Overview:

This project will implement a simple airport take-off time slot scheduler. Plane submit requests to take off at a certain time and tell the air traffic controller how long they will need the runway for. This program will process these requests and assign each plane a takeoff time. It will keep track of all the planes waiting to take off, print out the status at each time interval and at the end print out the actual time each plane was on the runway.

Design:

The program will be written in Python. The program will take in input from the user in the form of a text file (.txt). The input file should be in the following format:

ID, Submission Time, Requested Start, Requested Duration

An example would be:

Delta 160, 0, 0, 4  
UAL 120, 0, 5, 4  
Delta 6, 2, 3, 6

There will be a request class or data structure that will contain instance variables to store each of the input values. An input value is defined as one line of the input file. Every request will have an ID, submission time, request start time and requested duration. A request will also have an actual start and actual end time that will be given values as the requests are processed. It will contain getter methods to retrieve these values but not setters for most values as once a request is submitted it cannot be changed. However, there will be setters for the actual start and end times because these may change depending on the plane’s processing status. The class diagram can be seen in Figure 1.

There will be a PriorityQ class. This class will implement the functionality of a priority queue by extending Python’s built in heapq module. The item with the lowest priority will be at the front of the queue and will be popped first. It will give a user the ability to push an item with a primary and secondary priority, as well as an index for the tie breaker. It will also allow the user to pop the item with the lowest priority. In addition, it will have a printQ method for displaying the queue nicely and a QasList method to convert the queue to a list for accessing specific indices of the queue. The class diagram can be seen in Figure 2.

There will be a RequestProcess module that handles taking in and parsing the input. As the input is taken in, new request objects will be created. The objects will be temporarily stored in a PriorityQ so that they can be sorted by submission time. After they have been sorted they will be stored back in a list. After all the input is taken in and stored, the processing of the list can begin. The program will simulate time passing by sleeping for one second intervals. After each second some processing of the planes waiting will happen. The program will calculate and update the actual start and end times for all requests that have been submitted, check to see if any of the requests have been completed (actual end time equal to current time), check to see if any new requests have been submitted, sort the list by request time and then re-calculate the actual start and end times incase new requests have been added. All of this will happen in each time interval. At the end of the time interval the program will check to see if there any more requests to process and will end if there are not. The status of the list will also be printed out on each time interval. The class diagram can be seen in Figure 3.

There will also be a Main file that drives the program. It will create a RequestProcess object and pass the file name from the command line arguments to the program. This allows the program to be run by entering *python Main.py input.*txt on the command line. The class diagram can be seen in Figure 4, but it is empty since there are not actual methods or instance variables.

Figures:

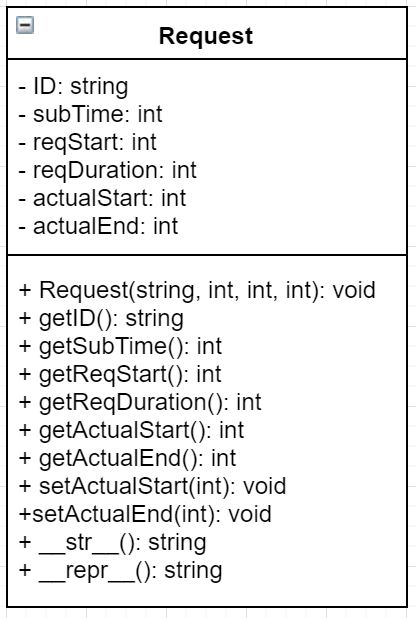
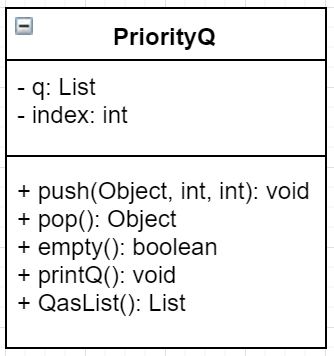
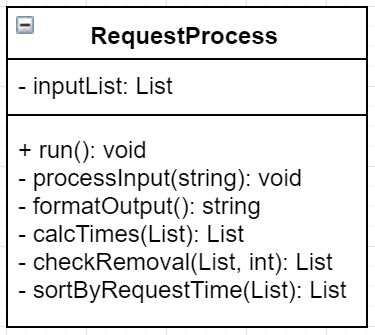
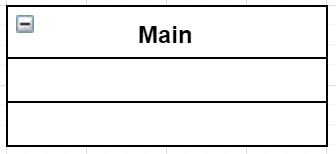
 

Figure 1 Figure 2

*Figure 3 Figure 4*

Change Log:

10/13/16 – 1:14PM – Initial SDD created

10/18/16 – 12:50PM – Updated diagrams, made design more detailed, added PriorityQ class to design