
        history=('amount deposited:',str(deposit))
        output.append(history)
    elif transaction == 'withdraw':
        withdraw = int(input('enter amount you want to withdraw:'))
        if withdraw > balance:
            print('insufficient balance')
        else:
            balance-=withdraw
            history=('amount withdraw:',str(withdraw))
            output.append(history)
print(output)
```

```
PS D:\Hexaware\Assignment\assignment 3\Python> & "C:/Program Files/Python312/python.exe" "D:/Hexaware/Assignment/assignment 3/Python/control_structure(1-6).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:

```python
class Customer:
    #Customer class representing a bank customer.
    def __init__(self, customer_id, first_name, last_name, email, phone_number, address):
        #Constructor for customer class.
        self.customer_id = customer_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.phone_number = phone_number
        self.address = address


class Account:
    #Account class representing a bank account.
    def __init__(self, account_number, account_type, account_balance=0.0):
        self.account_number = account_number
        self.account_type = account_type
        self.account_balance = account_balance

    def deposit(self, amount):
        self.account_balance += amount
        print(f"Deposited {amount} into account {self.account_number} . New balance: {self.account_balance}")

    def withdraw(self, amount):
        if amount <= self.account_balance:
            self.account_balance -= amount
            print(f"Withdrew {amount} from account {self.account_number}. New balance: {self.account_balance}")
        else:
            print("Insufficient balance. Withdrawal failed.")

    def calculate_interest(self):
        if self.account_type == "Savings":
            interest = 0.045 * self.account_balance
            self.account_balance += interest
            print(f"Interest calculated and added. New balance: {self.account_balance}")
```

```python
class Bank:
    def main(self):
        # Create a customer
        customer = Customer(customer_id=1, first_name="John", last_name="Doe",
                            email="john.doe@example.com", phone_number="1234567890", address="123 Main Street")


        # Create an account for the customer
        account = Account(account_number=1001, account_type="Savings")

        # Deposit and withdraw operations
        account.deposit(1000.0)
        account.withdraw(500.0)

        # Calculate and add interest for savings account
        account.calculate_interest()
```

Input Methods:

```python
# Create an instance of the Bank class and execute the main method
bank = Bank()
bank.main()


#example for task 8-1
# Example usage:

# Create an account for the customer
account = Account(account_number=1001, account_type="Savings")

# Deposit and withdraw operations with different data types
account.deposit(1000.0)   # float
account.withdraw(500.0)    # float

account.deposit(200)       # int
account.withdraw(100)      # int

account.deposit(300.5)     # double
account.withdraw(150.25)   # double
```

Output:

```
PS D:\Hexaware\Assignment\assignment 3\Python> & "C:/Program Files/Python312/python.exe" "d:/Hexaware/Assignment/assignment 3/Python/oops/main.py"
Deposited 1000.0 into account 1001 . New balance: 1000.0
Withdrew 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
Withdrew 500.0 from account 1001. New balance: 500.0
Deposited 200 into account 1001 . New balance: 700.0
Withdrew 100 from account 1001. New balance: 600.0
Deposited 300.5 into account 1001 . New balance: 900.5
Withdrew 150.25 from account 1001. New balance: 750.25
PS D:\Hexaware\Assignment\assignment 3\Python>
```

Task(8,9):

```python
from main import Account
#task 8-2 subclass inherited from accountclass
class SavingsAccount(Account):

    #subclass representing a Savings Account.

    def __init__(self, account_number, interest_rate, account_balance=0.0):
        super().__init__(account_number, "Savings", account_balance)
        self.interest_rate = interest_rate

    def calculate_interest(self):
        #calculate and add interest to the savings account balance.

        interest = self.interest_rate * self.account_balance
        self.account_balance += interest
        print(f"Interest calculated and added. New balance: {self.account_balance}")


class CurrentAccount(Account):

    #subclass representing a Current Account.

    OVERDRAFT_LIMIT = 1000.0  # Example overdraft limit

    def __init__(self, account_number, overdraft_limit, account_balance=0.0):
        super().__init__(account_number, "Current", account_balance)
        self.overdraft_limit = overdraft_limit

    def withdraw(self, amount):

        #withdraw the specified amount from the current account with overdraft limit.
        available_balance = self.account_balance + self.overdraft_limit
        if amount <= available_balance:
            self.account_balance -= amount
            print(f"Withdrew {amount} from account {self.account_number}. New balance: {self.account_balance}")
        else
            print("Insufficient funds. Withdrawal failed.")
```

