SWEN30006 Software Modelling and Design
Workshop 4: Fundamental Design Patterns – Partial Solution

School of Computing and Information Systems
University of Melbourne
Semester 2, 2019

Fundamental Design Patterns In this workshop we provide written discussion as to the decisions that should be made from a GRASP point of view, but have omitted the actual design class diagram, because the choices of method names, arguments etc are all so varied that it makes no sense to provide a particular diagram as the solution. You should review your diagram you created from the workshop and see if it agrees with the principles discussed below.

Exercise One - Determining Experts

Your first exercise is to use the *Information Expert* pattern to determine the appropriate design for the following actions: Determining the highest bidder on an auction at any given time. - The information required to determine the highest bidder is the list of all bids, and the users associated. - The expert with respect to this information is the Listing.

Finding all the items within a subcategory. - The information required to find all items within a subcategory is the list of all items which belong to a given subcategory. - This could be retrieved through the Item class (retrieving all items and sorting), or by retrieving the items through their relationship with the SubCategory class - The SubCategory class has the most direct access to this information.

Retrieving past transactions for a user. - The information required to retrieve all past transactions lies with the User class only, as we need knowledge of both the sold and bought transactions. - The expert is therefore the User class.

Exercise Two - Handling Creation

Applying the *Creator Pattern*, assign the appropriate functionalities for creation of the following objects: ***Transactions*** In creating transactions, we need information about the listing itself, information about the buyer (the highest bidder) and the seller, along with the payment method and status of payment.

The creator pattern states B should create A if:

- Instances of B contain or compositely aggregate instances of A
- Instances of B record instances of A
- Instances of B closely use instances of A
- Instances of B have the initializing information for instances of A and pass it on creation.

We can see that the Transaction class will be closely used by a variety of different classes. It is closely connected to the User class, so it could be created by the User class. Every transaction, however, has associated buyer and seller users, which raises the question: which user would create the Transaction? A transaction will involve transferring payment from buyer to seller and so will involve information (part of payment method or not covered by domain model?) from both buyer and seller. However, it is the buyer who effectively triggers the transaction by effectively authorising the payment. Whatever the choice, we don't want the same functionality duplicated, and we don't want ambiguity in our control flow.

There is however, an other option which is consistent with the creator pattern; the Listing could create the Transaction. The Listing has access to all the initialization information for transactions, including both the buyer and the seller. In addition, while the buyer effectively authorises the transaction, the buyer may do this, for example, at the time they join the Online Marketplace by agreeing upfront to a transfer for any winning bid they make. Having Listing create the Transaction makes the responsibility assignment less dependent on the authorisation process and allows for the update of both the Buyer and Seller objects at the same time to keep the data consistent. There is no need for a decision between the two User relations, and there is clarity around the decision reasoning. This results in the nice ability to have Listing finalize itself (and generate the Transaction).

Bids The case with bids is clearer, the clear choice being to make the Listing create the bids. The Listing closely uses, aggregates and has most of the information required to create a bid. You could argue that a User also aggregates and has some information (namely the bidder), but the User has no need to use the bid after creation, other than for interest purposes (i.e. check the previous bids) and so the responsibility is less clear from the User perspective.

Exercise Three - Coupling and Cohesion

The main point in this exercise is to realise that not all of the GRASP design patterns are mutually exclusive, and that in choosing to apply a given GRASP principle, you will most likely be making a trade-off and will need to be able to justify that design decision when working in software design group.

You may want to pay particular attention to the coupling created if you choose to assign the responsibility of creating Transactions to a User class, which is now coupled with the Listing, Bid and User classes. There is also possibility to reduce cohesion depending on your assignment of the creation behaviour for Transactions. If you chose to assign this responsibility to the User class, you will be now spreading the behaviour required for interacting with transactions between the Transaction, Listing and User class. This reduces cohesion for that set of functionality. If however you choose to assign the responsibility to the Listing class, the behaviour is now only spread between the Listing and Transaction class. Therefore, the system has higher cohesion with that choice.

Exercise Four - Controllers

Here again your choice depends on the decisions you have made in Exercise Two. If you have assigned the behaviour to Listing and to User (for creation of listings and bids respectively) then you may want two controllers to maintain cohesion and separation of concerns. If you have all of this functionality on Listing, then you will probably only need one.

Your controllers should offer the appropriate methods (i.e. `createListing` and `makeBid`) for the required functionality and you should remember to include all required input data from the user to perform these actions.