This lab implements Djikstra's Algorithm for Shortest Paths. Our Minimum-First Priority Queue is implemented using an Arraylist that contains Handle objects, with the first slot being set to null. Handles themselves are parameterized with type T, and can hold an arbitrary object of that type, and an associated key. Further, each Handle maintains maintains the value of its index in the ArrayList in order to allow constant-time lookup. All Queue operations are implemented as described in class. The Insert operation returns a new Handle for the object. A swap method, used to swap the Handles at 2 provided indices, and a Heapify method similar to the one described in class are also implemented and called by other methods.

The Shortest Path Algorithm required minor changes to be made to the provided Vertex class. Each Vertex object now contains a Handle Object that is set in ShortestPath to its own Handle, and a parentEdge, which contains the edge from the parent node to this vertex. If this vertex is the start, parentEdge is set to null. Djikstra's Algorithm is implemented as it was described in class. A slight change was made to implement the extra-credit feature. Rather than just taking into account the length of the flight, we now take into account the layover and schedule between the flights. We do this by maintaining a flight arrivalTime, initially, it is set to the startTime variable provided, that is updated with each path examined. We calculate the layover as the difference between the arrivalTime and the departure flight's start time. Given that we have a 45 minute minimum layover, if this difference is less than 45, we add another 1440 minutes (or one day) to the layover time to reflect the fact that only the same flight the next day could be taken. If a startTime of 0 is provided, we maintain the original, schedule-less implementation.

In order to return our shortest path, we create an ArrayList of edges, and follow the destination's parent node to the start, adding each path the arrayList along the way. we than place the flight ID of each element in the ArrayList into an Array of the same size in reverse order, in order to have our flights begin at our starting node. We then return this Array.