Math Question Generator with different data structures (Hashset, Linkedlist, Arraylist)

Final Project Documentation

Kaneko Kenichi Josiah (2902647462)

Jeremy Nathanael Gunawan (2802522960)

Class L2BC

Computer Science Program

School of Computing and Creative Arts

Jude Joseph Lamug Martinez MCS

Dr. MARIA SERAPHINA ASTRIANI, S.Kom., M.T.I.

Data Structures

Object-Oriented Programming

Bina Nusantara International University

Jakarta

TABLEOFCONTENTS

Background:

In these modern times, mathematics has become a pillar to critical thinking, problem-solving, and logical reasoning skills have continued to increase. However, many have failed to realize that the simplest arithmetic mathematics problems are still as important as a daily necessity (Think Academy, 2023). Traditional teaching methods often rely on static problem sets that rely on a strong basis on its roots; however, that is a problem when. It is shown that students with variability in practice while also experiencing similar but progressively difficult problems have enhanced their retention and adaptability. This underscores the need for accessible tools such as randomized math problem generators- creating endless questions that one can customize the difficulty.

In terms of experience and from others as well, they have experienced a multitude of problems even if the tasks were only arithmetic operations. Some have said that, even though they have finished their exam an hour early, they experienced a lot of careless calculations as the exam does not tolerate calculators. I may even be included to that sort of situation as my marks were affected just because of careless mistakes in simple arithmetic operations. It is not that one could not concentrate that makes these kinds of situations appear, but there are a variety of factors that induce this phenomenon. Such factors mentioned include: the stress induced by exams, the barrage of calculations needed without a calculator that creates misreading, transfer and encoding errors, and miscalculations (Think Academy, 2023).

Creating this kind of program (math question generator) also tackles computational efficiency with educational effectiveness. The system we have developed is programmed in hopes to generate questions rapidly, uniqueness, adapt difficulty, and minimize memory overhead. Generating limitless questions is one thing, but being able to optimize the difficulties of the questions (e.g. increasing number of operands, digits) will aid in providing uniqueness that will prevent repetition.

This project also will tackle evaluating which data structures works best for a given program in circumstances. The data structures to be evaluated include: ArrayList, LinkedList, and Hashset. Furthermore, data structures is extremely important for effective programs to work efficiently in terms of space and time. Being adept to which data structures work best will be the building blocks of effective computer programs. Therefore, finding out and applying the best data structures for this random math question generator is of utmost importance to increase the effectiveness and efficiency of the program, resulting in the perfect math training program.

Problem description

Similarly to the mentioned importance of choosing the right data structure for the most effective program, the right data structure for a program will help various devices and it will ensure that those devices will experience the best performance. This math question generator allows thousands of questions which will be endless, and that more than 7 digits and operands is already a lot to deal with. Inefficient data structures will result the math question generator to be: slow question generation (increasing the digits and operands will take a lot of time especially when asking to generate more than 50 questions), high memory usage that will prevent the program from working and it will display an error as it will have difficulties storing thousands of questions, redundant questions (duplicate questions), and a non-smooth program will result in poor experience.

In this project, we will test 3 different data structures: ArrayList, LinkedList, and Hashset. Each data structure will be compared to one another to obtain data and provide results which data structure is best and their best-performing specific action (e.g. ArrayList possibly being the fastest in terms of generating a lot of questions in terms of digits and operands). We will obtain datas in terms of the speed and memory usage of: handling questions with higher amount of digits and operands, efficiency of generating higher number of questions, and which data structure is best for generating specific operations (addition, subtraction, division, multiplication, and mixed).

Calculations with the help of a specific program created for these programs will help comparing and obtaining an average for runtime speed and memory usage for each function of each data structure. The data structure with the best speed and memory usage will be considered the best data structure for this math question generation program.

Proposed solution

1. Hashset map

a. Interface

We will use the initializeUI method to set up the GUI components, including JSpinner for selecting the number of digits, questions, and operands (Geeksforgeeks, 2021). This method will also configure the layout and add all necessary panels to the frame. initializeUI() assembles all Swing widgets (spinners, check-boxes, buttons, list, labels) and installs them in a BorderLayout.

b. Generating Math Problems

We will use the generateProblems method to create a set of math problems based on user-selected operations. The generated problems will be stored in a Set<Problem> to ensure uniqueness. generateProblems() checks at least one operation, creates random operands, builds a Problem object, and adds it with generated.add(p) to a LinkedHashSet<Problem> so duplicates are automatically skipped.

c. Displaying Selected Question

We will use the showSelectedQuestion method to display the current question in the questionLabel. This method will also update the progressLabel to indicate the current question number and retrieve any previous answer from the userInputs map to populate the answerField. showSelectedQuestion() pulls Problem p = (Problem) problems.toArray()[currentIndex], updates questionLabel, progressLabel, and restores any prior input from the userInputs map.

d. Submitting Answers

We will use the nextQuestion method to process the user's answer when the nextButton is clicked. This method will check if the answer is correct, update the score, and provide feedback in the feedbackLabel. If the user has answered all questions, it will call the showResults method to display the final score. nextQuestion(ActionEvent) reads answerField, validates it, updates score, puts feedback text, then advances currentIndex or ends the quiz.

e. Displaying Results

We will use the showResults method to compile and display the final score along with a summary of each question and the user's answers. This method will iterate through the userInputs map to check each answer against the correct answer stored in the “Problem” objects. showResults() walks userInputs.entrySet(), builds a summary string, and shows it in a JOptionPane.

f. Random Operation Selection

We will use the getRandomOperation method to randomly select an operation based on the user's selections. This method will build a list of available operations and return one at random, allowing flexibility in question generation. getRandomOperation({content}) builds a list of symbols (+ - × ÷), then returns ops.get(random.nextInt(ops.size())).

g. Random Number Generation

We will use the rand method to generate random integers within a specified range. This method will be called when creating random operands for the math problems, ensuring that the generated numbers adhere to the user-defined digit limits. rand(int min,int max,Random r) returns a value in [min,max] and is used to create every operand.

h. Dynamic Question List Management

We will use the DefaultListModel<String> to manage the list of questions displayed in the JList. This model allows for dynamic updates as new questions are generated, ensuring that the user always sees the current set of questions. A DefaultListModel<String> is filled with each new equation (questionListModel.addElement({content}) the JList auto-updates.

i. Handling Mixed Operations

We will use an action listener on the mixedCheck checkbox to enable or disable the operation checkboxes based on whether mixed operations are selected. This will ensure that the user cannot select individual operations when mixed operations are chosen. An ActionListener on mixedCheck toggles the four individual operation check-boxes so they're disabled when "Mixed" is selected.

j. User Input Validation

We will validate user input in the nextQuestion method by checking if the input can be parsed as an integer. If the input is invalid, we will provide appropriate feedback in the feedbackLabel. In nextQuestion a try{ Integer.parseInt({content})} block catches NumberFormatException; invalid input triggers "Invalid input. Ans: {}" feedback.

2. LinkedList

1. User Interface Initialization

We will use the initializeUI method to set up the GUI components, including JSpinner for selecting the number of digits, questions, and operands. This method will also configure the layout and add all necessary panels to the frame which is mirroring the HashSet file.

2. Generating Math Problems

We will use the generateProblems method to create a list of math problems based on user-selected operations. The generated problems will be stored in a LinkedList<Problem> to allow for efficient insertions and deletions. generateProblems() verifies at least one operation, builds each Problem, and appends it to a LinkedList<Problem> with problems.add({content}).

3. Displaying Selected Question

We will use the showSelectedQuestion method to display the current question in the questionLabel. This method will also update the progressLabel to indicate the current question number and retrieve any previous answer from the userInputs list to populate the answerField. showSelectedQuestion() fetches Problem p = problems.get(currentIndex), populates questionLabel, progressLabel, and pre-loads any previous answer from the parallel userInputs list.

4. Submitting Answers

We will use the nextQuestion method to process the user's answer when the nextButton is clicked. This method will check if the answer is correct, update the score, and provide feedback in the feedbackLabel. If the user has answered all questions, it will call the showResults method to display the final score. nextQuestion(ActionEvent) reads and validates the answer, records it in userInputs, adjusts score, writes feedback, and moves to the next index or ends the quiz.

5. Displaying Results

We will use

the showResults method to compile and display the final score along with a summary of each question and the user's answers. This method will iterate through the problems list and the corresponding user inputs to check each answer against the correct answer stored in the “Problem” objects. showResults() iterates from 0 to problems.size()-1, compares each stored answer to p.a(), formats a per-question report, and shows everything in a dialog ui.

6. Random Operation Selection

We will use the getRandomOperation method to randomly select an operation based on the user's selections. This method will build a list of available operations and return one at random, allowing for flexibility in question generation. getRandomOperation({content}) builds a list of allowed symbols and randomly returns one, similar to the HashSet map version.

7. Random Number Generation

We will use the rand method to generate random integers within a specified range. This method will be called when creating random operands for the math problems, ensuring that the generated numbers adhere to the user-defined digit limits. rand(int min,int max,Random r) produces operands within the requested digit range for every question.

8. Dynamic Question List Management

We will use the DefaultListModel<String> to manage the list of questions displayed in the JList. This model allows for dynamic updates as new questions are generated, ensuring that the user always sees the current set of questions. A DefaultListModel<String> feeds the JList; each generated question string is added with questionListModel.addElement({content}).

9. Handling Mixed Operations

We will use an action listener on the mixedCheck checkbox to enable or disable the operation checkboxes based on whether mixed operaListtions are selected. The mixedCheck listener disables or re-enables the individual operation boxes depending on “Mixed” option.

10. User Input Validation

We will validate user input in the nextQuestion method by checking if the input can be parsed as an integer. If the input is invalid, we will provide appropriate feedback in the feedbackLabel.

nextQuestion wraps Integer.parseInt in try/catch. Improper input displays "Invalid input" feedback without crashing.

3. ArrayList

1. User Interface Initialization

We will use the initializeUI method to set up the GUI components, including JSpinner for selecting the number of digits, questions, and operands. This method will also configure the layout and add all necessary panels to the frame. initializeUI() builds all Swing widgets (three JSpinners, five check-boxes, text-field, list, labels, buttons) and lays them out with BorderLayout.

2. Generating Math Problems

We will use the generateProblems method to create a list of math problems based on user-selected operations. This method will check if at least one operation is selected and generate random operands within the specified range. The generated problems will be stored in an ArrayList<Problem>.

generateProblems() verifies an operation is selected, then: clears problems and userInputs,
, builds each equation, appends to ArrayList<Problem> through problems.add({content}), and adds a parallel empty string to userInputs for answer storage.

