**Objective:**

- Exploring the design aspect of Object Orientation.
- Focuses on the Syntactical issues related to inheritance relationship in C++.

## Task-1:

(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.

## Task-2:

The following class hierarchy represents class design for 'R S J Sales Store' (you already familiar with it.) Provide the definition of the functions written in the following hierarchy.

*[Solution Provided by TA: Ahmed Sadiq*
*Note: Keeping in mind only the requirements given in case study, this design hierarchy can be criticized very heavily. Try to find out those problem(s).]*

```
class Contact
{
   CString name;
   CString address;
   CString phonenumber;
        //Provide default and parameterized constructors appropriately
        //Provide Getters/Setters
}
class Customer
{
   Contact address;
   CString shipingAddress;
   CString credCardNumber;

   void printShippingLabel();

        //Provide default and parameterized constructors appropriately
        //Provide Getters/Setters
};
```

```
class Invoice
{
   Customer client;
   SalesAgent sA;

   Date orderDate;
   bool isShiped
   bool isPayed;
   bool isReceived;

   Item * shopingCart;
   int * quantity;


   void pay();
   void ship();
   void received();

   int calculateBill() // calculate bill of the customer


};
class SalesAgent
{
   Contact address
   int lastCommisionAmount;
        //Provide default and parameterized constructors appropriately
        //Provide Getters/Setters

};
class Item
{
   int price;
   CString name;
   CString description;
        //Provide default and parameterized constructors appropriately
        //Provide Getters/Setters
};
class CD : public Item
{
   CString *publisher;
        //Provide default and parameterized constructors appropriately
        //Provide Getters/Setters
};
class DVD : public Item
{
   CString *publisher;
        //Provide default and parameterized constructors appropriately
        //Provide Getters/Setters
};
class Book : public Item
{
   CString *author;
        //Provide default and parameterized constructors appropriately
        //Provide Getters/Setters
}
```

## Home Task-1:

*[Taken From: Thinking in C++]*

This exercise creates the design pattern called *proxy*. Start with a base class **Subject** and give it three functions: **f( )**, **g( )**, and **h( )**. Now inherit a class **Proxy** and two classes **Implementation1** and **Implementation2** from **Subject**. **Proxy** should contain a pointer to a **Subject**, and all the member functions for **Proxy** should just turn around and make the same calls through the **Subject** pointer. The **Proxy** constructor takes a pointer to a **Subject** that is installed in the **Proxy**(usually by the constructor). In **main( )**, create two different **Proxy** objects that use the two different implementations. Now modify **Proxy** so that you can dynamically change implementations.

*Though no one can go back and make a brand new start, anyone can start from now and make a brand new ending.*
[Anonymous]