National University of Computer and Emerging Sciences



Lab Exercise 06

For

Object Oriented Programming Lab

|  |  |
| --- | --- |
| Course Instructor(s) | Mr. Usman Ghous |
| Lab Instructor(s) | Mr. Usman Ghous |
| Semester | Spring 2021 |

**FAST School of Computing**

# Instructions:

1. Make a word document with the naming convention “SECTION\_ LAB#\_ROLLNO” and put all your source code and snapshots of its output in it. Make sure your word file is formatted properly.
2. **Plagiarism is strictly prohibited.**
3. Do not discuss solutions with one another.

# Useful links

**Question#1**

Write a class **Employee**.

Write another class **Manager** inherited from **Employee**.

Write another class **Project** inherited from **Manager**.

With required data members, write two functions in each class **inputData()** and **printData().** For

each class, these functions will input and print their attributes. Use **function overriding**.

Create an object of class **Project** and demonstrate the working in a program.

|  |
| --- |
| **Question#2** |

A point in the x-y plane is represented by its x-coordinate and y-coordinate.  
Design a class, pointType, that can store and process a point in the x-y  
plane. You should then perform operations on the point, such as setting the  
coordinates of the point, printing the coordinates of the point, returning the  
x-coordinate, and returning the y-coordinate. Also, write a program to test  
various operations on the point.

|  |
| --- |
| **Question#3** |

Every circle has a center and a radius. Given the radius, we can determine  
the circle’s area and circumference. Given the center, we can determine its  
position in the x-y plane. The center of the circle is a point in the x-y plane.  
Design a class, circleType, that can store the radius and center of the  
circle. Because the center is a point in the x-y plane and you designed the  
class to capture the properties of a point in Question 2 you  
must derive the class circleType from the class pointType. You  
should be able to perform the usual operations on the circle, such as setting  
the radius, printing the radius, calculating and printing the area and circumference, and carrying out the usual operations on the center. Also, write a  
program to test various operations on a circle.

|  |
| --- |
| **Question#4** |

Every cylinder has a base and height, wherein the base is a circle. Design a class, cylinderType, that can capture the properties of a cylinder and perform the usual operations on the cylinder. Derive this class from the class circleType designed in Question 3. Some of the operations that can be performed on a cylinder are as follows: calculate and print the volume, calculate and print the surface area, set the height, set the radius of the base, and set the center of the base. Also, write a program to test various operations on a cylinder.

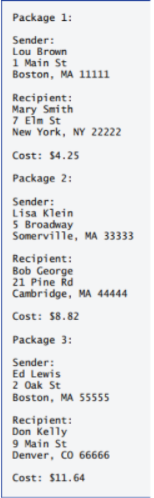
**Question#5**

Create an inheritance hierarchy that a bank might use to represent customers’ bank accounts. All customers at this bank can deposit (i.e., credit) money into their accounts and withdraw (i.e., debit) money from their accounts. More specific types of accounts also exist. Savings accounts, for instance, earn interest on the money they hold. Checking accounts, on the other hand, charge a fee per transaction (i.e., credit or debit). Create an inheritance hierarchy containing base class Account and derived classes SavingsAccount and CheckingAccount that inherit from class Account. Base class Account should include one data member of type double to represent the account balance. The class should provide a constructor that receives an initial balance and uses it to initialize the data member. The constructor should validate the initial balance to ensure that it’s greater than or equal to 0.0. If not, the balance should be set to 0.0 and the constructor should display an error message, indicating that the initial balance was invalid. The class should provide three member functions. Member function credit should add an amount to the current balance. Member function debit should withdraw money from the Account and ensure that the debit amount does not exceed the Account’s balance. If it does, the balance should be left unchanged and the function should print the message “Debit amount exceeded account. Balance” Member function get Balance should return the current balance. Derived class SavingsAccount should inherit the functionality of an Account, but also include a data member of type double indicating the interest rate (percentage) assigned to the Account. SavingsAccount’s constructor should receive the initial balance, as well as an initial value for the SavingsAccount’s interest rate. SavingsAccount should provide a public member function calculateInterest that returns a double indicating the amount of interest earned by an account. Member function calculateInterest should determine this amount by multiplying the interest rate by the account balance. [Note: SavingsAccount should inherit member functions credit and debit as is without redefining them.] Derived class CheckingAccount should inherit from base class Account and include an additional data member of type double that represents the fee charged per transaction. CheckingAccount’s constructor should receive the initial balance, as well as a parameter indicating a fee amount. Class CheckingAccount should redefine member functions credit and debit so that they subtract the fee from the account balance whenever either transaction is performed successfully. CheckingAccount’s versions of these functions should invoke the base-class Account version to perform the updates to an account balance. CheckingAccount’s debit function should charge a fee only if money is actually withdrawn (i.e., the debit amount does not exceed the account balance). [Hint: Define Account’s debit function so that it returns a bool indicating whether money was withdrawn. Then use the return value to determine whether a fee should be charged.] After defining the classes in this hierarchy, write a program that creates objects of each class and tests their member functions. Add interest to the SavingsAccount object by first invoking its calculateInterest function, then passing the returned interest amount to the object’s credit function. Easy?

