Hitesh Sapkota

Liskov Substitution Principle:

In design model, we fully considered Liskov Substituiton Principle by figuring potential exceptions that might come while extending one class from another . We are planning to consider all exceptions before extending to the subclass. We are doing so in order to ensure that our model is cohered with LSP. So, in my opinion our design considers LSP principle as much as possible. The potential scenario that might violates LSP in our model is described below. After that I provide potential solution that can prevent from violation of LSP principle.

At first, we planned to make Disk Movement as a subclass of a superclass King Movement. Here, Disk Movement class takes certain input(always forward status) and increases the position by one. Furthermore, King Movement class can take two status (forward and backward) and increases/decreases the position according to input status. I think extending Disk Movement class from King Movement class violates LSP. Because whenever someone tries to move king in backward direction and when this is referenced to backward movement method of subclass Disk Movement, subclass always increases the position by one irrespective of input .This causes conflict with the behavior of same method of superclass. Figure below shows the violation of Liskov Substitution Principle.

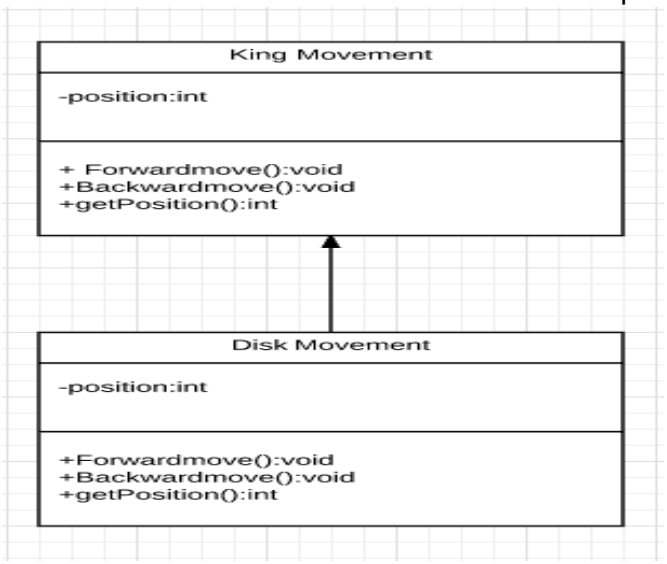

Fig: Diagram Showing violation of LSP

There might be different possible solutions. One is that, Disk Movement can only be considered as a sub class of superclass King Movement when there is forward movement request as an input. On the other hand, when backward status request comes as an input to King Movement, Disk Movement no longer will be the subclass.

User story contains Disk Movement and King Movement two different stories in game play option. So, we can modify our design in such a way that Disk Movement class is inherited only for forward move request in King Movement class. In case of backward move, Disk Movement won't be the subclass and hence method of King Movement can be called. This makes model adhered to Liskov Substitution Principle.

Hitesh Sapkota

Interface Segregation Principle(ISP):

In our project, we tried to make each interfaces as specific as possible. We are planning to make each interfaces in such a way that they contain only one specific method. Furthermore, those specific interfaces can be implemented from another interface that implements all related interfaces. We have made design model in such a way that it confirms all five SOLID principles as much as possible. So, we can say that our model is more cohered with ISP.

While talking about our project, we were planning to make all three methods: view history, watch live and play game methods from a single interface called player. But, the problem is that player interface becomes more fat and hence violates the ISP principle. Possible scenario where violation might occur is described below:
Suppose someone is playing game for the first time. In such case, the player does not contain any history record. Despite the fact that he does not contain have any past records, he might be assigned to the method related to view history . This is completely inappropriate for him at least for the first time. This means our design violated Interface Segregation Principle(ISP).

In order to solve this problem we can modify the design in such a way that we can create three different interfaces related to view history, watch live and play game. Furthermore, we can implement all those interfaces from the interface named player interface. The possible solution is illustrated by diagram shown in figure below:

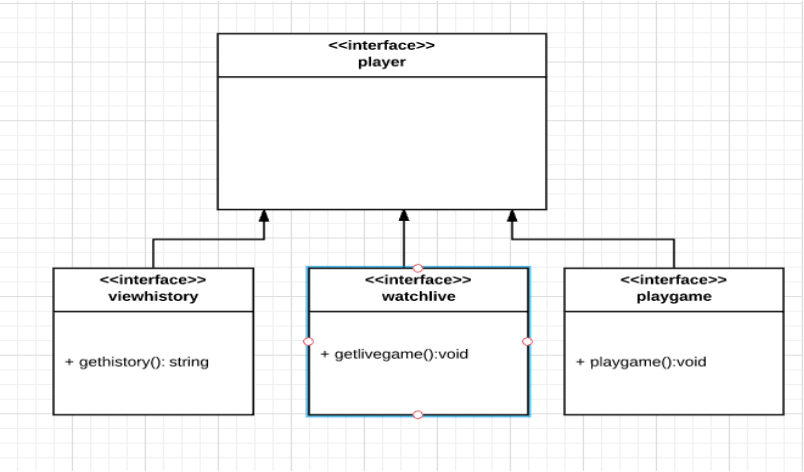

Fig:Figure showing confirmation of ISP .

Similarly, one player(while playing game) might be assigned to the interface called movement that contains two method Disk Movement and King Movement. Actually, in this case, player acutally deals with one of these two methods. So, this flaw in our model also leads to violation of the ISP.
We can solve this by making two different interfaces related to Disk Movement and King Movement. Furthermore, we can create Movement interface that implements both Disk

Movement and King Movement interfaces. In doing so, player deals with only one interface and does not depend on another interface he/she is not dealing.
In user story we have kept three options after Sign In: view history, watch live and play game. We are going to modify design in such a way that for old player, he/she will have access to all three option, whereas for new players he/she will only have two active options: i.e. watch live and play game.