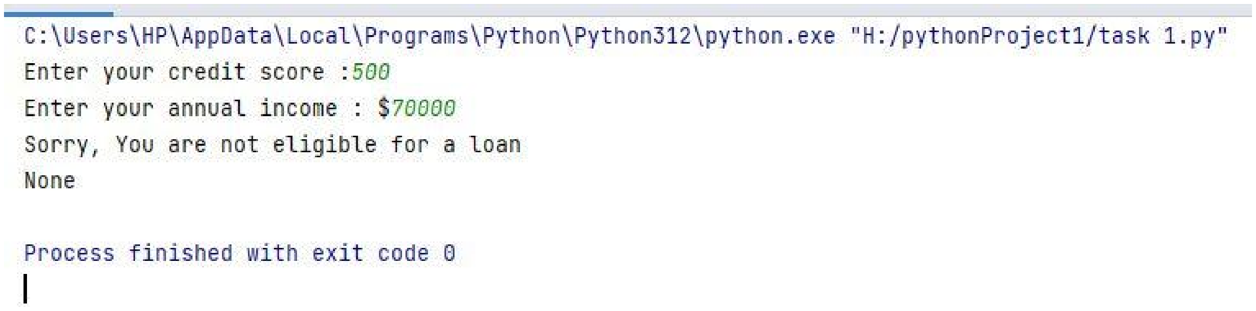
Banking system

Name : ILAKIYA R

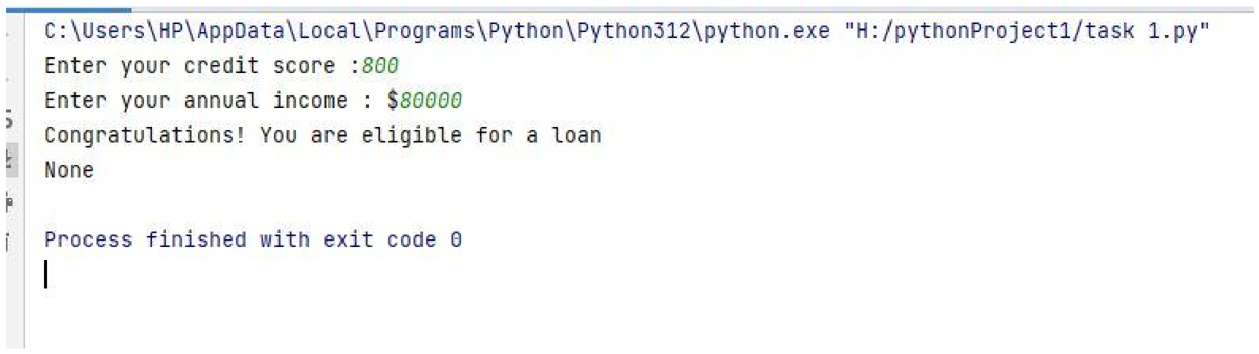
Task 1

In a bank, you have been given the task is to create a program that checks if a customer is eligible for a loan based on their credit score and income. The eligibility criteria are as follows:
Credit Score must be above 700.
Annual Income must be at least $50,000.
Tasks:
Write a program that takes the customer's credit score and annual income as input.
Use conditional statements (if-else) to determine if the customer is eligible for a loan.
Display an appropriate message based on eligibility.

```python
def check_loan_eligibility(credit_score,annual_income):
   if credit_score > 700 and annual_income >= 50000:
       print("Congratulations! You are eligible for a loan")
   else:
       print("Sorry, You are not eligible for a loan")
credit_score = int(input("Enter your credit score :"))
annual_income = float(input("Enter your annual income : $"))
print(check_loan_eligibility(credit_score,annual_income))
```



```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 1.py"
Enter your credit score :500
Enter your annual income : $70000
Sorry, You are not eligible for a loan
None

Process finished with exit code 0
```

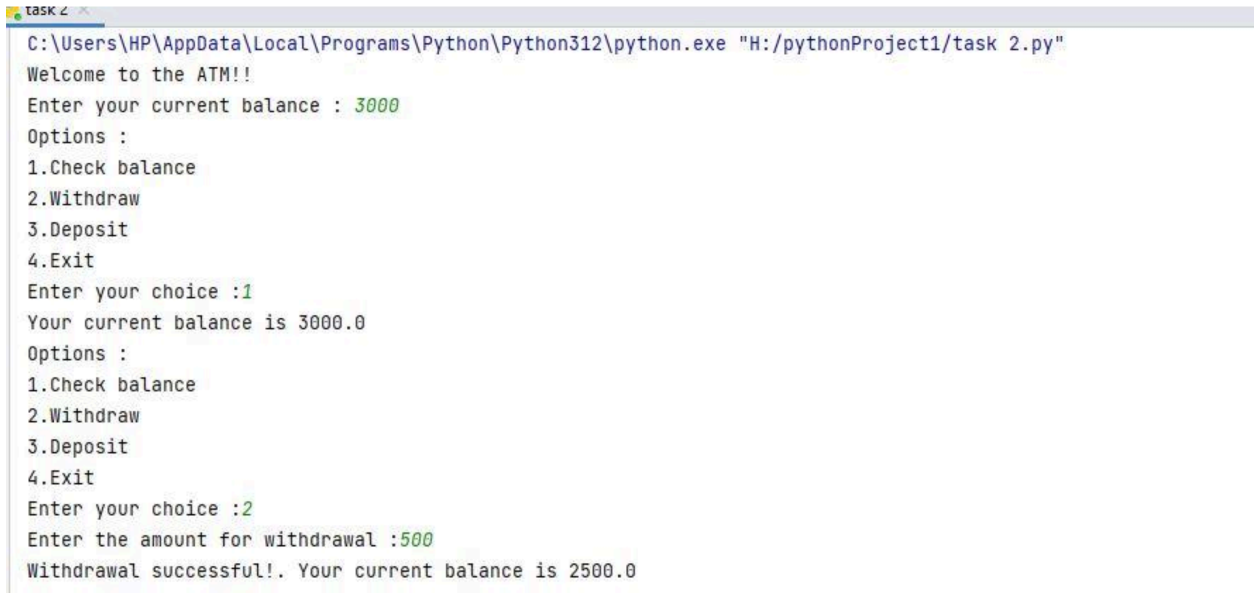


```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 1.py"
Enter your credit score :800
Enter your annual income : $80000
Congratulations! You are eligible for a loan
None

Process finished with exit code 0
```

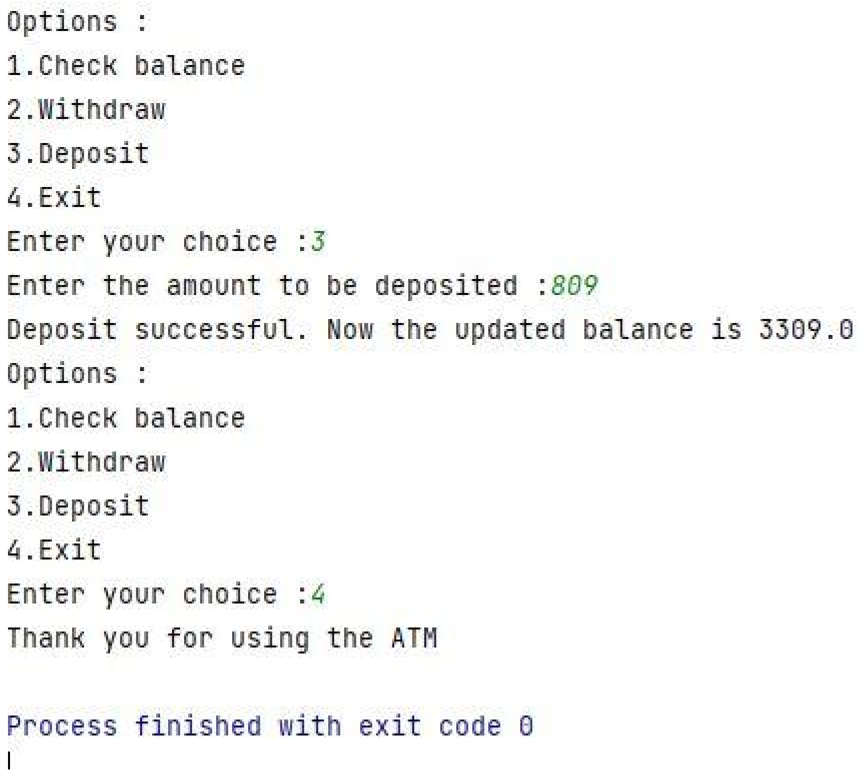
Task 2

Create a program that simulates an ATM transaction. Display options such as "Check Balance," "Withdraw," "Deposit,". Ask the user to enter their current balance and the amount they want to withdraw or deposit. Implement checks to ensure that the withdrawal amount is not greater than the available balance and that the withdrawal amount is in multiples of 100 or 500. Display appropriate messages for success or failure.

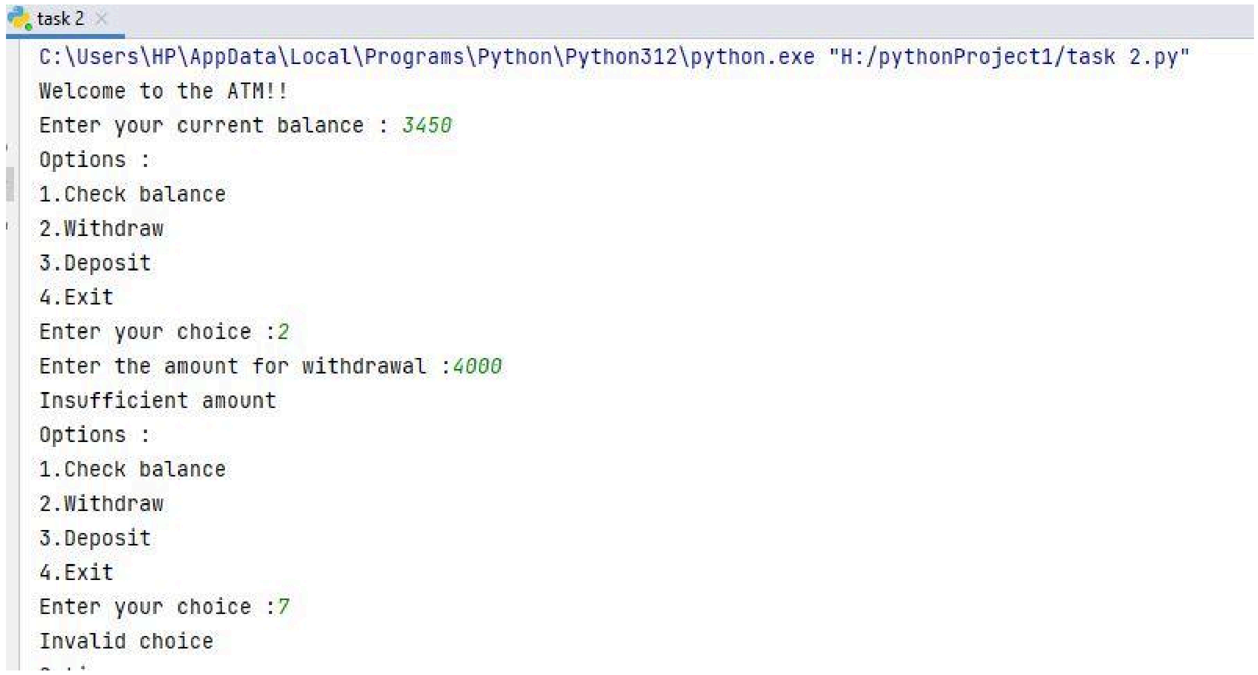
```python
print("Welcome to the ATM!!")
balance = float(input("Enter your current balance : "))
while True:
   print("Options :")
   print("1.Check balance")
   print("2.Withdraw")
   print("3.Deposit")
   print("4.Exit")
   choice = input("Enter your choice :")
   if choice == "1":
       print(f"Your current balance is {balance}")
   elif choice == "2":
       amount = float(input("Enter the amount for withdrawal :"))
       if amount > balance:
           print("Insufficient amount")
       elif amount % 100 != 0 and amount % 500 != 0:
           print("Withdrawal amount must be in multiples of 100 or 500")
       else:
           balance = balance - amount
           print(f"Withdrawal successful!. Your current balance is {balance}")
   elif choice == "3":
       amount = float(input("Enter the amount to be deposited :"))
       balance = balance + amount
       print(f"Deposit successful. Now the updated balance is {balance}")
   elif choice == "4":
       print("Thank you for using the ATM")
       break
   else:
       print("Invalid choice")
```



```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 2.py"
Welcome to the ATM!!
Enter your current balance : 3000
Options :
1.Check balance
2.Withdraw
3.Deposit
4.Exit
Enter your choice :1
Your current balance is 3000.0
Options :
1.Check balance
2.Withdraw
3.Deposit
4.Exit
Enter your choice :2
Enter the amount for withdrawal :500
Withdrawal successful!. Your current balance is 2500.0
```



```
Options :
1.Check balance
2.Withdraw
3.Deposit
4.Exit
Enter your choice :3
Enter the amount to be deposited :809
Deposit successful. Now the updated balance is 3309.0
Options :
1.Check balance
2.Withdraw
3.Deposit
4.Exit
Enter your choice :4
Thank you for using the ATM

Process finished with exit code 0
```



```
task 2
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 2.py"
Welcome to the ATM!!
Enter your current balance : 3450
Options :
1.Check balance
2.Withdraw
3.Deposit
4.Exit
Enter your choice :2
Enter the amount for withdrawal :4000
Insufficient amount
Options :
1.Check balance
2.Withdraw
3.Deposit
4.Exit
Enter your choice :7
Invalid choice
```

Task 3

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

Tasks:

Create a program that calculates the future balance of a savings account.

Use a loop structure (e.g., for loop) to calculate the balance for multiple customers.

Prompt the user to enter the initial balance, annual interest rate, and the number of years.

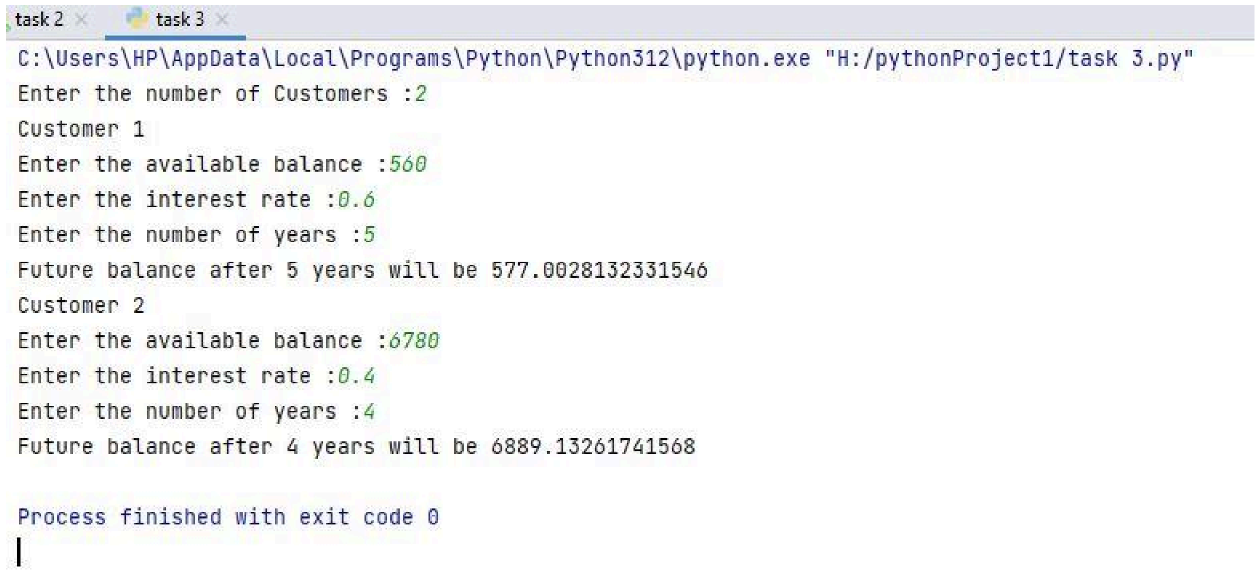
Calculate the future balance using the formula:

future_balance = initial_balance * (1 + annual_interest_rate/100)^years.

Display the future balance for each customer.

```python
a = int(input("Enter the number of Customers :"))
for i in range(a):
   print("Customer",i+1)
   available_balance = int(input("Enter the available balance :"))
   interest_rate = float(input("Enter the interest rate :"))
   years = int(input("Enter the number of years :"))
   future_balance = available_balance * ((1+ (interest_rate/100)) ** years)
```

```
    print(f"Future balance after {years} years will be {future_balance}")
```



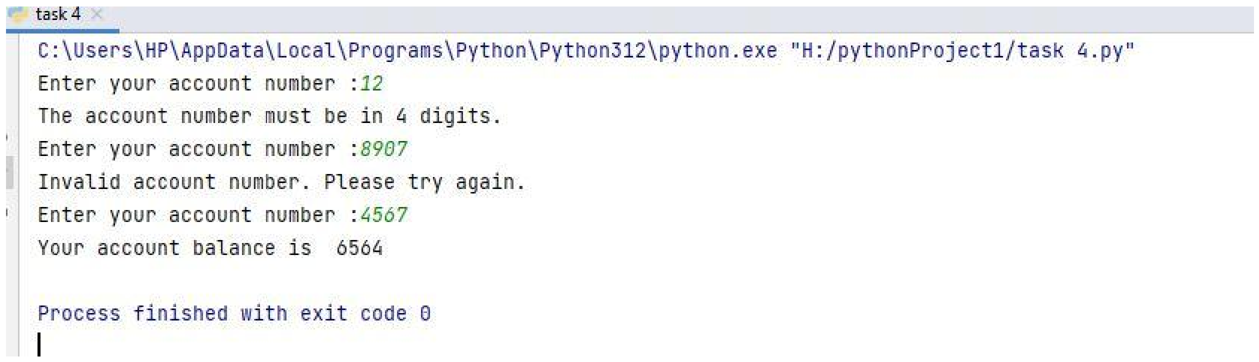
task 2 task 3

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 3.py"
Enter the number of Customers :2
Customer 1
Enter the available balance :560
Enter the interest rate :0.6
Enter the number of years :5
Future balance after 5 years will be 577.0028132331546
Customer 2
Enter the available balance :6780
Enter the interest rate :0.4
Enter the number of years :4
Future balance after 4 years will be 6889.13261741568

Process finished with exit code 0
```

Task 4

You are tasked with creating a program that allows bank customers to check their account balances. The program should handle multiple customer accounts, and the customer should be able to enter their account number, balance to check the balance.
Tasks:
Create a Python program that simulates a bank with multiple customer accounts.
Use a loop (e.g., while loop) to repeatedly ask the user for their account number and balance until they enter a valid account number.
Validate the account number entered by the user.
If the account number is valid, display the account balance. If not, ask the user to try again.

```
accounts = {"1234":1000,"2345":3479,"3456":4568,"4567":6564}
while True:
    account_number = input("Enter your account number :")
    if account_number in accounts:
        print("Your account balance is ",accounts[account_number])
        break
    elif len(account_number) < 4:
        print("The account number must be in 4 digits.")
    else:
        print("Invalid account number. Please try again.")
```

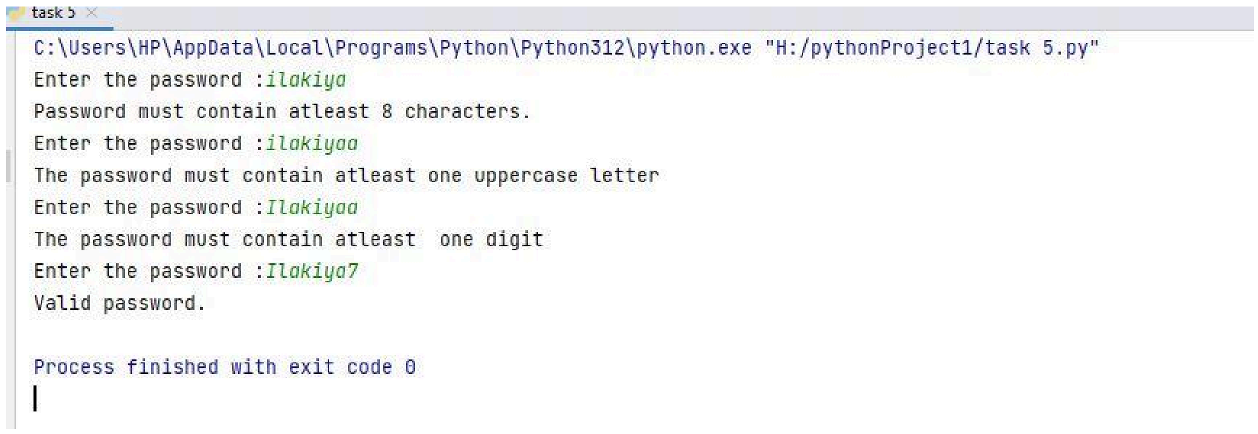


```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 4.py"
Enter your account number :12
The account number must be in 4 digits.
Enter your account number :8907
Invalid account number. Please try again.
Enter your account number :4567
Your account balance is  6564

Process finished with exit code 0
```

Task 5

Write a program that prompts the user to create a password for their bank account. Implement if conditions to validate the password according to these rules:
The password must be at least 8 characters long.
It must contain at least one uppercase letter.
It must contain at least one digit.
Display appropriate messages to indicate whether their password is valid or not.

```
while True:
      password = str(input("Enter the password :"))
      if len(password) < 8:
          print("Password must contain atleast 8 characters.")
      elif not any(char.isupper() for char in password):
          print("The password must contain atleast one uppercase letter")
      elif not any(char.isdigit() for char in password):
          print("The password must contain atleast  one digit")
      else:
          print("Valid password.")
          break
```


task 5

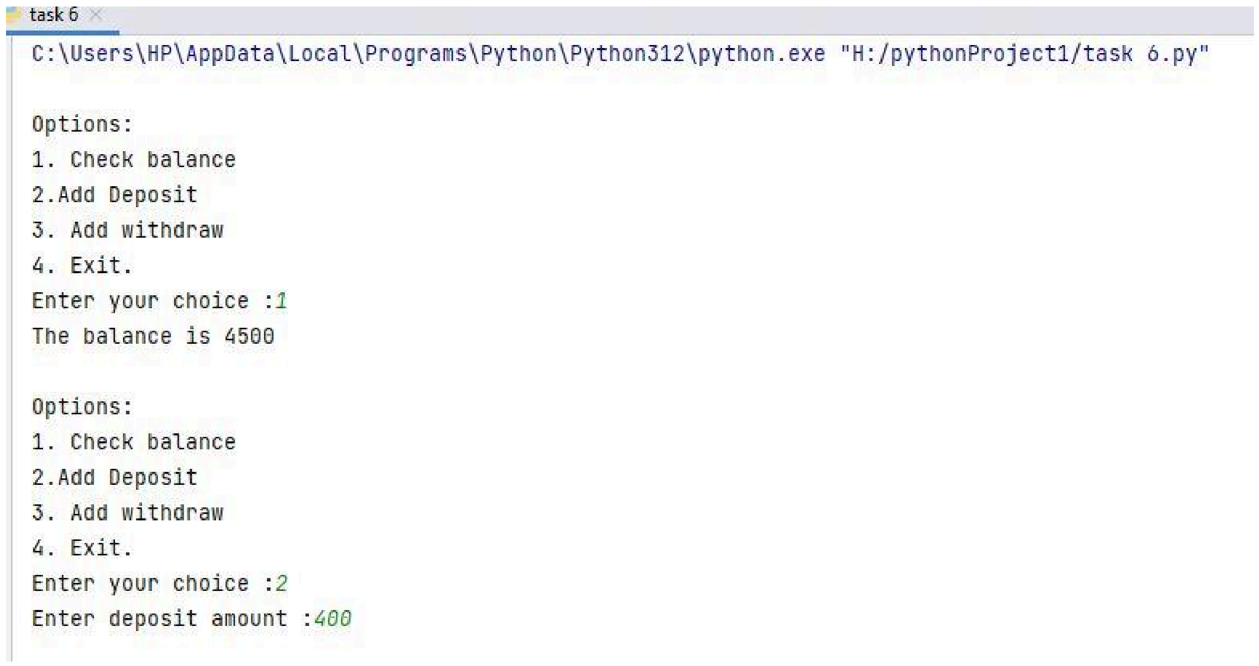
```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 5.py"
Enter the password :ilakiya
Password must contain atleast 8 characters.
Enter the password :ilakiyaa
The password must contain atleast one uppercase letter
Enter the password :Ilakiyaa
The password must contain atleast  one digit
Enter the password :Ilakiya7
Valid password.

Process finished with exit code 0
```

Task 6

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

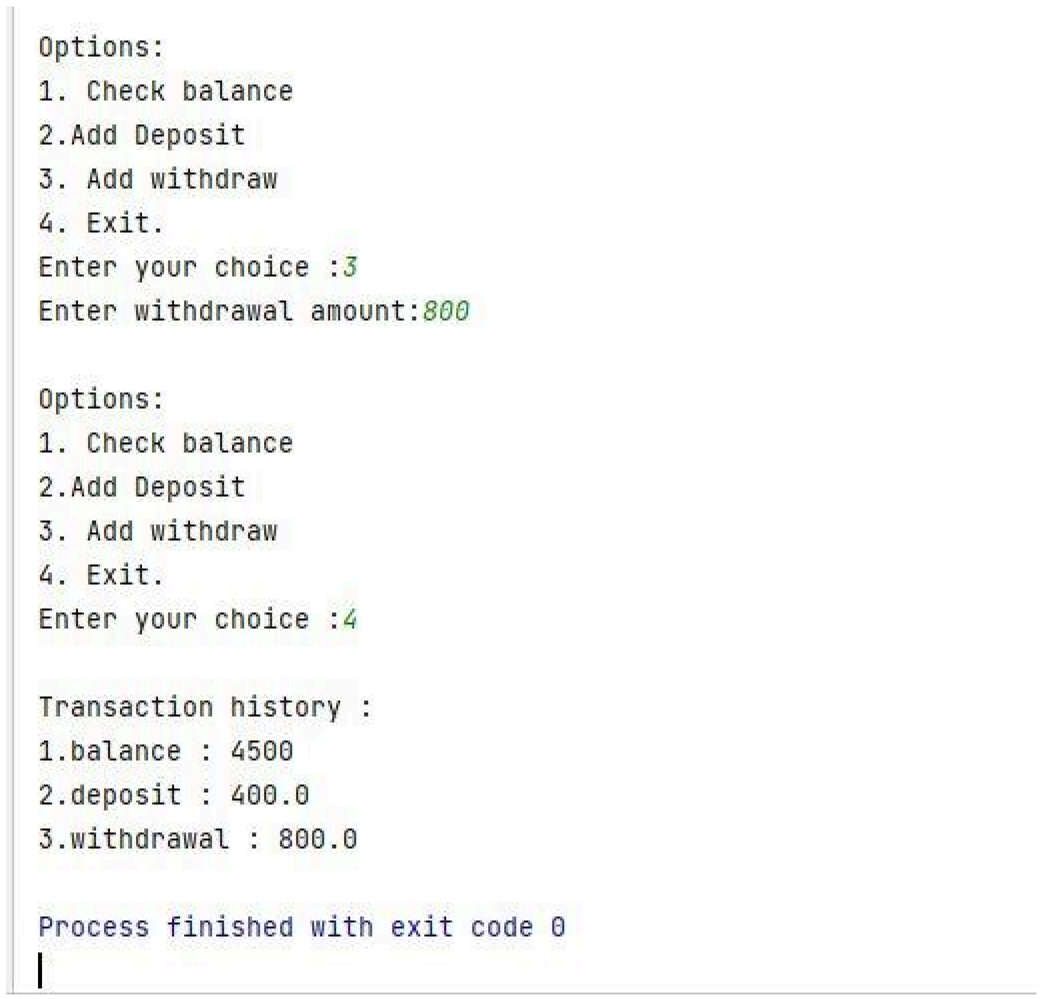
```
class Transaction:
   def __init__(self,transaction_type,amount):
       self.transaction_type = transaction_type
       self.amount = amount
transactions = []
while True:
   print("\nOptions:")
   print("1. Check balance")
   print("2.Add Deposit")
   print("3. Add withdraw")
   print("4. Exit.")
   choice = input("Enter your choice :")
   if choice == "1":
       balance = 30000
       print(f"The balance is {balance}")
       transactions.append(Transaction("balance",balance))
   elif choice == "2":
       amount = float(input("Enter deposit amount :"))
       transactions.append(Transaction("deposit",amount))
   elif choice == "3":
       amount = float(input("Enter withdrawal amount:"))
       transactions.append(Transaction("withdrawal", amount))
   elif choice == "4":
```

```
            break
        else:
            print("Invalid choice. Please try again.")
print("\nTransaction history :")
for index,transactions in enumerate(transactions,start=1):
    print(f"{index}.{transactions.transaction_type} : {transactions.amount}")
```



task 6

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 6.py"

Options:
1. Check balance
2.Add Deposit
3. Add withdraw
4. Exit.
Enter your choice :1
The balance is 4500

Options:
1. Check balance
2.Add Deposit
3. Add withdraw
4. Exit.
Enter your choice :2
Enter deposit amount :400
```


```
Options:
1. Check balance
2.Add Deposit
3. Add withdraw
4. Exit.
Enter your choice :3
Enter withdrawal amount:800

Options:
1. Check balance
2.Add Deposit
3. Add withdraw
4. Exit.
Enter your choice :4

Transaction history :
1.balance : 4500
2.deposit : 400.0
3.withdrawal : 800.0

Process finished with exit code 0
```

Task 7

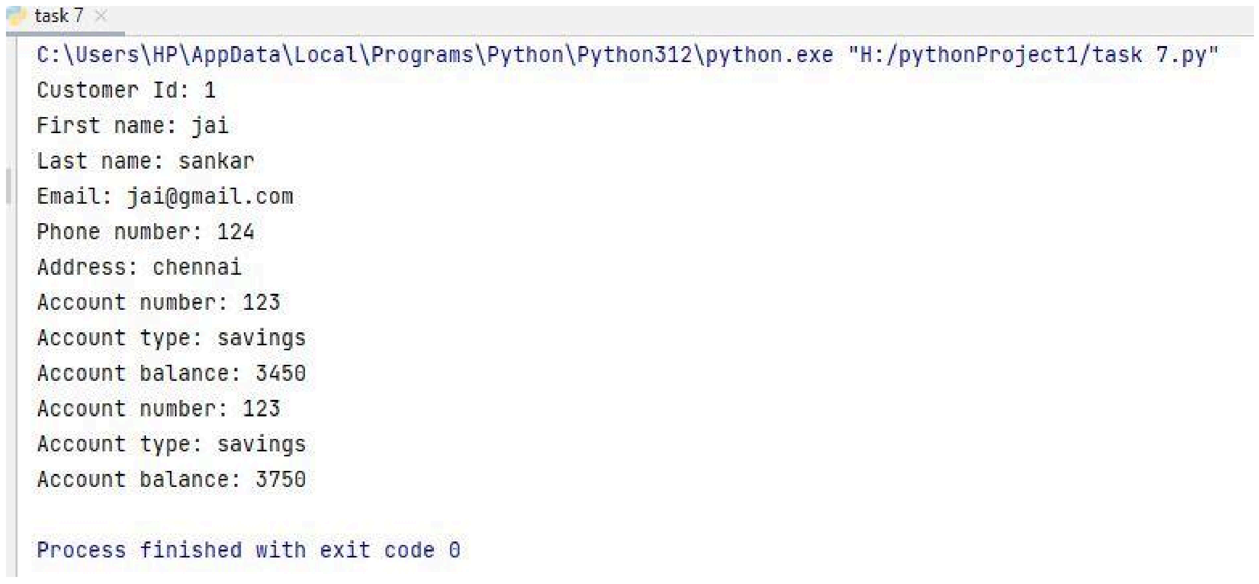
Create a `Customer` class with the following confidential attributes:
Attributes
Customer ID
First Name
Last Name
Email Address
Phone Number
Address
Constructor and Methods
Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.
Create an `Account` class with the following confidential attributes:
Attributes

Account Number
Account Type (e.g., Savings, Current)
Account Balance
Constructor and Methods
Implement default constructors and overload the constructor with Account attributes,
Generate getter and setter, (print all information of attribute) methods for the attributes.
Add methods to the `Account` class to allow deposits and withdrawals.
deposit(amount: float): Deposit the specified amount into the account.

withdraw(amount: float): Withdraw the specified amount from the account. withdraw amount only if there is sufficient fund else display insufficient balance.
calculate_interest(): method for calculating interest amount for the available balance. interest rate is fixed to 4.5%
Create a Bank class to represent the banking system. Perform the following operation in main method:
create object for account class by calling parameter constructor.
deposit(amount: float): Deposit the specified amount into the account.
withdraw(amount: float): Withdraw the specified amount from the account.
calculate_interest(): Calculate and add interest to the account balance for savings accounts.

```
class Customer:
                                                                            def
__init__(self,customer_id=None,first_name=None,last_name=None,email=None,phone
_number=None,address=None):
       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
   def __str__(self):
                  return   f"Customer   Id:   {self.customer_id}\nFirst   name:
{self.first_name}\nLast  name:  {self.last_name}\nEmail:  {self.email}\nPhone
number: {self.phone_number}\nAddress: {self.address}"
   def get_customer_id(self):
       return self.customer_id
   def set_customer_id(self,customer_id):
       self.customer_id = customer_id
class Account:
   def __init__(self,account_number=None,account_type=None,account_balance=0):
       self.account_number = account_number
       self.account_type = account_type
       self.account_balance = account_balance
   def __str__(self):
```

```
              return  f"Account  number:  {self.account_number}\nAccount  type:
{self.account_type}\nAccount balance: {self.account_balance}"
   def get_account_number(self):
       return self.account_number
   def set_account_number(self,account_number):
       self.account_number = account_number
   def deposit(self,amount):
       self.account_balance += amount
   def withdraw(self,amount):
       if self.account_balance >= amount:
           self.account_balance -= amount
       else:
           print("Insufficient balance.")
   def calculate_interest(self):
       interest = self.account_balance * 0.045
       self.account_balance += interest
class Bank:
   def __init__(self):
       pass
   def operate_account(self,account,deposit_amount,withdraw_amount):
       account.deposit(deposit_amount)
       account.withdraw(withdraw_amount)
       if account.account_type == "Savings":
           account.calculate_interest()
if __name__ == "__main__":
      customer  =  Customer(customer_id  =  1,first_name  =  "jai",last_name  =
"sankar",email = "jai@gmail.com",phone_number = "124",address = "chennai")
   print(customer)
                                                     account                    =
Account(account_number="123",account_type="savings",account_balance=3450)
   print(account)
   bank = Bank()
   bank.operate_account(account,500,200)
   print(account)
```


task 7

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 7.py"
Customer Id: 1
First name: jai
Last name: sankar
Email: jai@gmail.com
Phone number: 124
Address: chennai
Account number: 123
Account type: savings
Account balance: 3450
Account number: 123
Account type: savings
Account balance: 3750

Process finished with exit code 0
```

Task 8

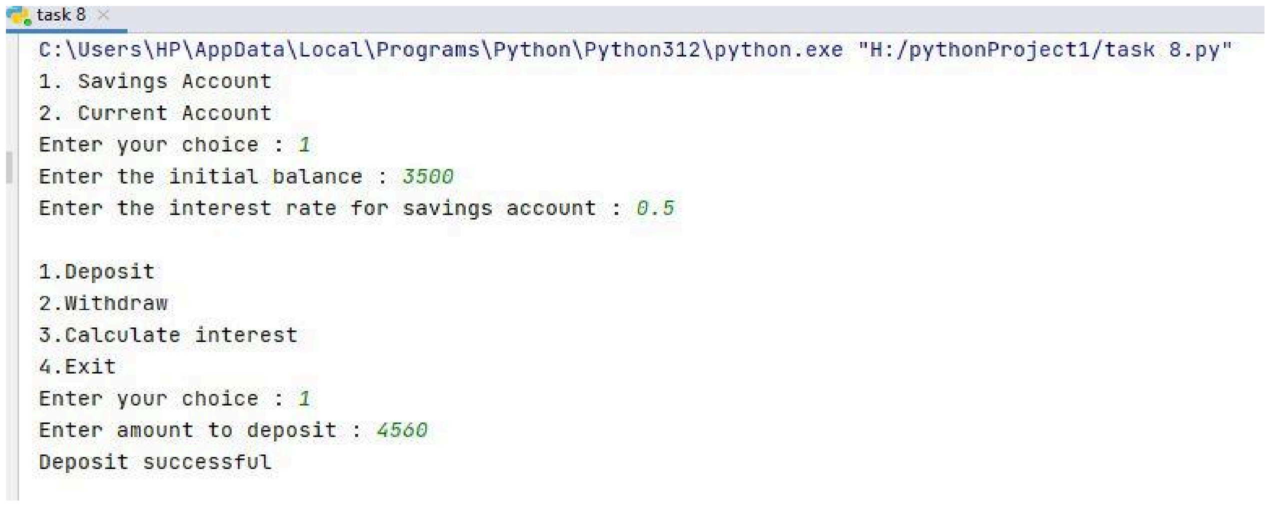
Overload the deposit and withdraw methods in Account class as mentioned below.
deposit(amount: float): Deposit the specified amount into the account.
withdraw(amount: float): Withdraw the specified amount from the account. withdraw amount only if there is sufficient fund else display insufficient balance.
deposit(amount: int): Deposit the specified amount into the account.
withdraw(amount: int): Withdraw the specified amount from the account. withdraw amount only if there is sufficient fund else display insufficient balance.
deposit(amount: double): Deposit the specified amount into the account.
withdraw(amount: double): Withdraw the specified amount from the account. withdraw amount only if there is sufficient fund else display insufficient balance.
Create Subclasses for Specific Account Types
Create subclasses for specific account types (e.g., `SavingsAccount`, `CurrentAccount`) that inherit from the `Account` class.
SavingsAccount: A savings account that includes an additional attribute for interest rate. override the calculate_interest() from Account class method to calculate interest based on the balance and interest rate.
CurrentAccount: A current account that includes an additional attribute
overdraftLimit. A current account with no interest. Implement the withdraw() method to allow overdraft up to a certain limit (configure a constant for the overdraft limit).
Create a Bank class to represent the banking system. Perform the following operation in main method:
Display menu for user to create object for account class by calling parameter

constructor. Menu should display options `SavingsAccount` and `CurrentAccount`. user can choose any one option to create account. use switch case for implementation.
deposit(amount: float): Deposit the specified amount into the account.
withdraw(amount: float): Withdraw the specified amount from the account. For saving account withdraw amount only if there is sufficient fund else display insufficient balance.

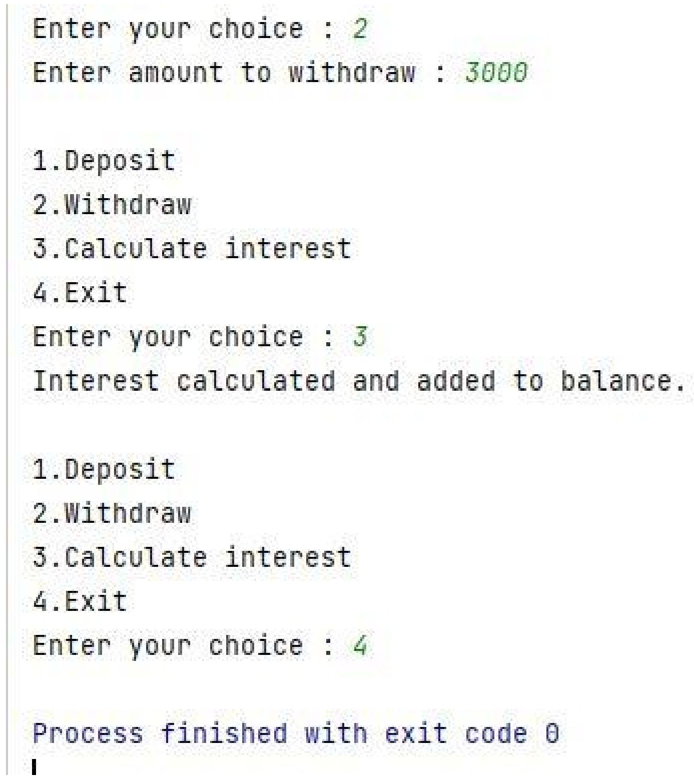
For Current Account withdraw limit can exceed the available balance and should not exceed the overdraft limit.
calculate_interest(): Calculate and add interest to the account balance for savings accounts.

