CIS 3309 Final Project – Inheritance, Part I: Employee/Manager/Client Example (Ver 3a, Spring 2010)

The Employee/Manager/Client example code is closely tied to your Final Project, Part I, on Inheritance. The intent of this example is to provide you with examples to illustrate the concepts you need to understand to do Part I of the Final Project. This document is a supplement to the Final Project Part 1 Specifications document (located in the Lab Schedule Document as [HWA #8](HWA08%20-%20Final%20Project%20Part%20I%20-%20Lab%20Assignment%20with%20Worked%20Example%2C%20and%20Starter%20Code.htm)). It contains a basic description of the Employee/Manager/Client example and very brief discussions of the mechanics of .NET classes. For more in depth information, refer to the Project Specifications document, the [Lecture Notes on inheritance](Lecture%20Set%20ZZZ%20Appendix%20E%20-%20Inheritance%20v7%20%2811-25-2007%29%20Fall%2007.doc) posted on the CIS 3309 Website (Appendix E, especially pp. 1-14), and the code Example also posted in your Lecture Notes, [Appendix E Part D2](Lecture%20Set%20ZZZ%20Appendix%20E%20-%20EmpManV3a-VB2008%20Spring%202010.zip). You should also do a quick pass through your textbook (Chapters 17 – 20) to reinforce the material on the website. Please look at the code and comments in the Employee/Manager/Client example project before attempting to code the Product hierarchy in the Final Project. Also, take note to how dynamic Tooltips are implemented in this project.

The file “EmpManV3a-VB2008.zip” (click on the link for Appendix E Part D2) contains a basic example of an organization’s data that has been converted to an object oriented data model. The organization has three data entities: managers, workers and clients. The entities have unique data sets but some fields are common to two or more entities. A good object model arranges data in the same way as is done in a normalized database. Consider the data entities:



Notice that all three entities have data attributes {Name, Birth Date, ID} in common. As part of our object-oriented modeling of the above data, we create a fourth object, a Person, to model these three common entities. The three entities, each of which can be categorized as a Person, can be related in a tree structure as shown next with Person at the root node. This subclass hierarchy illustrates the use of a technique known as Subclassing for Specialization. Here, each of the subclasses specializes on the attributes encapsulated in the parent class (Person) by adding additional attributes appropriate to the this subclass, but not common to all of them.



For example, the Client subclass (entity) has data field “Type”, which does not appear in the two other entities. This will be a leaf node in the tree. Next, we see that both the Manager and Worker have the data set {Job Title} in common. The Manager and Worker are Employee(s) of the organization and can be represented in the Employee sub tree:



The three leaf nodes have no remaining common data. The Person object hierarchy has five nodes: one root node, one inner node and three leaf nodes. The “is-a” test can be used to check object hierarchies: A Manager is an Employee and an Employee is a Person. Replace “is a(n)” in the previous statements with inherits and Manager inherits Employee and Employee inherits Person. Notice the second line in the example below: “Inherits Person”. This line creates the inheritance relationship between the Employee and the Person classes. The “<Serializable()>” tag in the first line will force .NET to include functions that allow objects to be saved to a file (similar to Java). The “MustInherit” term disallows the direct instantiation of an Employee. Person is also a MustInherit class. Thus, our application cannot have objects of type Person or type Employee. It can only have objects instantiated from the subclasses that are leaf nodes of the tree (Manager, Worker and Client).

<Serializable()> Public MustInherit Class Employee

 Inherits Person

 Private HiddenEmployeeJobTitle As String ' Employee job title

. . .

End Class

The remaining tasks are to code and test the class hierarchy. Each node in the Person hierarchy includes the following:

* The private data
* Default and parameterized constructors
* Properties to access the private data
* Sub procedures to save data from the form to the list and display data in the list on the form
* An overridden toString function to convert the data in the objects to a string
* A shared Validate function to check that all necessary data is entered on the form

The tree below shows the complete Person class hierarchy. It is rendered automatically by VB .NET 2008 by right clicking on the project in the Solution Explorer and selecting “View Class Diagram”.



A couple of notes on the full blown example code contained in Appendix E Part D2.

1. The PersonList data is a list of Person objects. Because the Manager, Worker and Client classes directly or indirectly inherit the Person class, the PersonList can contain instances of all three objects. This is called late binding.
2. The use of the <Serializable()> tag facilitates the conversion of an instantiated object to a file. This is referred to as a persistent object. This object can also be read from a file back to an instance in memory. The Project contains information about persistent objects.
3. The FormController class contains methods to activate and deactivate parts of the form based on the current object. The figure below shows the form after the “MakeClient” button is clicked. Run the application to observe the forms behavior.

