

Practice questions for week 7

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`.

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 is 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.

Develop a polymorphic banking program using the `Account` hierarchy created in the above question. Create a vector of `Account` pointers to `SavingsAccount` and `CheckingAccount` objects. For each `Account` in the vector, allow the user to specify an amount of money to withdraw from the `Account` using member function `debit` and an amount of money to deposit into the `Account` using member function `credit`. As you process each `Account`, determine its type. If an `Account` is a `SavingsAccount`, calculate the amount of interest owed to the `Account` using member function `calculateInterest`, then add the interest to the account balance using member function `credit`. After processing an `Account`, print the updated account balance obtained by invoking base class member function `getBalance`.