Software Design Documentation

Design and Implementation of a
Student Information Organizational System (SIOS)

Professor Shaun Gao
CP317 Software Engineering
Group Assignment
Thursday, December 8th, 2022

Group 1
Lubna Al Rifaie [200821590]
Simrah Azfar [200291450]
Mobina Tooranisama [200296720]
Abdul Hafeez Mohammad [200625810]

Table of Contents

1 Introduction

1.1 Document overview

The purpose of this document is to provide the user with an understanding of the design and implementation of the software. The document will provide an overview of the high-level properties of the software, including user interfaces, architectural design, and software design principles. Diagrams made with the unified modeling language (UML) will provide a visual aid to ensure transparency when understanding the development of the software. Additionally, the document will briefly discuss the subject of race conditions within the software. Components of the software will be discussed and provide an in-depth analysis of their function and purpose.

1.2 Purpose of Software

Every individual student contains information pertaining to their academic performance. This information includes grades and enrolled courses. It is crucial to be able to efficiently and reliably calculate final grades for each course that a student is enrolled in. Developing student information organizational software is an effective method of being able to achieve this goal.

1.3 Software Goals

The goal of this software is that it adheres to a design in which inheritance, polymorphism, and encapsulation are utilized for maximum performance capabilities. Additionally, the simplicity and accessibility of this software should allow devices running Windows, Mac, or Linux-based operating systems to perform their duties with minimal disruptions and technical difficulties, as it is intended to be designed with an offensive programming approach. Aimed to be designed and programmed using the Java language, ideally, the software can be easily manipulated to comply with more specific or individual needs its user may have. Organized source code will allow the user to perform necessary alterations to the implementation to accept file types of various data types and input formats, which includes the opportunity to export organized data in more personalized ways that its users may find suitable. Additionally, the capabilities of the software program should be endless as future updates and deployments can include user-friendly' Graphical User Interfaces.

2 Software Architecture Overview

2.1 User Interface

While the software will lack a graphical user interface, the overall simplicity of the software will allow the user to simply use a mouse and keyboard to interact with the program. There is no need for external physical hardware, and the software will function independently without needing additional virtual aid aside from a functioning operating system.

2.2 Architectural Design

The software is intended to interact with the individual components while encapsulating certain functionalities. There is a priority placed on the decomposition of the design into logical components that represent well-defined communication interfaces, which include but are not limited to methods, classes, events, and other properties. When considering these guidelines, the software will follow a component-based architecture. This is due to the system being easier and faster to implement, which will shorten the production period and delivery time. Although certain disadvantages of this architecture type may pose a threat to specific software, this software will not be impacted due to the simplicity of its task and the lack of a network system or database. The disadvantages of component-based architecture are its lack of flexibility and scalability.

3 Software design description

The SIOS software will utilize a three-step approach in its data organization algorithm. There will be three main components, in which each respective section of the software will be responsible for critical data handling and manipulation to achieve the desired output of the data. The input, calculation, and output components will be structured such that they adhere to the policies outlined by the programming practices: inheritance, polymorphism, and encapsulation. Additionally, input and calculation components will utilize offensive programming strategies to prioritize error handling. This will ensure the stability and reliability of the SIOS software and provide its user with detailed error reports when presented with any error, ranging from corrupted data to incorrect file signatures.

3.1 User Data Input

3.1.1 Component interfaces

The user input component will be the only instance in which the software requires communication from the user. The software will intake data from two individual files, both titled “CourseFile” and "NameFile," respectively. Both files will have the text(.txt) format extension and will consist of comma-separated values. The former will contain data related to the course and grades any given student is enrolled in. The latter will contain student information such as names and studentID. Refer to tables 3.1 and 3.1.1 below.

studentID (9 bytes)	CourseID (5 bytes)	Test-1 (2 Bytes)	Test-2 (2 Bytes)	Test-3 (2 Bytes)	Exam (2 Bytes)
321654789	CP317	72.0	96.0	65.5	82.5

