**The State Pattern**

**Object Oriented Goal**: Encapsulate what varies. Favor composition over inheritance. Strive for loosely coupled designs. Depend on abstractions. Do not depend on concrete classes.

**Definition**: The state pattern allows an object to alter its behavior when its internal state changes. The object will appear to change its class.

**Implemented on classes**:

1. OrderStatus (Context)
2. OrderState (Interface)
3. Started
4. Processing
5. Complete
6. LeftForDelivery
7. Delivered

**Design Snapshot**:



**Design Explanation:**

There is a functionality in our design in which the user can track the order status in the app and the app updates the order status whenever the order makes a transition from one state to another. We have 5 different states for our order.

To implement this functionality, we have used the state design pattern and its implementation in our design is as follows:

* The OrderStatus is our context class and it can have a number of internal states. This is the class where all the changes of the events will be done. Every time a change of event occurs, the current state is changed and a new state is delegated to the current state object. This class has reference to all the concrete state objects and thus it makes easier to delegate the objects to the current state.
* The OrderStatus class also has setState method which will change the state to the required state as soon as an event occur. The different events have been implemented as methods which will call the particular State object to perform its functionality.
* The OrderState is the State Interface which has some methods which needs to be implemented by the concrete state classes. This interface defines a template for the concrete states. This methods are basically the events which can occur on the order status and thus will change the state of the order.
* The concrete state classes are Started, Processing, Complete, LeftForDelivery and Delivered. These classes implement all the methods from the OrderState interface and implements the method according to their own state.
* The concrete states handle requests from the context. Each concrete state provides its own implementation. Thus, when the context changes state, its behavior will also change.