I chose to include enums to ensure that only the required types are passed as values. My DroneMove class represents the possible moves a drone can make. I throw an error if the distance is not positive. I have a DroneTricks class where I add the moves of the trick to the moveSequence based on the trick inputted by the client. I interpreted TAKEOFF and LAND tricks as always being the same, whereas the pucker trick distance and speeds can be varied by the client. This design works well with the three tricks I am using, but a trade-off is when considering more tricks, it could be confusing to the client to override the constructor with the correct parameters of the different tricks. Another design option would be to represent each trick as a separate class. I have a Flight class which takes in an array of DroneTricks and a name (to easily compare the flights). Finally, my Drone class extends the Movements interface, where the execution of a flight, move, or trick can be executed.

The fields in all classes are final. I chose to use the Comparator to order the flights because there are two different flight comparisons to consider. I chose to use anonymous classes for my Comparators because it makes my code more concise. I created three flight objects in my RunDrone class which give different orders based on what is being compared. When comparing flights, I am assuming “unique movements” means unique number of movements.