Encapsulation Using Traditional Accessors and Mutators

Task 1

Using Visual Studio, create a new Console Application project named EmployeeApp and insert a new class file (named Employee.cs) using the Project Add class menu item. Update the Employee class with the following fields, methods, and constructors:

```
class Employee
{
    // Field data.
    private string empName;
    private int empID;
    private float currPay;
    // Constructors.
    public Employee() { }
    public Employee(string name, int id, float pay)
    {
        empName = name;
        empID = id;
        currPay = pay;
    }
    // Methods.
    public void GiveBonus(float amount)
    {
        currPay += amount;
    }
    public void DisplayStats()
    {
        Console.WriteLine("Name: {0}", empName);
        Console.WriteLine("ID: {0}", empID);
        Console.WriteLine("Pay: {0}", currPay);
    }
}
```

Code Snippet 1.0

Notice that the fields of the Employee class are currently defined using the private keyword. Given this, the empName, empID, and currPay fields are not directly accessible from an object variable. Therefore, the following logic in Main() would result in compiler errors:

```
static void Main(string[] args)
{
   Employee emp = new Employee();
   // Error! Cannot directly access private members
   // from an object!
   emp.empName = "Marv";
}
```

Code Snippet 1.1

Update the class properties and constructors so that we write the code shown below in the Main method. You may make use of C# .Net automatic properties.

```
static void Main(string[] args)
{
    Console.WriteLine("***** Fun with Encapsulation *****\n");
    Employee emp = new Employee("Marvin", 456, 30000);
    emp.GiveBonus(1000);
    emp.DisplayStats();
    // Reset and then get the Name property.
    emp.Name = "Marv";
    Console.WriteLine("Employee is named: {0}", emp.Name);
    Console.ReadLine();
}
```

Code Snippet 1.2

Polymorphism The Details of Inheritance

Task 2

Now that you have seen the basic syntax of inheritance, let's create a more complex example and get to know the numerous details of building class hierarchies. To do so, you will be reusing the Employee class you designed in previous task. To begin, create a new C# Console Application project named EmployeeManager.

Add a Employee.cs file to the project. You can do this by Right click on project -> Add -> New Item…

You can make use of the C# automatic properties in already class shown above in code snippet 1.0.

After updating the Employee class should look like:

```
public class Employee
{
    // Field data.
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
    public float CurrentPay { get; set; }

    // Constructors.
    public Employee() { }
    public Employee(int id, string name, float pay)
    {
        Name = name;
        Id = id;
        CurrentPay = pay;
    }
    // Methods.
    public void GiveBonus(float amount)
    {
        CurrentPay += amount;
    }
    public void DisplayStats()
    {
        Console.WriteLine("Name: {0}", Name);
        Console.WriteLine("ID: {0}", Id);
        Console.WriteLine("Pay: {0}", CurrentPay);
    }

}
```

Code Snippet 1.3

Task 3

Your goal is to create a family of classes that model various types of employees in a company. Assume you want to leverage the functionality of the above defined Employee class to create two new classes (SalesPerson and Manager). The new SalesPerson class “is-an” Employee (as is a Manager). Remember that under the classical inheritance model, base classes (such as Employee) are used to define general characteristics that are common to all descendants. Subclasses (such as SalesPerson and Manager) extend this general functionality while adding more specific functionality.

Further Manager class extends Employee by recording the number of stock options, while the SalesPerson class maintains the number of sales made.

After adding above mentioned requirements, the code should look like :

```
// Managers need to know their number of stock options.
class Manager : Employee
{
    public int StockOptions { get; set; }
}

// Salespeople need to know their number of sales.
class SalesPerson : Employee
{
    public int SalesNumber { get; set; }
}
```

Try yourself:

- Create a sales person object that should accept id, name, age, salesNumber parameters.
- Create a manager object that should accept id, name, age, stockptions parameters.
- Controlling creation using constructor calling.
    - Using this and base keywords
- Redefine GiveBonus as a virtual function
- Create a Benefits Package for employee
    - Benefit can Medical allowance
    - Education allowance
    - Death Benefit
-