**Question#6**

Package-delivery services, such as FedEx®, DHL® and UPS®, offer a number of different shipping options, each with specific costs associated. Create an inheritance hierarchy to represent various types of packages.

* Use class **Package** as the base class of the hierarchy, then include classes **TwoDayPackage** and **OvernightPackage** that derive from Package.
* Base class Package should include data members representing the name, address, city, state and ZIP code for both the sender and the recipient of the package, in addition to data members that store the weight (in ounces) and cost per ounce to ship the package.
* Package’s constructor should initialize these data members. Ensure that the weight and cost per ounce contain positive values.
* Package should provide a public member function calculateCost that returns a double indicating the cost associated with shipping the package. Package’s calculateCost function should determine the cost by multiplying the weight by the cost per ounce.
* Derived class TwoDayPackage should inherit the functionality of base class Package, but also include a data member that represents a flat fee that the shipping company charges for two-day-delivery service.
* TwoDayPackage’s constructor should receive a value to initialize this data member. TwoDayPackage should redefine member function calculateCost so that it computes the shipping cost by adding the flat fee to the weight-based cost calculated by base class Package’s calculateCost function.
* Class OvernightPackage should inherit directly from class Package and contain an additional data member representing an additional fee per ounce charged for overnight-delivery service. OvernightPackage should redefine member function calculateCost so that it adds the additional fee per ounce to the standard cost per ounce before calculating the shipping cost.
* Write a test program that creates objects of each type of Package and tests member function calculateCost. (i.e. Package package1,TwoDayPackage package2, OvernightPackage package3)
* Example output looks like:



**Question#7**

Design a class named Person with the following member variables:

* FirstName
* LastName
* Address
* City
* State
* Zip
* PhoneNo
* Struct DOB (day, month, year)

Having a private constructor.

Write the appropriate accessor (getter) and mutator (setter) functions for these member variables.

In the main() function, create an array of objects of user defined size by calling a static function for making objects.

Input data of all the persons and then display it by using a constant function.

**Question#8**

Design a class Car having attributes.

PersonData Driver (an object of class done in Q#7)

String carName

Int carID

String type (SUV, Wagon, Electrical etc)

Static int count

Const int wheels (assuming every car will have four wheels)

Having default, overloaded, copy constructors with initializer lists and a destructor,

Having functions addCar(), deleteCar(), updateCar(), printCar().

In the main(), you’ll only have a pointer of size 1 at start.

Here, you’ll have a constant object initialized explicitly like this const Car constantObj (Person, cName, cID, cType);

Now, print this object at the start of the program and then print the menu.

You will have a menu driven program to perform following functionalities.

1. Add Car

2. DeleteCar

3. Update Car Attributes

4. Print Details

5. Print List of cars

One thing, while printing the Driver details display only his name.

**Question#9**

Design a class named PersonData with the following member variables:

* FirstName
* LastName
* Address
* City
* State
* Zip
* PhoneNo

Write the appropriate accessor (getter) and mutator (setter) functions for these member variables. Next, design a class named CustomerData, which is derived from the PersonData class. The CustomerData class should have the following member variables:

* CustomerNumber
* MailingList

The CustomerNumber variable will be used to hold a unique integer for each customer. The MailingList variable should be a bool. It will be set to true if the customer wishes to be on a mailing list, or false if the customer does not wish to be on a mailing list. Write appropriate Accessor and Mutator functions for these member variables. CustomerData class will have the

• InputCustomerData member function which will Input all the data for customer. (use function over riding).

• DisplayCustomerData member function which will display all the data for customer. (use function over riding).

Demonstrate an object of the CustomerData class in a simple program.

**Question#10**

A retail store has a preferred customer plan where customers may earn discounts on all their purchases. The amount of a customer’s discount is determined by the amount of the customer’s cumulative purchases in the store.

• When a preferred customer spends $500, he or she gets a 5% discount on all future purchases.

• When a preferred customer spends $1,000, he or she gets a 6% discount on all future purchases.

• When a preferred customer spends $1,500, he or she gets a 7% discount on all future purchases.

• When a preferred customer spends $2,000 or more, he or she gets a 10% discount on all future purchases.

Design a class named PreferredCustomer, which is derived from the CustomerData class you created in **problem 9**. The PreferredCustomer class should have the following member variables:

• purchasesAmount (a double)

• discountLevel (a double)

The purchasesAmount variable holds the total of a customer’s purchases to date. The discountLevel variable should be set to the correct discount percentage, according to the store’s preferred customer plan. Write appropriate member functions for this class and demonstrate it in a simple program.

**Question#11**

Banks offer various types of accounts, such as savings, checking, certificate of deposits, and money market, to attract customers as well as meet with their specific needs. Two of the most commonly used accounts are savings and checking. Each of these accounts has various options. For example, you may have a savings account that requires no minimum balance but has a lower interest rate. Similarly, you may have a checkingaccount that limits the number of checks you may write. Another type of account that is used to save money for the long term is certificate of  
deposit (CD).

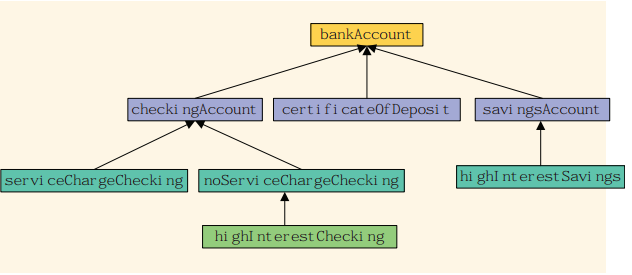
In this programming exercise, you **use abstract classes and pure virtual functions** to design classes to manipulate various types of accounts. For simplicity, assume that the bank offers three types of accounts: savings, checking, and certificate of deposit, as described next.

**Savings accounts:** Suppose that the bank offers two types of savings accounts: one that has no minimum balance and a lower interest rate and another that requires a minimum balance and has a higher interest rate.

**Checking accounts:** Suppose that the bank offers three types of checking  
accounts: one with a monthly service charge, limited check writing, no minimum balance, and no interest; another with no monthly service charge, a minimum balance requirement, unlimited check writing and lower interest; and a third with no monthly service charge, a higher minimum requirement, a higher interest rate, and unlimited check writing.

**Certificate of deposit (CD):** In an account of this type, money is left for  
some time, and these accounts draw higher interest rates than savings or  
checking accounts. Suppose that you purchase a CD for six months. Then  
we say that the CD will mature in six months. Penalty for early withdrawal  
is stiff.

Figure shows the inheritance hierarchy of these bank accounts.



Note that the classes **bankAccount** and **checkingAccount** are abstract. That is, we cannot instantiate objects of these classes. The other classes in Figure are not abstract.  
**bankAccount:** Every bank account has an account number, the name of the owner, and a balance. Therefore, instance variables such as **name**, **accountNumber**, and **balance** should be declared in the abstract class **bankAccount**. Some operations common to all types of accounts are retrieve account owner’s name, account number, and account balance; make deposits; withdraw money; and create monthly statement. So include functions to implement these operations. Some of these functions will be pure virtual.

**checkingAccount:** A checking account is a bank account. Therefore, it inherits all the properties of a bank account. Because one of the objectives of a checking account is to be able to write checks, include the pure virtual function **writeCheck** to write a check.  
**serviceChargeChecking:** A service charge checking account is a checking account. Therefore, it inherits all the properties of a checking account. For simplicity, assume that this type of account does not pay any interest, allows the account holder to write a limited number of checks each month, and does not require any minimum balance. Include appropriate named constants, instance variables, and functions in this class.  
**noServiceChargeChecking:** A checking account with no monthly service charge is a checking account. Therefore, it inherits all the properties of a checking account. Furthermore, this type of account pays interest, allows the account holder to write checks, and requires a minimum balance.

**highInterestChecking**: A checking account with high interest is a checking account with no monthly service charge. Therefore, it inherits all the properties of a no service charge checking account. Furthermore, this type of account pays higher interest and requires a higher minimum balance than the no service charge checking account.  
**savingsAccount:** A savings account is a bank account. Therefore, it inherits all the properties of a bank account. Furthermore, a savings account also pays interest.  
**highInterestSavings:** A high-interest savings account is a savings account. Therefore, it inherits all the properties of a savings account. It also requires a minimum balance.  
**certificateOfDeposit:** A certificate of deposit account is a bank account. Therefore, it inherits all the properties of a bank account. In addition, it has instance variables to store the number of CD maturity months, interest rate, and the current CD month.

Write the definitions of the classes described in this programming exercise and a  
program to test your classes.

**Question#12**

1. Define a class person with data members Name (string), Age (int\*), Employment status (Boolean) and favorite sports (string).

2. Write a default constructor to initialize the data members to the following values:

1. Name = null
2. Age =0
3. Employment status = 1/true (for employed)
4. Favorite sports = swimming

3. Create an object p1.

4. Copy p1 object to p2 object using a copy constructor.

5. Output the values of p2 object.