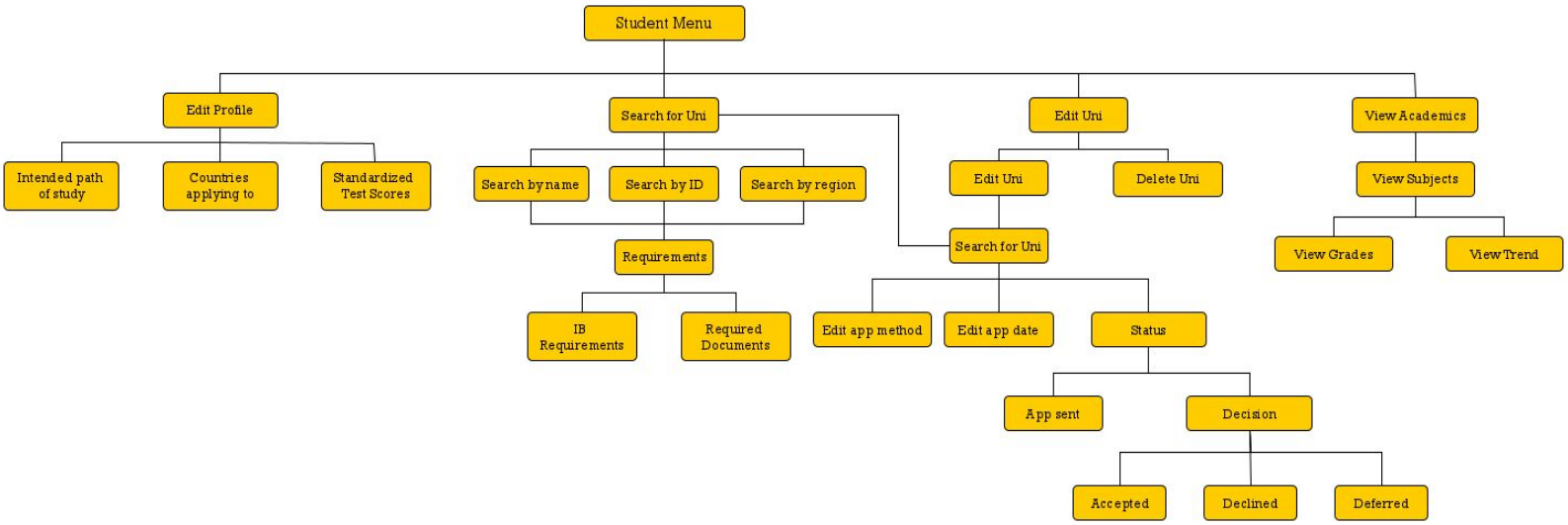
Criterion B - Design

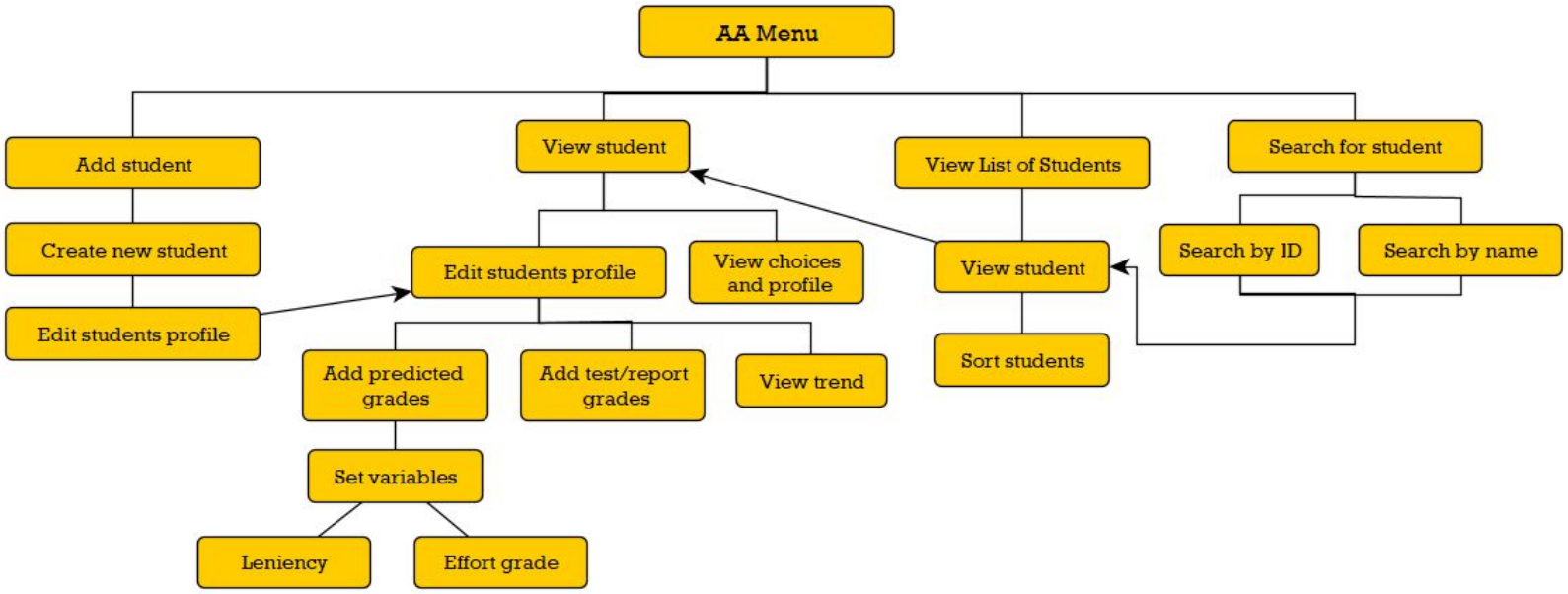
Decomposition Diagrams

There will be 2 separate UIs (user interfaces); one for AAs (very slight difference in administrative powers) and one for students. Thus, there will be 2 decomposition diagrams. I will be using the software yEd to illustrate my diagrams. “University” will be abbreviated to “Uni”, and “Application” to “App”, for the sake of conciseness and cohesion.

The student decomposition is shown below. All the images are in the folder “Design”; the full size, high resolution image is in the folder titled “studentdecomposition.png”.



The AA decomposition is shown below; the full size, high resolution image is titled “AAdecomposition.png”.



Test Plan

The test plan is to depict the validation that will be done in the system to prohibit incorrect values and errors.

Class: Person

This class applies to everybody: students and AAs. Each faculty has a unique 5-digit ID number and a first & last name.

ID number: int

Validation Type	Data Type	Input Data	Intended Output	Test Pass/Fail
Range Check: 00001≤x≤99999 Presence Check Type Check: integer Length Check: 5 integers long	Normal	42069	System accepts the data	Pass
	Extreme	00001, 99999	System accepts the data	Pass
	Abnormal	Hello, 420xx, 420420, f%$k	System shows error message and prompts user to input valid data	Pass

fristName, lastName: String

Validation Type	Data Type	Input Data	Intended Output	Test Pass/Fail
Presence Check Type Check: String Format Check: Alphabets only	Normal	Lionel Messi	System accepts the data	Pass
	Extreme			
	Abnormal	LM10, 420, h3ll0, “” (empty)	System shows error message and prompts user to input valid data	Pass

Class: Class

classsubjectName: String - For this, there will be a dropdown with all subjects taught to be chosen from.

Validation Type	Data Type	Input Data	Intended Output	Test Pass/Fail
Presence Check	Normal	!= null	System accepts data	Pass
	Extreme	null	System shows error message and prompts user to input valid data	Pass
	Abnormal	null	System shows error message and prompts user to input valid data	Pass

teacherName: String

Validation Type	Data Type	Input Data	Intended Output	Test Pass/Fail
Presence Check Type Check: String	Normal	!= null	System accepts data	Pass
	Extreme	null	System shows error message and prompts user to input valid data	Pass
	Abnormal	null	System shows error message and prompts user to input valid data	Pass

Class: Subject

subjectName[]: String - array of 8 elements -- 3 HL, 3 SL, TOK, EE

Validation Type	Data Type	Input Data	Intended Output	Test Pass/Fail
Presence Check	Normal	!= null	System accepts data from dropdown	Pass
	Extreme	null	System shows error message and prompts user to input valid data	Pass
	Abnormal	null	System shows error message and prompts user to input valid data	Pass

subjectLevel: String - Implement a dropdown for HL / SL. After dropdown chosen; user will be prompted to add class number. The subject Level will have dropdown right before the class number. The subjectLevel test plan is identical to subjectName since it is also a dropdown

Validation Type	Data Type	Input Data	Intended Output	Test Pass/Fail
Presence Check Type Check: int Format Check: ##	Normal	“02”	System accepts data	Pass
	Extreme	“01”	System accepts data	Pass
	Abnormal	“P6f5”, ”1”, “00”, “hello”	System shows error message and prompts user to input valid data	Pass

Class: Profile

standardTests: int (double for IELTS) - A list/option of various mainstream tests will be available - each having different ranges of integer scores

Standardized Test	Lowest Score	Highest Score	Intervals
SAT	400	1600	10
ACT	1	36	1

SAT Subject	200	800	10
IELTS	1	9	0.5
TOEFL	0	120	1

Will definitely include SAT, may not include others. Each will have Presence Check, respective Format and Type Check

country: String

Class: University

universityName: String

Implement a searchable dropdown (using CSS & Javascript), with a (test) CSV file of universities

applicationDate: Date

Validation Type	Data Type	Input Data	Intended Output	Test Pass/Fail
Presence Check Format Check: DD-MM-YY Type Chec: DD, MM, YY - int Range Check: 01<=DD<=31 01<=MM<=12 20<=DD<=21	Normal	02-04-2020	System accepts data	Pass
	Extreme	01-01-2020	System accepts data	Pass
	Abnormal	0h-3l-l0 35-32-30	System shows error message and prompts user to input valid data	Pass

stats: String

Will have statistics such as general: location, size, faculty ratio, etc, academic: IB requirement/score, standardized scores.

