**Problem:**

There is a list of jobs being offered to a freelancer. As a freelancer, he/she needs to select and schedule the jobs in order to gain the maximum profit in a certain duration of time. Since a job has a different duration and deadline, it is a burden for the freelancer to select and schedule all the jobs one by one. A job consists of few attributes as mentioned below:

Job:

* Profit
* Duration (days)
* Deadline
* Profit per day

**Proposed solution:**

Data structure:

A “Freelancer” class consists of two attributes which are “dayList” and “jobs". The “dayList” is an instance of the data structure that was custom created, which is called “DayList”, it contains an array of Day. The DayList class will assist the freelancer to consider whether is a job is suitable to be scheduled in his day list or not. The DayList will also produce a well-defined job schedule that will maximize the total profit from all the possibilities after every time a new job is added.

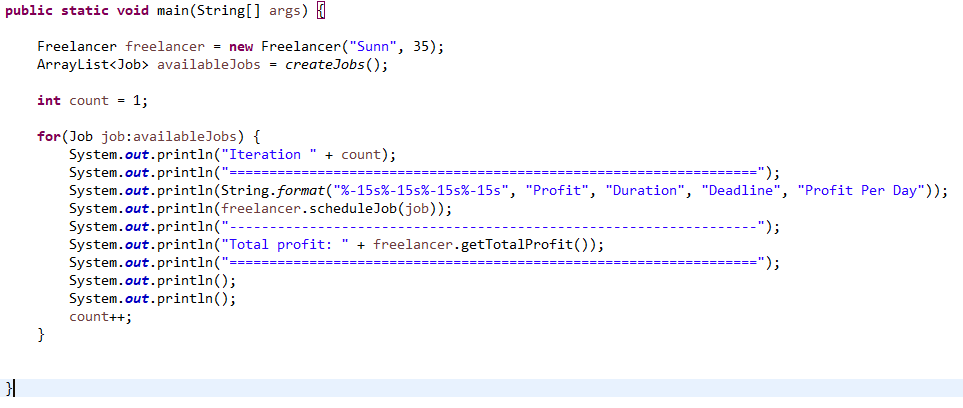
On the other hand, the “jobs” is an instance of the priority queue. The “jobs” is sorted based on the profit per day in descending order to satisfy requirements of the algorithm.

Strategy:

First of all, the highest priority job will be taken from the “jobs”. Then, the duration of the job is checked to make sure that it is lesser than the number of available days, which means that the job is able to fit into the day list.

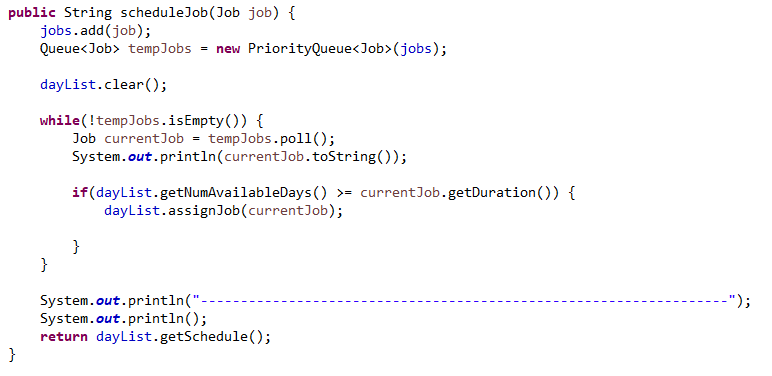
After that, the deadline of the job is checked to make sure that it is able to done before the deadline. The deadline will be the end day, the days before the end day will be checked consecutively for availability for n times, where n is the duration of the job. If all the checked days are available, assign the job to that period of days, else do not assign the jobs to any period of days.

**Complexity analysis of the application:**



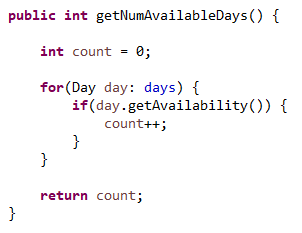
**Time complexity:** T(n) = c \* n = O(n), where n = availableJobs.size()

Since this function only implement a simple for-loop, the equation for the time complexity of this function is O(n), where n = availableJobs.size().



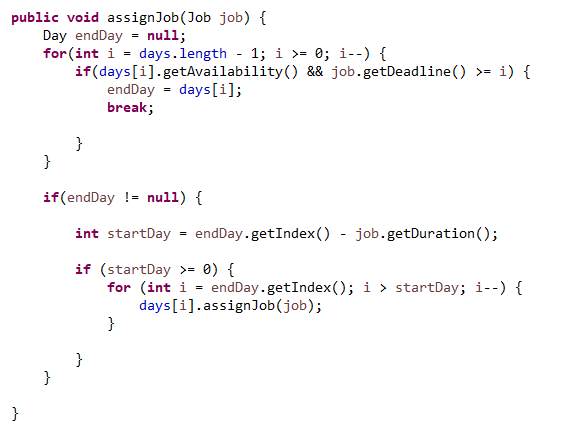
**Time complexity:** T(n) = c \* n = O(n), where n = tempJobs.size()

Since this function only implement a simple for-loop, the equation for the time complexity of this function is O(n), where n = tempJobs.size().



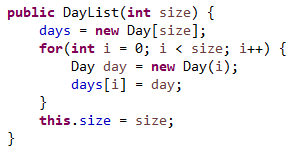
**Time complexity:** T(n) = c \* n = O(n), where n = days.length

Since this function only implement a simple for-loop, the equation for the time complexity of this function is O(n), where n = days.length.



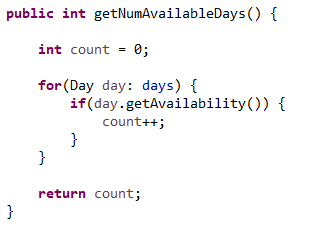
**Time complexity:** T(n) = c \* m + c \* n = O(m) + O(n), where m = days.length, n = startDay

Since this function only implement a sequence for-loop, the equation for the time complexity of this function is O(m) + O(n), where m = days.length, n = startDay.



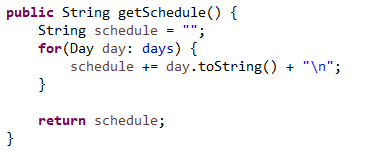
**Time complexity:** T(n) = c \* n = O(n), where n = size

Since this function only implement a simple for-loop, the equation for the time complexity of this function is O(n), where n = size.



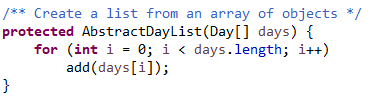
**Time complexity:** T(n) = c \* n = O(n), where n = days.length

Since this function only implement a simple for-loop, the equation for the time complexity of this function is O(n), where n = days.length.



**Time complexity:** T(n) = c \* n = O(n), where n = days.length

Since this function only implement a simple for-loop, the equation for the time complexity of this function is O(n), where n = days.length.



**Time complexity:** T(n) = c \* n = O(n), where n = days.length

Since this function only implement a simple for-loop, the equation for the time complexity of this function is O(n), where n = days.length.

In conclusion, since the loops in this program is based on linear time, the time complexity for all of the functions will be O(n).Therefore, the time complexity increases linearly to n when n increases.