STRUCTURAL OVERVIEW OF OUR APPLICATION

- Slot
  - Slot Unit array
  - static Slot union(Slot[])

- Session
  - start date
  - end date(optional)
  - programs array
    - departments array
      - classroom array
        - classroomId
        - department
        - strength
      - instructors array
        - name
        - department
          - courses array
      - students array
        - name
        - department
        - program
        - courses array
  - courses array
    - void updateStudents(Students[])
      {first update the students array and then computeOverlaps()}
    - computeOverlaps()
      {
       for each course in overlaps{course.overlaps.remove(thisCourse)}
       //update overlaps variable;
       computeAvailableTimings();
       for each course in overlaps{course.overlaps.add(thisCourse);
       course.computeAvailableTimings}
       }
       (this function is going to take a lot of time)
       (can't be avoided. after all, we want a robust checking)
    - computeAvailableTimings()
    - students hash map
      (will be stored as hash map for O(1) insertions and deletions)

(only those students should be added in this array which are present in some department member object of the current session instance)

- instructors array
  (only those instructors should be added in this array which are present in some department member object of the current session instance)
- overlaps hash map
  (list of all courses which has at least one student in common with the current course, this is a private member and will be auto updated whenever students hash map is changed)
  (this member is used to determine available timings for the current course quickly)
- Slot availableTimings
- Slot courseSlot
  (on the frontend, there will be a nice table like interface to select the course, all the timings on which any student of this course has another class scheduled will be blacked out i.e not available for selection for this course's slot)

~~- extra classes~~

~~- cancelled classes~~