Inputs for above methods:

```python
# Example usage:

# Create a savings account
savings_account = SavingsAccount(account_number=2001, interest_rate=0.05)
savings_account.deposit(1000.0)
savings_account.calculate_interest()

# Create a current account
current_account = CurrentAccount(account_number=3001, overdraft_limit=500.0)
current_account.deposit(200.0)
current_account.withdraw(300.0)
```

Output:

```
PS D:\Hexaware\Assignment\assignment 3\Python> & "C:/Program Files/Python312/python.exe" "d:/Hexaware/Assignment/assignment 3/Python/oops/main1(task8).py"
Deposited 1000.0 into account 1001 . New balance: 1000.0
Withdrew 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
Withdrew 500.0 from account 1001. New balance: 500.0
Deposited 200 into account 1001 . New balance: 700.0
Withdrew 100 from account 1001. New balance: 600.0
Deposited 300.5 into account 1001 . New balance: 900.5
Withdrew 150.25 from account 1001. New balance: 750.25
Deposited 1000.0 into account 2001 . New balance: 1000.0
Interest calculated and added. New balance: 1050.0
Deposited 200.0 into account 3001 . New balance: 200.0
Withdrew 300.0 from account 3001. New balance: -100.0
PS D:\Hexaware\Assignment\assignment 3\Python>
```

Implementation of bank class & full operations:

```python
class Bank:
    """
    Bank class representing the banking system.
    """
    def main(self):
        while True:
            print("\nBanking System Menu:")
            print("1. Create Savings Account")
            print("2. Create Current Account")
            print("3. Exit")

            choice = input("Enter your choice (1-3): ")

            if choice == '1':
                # Create Savings Account
                account_number = int(input("Enter account number: "))
                interest_rate = float(input("Enter interest rate for the savings account: "))
                savings_account = SavingsAccount(account_number, interest_rate)
                self.perform_operations(savings_account)

            elif choice == '2':
                # Create Current Account
                account_number = int(input("Enter account number: "))
                overdraft_limit = float(input("Enter overdraft limit for the current account: "))
                current_account = CurrentAccount(account_number, overdraft_limit)
                self.perform_operations(current_account)

            elif choice == '3':
                print("Exiting the Banking System.")
                break

            else:
                print("Invalid choice. Please enter a valid option.")

    def perform_operations(self, account):

```

```python
    def perform_operations(self, account):

        while True:
            print("\nAccount Operations Menu:")
            print("1. Deposit")
            print("2. Withdraw")
            print("3. Calculate Interest (for Savings Account)")
            print("4. Exit")

            operation_choice = input("Enter your choice (1-4): ")

            if operation_choice == '1':
                amount = float(input("Enter the deposit amount: "))
                account.deposit(amount)

            elif operation_choice == '2':
                amount = float(input("Enter the withdrawal amount: "))
                account.withdraw(amount)

            elif operation_choice == '3' and isinstance(account, SavingsAccount):
                account.calculate_interest()

            elif operation_choice == '4':
                print("Exiting Account Operations.")
                break

            else:
                print("Invalid choice. Please enter a valid option.")



bank = Bank()
bank.main()
```

Outputs:

```
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/oops/main1(task8).py"
Deposited 1000.0 into account 1001 . New balance: 1000.0
Withdrew 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
Withdrew 500.0 from account 1001. New balance: 500.0
Deposited 200 into account 1001 . New balance: 700.0
Withdrew 100 from account 1001. New balance: 600.0
Deposited 300.5 into account 1001 . New balance: 900.5
Withdrew 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:\Hexaware\Assignment\assignment 3\Python>
```

