For my design I decided to use polymorphism and create a second interface Trick that extends Movement so that it is easy for future new tricks to be created and so that they have all the necessary attributes and methods. I kept the Movement interface to define moves generally as well as tricks. I made the BasicMove class to represent basic movements that make up tricks. In my design I was sure to keep all my classes immutable. I used private final fields where necessary and only returned copies. I used enums to represent Direction, Speed, and VideoFormats so that there are only set options and so that they are immutable. In all Trick classes and the BasicMove class, all attributes are final so as to keep them immutable. In my Flight class I decided to use a List of tricks that clients can add to one trick at a time to create Flights, in order to keep the class immutable I return only copies of the list. For the Drone class, I decided to group together flights that can be executed individually, this allows the client to compare all the flights for a given drone. I used separate classes to define the functionality for each comparator. RunDrone then uses Collections.sort and whichever comparator to sort flights. A client must create tricks, then make flights from them, then add the flight to a drone to execute. This ensures that a client can’t execute a flight without a drone. A client can also execute BasicMoves and tricks not in a flight using .execute(Drone d), this ensures that movements can’t be executed without a drone. 