# Criterion C: Development:

|  |
| --- |
| List of simple techniques used throughout the IA, however not included due to word count restrictions: |
| Encapsulation  If Statements  While loops  For loops  Data hiding  Polymorphism  Simple Data structure (Arrays)  Sequential search Exception handling using try catch blocks  Serialization and deserialization of data to disk |

|  |
| --- |
| **Technique :** Implementing self referential abstract data structures – Singly linked lists |
| Link to success criteria / criterion:   * The client must be able to enter the homeworks, quarter tests and semester exams given to students and mark whether or not they have been completed/the score the student has received on them on a per student basis * The application should be less than 2 GB in total |
| Sources / Notes :  GeeksforGeeks. “Linked List | Set 1 (Introduction) - GeeksforGeeks.” GeeksforGeeks, 8 Mar. 2013, [www.geeksforgeeks.org/linked-list-set-1-introduction/](http://www.geeksforgeeks.org/linked-list-set-1-introduction/).  Drien, Marcos. Singly Linked Lists. 2021.  The class defining the linked list and the nodes utilized were written completely by me. |

### Justification:

The **HomeworkLinkedList** and **AssessmentLinkedList** classes manipulate homework and assessments on per class basis. The student’s assessment scores and homework completion are required, to generate an effort grade, satisfying success criteria. Attributes associated with each homework/assessment are different data types, hence a linked list used to store this in linked nodes.

### Explanation:

The LinkedList classes instantiates a new linked list object with a **start** node set to be null. Both **AssessmentNodeList**s and **HomeworkNodeList**s node types contain self-referential pointers to the **next** node. Traversal was performed sequentially via a temp node through self-references within each node to the next node in the list. The ***addHomework()*** method appends nodes via sequential traversal with 1 temp node. The ***removeHomework()*** method achieved removal via sequential traversal with 2 nodes (see examples). ***addStudent()*** and ***removeStudent()*** implemented to add and remove students from within each nodes’ 2D **ArrayList** of students using their ID as identifiers.

### Example:

### Text Description automatically generated

Figure 1: Initialization and calculating the length of the **HomeworkLinkedList**

### Graphical user interface, text Description automatically generated

Figure 2: Adding a homework to the **HomeworkLinkedList**

### Text Description automatically generated

Figure 3: Removing a homework from the **HomeworkLinkedList**

### Text Description automatically generated

Figure 4: Synchronizing student addition/removals with each node of the **HomeworkLinkedList**

### 

|  |
| --- |
| **Technique :** Implementing self referential abstract data structures – Stack |
| Link to success criteria / criterion:   * The system should be able to generate the effort grade for a student based on test scores, homework completion rate and the client’s perceived effort grade, with a 50%, 25% and 25% weighting to each respective factor * The client must be able to enter the homeworks, quarter tests and semester exams given to students and mark whether or not they have been completed/the score the student has received on them on a per student basis * The client should be able to view and enter the classes they teach and assign students to each class, with the data for each student being separate from other students, as no student can be in 2 classes for the same subject for the same teacher |
| Sources / Notes :  GeeksforGeeks. “Stack Class in Java.” GeeksforGeeks, 4 Feb. 2016, [www.geeksforgeeks.org/stack-class-in-java/](http://www.geeksforgeeks.org/stack-class-in-java/).  The class defining the stack and the nodes utilized were written completely by me. |

### Justification:

Stacks synchronized homework and assessment information from per class to a per student basis. The Exponential Moving Average (EMA) and homework completion components under each **Student** object would need to be recalculated every time assessments/homeworks were added/removed. Data of multiple types of each homework/assessment had to be stored together. Hence, a stack was utilised, with the EMA/Homework completion rate stored as the top, which can simply be popped to get EMA/completion rate data. new EMAs/completion rate can just be pushed to the Stacks.

### Explanation:

In the case of the **AssessmentStack**s, the class instantiates a new Stack object with a *top* node set to be *null*. An initial EMAvalue of 0 is pushed to the stack when it is empty using ***pushEMANode()***. When stack is modified using ***addAssessment()*** or ***removeAssessment()*** or scores within the stack are modified, the *top* node containing the EMA value is popped. Following modification of stack, the EMA of the scores in the stack is recalculated and pushed using ***pushEMANode()***.

The same applies to **HomeworkStack**. Completion rate of homeworks in the stack is calculated using the method ***HomeworkPercentageCompletedHelper.calculatePercentageCompleted()***.

