Student Management System

CW1

Adam Baguma 24903311

Table of Contents

[Executive Summary 4](#_Toc121519866)

[Introduction 4](#_Toc121519867)

[Requirements 7](#_Toc121519868)

[Design 11](#_Toc121519869)

[Overview 11](#_Toc121519870)

[Sequence diagrams 11](#_Toc121519871)

[Search for Staff members 11](#_Toc121519872)

[Add/delete students and modules 12](#_Toc121519873)

[Search for student information and module information 12](#_Toc121519874)

[User Interface Design layout 14](#_Toc121519875)

[Object Orientated Design and Concepts 18](#_Toc121519876)

[Polymorphism 18](#_Toc121519877)

[Encapsulation 18](#_Toc121519878)

[Inheritance 18](#_Toc121519879)

[Abstraction 18](#_Toc121519880)

[Instantiation 18](#_Toc121519881)

[Classes 19](#_Toc121519882)

[Class: Person 19](#_Toc121519883)

[Class: Student 19](#_Toc121519884)

[Class: Academics 20](#_Toc121519885)

[Class: Module 20](#_Toc121519886)

[Class: Application 20](#_Toc121519887)

[Class: GUI 21](#_Toc121519888)

[Summary 22](#_Toc121519889)

[References 23](#_Toc121519890)

[Appendices 23](#_Toc121519891)

[Activity diagram 23](#_Toc121519892)

[Request student details 23](#_Toc121519893)

[Use case diagram 24](#_Toc121519894)

[Class diagram 25](#_Toc121519895)

[Object diagram 26](#_Toc121519896)

[ER diagrams 27](#_Toc121519897)

# Table of Figures

[Figure 1 12](https://edgehill-my.sharepoint.com/personal/24903311_edgehill_ac_uk/Documents/CW1%20OOP.docx#_Toc121519844)

[Figure 2 13](https://edgehill-my.sharepoint.com/personal/24903311_edgehill_ac_uk/Documents/CW1%20OOP.docx#_Toc121519845)

[Figure 3 14](https://edgehill-my.sharepoint.com/personal/24903311_edgehill_ac_uk/Documents/CW1%20OOP.docx#_Toc121519846)

[Figure 4 15](#_Toc121519847)

[Figure 5 16](#_Toc121519848)

[Figure 6 17](#_Toc121519849)

[Figure 7 17](#_Toc121519850)

[Figure 8 18](#_Toc121519851)

[Figure 9 18](#_Toc121519852)

[Figure 10 20](https://edgehill-my.sharepoint.com/personal/24903311_edgehill_ac_uk/Documents/CW1%20OOP.docx#_Toc121519853)

[Figure 11 20](#_Toc121519854)

[Figure 12 21](#_Toc121519855)

[Figure 13 21](#_Toc121519856)

[Figure 14 21](https://edgehill-my.sharepoint.com/personal/24903311_edgehill_ac_uk/Documents/CW1%20OOP.docx#_Toc121519857)

[Figure 15 25](#_Toc121519858)

[Figure 16 25](#_Toc121519859)

[Figure 17 26](#_Toc121519860)

[Figure 18 27](#_Toc121519861)

[Figure 19 27](#_Toc121519862)

[Figure 20 28](#_Toc121519863)

[Figure 21 28](#_Toc121519864)

[Figure 22 28](#_Toc121519865)

Executive Summary

The aim of this project was to create a student management system that enables staff members to add modules, students and other users as well as remove them. The programme had to utilise a Java Swing-based GUI which was also achieved to create a user interface for the end the user. The program had to enable students to have the ability to search for staff members and modules to see information such as the staff members details and the module leaders / moderator for the specific module they searched.

The completed programme used an object orientated approach to loads and instantiate the information from the various databases to their respective objects to in order to create an environment where the user can interact / alter the information held in the database.

The objects also respond differently when certain methods are called such as the objects first name and last name.