3. Displaying Selected Question

We will use the showSelectedQuestion method to display the current question in the questionLabel. This method will also update the progressLabel to indicate the current question number and retrieve any previous answer from the userInputs list to populate the answerField. showSelectedQuestion() reads Problem p = problems.get(currentIndex), updates progressLabel, shows the equation text, and restores the stored answer from userInputs.

4. Submitting Answers

We will use the nextQuestion method to process the user's answer when the nextButton is clicked. This method will check if the answer is correct, update the score, and provide feedback in the feedbackLabel. If the user has answered all questions, it will call the showResults method to display the final score. nextQuestion(ActionEvent) saves the current input in userInputs, parses it, updates score, writes feedback, and advances the index or calls showResults().

5. Displaying Results

We will use the showResults method to compile and display the final score along with a summary of each question and the user's answers. This method will iterate through the problems list and the corresponding user inputs to check each answer against the correct answer stored in the “Problem” objects. showResults() iterates i = 0· ·n-1, compares each stored answer to p.getAnswer(), builds a report, and shows it through JOptionPane.

6. Random Operation Selection

We will use the getRandomOperation method to randomly select an operation based on the user's selections. This method will build a list of available operations and return one at random, allowing for flexibility in question generation. getRandomOperation({content}) assembles a temporary ArrayList<String> of the allowed symbols, then returns ops.get(random.nextInt(ops.size())).

7. Random Number Generation

We will use the rand method to generate random integers within a specified range. This method will be called when creating random operands for the math problems, ensuring that the generated numbers adhere to the user-defined digit limits. rand(min,max,rnd) returns min + rnd.nextInt(max-min+1) for each operand; digit count boundaries come from spinners.

8. Dynamic Question List Management

We will use the DefaultListModel<String> to manage the list of questions displayed in the JList. This model allows for dynamic updates as new questions are generated, ensuring that the user always sees the current set of questions. A DefaultListModel<String> feeds the JList; each new equation string is added with questionListModel.addElement(...), so the sidebar updates as soon as problems are generated.

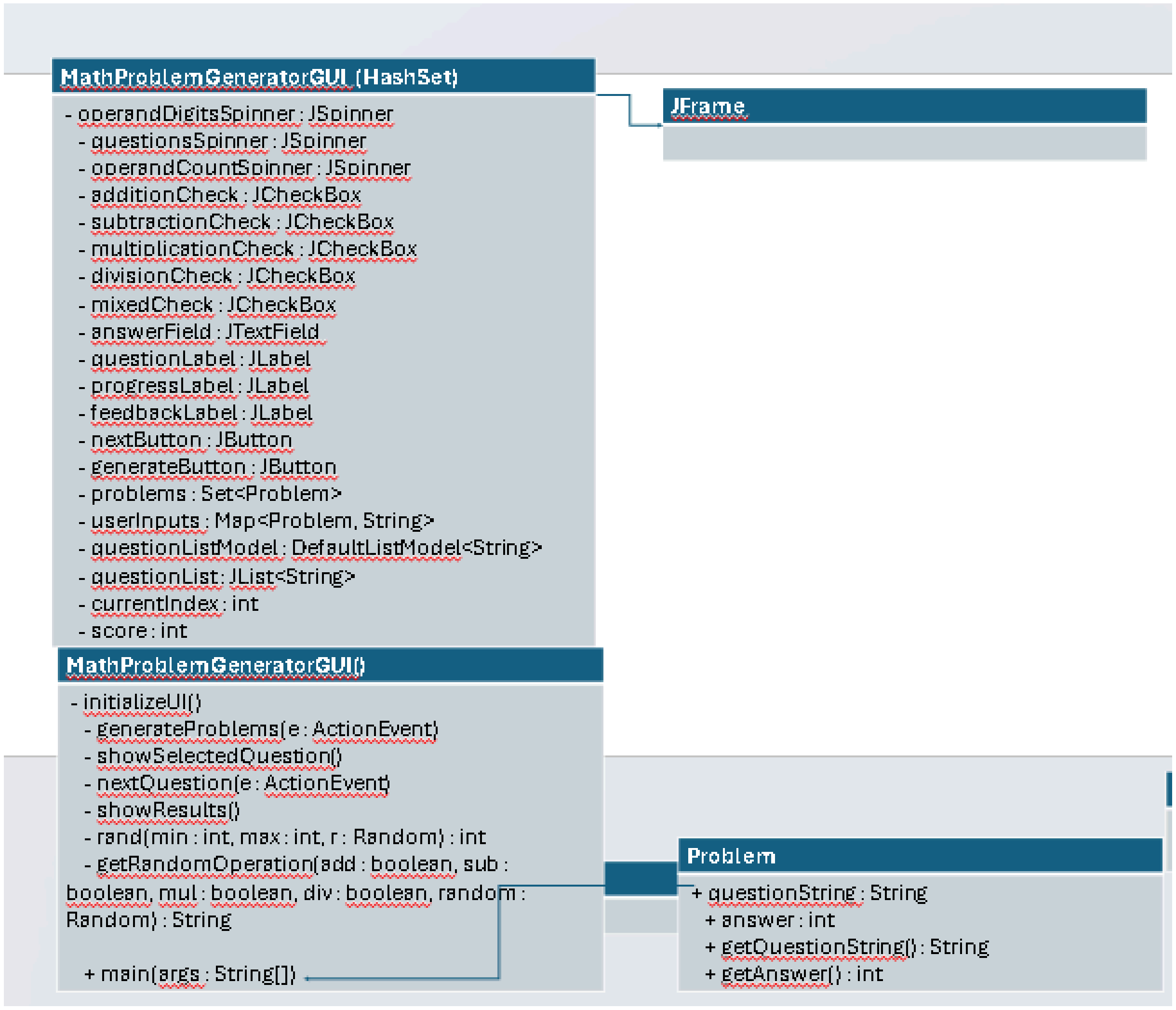
9. Handling Mixed Operations

We will use an action listener on the mixedCheck checkbox to enable or disable the operation checkboxes based on whether mixed operations are selected. This will ensure that the user cannot select individual operations when mixed operations are chosen. An ActionListener on mixedCheck will allow to disables (or re-enables) the four individual operation check-boxes while “Mixed (Random)” is selected.

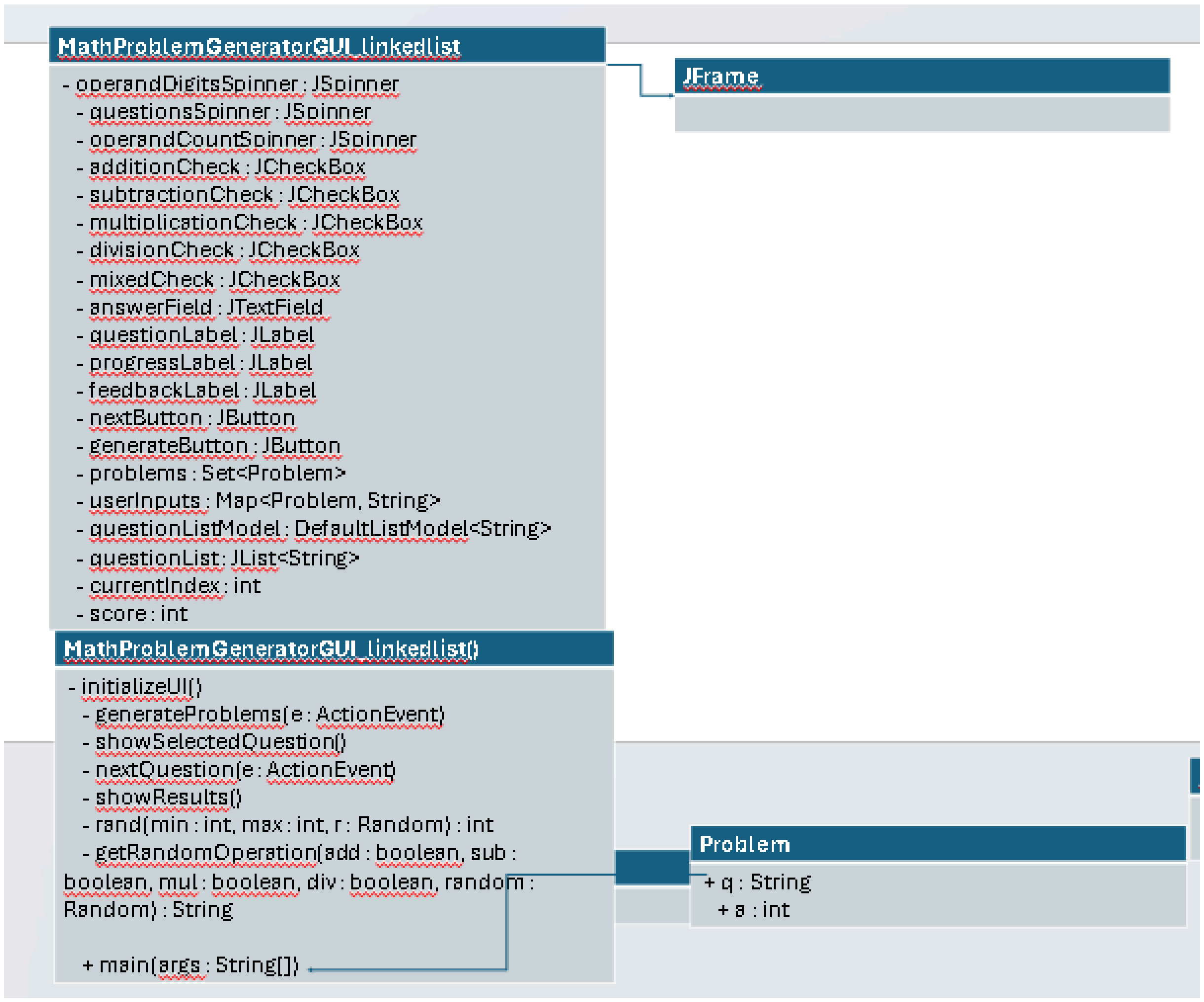
10. User Input Validation

We will validate user input in the nextQuestion method by checking if the input can be parsed as an integer. If the input is invalid, we will provide appropriate feedback in the feedbackLabel. nextQuestion encloses Integer.parseInt(input) in a try/catch; on NumberFormatException it posts “Invalid input. Ans: {content}” and does not crash.