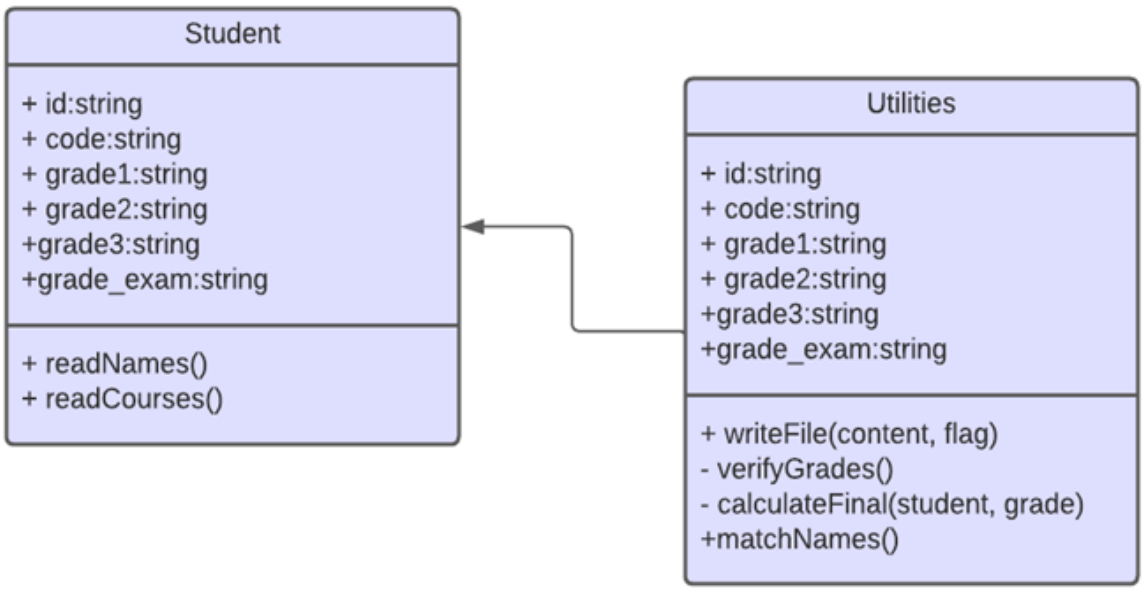
Table 3.1

studentID (9 bytes)	Student Name (Max length 20)
321654789	Chinnaswami Ramagopal

Table 3.1.1

Note: The software will reject incorrectly named files and will not tolerate mis-inputted data.

3.1.2 Component design description



The software we created takes an input file and performs methods and calculations to create an output file. At a high level, the architectural design will be component based. Patterns are high-level techniques that outline how a software system is set up and structured overall. In other words, they specify the various components, classes, and subsystems that make up a software system. The criteria and principles for defining these relationships are also indicated, in addition to the link between the elements. Software architecture is commonly equated with architectural patterns. The high-level design analyzes the architectural design concept into a less abstract perspective of subsystems and modules and shows how they interact with one another. The system's high-level architecture focuses on how each of its parts can be implemented as modules.

The UML diagram shown above demonstrates the structure of the two classes that we implemented in our software. The basis of any object-oriented approach is class diagrams. It displays a system’s classes, along with each class’s attributes, operations, and relationships to other classes. The parent student class has numerous attributes, which the utilities class will inherit. The student class handles the input portion of the software, while the utilities class handles the calculation and output. The program runs on the Microsoft Windows operating system. A UML model's organizational components are shown in the component diagram. These are the sub-boxes that one can break down a model into. They display the actual code's structure. In a design, they simulate tangible elements like the source code and user interface.