Task(10,11,12,13):

main2(task10-13).py

oops > main2(task10-13).py > Customer

```python
import re
class InsufficientFundException(Exception):
    pass


class InvalidAccountException(Exception):
    pass


class OverDraftLimitExceeded(Exception):
    pass

class Customer:

    #Customer class representing a bank customer.

    def __init__(self, customer_id, first_name, last_name, email, phone_number, address):
        #Constructor for Customer class.

        self.customer_id = customer_id
        self.first_name = first_name
        self.last_name = last_name
        self.set_email(email)
        self.set_phone_number(phone_number)
        self.address = address

    def set_email(self, email):
        # Simple email validation using regex
        email_pattern = re.compile(r"[^@]+@[^@]+\.[^@]+")
        if email_pattern.match(email):
            self.email = email
        else:
            raise ValueError("Invalid email address.")
```

main2(task10-13).py

oops > main2(task10-13).py > Bank

```python
    def set_phone_number(self, phone_number):

        # Simple phone number validation for 10 digits
        if re.match(r"^\d{10}$", phone_number):
            self.phone_number = phone_number
        else:
            raise ValueError("Invalid phone number.")


class Account:

    #Account class representing a bank account.

    account_number_generator = 1000  # Starting account number

    def __init__(self, customer, account_type, balance=0.0):

        self.account_number = Account.account_number_generator + 1
        Account.account_number_generator += 1
        self.customer = customer
        self.account_type = account_type
        self.balance = balance


class Bank:

    #Bank class representing the banking system.

    accounts = []  # List to store created accounts

    def create_account(self, customer, acc_type, balance):

        account = Account(customer, acc_type, balance)
        Bank.accounts.append(account)
        return account
```

main2(task10-13).py

oops > main2(task10-13).py > Bank

```python
    def get_account_balance(self, account_number):

        for account in Bank.accounts:
            if account.account_number == account_number:
                return account.balance
        raise InvalidAccountException("Account not found.")

    def deposit(self, account_number, amount):

        for account in Bank.accounts:
            if account.account_number == account_number:
                account.balance += amount
                return account.balance
        raise InvalidAccountException("Account not found.")

    def withdraw(self, account_number, amount):

        for account in Bank.accounts:
            if account.account_number == account_number:
                if account.balance >= amount:
                    account.balance -= amount
                    return account.balance
                else:
                    raise InsufficientFundException("Insufficient funds.")
        raise InvalidAccountException("Account not found.")

    def transfer(self, from_account_number, to_account_number, amount):
        #Transfer money from one account to another.

        from_account = None
        to_account = None

        for account in Bank.accounts:
            if account.account_number == from_account_number:
                from_account = account
            elif account.account_number == to_account_number:
                to_account = account
```

main2(task10-13).py

oops > main2(task10-13).py > Bank

```python
                to_account = account

        if from_account and to_account:
            if from_account.balance >= amount:
                from_account.balance -= amount
                to_account.balance += amount
                return from_account.balance, to_account.balance
            else:
                raise InsufficientFundException("Insufficient funds.")
        else:
            raise InvalidAccountException("One or both accounts not found.")

    def get_account_details(self, account_number):

        #Retrieve account and customer details for a given account number.
        for account in Bank.accounts:
            if account.account_number == account_number:
                return account, account.customer
        raise InvalidAccountException("Account not found.")


class BankApp:

    #BankApp class to simulate the banking system.

    def main(self):
        bank = Bank()
        while True:
            try:
                print("\nBanking System Menu:")
                print("1. Create Account")
                print("2. Deposit")
                print("3. Withdraw")
                print("4. Transfer")
                print("5. Get Account Balance")
                print("6. Get Account Details")
                print("7. Exit")
```

