Programming Assignment 3 Write Up

1. (Daniel Davidson)

* The naming scheme I chose is based on the heirarchy the would take place in a restaurant kitchen. Each class is related to the abstract KitchenStaff class and inherits from that class. The HeadChef is the manager of the kitchen, at the highest employee level. Then the SousChef is at the second level and has the LineCook inheriting from it.
* To create the class hierarchy, I used the toolbox to drag and add a class to the diagram. After adding all of the classes, I used the inheritance arrow to link all of the classes that have inheritance.

The class diagram allows you to set up you class hierarchy first, without coding. When you add a class to the diagram a class file is generated. You can select which classes inherit from each other. Also, you can define fields, methods, properties, and events.

* The Abstract keyword is used to prevent instantiation of the Kitchen worker class. The constraints that it puts on the class are, you can’t create an instance, the class can’t be static, and it can’t support multiple interfaces. Abstract classes affect class members by restricting child classes from changing fields and methods. The override keyword has to be used before the method.
* The code that gets generated shows what the classes are inheriting from. This makes it easier to code because it saves the programmer time.

1. (Amina Khan)

* Naming Scheme: I chose colors for my naming convention. It may seem unorthodox, but I considered different colors as part of a category of colors being either “warm” or “cool”, which are all part of an abstract class of “colors”.
* To create a class diagram, I followed these steps:

1. Click on **Project** (in the menu bar) and select **Add New Item…**
2. From the Item box, select **Class Diagram**.
3. Enter a name for the new diagram and click **Add.**
4. This opens the new, empty diagram window. Click **View** and select **ToolBox**
5. From the toolbox, drag a few **Class** items and place them in the diagram window.
6. Name the classes and organize in the form of a “hierarchy” or “tree” depending on the level the class is at.
7. Click on **Inheritance** in the toolbox. The icon will change to an arrow to link two classes. First, click on a “child class” and drag the line to its “parent class”. This will establish an “inheritance” relationship between the classes.

* The abstract keyword is basically used to define a base class in a hierarchy. This class does not instantiate any objects but contains abstract methods that the inherited class must override when implementing. The abstract class contains at least one abstract method.
* The generated code helped me understand which classes were inherited from the abstract class or their parent class.

**Prob 2:**

* Access Modifiers constrain access to child classes. With different access modifiers, types are constrained based on which access modifier is being used. The Private keyword limits other classes, from invoking on the object. Nested types and non nested types are limited to what the access modifier allows.
* The interaction with the abstract class modifier, takes place when the derived class are required to account for implementation of the abstract classes.
* The results of the constructor calling demo is, to show the relationship in hierarchy that the derived classes have with the base class. Also to prevent derived classes from creating instances of the base class; directly through polymorphism.
* Results of the constructor calling sequence. The results of the sequence show the way of inheritance. The derived classes were all affected by the constructors of the base class when the program was ran. They were affected through the base class constructors. When the program is run the derived class constructors add functionality to the base class.
* Protected keyword:
* The protected keyword allows access with in the class and by derived class instances.
* The functionality is from base class to derived classes only.
* Protected access modifiers are meant to give derived classes access to the properties of the base class.
* The Protected property gives control over its properties.

**Prob 3:**

* Aggregation - Aggregation gives a “has a” relationship. When one class is directly connected to another class. Like a customer “has a” address, the customer owns the address object. The Address object can’t relate to any other entity.
* Composition - Composition occurs when two objects are dependent on each other. Similar to a computer needing battery to turn on. If the computer no longer exists, then the battery is gone also. If the computer no longer exists then the battery has no use.

<https://medium.com/@ibrahimyengue/association-aggregation-and-composition-in-c-8cbeaa81201d>

* This article gives the example of aggregation by showing the relationship that a manager has with employees. The author refers to aggregation as one object having ownership over another. When one object only reports to one other entity in. The authors explanation of composition is when two objects depend on each other for the span of their operation. When one of two objects depends on the other object. These two definitions are both comparable to my definition of aggregation and composition. My aggregation definition was similar, because I also describe aggregation as being a “has a” relationship. The definition of composition is similar, because I described one object being dependent on another. Which is similar to what the article describes.
* The differences in the two relationships. Aggregation states that the derived class can survive independently from the parent class. Composition relies on the base class and can’t survive on its own.