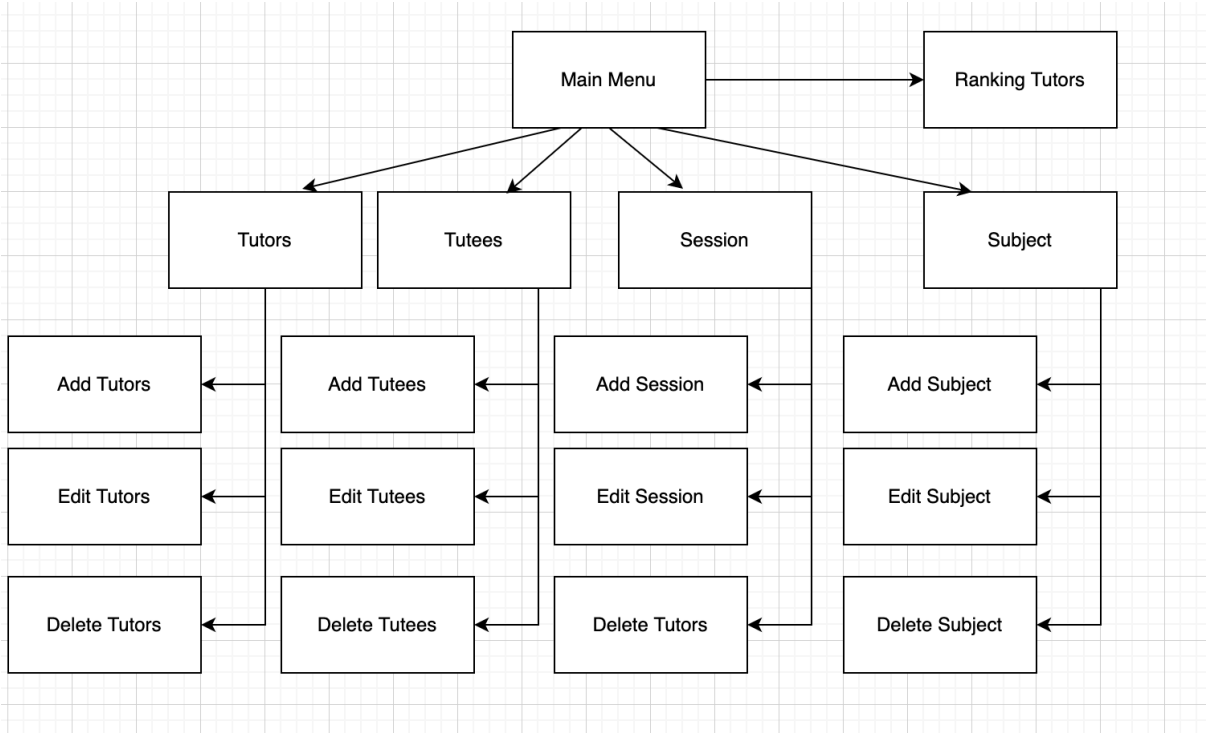
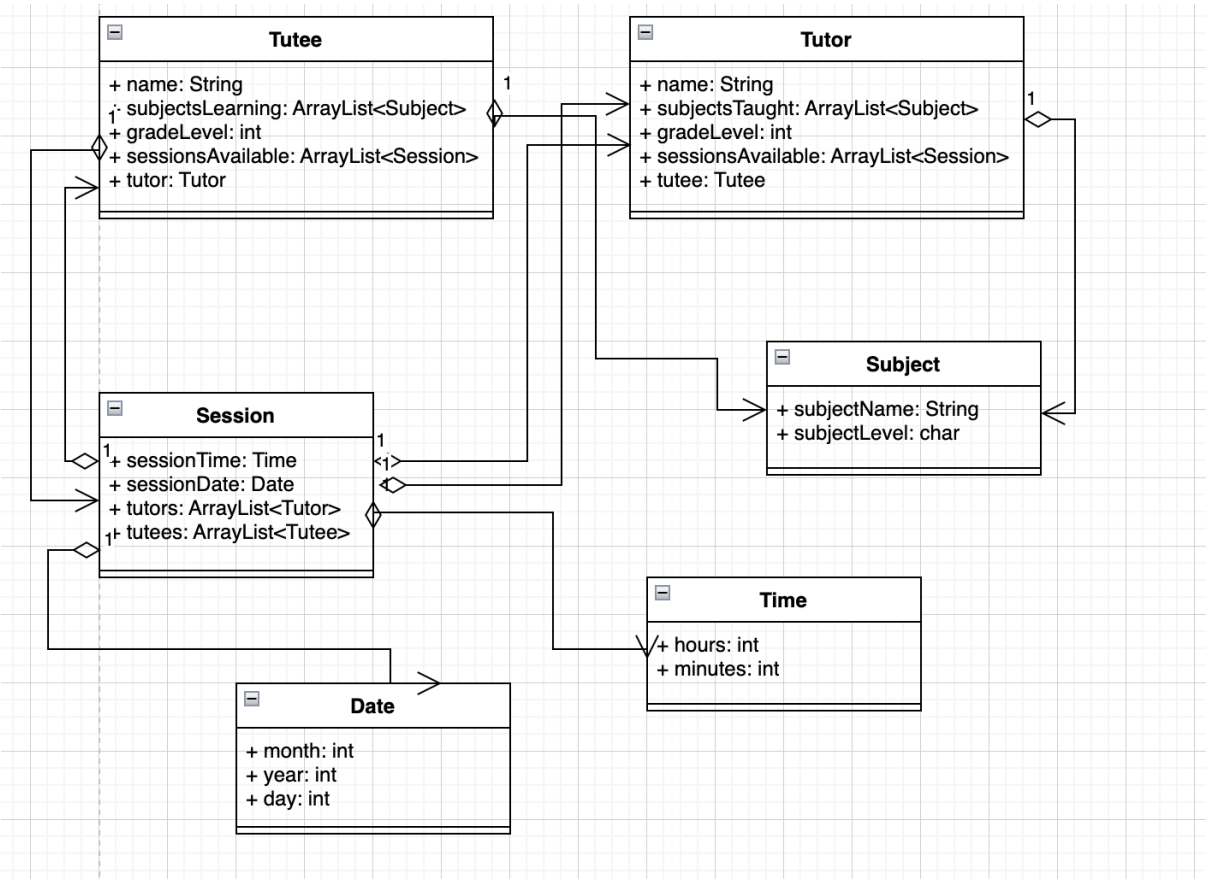
Criterion B: Design

1. **Decomposition Diagram:** Highlight the flow of operation between the different stakeholders in the system and its functionality, easing development later on.

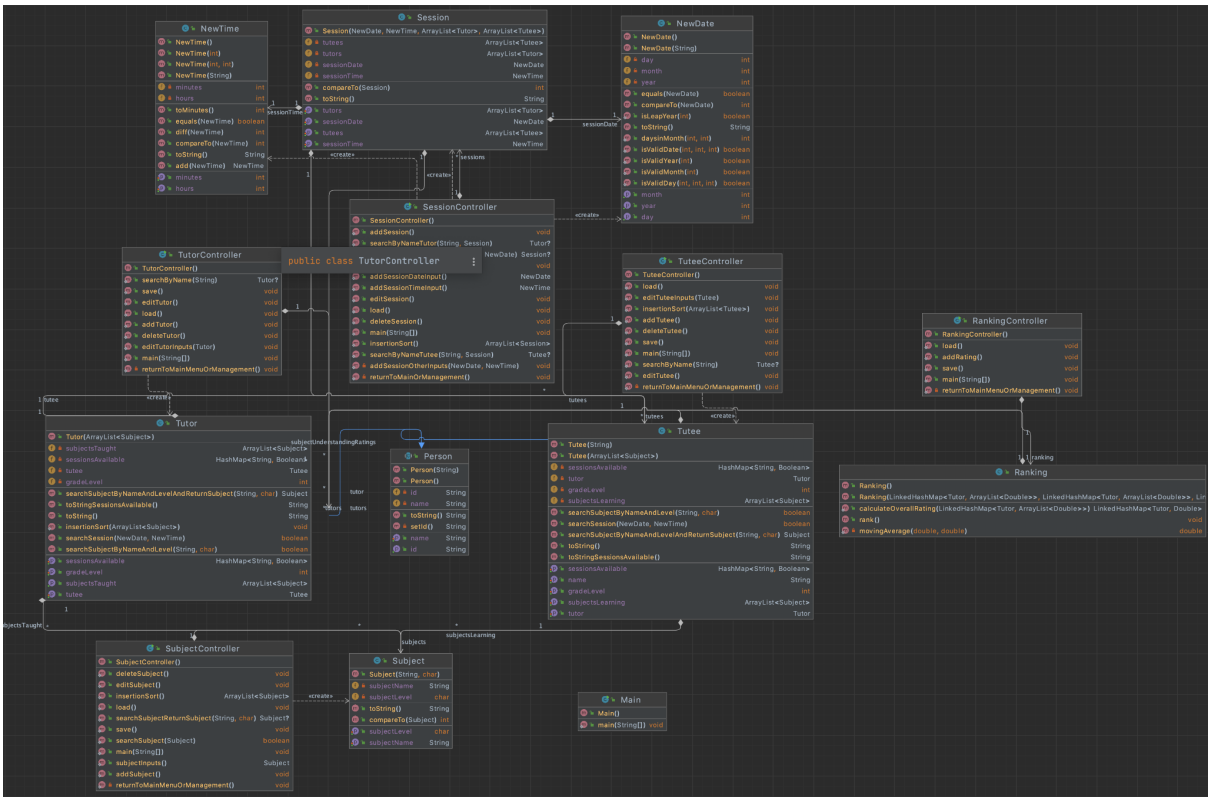


2. **UML Class Diagram:** This diagram highlights my pre-development ideas for my classes and their dependencies, which will be helpful when beginning my development. This includes the Time and Date classes, which are examples from school lectures (Drien Vargas).

Initial Class Diagram



Final class Diagram (after development):



3. **Test Plan:** Planning the different types of validation required for the system

Class: Main
input: int

Test Data Type	Type of Validation	Input Data	Expected Output	Test Passed/Failed
Normal	Range Check: 1-6, Type Check: int	3	Accept the input	Passed
Extreme		76; -9;	Reject and ask to input again	Passed
Abnormal		‘h’	Reject and ask to input again	Passed

Class: RankingController - addRating()
newRating: double

Test Data Type	Type of Validation	Input Data	Expected Output	Test Passed/Failed
Normal	Type Check: double	7.4	Accept the input	Passed
Abnormal		‘h’; true;	Reject and ask to input again	Passed

Class: NewTime
setMinutes: int

Test Data Type	Type of Validation	Input Data	Expected Output	Test Passed/Failed
Normal	Range Check: 0-59	30	Accept the input	Passed
Extreme		76; -9;	Reject and ask to input again	Passed
Abnormal		“27”	Reject and ask to input again	Passed

Class: TuteeController
hours: int

Test Data Type	Type of Validation	Input Data	Expected Output	Test Passed/Failed
Normal	Range check: 11-12	11	Accept the input	passed
Extreme		10	Throw Exception and ask to input again	Passed
Abnormal		“12”	Throw Exception and ask to input again	Passed

Class: TuteeController
char: yesOrNo

Test Data Type	Type of Validation	Input Data	Expected Output	Test Passed/Failed
Normal	Content and Type Check: ‘Y’, ‘N’	‘Y’	Accept the input	Passed
Extreme		‘F’, ‘L;	Reject and ask to input again	Passed
Abnormal		7	Reject and ask to input again	Passed

Class: NewDate
day: int

Test Data Type	Type of Validation	Input Data	Expected Output	Test Passed/Failed
Normal	Range Check: 1-31	4	Accept the input	Passed
Extreme		35; -13;	Reject and ask to input again	Passed
Abnormal		“7”	Reject and ask to input again	passed

Class: TutorController
boolean searchByName(name):

Test Data Type	Type of Validation	Input Data	Expected Output	Test Passed/Failed
Normal	Presence check	!null	Accept the input	passed
Extreme		null	Print("No tutor found with that name")	Passed
Abnormal		"2025"	Print("No tutor found with that name")	Passed

Class: NewDate
NewDate dateToString()

Test Data Type	Type of Validation	Input Data	Expected Output	Test Passed/Failed
Normal	Format and Content Check: DD/MM/YYYY	12/03/2023	Accept the input	passed
Extreme		35/15/98	Reject and ask to input again	Passed
Abnormal		"string"	Reject and ask to input again	Passed

4. **Pseudocode:** Highlights a few of the algorithms that will be used in the development

Searching Algorithms

The system will store an ArrayList of subjects tutored and learned in Peer Tutoring. When the client needs to look at the details of a specific subject, they will need to search for the subject. The searching algorithm chosen is binary search for greater efficiency in searching. This is because the efficiency of this algorithm is $O(\log_2 n)$ which is more efficient than other algorithms, such as Linear search, which has an efficiency of O(n).

Assuming that the length of the ArrayList is already there and sorted by subject name and subject level, the algorithm is

```
function binarySearch(subject, subjects)
        set low to 0
        set high to subjects.size()
        while (low <= high)
                set mid to (low + high)/2
                if subjects.get(mid) equals (subject) then
                        return true
                        exit loop
                else if subjects.get(mid) < subject then
```

```
                set low to index + 1
                end if
            else
                set high to index - 1
            end if
        end loop
        return false
end function
```

Where:
subjects - ArrayList of subjects being searched
mid - index of the centre element of sub-ArrayList
subjects.size(): length of subjects ArrayList
low: lowest index of sub-ArrayList
high: highest index of sub-ArrayList

(adapted from Drien Vargas)

Sorting Algorithms
To sort the subjects by subject name and level, one of the best sorting algorithms is the insertion sort, which is versatile and adaptable to the data given.

Since the data is stored as an ArrayList of subjects, the algorithm is:

```
function insertionSort(subjects)
        set x to 0
        set length to subjects.length
        loop for i <- 1 to length
            set temp to subjects[i]
            set j to i-1
            loop while j >=0 and subjects[j] > temp
                set x to x + 1
                set subjects[j+1] to subjects[j]
                set j to j -1
            end loop
            set subjects[j+1] to temp
        end loop
end function
```

where:

subjects: ArrayList of subjects being sorted
length: length of subjects ArrayList
x: value of the current position to place the smallest element
i: index for outer loop
j: index for inner loop
temp: temporary variable to compare to

This algorithm was adapted from (Drien Vargas)

Extensions and Modifications (from Criterion D)

- Changing Person ID generator
    - Location: Person.java
    - Development: Instead of auto-generating the string Person IDs if wanted, an integer ID could be generated instead. This could be done by changing the id to integer and changing the **setID** method to add to the **idNumbers** list without using the String.format. Furthermore, instead of auto-generating the IDs, it could become an attribute set by the user for each person. This would make the ID more involved in the system, allowing methods to search by the unique ID to happen.
- Developing a Pairing Algorithm
    - Location: RankingController.java
    - Development: Instead of leaving the pairing of the tutor and tutees to the client after they look at the tutor rankings, a different class/algorithm called pairing could be created. This would create an elaborate pairing of tutors and tutees based on more factors than just the tutor rankings such as the subjects they tutor/learn, their grade level, the sessions they are available for, and the amount of help a tutee actually needs.
- Changing the Subject class to include IB or non-IB students
    - Location: Subject.java, Tutee.java, Tutor.java
    - Development: This system only caters towards IB students in Peer Tutoring. However, peer Tutoring also includes non-IB students in grades 9 and 10. The system could be extended to have a boolean indicating whether the subject is IB or non-IB, and the same for Tutor and Tutee classes.