```python
class Account:
   def __init__(self,balance):
       self.balance = balance
   def deposit(self,amount):
       self.balance += amount
   def withdraw(self,amount):
       if self.balance >= amount:
           self.balance -= amount
       else:
           print("Insufficient balance")
   def calculate_interest(self):
       pass
class SavingsAccount(Account):      #inherits from Account class
   def __init__(self,balance,interest_rate):
       super().__init__(balance)
       self.interest_rate = interest_rate
   def calculate_interest(self):
       interest = self.balance * self.interest_rate / 100
       self.balance += interest
class CurrentAccount(Account):
   OVERDRAFT_LIMIT = 1000
   def __init__(self,balance):
       super().__init__(balance)
   def withdraw(self,amount):
       if self.balance + self.OVERDRAFT_LIMIT >= amount:
           self.balance -= amount
       else:
           print("Withdraw amount exceeds the overdraft limit")
class Bank:
   @staticmethod
   def display_menu():
       print("1. Savings Account")
       print("2. Current Account")
       choice = int(input("Enter your choice : "))
       return choice
   @staticmethod
   def create_account():
       choice = Bank.display_menu()
       balance = float(input("Enter the initial balance : "))
       if choice == 1:
             interest_rate = float(input("Enter the interest rate for savings
account : "))
           return SavingsAccount(balance,interest_rate)
       elif choice == 2:
           return CurrentAccount(balance)
       else:
           print("Invalid choice.")
           return None
   @staticmethod
   def perform_operations(account):
       while True:
           print("\n1.Deposit")
           print("2.Withdraw")
           print("3.Calculate interest")
           print("4.Exit")
           option = int(input("Enter your choice : "))
           if option == 1:
               amount = float(input("Enter amount to deposit : "))
               account.deposit(amount)
               print("Deposit successful")
           elif option == 2:
               amount = float(input("Enter amount to withdraw : "))
               account.withdraw(amount)
           elif option == 3:
               if isinstance(account,SavingsAccount):
                   account.calculate_interest()
                   print("Interest calculated and added to balance.")
               else:
                   print("This option is available only for Savings Account")
           elif option == 4:
               break
           else:
               print("Invalid choice")
#main method
if __name__ == '__main__':
   account = Bank.create_account()
   if account:
       Bank.perform_operations(account)
```

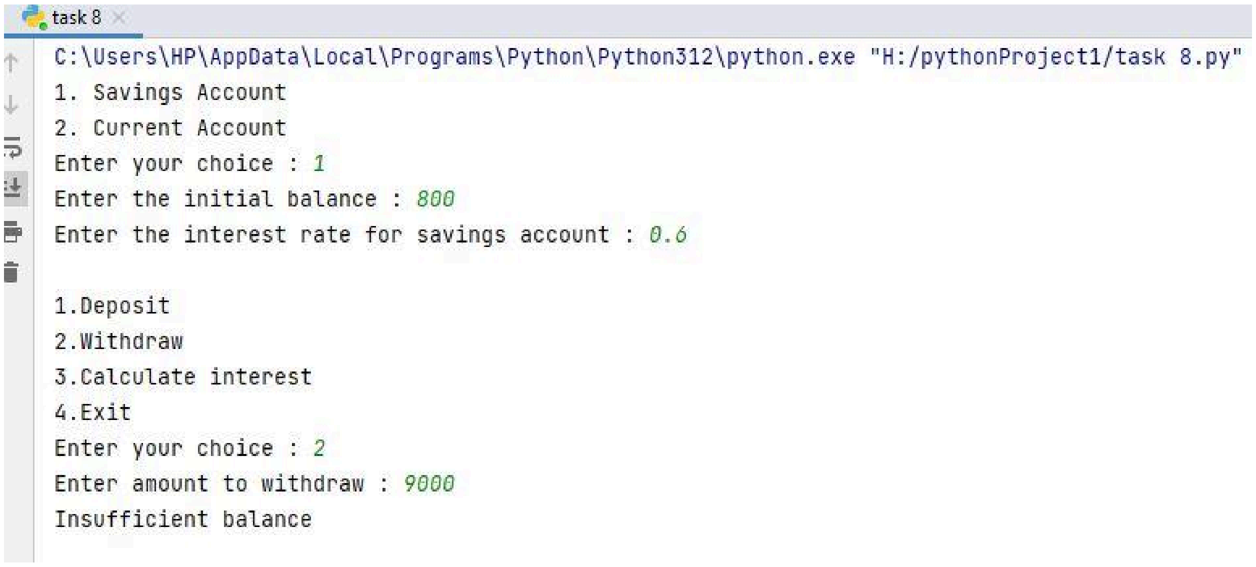
Savings account



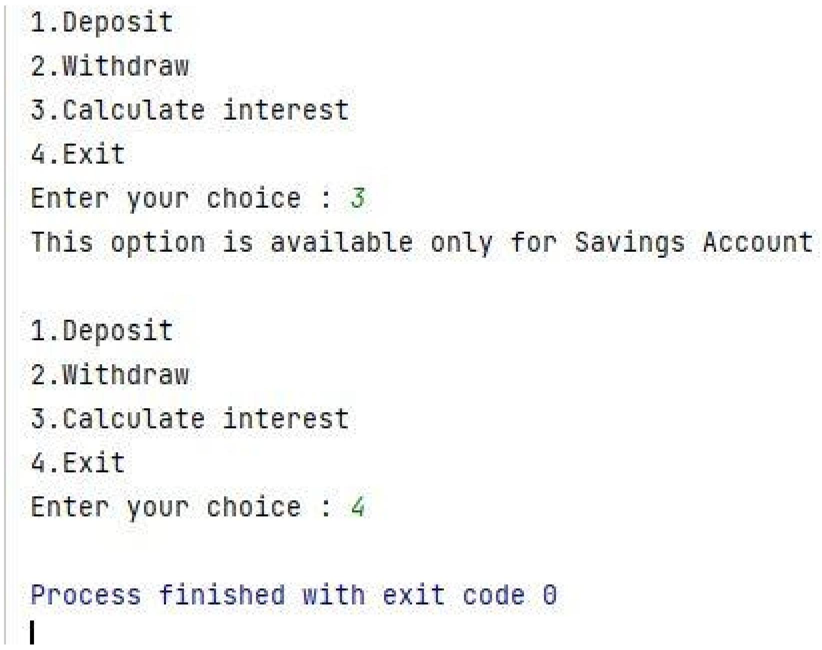
```
task 8
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 8.py"
1. Savings Account
2. Current Account
Enter your choice : 1
Enter the initial balance : 3500
Enter the interest rate for savings account : 0.5

1.Deposit
2.Withdraw
3.Calculate interest
4.Exit
Enter your choice : 1
Enter amount to deposit : 4560
Deposit successful
```



```
Enter your choice : 2
Enter amount to withdraw : 3000

1.Deposit
2.Withdraw
3.Calculate interest
4.Exit
Enter your choice : 3
Interest calculated and added to balance.

1.Deposit
2.Withdraw
3.Calculate interest
4.Exit
Enter your choice : 4

Process finished with exit code 0
```


```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 8.py"
1. Savings Account
2. Current Account
Enter your choice : 1
Enter the initial balance : 800
Enter the interest rate for savings account : 0.6

1.Deposit
2.Withdraw
3.Calculate interest
4.Exit
Enter your choice : 2
Enter amount to withdraw : 9000
Insufficient balance
```

Current account


```
1.Deposit
2.Withdraw
3.Calculate interest
4.Exit
Enter your choice : 3
This option is available only for Savings Account

1.Deposit
2.Withdraw
3.Calculate interest
4.Exit
Enter your choice : 4

Process finished with exit code 0
```

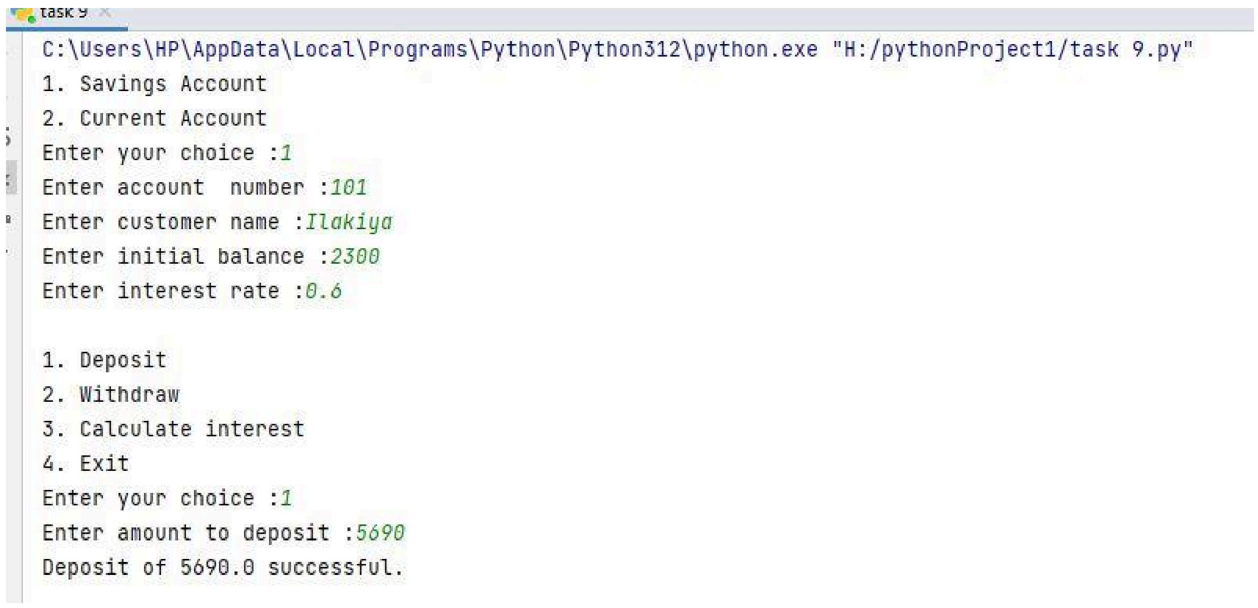
Task 9

Create an abstract class BankAccount that represents a generic bank account. It should include the following attributes and methods:
Attributes:
Account number.
Customer name.
Balance.
Constructors:
Implement default constructors and overload the constructor with Account attributes, generate getter and setter, print all information of attribute methods for the attributes.
Abstract methods:
deposit(amount: float): Deposit the specified amount into the account.
withdraw(amount: float): Withdraw the specified amount from the account (implement error handling for insufficient funds).
calculate_interest(): Abstract method for calculating interest.
Create two concrete classes that inherit from BankAccount:
SavingsAccount: A savings account that includes an additional attribute for interest rate. Implement the calculate_interest() method to calculate interest based on the balance and interest rate.
CurrentAccount: A current account with no interest. Implement the withdraw() method to allow overdraft up to a certain limit (configure a constant for the overdraft limit).
Create a Bank class to represent the banking system. Perform the following operation in main method:
Display menu for user to create object for account class by calling parameter
constructor. Menu should display options `SavingsAccount` and `CurrentAccount`. user can choose any one option to create account. use switch case for implementation.
create_account should display sub menu to choose type of accounts.
Hint: Account acc = new SavingsAccount(); or Account acc = new CurrentAccount();
deposit(amount: float): Deposit the specified amount into the account.
withdraw(amount: float): Withdraw the specified amount from the account. For saving
account withdraw amount only if there is sufficient fund else display insufficient balance.
For Current Account withdraw limit can exceed the available balance and should not exceed the overdraft limit.

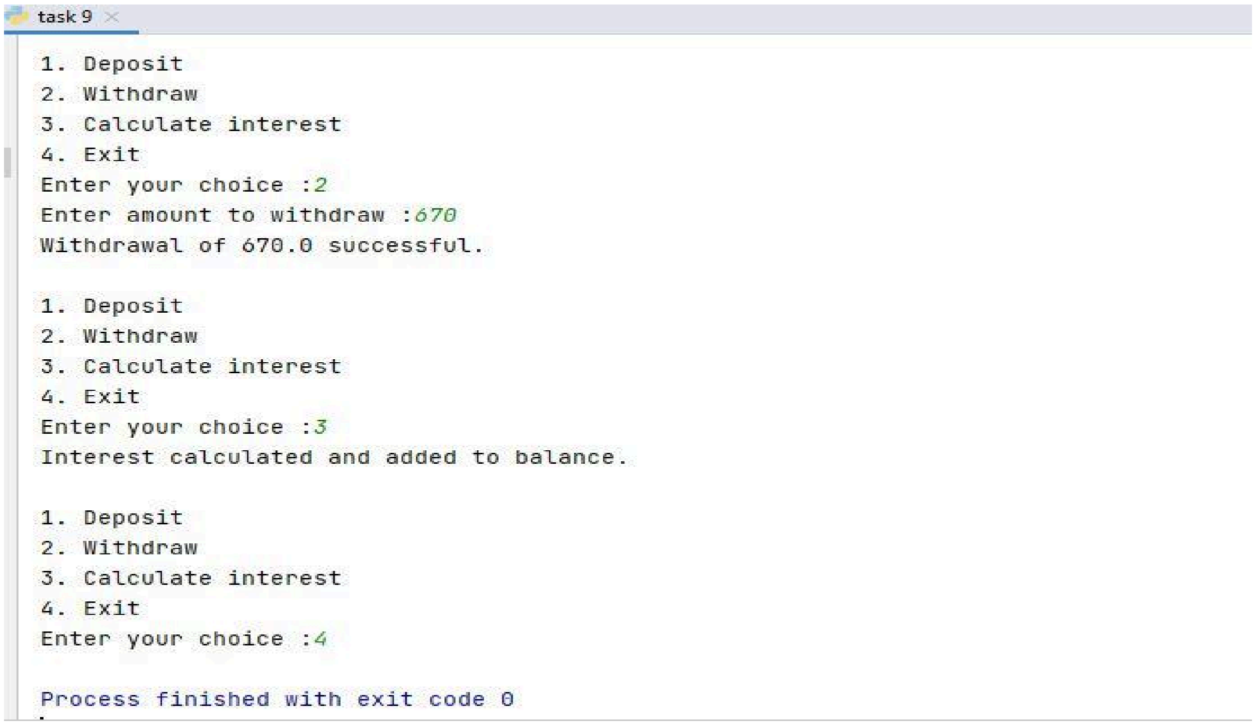
calculate_interest(): Calculate and add interest to the account balance for savings accounts.

```python
from abc import ABC,abstractmethod
class BankAccount(ABC):
   def __init__(self,account_number = "",customer_name = "",balance = 0.0):
       self._account_number = account_number
       self._customer_name = customer_name
       self._balance = balance
   @property
   def account_number(self):
       return self._account_number
   @account_number.setter
   def account_number(self,value):
       self._account_number = value
   @property
   def customer_name(self):
       return self._customer_name
   @customer_name.setter
   def customer_name(self,value):
       self._customer_name = value
   @property
   def balance(self):
       return self._balance
   @balance.setter
   def balance(self,value):
       self._balance = value
   def print_info(self):
       print("Account number :",self._account_number)
       print("Customer name :",self._customer_name)
       print("Balance :",self._balance)
   @abstractmethod
   def deposit(self,amount):
       pass
   @abstractmethod
   def withdraw(self,amount):
       pass
   @abstractmethod
   def calculate_interest(self):
       pass
class SavingsAccount(BankAccount):
       def  __init__(self,account_number  =  "",customer_name  =  "",balance  =
0.0,interest_rate = 0.0):
       super().__init__(account_number, customer_name, balance)
       self._interest_rate = interest_rate
   @property
   def interest_rate(self):
       return self._interest_rate
   @interest_rate.setter
   def interest_rate(self,value):
       self._interest_rate = value
   def deposit(self,amount):
       self.balance += amount
       print("Deposit of",amount,"successful.")
   def withdraw(self,amount):
       if self.balance >= amount:
           self.balance -= amount
           print("Withdrawal of",amount,"successful.")
       else:
           print("Insufficient balance.")
   def calculate_interest(self):
       interest = self.balance * self._interest_rate / 100
       self.balance += interest
       print("Interest calculated and added to balance.")
class CurrentAccount(BankAccount):
   OVERDRAFT_LIMIT = 1000
   def deposit(self,amount):
       self.balance += amount
       print(f"Deposit of {amount} successful.")
   def withdraw(self,amount):
       if self.balance + self.OVERDRAFT_LIMIT >= amount:
           self.balance -= amount
           print(f"Withdrawal of {amount} successful.")
       else:
           print("Withdrawal amount exceeds overdraft limit.")
   def calculate_interest(self):
       print("Current account does not earn interest.")
class Bank:
   @staticmethod
   def display_menu():
       print("1. Savings Account")
       print("2. Current Account")
       choice = input("Enter your choice :")
       return choice
   @staticmethod
   def create_account():
       choice = Bank.display_menu()
       if choice == "1":
           account_number = input("Enter account  number :")
           customer_name = input("Enter customer name :")
           balance = float(input("Enter initial balance :"))
           interest_rate = float(input("Enter interest rate :"))
                 return  SavingsAccount(account_number,  customer_name,  balance,
interest_rate)
       elif choice == "2":
           account_number = input("Enter account  number :")
           customer_name = input("Enter customer name :")
           balance = float(input("Enter initial balance :"))
           return CurrentAccount(account_number, customer_name, balance)
       else:
           print("Invalid choice")
           return None
   @staticmethod
   def perform_operations(account):
       while True:
           print("\n1. Deposit")
           print("2. Withdraw")
           print("3. Calculate interest")
           print("4. Exit")
           option = input("Enter your choice :")
           if option == "1":
               amount = float(input("Enter amount to deposit :"))
               account.deposit(amount)
           elif option == "2":
               amount = float(input("Enter amount to withdraw :"))
               account.withdraw(amount)
           elif option == "3":
               account.calculate_interest()
           elif option == "4":
               break
           else:
               print("Invalid choice")
if __name__ == '__main__':
   account = Bank.create_account()
   if account:
       Bank.perform_operations(account)
```

Savings account



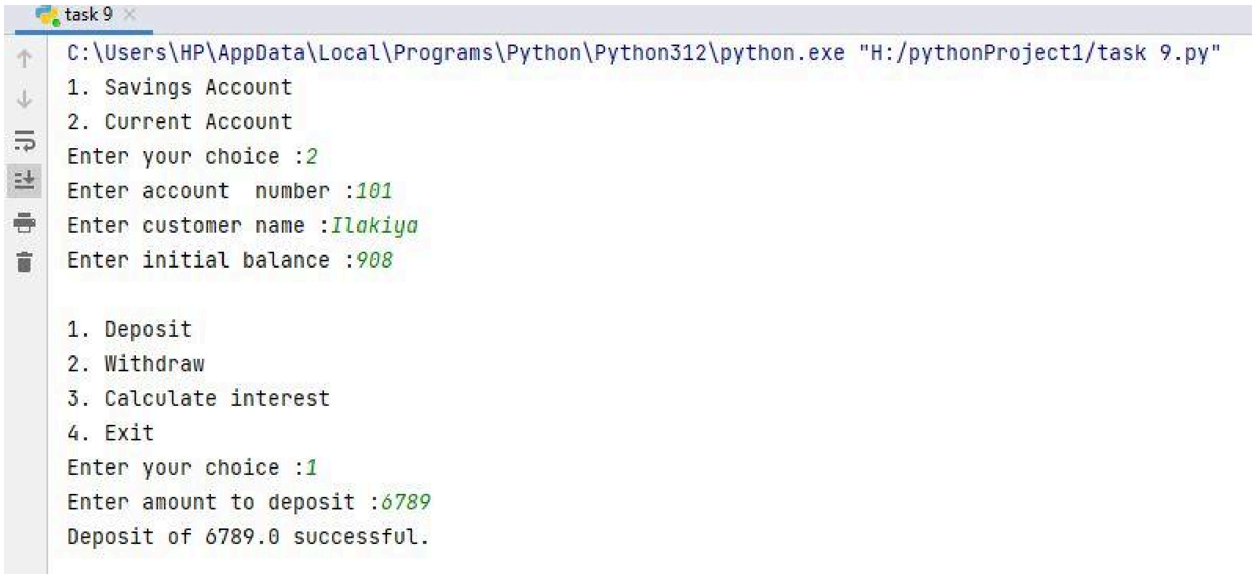
```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 9.py"
1. Savings Account
2. Current Account
Enter your choice :1
Enter account  number :101
Enter customer name :Ilakiya
Enter initial balance :2300
Enter interest rate :0.6

