1. Strategy Pattern

Problem:

The strategy pattern is ideal for situations where a set of objects should be interchangeable. In this project, the .xml layouts displayed depend on the type of the map marker clicked. (Figure 1) Hence the *ViewFacilityActivity* object should be interchangeable depending on what the user clicks to facilitate the upgrading of the application in the future.

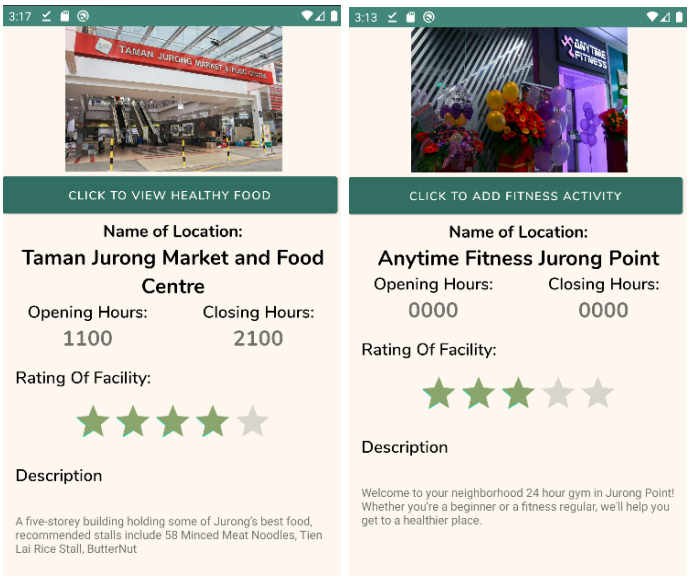

Figure 1: Layout for eatery (left) and fitness facility (right)

Solution:

The *ViewFacilitiesActivity* interface is created to interact with the explorer fragment boundary class. (Figure 2) It allows switching between the two different activity objects, namely *ViewFitnessActivity* and *ViewHawkerActivity*, as they implement the *onCreate()* function in the interface. Therefore, during runtime, the appropriate *OnCreate()* method will be executed depending on the marker chosen by the user. This enables easy extension of new facilities in the future. For example, if a new *ViewMedicalClinics* activity is to be added in the future, the new activity object can also implement the *onCreate()* function in the interface. Minimal changes are required to be made to the explorer fragment because the implementation of the *onCreate()* method is encapsulated.

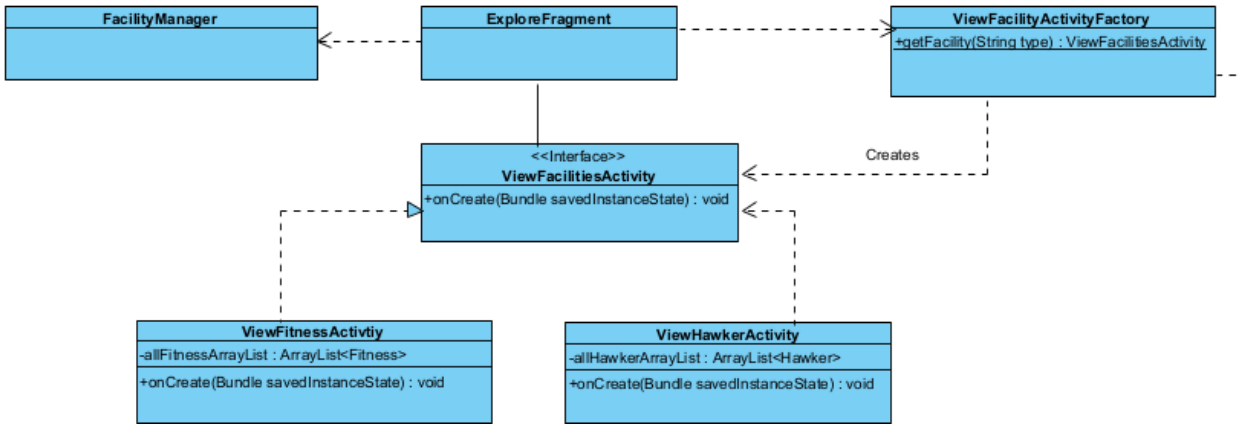

Figure 2: Class Diagram that focuses on *ViewFacilitiesActivity* interface

2. Factory Pattern

Problem:

Strategy Pattern is insufficient as the particular *ViewFacilitiesActivity* object to be instantiated will only be known at runtime.

Solution:

The factory method was adopted by defining a *ViewFacilitiyActivityFactory*, which consists of a static getFacility() to instantiate the different *ViewFacilitiesActivity* objects. (Figure 3)

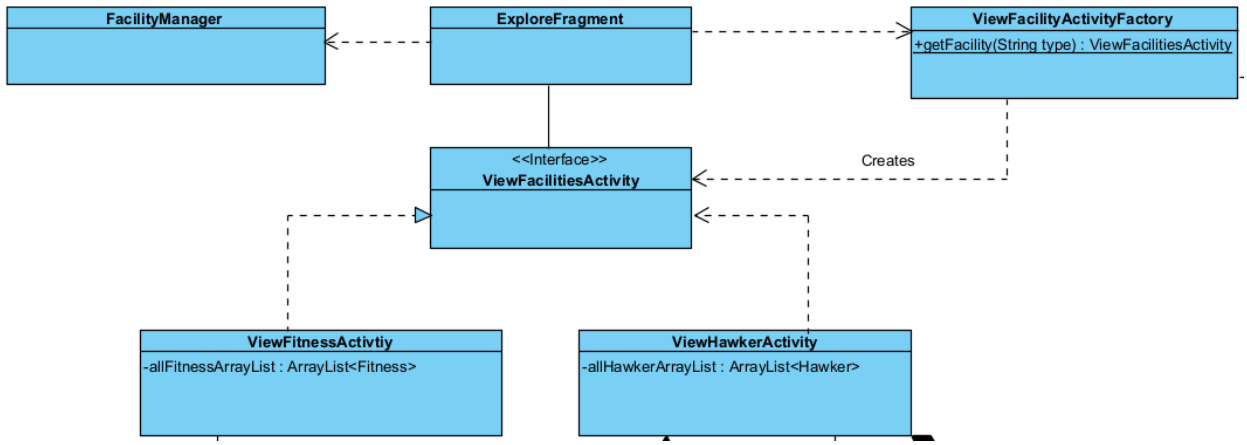

Figure 3: Class Diagram that focuses on *ViewFacilitiyActivityFactory*

During runtime, when the explorer fragment wants to use a particular view class, it will call the static method to create the object and retrieve the concrete class. The context can then interact with the concrete product but using the methods of the interface. This reduces the amount of code changes required when a new facility is added. By localising the logic to instantiate the view object, the class selection and object creation are decoupled from the explorer fragment where the object is used. This allows for greater flexibility in object creation.