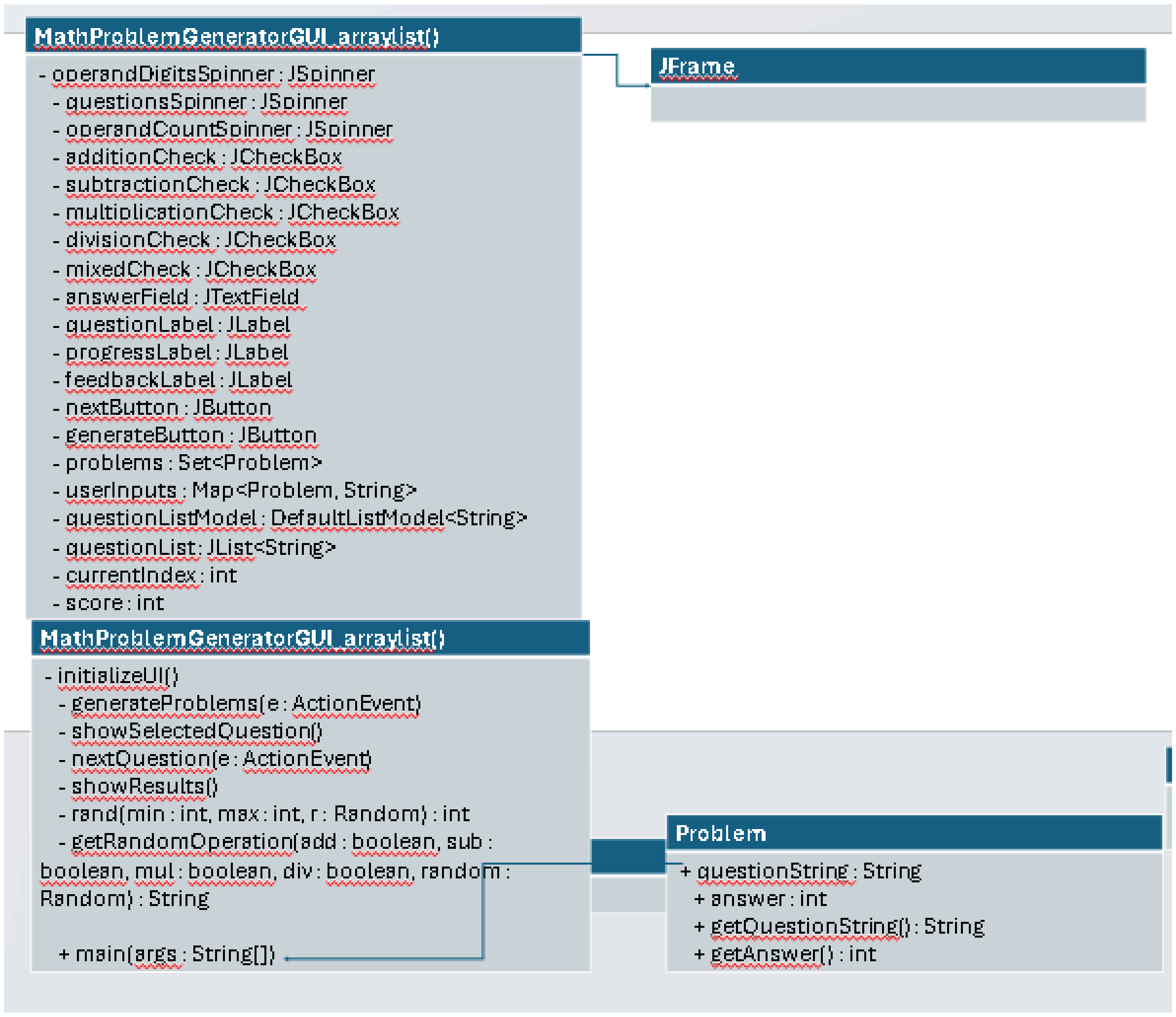
Class UML:
new_hashset:


MathProblemGeneratorGUI (HashSet)
- operandDigitsSpinner : JSpinner
- questionsSpinner : JSpinner
- operandCountSpinner : JSpinner
- additionCheck : JCheckBox
- subtractionCheck : JCheckBox
- multiplicationCheck : JCheckBox
- divisionCheck : JCheckBox
- mixedCheck : JCheckBox
- answerField : JTextField
- questionLabel : JLabel
- progressLabel : JLabel
- feedbackLabel : JLabel
- nextButton : JButton
- generateButton : JButton
- problems : Set<Problem>
- userInputs : Map<Problem, String>
- questionListModel : DefaultListModel<String>
- questionList : JList<String>
- currentIndex : int
- score : int
MathProblemGeneratorGUI()
- initializeUI()
- generateProblems(e : ActionEvent)
- showSelectedQuestion()
- nextQuestion(e : ActionEvent)
- showResults()
- rand(min : int, max : int, r : Random) : int
- getRandomOperation(add : boolean, sub : boolean, mul : boolean, div : boolean, random : Random) : String
+ main(args : String[])
JFrame
Problem
+ questionString : String
+ answer : int
+ getQuestionString() : String
+ getAnswer() : int

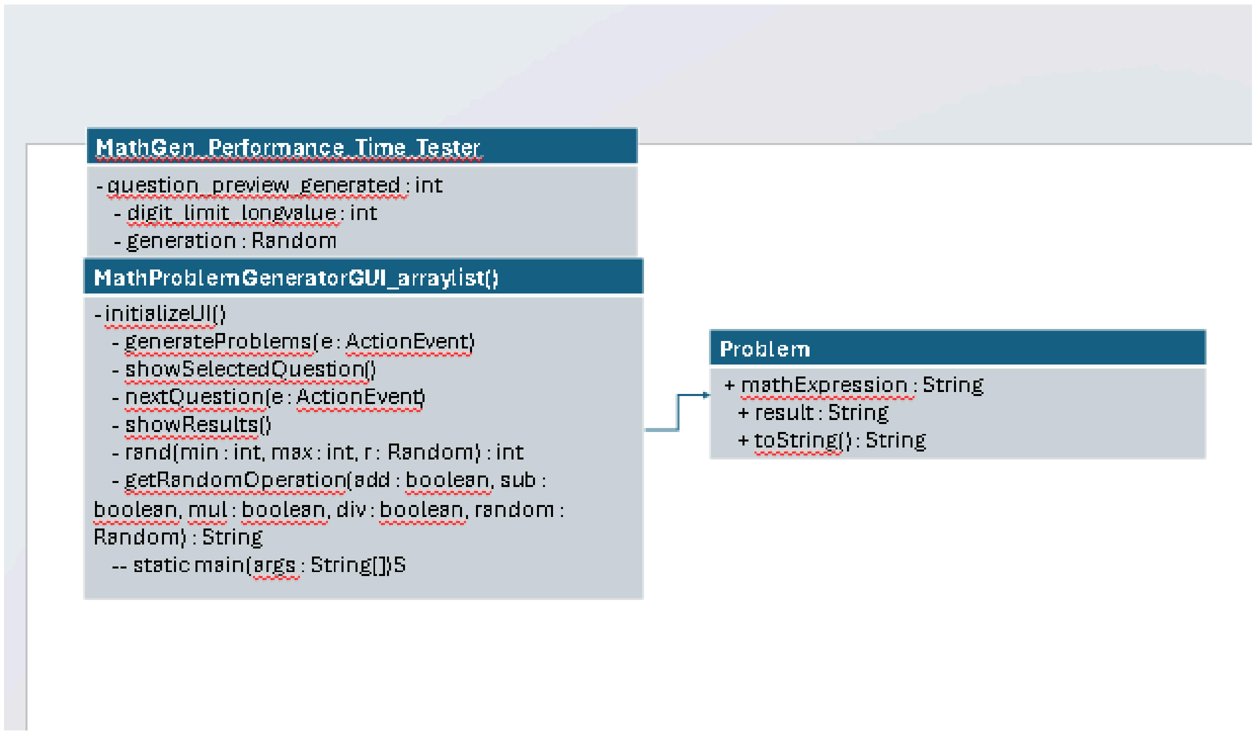
new_linkedlist:


MathProblemGeneratorGUI_linkedlist
- operandDigitsSpinner : JSpinner
- questionsSpinner : JSpinner
- operandCountSpinner : JSpinner
- additionCheck : JCheckBox
- subtractionCheck : JCheckBox
- multiplicationCheck : JCheckBox
- divisionCheck : JCheckBox
- mixedCheck : JCheckBox
- answerField : JTextField
- questionLabel : JLabel
- progressLabel : JLabel
- feedbackLabel : JLabel
- nextButton : JButton
- generateButton : JButton
- problems : Set<Problem>
- userInputs : Map<Problem, String>
- questionListModel : DefaultListModel<String>
- questionList: JList<String>
- currentIndex : int
- score : int
MathProblemGeneratorGUI_linkedlist()
- initializeUI()
- generateProblems(e : ActionEvent)
- showSelectedQuestion()
- nextQuestion(e : ActionEvent)
- showResults()
- rand(min : int, max : int, r : Random) : int
- getRandomOperation(add : boolean, sub : boolean, mul : boolean, div : boolean, random : Random) : String
+ main(args : String[])
JFrame
Problem
+ q : String
+ a : int

new_arraylist:


MathProblemGeneratorGUI_arraylist()
- operandDigitsSpinner : JSpinner
- questionsSpinner : JSpinner
- operandCountSpinner : JSpinner
- additionCheck : JCheckBox
- subtractionCheck : JCheckBox
- multiplicationCheck : JCheckBox
- divisionCheck : JCheckBox
- mixedCheck : JCheckBox
- answerField : JTextField
- questionLabel : JLabel
- progressLabel : JLabel
- feedbackLabel : JLabel
- nextButton : JButton
- generateButton : JButton
- problems : Set<Problem>
- userInputs : Map<Problem, String>
- questionListModel : DefaultListModel<String>
- questionList: JList<String>
- currentIndex : int
- score : int
JFrame
MathProblemGeneratorGUI_arraylist()
- initializeUI()
- generateProblems(e : ActionEvent)
- showSelectedQuestion()
- nextQuestion(e : ActionEvent)
- showResults()
- rand(min : int, max : int, r : Random) : int
- getRandomOperation(add : boolean, sub : boolean, mul : boolean, div : boolean, random : Random) : String
+ main(args : String[])
Problem
+ questionString : String
+ answer : int
+ getQuestionString() : String
+ getAnswer() : int

MathGen_Performance_Time_Tester:


MathGen_Performance_Time_Tester
- question_preview_generated : int
- digit_limit_longvalue : int
- generation : Random
MathProblemGeneratorGUI_arraylist()
- initializeUI()
- generateProblems(e : ActionEvent)
- showSelectedQuestion()
- nextQuestion(e : ActionEvent)
- showResults()
- rand(min : int, max : int, r : Random) : int
- getRandomOperation(add : boolean, sub : boolean, mul : boolean, div : boolean, random : Random) : String
-- static main(args : String[])S
Problem
+ mathExpression : String
+ result : String
+ toString() : String

Testing:

Average Time in increasing amount of generating questions but with digits per operand (20) and operand per question (20)/ millisecond (ms)

Operation type	Number of questions generated	ArrayList	LinkedList	HashSet map	Best data structure for this
Mix	500 000	5825.05	6363.05	5950.99	ArrayList
Mix	1 000 000	9296.66	9961.45	9973.29	ArrayList
Mix	1,500,000	15177.73	17779.86	17156.27	ArrayList
Mix	2 000 000	25680.03	24398.70	49289.42	LinkedList
Addition	500 000	3899.82	6604.64	11544.53	ArrayList
Addition	1 000 000	22657.42	19905.35	28910.92	LinkedList
Addition	1 500 000	10632.51	11947.81	11534.47	ArrayList
Addition	2 000 000	34946.14	53038.12	50548.32	ArrayList
Subtraction	500 000	3916.55	4088.03	4054.67	ArrayList
Subtraction	1 000 000	7711.33	8416.29	8042.52	ArrayList
Subtraction	1 500 000	11232.71	12333.52	16767.80	ArrayList
Subtraction	2 000 000	26648.32	16322.06	19487.99	LinkedList
Multiplication	500 000	7679.67	7832.13	7960.09	ArrayList
Multiplication	1 000 000	15977.50	16060.73	16380.24	ArrayList

Multiplication	1 500 000	35914.45	24198.91	26166.65	LinkedList
Multiplication	2 000 000	42367.25	32639.85	33468.05	LinkedList
Division	500 000	3710.02	3950.70	3836.21	ArrayList
Division	1 000 000	7504.45	13893.67	7504.45	ArrayList/ HashSet
Division	1 500 000	18105.60	11914.23	11236.38	HashSet Map
Division	2 000 000	14563.02	16189.95	15957.02	ArrayList
Average(sigma)/count:	X	14,342.74	15,217.46	16,540.74	ArrayList

Average Time in increasing amount of digits per operand but with constant amount of questions generate (500 000) and operand (20)/ millisecond (ms)

Operation type	Digits per operand	ArrayList	LinkedList	HashSet map	Best data structure for this
Mix	20	5825.05	6363.05	5950.99	ArrayList
Mix	40	21420.77	21251.78	21392.48	LinkedList
Mix	60	27211.37	30499.88	53338.64	ArrayList
Mix	80	65693.96	41311.77	42706.30	LinkedList
Addition	20	3899.82	6604.64	11544.53	ArrayList
Addition	40	27653.08	32581.32	18086.69	HashSet Map
Addition	60	38788.39	44320.10	32331.44	HashSet Map
Addition	80	42975.05	59136.14	47002.18	ArrayList
Subtraction	20	3916.55	4088.03	4054.67	ArrayList
Subtraction	40	12856.86	10146.76	9064.47	HashSet Map
Subtraction	60	12176.62	37582.30	21082.68	ArrayList
Subtraction	80	15017.88	15186.89	15623.70	ArrayList
Multiplication	20	7679.67	7832.13	7960.09	ArrayList
Multiplication	40	18341.25	18565.14	18737.56	ArrayList
Multiplication	60	67142.87	30004.14	30674.03	LinkedList
Multiplication	80	44411.74	41405.21	43234.04	LinkedList
Division	20	3710.02	3950.70	3836.21	ArrayList
Division	40	19146.24	20051.38	19228.81	ArrayList
Division	60	27480.03	31056.27	38408.13	ArrayList
Division	80	44502.07	38644.57	54534.17	LinkedList
Average(sigma/count):	X	25,078.71	24,623.60	25,083.86	LinkedList

Average Time in increasing amount of operands per question but with constant number of digits per operand (20) and amount of questions generated(500 000)/ millisecond (ms)

Operation type	Number of operand per question	ArrayList	LinkedList	HashSet map	Best data structure
Mix	20	5825.05	6363.05	5950.99	ArrayList
Mix	40	20694.19	22204.42	37149.71	ArrayList
Mix	60	46273.69	45022.97	34155.13	HashSet Map
Mix	80	41949.89	67312.11	49964.25	ArrayList
Addition	20	3899.82	6604.64	11544.53	ArrayList
Addition	40	20296.74	27515.13	26284.71	ArrayList
Addition	60	23255.44	37500.81	53435.34	ArrayList
Addition	80	62266.67	52326.18	35171.39	HashSet map
Subtraction	20	8937,61	9172,98	8443,54	HashSet map
Subtraction	40	22078.38	16058.53	22034.13	LinkedLists
Subtraction	60	29077.50	24065.81	29417.47	LinkedLists
Subtraction	80	33972.35	56708.28	53438.53	ArrayList
Multiplication	20	7679.67	7832.13	7960.09	ArrayList
Multiplication	40	74653.39	57118.89	33611.82	HashSet Map
Multiplication	60	35540.91	32887.96	42842.79	LinkedList
Multiplication	80	57149.70	58830.36	66841.87	ArrayList
Division	20	3710.02	3950.70	3836.21	ArrayList
Division	40	7341.60	7749.21	7269.03	HashSet Map
Division	60	9896.88	10240.89	21735.63	ArrayList
Division	80	25565.88	13437.95	15140.34	LinkedList
Average(sigma/count):	X	24,623.60	25,078.71	25,083.86	ArrayList

Memory usage increasing questions / byte

Operation type	Number of questions generated	ArrayList	LinkedList	HashSet map	Best data structure
Mix	500 000	323,111,736	331,413,544	341,653,872	LinkedList
Mix	1 000 000	628,352,040	645,775,536	664,169,048	ArrayList
Mix	1 500 000	941,554,728	969,360,392	991,924,584	ArrayList
Mix	2 000 000	1,256,383,048	1,293,148,544	1,327,960,208	ArrayList

Addition	500 000	300,703,888	308,736,280	318,882,408	ArrayList
Addition	1 000 000	601,138,128	616,745,816	637,919,424	ArrayList
Addition	1 500 000	924,915,624	897,185,296	947,859,600	LinkedList
Addition	2 000 000	1,233,624,928	1,197,688,528	1,269,197,048	LinkedList
Subtraction	500 000	300,285,512	308,197,128	318,921,368	ArrayList
Subtraction	1 000 000	598,696,632	617,084,416	635,209,136	ArrayList
Subtraction	1 500 000	897,223,512	925,378,376	947,737,704	ArrayList
Subtraction	2 000 000	1,198,439,360	1,233,499,800	1,268,638,360	ArrayList
Multiplication	500 000	488,254,664	496,602,992	506,828,040	ArrayList
Multiplication	1 000 000	975,360,376	992,842,880	1,011,294,336	ArrayList
Multiplication	1 500 000	1,461,434,704	1,488,632,440	1,511,989,224	ArrayList
Multiplication	2 000 000	1,950,412,936	1,985,152,448	2,020,451,656	ArrayList
Division	500 000	268,107,240	276,431,160	286,796,576	ArrayList
Division	1 000 000	535,221,904	553,085,552	571,409,632	ArrayList
Division	1 500 000	801,600,520	829,322,344	851,634,208	ArrayList
Division	2 000 000	1,070,374,528	1,105,175,904	1,140,651,808	ArrayList
Average(sigma/count);	X	802,456,023	821,313,016	845,441,832	ArrayList

Memory Usage increasing digits per operand/ bytes

Operation type	Digits per Operand	ArrayList	LinkedList	HashSet map	Best Data Structure
Mix	20	315,486,976	323,226,184	335,842,544	ArrayList
Mix	40	542,338,640	550,612,696	562,423,928	ArrayList
Mix	60	770,614,464	776,930,712	789,461,744	ArrayList
Mix	80	997,839,072	1,004,277,072	1,004,277,072	ArrayList
Addition	20	300,490,160	308,398,680	321,358,584	ArrayList
Addition	40	513,275,160	521,311,664	533,403,760	ArrayList
Addition	60	722,239,712	730,050,688	743,443,760	ArrayList
Addition	80	936,747,880	942,665,576	955,075,504	ArrayList

Subtraction	20	300,463,552	308,210,632	320,786,888	ArrayList
Subtraction	40	513,164,008	520,882,864	534,283,776	ArrayList
Subtraction	60	723,107,320	730,416,680	742,979,416	ArrayList
Subtraction	80	936,265,096	941,941,088	956,507,208	ArrayList
Multiplication	20	489,006,736	496,025,080	509,440,640	ArrayList
Multiplication	40	890,889,352	898,841,136	910,579,368	ArrayList
Multiplication	60	1,294,752,136	1,300,791,848	1,315,175,520	ArrayList
Multiplication	80	1,696,942,400	1,705,832,992	1,715,584,240	ArrayList
Division	20	268,054,408	276,512,272	289,316,880	ArrayList
Division	40	469,241,896	477,193,744	489,789,544	ArrayList
Division	60	669,584,960	677,953,560	691,395,704	ArrayList
Division	80	872,886,184	878,258,848	892,525,192	ArrayList
Average	X	711,126,456	719,411,596	730,903,414	ArrayList

Operation type	Number of operands per question	ArrayList	LinkedList	HashSet map	Best data structure list
Mix	20	314,649,736	323,208,240	335,450,552	ArrayList
Mix	40	554,907,616	562,505,016	575,136,928	ArrayList
Mix	60	797,057,112	803,379,320	816,406,168	ArrayList
Mix	80	1,033,590,232	1,039,488,704	1,052,875,488	ArrayList
Addition	20	300,383,416	308,502,984	320,639,080	ArrayList
Addition	40	528,349,472	537,223,032	550,124,136	ArrayList
Addition	60	762,169,336	769,674,608	783,153,904	ArrayList
Addition	80	993,152,304	998,796,952	1,012,189,136	ArrayList
Subtraction	20	300,216,840	306,639,000	321,538,064	ArrayList
Subtraction	40	529,384,856	537,083,160	550,404,120	ArrayList
Subtraction	60	762,409,512	769,535,552	782,746,584	ArrayList
Subtraction	80	991,776,704	998,691,568	1,011,842,968	ArrayList
Multiplication	20	489,241,248	496,820,368	509,097,080	ArrayList
Multiplication	40	916,604,728	922,559,160	937,305,232	ArrayList
Multiplication	60	1,347,496,192	1,355,516,680	1,355,516,680	ArrayList

Multiplication	80	1,775,855,088	1,783,079,848	1,791,849,768	ArrayList
Division	20	268.654.208	276.340.080	286.644.688	ArrayList
Division	40	497.162.104	504.860.672	515.579.616	ArrayList
Division	60	729.945.552	737.946.880	748.549.960	ArrayList
Division	80	959.115.040	967.239.256	977.620.376	ArrayList
Average	X	792,656,044	799,904,454	811,683,106	ArrayList

Result

ArrayList scored the best and performed best.

ArrayList performed better in majority of the testing.

1. 500,000 Questions 20 operand 20 digits

ArrayList: Time = 1.00, Memory = 1.00

LinkedList: Time = 6363.05 / 5825.05 = 1.09, Memory = 331413544 / 323111736 = 1.03

HashSet: Time = 5950.99 / 5825.05 = 1.02, Memory = 341653872 / 323111736 = 1.06

Time Efficiency:

LinkedList is 9% slower than ArrayList

HashSet Map is 2% slower than ArrayList

Memory Efficiency:

LinkedList uses 3% more memory than ArrayList

HashSet Map uses 6% more memory than ArrayList

Performance Trade-offs:

ArrayList shows the best balance of speed and memory usage

HashSet Map offers slightly better time performance than LinkedList (7% faster) but at higher memory cost

LinkedList is the worst performer in both metrics for this dataset

Efficiency Score (Time × Memory):

ArrayList: 1.00 (baseline)

LinkedList: 1.09 × 1.03 = 1.12

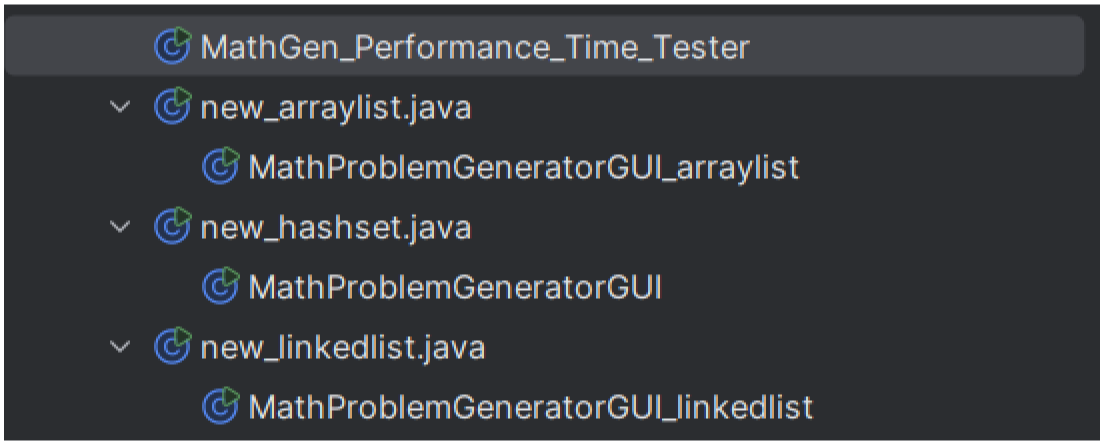
HashSet Map: 1.02 × 1.06 = 1.08

ArrayList has better cache locality, memory-efficient, and easier implementation. In addition, the ArrayList data structure stored elements in a single, contiguous block of memory- which reduces the overhead (Vogel, 2008). Another reason that ArrayList scored best for memory usage is that when compared to LinkedList, LinkedLists use node-based storage and doubly-linked goes prev and next. This increases memory usage by LinkedList by an approximate of 12% (based from testing). In terms of Hashset, it is heavier by around 6% than ArrayList due to hashing overhead (Tutorialspoint, n.d.).

Demonstration

Video link demonstration:
https://drive.google.com/file/d/14m16uUc0804QjOozKlbptiwGgGvOAdlo/view?usp=drivesdk

Demosntration#0:

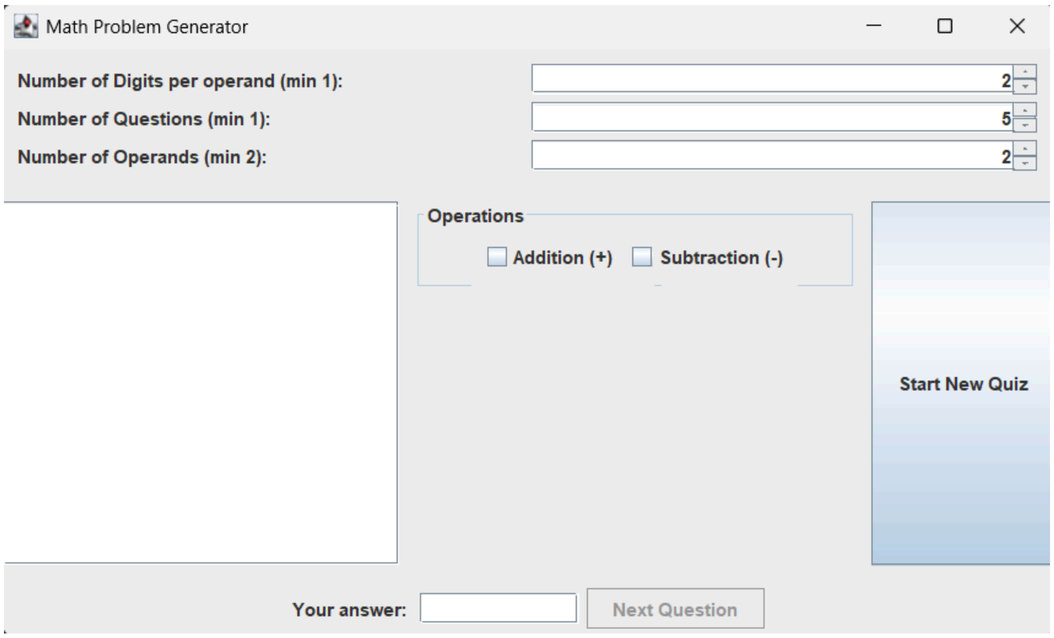


```
MathGen_Performance_Time_Tester
new_arraylist.java
    MathProblemGeneratorGUI_arraylist
new_hashset.java
    MathProblemGeneratorGUI
new_linkedlist.java
    MathProblemGeneratorGUI_linkedlist
```

There are 3 files that each function as the random math generator but with different data structures: Hashset, LinkedList, and Arraylist. The MathGen_Performance_Time_Tester is optional to be used, the main function (though self-explanatory) is to provide the time-complexity and space-complexity of the different data structures.

Firstly, upon executing any of the 3 files (the math question generator programs), it will open a popup window like below (Demonstration#1). However, the program is supposed to be utilized in full screen (Demonstration#2).

Demonstration#1:



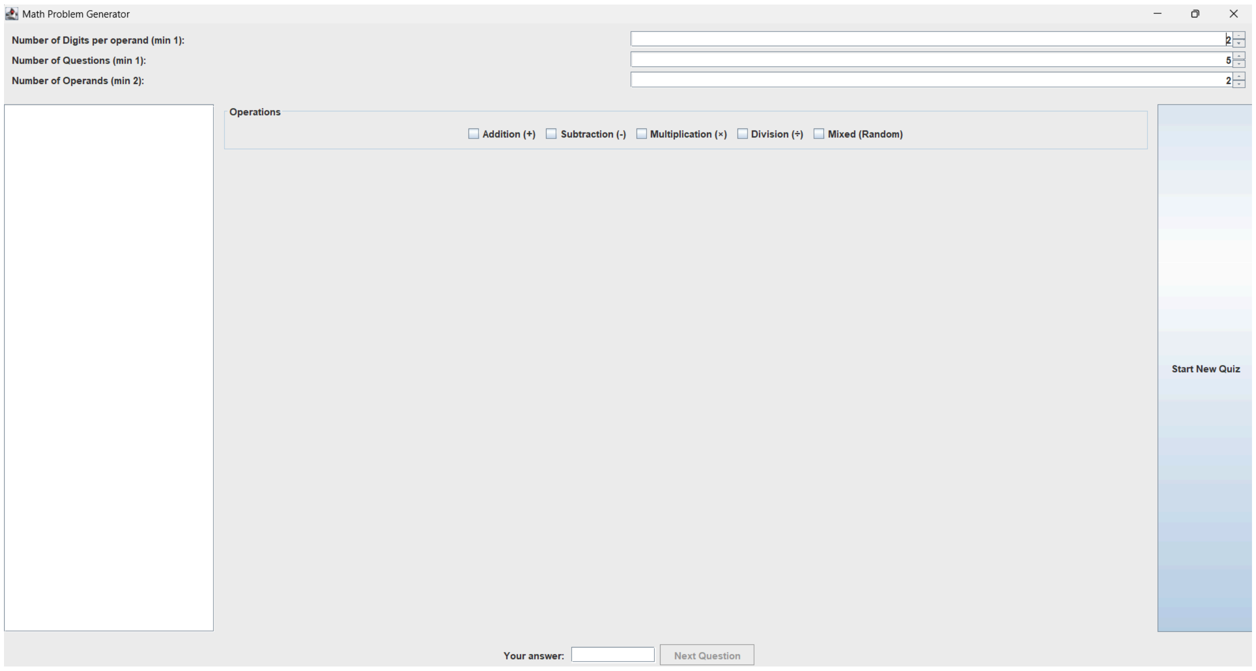
Math Problem Generator

Number of Digits per operand (min 1): 2

Number of Questions (min 1): 5

Number of Operands (min 2): 2

Demonstration#2:



Math Problem Generator

Number of Digits per operand (min 1): 2

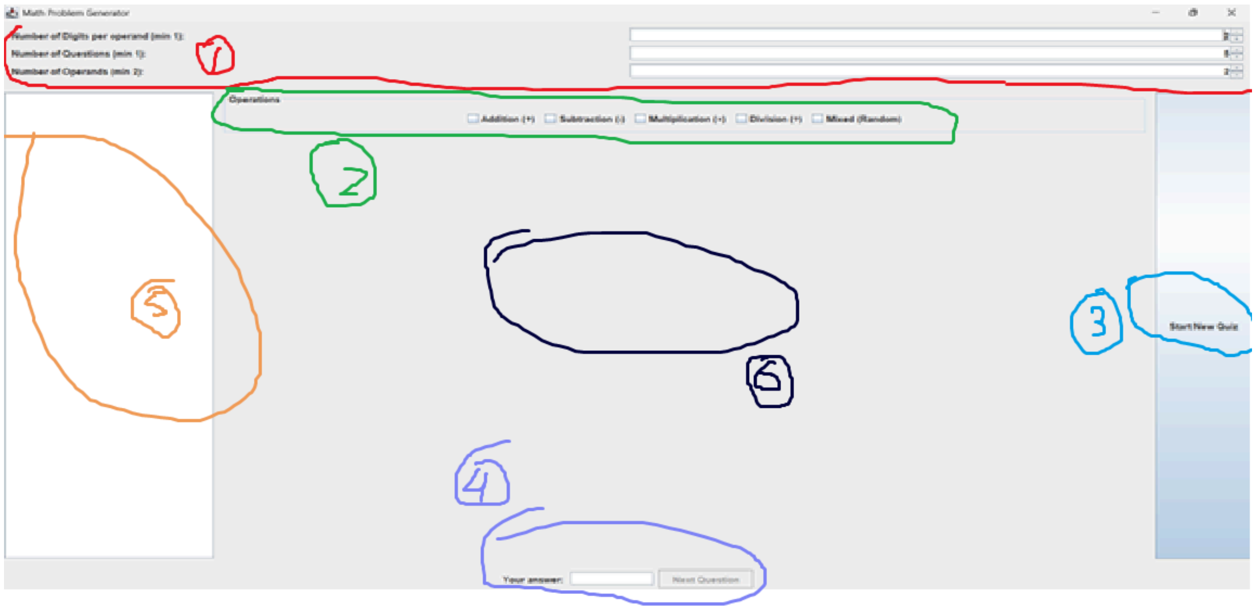
Number of Questions (min 1): 5

Number of Operands (min 2): 2

Operations

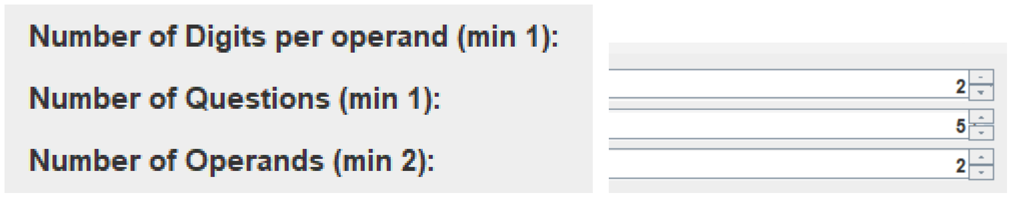
☐ Addition (+) ☐ Subtraction (-) ☐ Multiplication (×) ☐ Division (÷) ☐ Mixed (Random)

Demonstration#3:



Once set to full screen, there are various functionalities and features that you will be able to see and experience. From the Demonstration#3, shows the number of functionalities the program has that all 3 files share.

Demonstration#4:

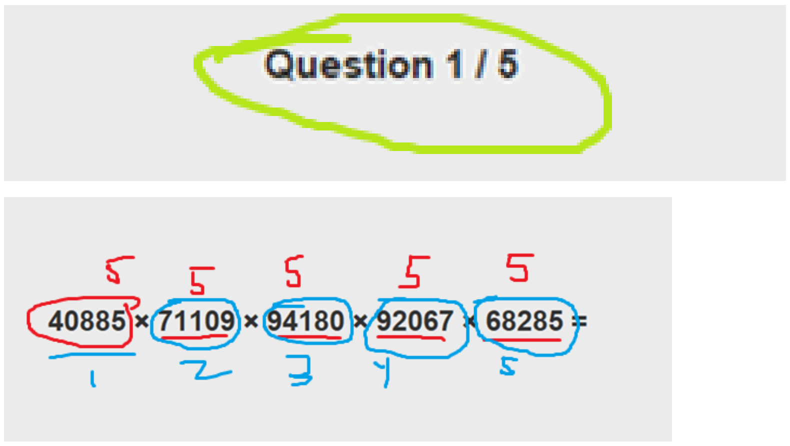

Number of Digits per operand (min 1): 2

Number of Questions (min 1): 5

Number of Operands (min 2): 2

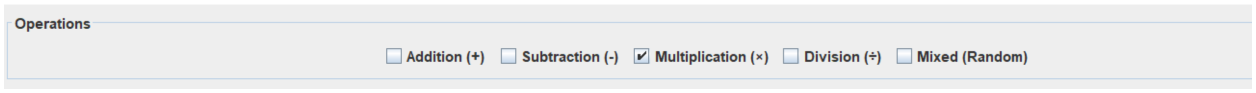
The above is the first functionality. Users will be able to customize the number of digits, operands, and number of questions to be generated. To clarify, the number of digits means the number of digits per between operand- highlighted red. The number of operands mean the number of values with an operator symbol to be generated in each question- highlighted blue. Finally, the number of questions generated- highlighted green. In here, the number of questions, digits, and operand are all 5.

Demonstration#5:


Question 1 / 5

40885 × 71109 × 94180 × 92067 × 68285 =

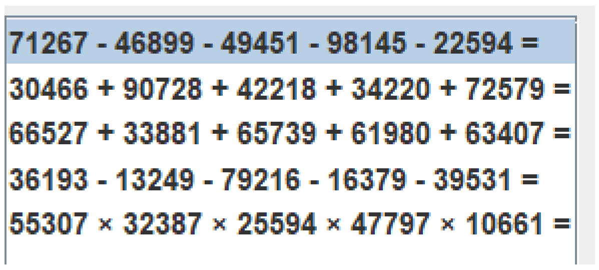
Demonstration#6:


Operations

Addition (+) Subtraction (-) [✓ Multiplication (×)] Division (÷) Mixed (Random)

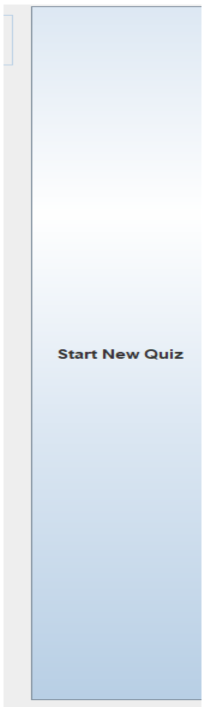
Here is the second functionality. This requires users to provide input on which kind of operation for the questions that they would like to answer or have the questions generated. The functionality only works with only one input, meaning that even if the program allows multiple inputs, only one input will be taken. For example, ticking both multiplication and division means that recent tick will be submitted. As a heads up, the mixed category means that for each question, they will have only one focused operation symbol but the next question will be different or randomized. To summarize, the program only allows one operation input with mixed having each question having a randomized operation but in one question it will have the same operation. In Demonstration#7, this is a mixed operation generated question with 5 operand, 5 digits, and 5 questions.

Demonstration#7:



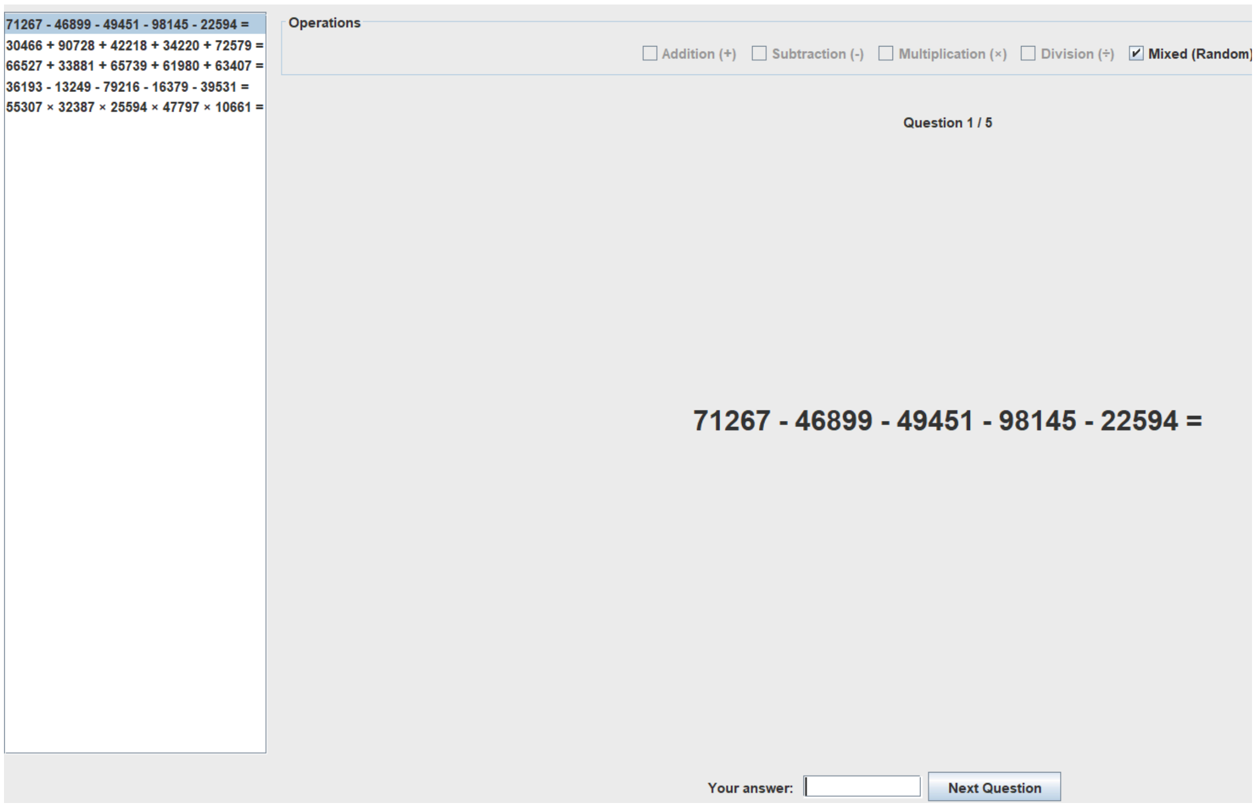
```
71267 - 46899 - 49451 - 98145 - 22594 =
30466 + 90728 + 42218 + 34220 + 72579 =
66527 + 33881 + 65739 + 61980 + 63407 =
36193 - 13249 - 79216 - 16379 - 39531 =
55307 × 32387 × 25594 × 47797 × 10661 =
```

Demonstration#8:



Demonstration#8 is the third functionality of the program. It is a simple button that starts generating the questions once and only if functionalities 1 and 2 has a user input already.

Demonstration#9:



71267 - 46899 - 49451 - 98145 - 22594 =
30466 + 90728 + 42218 + 34220 + 72579 =
66527 + 33881 + 65739 + 61980 + 63407 =
36193 - 13249 - 79216 - 16379 - 39531 =
55307 × 32387 × 25594 × 47797 × 10661 =

Operations

☐ Addition (+) ☐ Subtraction (-) ☐ Multiplication (×) ☐ Division (÷) ☑ Mixed (Random)

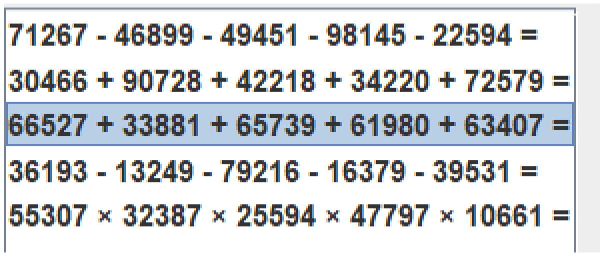
Question 1 / 5

71267 - 46899 - 49451 - 98145 - 22594 =

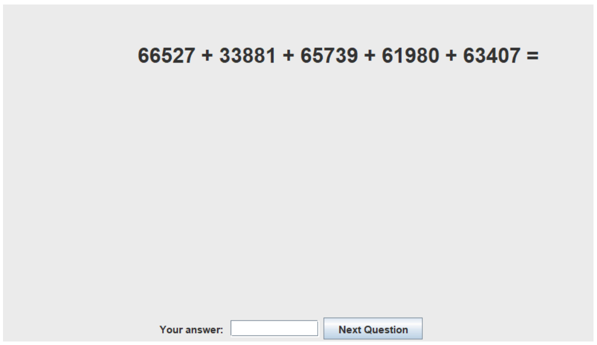
Your answer:

In the image above, 3 new functionalities are displayed, this includes: Next question and your answer, the column box at the left, and finally the space where it indicates which question is being focused and answered right now.

Demonstration#10:



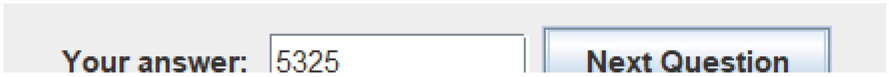
71267 - 46899 - 49451 - 98145 - 22594 =
30466 + 90728 + 42218 + 34220 + 72579 =
66527 + 33881 + 65739 + 61980 + 63407 =
36193 - 13249 - 79216 - 16379 - 39531 =
55307 × 32387 × 25594 × 47797 × 10661 =



66527 + 33881 + 65739 + 61980 + 63407 =

Your answer:

In the left box, users could actually click which question to focus on. Simply clicking it once will change which question is being focused on right now.

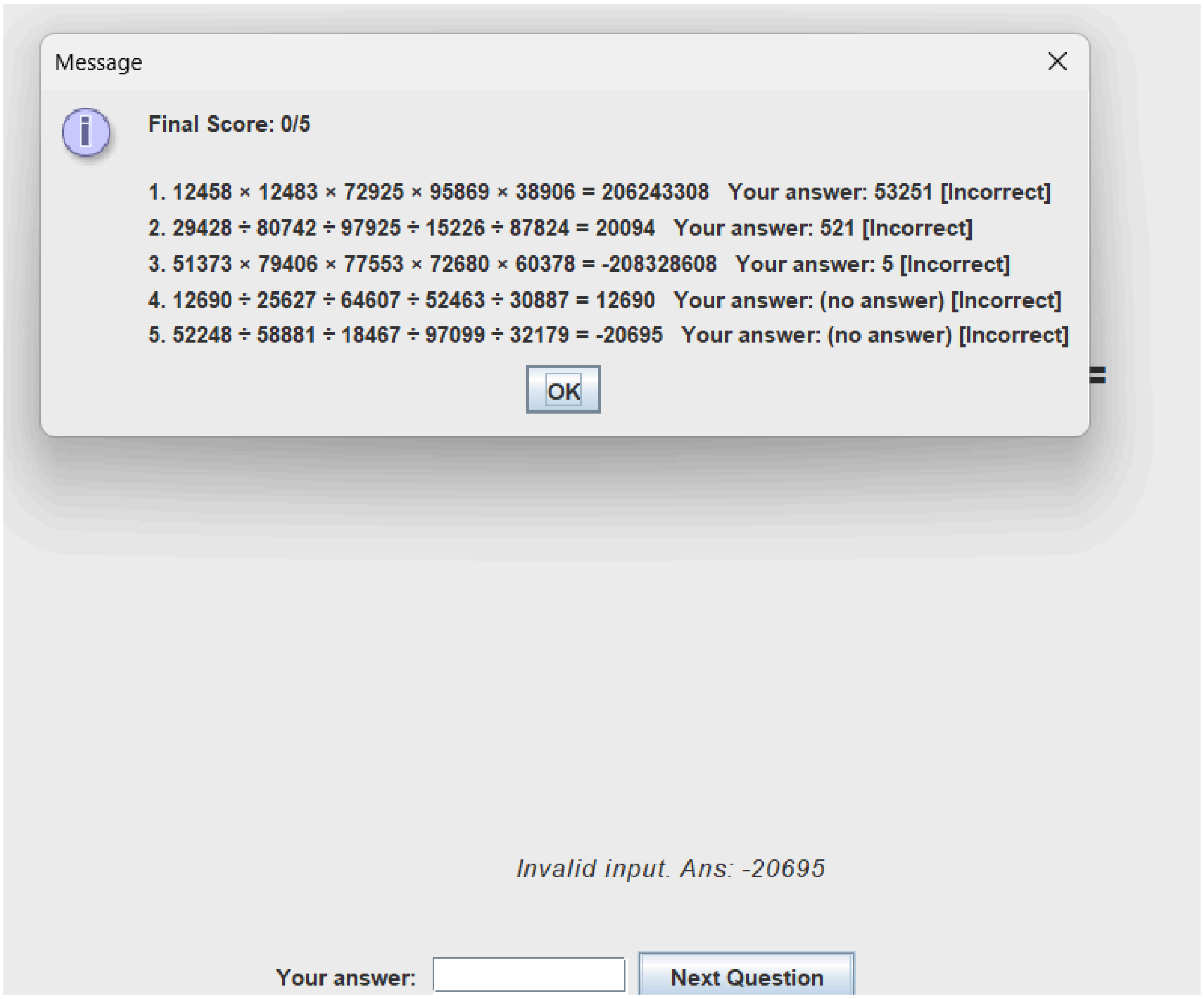


Your answer: 5325

In the bottom side, users can input their answers in the empty box and clicking on "Next Question" will submit their answer for that specific question. If the user did not click "Next

Question” on the final question, it will continue to the next question. However, clicking the Next Question function on the last question, will submit all their work. It will also open a popup window that will show each question, the correct answer to the question, the user’s answer, and showing whether it is correct or not as shown in Demonstration#12.

Demonstration#12:



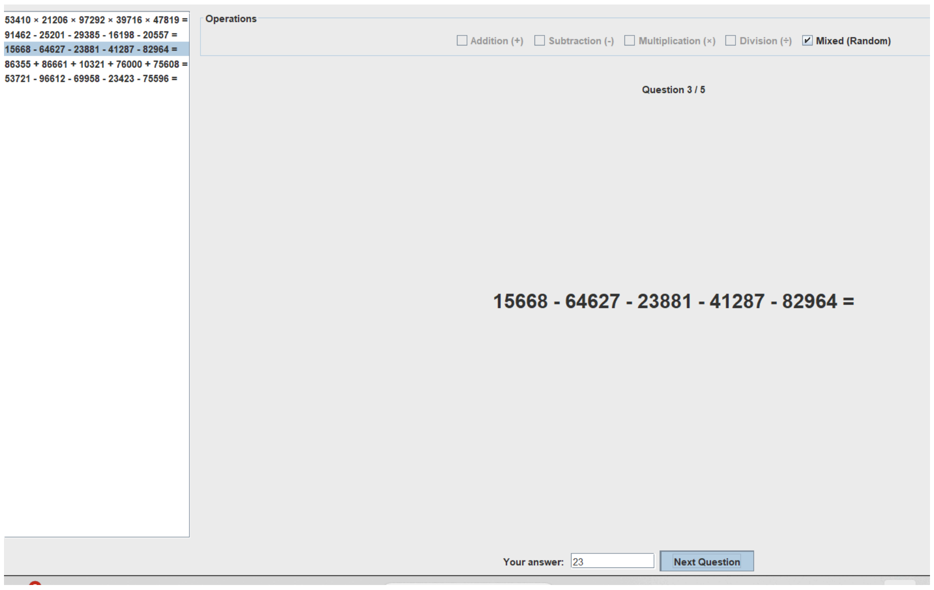
```
Message

Final Score: 0/5

1. 12458 × 12483 × 72925 × 95869 × 38906 = 206243308   Your answer: 53251 [Incorrect]
2. 29428 ÷ 80742 ÷ 97925 ÷ 15226 ÷ 87824 = 20094   Your answer: 521 [Incorrect]
3. 51373 × 79406 × 77553 × 72680 × 60378 = -208328608   Your answer: 5 [Incorrect]
4. 12690 ÷ 25627 ÷ 64607 ÷ 52463 ÷ 30887 = 12690   Your answer: (no answer) [Incorrect]
5. 52248 ÷ 58881 ÷ 18467 ÷ 97099 ÷ 32179 = -20695   Your answer: (no answer) [Incorrect]
```

In addition, when a user inputs an answer to a question and goes Next Question (but not at the final question), the user can click onto the question they would want to check again and to see their answer. Furthermore, they could also erase and input a new answer in the box and then resubmitting their renewed answer by clicking “Next Question” again. This is demonstrated below with a detailed explanation on each picture.

Demonstration#13: Shows user inputting their answer “23” and submitting by clicking Next Question.



```
53410 × 21206 × 97292 × 39716 × 47819 =
91462 - 25201 - 29385 - 16198 - 20557 =
15668 - 64627 - 23881 - 41287 - 82964 =
86355 + 86661 + 10321 + 76000 + 75608 =
53721 - 96612 - 69958 - 23423 - 75596 =
```

Operations

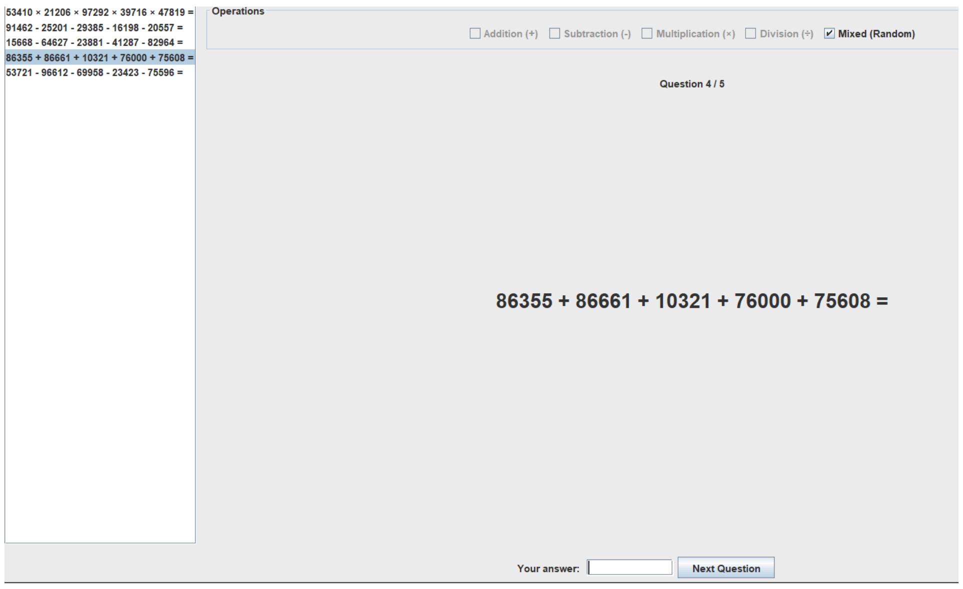
☐ Addition (+) ☐ Subtraction (-) ☐ Multiplication (×) ☐ Division (÷) ☑ Mixed (Random)