### Example with AssessmentStack:

Text

Description automatically generated

Figure 5: Initialisation of the **AssessmentStack**

Text

Description automatically generated

Figure 6: Calculation for the EMA of all the scores in the **AssessmentStack** and creation of a new **AssessmentNodeStack** with the EMA value to push to the **AssessmentStack**

Text

Description automatically generated

Figure 7: Removing the EMANode from the top of the **AssessmentStack**

Graphical user interface

Description automatically generated

Figure 8: Adding a node to the **AssessmentStack**

Chart

Description automatically generated

Figure 9: Removing an assessment from the AssessmentStack

Graphical user interface, text

Description automatically generatedFigure 10: Viewing the contents of the EMANode without popping it from the **AssessmentStack**

|  |
| --- |
| **Technique :** Data structures: Use of ArrayLists, HashMaps |
| Link to success criteria / criterion:   * The client should be able to view and enter the classes they teach and assign students to each class, with the data for each student being separate from other students, as no student can be in 2 classes for the same subject for the same teacher * The application should be less than 2 GB in total * The client should be able to view the history of effort grade data corresponding to a student * The system should be able to generate the effort grade for a student based on test scores, homework completion rate and the client’s perceived effort grade, with a 50%, 25% and 25% weighting to each respective factor |
| Sources / Notes:  W3Schools. “Java ArrayList.” W3schools.com, 2019, [www.w3schools.com/java/java\_arraylist.asp](http://www.w3schools.com/java/java_arraylist.asp).  ---. “Java HashMap.” W3schools.com, 2019, www.w3schools.com/java/java\_hashmap.asp. |

### Justification:

The main reason I used an **ArrayList** throughout my program was due to cases of needing to store information in a dynamic data structure as the number of elements to be stored can vary, such as in the number of classes a teacher may have. I used **HashMaps** as the return data type for my **AssessmentStack** and **HomeworkStack** to allow for data of two types to be returned in a single object.

### Explanation:

One of the primary uses of **ArrayLists** was an **ArrayList** of **ArrayLists** of doubles and of integers, i.e. a 2D **ArrayList** of student IDs and scores implemented to map each student’s ID to a score on an assessment and of Student IDs and an integer value of 0 or 1 representing a Boolean for homework completed, for each **HomeworkNodeList** and **AssessmentNodeList** object. **HashMaps** were used as return data types to ***popEMANode()*** and ***peekEMAValue()***, to store a String mapping the assessmentName of the EMANode (set to be “EMANode”, if the node contains an EMA, used to verify whether the score being returned is the EMA score), to a Double of the generated EMA score.

### Examples:

Text

Description automatically generated

Figure 11: HashMap return type

### Text Description automatically generated

Figure 12: Method similar to **addStudent()**, implemented in **AssessmentLinkedList**. This is an illustration of one case where a 2D ArrayList was used

|  |
| --- |
| **Technique :** Graphical User Interface |
| Link to success criteria / criterion:   * The client must be able to enter the homeworks, quarter tests and semester exams given to students and mark whether or not they have been completed/the score the student has received on them on a per student basis * The client should be able to view the history of effort grade data corresponding to a student * The application should be less than 2 GB in total * The system should be able to generate the effort grade for a student based on test scores, homework completion rate and the client’s perceived effort grade, with a 50%, 25% and 25% weighting to each respective factor * The client should be able to view and enter the classes they teach and assign students to each class, with the data for each student being separate from other students, as no student can be in 2 classes for the same subject for the same teacher * The system should not allow for incorrect data types, incorrect data format or duplicate data to be entered, i.e. there should be error management in case invalid input is entered |
| Sources / Notes :  Java T point. “Java Swing Tutorial - Javatpoint.” Www.javatpoint.com, www.javatpoint.com/java-swing.  All code is original. I used tutorials from online to learn the basics and wrote the entire GUI myself. |

### Justification:

To increase usability and aesthetic appeal for individuals who find using a menu driven or command line interface difficult. WIMP allows for fluid navigation, easy entry of data and simplified layouts, making the user experience more comfortable.

### Explanation:

The GUI was made in Java swing. Using NetBeans scene builder, the layout of each page was arranged, generating a **.Form** file and a corresponding controller. I utilised **JLabels**, **JComboBoxes**, **JButtons**, **JLists**, **JTabbedPanes**, **JScrollPanes**, **JDateChooser**, **JTextField**, **JPasswordField** and **JTextArea** to develop designs specified in Criter. For validation to prevent entry of data of undesired types, **keyTyped** events were used to monitor the type of data inputted. Calls for serialization and deserialization are made by actions performed on the GUI.

### Examples:

### Graphical user interface, text Description automatically generated

Figure 13: KeyTyped Event actions to ensure only numbers can physically be inputted for IDs

### Text Description automatically generated Text Description automatically generated

Figure 14: Example of various swing components utilised

### Text Description automatically generated[[1]](#endnote-1)

Figure 15: A student can be removed from a **Class** but still be added to back to any **Class** as an existing student, as they are stored in a centralized list of all students when added to the system

### 

|  |
| --- |
| **Technique :** Exponential Moving Average and weighting |
| Link to success criteria / criterion:   * The system should be able to generate the effort grade for a student based on test scores, homework completion rate and the client’s perceived effort grade, with a 50%, 25% and 25% weighting to each respective factor * The system should weight recent test scores more than previous test scores for a student * The system should weight semester exam scores as 70% and average quarter test scores as 30% of the 50% weighting given to the final effort grade |
| Sources / Notes :  Pieter P. “Exponential Moving Average.” Github.io, 2020, tttapa.github.io/Pages/Mathematics/Systems-and-Control-Theory/Digital-filters/Exponential%20Moving%20Average/Exponential-Moving-Average.html. Accessed 16 Feb. 2022.  The program was self written after I understood the theory behind calculating EMA values. |

### Justification:

The **EMA** class generates a weighted moving average, with recent grades being weighted more to allow for a trendline/judgement of a student’s performance to be made, achieving a success criteria. Weightings for each component of the effort grade were performed using client’s requests to achieve the success criteria.