Introduction

This project aims to build a student database system using java that enables the user to create, add and delete information stored within the csv file all within a Java Swing-based graphical user interface for ease of use. The project will last 2 months and will be split into 8 small segments stating what needs to be done in order to meet the deadline in December.by the end of the deadline, the project should deliver a functional java application that implements a range of techniques and methods in order to achieve and meet desired requirements for the user.

|  |  |
| --- | --- |
| Week Beginning | To Do |
| 03/10/2022 | Read through the course work one document and pick the project to work on for the next couple of weeks |
| 10/10/2022 | Create the project introduction and table of requirements. begin explaining potential requirements that the project will need. refer to the project requirements  begin creating the various models and diagrams that the programme will utilise. |
| 17/10/2022 | Break down the requirements into low- level and high-level requirements and place these into a table on the Course work 1 report draft. Begin working on the design section of your report.  begin the process to design the layout of the report, ensuring that object orientation is implemented and explained within the design and code. create a use case model to break down the first steps to tackling the larger problem at hand.  include a range of use case models such as activity diagrams and sequence diagrams to better illustrate and break down the various complexities that the programme will have the design should link to the lower level requirements. |
| 24/010/2022 | further develop the use case diagrams and associations and include uml diagrams for the software.  Begin to incorporate Object Orientated principles such as inheritance, polymorphism, instantiation, encapsulation and abstraction.  Begin to explain the reason behind the design of the programme and highlight the object orientated principles included. |
| 31/10/2022 | Begin to finalise the use case diagrams, uml diagrams and activity diagrams. once complete include them into the report and break down the different links incorporating object orientated principles within the explanation to better understand the link between the diagrams and the project.  Ensure the requirements are being met in the design before beginning to write the software that the programme will run.  ensure code is commented with sensible variable names and methods. |
| 07/11/2022 | work on the code and introduce object orientated programming ( OOP) concepts with in the software. begin with the person class as it will be required to create both the student and staff objects. ensure the concepts are implemented correctly and go over the current code to check variables are used correctly, classes form the diagrams are included and a range of basic requirements are included. |
| 14/11/2022 | Continue developing the code and begin planning the layout of the graphical user interface and explain how it will be used by the user to generate the desired outputs from the software. |
| 21/11/2022 | Continue developing the code ensuring that the basic, intermediate and advanced requirements are being met.  Refer to the mark scheme to aid this process |
| 28/11/2022 | Aim to complete the code during this week with a list of requirements that have been implemented that should be included in the report along with the list of requirements that have not been implemented and why they couldn’t be included in the software.  Ensure intermediate requirements are also included in the software. |
| 05/12/2022 | Complete the report by including the executive summary and the summary which should hold a deeper dive into the executive summary.  Ensure the document is structed correctly with references and spell checking to ensure no mistakes have been made. |

Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| High Level | Low level | Description | Was Requirement met? |
| The application must have a Java Swing-based GUI. | Creating user GUI. | An interface to bridge the gap between the user and the software. | Requirement was met. |
| Uml behavioural and structural Diagrams. | Behavioural:  Create sequence, activity and use case diagrams. | This is to provide clarity for the internal processes and different interactions within the system. | Requirement was met. |
| Structural:  Create class  And object Diagrams. | This is to show how one object relates to another. | Requirement was met. |
| Students should be able to view all modules and the course they are taking. | Create academics and Student csv files and populate the files with test data. | The academics and Student csv files will simulate a real-world scenario to show how the software could be implemented and to find bugs that may occur. | Requirement was met. |
| Create a student class and object. | The student class will be used to create objects of students. The class will inherit methods and attributes such as first name from the person class. | Requirement was met. |
| Create an academics class and object. | The academics class will retrieve data from the academic’s csv file and populate the object with the staff data so when the student searches for a module, the module leaders and moderators will be searched, and their details will be displayed on the module the student searched. | Requirement was met. |
| Academics should be able to add and remove modules. | Create Module class and csv file and populate the file with test data. | The module class will receive information from the csv file and create objects of modules that the user can add and remove students from before saving this information to the csv database. | Requirement was met. |
| The application must allow users to create and delete students. | Creating a Sub routing to add Students to the database. | An array containing all the students is used to store the student objects. When a user wants to add a student, the new student is placed in an empty section of the array, and this is then updated in the database. | Requirement was met. |
| Remove students from the database. | If a user needs to be removed from the array, the position of the object is located before clearing the attributes and declaring the position the object was in empty | Requirement was met. |
| The application must allow users to add students to modules. | Creating a class to add students to a list containing modules | An object of the instance ‘module’ should be created containing a list of all modules where students can be added to each individual module. | Requirement was met. |
| The application must allow users to remove students from modules. | Create a subroutine to remove students from the module list. | From the list of students within the module, using list.remove(); will remove the relevant student form the module and the update will then be saved to the database containing the modules. | Requirement was met. |
| The application must allow users to create and delete other types of people, e.g., tutors. | Use an array with a fixed size to store the student / staff objects. | The array will hold the staff and student objects making searching for users easy and when the object the user wants to delete is found, the object’s attributes are set to blank and the position of the object within the database is removed | Requirement was met. |
| The application must allow users to record students’ results. For each assignment it shall record the week it was assigned, the grade, and the feedback. | Create a subroutine that will read and write to the student database. | This will fetch and add information such as the week, grade and module for each student along with the results for each assignment. | Requirement was met. |
| The application must display module class lists. | Fetch the module data form the module database and instantiate the objects. | This will enable the user to query a module and receive the relevant information such as students taking the module, module leader, module moderator and module id. | Requirement was met. |
| Application should have a log in system. | Create a subroutine that fetches the users first name and password. | The subroutine will find a user’s name and password, then decrypt the password and compare it to the input the users used in the password field. | Requirement was met. |
| Passwords should not be easily accessed. | Implement Caesars cipher and encrypt passwords. | Each password will have its own shift value and the individual characters will then be placed in random locations within a random 400-character string to ensure attackers will not be able to find the users password in case there was a data breach. | Requirement was met and each password was given a key generated from the user’s details in order to be able to decrypt the string when it was needed to compare the users input to the password on the log in screen. |

Design

## Overview

The student management system consists of 3 object classes (Student, academics and modules) that fetch data from their respective class databases before instantiating the relevant information into objects stored within arrays. The Student and Academics objects will be instantiated by the person class and certain attributes and methods from the person class will be implementing whilst also having their own individual attributes such as modules taught, Office location, Position, employment status and Students taught for staff, and modules taken for students.

This programme also consists of three databases that possess student details, staff details and module information. If the user wants to add/remove or edit a member of staff, student or module, they can do so using the “application” class that writes to the databases respectively.

## Sequence diagrams

### Search for Staff membersDiagram Description automatically generated

This sequence diagram illustrates how the different classes and databases interact with one another, to return the search query the user has used to look for a member of staff. The academics class will fetch the data from the academics database to find the user if they exist and retrieve their user details before accessing the module class for the module the member of staff is teaching/ moderating. The module class searches the module database and once the module is found, the staff members details are returned to the user along with the modules they teach and the relevant information regarding the staff member.

Figure

### Add/delete students and modulesTable Description automatically generated

Figure

The following Figure illustrates the steps taken when a user adds/ deletes a student or module from the database.

when the user adds a student to the database. A query is sent to the student class with the new students name, surname and other personal information needed to create an object of the student. This information is then passes and written to the student database. The same method is used to remove a student but since the student is already an object that can be found within the student class, the student class rewrites the whole database leaving out the student that needs to be removed before writing the changed to the student database.

The same method for adding and removing a database.

### Search for student information and module information

A picture containing diagram

Description automatically generated

Figure

## User Interface Design layout

This section covers the design of the user interface and the steps taken to achieve the interface that the user interacts with to manipulate the data and information from within the software.

Shape

Description automatically generated

Figure 4

The first steps of creating the Graphical user interface (GUI), were to create a layout of panels to get an idea of the shape and design of the GUI.

The user interface consists of 4 panels with the red centre panel utilising java swing card layouts to change the information displayed on the screen. The card layout system enables the switching between panels and can be triggered when the user interacts with a button within the panel.

Figure 5

Graphical user interface

Description automatically generated

With the layout in place, the next steps were to create the various buttons, text fields and labels that the user will interact with on the card layout and other panels. The first window the user sees when the program is run, is the login page. This separates students, academics and tutors as the different roles come with different powers of control. For instance, a student can only view staff details, module details and other student details. Can cannot edit or alter any information from the various databases whereas the Tutors have the same controls with the additional ability to edit an individual students’ details. A user with the academic’s title can perform the same tasks as a tutor with the additional abilities to create, edit and remove both modules, student and staff members. An academic has full control of the software and can add students to modules or remove them.

Figure 6

Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated

Figure 7

If the student’s login details are correct, they are brought to the student card on the card layout which consists of an output box, combo box and search field. The student can search modules, other student and staff members.

Graphical user interface, text

Description automatically generated

Figure 8

Adding a student accountGraphical user interface

Description automatically generated

Figure 9

Adding a staff account

At any point the user can add new students or staff members and the new user will be stored on the relevant database.

## Object Orientated Design and Concepts

This section talks about object orientated programming techniques and how they were implemented within the student management system to achieve the desired outcome.

### Polymorphism

The first core concept of object orientated programming is Polymorphism. this refers to an object having multiple states that use the same class. This can be seen within the student class. When the database for students is loaded, the information is instantiated and turned into objects and if the getfName() method of an object is called, each object will reply differently as each object has a different first name. (Janssen, 2022)

### Encapsulation

The second core concept is Encapsulation. This refers to the variables of a class being unreachable by other classes with the only form of interaction a programme may have with the variables are through getters and setters. Hiding a variable from other classes can be achieved through setting the modifier of the variable to private which makes means the variable can only be changed / called from within the same class it was created. (Janssen, 2022)

### Inheritance

The third core concept is Inheritance. This can be interpreted as the mechanism in which a class can be derived from another class and often follows a hierarchy in which the child class

Takes on the attributes and methods of the parent class. This can be seen with the person class in the student management system, which acts as the parent class for student and academics. The two subclasses inherit the methods such as getFname() and attributes from the person class as well as implementing their own individual methods and attributes to create an instance of an object such as a student or staff member in this case. (Janssen, 2021)

### Abstraction

The fourth core concept is Abstraction. The main goal for this core concept is to hide the complexities of a class whilst enabling the ability to add more complex logic to that class.

An example of this can be seen with the academic’s class which processes the exchange of information between the objects and classes such as retrieving a student’s first name, last name, course and module. Whereas the graphical user interface, simply passes through the file and name of the student so the application class can locate the student and return the relevant student details. (Janssen, 2021)

### Instantiation

The fifth and last core concept is Instantiation. This simply refers to the creation of an object. An example of this can be seen within the application class where the data is extracted from the relevant database and instances of the student, academics and module objects are created and stored within an array. The values from within the csv files are passed through the respective classes i.e., the student class which assigns the relevant values within the object through a series of setter methods.

## Classes

### A picture containing diagram Description automatically generatedClass: Person

Figure

The person class is the template that the student and Academics class inherit attributes and methods from in order to create a student and Staff object. The class itself is simply a way to create a generic, incomplete person and requires either the student class or academics class in order for an instance of an object to be created.

### Class: Student

The student class passes through the same attributes and methods of the Person class whilst also implementing its own methods and attributes such as getModule(). When the programme is run, parameters from the student csv file are passed through this class, creating instances of student and populating the programme with student objects for the user to interact with.

A picture containing diagram

Description automatically generated

Figure 11

### Class: Academics

A picture containing diagram

Description automatically generated

The academics class operates in the same way the student class does. This object class however passes through different attributes and methods form the student class including the staff members employment status and position whilst also implementing the Person class attributes and methods to create an instance of the academic’s objects.

Figure 12

### Class: Module

A picture containing diagram

Description automatically generated

The modules class converts the information from the modules csv file into module objects that enable the user to add and remove students from. This class does not share any methods or attributes from the other classes as it stores different information and is not considered a person.

Figure 13

### Class: Application

A picture containing diagram

Description automatically generated

Figure

The application class bridges the gap between the Student, Academics and Modules classes and acts as an interface for the other classes to exchange information. This is where the Student, Academics and Module objects are all instantiated when the data from the csv files is extracted, the information is then turned into different instances of the student, academics and module objects respectfully.

### Class: GUI

The Graphical user interface class or GUI handles the way a user will interact with the Student Management system. This class only communicates with the application class to retrieve the relevant information needed from the student, academics and modules database’s before presenting it to the user. This class is vital as it bridges the gap between the user and the software enabling users to search edit and remove information from the databases without the need of the console.

