Bad Practice :

```
data class Order(var orderedItems : MutableList<OrderedItems>, var amount : Float,
                    var providerId : Int, var customerId : Int,var address :String , var status :
StatusType = StatusType.BOOKED){
   private var orderId : Int =0
   init{
      orderId = OrdersList.getOrderListCount() + 1
   }
   fun getOrderId() : Int{
      return this.orderId
   }
}
```

In the above data class Order , fields are declared with var, var is mutable. This leads to changes in states of an Order instance. The field except status are not needed to change in an order class. So other than that, all other are needed to changed as val.

Also orderedItems has a type of MutableList. It is not a good way to denote ordered Items as MutableList. Although we denoted the orderedItems as val, there is a chances for changing the states of orderedItems.

NOTE : If val holds mutable objects, then there is a chance of changing its state.

Like , currentOrder.orderedItems.add(OrderedItems(....))

Good Practice :

```
data class Order(val orderedItems : List<OrderedItems>, val amount : Float,val providerId : Int,
                    val customerId : Int, val  address : String , var status : StatusType =
StatusType.BOOKED){
   private var orderId : Int =0
   init{
      orderId = OrdersList.getOrderListCount() + 1
   }
   fun getOrderId() : Int{
      return this.orderId
   }
}
```

References :
https://hub.packtpub.com/how-to-implement-immutability-functions-in-kotlin/

https://artemzin.com/blog/kotlin-val-does-not-mean-immutable-it-just-means-readonly-yeah/

https://proandroiddev.com/kotlin-for-beginners-immutability-and-the-value-of-val-78ab45b60b57#:~:text=Because%20you%20are%20forced%20to,possible%20states%20within%20your%20app.

https://appmattus.medium.com/effective-kotlin-item-17-minimize-mutability-7f68de945ea3