main2(task10-13).py

oops > main2(task10-13).py > Bank

```python
            choice = input("Enter your choice (1-7): ")

            if choice == '1':
                customer_id = int(input("Enter Customer ID: "))
                first_name = input("Enter First Name: ")
                last_name = input("Enter Last Name: ")
                email = input("Enter Email Address: ")
                phone_number = input("Enter Phone Number: ")
                address = input("Enter Address: ")

                try:
                    customer = Customer(customer_id, first_name, last_name, email, phone_number, address)
                    acc_type = input("Choose Account Type (Savings/Current): ").capitalize()
                    balance = float(input("Enter Initial Balance: "))
                    bank.create_account(customer, acc_type, balance)
                    print("Account created successfully.")
                except ValueError as e:
                    print(f"Error: {e}")

            elif choice == '2':
                account_number = int(input("Enter Account Number: "))
                amount = float(input("Enter Deposit Amount: "))
                try:
                    balance = bank.deposit(account_number, amount)
                    print(f"Deposit successful. Current balance: {balance}")
                except ValueError as e:
                    print(f"Error: {e}")

            elif choice == '3':
                account_number = int(input("Enter Account Number: "))
                amount = float(input("Enter Withdrawal Amount: "))
                try:
                    balance = bank.withdraw(account_number, amount)
                    print(f"Withdrawal successful. Current balance: {balance}")
                except ValueError as e:
                    print(f"Error: {e}")
```

main2(task10-13).py

oops > main2(task10-13).py > Bank

```python
            elif choice == '4':
                from_account_number = int(input("Enter Source Account Number: "))
                to_account_number = int(input("Enter Destination Account Number: "))
                amount = float(input("Enter Transfer Amount: "))
                try:
                    from_balance, to_balance = bank.transfer(from_account_number, to_account_number, amount)
                    print(f"Transfer successful. Source balance: {from_balance}, Destination balance: {to_balance}")
                except ValueError as e:
                    print(f"Error: {e}")

            elif choice == '5':
                account_number = int(input("Enter Account Number: "))
                try:
                    balance = bank.get_account_balance(account_number)
                    print(f"Current balance: {balance}")
                except ValueError as e:
                    print(f"Error: {e}")

            elif choice == '6':
                account_number = int(input("Enter Account Number: "))
                try:
                    account, customer = bank.get_account_details(account_number)
                    print(f"Account Details:\n{account.__dict__}\nCustomer Details:\n{customer.__dict__}")
                except ValueError as e:
                    print(f"Error: {e}")

            elif choice == '7':
                print("Exiting the Banking System.")
                break

            else:
                print("Invalid choice. Please enter a valid option.")

        except (InsufficientFundException, InvalidAccountException, OverDraftLimitExceeded) as e:
            print(f"Error: {e}")

        except Exception as e:
```

```python
                break

            else:
                print("Invalid choice. Please enter a valid option.")

        except (InsufficientFundException, InvalidAccountException, OverDraftLimitExceeded) as e:
            print(f"Error: {e}")

        except Exception as e:
            print(f"An unexpected error occurred: {e}")


bank_app = BankApp()
bank_app.main()
```

Outputs:

```
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/oops/main2(task10-13).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: jalgaon
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
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                                                    Python

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@eg.com', 'phone_number': '1234567895', 'address': 'jalgaon'}

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:

```python
database.py
import mysql.connector
from mysql.connector import Error
from datetime import datetime
class InvalidAccountException(Exception):
    pass

class InsufficientFundException(Exception):
    pass

class OverDraftLimitExceeded(Exception):
    pass


class Customer:
    def __init__(self, customer_id, first_name, last_name, email, phone_number, address):
        self.customer_id = customer_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.phone_number = phone_number
        self.address = address


class Account:
    last_account_number = 0

    def __init__(self, customer, account_type, balance=0.0):
        Account.last_account_number += 1
        self.account_number = Account.last_account_number
        self.customer = customer
        self.account_type = account_type
        self.balance = balance
```

database.py

oops > database.py > ...

```python
class Transaction:
    def __init__(self, account, description, transaction_type, transaction_amount):
        self.account = account
        self.description = description
        self.date_time = datetime.now()
        self.transaction_type = transaction_type
        self.transaction_amount = transaction_amount


class SavingsAccount(Account):
    def __init__(self, customer, interest_rate):
        super().__init__(customer, "Savings")
        self.interest_rate = interest_rate


class CurrentAccount(Account):
    def __init__(self, customer, overdraft_limit):
        super().__init__(customer, "Current")
        self.overdraft_limit = overdraft_limit


class ZeroBalanceAccount(Account):
    def __init__(self, customer):
        super().__init__(customer, "zerobalance")

class DBUtil:
    @staticmethod
    def get_db_conn():
        try:
            connection = mysql.connector.connect(
                host="localhost",
                user="root",
                password="root",
                database="hmbank"
            )
            return connection
```

database.py

oops > database.py > ...

```python
            return connection

        except Exception as e:
            print(f"Error: {e}")
            return None

class BankRepositoryImpl():
    def create_account(self, customer, acc_type, balance):
        try:
            conn = DBUtil.get_db_conn()
            cursor = conn.cursor()

            # Insert customer details
            cursor.execute("INSERT INTO customers (customer_id, first_name, last_name, email, phone_number, address) "
                           "VALUES (%s, %s, %s, %s, %s, %s)",
                           (customer.customer_id, customer.first_name, customer.last_name, customer.email,
                            customer.phone_number, customer.address))
            conn.commit()

            # Insert account details
            cursor.execute("INSERT INTO accounts (account_number, account_type, balance, customer_id) "
                           "VALUES (%s, %s, %s, %s)",
                           (Account.last_account_number, acc_type, balance, customer.customer_id))
            conn.commit()

            return Account.last_account_number

        except Error as e:
            print(f"Error: {e}")
            return None

        finally:
            if conn.is_connected():
                cursor.close()
                conn.close()
```

database.py

oops > database.py > ...

```python
    def list_accounts(self):
        try:
            conn = DBUtil.get_db_conn()
            cursor = conn.cursor()

            # Retrieve all accounts
            cursor.execute("SELECT * FROM accounts")
            accounts = cursor.fetchall()

            return [self.map_account(row) for row in accounts]

        except Error as e:
            print(f"Error: {e}")
            return None

        finally:
            if conn.is_connected():
                cursor.close()
                conn.close()

    def calculate_interest(self):
        try:
            conn = DBUtil.get_db_conn()
            cursor = conn.cursor()

            # Calculate interest for savings accounts
            cursor.execute("UPDATE accounts SET balance = balance * (1 + interest_rate / 100) "
                           "WHERE account_type = 'Savings'")
            conn.commit()

        except Error as e:
            print(f"Error: {e}")

        finally:
            if conn.is_connected():
                cursor.close()
                conn.close()
```

database.py

oops > database.py > ...

```python
    def get_account_balance(self, account_number):
        try:
            conn = DBUtil.get_db_conn()
            cursor = conn.cursor()

            # Retrieve account balance
            cursor.execute("SELECT balance FROM accounts WHERE account_number = %s", (account_number,))
            balance = cursor.fetchone()

            if balance:
                return balance[0]
            else:
                return "Account number is not present"

        except Error as e:
            print(f"Error: {e}")
            return None

        finally:
            if conn.is_connected():
                cursor.close()
                conn.close()

    def deposit(self, account_number, amount):
        try:
            conn = DBUtil.get_db_conn()
            cursor = conn.cursor()

            # Update account balance for deposit
            cursor.execute("UPDATE accounts SET balance = balance + %s WHERE account_number = %s", (amount, account_number))
            conn.commit()

            # Record the transaction
            self.record_transaction(account_number, "Deposit", "Deposit", amount)

            return self.get_account_balance(account_number)
```

database.py

oops > database.py > ...

```python
        except Error as e:
            print(f"Error: {e}")
            return None

        finally:
            if conn.is_connected():
                cursor.close()
                conn.close()

    def withdraw(self, account_number, amount):
        try:
            conn = DBUtil.get_db_conn()
            cursor = conn.cursor()

            # Retrieve current account balance
            cursor.execute("SELECT balance, account_type FROM accounts WHERE account_number = %s", (account_number,))
            result = cursor.fetchone()

            if result:
                balance, account_type = result
                if account_type == "Savings" and (balance - amount) < 500:
                    raise InsufficientFundException("Withdrawal violates minimum balance rule.")

                if account_type == "Current" and (balance - amount) < 0:
                    raise OverDraftLimitExceeded("Withdrawal exceeds available balance and overdraft limit.")

                # Update account balance for withdrawal
                cursor.execute("UPDATE accounts SET balance = balance - %s WHERE account_number = %s",
                               (amount, account_number))
                conn.commit()

                # Record the transaction
                self.record_transaction(account_number, "Withdrawal", "Withdrawal", amount)

                return self.get_account_balance(account_number)

            else:
```

database.py

oops > database.py > ...

```python
            else:
                raise InvalidAccountException("Account not found.")

        except (InsufficientFundException, OverDraftLimitExceeded) as e:
            raise e

        except Error as e:
            print(f"Error: {e}")
            return None

        finally:
            if conn.is_connected():
                cursor.close()
                conn.close()

    def transfer(self, from_account_number, to_account_number, amount):
        try:
            conn = DBUtil.get_db_conn()
            cursor = conn.cursor()

            # Check if both accounts exist
            if not self.account_exists(from_account_number) or not self.account_exists(to_account_number):
                raise InvalidAccountException("One or both accounts not found.")

            # Check if the source account has sufficient funds
            from_balance = self.get_account_balance(from_account_number)
            if from_balance < amount:
                raise InsufficientFundException("Insufficient funds for the transfer.")

            # Update source account balance for withdrawal
            cursor.execute("UPDATE accounts SET balance = balance - %s WHERE account_number = %s",
                           (amount, from_account_number))
            conn.commit()

            # Update destination account balance for deposit
            cursor.execute("UPDATE accounts SET balance = balance + %s WHERE account_number = %s",
                           (amount, to_account_number))
```

database.py

oops > database.py > ...

```python
                        (amount, to_account_number))
            conn.commit()

            # Record the transactions
            self.record_transaction(from_account_number, "Transfer", "Transfer to " + str(to_account_number), amount)
            self.record_transaction(to_account_number, "Transfer", "Transfer from " + str(from_account_number), amount)

            return self.get_account_balance(from_account_number), self.get_account_balance(to_account_number)

        except (InsufficientFundException, InvalidAccountException) as e:
            raise e

        except Error as e:
            print(f"Error: {e}")
            return None

        finally:
            if conn.is_connected():
                cursor.close()
                conn.close()

    def get_account_details(self, account_number):
        try:
            conn = DBUtil.get_db_conn()
            cursor = conn.cursor()

            # Retrieve account details
            cursor.execute("SELECT * FROM accounts INNER JOIN customers "
                           "ON accounts.customer_id = customers.customer_id "
                           "WHERE account_number = %s", (account_number,))
            result = cursor.fetchone()

            if result:
                account_details = {
                    "account_number": result[0],
                    "account_type": result[1],
                    "balance": result[2],
```

database.py

oops > database.py > ...

```python
                    "account_type": result[1],
                    "balance": result[2],
                    "customer_id": result[3],
                    "first_name": result[5],
                    "last_name": result[6],
                    "email": result[7],
                    "phone_number": result[8],
                    "address": result[9]
                }
                return account_details

            else:
                raise InvalidAccountException("Account not found.")

        except InvalidAccountException as e:
            raise e

        except Error as e:
            print(f"Error: {e}")
            return None

        finally:
            if conn.is_connected():
                cursor.close()
                conn.close()

    def get_transactions(self, account_number, from_date, to_date):
        try:
            conn = DBUtil.get_db_conn()
            cursor = conn.cursor()

            # Retrieve transactions within the date range
            cursor.execute("SELECT * FROM transactions WHERE account_number = %s "
                           "AND date_time BETWEEN %s AND %s",
                           (account_number, from_date, to_date))
            transactions = cursor.fetchall()
```

database.py

oops > database.py > ...

