**Abstract Class and Polymorphism**

1. Create an Abstract Class Employee (firstName, lastName, SSN and an abstract function “earnings()”).

Create Sub Classes

* Weekly Employee (weekly salary)
* Hourly Employee (hours, wagePerHour)
* CommisionEmployee (sales, commission Rate)
  + BasePlusCommisionEmployee (basicSalary)

Call the earning Method Polymorphically. Reset the salary for BasePlusCommisionEmployee using down casting.

1. Package-delivery services, 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 super class of the hierarchy, then include classes TwoDayPackage and OvernightPackage that derive from Package.

super class **Package** should include data members representing the *name and address* 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'sconstructor 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 fe*e to the *cost* calculated by base class Package's *calculateCost*() function.

Class **OvernightPackage** should inherit from class Package and contain an additional data member representing an *additionalfee* charged for overnight-delivery service.

OvernightPackage should redefine member function *calculateCost*() so that it computes the *shipping cost* by adding the *additionalfee* to the *cost* calculated by base class Package's *calculateCost*() function.

Write a test program that creates objects of each type of Package and tests member function *calculateCost*() using polymorphism .

1. Create a class named Movie that can be used with your video rental business.

The Movie class should have ID Number, movie title and number of days for which movie is rented. The class should have appropriate accessor and mutator methods and an equals() method that determines whether two movies are equal.

Next, create three additional classes named Action , Comedy , and

Drama that are derived from Movie .

Finally, create an overridden method named calcLateFees that takes as input the number of days a movie is late and returns the late fee for that movie.

Action movies have a late fee of $3/day, comedies are $2.50/day, and dramas are $2/day.

Note: Declare calcLateFees() method as abstract in super class

Call the calcLateFees method polymorphically in main. Reset Id for Action Movies using downcasting.

Note: Call equals method on objects of Action and comedy objects to verify the concept of type compatibility.

Create an abstract class “Person”, with data member “name”. Create set and get methods, and an abstract Boolean method “isOutstanding()”.

Derive two classes Student and Professor. Student class has data member CGPA.

Professor Class has data member numberOfPublications. Provide setters and getters and implementation of abstract function in both classes.

In student class isOutstanding() will return true if CGPA is greater than 3.5. In the Professor class isOutstanding() will return true, if numberOfPublications> 50.

In the main class create an array of Person class and call isOutstanding() function for student and professor. isOutstanding() for professor should be called after setting the publication count to 100.

5. Create a class hierarchy that performs conversions from one system of units to another. Your program should perform the following conversions,

i. Liters to Gallons, ii. Fahrenheit to Celsius and iii. Feet to Meters

The Super class **convert** declares two variables, val1 and val2, which hold the initial and converted values, respectively. It contains an abstract function “compute()”.

The function that will actually perform the conversion, compute() must be defined by the classes derived from convert. The specific nature of compute() will be determined by what type of conversion is taking place.

Three classes will be derived from convert to perform conversions of Liters to Gallons (l\_to\_g), Fahrenheit to Celsius (f\_to\_c) and Feet to Meters (f\_to\_m), respectively. Each derived class implements compute() in its own way to perform the desired conversion.

Test these classes from main() to demonstrate that even though the actual conversion differs between l\_to\_g, f\_to\_c, and f\_to\_m, the interface remains constant.