# **Exercise #1**

## **1.1**

For this particular scenario, I think the best design pattern would be the Builder Pattern. Why? Well because, at the end of the day, all the multi-course dinner objects are just slight variations of each other.

Starting with the base Meal object, we can enforce that a Meal must have at least one Course, and vary this course between each of the options (starters, main, desserts), additionally, with the Builder pattern, we could allow two or three course Meals, simply by adding the additional Course Options via the Builder. In fact, this would even allow us to make Meals with two Courses of the same type (two starters etc..), or even more than just two.

## **1.2**

This is clearly a situation to employ the Adapter pattern, as we can bring in the output for the US app, in MPH, and parse it through our Adapter to get the new output in KM/H. This would be quite simple to implement, as the change between MPH and KM/H is always the same, and can be obtain using simple math.

## **1.3**

This is again, a clear cut scenario, as what we need here is an Observational Pattern. The “notification app,” only really needs to keep an eye on the bid, and compare it to the last bid the bidder in question made, and if the current bid is higher than the bidder's bid, notify them.

# **Exercise #2**

Since we know that the team’s monthly cost is $17,000 ($8,000 + $6,000 + $5,000), and that the team, on average, deals with 8 Functional Points per month, then if we divide 352 by 8 we get 44 months, so, 44 times $17,000 gets us $748,000 in total.

# **Exercise #3**

First we can calculate the UFP to do that we follow the equation (adjusting for the two user inputs with high complexity):

From here, we can calculate the adjustment factor:

With this we can finally get our result: