Visitor Design Pattern:

We implemented visitor design pattern to separate the operational logic from the objects. We don't want to have to modify our item and itemContainer class every time we want to add new functionality so we will create a NodeVistor interface which will then have an abstract method called visit which will take the object that we want to add said functionality to, so in this case it will be a Component object since this is the parent class of item and itemContainer. Then we would create classes for every type of operation that we want to add which in this case is the total price and market value. So, both of these classes will implement the NodeVisitor interface. They will then implement the method within the NodeVisitor interface and perform their respected functionalities for each class. Then within the Component class we created an accept method which takes in a NodeVisitor object. Thus, we must then implement the accept method within the item and itemContainer classes.

Adapter Design Pattern:

The point of adapter design is to make a link between 2 incompatible interfaces. For us this meant wanting to adapt the functionalities we have for drawing the drone to the dashboard into the physical flight of the drone itself. This meant creating an interface called DroneAnimation with 2 methods scarnFarm and visitItem. Then we created a DrawAnimation class which implements the DroneAnimation interface and thereby implemented the 2 methods within the interface. And this class's implementation of the methods included making the drone image move on the farm dashboard. Then we created another class called TelloAdapter which implements the DroneAnimation interface and thus will implement the 2 methods within that interface. The TelloAdapter class takes in a TelloDrone object within its constructor and then uses that object to implement the 2 methods from the interface. This would then allow us to make a link between 2 incompatible interfaces.