### Explanation:

The **EMA** object calculates a multiplier based on the number of assessments and generates an EMA score based on the scores achieved by the students using the mathematics described in B. Each EMA object has a function calculate ***calculateEMAMultiplier()***, generating the multiplier value based on how many assessments EMA is being calculated for. Weighted average of EMA score for semester and quarter exams is computed using success criteria weightings as test score component of the effort grade. Homework completion rate calculated as percentage of homeworks completed. Each component is weighted on a scale of 0 to 100, and weighted average of each weighted component, specified in success criteria, is used to determine the effort grade. Final weighted value between 0-100 is converted back to effort score in letter using boundaries set by client.

### Example:

Text

Description automatically generated

Figure 16: Implementation of Exponential Moving Average Mathematics to calculate multiplier and final EMA value

Text

Description automatically generated

Figure 17: Calculation for homework completion rate of a student

|  |
| --- |
| **Technique :** Aggregation |
| Link to success criteria / criterion:   * The client must be able to enter the homeworks, quarter tests and semester exams given to students and mark whether or not they have been completed/the score the student has received on them on a per student basis * The client should be able to view and enter the classes they teach and assign students to each class, with the data for each student being separate from other students, as no student can be in 2 classes for the same subject for the same teacher * The client should be able to view the history of effort grade data corresponding to a student |
| Sources / Notes :  Marcos, Drien. Aggregation and Class Design. 2021. |

### Justification:

Aggregation allows for a teacher to be able to access all the classes he teaches, all the students, homeworks and associated assessments. This allows for a more connected system, where all components can be accessed by a single teacher, thus achieving the success criteria for easy editing of information.

### Explanation:

**TeacherManager** class contains **ArrayList** of every **Teacher** in the system. Each **Teacher** object aggregates all classes they teach. Each class contains 2 **AssessmentLinkedLists** for quarter tests and semester exams, 1 **HomeworkLinkedList** of homeworks of the class. Each **node** of the linked lists contrain 2D **ArrayList** of student IDs and scores/integers representing booleans for whether a homework is complete. **Class** class contains an **ArrayList** of **Students**, with each student object aggregating a **HomeworkStack**, 2 **AssessmentStacks** for quarter and semester exams, an **ArrayList** of effort grade influence scores from the teacher and an **ArrayList** of all previous effort grades.

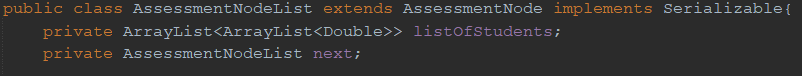
### Examples:

Text

Description automatically generated

Text

Description automatically generated



Text

Description automatically generated

Text

Description automatically generated

|  |
| --- |
| **Technique :** QuickSort |
| Link to success criteria / criterion:   * The client must be able to enter the homeworks, quarter tests and semester exams given to students and mark whether or not they have been completed/the score the student has received on them on a per student basis * The client should be able to view and enter the classes they teach and assign students to each class, with the data for each student being separate from other students, as no student can be in 2 classes for the same subject for the same teacher |
| Sources / Notes:  GeeksforGeeks. “QuickSort - GeeksforGeeks.” GeeksforGeeks, 7 Jan. 2014, www.geeksforgeeks.org/quick-sort/. |

### Justification:

The implementation of Binary Search requires a sorted array, helping the success criteria.

### Explanation:

The classes **AssessmentLinkedList**, **HomeworkLinkedList** and **TeacherManager** (with list of all Teachers) sort using QuickSort.

### Examples:

Text

Description automatically generated

Figures 18: Partition method for 2D ArrayLists of Integers

Text

Description automatically generated

Figure 19: Partition method for 2D ArrayLists of Double

Text

Description automatically generated

Figure 20: Partition method for 2D ArrayLists of Teacher objects

Text

Description automatically generated

Figure 21: Quicksort for 2D ArrayLists of various data types

|  |
| --- |
| **Technique :** Binary Search |
| Link to success criteria / criterion:   * The client must be able to enter the homeworks, quarter tests and semester exams given to students and mark whether or not they have been completed/the score the student has received on them on a per student basis * The client should be able to view and enter the classes they teach and assign students to each class, with the data for each student being separate from other students, as no student can be in 2 classes for the same subject for the same teacher |
| Sources / Notes :  Marcos, Drien. *Binary Search*. 2021. |

### Justification:

To search teachers in **TeacherManager**, and student IDs in **HomeworkLinkedList**/**AssessmentLinkedList**s.

### Explanation:

Binary Search is written in **ArrayListHelper** class.

### Example:

### Text Description automatically generated

Figure 22: Binary Search for data of various types

[Word count: 999]

1. [↑](#endnote-ref-1)