**Introduction:**

What is the application?

The application I have created is a Grade Calculator using Java, which is designed to allow students to calculate their actual or expected final grade based on their actual or potential marks for their undergraduate studies (Levels 5 and/or 6). Core functionality is as follows ;

* The ability for users to input a list of modules.
* The ability for users to input a list of Integer Credit values for each module.
* The ability for users to input a list of Integer Marks values for each module.
* The ability for users to modify the number of modules to be accounted for.
* The ability for users to select between combined level 5/6 undergraduate study, and a direct level 6 entry option.
* The ability for users to select existing saved student profiles with pre-filled modules/credits/marks
* The ability to calculate and then display the resultant mark and Award Class of input data, based on Methods A, B, C, and D, along with a “Final Mark” derived from the highest grade achieved.

Key Goals:

* Accuracy.
* Ease of Use.
* User-Friendliness.
* Scalability.
* Maintainability.

Design Principles:

Modularity: The core principle of my application design is modularity. Each relevant functional component is designed to be a self-contained module responsible for a single task. For example, the UI rendering, input validation (in the form of document filters), temporary data storage, and backend operations (In the form of JSON documents) are separated into distinct classes. This modularity ensures that changes in one component do not ripple through the system and cause errors. This also allows me to add new features and aspects to my program as desired, without having to make major changes to existing modules.

User Experience: I have made use of a singular Swing UI window, containing all controls for my application. This ensures that users do not need to search for functionality.

Input/Error handling: I have made use of document filters to ensure that JtextFields within my program accept only valid data types. Module codes can only be entered as strings that are matched against a saved array of level 5 and 6 module codes, with each level being represented separately. Credit and Marks may only be input as integers. Error handling is also implemented in my GradeCalc class, in the form of specific checks to ensure that the following validation checks are passed by all data being calculated:

* Pass Marks: All marks must individually greater than 40, otherwise they result in a failed grade overall.
* Categorical Marks: A further check is done to ensure that all marks are integer values, all non-integer values result in receiving an “N/A” “Grade”. This check is done in addition to documentFiltration.
* Credits per level: Credits for levels 5 and 6 (or just 6 for Direct Entry) are checked to ensure that they total 120 precisely, though credits per module vary.
* Module codes: Module codes are checked to ensure that they match valid module codes, invalid codes result in an “N/A” “Grade”.

Testing and Validation: A suite of Junittesting unit tests have been created alongside my application to ensure correctness. A series of unit tests are run to ensure that the above validation checks are strictly enforced.

Program Structure:

The grade calculator is split into the following major classes:

**Main** – Main class through which load order is scheduled as follows:

* Loading and assigning StudentData and ModuleCodes to their respective variables within DataStorage
* Initialisation of CalcUI swing object.

**GradeCalcCardUI** – Swing UI class that handles the creation and rendering of user-facing UI elements such as labels, text input boxes, checkboxes, and sliders. Contains several inner classes that are used to encapsulate parts of program functionality such as “GradePanel”, “GradeColumn”, “CalculationPanel”, and “CalculationSection”.

**GradeCalc –** Class handling arithmetic and comparison operations, taking data input from GradeCalcCardUI and returning formatted set of results after various validation checks and calculations are performed.

**DataStorage** – Data storage class handling StudentData and both Level5ModuleCodes and Level6ModuleCodes String[] objects that are used in several other classes.

**StudentData** – Data format class used for handling student information.

**JSONHandler –** JSON manipulation class using Gson for loading StudentData and ModuleCodes from file.

**Junit unit testing files –** A series of Junit classes are included that I have used to test various functionality within my program to ensure that everything works as intended.

**UML Diagram:**

A screenshot of a computer screen

Description automatically generated