THE UNIVERSITY OF SYDNEY

# SOFT2201/COMP9201 Tutorial 12

**Exercises Review**

This tutorial is mainly focusing on a review of some weekly exercises you have done for design principles and design patterns. It is also a good chance for you to communicate with your tutors if you have any questions on them. None of these questions are new. You can find more details of each weekly exercise on canvas according to the week number.

## Question 1: Design Exercise on week 3

You will need to construct a UML class diagram, assigning responsibility to elements identified. After you have finished mapping the UML class diagram, discuss and rationalise your decisions with your tutor.



You are designing a point of sale system for a bike shop, the bike shop sells variety of bicycles such as mountain, commuter, road, single speed and electronic bikes. A customer is able to purchase a bike, services (such as repairs and maintenance) or other products such as a tubes, locks, grips, helmets and lights.



To ensure the system can be audited and for orders to be managed, purchases are grouped by an order, one or more items that are purchases are incorporated in an order. Each order belongs to a customer. Each order has a total amount to be paid, payment method (BankTransfer, Cash, Credit) order number and payment date.



## Question 2: Factory Method Exercise on week 5

Factory Method specifies construction of multiple related objects under a single method, the object can decide what object to create and how to create it and return to the caller. The object created is utilised through a common type that all objects created from the factory method type share.

Using your knowledge of factory method, refactor the VehicleApplication codebase to utilise factory method pattern.

## Question 3: Strategy Exercise on week 6

One thing to note is that behavioural patterns encapsulate information required to act in different ways. For strategy pattern, selecting an algorithm (or method) based on run-time instructions, this information is commonly encoded within the invoking method.

Using your knowledge of strategy, refactor the StrategyBalls codebase to utilise strategy pattern.

## Question 4: Observer Exercise on week 9

Observer is quite different from the structural pattern Adapter because it is something to include 'in advance', rather than as a modification to an existing system like Adapter - though it can be involved in extensions or refactoring as well. Observer handles issues of deep and complex coupling, not by removing the coupling (the line on a UML diagram will still be there) - but by lessening how involved that coupling is (the technical term for this is improving the 'connascence' between the classes).

Using your knowledge of observer, refactor the Coloured Balls codebase to utilise observer pattern.

## Question 5: Prototype and Memento Exercise on week 10

Prototype is a pattern that is easy to explain in concept, but very easy to implement poorly (as is Memento). In order to meet the benefits the lecture describes for this pattern it is necessary to have a deep understanding of how memory, scope, and variable storage work in the programming language you are implementing it in (if you had trouble in the early weeks of this unit a review of this material would be useful).



Memento is somewhat similar to Prototype in its internal structure, but solves a very different problem. Rather than storing different versions of classes, Memento stores different versions of the internals of an object. In our previous example, if we were swapping Strategies, or changing strategy states, Memento would store deep copies of those strategies (unless they were immutable, in which case they could be shallow copies) in an encapsulated way, ready to be swapped back into our ComplexObject to revert it to some previous state.

Using your knowledge of prototype and memento, refactor the Coloured Balls codebase to utilise these two patterns.