```python
            return [self.map_transaction(row) for row in transactions]

        except Error as e:
            print(f"Error: {e}")
            return None

        finally:
            if conn.is_connected():
                cursor.close()
                conn.close()

    def record_transaction(self, account_number, description, transaction_type, transaction_amount):
        try:
            conn = DBUtil.get_db_conn()
            cursor = conn.cursor()

            # Record the transaction
            cursor.execute("INSERT INTO transactions (account_number, description, date_time, transaction_type, transaction_amount) "
                           "VALUES (%s, %s, %s, %s, %s)",
                           (account_number, description, datetime.now(), transaction_type, transaction_amount))
            conn.commit()

        except Error as e:
            print(f"Error: {e}")

        finally:
            if conn.is_connected():
                cursor.close()
                conn.close()

    @staticmethod
    def map_account(row):
        return {
            "account_number": row[0],
            "account_type": row[1],
            "balance": row[2],
```

database.py

oops > database.py > ...

```python
    def map_account(row):
        return {
            "account_number": row[0],
            "account_type": row[1],
            "balance": row[2],
            "customer_id": row[3]
        }

    @staticmethod
    def map_transaction(row):
        return {
            "account_number": row[0],
            "description": row[1],
            "date_time": row[2],
            "transaction_type": row[3],
            "transaction_amount": row[4]
        }

    def account_exists(self, account_number):
        try:
            conn = DBUtil.get_db_conn()
            cursor = conn.cursor()

            # Check if the account exists
            cursor.execute("SELECT * FROM accounts WHERE account_number = %s", (account_number,))
            return cursor.fetchone() is not None

        except Error as e:
            print(f"Error: {e}")
            return False

        finally:
            if conn.is_connected():
                cursor.close()
                conn.close()
```

Inputs :

```python
# Example usage:

# Create an instance of BankRepositoryImpl and test the methods
bank_repository = BankRepositoryImpl()

# Create a customer
customer = Customer(1, "John", "Doe", "john.doe@example.com", "1234567890", "123 Main St")
customer1 = Customer(2, "John1", "Doe1", "john1.doe@example.com", "1234567890", "12345 Main St")

# Create a SavingsAccount
savings_account_number = bank_repository.create_account(customer, "Savings", 1000.0)
print("Savings Account Number:", savings_account_number)

# Create a CurrentAccount
current_account_number = bank_repository.create_account(customer1, "Current", 2000.0)
print("Current Account Number:", current_account_number)

# Perform transactions
bank_repository.deposit(1, 500)
bank_repository.withdraw(1, 100)
#bank_repository.transfer(savings_account_number, current_account_number, 100.0)

# Get account details
savings_account_details = bank_repository.get_account_details(1)
current_account_details = bank_repository.get_account_details(1)

print("Savings Account Details:", savings_account_details)
print("Current Account Details:", current_account_details)

# Get account balance
savings_balance = bank_repository.get_account_balance(1)
current_balance = bank_repository.get_account_balance(1)

print("Savings Account Balance:", savings_balance)
#print("Current Account Balance:", current_balance)
```

```python
# List all accounts
accounts_list = bank_repository.list_accounts()
print("All Accounts:", accounts_list)

# Calculate interest for savings accounts
bank_repository.calculate_interest()

# Get transactions
transactions = bank_repository.get_transactions(1, datetime(2023, 1, 1), datetime.now())
print("Savings Account Transactions:", transactions)
```

outputs:

```
PS D:\Hexaware\Assignment\assignment 3\Python> & "C:/Program Files/Python312/python.exe" "d:/Hexaware/Assignment/assignment 3/Python/oops/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('1400.00'), 'first_name': 'Rohan', 'last_name': 'Chaud
hari', 'email': None, 'phone_number': 'Rohan@example.com', 'address': '1234567890'}
Current Account Details: {'account_number': 1, 'account_type': 1, 'balance': 'Savings', 'customer_id': Decimal('1400.00'), 'first_name': 'Rohan', 'last_name': 'Chaud
hari', 'email': None, 'phone_number': 'Rohan@example.com', 'address': '1234567890'}
Savings Account Balance: 1400.00
All Accounts: [{'account_number': 1, 'account_type': 1, 'balance': 'Savings', 'customer_id': Decimal('1400.00')}, {'account_number': 2, 'account_type': 2, 'balance':
 'Current', 'customer_id': Decimal('2000.00')}]
```

Database Changes: