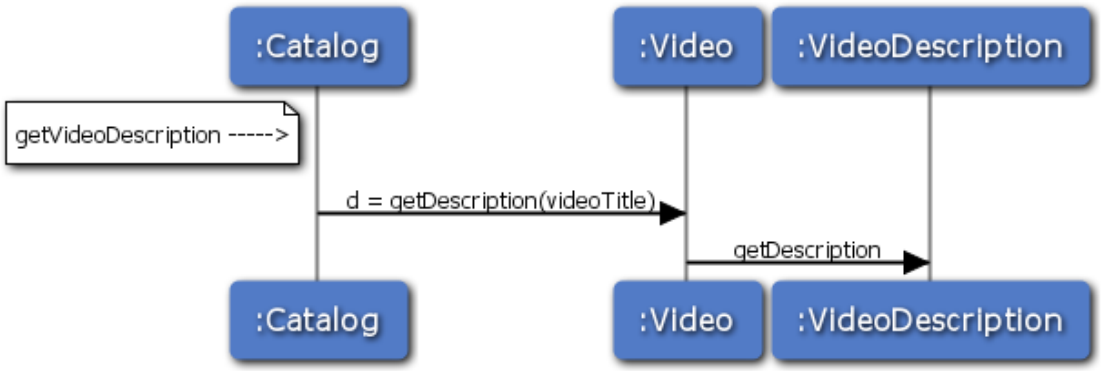
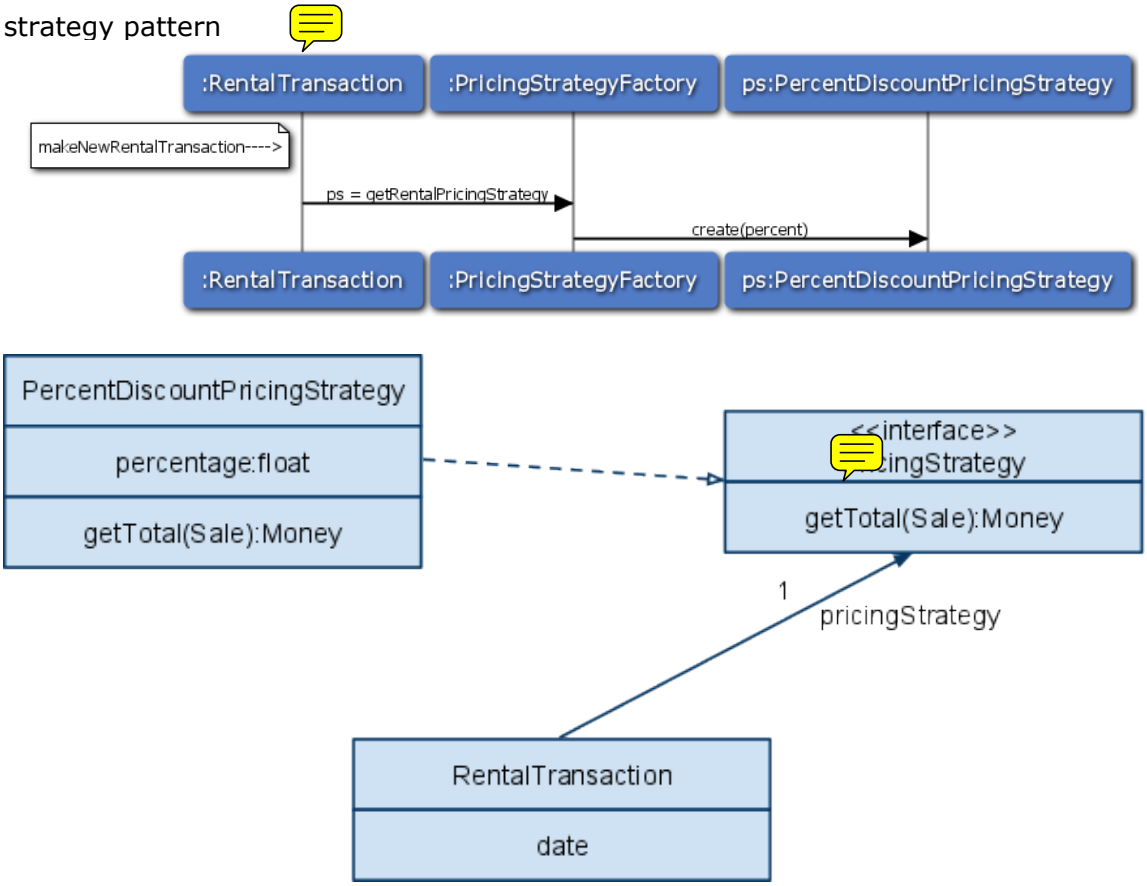
David Pick
CM 2403

1. Adapter pattern



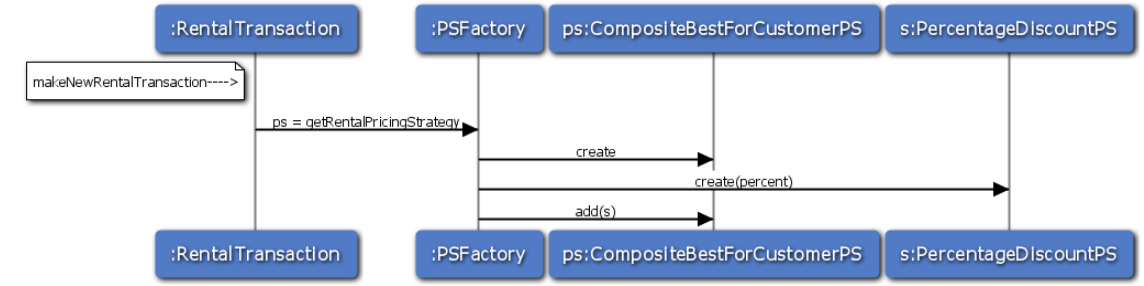
In this example Catalog is used as an adapter for getting Video information from many different kinds of videos. It makes sense to have an adapter for returning information from many different kinds of videos, such as movies, tv shows, or documentaries since it is possible to implement each kind of video as a different class. By having an adapter every class that access' video information does not need to know about all the different classes. This adapter also makes it much easier to add another kind of video class, should the need arise.

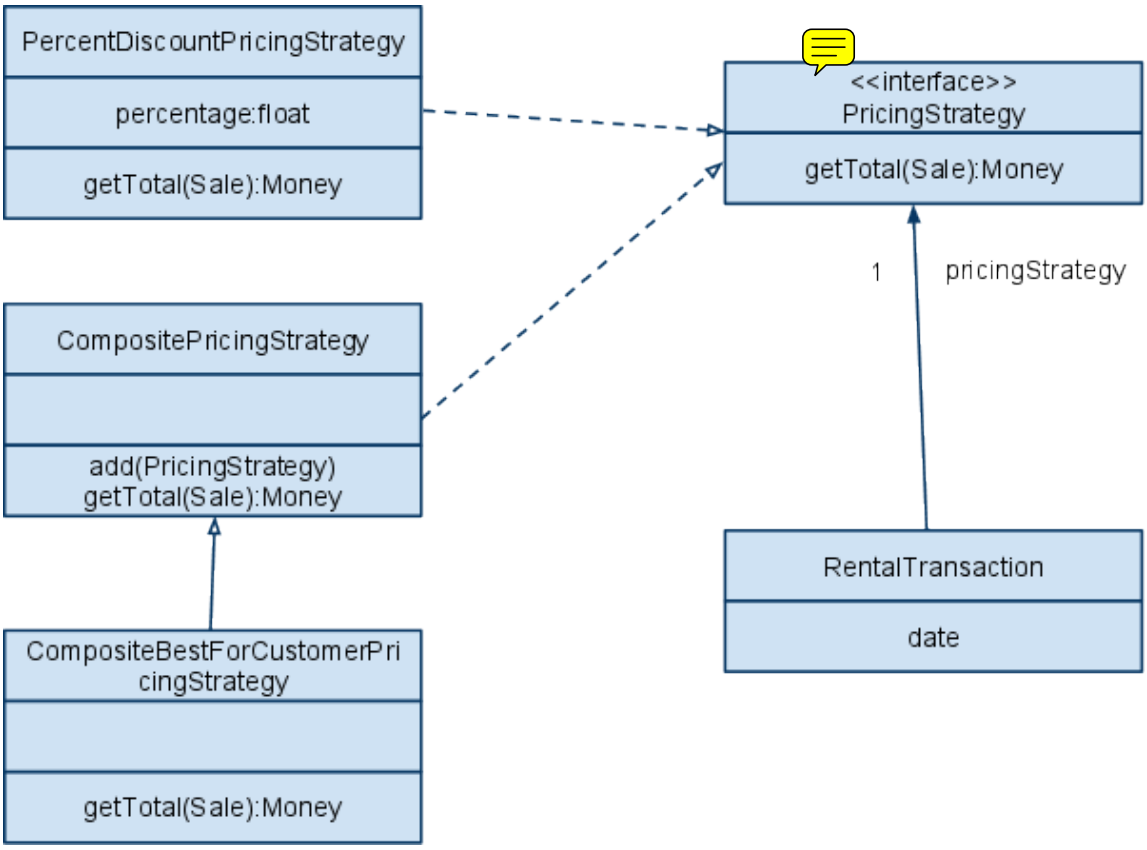
2. strategy pattern



Since it is very likely that there will need to be different pricing strategies applied upon checkout, such as discounts, coupons, or holiday specials. It makes sense to create a single interface PricingStrategy, this way it is very easy to add another strategy, and the RentalTransaction class does not have to do the work of calculating the total cost of a rental.

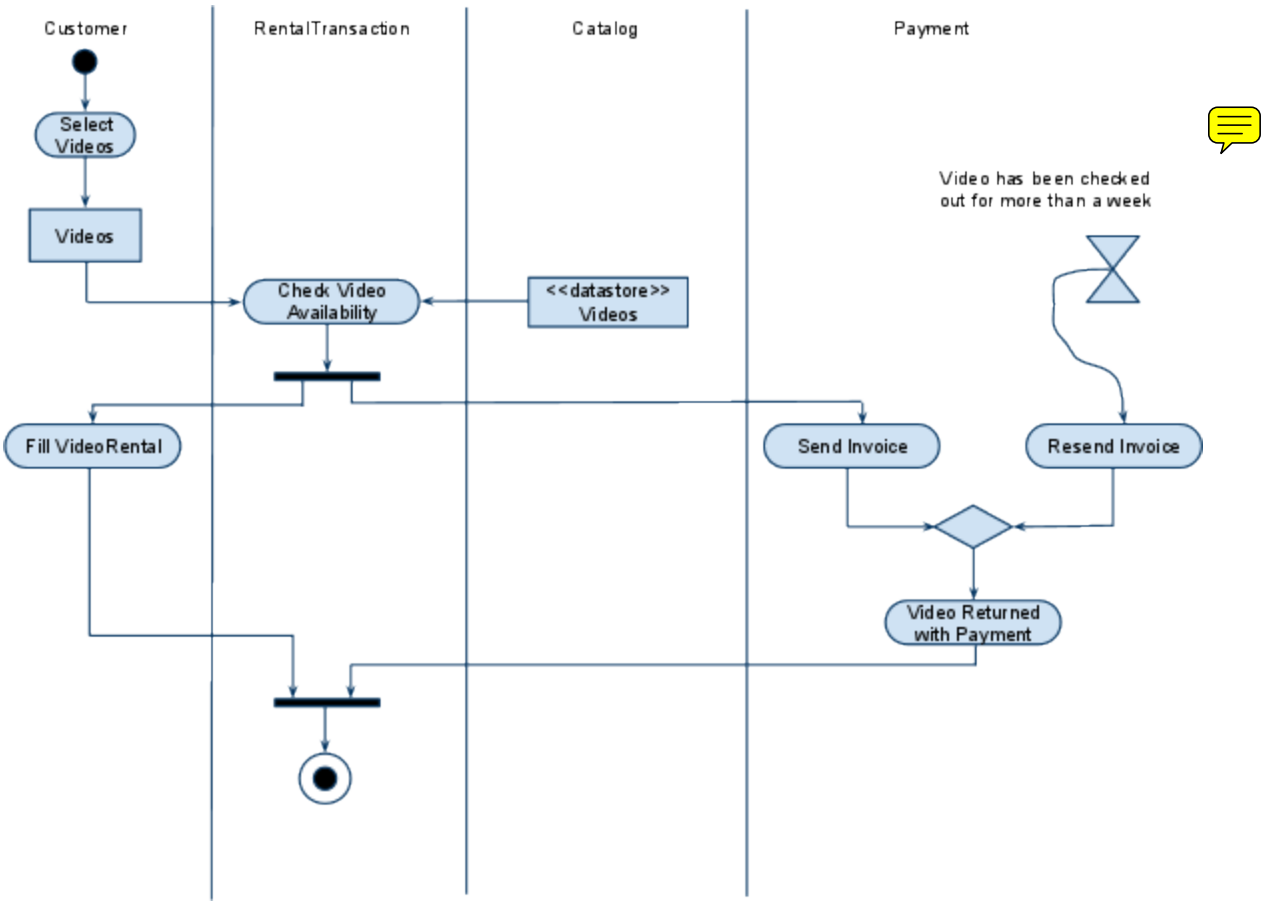
3. Composite Pattern





With multiple pricing strategies, it is very likely that they will eventually conflict. This can be handled by creating a composite pricing strategies, such as CompositeBestForCustomerPricingStrategy, which will take all the different pricing strategies into account, and determine the best possible price for the customer. Or if need be, we can still just assign a simple PercentDiscountPricingStrategy to the RentalTransaction.

4.



This activity diagram shows the full life cycle of a video rental. It starts by having the customer select the videos they would like to rent, it then checks if they are available in the catalog, if they are the customer is allowed to rent the videos, and an invoice is given to the customer. If the videos are not returned within a week, another invoice is sent, until the videos are returned and payed for.