Question 3 / 5

15668 - 64627 - 23881 - 41287 - 82964 =

Your answer: 23

Demonstration#14: Shows what happens after clicking Next Question- which is progressing to the next question.



```
53410 × 21206 × 97292 × 39716 × 47819 =
91462 - 25201 - 29385 - 16198 - 20557 =
15668 - 64627 - 23881 - 41287 - 82964 =
86355 + 86661 + 10321 + 76000 + 75608 =
53721 - 96612 - 69958 - 23423 - 75596 =
```

Operations

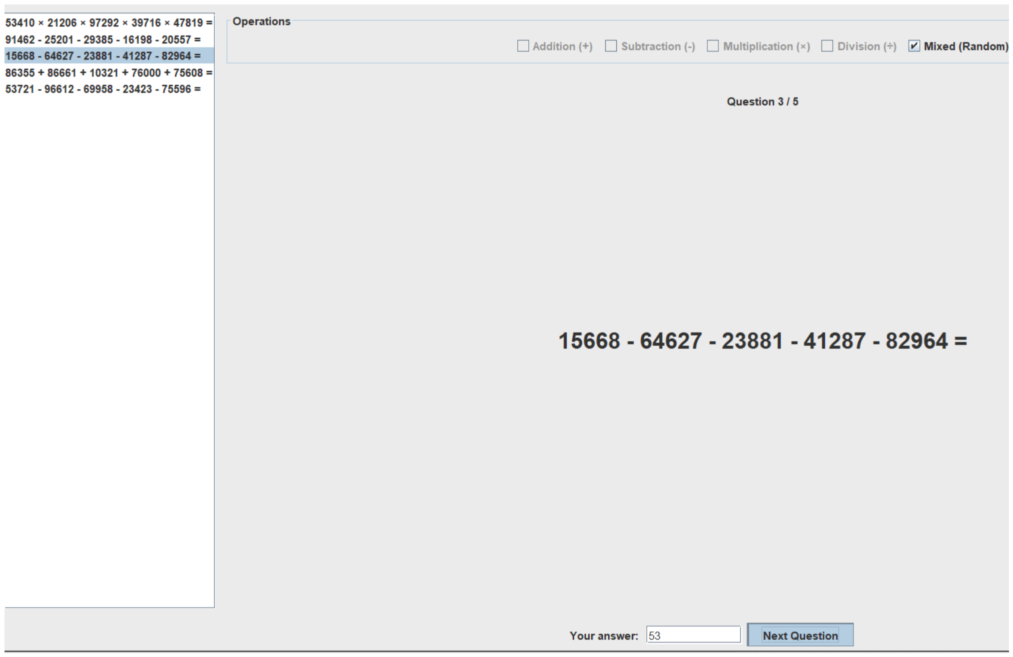
☐ Addition (+) ☐ Subtraction (-) ☐ Multiplication (×) ☐ Division (÷) ☑ Mixed (Random)

Question 4 / 5

86355 + 86661 + 10321 + 76000 + 75608 =

Your answer:

Demonstration#15: Shows user going back to the previous question by clicking once onto that question. Which at first, shows “23”. User then renewed their answer by inputting their new answer and then resubmitting by clicking “Next Question”.



```
53410 × 21206 × 97292 × 39716 × 47819 =
91462 - 25201 - 29385 - 16198 - 20557 =
15668 - 64627 - 23881 - 41287 - 82964 =
86355 + 86661 + 10321 + 76000 + 75608 =
53721 - 96612 - 69958 - 23423 - 75596 =
```

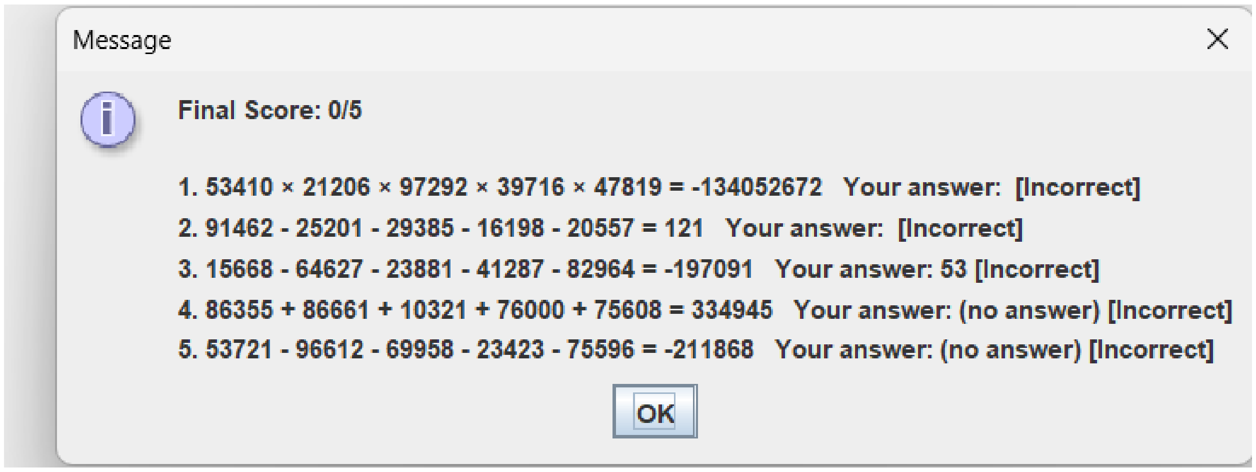
Operations: Addition (+) [ ], Subtraction (-) [ ], Multiplication (×) [ ], Division (÷) [ ], Mixed (Random) [✓]

Question 3 / 5

15668 - 64627 - 23881 - 41287 - 82964 =

Your answer: 53

Demonstration#16: Shows the results and that user resubmission of their answer has been updated.



```
Message

Final Score: 0/5

1. 53410 × 21206 × 97292 × 39716 × 47819 = -134052672   Your answer:  [Incorrect]
2. 91462 - 25201 - 29385 - 16198 - 20557 = 121   Your answer:  [Incorrect]
3. 15668 - 64627 - 23881 - 41287 - 82964 = -197091   Your answer: 53 [Incorrect]
4. 86355 + 86661 + 10321 + 76000 + 75608 = 334945   Your answer: (no answer) [Incorrect]
5. 53721 - 96612 - 69958 - 23423 - 75596 = -211868   Your answer: (no answer) [Incorrect]
```

Program functionality has all been covered for the random math question generator. To summarize, executing the program displays a popup window. There, users can customize the difficulty by inputting their desired amount of digits and operands, and also choosing how many questions to generate by the program. Questions will be displayed with the current question focused displayed in the middle while the other questions will be displayed at the left column box. Users can input their answers by clicking on the empty box at the bottom and then input their answers with clicking on the “Next Question” button to submit their answers. Likewise, users can return to previous or go to other questions by clicking once on the questions at the left column box where they are able to view their answer for that question in the box and even resubmitting by reinputting their answer and then clicking “Next Question” again. Clicking “Next Question” on the last question will display another popup window that shows all user’s answers, questions, the correct answers, and displaying feedback whether user’s answers are correct or not.

Team Workload

Shared team workload:

- **Core Software Development**
- **Question Generation Logic**
- **Testing & Debugging**
- **UI Development**

Kaneko:

- Implements the core functionality of the generator.
- Develops and integrates algorithms for random question generation.
- Reviews the mathematical accuracy and educational value of generated questions.
- Development of basic User Interface

Jeremy:

- Design time-tracking for operations
- Design time-complexity for operations
- Compare time-complexity and space-complexity for different structures
- Create charts and graphs for analysis
- Poster
- Documentation

- Report

References:

Educative. (n.d.). How to generate random numbers in java.

https://www.educative.io/answers/how-to-generate-random-numbers-in-java

Ene, A., & Stirbu, C. (2019). Automatic generation of quizzes for Java programming

language. 2019 11th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Pitesti, Romania.

ResearchGate. https://doi.org/10.1109/ECAI46879.2019.9042052

GeeksforGeeks. (2025). Data Structures Tutorial.

https://www.geeksforgeeks.org/data-structures/

Geeksforgeeks. (2021). Java Swing - JPanel with examples.

https://www.geeksforgeeks.org/java/java-swing-jpanel-with-examples/

Geeksforgeeks. (2024). Time Complexity and Space Complexity.

https://www.geeksforgeeks.org/dsa/time-complexity-and-space-complexity/

Geeksforgeeks. (2023). Java runtime totalmemory() method with examples.

https://www.geeksforgeeks.org/java/java-runtime-totalmemory-method/

Geeksforgeeks. (2025). Generating random numbers in java.

https://www.geeksforgeeks.org/java/generating-random-numbers-in-java/

Geeksforgeeks. (2025). Java user input- scanner class.

https://www.geeksforgeeks.org/java/java-user-input-scanner-class/

Prodigy Education. (2022). Why is math important? *Prodigy Game*.

https://www.prodigygame.com/main-en/blog/why-is-math-important/

Katz, L., Carlgren, D., Wright-Maley, C., Hallam, M., Forder, J., Milner, D., &

Finestone, L. (2024). Student-generated multiple-choice questions: A Java and

web-based tool for students to create multiple choice tests. The Canadian Journal

for the Scholarship of Teaching and Learning, 15(2).

ResearchGate. https://doi.org/10.5206/cjsotlrcacea.2024.2.16625

Think Academy. (2023). Avoiding careless mistakes in math exams: a parent's guide.

https://www.thethinkacademy.com/blog/avoiding-careless-mistakes-in-math-exams-a-parents-guide/#:~:text=Understanding%20Careless%20Errors%20in%20Math,subtraction%2C%20multiplication%2C%20and%20division.

Tutorialspoint. (n.d.). Check memory used by a program in java.

https://www.tutorialspoint.com/how-to-check-the-memory-used-by-a-program-in-java

#:~:text=getRuntime().,memory%20available%20to%20the%20program.

Vogel, L. (2008). java performance - memory and runtime analysis - tutorial.

https://www.vogella.com/tutorials/JavaPerformance/article.html

W3school. (n.d.). Java user input (scanner).

https://www.w3schools.com/java/java_user_input.asp

Appendix:

Github repo (source code, report, PPT, documentation, demonstration):

https://github.com/kanekojosiah/aaaaa.git

GitHub - kanekojosiah/aaaaa