Class: Academics

grades[]: double

Validation Type	Data Type	Input Data	Intended Output	Test Pass/Fail
Presence Check Type Check: double Format Check: #.## Range Check: 1.00<=X<=7.90	Normal	“6.50”	System accepts data	Pass
	Extreme	“1.00”, “7.90”	System accepts data	Pass
	Abnormal	‘7’, ‘H’, “hello”	System shows error message and prompts user to input valid data	Pass

effortGrade: char

Validation Type	Data Type	Input Data	Intended Output	Test Pass/Fail
Presence Check Type Check: char Format Check: ‘C’	Normal	‘C’	System accepts data	Pass
	Extreme	‘A’, ‘F’	System accepts data	Pass
	Abnormal	‘7’, ‘H’, “hello”	System shows error message and prompts user to input valid data	Pass

predictedGrades: int

Validation Type	Data Type	Input Data	Intended Output	Test Pass/Fail
Presence Check Type Check: int Format Check: # Range Check: 1<=X<=7	Normal	“6”	System accepts data	Pass
	Extreme	“1”, “7”	System accepts data	Pass
	Abnormal	‘8’, ‘H’, “hello”	System shows error message and prompts user to input valid data	Pass

Use Case Diagrams

The use case diagram highlights the potential scope of the proposed solutions. It explores how the student and AA interact with the system and with each other and provides a basic understanding of the system. A full resolution image may be found titled “Usecasediagram.jpg”

Data Dictionary

The data dictionary is a centralized set of information about data, contents, format, relationships, used to control access to and manipulate a database. The table below shows a rough draft of the data dictionary, conditions, description and attributes.

Attribute	Data Type	Modifier	Description
Person			
firstName	String	private	Every person has a first name consisting on only alphabets -- String
lastName	String	private	Every person has a last name consisting on only alphabets -- String
ID number	int	private	Each person has a unique 5-digit ID number, that matches the school ID number for ease and completeness of system
Class			
classsubjectName	String	private	Dropdown of available IB subjects at (client's) school
teacherName	String	private	Each class will have a teacher, that may be referred to for recommendation letters, messages, etc
Subject			
subjectName[]	String	private	Array of 8 elements -- 3 HL, 3 SL, TOK, EE
subjectLevel	String & int	private	subjectLevel will contain HL or SL and a class number
Profile			
standardTests	int	private	A list/option of various mainstream tests will be available - each having different ranges of integer scores - this will be visible for student profile
country	String	private	Each Student will be applying to a University which contains a location/country
University			
universityName	String	private	Implement a searchable dropdown (using CSS & Javascript), with a (test) CSV file of universities -- help AA with application

applicationDate	Date	private	Illustrate deadline dates for university applications -- each university may have a different date (CSV)
stats	String	private	Will have statistics such as general: location, size, faculty ratio, etc, academic: IB requirement/score, standardized scores (CSV)
Academics			
grades[]	double	private	Includes internal test/report grades -- 4 scores per subject per semester
effortGrade	char	private	Includes 2 effort grades per semester
predictedGrades	int	private	AA inputted predicted grades (discussed with subject teachers)

IPO Tables

The tables below shows the various intended functions in terms of Input --> Processing --> Output from a student's and AA's point of view.

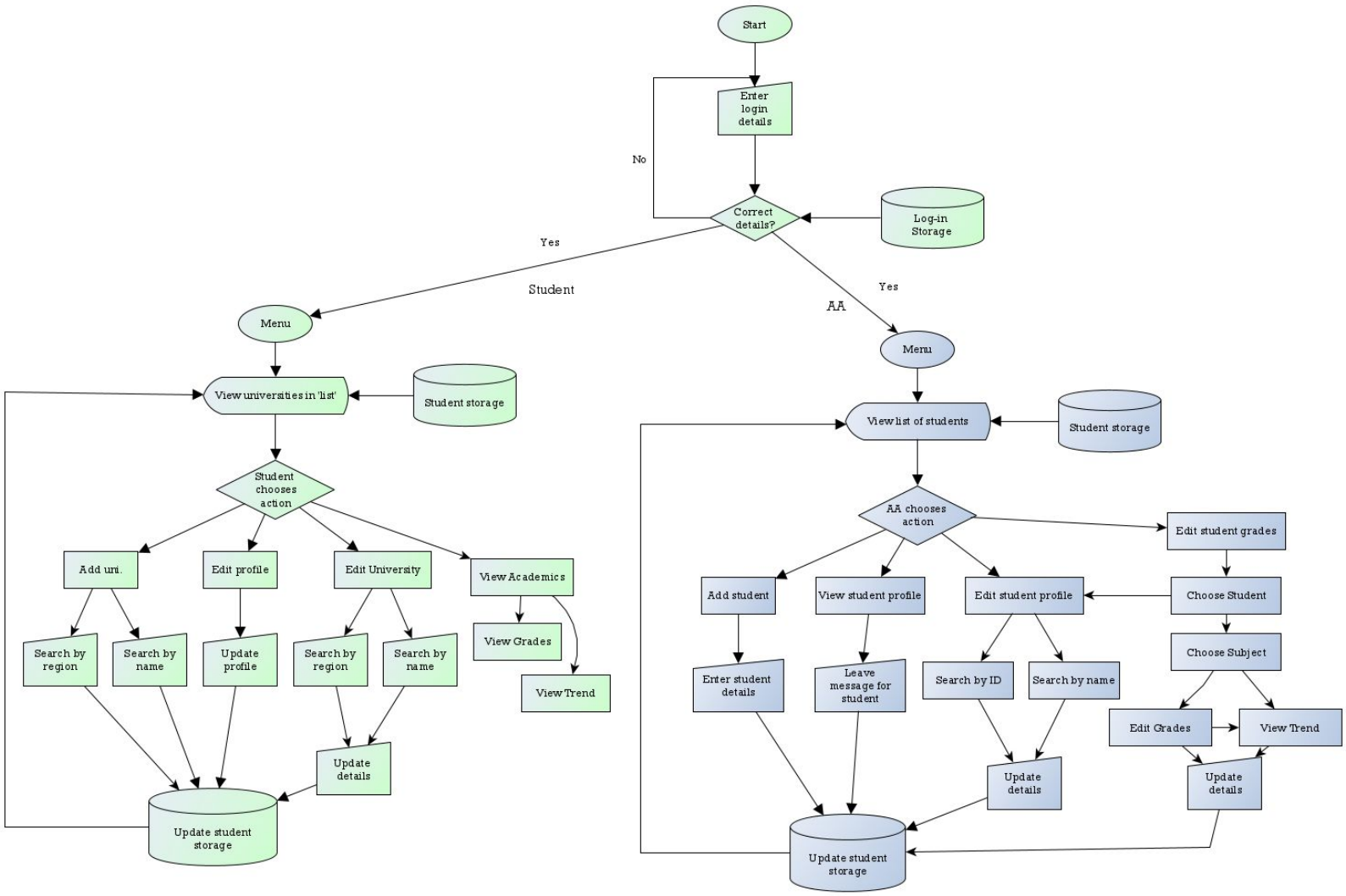
Student IPO Table		
Input	Processing	Output
Edit Profile	Loads profile and its attributes from student storage	Prompts and takes edits and updates student storage
Add University	Creates new university object	Takes user input and populates object
Search University	Searches (potentially CSV file) by name OR Searches by region	Displays university details such as location, scores/requirements, statistics
View Academics	Loads student storage, AA inputted subject & grades	Displays chosen subjects and grades along with a trend

Edit University	Adds, Deletes University (object), loads application date & method options	Takes user input and populates/updates object(s)

AA IPO Table		
Input	Processing	Output
Search Student	Search by Student ID OR Search by Student Name	Displays Student Profile, grades trend and option to Add Grades & Predicted Grades
View List of Students	Loads AA Storage/list of students	Displays all students designated to the AA
Add Student	Creates new Student object Creates username & password (random)	Prompts input for Student name Displays username & password on screen after completion
Edit Student grades	Loads chosen student from storage and stores variables After edits are finalised, variables are updated and student object is stored	Prompts AAs to make edits -- predicted grades OR test/grades After edits, a message is displayed to confirm actions

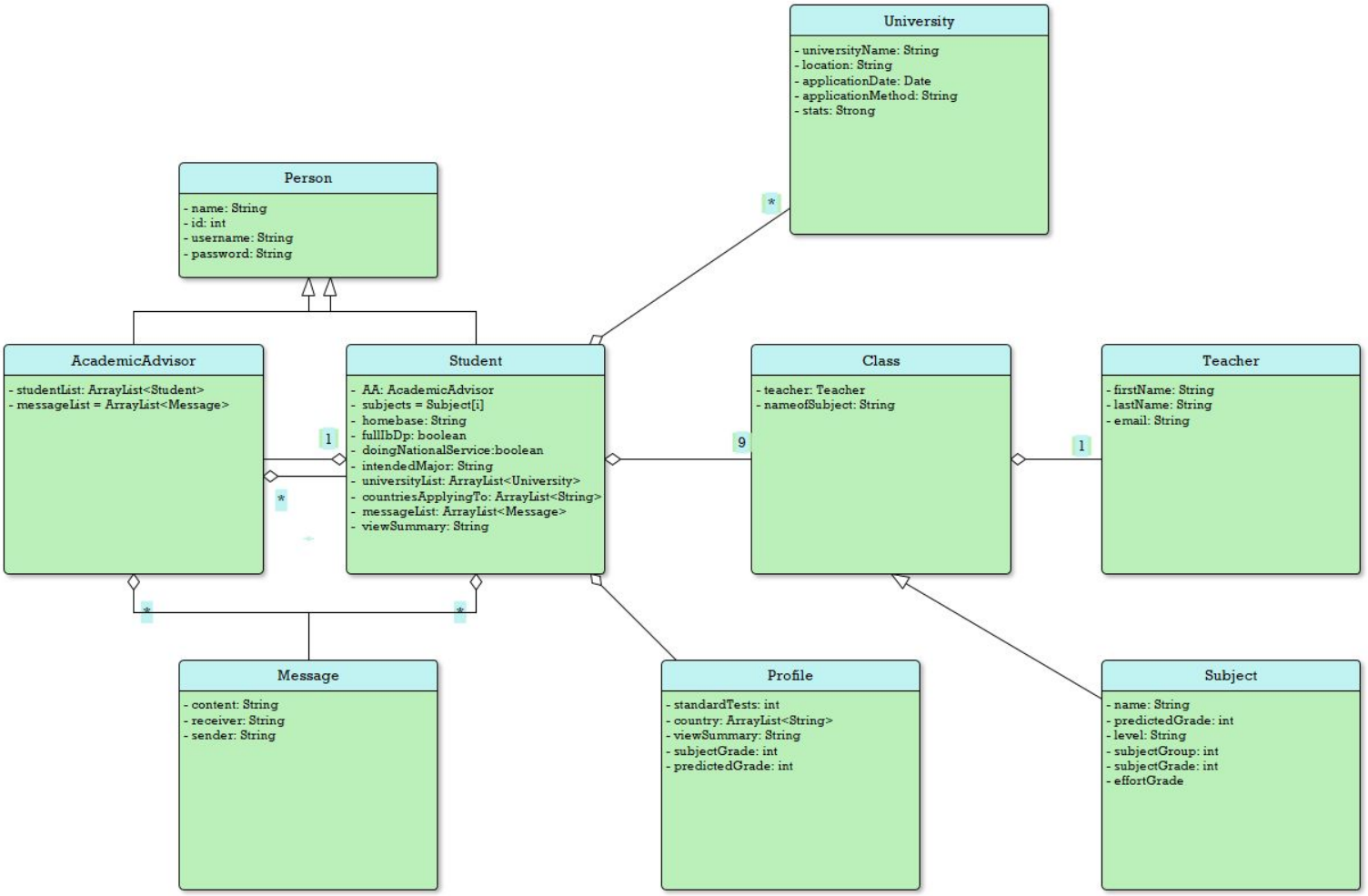
Flowchart Diagrams

Below is a basic flowchart of how the system should work. After the correct login details are entered, the flowchart splits in 2 paths -- AA & Student, in which only 1 will be chosen.

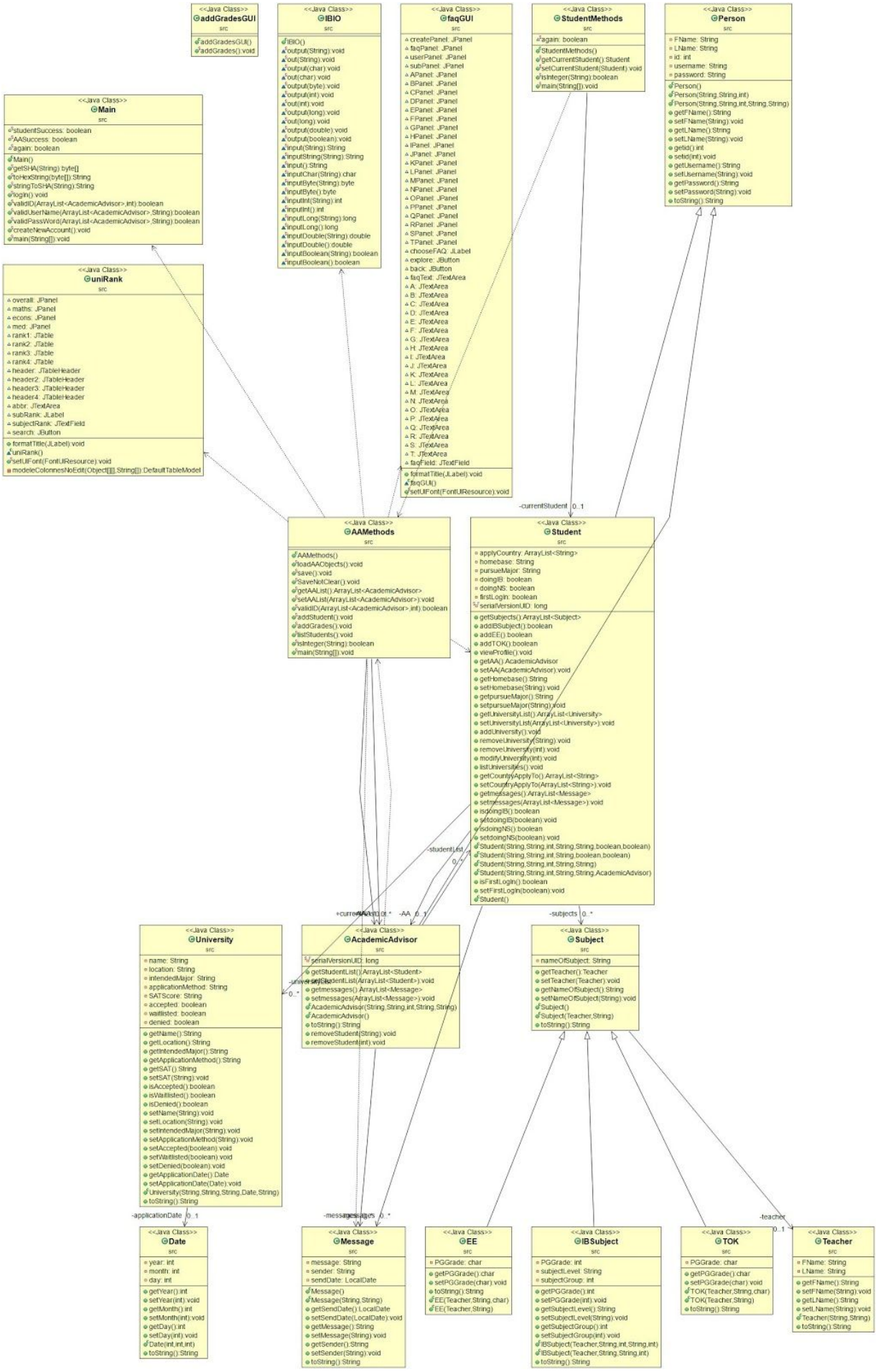


UML Class Diagrams

The below image shows a potential UML diagram of the product, though very simplified, shows the main features of the code.



The image below, accessible as postUMLDiagram.jpg, displays the UML Diagram formulated by Eclipse (ObjectAidUML) of the final product. It is generated from the software that takes information from the product code, thus it should have almost 0 uncertainties and errors.


addGradesGUI
src
addGradesGUI()
addGrades():void
IBIO
src
IBIO()
output(String):void
out(String):void
output(char):void
out(char):void
output(byte):void
output(int):void
out(int):void
output(long):void
out(long):void
output(double):void
output(boolean):void
input(String):String
inputString(String):String
input():String
inputChar(String):char
inputByte(String):byte
inputByte():byte
inputInt(String):int
inputInt():int
inputLong(String):long
inputLong():long
inputDouble(String):double
inputDouble():double
inputBoolean(String):boolean
inputBoolean():boolean
faqGUI
src
createPanel: JPanel
faqPanel: JPanel
userPanel: JPanel
subPanel: JPanel
APanel: JPanel
BPanel: JPanel
CPanel: JPanel
DPanel: JPanel
EPanel: JPanel
FPanel: JPanel
GPanel: JPanel
HPanel: JPanel
IPanel: JPanel
JPanel: JPanel
KPanel: JPanel
LPanel: JPanel
MPanel: JPanel
NPanel: JPanel
OPanel: JPanel
PPanel: JPanel
QPanel: JPanel
RPanel: JPanel
SPanel: JPanel
TPanel: JPanel
chooseFAQ: JLabel
explore: JButton
back: JButton
faqText: JTextArea
A: JTextArea
B: JTextArea
C: JTextArea
D: JTextArea
E: JTextArea
F: JTextArea
G: JTextArea
H: JTextArea
I: JTextArea
J: JTextArea
K: JTextArea
L: JTextArea
M: JTextArea
N: JTextArea
O: JTextArea
P: JTextArea
Q: JTextArea
R: JTextArea
S: JTextArea
T: JTextArea
faqField: JTextField
formatTitle(JLabel):void
faqGUI()
setUIFont(FontUIResource):void
StudentMethods
src
again: boolean
StudentMethods()
getCurrentStudent():Student
setCurrentStudent(Student):void
isInteger(String):boolean
main(String[]):void
Person
src
FName: String
LName: String
id: int
username: String
password: String
Person()
Person(String,String,int)
Person(String,String,int,String,String)
getFName():String
setFName(String):void
getLName():String
setLName(String):void
getId():int
setId(int):void
getUsername():String
setUsername(String):void
getPassword():String
setPassword(String):void
toString():String
Main
src
studentSuccess: boolean
AASuccess: boolean
again: boolean
Main()
getSHA(String):byte[]
toHexString(byte[]):String
stringToSHA(String):String
login():void
validID(ArrayList<AcademicAdvisor>,int):boolean
validUserName(ArrayList<AcademicAdvisor>,String):boolean
validPassWord(ArrayList<AcademicAdvisor>,String):boolean
createNewAccount():void
main(String[]):void
uniRank
src
overall: JPanel
maths: JPanel
econs: JPanel
med: JPanel
rank1: JTable
rank2: JTable
rank3: JTable
rank4: JTable
header: JTableHeader
header2: JTableHeader
header3: JTableHeader
header4: JTableHeader
abbr: JTextArea
subRank: JLabel
subjectRank: JTextField
search: JButton
formatTitle(JLabel):void
uniRank()
setUIFont(FontUIResource):void
modeleColonnesNoEdit(Object[][],String[]):DefaultTableModel
AAMethods
src
AAMethods()
loadAAObjects():void
save():void
SaveNotClear():void
getAAList():ArrayList<AcademicAdvisor>
setAAList(ArrayList<AcademicAdvisor>):void
validID(ArrayList<AcademicAdvisor>,int):boolean
addStudent():void
addGrades():void
listStudents():void
isInteger(String):boolean
main(String[]):void
Student
src
applyCountry: ArrayList<String>
homebase: String
pursueMajor: String
doingIB: boolean
doingNS: boolean
firstLogin: boolean
serialVersionUID: long
getSubjects():ArrayList<Subject>
addIBSubject():boolean
addEE():boolean
addTOK():boolean
viewProfile():void
getAA():AcademicAdvisor
setAA(AcademicAdvisor):void
getHomebase():String
setHomebase(String):void
getpursueMajor():String
setpursueMajor(String):void
getUniversityList():ArrayList<University>
setUniversityList(ArrayList<University>):void
addUniversity():void
removeUniversity(String):void
removeUniversity(int):void
modifyUniversity(int):void
listUniversities():void
getCountryApplyTo():ArrayList<String>
setCountryApplyTo(ArrayList<String>):void
getmessages():ArrayList<Message>
setmessages(ArrayList<Message>):void
isdoingIB():boolean
setdoingIB(boolean):void
isdoingNS():boolean
setdoingNS(boolean):void
Student(String,String,int,String,String,boolean,boolean)
Student(String,String,int,String,boolean,boolean)
Student(String,String,int,String,String)
Student(String,String,int,String,String,AcademicAdvisor)
isFirstLogin():boolean
setFirstLogin(boolean):void
Student()
University
src
name: String
location: String
intendedMajor: String
applicationMethod: String
SATScore: String
accepted: boolean
waitlisted: boolean
denied: boolean
getName():String
getLocation():String
getIntendedMajor():String
getApplicationMethod():String
getSAT():String
setSAT(String):void
isAccepted():boolean
isWaitlisted():boolean
isDenied():boolean
setName(String):void
setLocation(String):void
setIntendedMajor(String):void
setApplicationMethod(String):void
setAccepted(boolean):void
setWaitlisted(boolean):void
setDenied(boolean):void
getApplicationDate():Date
setApplicationDate(Date):void
University(String,String,String,Date,String)
toString():String
AcademicAdvisor
src
serialVersionUID: long
getStudentList():ArrayList<Student>
setStudentList(ArrayList<Student>):void
getmessages():ArrayList<Message>
setmessages(ArrayList<Message>):void
AcademicAdvisor(String,String,int,String,String)
AcademicAdvisor()
toString():String
removeStudent(String):void
removeStudent(int):void
Subject
src
nameOfSubject: String
getTeacher():Teacher
setTeacher(Teacher):void
getNameOfSubject():String
setNameOfSubject(String):void
Subject()
Subject(Teacher,String)
toString():String
Date
src
year: int
month: int
day: int
getYear():int
setYear(int):void
getMonth():int
setMonth(int):void
getDay():int
setDay(int):void
Date(int,int,int)
toString():String
Message
src
message: String
sender: String
sendDate: LocalDate
Message()
Message(String,String)
getSendDate():LocalDate
setSendDate(LocalDate):void
getMessage():String
setMessage(String):void
getSender():String
setSender(String):void
toString():String
EE
src
PGGrade: char
getPGGrade():char
setPGGrade(char):void
toString():String
EE(Teacher,String,char)
EE(Teacher,String)
IBSubject
src
PGGrade: int
subjectLevel: String
subjectGroup: int
getPGGrade():int
setPGGrade(int):void
getSubjectLevel():String
setSubjectLevel(String):void
getSubjectGroup():int
setSubjectGroup(int):void
IBSubject(Teacher,String,int,String,int)
IBSubject(Teacher,String,String,int)
toString():String
TOK
src
PGGrade: char
getPGGrade():char
setPGGrade(char):void
TOK(Teacher,String,char)
TOK(Teacher,String)
toString():String
Teacher
src
FName: String
LName: String
getFName():String
setFName(String):void
getLName():String
setLName(String):void
Teacher(String,String)
toString():String
-currentStudent 0..1
-studentList 0..*
+currentAA 0..1
-AA 0..1
-subjects 0..*
-university 0..*
-applicationDate 0..1
-messages 0..*
-teacher 0..1

Pseudocode

Below is a rough pseudocode on possibly the most important method in the program, that is the addStudent method that Academic Advisors can use to add students into the program. This method will allow efficient organisation and implementation that grants AAs power to make account. This code will contain validation such that the program doesn't crash and invalid & insensible data is not inputted. After the validation, this data is stored in an ArrayList, that can be used to sign in the program, later on.

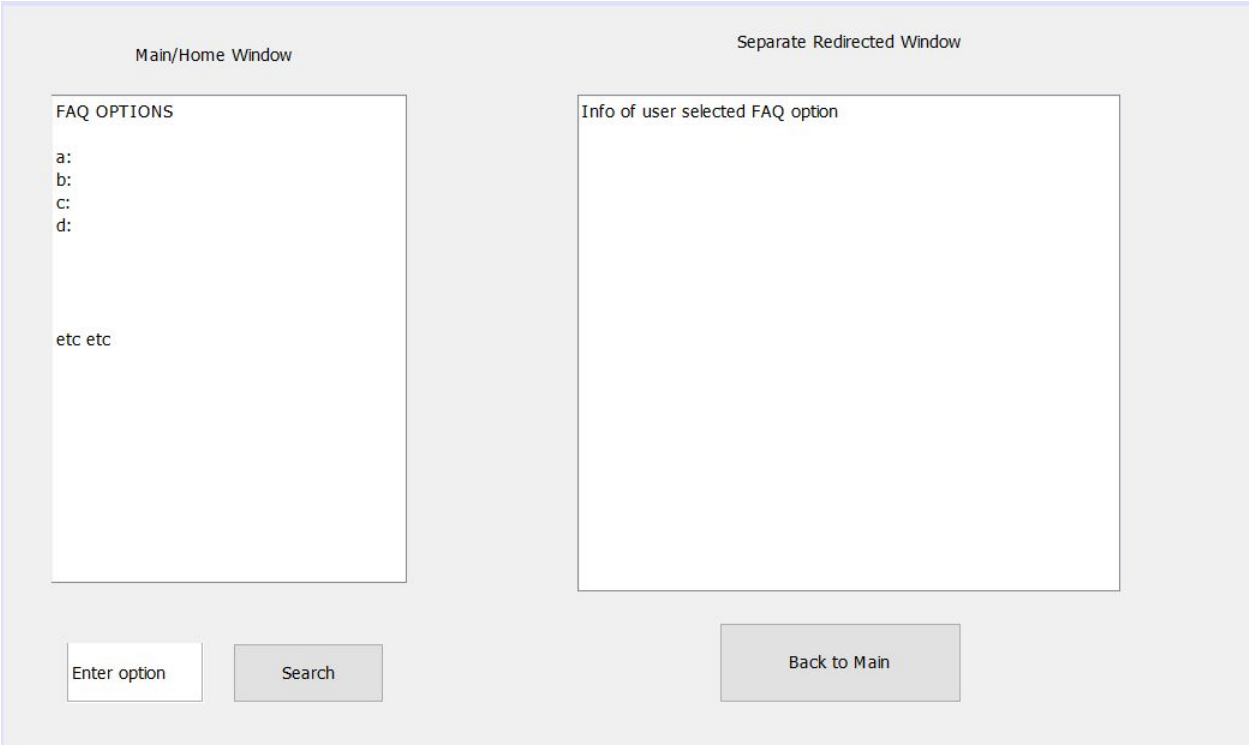
```
addStudent()
    Creating Student under AA
    input firstName;
    input lastName;
    loop while(not validID)
          enteredID = input("Input ID"); \\asks for input
    end while
    loop while(username found OR duplicate OR blank)
          enteruser = input("Input username");
          loop for i = 0; i < AAList.size; i++
               if AAList.get(i).equals(enteruser) OR StudentList.get(i).equals(enteruser)
                    break;
               end if
    end while
    boolean match == false;
    loop while match == false; \\password
          firstpass = input("Input password");
          secondpass = input("Confirm password");
          if firstpass not secondpass OR firstpass equals "" OR secondpass equals ""
               output Error Message
          end if
    end while
    confirm = input("Confirm?")
    if confirm == y
          add in StudentList
    else
          output Details not Saved
    end if

validID()
     loop for i < AAList.size; i++
        if(AList.get(i).getid() == ID){
        System.out.println("Error, Invalid ID");
        return false;
      end loop
      loop for i < StudentList.size; i++
        if(AList.get(i).getid() == ID){
        System.out.println("Error, Invalid ID");
        return false;
      end loop
```

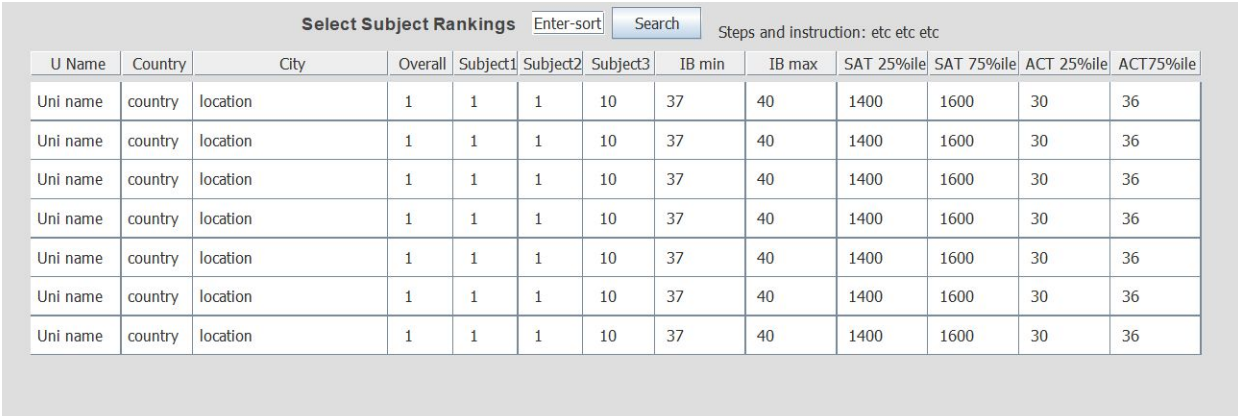
GUI Draft

Instead of implement a GUI for the whole program, I've decided to use GUI -- Java Swing, to display information and input information. Making a GUI for the whole menu would not be feasible because

it would take too much time, it would become too complex, it could take too much space, etc. To reduce complications, I've opted to have GUI windows open when selected a specific menu item, from the menu driven interface. Once selected, the option will call a separate GUI class, in which I'd be coding, to function as an easier way to view information. It would be easier to view information such as interactive FAQ, large information, inputting numerous grades, etc, into a GUI rather than a Command Line Interface.



A simple GUI like this will allow for easy navigation, especially for my clients and his students. Complexity comes due to the redirection from the main window to the other window. There will only be one window on display at 1 time.



Select Subject Rankings Enter-sort Search Steps and instruction: etc etc etc

U Name	Country	City	Overall	Subject1	Subject2	Subject3	IB min	IB max	SAT 25%ile	SAT 75%ile	ACT 25%ile	ACT75%ile
Uni name	country	location	1	1	1	10	37	40	1400	1600	30	36
Uni name	country	location	1	1	1	10	37	40	1400	1600	30	36
Uni name	country	location	1	1	1	10	37	40	1400	1600	30	36
Uni name	country	location	1	1	1	10	37	40	1400	1600	30	36
Uni name	country	location	1	1	1	10	37	40	1400	1600	30	36
Uni name	country	location	1	1	1	10	37	40	1400	1600	30	36
Uni name	country	location	1	1	1	10	37	40	1400	1600	30	36

The picture above may be the layout for the sorting subject rankings of universities. As showed when inputted a subject in the text area, the table should sort into ascending order of user-inputted subject rankings.

<u>Possible Modifications</u>:

- UI can be altered in classes Main.java, Student.java, StudentMethods.java, AcademicAdvisor.java and AAMethods.java.
- If needed to access or clean wipe previous data, AAObjects.txt can be altered. Though it is using encryption, thus would be tough to decode.

<u>Note:</u>

System cannot run if there is no AA beforehand. To overcome this ‘bug’, within the code, AAMethods.loadAAObjects(); in the Main.java line 227, needs to be commented for the first time of usage. After that it should be uncommented.