CS1004 - Object-oriented Programming (OOP)

Assignment # 2

Max Points: **60**

Due Date: **Sunday, November 7, 2021, 11:59 p.m.**

**Carefully read the following instructions!**

* It should be clear that your assignment would **not get any credit** if the assignment is submitted after the **due date**. **No** assignment will be **accepted after the due date**.
* Strict action will be taken if the submitted solution is copied from any other student.
* If you people find any mistake or confusion in the assignment (Question statement), please consult before the deadline. After the deadline no queries will be entertained in this regard.
* For any query, feel free to email at: **farah.sadia@nu.edu.pk**
* Submission: Submission will only be accepted through **GOOGLE CLASSROOM**.
* Submit all your codes with your Student ID and task number. **“K211234\_Q1”**.
* Every code should be with proper **commenting**.

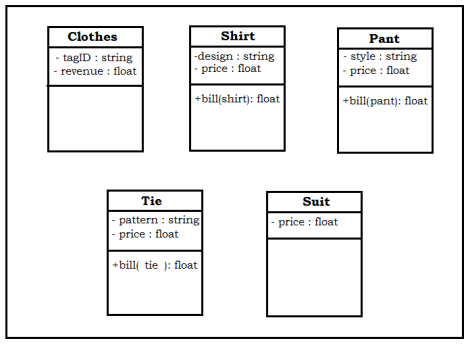
**Question # 01:**

Implement the inheritance depicted in Figure 2 and provide a constructor to initialize variables as A::x = 2, B::y = 5, C::z = 12 using object declaration of class C i.e. C ob(2, 5, 12);



**Question # 02:**

Zed Clothing is a top-notch brand that sells shirts, pants and ties to its dedicated buyers. Apart from selling shirts, pants and ties, Zed Clothing also offers complete suits that comprise shirt, pant and a tie. There is also a special tag ID for each clothing type. Furthermore, there is another process involved, that is tax calculation. This tax calculation is done on the basis of attributes of the clothing types. It must have complete access to the prices of each clothing type as well as the overall revenue of Zed Clothing. The classes for this scenario are presented in Figure 1. Read the assumptions below and then perform the tasks. Note that Figure 1 contains some of the variables that should be present in the classes, other attributes and functions can be defined as per the requirement of each task.



**Assumptions:**

1. You may create or modify any variables and override any functions if needed to satisfy the requirements of the question. But their role in the program have to be justifiable.
2. The design for shirts can either be formal, traditional or casual. The unit price for each of these types is Rs. 1500.
3. Shirts and pants are taxable items, but ties are not.
4. The pant styles available are either bell-bottomed, straight fit or narrow fit. The unit price for pant is Rs. 1200 for bell-bottomed, Rs. 1600 for straight fit and Rs. 2000 for narrow fit.
5. The pattern for tie can either be stripes or checks.
6. The price for striped tie is Rs. 700 and checkered tie is Rs. 800.
7. The tax rate for shirts is 7% of the price and for pants it is 4% of the price.

**Tasks to be performed:**

1. Using Figure 1, illustrate how different objects will interact with each other using an Object interaction model.
2. Identify the type(s) of inheritance present in the model and list different classes, which are involved in that particular type of inheritance.
3. Declare variables and also provide suitable implementation for default and parameterized constructor(s) of each class.
4. Add a function bill( ) in the class Suit that calculates the total price of suit based on the items present in the suit.
5. Overload the function bill( ) in Shirt class to accept discount vouchers (7-letter string as discount code) along with the shirt object and returns the discounted price.
6. Overload the " < " operator such that it displays if a given Shirt instance gives more profit than a Pant instance.
7. For keeping track of the inventory, there must be a mechanism to find how much of each individual item (shirt, pant or tie) is remaining in stock. Also provide mechanism to see the current revenue (overall profit). Keep in mind that each type of clothing has a different price.
8. Provide a copy constructor for copying objects of Pant class.
9. Provide a global mechanism for tax calculation such that it has access to all of Zed Clothing's revenue details.

**Question # 03:***(Package Inheritance Hierarchy)*

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 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.

**Question # 04:** *(Airline Reservations System)*

A small airline has just purchased a computer for its new automated reservations system. You’ve been asked to program the new system. You are to write a program to assign seats on each flight of the airline’s only plane (capacity: 10 seats). Your program should display the following menu of alternatives—Please type 1 for "First Class" and Please type 2 for "Economy". If the person types 1, your program should assign a seat in the first class section (seats 1–5). If the person types 2, your program should assign a seat in the economy section (seats 6–10). Your program should print a boarding pass indicating the person’s seat number and whether it’s in the first class or economy section of the plane. Use a one-dimensional array to represent the seating chart of the plane. Initialize all the elements of the array to false to indicate that all seats are empty. As each seat is assigned, set the corresponding elements of the array to true to indicate that the seat is no longer available.

Your program should, of course, never assign a seat that has already been assigned. When the first class section is full, your program should ask the person if it’s acceptable to be placed in the economy section (and vice versa). If yes, then make the appropriate seat assignment. If no, then print the message "Next flight leaves in 3 hours."

**Question # 5:***(Account Inheritance Hierarchy)*

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 getBalance 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.