Summary

Overall, the completed project implemented all the basic, intermediate and advanced requirements. One limitation the program has is in the way the objects are stored. The programme utilizes arrays to store the objects once created. This proved to be a problem down the line as arrays are allocated a fixed amount of storage and attempting to go beyond the specified amount results in an index out of bounds error. This proved to be a challenge when attempting to add a new student/ staff member to the database as the array had a pre allocated memory size therefore if the array was full then the user could not be added. Furthermore, when searching for a user the algorithm would search every slot within the array including the empty slots which would bring up an error as the algorithm would try to create objects with the empty fields. A small work around to this problem was using an end of file (EOF) counter which would retain the position of the last user within the database and if a user was added or removed the counter would increase or decrease depending on the situation. A more appropriate solution to this problem would be utilizing array lists as unlike arrays, lists do not have a fixed size and can increase or decrease when a user is added or removed from the database. To avoid running into the same problems for the modules class, a list was used to store the modules and adding/removing students to a module was easily achieved with ‘listname.add(item);’ or ‘listname.remove(item);’. (Bolaji, 2018)

Implementing the Gui was another element of the student management system that caused a lot of problems. As a result, there are elements such as the login screen that would not function in the intended way and constantly caused the system to crash. With further debugging, this problem could be resolved however, with the allocated time this was not possible.

The final hurdle in the project was the encryption method used to store and hide passwords. The method used involves creating and utilising Caesar’s cipher to shift the individual characters of the string before placing them in a 400 character string, generated from the first and last name of the user, with the position of each character stored within the encrypted password(Moore, *Caesar cipher* 2022). This method is very hard to retrieve the original password but does not utilise encryption methods such as DES, RSA and AES therefore making it less secure. (Simplilearn, 2022)

References

Bolaji, B. (2018) *List vs array - data types*, *Medium*. Backticks & Tildes. Available at: https://medium.com/backticks-tildes/list-vs-array-python-data-type-40ac4f294551 (Accessed: December 9, 2022).

Janssen, T. (2021) *OOP concept for beginners: What is abstraction?*, *Stackify*. Available at: https://stackify.com/oop-concept-abstraction/#:~:text=Abstraction%20is%20a%20general%20concept,internal%20details%20provide%20an%20abstraction (Accessed: December 9, 2022).

Janssen, T. (2021) *OOP concept for beginners: What is inheritance?*, *Stackify*. Available at: https://stackify.com/oop-concept-inheritance/#:~:text=Inheritance%20is%20one%20of%20the,set%20of%20attributes%20and%20methods (Accessed: December 9, 2022).

Janssen, T. (2022) *OOP concept for beginners: What is encapsulation*, *Stackify*. Available at: https://stackify.com/oop-concept-for-beginners-what-is-encapsulation/ (Accessed: December 9, 2022).

Janssen, T. (2022) *OOP concepts for beginners: What is polymorphism*, *Stackify*. Available at: https://stackify.com/oop-concept-polymorphism/ (Accessed: December 9, 2022).

Moore, K. (no date) *Caesar cipher*, *Brilliant Math & Science Wiki*. Available at: https://brilliant.org/wiki/caesar-cipher/ (Accessed: December 9, 2022).

Simplilearn, P. (2022) *What is Data Encryption: Algorithms, methods and techniques [2022 edition]: Simplilearn*, *Simplilearn.com*. Simplilearn. Available at: https://www.simplilearn.com/data-encryption-methods-article (Accessed: December 9, 2022).

Appendices

Activity diagram

## Request student details

Diagram

Description automatically generated

Figure 15

## Use case diagram

Diagram

Description automatically generated

Figure 16

This diagrams shows the interactions each user has with the student management system with the students only being able to view staff, student and module information. Tutors having slightly more power as they can add students to the database, module leaders having the same abilities as the tutors with the addition to edit, add and remove students to a module and moderators having full control over the program and being able to edit student details/staff details. Add and remove staff, students and modules.

## Class diagram

Table

Description automatically generated with low confidence

Figure 17

A picture containing diagram

Description automatically generated

Figure 18

Children can’t exist without a parent aggregation. Dark diamond

Children exist with a parent composition

Object diagram

Diagram

Description automatically generated

Figure 19

## ER diagrams

Entity relationship diagrams are a type of flow chart that illustrates how the entities relate to each other within a system.

Diagram

Description automatically generated

Figure 20

The relationship between a module to a module leader and moderator is one to one a module can only have one module leader and one module moderator.

Shape, rectangle

Description automatically generated

Figure 21

The relationship between a tutor and student is one to many as one tutor can have multiple students but a student can only have one tutor

Shape, rectangle

Description automatically generated

Figure 22

The relationship between a student and a module is many to many as one student may have multiple modules and each module contains multiple students.