Question 1. My answer was more describing encapsulation instead of how it’s implemented, and the example I gave didn’t refer to encapsulation. A correct answer would be that encapsulation bundles data and methods within one unit, kind of like a class. An example of how encapsulation would be implemented could be using getter or setter methods, with an access modifier to define the visibility and accessibility of classes. This is correct as users of the class don’t learn how the class store’s the data, and the developer can change the data in the field without forcing anyone else to change their code.

Question 2. My answer was very vague and was not really the purpose of interfaces. A correct answer would be that interfaces are used to capture similarities between classes , without changing those classes relationships. Another way to describe it would be a structure that allows the computer to enforce certain properties on a object(class).

Question 6. The answer I gave was wrong because public is not describing everything including the assembly, but everything in the project, folder or solution. The correct answer would be the property can be accessed from anywhere. This is correct, as any class or file can access the property that is within the same namespace.

Question 7. The answer I gave was wrong, as the composing class creates the instances of the class composing it, and likewise are destroyed when the composing class is destroyed. That is why the correct answer is a strong has-a relationship, as it has a reference to the other classes.

Question 10. The answer I gave was wrong, as the aggregate class has references to the other classes. The correct answer is a weak-has-a. This is because the aggregate classes are collected by the aggregating classes and can exist on their own.

Question 12. The answer I gave was partly wrong, as you can’t set Calories. A better way do it would instead only have cal, and that have it in the Assert.Equals(cal, lv.Calories);

Question 14. The part of the question that I got wrong was Name, Price and Calories. As they are declared abstract in the base class I needed to provide something in the declaration that I didn’t. A correct answer would be to declare them with the override keyword. This allows us to define an implementation for the abstract class.