3.1.3 Workflows and algorithms

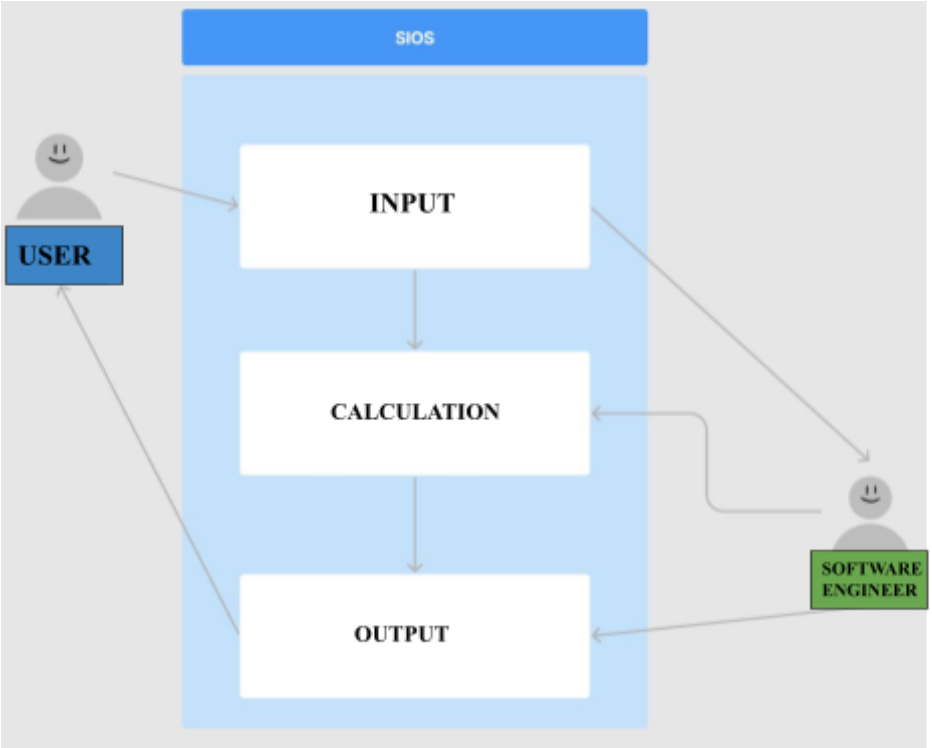
Collaboration diagrams are interaction diagrams that show the design of the objects involved in message sending and receiving. Class, interface, cooperation, use case, active class, components, and nodes are examples of structural elements. In the input component, the diagram will include the use case between the user and the file inputs. “Components are presented on different platforms and several components can cooperate with one another over a communication network in order to achieve a specific objective or goal." (Gao, Shaun).

3.2 Student Grade Calculation

3.2.1 Component interfaces

Using the grades of tests 1-3 and the final grade, we will calculate the student’s average for the given course. This component will be further used in the output file. To calculate the final grade of each student, we use data from the two input files that the user inputted and use a formula to store the final exam grade for each course. The grade calculation should have no errors, meaning they should be in the range of 0-100, and they should not include letters and be all numerical.

3.2.2 Component design description



To get an outside view of the system, we can identify the external and internal factors influencing it, which can be seen in the use case diagram above.

3.2.3 Workflows and algorithms

A sequence diagram is a graphic representation of a situation that illustrates item interplay in a time-based sequence of what occurs first and what follows. The primary distinction between a sequence diagram and a collaboration diagram is that the former depicts time-based interaction, while the latter displays items related to one another. For the grade calculation component, we can use the sequence diagram to illustrate how and in what order items interact with one another. The benefit of the sequence diagram is to display a UML use case's specifics. It also models the logic of the method, observing how elements and objects work together to complete a process. Create a detailed functional plan for a current or hypothetical circumstance.

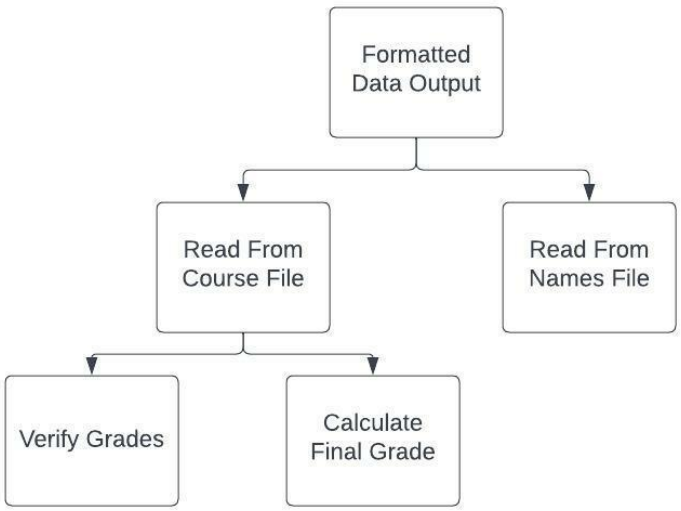
3.3 The Output Component

3.3.1 Component interfaces

After we are done verifying the grades of the students are correct, ensuring there are no errors, we plan to output our data into the output file called Output.txt. The output file will consist of the student's ID (9 bytes), student's name (max 20 bytes), course name (5 bytes) and their overall grade (max 2 bytes) which is calculated in the second component.

3.3.2 Component design description

The formatted output of the file is made to combine the name and the course files to create a clear representation of a student's final grades for the courses they are enrolled in. Using the decomposition design principle, we broke down the tasks we needed to do to get the output file that we needed. The decomposition design principle's purpose is breaking a complex problem or system into parts that are easier to conceive, understand, program, build, and maintain.



4 References

Gao, S. (2022 Sep 22-27). Software Engineering-week-3-1-High-level-design.