1. Deposit
2. Withdraw
3. Calculate interest
4. Exit
Enter your choice :1
Enter amount to deposit :5690
Deposit of 5690.0 successful.
```



task 9

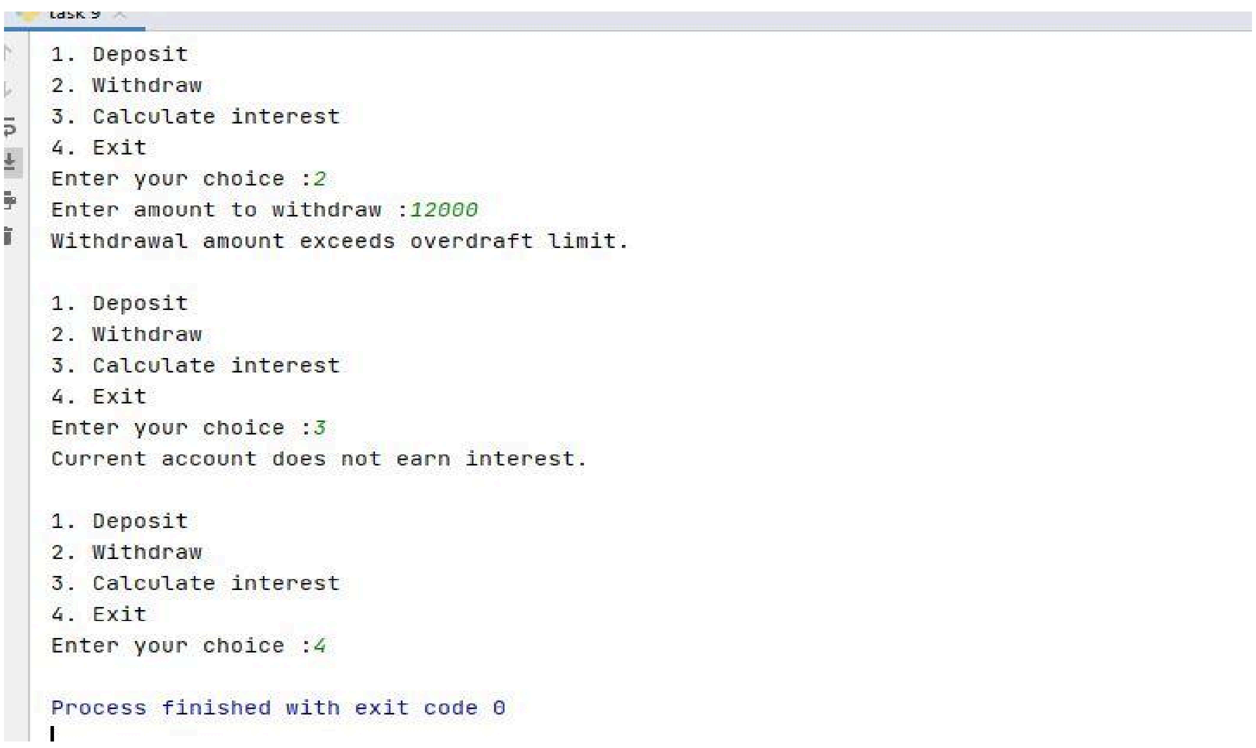
```
1. Deposit
2. Withdraw
3. Calculate interest
4. Exit
Enter your choice :2
Enter amount to withdraw :670
Withdrawal of 670.0 successful.

1. Deposit
2. Withdraw
3. Calculate interest
4. Exit
Enter your choice :3
Interest calculated and added to balance.

1. Deposit
2. Withdraw
3. Calculate interest
4. Exit
Enter your choice :4

Process finished with exit code 0
```

Current account



task 9

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 9.py"
1. Savings Account
2. Current Account
Enter your choice :2
Enter account  number :101
Enter customer name :Ilakiya
Enter initial balance :908

1. Deposit
2. Withdraw
3. Calculate interest
4. Exit
Enter your choice :1
Enter amount to deposit :6789
Deposit of 6789.0 successful.
```



```
1. Deposit
2. Withdraw
3. Calculate interest
4. Exit
Enter your choice :2
Enter amount to withdraw :12000
Withdrawal amount exceeds overdraft limit.

1. Deposit
2. Withdraw
3. Calculate interest
4. Exit
Enter your choice :3
Current account does not earn interest.

1. Deposit
2. Withdraw
3. Calculate interest
4. Exit
Enter your choice :4

Process finished with exit code 0
```

Task 10

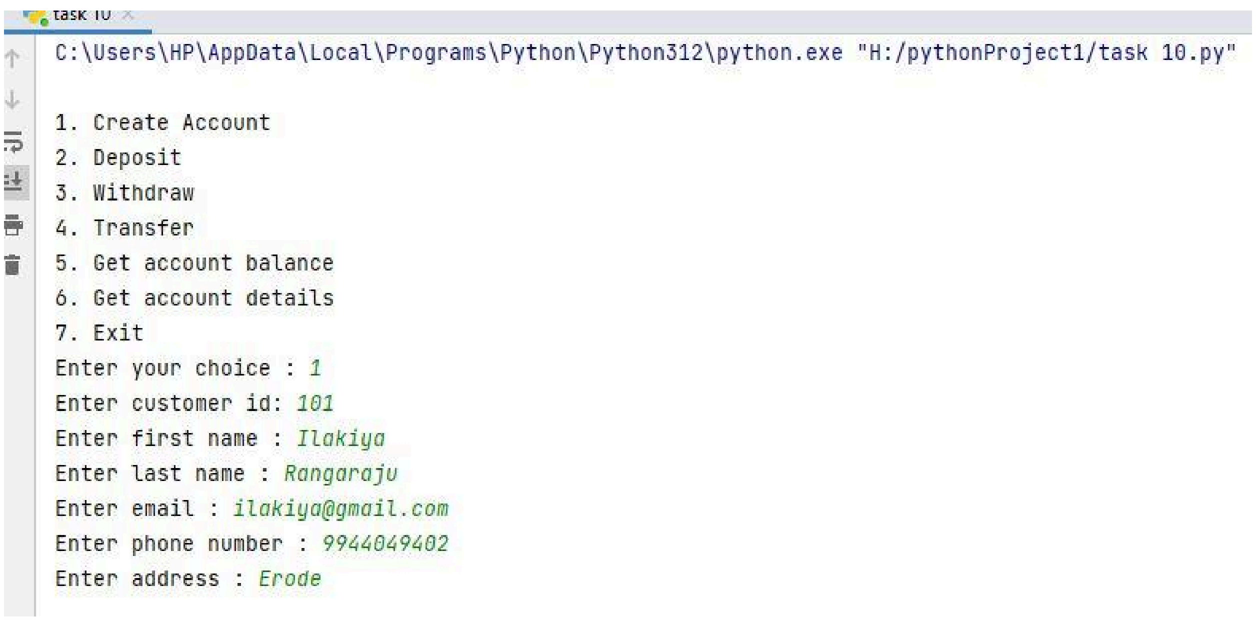
Create a `Customer` class with the following attributes:
Customer ID
First Name
Last Name
Email Address (validate with valid email address)
Phone Number (Validate 10-digit phone number)
Address
Methods and Constructor:
Implement default constructors and overload the constructor with Account attributes, generate getter, setter, print all information of attribute) methods for the attributes.
Create an `Account` class with the following attributes:
Account Number (a unique identifier).
Account Type (e.g., Savings, Current)
Account Balance
Customer (the customer who owns the account)
Methods and Constructor:
Implement default constructors and overload the constructor with Account attributes, generate getter, setter, (print all information of attribute) methods for the attributes.

Create a Bank Class and must have following requirements:
Create a Bank class to represent the banking system. It should have the following methods:
create_account(Customer customer, long accNo, String accType, float balance): Create a new bank account for the given customer with the initial balance.
get_account_balance(account_number: long): Retrieve the balance of an account given its account number. should return the current balance of account.
deposit(account_number: long, amount: float): Deposit the specified amount into the account. Should return the current balance of account.
withdraw(account_number: long, amount: float): Withdraw the specified amount from the account. Should return the current balance of account.
transfer(from_account_number: long, to_account_number: int, amount: float): Transfer money from one account to another.
getAccountDetails(account_number: long): Should return the account and customer details.
Ensure that account numbers are automatically generated when an account is created, starting from 1001 and incrementing for each new account.

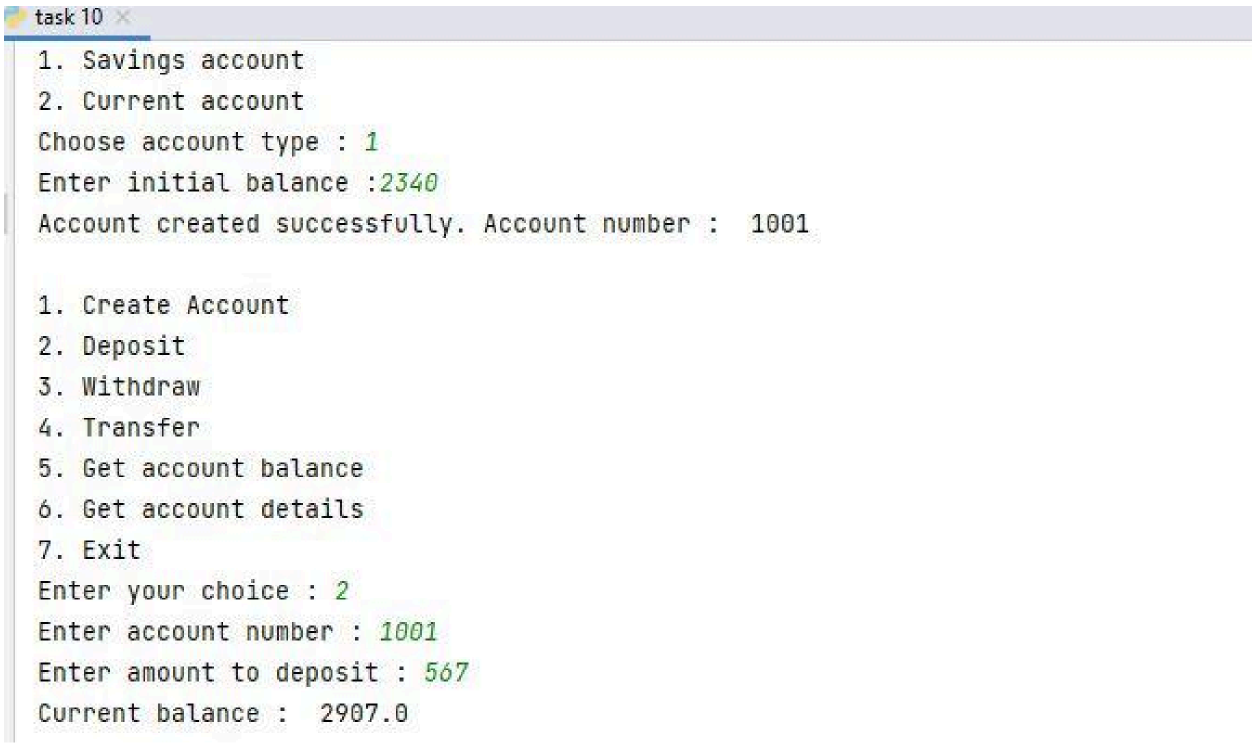
Create a BankApp class with a main method to simulate the banking system. Allow the user to interact with the system by entering commands such as "create_account", "deposit", "withdraw", "get_balance", "transfer", "getAccountDetails" and "exit." create_account should display sub menu to choose type of accounts and repeat this operation until user exit.

```
import re
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
   @property
   def customer_id(self):
       return self._customer_id
   @customer_id.setter
   def customer_id(self,value):
       self._customer_id = value
   @property
   def first_name(self):
       return self._first_name
   @first_name.setter
   def first_name(self, value):
       self._first_name = value
   @property
```

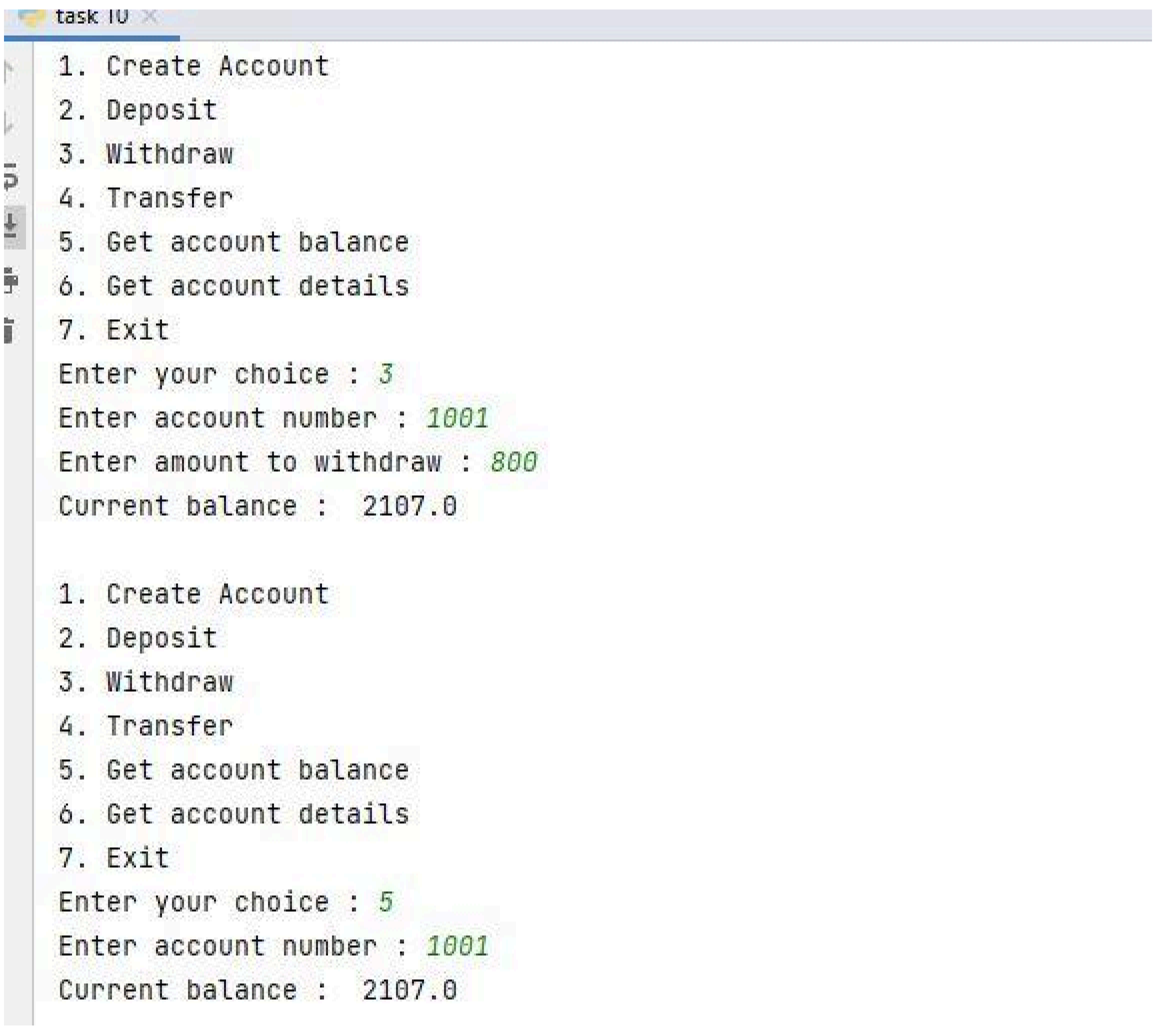
```
    def last_name(self):
        return self._last_name
    @last_name.setter
    def last_name(self,value):
        self._last_name = value
    @property
    def email(self):
        return self._email
    @email.setter
    def email(self,value):
        if re.match(r"[^@]+@[^@]+\.[^@]+",value):
            self._email = value
        else:
            print("Invalid email address.")
    @property
    def phone_number(self):
        return self._phone_number
    @phone_number.setter
    def phone_number(self,value):
        if re.match(r"^\d{10}$",value):
            self._phone_number = value
        else:
            print("Invalid phone number.")
    @property
    def address(self):
        return self._address
    @address.setter
    def address(self,value):
        self._address = value
    def print_info(self):
        print("Customer id : ",self._customer_id)
        print("First name : ",self._first_name)
        print("Last name : ",self._last_name)
        print("Email : ",self._email)
        print("Phone number : ",self._phone_number)
        print("Address : ",self._address)
class Account:
    account_counter = 1000
    def __init__(self,account_type = "",balance = 0.0,customer = None):
        self._account_number = Account.generate_account_number()
        self._account_type = account_type
        self._balance = balance
        self._customer = customer
    @staticmethod
    def generate_account_number():
        Account.account_counter += 1
        return Account.account_counter
    @property
    def account_number(self):
        return self._account_number
    @property
    def account_type(self):
        return self._account_type
    @account_type.setter
    def account_type(self,value):
        self._account_type = value
    @property
    def balance(self):
        return self._balance
    @balance.setter
    def balance(self,value):
        self._balance = value
    @property
    def customer(self):
        return self._customer
    @customer.setter
    def customer(self,value):
        self._customer = value
    def print_info(self):
        print("Account number : ",self._account_number)
        print("Account type : ",self._account_type)
        print("Balance : ",self._balance)
        print("Customer Details")
        self._customer.print_info()
class Bank:
    accounts = {}
    @staticmethod
    def create_account(customer,acc_type,balance):
        account = Account(acc_type,balance,customer)
        Bank.accounts[account.account_number] = account
                  print("Account   created   successfully.   Account   number   :
",account.account_number)
    @staticmethod
    def get_account_balance(account_number):
        if account_number in Bank.accounts:
            return Bank.accounts[account_number].balance
        else:
            print("Account not found.")
    @staticmethod
    def deposit(account_number,amount):
        if account_number in Bank.accounts:
            Bank.accounts[account_number].balance += amount
            return Bank.accounts[account_number].balance
        else:
            print("Account not found")
    @staticmethod
    def withdraw(account_number,amount):
        if account_number in Bank.accounts:
            if Bank.accounts[account_number].balance >= amount:
               Bank.accounts[account_number].balance -= amount
               return Bank.accounts[account_number].balance
            else:
                print("Insufficient balance")
        else:
            print("Account not found")
    @staticmethod
    def transfer(from_account_number,to_account_number,amount):
            if  from_account_number  in  Bank.accounts  and  to_account_number  in
Bank.accounts:
            if Bank.accounts[from_account_number].balance >= amount:
                Bank.accounts[from_account_number].balance -= amount
                Bank.accounts[to_account_number].balance += amount
                print("Transfer successful")
            else:
                print("Insufficient balance")
        else:
            print("One or both accounts not found")
    @staticmethod
    def get_account_details(account_number):
        if account_number in Bank.accounts:
            Bank.accounts[account_number].print_info()
        else:
            print("Account not found")
class BankApp:
    @staticmethod
    def main():
        while True:
            print("\n1. Create Account")
            print("2. Deposit")
            print("3. Withdraw")
            print("4. Transfer")
            print("5. Get account balance")
            print("6. Get account details")
            print("7. Exit")
            choice = input("Enter your choice : ")
            if choice == "1":
                customer = Customer()
                customer.customer_id = int(input("Enter customer id: "))
                customer.first_name = input("Enter first name : ")
                customer.last_name = input("Enter last name : ")
                customer.email = input("Enter email : ")
                customer.phone_number = input("Enter phone number : ")
                customer.address = input("Enter address : ")
                print("\n1. Savings account")
                print("2. Current account")
                account_type = input("Choose account type : ")
                balance = float(input("Enter initial balance :" ))
                Bank.create_account(customer,account_type,balance)
            elif choice == "2":
                account_number = int(input("Enter account number : "))
                amount = float(input("Enter amount to deposit : "))
                       print("Current balance  : ",Bank.deposit(account_number,
amount))
            elif choice == "3":
                account_number = int(input("Enter account number : "))
                amount = float(input("Enter amount to withdraw : "))
                      print("Current balance  : ",Bank.withdraw(account_number,
amount))
            elif choice == "4":
                      from_account_number  = int(input("Enter account number to
transfer from :"))
                to_account_number = int(input("Enter account number to transfer
to : "))
                amount = float(input("Enter amount to transfer : "))
                Bank.transfer(from_account_number, to_account_number, amount)
            elif choice == "5":
                account_number = int(input("Enter account number : "))
                                                       print("Current   balance   :
",Bank.get_account_balance(account_number))
            elif choice == "6":
                account_number = int(input("Enter account number : "))
                Bank.get_account_details(account_number)
            elif choice == "7":
                print("Exiting.....")
                break
            else:
                print("Invalid choice.")
if __name__ == "__main__":
   BankApp.main()
```



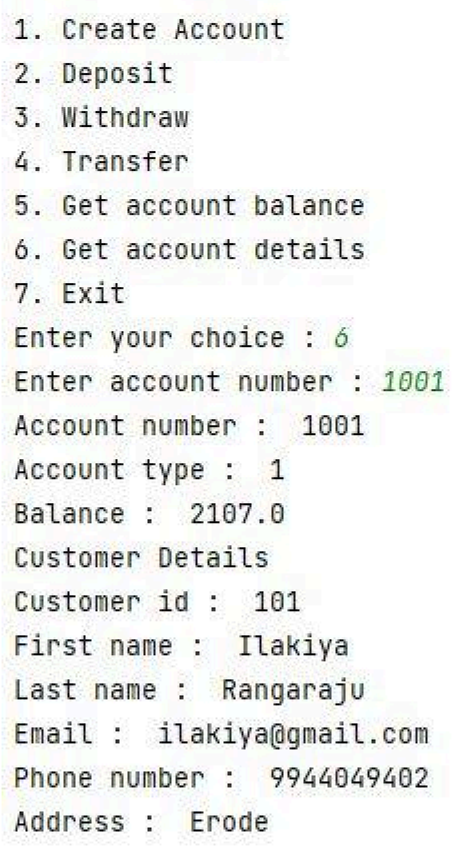
```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 10.py"

1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get account balance
6. Get account details
7. Exit
Enter your choice : 1
Enter customer id: 101
Enter first name : Ilakiya
Enter last name : Rangaraju
Enter email : ilakiya@gmail.com
Enter phone number : 9944049402
Enter address : Erode
```



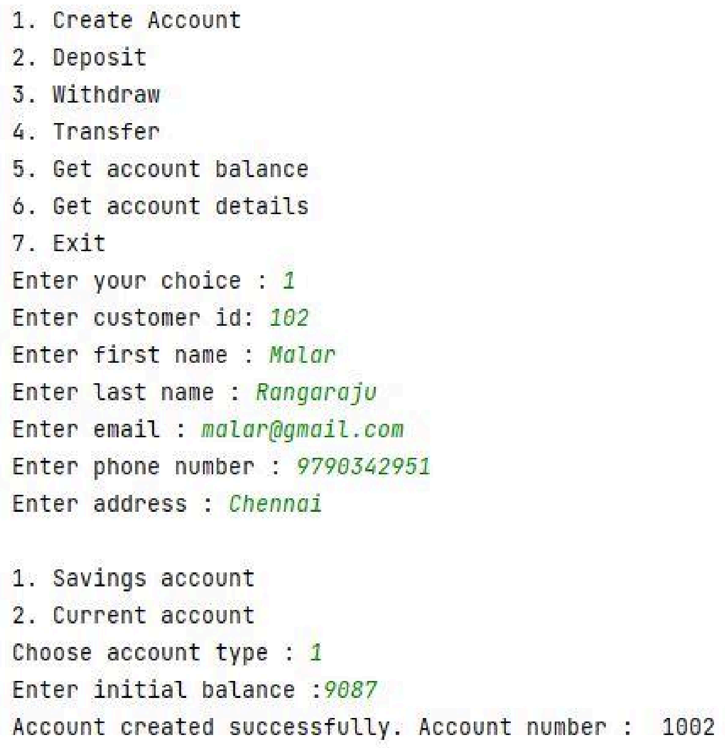
```
1. Savings account
2. Current account
Choose account type : 1
Enter initial balance :2340
Account created successfully. Account number :  1001

1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get account balance
6. Get account details
7. Exit
Enter your choice : 2
Enter account number : 1001
Enter amount to deposit : 567
Current balance :  2907.0
```



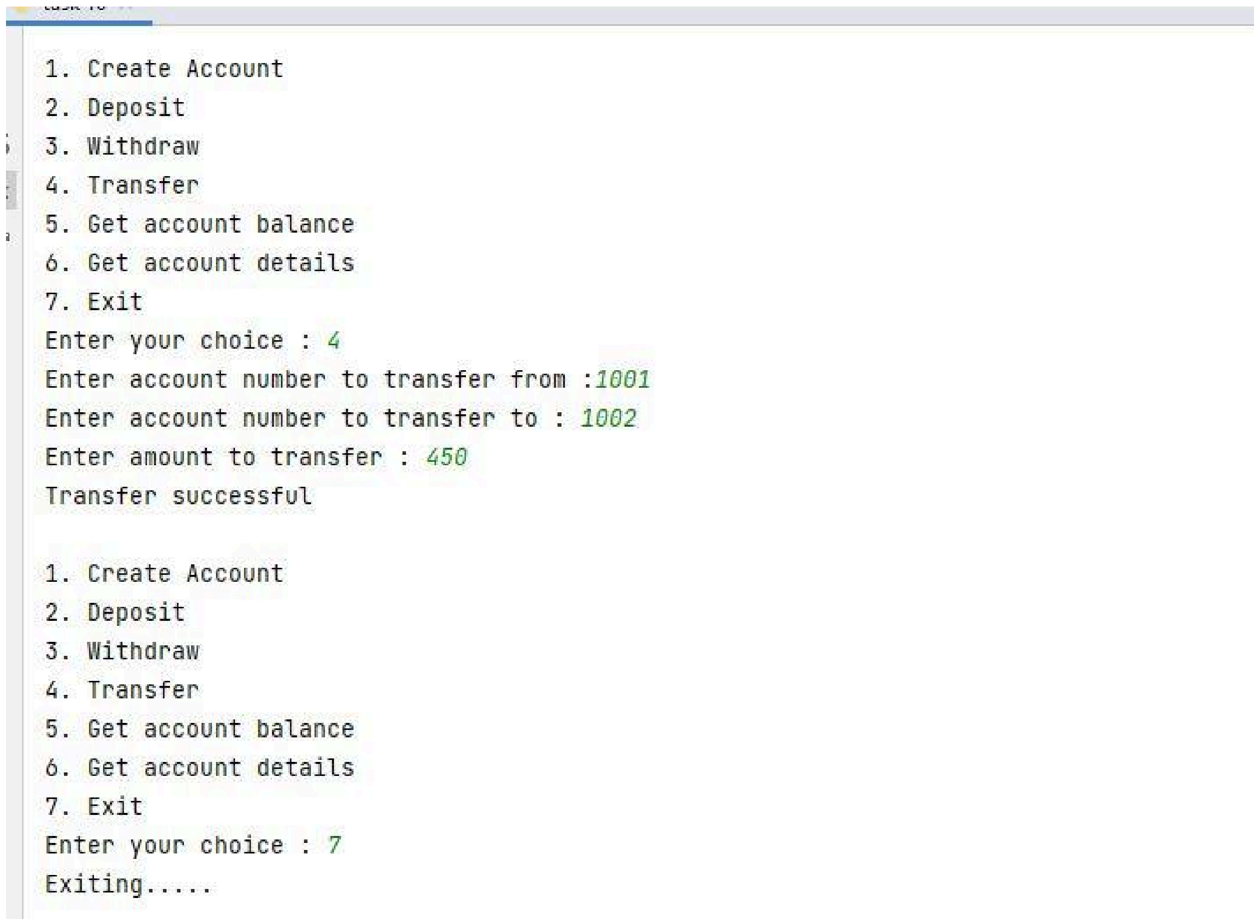
```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get account balance
6. Get account details
7. Exit
Enter your choice : 3
Enter account number : 1001
Enter amount to withdraw : 800
Current balance :  2107.0

1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get account balance
6. Get account details
7. Exit
Enter your choice : 5
Enter account number : 1001
Current balance :  2107.0
```



```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get account balance
6. Get account details
7. Exit
Enter your choice : 6
Enter account number : 1001
Account number :  1001
Account type :  1
Balance :  2107.0
Customer Details
Customer id :  101
First name :  Ilakiya
Last name :  Rangaraju
Email :  ilakiya@gmail.com
Phone number :  9944049402
Address :  Erode
```



```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get account balance
6. Get account details
7. Exit
Enter your choice : 1
Enter customer id: 102
Enter first name : Malar
Enter last name : Rangaraju
Enter email : malar@gmail.com
Enter phone number : 9790342951
Enter address : Chennai

1. Savings account
2. Current account
Choose account type : 1
Enter initial balance :9087
Account created successfully. Account number :  1002
```



```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get account balance
6. Get account details
7. Exit
Enter your choice : 4
Enter account number to transfer from :1001
Enter account number to transfer to : 1002
Enter amount to transfer : 450
Transfer successful

1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get account balance
6. Get account details
7. Exit
Enter your choice : 7
Exiting.....
```

Task 11

- Create a ‘Customer’ class as mentioned above task.
- Create an class ‘Account’ that includes the following attributes. Generate account number using static variable.
- Account Number (a unique identifier).
- Account Type (e.g., Savings, Current)
- Account Balance
- Customer (the customer who owns the account)
- lastAccNo

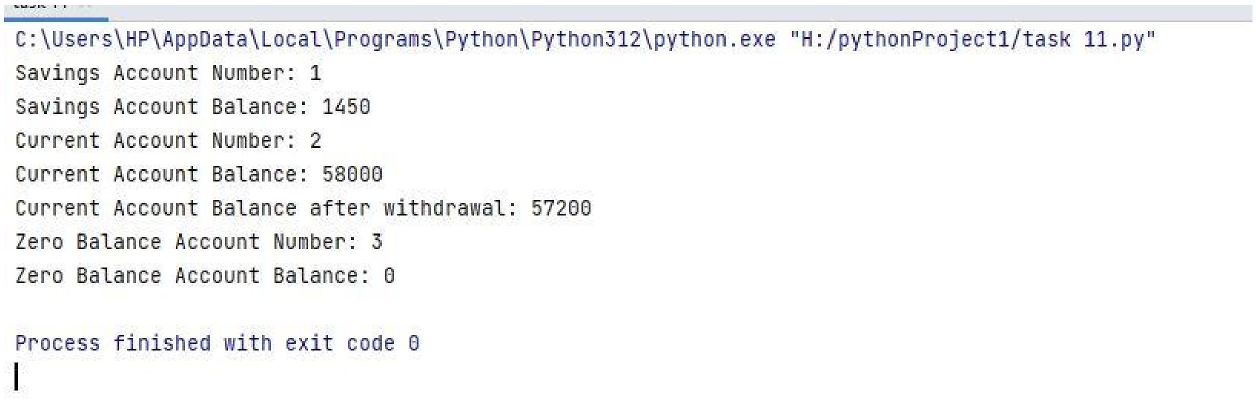
3. Create three child classes that inherit the Account class and each class must contain below mentioned attribute:

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

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

```
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_acc_no = 0

   def __init__(self, account_type, initial_balance, customer):
       Account.last_acc_no += 1
       self.account_number = Account.last_acc_no
       self.account_type = account_type
       self.balance = initial_balance
       self.customer = customer


class SavingsAccount(Account):
   def __init__(self, initial_balance, customer, interest_rate=0.08):
       super().__init__('Savings', initial_balance, customer)
       if initial_balance < 500:
           raise ValueError("Minimum balance for Savings Account must be 500")
       self.interest_rate = interest_rate

class CurrentAccount(Account):
   def __init__(self, initial_balance, customer, overdraft_limit=10000):
       super().__init__('Current', initial_balance, customer)
       self.overdraft_limit = overdraft_limit

   def withdraw(self, amount):
       if amount > self.balance + self.overdraft_limit:
             print("Withdrawal amount exceeds available balance and overdraft 
limit.")
       else:
           self.balance -= amount

class ZeroBalanceAccount(Account):
   def __init__(self, customer):
       super().__init__('Zero Balance', 0, customer)
```

```
customer1 = Customer(101, "Ilakiya", "Rangaraju", "ilakiya@gmail.com", "9852147632", "Erode")
savings_acc = SavingsAccount(1450, customer1.first_name)
print("Savings Account Number:", savings_acc.account_number)
print("Savings Account Balance:", savings_acc.balance)

customer2 = Customer(102, "Malar", "Rangaraju", "malar@gmail.com", "9760342951", "Chennai")
current_acc = CurrentAccount(58000, customer2.first_name)
print("Current Account Number:", current_acc.account_number)
print("Current Account Balance:", current_acc.balance)
current_acc.withdraw(800)
print("Current Account Balance after withdrawal:", current_acc.balance)

zero_balance_acc = ZeroBalanceAccount(customer2.first_name)
print("Zero Balance Account Number:", zero_balance_acc.account_number)
print("Zero Balance Account Balance:", zero_balance_acc.balance)
```



```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 11.py"
Savings Account Number: 1
Savings Account Balance: 1450
Current Account Number: 2
Current Account Balance: 58000
Current Account Balance after withdrawal: 57200
Zero Balance Account Number: 3
Zero Balance Account Balance: 0

Process finished with exit code 0
```

- Create ICustomerServiceProvider interface/abstract class with following functions:
- get_account_balance(account_number: long): Retrieve the balance of an account given its account number. should return the current balance of account.
- deposit(account_number: long, amount: float): Deposit the specified amount into the account. Should return the current balance of account.
- withdraw(account_number: long, amount: float): Withdraw the specified amount from the account. Should return the current balance of account. A savings account should maintain a minimum balance and checking if the withdrawal violates the minimum balance rule.
- transfer(from_account_number: long, to_account_number: int, amount: float): Transfer money from one account to another.
- getAccountDetails(account_number: long): Should return the account and customer details.

```
from abc import ABC, abstractmethod
```

```
class ICustomerServiceProvider(ABC):
   @abstractmethod
   def get_account_balance(self, account_number):
       pass

   @abstractmethod
   def deposit(self, account_number, amount):
       pass

   @abstractmethod
   def withdraw(self, account_number, amount):
       pass

   @abstractmethod
   def transfer(self, from_account_number, to_account_number, amount):
       pass

   @abstractmethod
   def get_account_details(self, account_number):
       pass
```

- Create IBankServiceProvider interface/abstract class with following functions:
- create_account(Customer customer, long accNo, String accType, float balance): Create a new bank account for the given customer with the initial balance.
- listAccounts():Account[] accounts: List all accounts in the bank.
- calculateInterest(): the calculate_interest() method to calculate interest based on the balance and interest rate.

```
from abc import ABC, abstractmethod

class IBankServiceProvider(ABC):
   @abstractmethod
   def create_account(self, customer, accNo, accType, balance):
       pass

   @abstractmethod
   def list_accounts(self):
       pass

   @abstractmethod
   def calculate_interest(self):
       pass
```

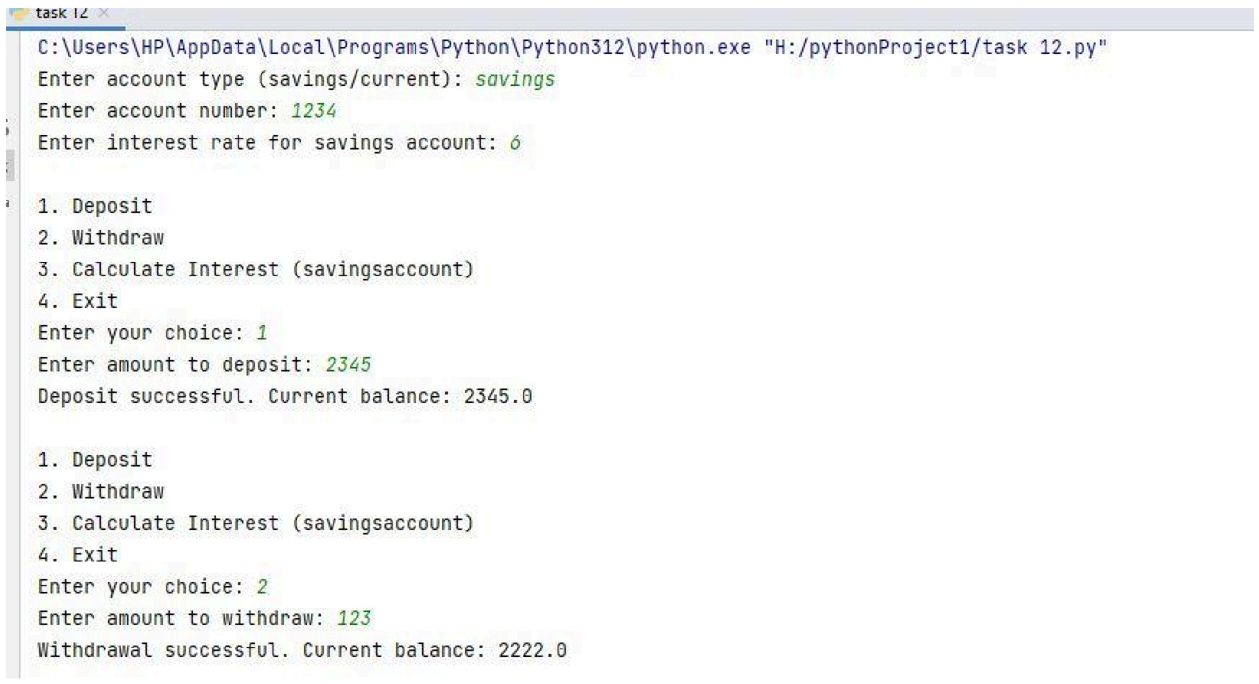
Task 12

throw the exception whenever needed and Handle in main method,

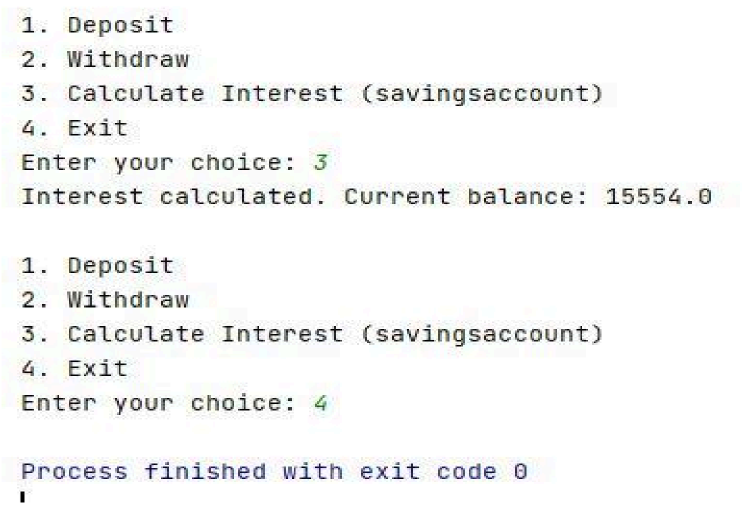
- InsufficientFundException throw this exception when user try to withdraw amount or transfer amount to another account and the account runs out of money in the account.
- InvalidAccountException throw this exception when user entered the invalid account number when tries to transfer amount, get account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

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

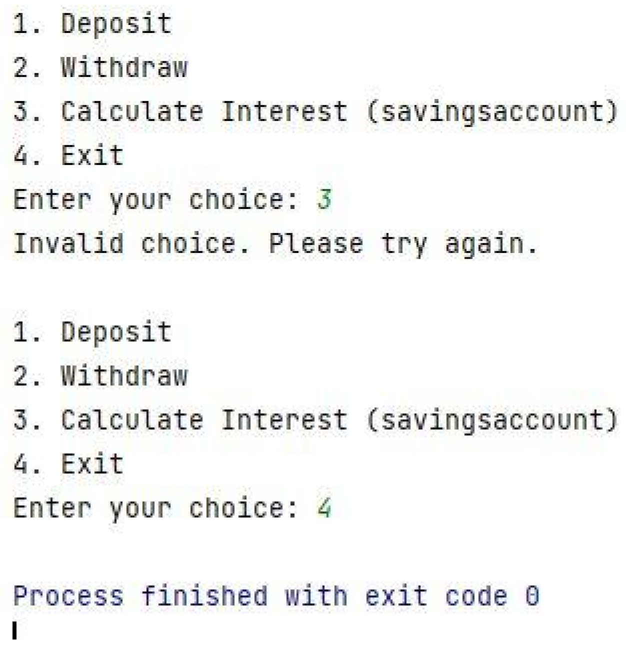
```
class InsufficientFundException(Exception):
   pass


class InvalidAccountException(Exception):
   pass


class OverDraftLimitExceededException(Exception):
   pass


class Account:
           def    __init__(self,    account_type,    account_number,    balance=0,
overdraft_limit=0):
       self.account_type = account_type
       self.account_number = account_number
       self.balance = balance
       self.overdraft_limit = overdraft_limit

   def deposit(self, amount):
       self.balance += amount

   def withdraw(self, amount):
       if self.account_type == "savings":
           if self.balance < amount:
                   raise InsufficientFundException("Insufficient balance in the
account.")
           else:
               self.balance -= amount
       elif self.account_type == "current":
           if amount > (self.balance + self.overdraft_limit):
               raise OverDraftLimitExceededException("Withdrawal amount exceeds
the overdraft limit.")
           else:
               self.balance -= amount

   def calculate_interest(self, interest_rate):
       if self.account_type == "savings":
           interest = self.balance * interest_rate
           self.balance += interest


def main():
   try:
       account_type = input("Enter account type (savings/current): ")
       account_number = int(input("Enter account number: "))
       if account_type not in ["savings", "current"]:
           raise InvalidAccountException("Invalid account type.")

       if account_type == "savings":
                 interest_rate = float(input("Enter interest rate for savings
account: "))
           account = Account(account_type, account_number)
       elif account_type == "current":
              overdraft_limit = float(input("Enter overdraft limit for current
account: "))
                            account = Account(account_type, account_number,
overdraft_limit=overdraft_limit)

       while True:
           print("\n1. Deposit")
           print("2. Withdraw")
           print("3. Calculate Interest (savingsaccount)")
           print("4. Exit")

           choice = int(input("Enter your choice: "))

           if choice == 1:
               amount = float(input("Enter amount to deposit: "))
               account.deposit(amount)
               print("Deposit successful. Current balance:", account.balance)
           elif choice == 2:
               amount = float(input("Enter amount to withdraw: "))
               account.withdraw(amount)
                            print("Withdrawal successful. Current balance:",
account.balance)
           elif choice == 3 and account_type == "savings":
               account.calculate_interest(interest_rate)
               print("Interest calculated. Current balance:", account.balance)
           elif choice == 4:
               break
           else:
               print("Invalid choice. Please try again.")
   except InsufficientFundException as e:
       print("Error:", e)
   except InvalidAccountException as e:
       print("Error:", e)
   except OverDraftLimitExceededException as e:
       print("Error:", e)
   except ValueError:
       print("Invalid input. Please enter a valid number.")
   except Exception as e:
       print("An error occurred:", e)


main()
```



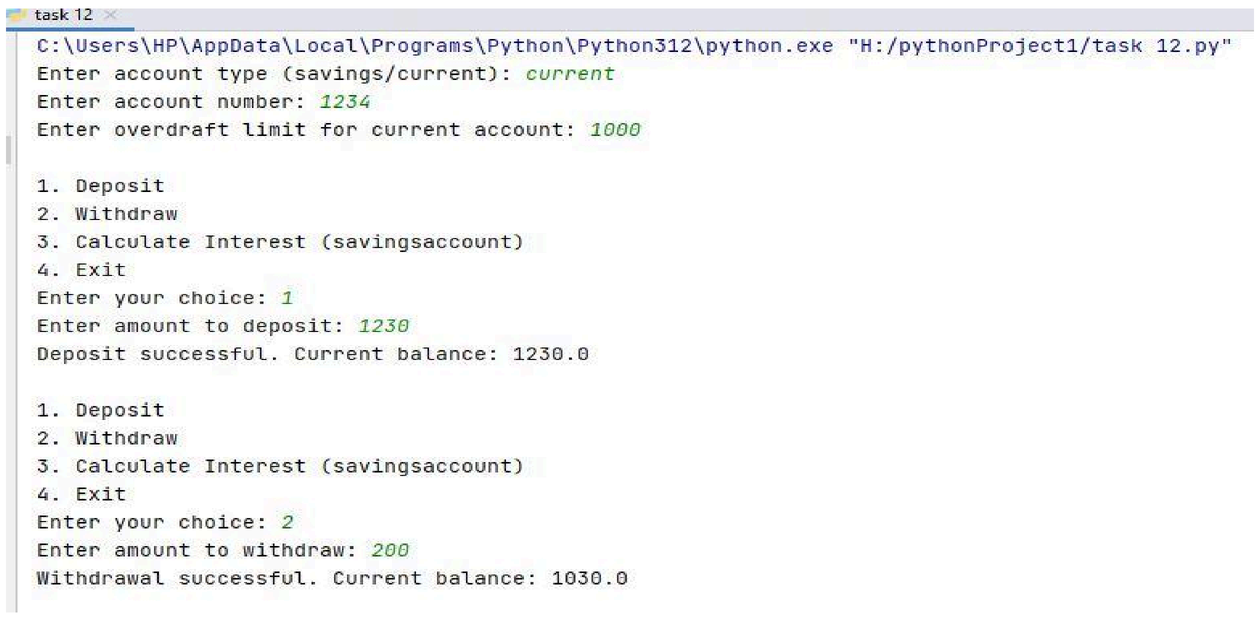
```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 12.py"
Enter account type (savings/current): savings
Enter account number: 1234
Enter interest rate for savings account: 6

1. Deposit
2. Withdraw
3. Calculate Interest (savingsaccount)
4. Exit
Enter your choice: 1
Enter amount to deposit: 2345
Deposit successful. Current balance: 2345.0

1. Deposit
2. Withdraw
3. Calculate Interest (savingsaccount)
4. Exit
Enter your choice: 2
Enter amount to withdraw: 123
Withdrawal successful. Current balance: 2222.0
```



```
1. Deposit
2. Withdraw
3. Calculate Interest (savingsaccount)
4. Exit
Enter your choice: 3
Interest calculated. Current balance: 15554.0

1. Deposit
2. Withdraw
3. Calculate Interest (savingsaccount)
4. Exit
Enter your choice: 4

Process finished with exit code 0
```



```
1. Deposit
2. Withdraw
3. Calculate Interest (savingsaccount)
4. Exit
Enter your choice: 3
Invalid choice. Please try again.

1. Deposit
2. Withdraw
3. Calculate Interest (savingsaccount)
4. Exit
Enter your choice: 4

Process finished with exit code 0
```



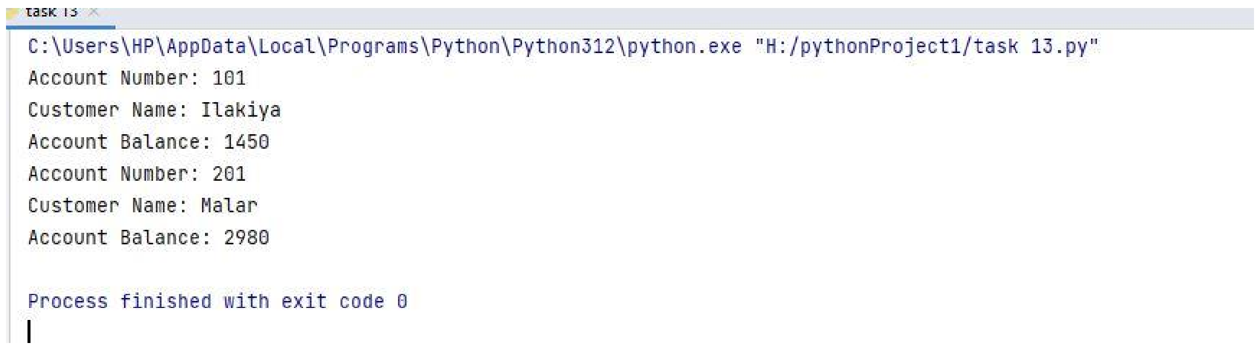
```
task 12
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 12.py"
Enter account type (savings/current): current
Enter account number: 1234
Enter overdraft limit for current account: 1000

1. Deposit
2. Withdraw
3. Calculate Interest (savingsaccount)
4. Exit
Enter your choice: 1
Enter amount to deposit: 1230
Deposit successful. Current balance: 1230.0

1. Deposit
2. Withdraw
3. Calculate Interest (savingsaccount)
4. Exit
Enter your choice: 2
Enter amount to withdraw: 200
Withdrawal successful. Current balance: 1030.0
```

Task 13

- From the previous task change the HMBank attribute Accounts to List of Accounts and perform the same operation.

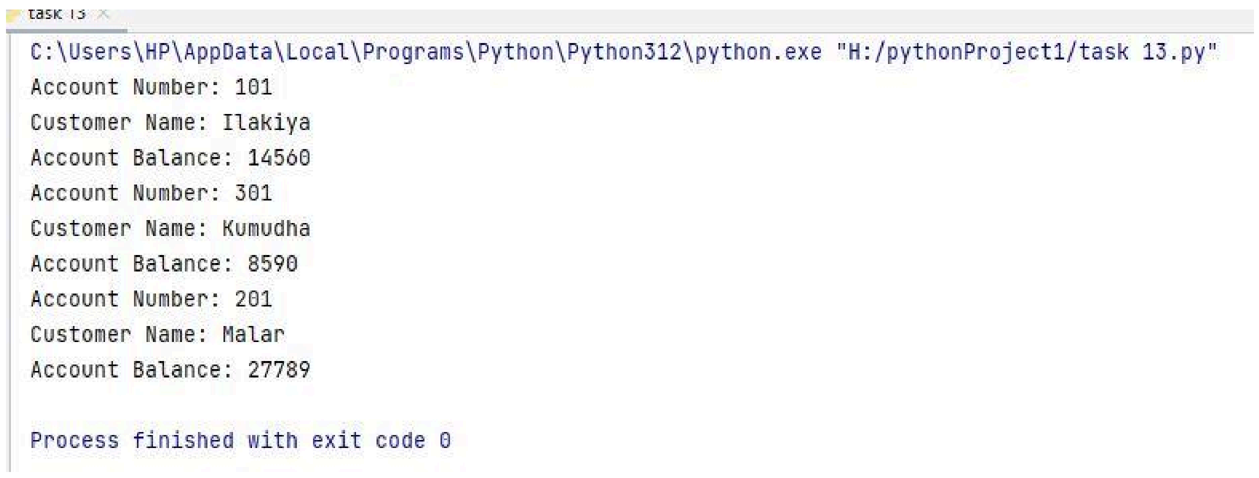
```
class BankAccount:
    def __init__(self, account_number, customer_name, balance):
        self.account_number = account_number
        self.customer_name = customer_name
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print("Balance After withdrawal: ", self.balance)
        else:
            print("Insufficient balance")

    def interest(self):
        intrate = float(input("Enter the interest rate: "))
```

```
            intamount = self.balance * (intrate / 100)
            self.balance += intamount
            print("Balance with interest: ", self.balance)

    def display(self):
        print("Account Number:", self.account_number)
        print("Customer Name:", self.customer_name)
        print("Account Balance:", self.balance)


class Bank:
    def __init__(self):
        self.accounts = []

    def add_account(self, account):
        self.accounts.append(account)

    def list_accounts(self):
        self.accounts.sort(key=lambda acc: acc.customer_name)
        for account in self.accounts:
            account.display()


bank = Bank()
acc1 = BankAccount(101, "Ilakiya", 1450)
acc2 = BankAccount(201, "Malar", 2980)
bank.add_account(acc1)
bank.add_account(acc2)
bank.list_accounts()
```


```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 13.py"
Account Number: 101
Customer Name: Ilakiya
Account Balance: 1450
Account Number: 201
Customer Name: Malar
Account Balance: 2980

Process finished with exit code 0
```

- From the previous task change the HMBank attribute Accounts to Set of Accounts and perform the same operation.
- Avoid adding duplicate Account object to the set.
- Create Comparator<Account> object to sort the accounts based on customer name when listAccounts() method called.

```python
class BankAccount:
   def __init__(self, account_number, customer_name, balance):
       self.account_number = account_number
       self.customer_name = customer_name
       self.balance = balance

   def deposit(self, amount):
       self.balance += amount

   def withdraw(self, amount):
       if amount <= self.balance:
           self.balance -= amount
           print("Balance After withdrawal: ", self.balance)
       else:
           print("Insufficient balance")

   def interest(self):
       intrate = float(input("Enter the interest rate: "))
       intamount = self.balance * (intrate / 100)
       self.balance += intamount
       print("Balance with interest: ", self.balance)

   def display(self):
       print("Account Number:", self.account_number)
       print("Customer Name:", self.customer_name)
       print("Account Balance:", self.balance)


class Bank:
   def __init__(self):
       self.accounts = {}

   def add_account(self, account):
       self.accounts[account.account_number] = account

   def list_accounts(self):
             sorted_accounts  =  sorted(self.accounts.values(),  key=lambda  acc:
acc.customer_name)
       for account in sorted_accounts:
           account.display()


bank = Bank()
acc1 = BankAccount(101, "Ilakiya", 14560)
acc2 = BankAccount(201, "Malar", 27789)
acc3 = BankAccount(301, "Kumudha", 8590)
bank.add_account(acc1)
bank.add_account(acc2)
bank.add_account(acc3)
bank.list_accounts()
```

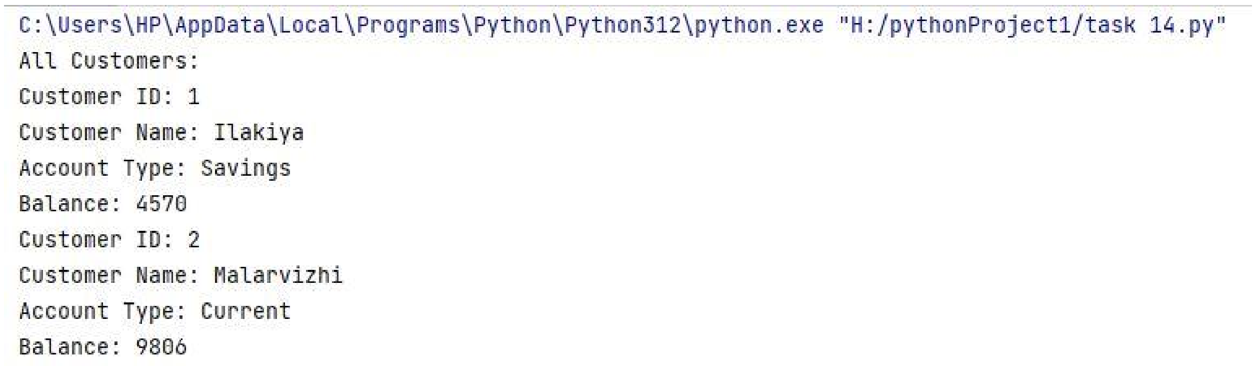


```
task 13
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 13.py"
Account Number: 101
Customer Name: Ilakiya
Account Balance: 14560
Account Number: 301
Customer Name: Kumudha
Account Balance: 8590
Account Number: 201
Customer Name: Malar
Account Balance: 27789

Process finished with exit code 0
```

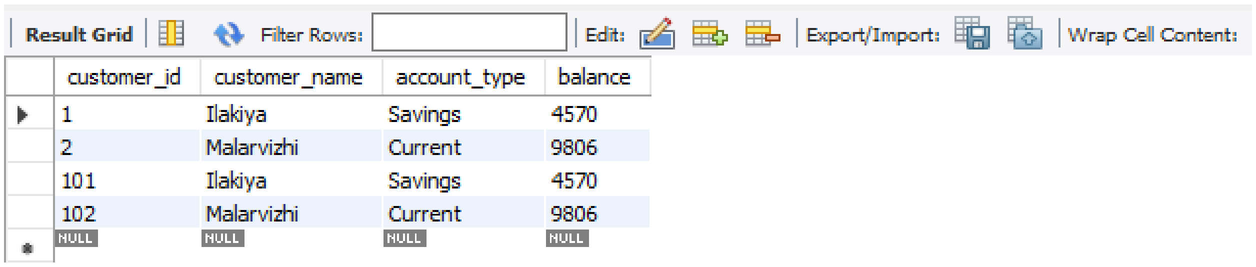
Task 14

• Create a 'Customer' class as mentioned above task.

```
import mysql.connector
class Customer:
   def __init__(self, customer_id, customer_name, account_type, balance):
       self.customer_id = customer_id
       self.customer_name = customer_name
       self.account_type = account_type
       self.balance = balance
   def display(self):
       print("Customer ID:", self.customer_id)
       print("Customer Name:", self.customer_name)
       print("Account Type:", self.account_type)
       print("Balance:", self.balance)


class Database:
   def __init__(self,db_name):
       self.connection = mysql.connector.connect(

host="localhost",user="root",password="root",port="3306",database="emp"
       )
       self.cursor = self.connection.cursor()
       self.cursor.execute('''CREATE TABLE IF NOT EXISTS customer
                                    (customer_id int PRIMARY KEY,
                                   customer_name text,
                                   account_type text,
                                   balance int)''')
       self.connection.commit()

   def add_customer(self, customer):
         query="INSERT INTO customer(customer_id, customer_name, account_type,
balance) VALUES (%s, %s, %s, %s)"
       self.cursor.execute(query,
                                     (customer.customer_id, customer.customer_name,
customer.account_type, customer.balance))
       self.connection.commit()

   def display_all_customers(self):
       self.cursor.execute('''SELECT * FROM customer''')
       rows = self.cursor.fetchall()
       for row in rows:
           cust = Customer(row[0], row[1], row[2], row[3])
           cust.display()


   def close(self):
       self.connection.close()

db = Database("customers")

cust1 = Customer(101, "Ilakiya", "Savings", 4570)
db.add_customer(cust1)

cust2 = Customer(102, "Malarvizhi", "Current", 9806)
db.add_customer(cust2)

print("All Customers:")
db.display_all_customers()

db.close()
```



```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 14.py"
All Customers:
Customer ID: 1
Customer Name: Ilakiya
Account Type: Savings
Balance: 4570
Customer ID: 2
Customer Name: Malarvizhi
Account Type: Current
Balance: 9806
```

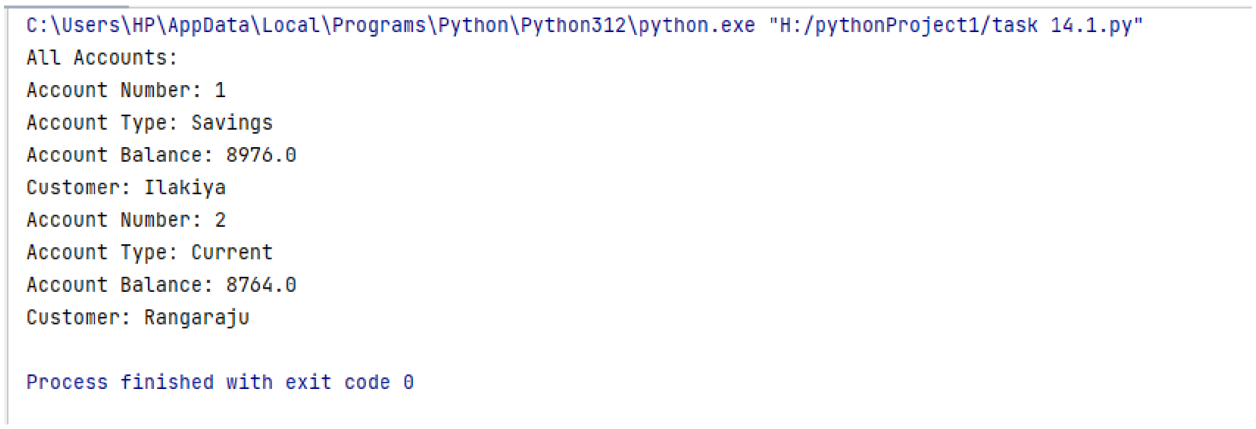

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

customer_id	customer_name	account_type	balance
1	Ilakiya	Savings	4570
2	Malarvizhi	Current	9806
101	Ilakiya	Savings	4570
102	Malarvizhi	Current	9806
NULL	NULL	NULL	NULL

- Create an class ‘Account’ that includes the following attributes. Generate account number using static variable.
- Account Number (a unique identifier).
- Account Type (e.g., Savings, Current)
- Account Balance
- Customer (the customer who owns the account)
- lastAccNo

```
import mysql.connector


class Account:
   lastAccNo = 0

   def __init__(self, acc_type, balance, customer):
       Account.lastAccNo += 1
       self.acc_no = Account.lastAccNo
       self.acc_type = acc_type
       self.balance = balance
       self.customer = customer

   def display(self):
       print("Account Number:", self.acc_no)
       print("Account Type:", self.acc_type)
       print("Account Balance:", self.balance)
       print("Customer:", self.customer)


class Database:
   def __init__(self, db_name):
       self.connection = mysql.connector.connect(
           host="localhost",
           user="root",
           password="root",
           port="3306",
           database= "emp"
           )
       self.cursor = self.connection.cursor()
```

```
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS accounts
                            (acc_no INTEGER PRIMARY KEY,
                            acc_type TEXT,
                            balance REAL,
                            customer TEXT)''')
        self.connection.commit()

    def add_account(self, account):
         self.cursor.execute('''INSERT INTO accounts(acc_no, acc_type, balance,
customer)VALUES       (%s,%s,%s,%s)''',(account.acc_no,       account.acc_type,
account.balance, account.customer))
        self.connection.commit()

    def display_all_accounts(self):
        self.cursor.execute('''SELECT * FROM accounts''')
        rows = self.cursor.fetchall()
        for row in rows:
            acc = Account(row[1], row[2], row[3])
            acc.acc_no = row[0]
            acc.display()

    def close(self):
        self.connection.close()

db = Database("customers")

acc1 = Account("Savings",8976, "Ilakiya")
db.add_account(acc1)

acc2 = Account("Current", 8764, "Rangaraju")
db.add_account(acc2)

print("All Accounts:")
db.display_all_accounts()

db.close()
```

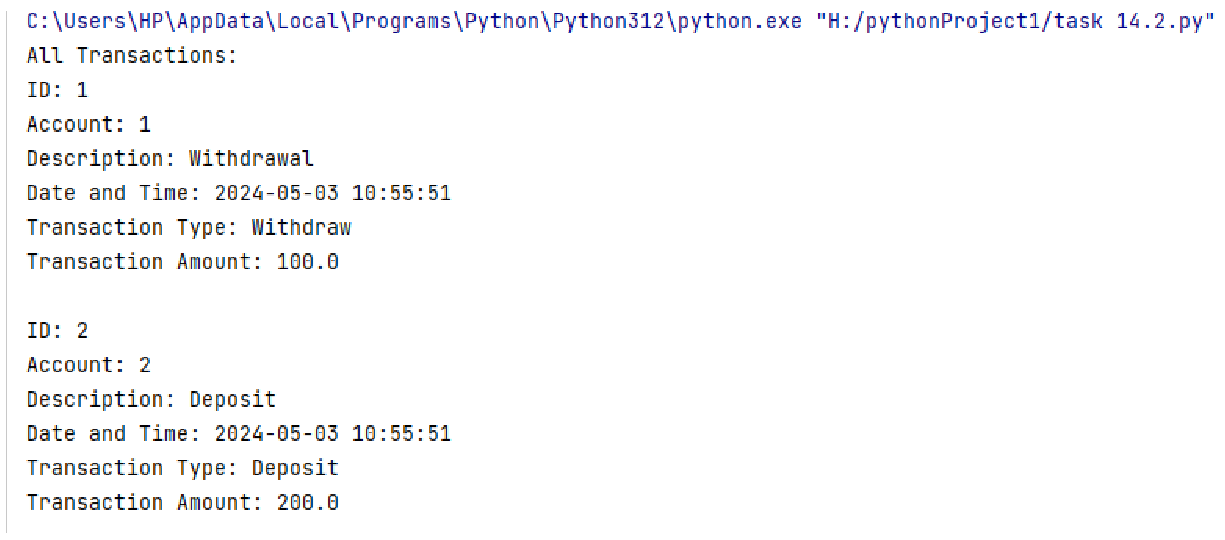


```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 14.1.py"
All Accounts:
Account Number: 1
Account Type: Savings
Account Balance: 8976.0
Customer: Ilakiya
Account Number: 2
Account Type: Current
Account Balance: 8764.0
Customer: Rangaraju

Process finished with exit code 0
```

- Create a class ‘TRANSACTION’ that include following attributes
- Account
- Description
- Date and Time
- TransactionType(Withdraw, Deposit, Transfer)
- TransactionAmount

```
import mysql.connector
from datetime import datetime


class Transaction:
           def    __init__(self,    account,    description,    transaction_type,
transaction_amount):
       self.account = account
       self.description = description
       self.date_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
       self.transaction_type = transaction_type
       self.transaction_amount = transaction_amount


class Database:
   def __init__(self, host, user, password,port, database):
       self.connection = mysql.connector.connect(
           host=host,
           user=user,
           password=password,
           port=port,
           database=database
       )
       self.cursor = self.connection.cursor()
       self.cursor.execute('''CREATE TABLE IF NOT EXISTS transactions
                            (id INT AUTO_INCREMENT PRIMARY KEY,
                            account INT,
                            description VARCHAR(255),
                            date_time DATETIME,
                            transaction_type VARCHAR(50),
                            transaction_amount FLOAT)''')
       self.connection.commit()

   def add_transaction(self, transaction):
         query = '''INSERT INTO transactions(account, description, date_time,
transaction_type, transaction_amount)
                  VALUES (%s, %s, %s, %s, %s)'''
                   values   =   (transaction.account,   transaction.description,
transaction.date_time, transaction.transaction_type,
                 transaction.transaction_amount)
       self.cursor.execute(query, values)
```

```
        self.connection.commit()

    def display_all_transactions(self):
        self.cursor.execute('''SELECT * FROM transactions''')
        rows = self.cursor.fetchall()
        for row in rows:
            print("ID:", row[0])
            print("Account:", row[1])
            print("Description:", row[2])
            print("Date and Time:", row[3])
            print("Transaction Type:", row[4])
            print("Transaction Amount:", row[5])
            print()

    def close(self):
        self.connection.close()


# Example Usage
db = Database(host="localhost", user="root", password="root",port="3306",
database="emp")

# Adding transactions
transaction1 = Transaction(account=1, description="Withdrawal",
transaction_type="Withdraw", transaction_amount=100)
db.add_transaction(transaction1)

transaction2 = Transaction(account=2, description="Deposit",
transaction_type="Deposit", transaction_amount=200)
db.add_transaction(transaction2)

# Displaying all transactions
print("All Transactions:")
db.display_all_transactions()

db.close()
```



```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 14.2.py"
All Transactions:
ID: 1
Account: 1
Description: Withdrawal
Date and Time: 2024-05-03 10:55:51
Transaction Type: Withdraw
Transaction Amount: 100.0

ID: 2
Account: 2
Description: Deposit
Date and Time: 2024-05-03 10:55:51
Transaction Type: Deposit
Transaction Amount: 200.0
```

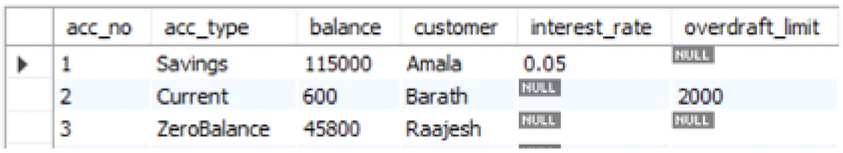
- Create three child classes that inherit the Account class and each class must contain below mentioned attribute:
- SavingsAccount: A savings account that includes an additional attribute for interest rate. Saving account should be created with minimum balance 500.
- CurrentAccount: A Current account that includes an additional attribute for overdraftLimit(credit limit).
- ZeroBalanceAccount: ZeroBalanceAccount can be created with Zero balance.

```python
import mysql.connector


class Account:

   def __init__(self, acc_type, balance, customer):
       self.acc_type = acc_type
       self.balance = balance
       self.customer = customer

   def display(self):
       print("Account Number:", self.acc_no)
       print("Account Type:", self.acc_type)
       print("Account Balance:", self.balance)
       print("Customer:", self.customer)


class SavingsAccount(Account):
   def __init__(self, balance, customer, interest_rate):
       super().__init__("Savings", balance, customer)
       self.interest_rate = interest_rate
       if balance < 500:
            raise ValueError("Minimum balance for a savings account is 500")


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


class ZeroBalanceAccount(Account):
   def __init__(self, customer):
       super().__init__("ZeroBalance", 0, customer)


class Database:
   def __init__(self, db_name):
       self.connection = mysql.connector.connect(
           host="localhost",
           user="root",
           password="root",
           port="3306",
           database="emp")
       self.cursor = self.connection.cursor()
       self.cursor.execute('''CREATE TABLE IF NOT EXISTS accounts
                            (acc_no INTEGER PRIMARY KEY AUTO_INCREMENT,
                            acc_type TEXT,
                            balance REAL,
                            customer TEXT,
                            interest_rate REAL,
                            overdraft_limit REAL)''')
       self.connection.commit()

   def add_account(self, account):
       if isinstance(account, SavingsAccount):
               self.cursor.execute('''INSERT INTO accounts( acc_type, balance,
customer,interest_rate)
                                VALUES ( %s, %s, %s,%s)''',
                                             (account.acc_type, account.balance,
account.customer, account.interest_rate))
       elif isinstance(account, CurrentAccount):
               self.cursor.execute('''INSERT INTO accounts( acc_type, balance,
customer,overdraft_limit)
                                VALUES ( %s, %s, %s,%s)''',
                                             (account.acc_type, account.balance,
account.customer, account.overdraft_limit))
       else:
               self.cursor.execute('''INSERT INTO accounts( acc_type, balance,
customer)
                                VALUES ( %s, %s, %s)''',
                                                       (account.acc_type,  account.balance,
account.customer))
       self.connection.commit()

   def display_all_accounts(self):
       self.cursor.execute('''SELECT * FROM accounts''')
       rows = self.cursor.fetchall()
       for row in rows:
           print(row)
           print(row[1])
           if row[1] == 'Savings':
               acc = SavingsAccount(row[2], row[3], row[4])
           elif row[1] == 'Current':
               acc = CurrentAccount(row[2], row[3], row[5])
           else:
               acc = ZeroBalanceAccount(row[3])
           acc.acc_no = row[0]
           acc.display()

   def close(self):
       self.connection.close()


db = Database("emp")

# Adding accounts
savings_acc        =        SavingsAccount(balance=789,        customer="Ankitha",
interest_rate=0.5)
db.add_account(savings_acc)

current_acc         =         CurrentAccount(balance=1890,         customer="Bala",
overdraft_limit=2000)
db.add_account(current_acc)

zero_balance_acc = ZeroBalanceAccount(customer="Miki")
db.add_account(zero_balance_acc)

current_acc        =        CurrentAccount(balance=1478,        customer="Pravin",
overdraft_limit=10000)
db.add_account(current_acc)

savings_acc        =       SavingsAccount(balance=67890,       customer="Vaishu",
interest_rate=0.08)
db.add_account(savings_acc)

print("All Accounts:")
db.display_all_accounts()

db.close()
```

- Create ICustomerServiceProvider interface/abstract class with following functions:
- get_account_balance(account_number: long): Retrieve the balance of an account given its account number. should return the current balance of account.
- deposit(account_number: long, amount: float): Deposit the specified amount into the account. Should return the current balance of account.
- withdraw(account_number: long, amount: float): Withdraw the specified amount from the account. Should return the current balance of account.
- A savings account should maintain a minimum balance and checking if the withdrawal violates the minimum balance rule.
- Current account customers are allowed withdraw overdraftLimit and available account balance. withdraw limit can exceed the available balance and should not exceed the overdraft limit.
- transfer(from_account_number: long, to_account_number: int, amount: float): Transfer money from one account to another. both account number should be validate from the database use getAccountDetails method.
- getAccountDetails(account_number: long): Should return the account and customer details.
- getTransations(account_number: long, FromDate:Date, ToDate: Date): Should return the list of transaction between two dates.

```
class ICustomerServiceProvider(ABC):
   @abstractmethod
   def get_account_balance(self, account_number):
       pass

   @abstractmethod
   def deposit(self, account_number, amount):
       pass

   @abstractmethod
   def withdraw(self, account_number, amount):
       pass

   @abstractmethod
   def transfer(self, from_account_number: int, to_account_number, amount):
       pass

   @abstractmethod
   def get_account_details(self, account_number):
       pass


class CustomerServiceProvider(ICustomerServiceProvider):
   def __init__(self, host, user, password, port, database):
       self.connection = mysql.connector.connect(
           host=host,
           user=user,
           password=password,
           port=port,
           database=database
       )
       self.cursor = self.connection.cursor()

   def display_all_accounts(self):
       self.cursor.execute('''SELECT * FROM accounts''')
       all_accounts = self.cursor.fetchall()
       if all_accounts:
           print("All Accounts Details:")
           for account in all_accounts:
               column_names = [i[0] for i in self.cursor.description]
               account_details = dict(zip(column_names, account))
               print(account_details)
       else:
           print("No accounts found in the database.")

   def get_account_balance(self, account_number):
         self.cursor.execute("SELECT balance FROM accounts WHERE acc_no = %s",
(account_number,))
       balance = self.cursor.fetchone()
       if balance:
           return balance[0]
           print(f"The balance of {account_number} is {balance}")
       else:
            raise ValueError(f"Account with account number {account_number} not
found.")

   def deposit(self, account_number, amount):
       current_balance = self.get_account_balance(account_number)
       new_balance = current_balance + amount
        self.cursor.execute('''UPDATE accounts SET balance = %s WHERE acc_no =
%s''', (new_balance, account_number))
       self.connection.commit()

   def withdraw(self, account_number, amount):
       current_balance = self.get_account_balance(account_number)
         self.cursor.execute('''SELECT acc_type, overdraft_limit FROM accounts
WHERE acc_no = %s''', (account_number,))
       account_info = self.cursor.fetchone()
       if account_info:
           acc_type, overdraft_limit = account_info
           if acc_type == 'Savings':
               if current_balance - amount < 500:
                         raise ValueError("Withdrawal violates minimum balance
rule.")
```

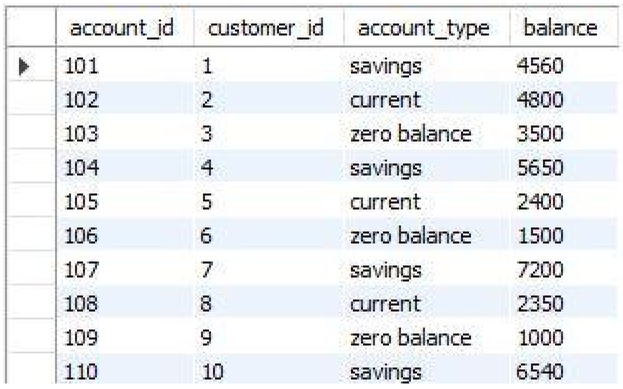
```
            elif acc_type == 'Current':
                available_balance = current_balance + overdraft_limit
                if amount > available_balance:
                      raise ValueError("Withdrawal exceeds available balance and
overdraft limit.")
        else:
             raise ValueError(f"Account with account number {account_number} not
found.")

        new_balance = current_balance - amount
         self.cursor.execute('''UPDATE accounts SET balance = %s WHERE acc_no =
%s''', (new_balance, account_number))
        self.connection.commit()

    def transfer(self, from_account_number, to_account_number, amount):
        self.withdraw(from_account_number, amount)
        self.deposit(to_account_number, amount)

    def get_account_details(self, account_number):
           self.cursor.execute('''SELECT * FROM accounts WHERE acc_no = %s''',
(account_number,))
        account_details = self.cursor.fetchone()
        if account_details:
            column_names = [i[0] for i in self.cursor.description]
            return dict(zip(column_names, account_details))
        else:
             raise ValueError(f"Account with account number {account_number} not
found.")

    def close_connection(self):
        self.connection.close()


db  =  CustomerServiceProvider(host="localhost",  user="root",  password="root",
port="3306",
                            database="emp")
db.get_account_balance(2)
db.deposit(4, 23000)
db.withdraw(4, 200)
db.get_account_details(4)
db.transfer(2, 4, 200)
db.display_all_accounts()
db.close_connection()
```



	acc_no	acc_type	balance	customer	interest_rate	overdraft_limit
▶	1	Savings	115000	Amala	0.05	NULL
	2	Current	600	Barath	NULL	2000
	3	ZeroBalance	45800	Raajesh	NULL	NULL

- Create IBankServiceProvider interface/abstract class with following functions:
- create_account(Customer customer, long accNo, String accType, float balance): Create a new bank account for the given customer with the initial balance.
- listAccounts(): Array of BankAccount: List all accounts in the bank.(List[Account] accountsList)
- getAccountDetails(account_number: long): Should return the account and customer details.
- calculateInterest(): the calculate_interest() method to calculate interest based on the balance and interest rate.

```
import mysql.connector
from abc import ABC, abstractmethod


class IBankServiceProvider(ABC):
   @abstractmethod
   def create_account(self, customer, acc_no, acc_type, balance):
       pass

   @abstractmethod
   def list_accounts(self):
       pass

   @abstractmethod
   def get_account_details(self, account_number):
       pass



class MySQLBankServiceProvider(IBankServiceProvider):
   def __init__(self, host, user, password, port, database):
       self.connection = mysql.connector.connect(
           host=host,
           user=user,
           password=password,
           port=port,
           database=database
       )
       self.cursor = self.connection.cursor()

   def create_account(self, customer,acc_no,acc_type, balance):
          query = "INSERT INTO accounts (customer,acc_no, acc_type, balance)
VALUES (%s, %s, %s,%s)"
       values = (customer, acc_no, acc_type, balance)
       self.cursor.execute(query, values)
       self.connection.commit()

   def list_accounts(self):
       self.cursor.execute("SELECT * FROM accounts")
       accounts = self.cursor.fetchall()
       return accounts

   def get_account_details(self, account_number):
       query = "SELECT * FROM accounts WHERE acc_no = %s"
       self.cursor.execute(query, (account_number,))
       account_details = self.cursor.fetchone()
       return account_details


   def close_connection(self):
       self.connection.close()



db  =  MySQLBankServiceProvider(host="localhost",  user="root",  password="root",
port="3306", database="hmbank")

# Create a new account
db.create_account("Ezhil",  185, "savings", 900.0)
db.create_account("Malar",  446, "current", 17890.0)
# List all accounts
accounts = db.list_accounts()
print("All accounts:", accounts)
print()
# Get account details
printing = db.get_account_details(123)
print()
print("Account details:", printing)

db.close_connection()
```

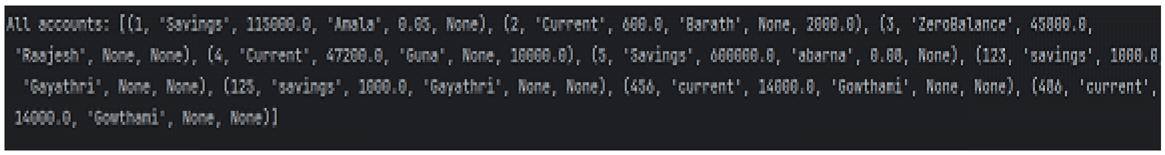


account_id	customer_id	account_type	balance
101	1	savings	4560
102	2	current	4800
103	3	zero balance	3500
104	4	savings	5650
105	5	current	2400
106	6	zero balance	1500
107	7	savings	7200
108	8	current	2350
109	9	zero balance	1000
110	10	savings	6540

- Create CustomerServiceProviderImpl class which implements ICustomerServiceProvider provide all implementation methods. These methods do not interact with database directly.

```
from sql_query_connection import Queryconnection
from abc import ABC, abstractmethod


class ICustomerServiceProvider(ABC):
    @abstractmethod
    def create_account(self, customer, acc_num, acc_type, balance):
        pass

    @abstractmethod
    def list_accounts(self):
        pass

    @abstractmethod
    def get_account_details(self, account_number):
        pass



class CustomerServiceProvider(ICustomerServiceProvider):
        db = Queryconnection(host="localhost", user="root", password="root", 
port="3306", database="customers")
    # Create a new account
    db.create_account("Ilakiya", 189, "savings", 1789.0)
    db.create_account("Malar", 446, "current", 1442256.0)
    # List all accounts
    accounts = db.list_accounts()
    print("All accounts:", accounts)
    print()
    # Get account details
    printing = db.get_account_details(125)
    print()
```

```
   print("Account details:", printing)

   db.close_connection()

#sql_quer_connection.py:
import mysql.connector
class Queryconnection:
   def __init__(self, host, user, password, port, database):
       self.connection = mysql.connector.connect(
           host=host,
           user=user,
           password=password,
           port=port,
           database=database
       )
       self.cursor = self.connection.cursor()

   def create_account(self, customer, acc_num, acc_type, balance):
            query  =  "INSERT  INTO  customerserviceprovider  (customer,acc_num,
acc_type, balance) VALUES (%s, %s, %s,%s)"
       values = (customer, acc_num, acc_type, balance)
       self.cursor.execute(query, values)
       self.connection.commit()

   def list_accounts(self):
       self.cursor.execute("SELECT * FROM customerserviceprovider")
       accounts = self.cursor.fetchall()
       return accounts

   def get_account_details(self, account_number):
       query = "SELECT * FROM customerserviceprovider WHERE acc_num = %s"
       self.cursor.execute(query, (account_number,))
       account_details = self.cursor.fetchone()
       return account_details

   def close_connection(self):
       self.connection.close()
```



```
All accounts: [(1, 'Savings', 113000.0, 'Amala', 0.05, None), (2, 'Current', 600.0, 'Barath', None, 2000.0), (3, 'ZeroBalance', 45000.0,
 'Raajesh', None, None), (4, 'Current', 47200.0, 'Guna', None, 10000.0), (5, 'Savings', 600000.0, 'abarna', 0.08, None), (123, 'savings', 1000.0
  'Gayathri', None, None), (123, 'savings', 1000.0, 'Gayathri', None, None), (456, 'current', 14000.0, 'Gowthami', None, None), (486, 'current',
 14000.0, 'Gowthami', None, None)]
```

.Create IBankRepository interface/abstract class which include following methods to interact with database.

- createAccount(customer: Customer, accNo: long, accType: String, balance: float): Create a new bank account for the given customer with the initial balance and store in database.
- listAccounts(): List<Account> accountsList: List all accounts in the bank from database.
- calculateInterest(): the calculate_interest() method to calculate interest based on the balance and interest rate.
- getAccountBalance(account_number: long): Retrieve the balance of an account given its account number. should return the current balance of account from database.
- deposit(account_number: long, amount: float): Deposit the specified amount into the account. Should update new balance in database and return the new balance.
- withdraw(account_number: long, amount: float): Withdraw amount should check the balance from account in database and new balance should updated in Database.
- A savings account should maintain a minimum balance and checking if the withdrawal violates the minimum balance rule.
- Current account customers are allowed withdraw overdraftLimit and available account balance. withdraw limit can exceed the available balance and should not exceed the overdraft limit.
- transfer(from_account_number: long, to_account_number: int, amount: float): Transfer money from one account to another. check the balance from account in database and new balance should updated in Database.
- getAccountDetails(account_number: long): Should return the account and customer details from databse.
- getTransations(account_number: long, FromDate:Date, ToDate: Date): Should return the list of transaction between two dates from database.

```
•   import mysql.connector
from abc import ABC, abstractmethod


class IBankRepository(ABC):
   @abstractmethod
   def get_account_balance(self, account_number):
       pass

   @abstractmethod
   def deposit(self, account_number, amount):
       pass

   @abstractmethod
   def withdraw(self, account_number, amount):
       pass

   @abstractmethod
   def transfer(self, from_account_number: int, to_account_number, amount):
       pass

   @abstractmethod
   def get_account_details(self, account_number):
       Pass
```

Create BankRepositoryImpl class which implement the IBankRepository interface/abstract class and provide implementation of all methods and perform the database operations.

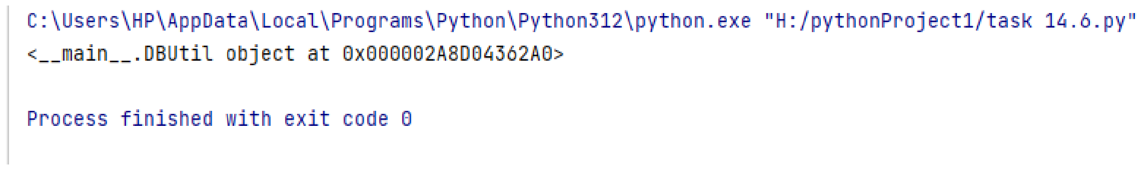
```
class IBankRepositoryImpl(IBankRepository):
    def __init__(self, host, user, password, port, database):
        self.connection = mysql.connector.connect(
            host=host,
            user=user,
            password=password,
            port=port,
            database=database
        )
        self.cursor = self.connection.cursor()

    def display_all_accounts(self):
        self.cursor.execute('''SELECT * FROM accounts''')
        all_accounts = self.cursor.fetchall()
        if all_accounts:
            print("All Accounts Details:")
            for account in all_accounts:
                column_names = [i[0] for i in self.cursor.description]
                account_details = dict(zip(column_names, account))
                print(account_details)
        else:
            print("No accounts found in the database.")

    def get_account_balance(self, account_number):
         self.cursor.execute("SELECT balance FROM accounts WHERE acc_no = %s",
(account_number,))
        balance = self.cursor.fetchone()
        if balance:
            return balance[0]
            print(f"The balance of {account_number} is {balance}")
        else:
            raise ValueError(f"Account with account number {account_number} not
found.")

    def deposit(self, account_number, amount):
        current_balance = self.get_account_balance(account_number)
        new_balance = current_balance + amount
         self.cursor.execute('''UPDATE accounts SET balance = %s WHERE acc_no =
%s''', (new_balance, account_number))
        self.connection.commit()

    def withdraw(self, account_number, amount):
        current_balance = self.get_account_balance(account_number)
         self.cursor.execute('''SELECT acc_type, overdraft_limit FROM accounts
WHERE acc_no = %s''', (account_number,))
        account_info = self.cursor.fetchone()
        if account_info:
            acc_type, overdraft_limit = account_info
            if acc_type == 'Savings':
                if current_balance - amount < 500:
                         raise ValueError("Withdrawal violates minimum balance
rule.")
            elif acc_type == 'Current':
                available_balance = current_balance + overdraft_limit
                if amount > available_balance:
                     raise ValueError("Withdrawal exceeds available balance and
overdraft limit.")
        else:
            raise ValueError(f"Account with account number {account_number} not
found.")

        new_balance = current_balance - amount
         self.cursor.execute('''UPDATE accounts SET balance = %s WHERE acc_no =
%s''', (new_balance, account_number))
        self.connection.commit()

    def transfer(self, from_account_number, to_account_number, amount):
        self.withdraw(from_account_number, amount)
        self.deposit(to_account_number, amount)

    def get_account_details(self, account_number):
          self.cursor.execute('''SELECT * FROM accounts WHERE acc_no = %s''',
(account_number,))
        account_details = self.cursor.fetchone()
        if account_details:
            column_names = [i[0] for i in self.cursor.description]
            print("ACCOUNT DETAILS")
            print(column_names, account_details)
        else:
            raise ValueError(f"Account with account number {account_number} not
found.")

    def close_connection(self):
        self.connection.close()
```

```
db   =   IBankRepositoryImpl(host="localhost",   user="root",   password="root",
port="3306",
                              database="hmbank")
db.get_account_balance(2)
db.get_account_details(4)
db.transfer(2, 4, 200)
db.display_all_accounts()
db.close_connection()
```

Create DBUtil class and add the following method.

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

```
•  import mysql.connector
class DBUtil:
   def getDBConn(self):
       con=mysql.connector.connect(
           host="localhost",
           user="root",
           password="root",
           port="3306",
           database="hmbank"
       )
       con.cursor()

obj=DBUtil()
print(obj)
```



```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 14.6.py"
<__main__.DBUtil object at 0x000002A8D04362A0>

Process finished with exit code 0
```

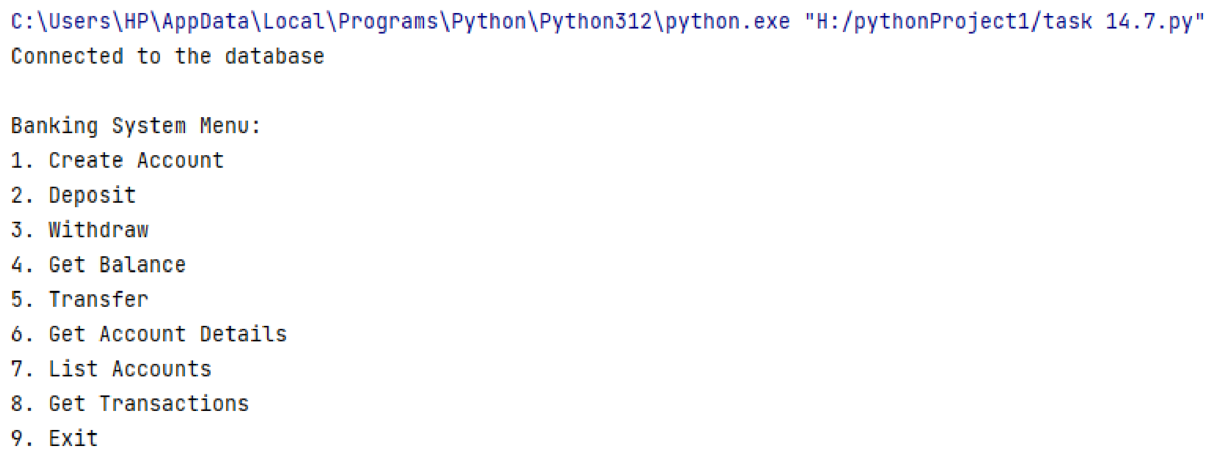
.Create BankApp class and perform following operation:
main method to simulate the banking system. Allow the user to interact with the system by entering choice from menu such as "create_account", "deposit", "withdraw",
"get_balance", "transfer", "getAccountDetails", "ListAccounts", "getTransactions" and "exit."

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

```
import mysql.connector
```

```
from mysql.connector import Error
from datetime import datetime


class BankApp:
   def __init__(self):
       self.connection = self.connect_to_database()

   def connect_to_database(self):
       try:
           connection = mysql.connector.connect(
               host="localhost",
               user="root",
               password="root",
               database="hmbank"
           )
           if connection.is_connected():
               print("Connected to the database")
               return connection
       except Error as e:
           print("Error while connecting to MySQL", e)

   def create_account(self):
       print("Creating a new account:")
       acc_num = input("Enter account number: ")
       acc_type = input("Enter account type (e.g., Savings, Current): ")
       balance = float(input("Enter initial balance: "))
       customer = input("Enter customer name: ")

       cursor = self.connection.cursor()
       try:
            query = "INSERT INTO accounts (acc_no, acc_type, balance,customer)
VALUES (%s, %s, %s,%s)"
           values = (acc_num, acc_type, balance, customer)
           cursor.execute(query, values)
           self.connection.commit()
           print("Account created successfully!")
       except Error as e:
           self.connection.rollback()
           print("Error while creating account:", e)

   def deposit(self):
       print("Depositing money:")
       account_number = input("Enter account number: ")
       amount = float(input("Enter amount to deposit: "))

       cursor = self.connection.cursor()
       try:
            query = "UPDATE accounts SET balance = balance + %s WHERE acc_no =
%s"
           values = (amount, account_number)
           cursor.execute(query, values)
           self.connection.commit()
           print("Deposit successful!")
       except Error as e:
           self.connection.rollback()
           print("Error while depositing money:", e)

   def withdraw(self):
       print("Withdrawing money:")
       account_number = input("Enter account number: ")
       amount = float(input("Enter amount to withdraw: "))

       cursor = self.connection.cursor()
       try:
            query = "UPDATE accounts SET balance = balance - %s WHERE acc_no =
%s AND balance >= %s"
           values = (amount, account_number, amount)
           cursor.execute(query, values)
           if cursor.rowcount > 0:
               self.connection.commit()
               print("Withdrawal successful!")
           else:
               print("Insufficient funds for withdrawal.")
       except Error as e:
           self.connection.rollback()
           print("Error while withdrawing money:", e)

   def get_balance(self):
       print("Getting account balance:")
       account_number = input("Enter account number: ")

       cursor = self.connection.cursor()
       try:
           query = "SELECT balance FROM accounts WHERE acc_no = %s"
           cursor.execute(query, (account_number,))
           result = cursor.fetchone()
           if result:
               print("Account balance:", result[0])
           else:
               print("Account not found.")
       except Error as e:
           print("Error while getting account balance:", e)

   def transfer(self):
       print("Transferring money:")
       from_account_number = input("Enter sender's account number: ")
       to_account_number = input("Enter receiver's account number: ")
       amount = float(input("Enter amount to transfer: "))

       cursor = self.connection.cursor()
       try:
            query = "UPDATE accounts SET balance = balance - %s WHERE acc_no =
%s AND balance >= %s"
            values = (amount, from_account_number, amount)
           cursor.execute(query, values)
           if cursor.rowcount > 0:
                   query = "UPDATE accounts SET balance = balance + %s WHERE
acc_no = %s"
                values = (amount, to_account_number)
               cursor.execute(query, values)
               self.connection.commit()
               print("Transfer successful!")
           else:
               print("Insufficient funds for transfer.")
       except Error as e:
           self.connection.rollback()
           print("Error while transferring money:", e)

   def get_account_details(self):
       print("Getting account details:")
       account_number = input("Enter account number: ")

       cursor = self.connection.cursor()
       try:
           query = "SELECT * FROM accounts WHERE acc_no = %s"
           cursor.execute(query, (account_number,))
           result = cursor.fetchone()
           if result:
               print("Account details:")
               print("Account Number:", result[0])
               print("Customer Name:", result[1])
               print("Account Type:", result[2])
               print("Balance:", result[3])
           else:
               print("Account not found.")
       except Error as e:
           print("Error while getting account details:", e)

   def list_accounts(self):
       print("Listing accounts:")

       cursor = self.connection.cursor()
       try:
           query = "SELECT * FROM accounts"
           cursor.execute(query)
           results = cursor.fetchall()
           if results:
               print("Accounts:")
               for result in results:
                   print("Account Number:", result[0])
                   print("Customer Name:", result[1])
                   print("Account Type:", result[2])
                   print("Balance:", result[3])
                   print("------------------------------")
           else:
               print("No accounts found.")
       except Error as e:
           print("Error while listing accounts:", e)

   def get_transactions(self):
       print("Getting transactions:")
       # Placeholder for transaction retrieval from database

   def display_menu(self):
       print("\nBanking System Menu:")
       print("1. Create Account")
       print("2. Deposit")
       print("3. Withdraw")
       print("4. Get Balance")
       print("5. Transfer")
       print("6. Get Account Details")
       print("7. List Accounts")
       print("8. Get Transactions")
       print("9. Exit")

   def main(self):
       while True:
           self.display_menu()
           choice = input("Enter your choice: ")

           if choice == "1":
               self.create_account()
           elif choice == "2":
               self.deposit()
           elif choice == "3":
               self.withdraw()
           elif choice == "4":
               self.get_balance()
           elif choice == "5":
               self.transfer()
           elif choice == "6":
               self.get_account_details()
           elif choice == "7":
               self.list_accounts()
```

```python
            elif choice == "8":
                self.get_transactions()
            elif choice == "9":
                print("Exiting the program")
                if self.connection.is_connected():
                    self.connection.close()
                break
            else:
                print("Invalid choice. Please try again.")


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



```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/task 14.7.py"
Connected to the database

Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. Get Balance
5. Transfer
6. Get Account Details
7. List Accounts
8. Get Transactions
9. Exit
```