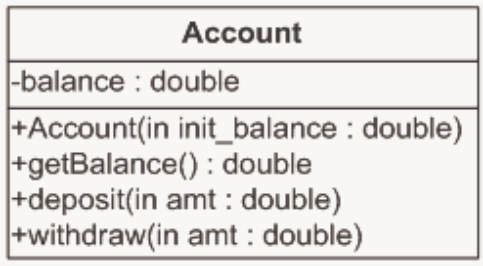
BLG332E
Object Oriented Programming

Practice Session 3

Exercise 4

In this exercise, you create a simple version of the `Account` class. A test program, `TestBanking.cpp`, has been written that creates a single account.

Task 1: Change your working directory to `chap03/exercise4`



Task 2: Create a class `Account` that implements the UML diagram given above.

a) Declare one private attribute: `balance`; tis attribute holds the current (or "running") balance of the bank account.
b) Declare a public constructor that takes one parameter (`init_balance`) that populates the `balance` attribute.
c) Declare a public method `getBalance` that retrieves the current balance.
d) Declare a public method `deposit` that adds the amount parameter to the current balance.
e) Declare a public method `withdraw` that removes the amount parameter from the current balance.

Task 3: Read the TestBanking.cpp code.

Task 4: Compile Account.cpp and TestBanking.cpp.

Task 5: Run the program. You should see the following output:

```
Create an account with 500.0 balance
Withdraw 150.0
Deposit 22.50
Withdraw 47.62
The account has a balance of 324.88
```

Exercise 5

In this exercise you will expand the Banking project by adding a `Customer` class. A customer will contain one `Account` object.

Task 1: Change your working directory to `chap03/exercise5`

Task 2: Copy `Account` class from the previous exercise lab

```
cp ../exercise4/Account.* .
```

Task 2: Create a class `Customer` class that implements the above UML diagram

a) Declare three private attributes: `firstName`, `lastName`, and `account`.
b) Declare a public constructor that takes two parameters (`f` and `l`) that populate the object attributes.
c) Declare two public accessors for the object attributes; the methods `getFirstName` and `getlastName` return the appropriate attribute.
d) Declare a public method `setAccount` to assign the `account` attribute.
e) Declare a public method `getAccount` to retrieve the `account` attribute.
f)

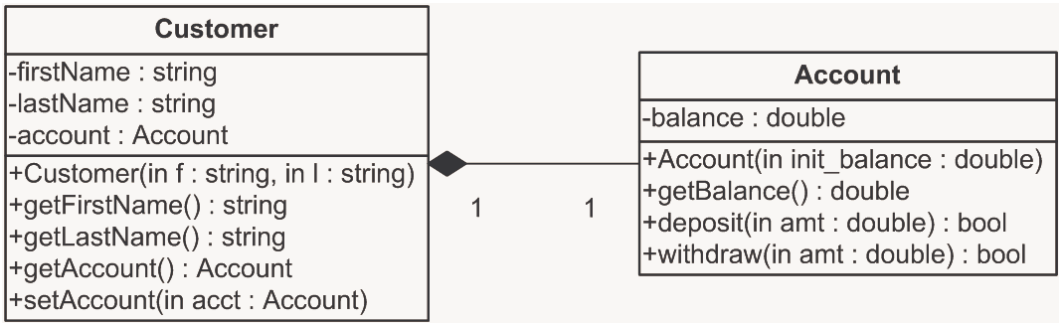
Task 3: Read the TestBanking.cpp code.
Task 4: Compile Account.cpp, Customer.cpp, and TestBanking.cpp.
Task 5: Run the program. The output generated should be:

```
Creating the Customer Jane Smith
Creating her account with a 500.0 balance.
Withdraw 150.0
Deposit 22.50
Withdraw 47.62
Customer [Smith, Jane] has a balance of 324.88
```

Exercise 6

In this exercise, you will modify the withdraw method to return a boolean value to specify whether the transaction was successful.

Task 1: Change your working directory to `chap03/exercise6`



Task 2: You can copy the Account.* and Customer.* files you created in Exercise 5.
Task 3: Modify the `Account` class to place conditions on the `withdraw` and `deposit` methods.

a) Modify the `deposit` method to return `true`.
b) Modify the `withdraw` method to check that the amount being withdrawn is not greater than the current balance. If `amt` is less than `balance`, then subtract the amount from the balance and return `true`; else leave the balance alone and return `false`.

Task 4: Read the TestBanking.cpp code.
Task 5: Compile Account.cpp, Customer.cpp, and TestBanking.cpp.
Task 6: Run the program. The output generated should be:

```
Creating the Customer Jane Smith
Creating her account with a 500.0 balance.
Withdraw 150.0:true
Deposit 22.50: true
Withdraw 47.62: false
Withdraw 400.0: false
Customer [Smith, Jane] has a balance of 324.88
```