Last Referenced Update

Commit: bd3ecd7

Contents

About The Project

MathTester is a simple Application using Swing and AWT from Java's JDK 21 API. It uses random number generation to give mathematical equations that are supposed to be solved by the user, displaying whether the given answer was correct. It also has interchangable diffiulty modes, making the numbers used greater or smaller according to the selected mode. Please note that all generated equations will result in an answer that is a whole number and floating point inputs will result in an error that can be ignored. The user will simply have to give a whole number as an input and giving a String on accident won't result in a program crash or Bug.

Pressing the Quit Button and closing the Frame will result in the program being exited either way.

Why It Is One Class Only

This was mainly a decision based on personal preferences. For simplicity I decided to wrap the project up in one class instead of making an unreasonable effort to combine single components into one class, such as the Menu Bar, used to switch between modes. It also lowers the risk of bugs due to type inconsistencies, as well as confusion about what functions to call. It also isn't a project that requires a large amount of development, hence the lack of reusability.

Build With

Java JDK 21:

- AWT
- Swing
- Util

Usage

- Training basic mathematical equations

License

OpenJDK 21

Documentation

Initializing all values

```
private final static List<String> operators = Arrays.asList("+", "-", "*",
"/");
private static Random rand = new Random();
private static String currentTask;
private static int currentResult;
private static final int amountPanels = 4;
private static List<Color> colors = Arrays.asList(Color.WHITE,
Color.GREEN, Color.YELLOW, Color.RED);
private static int currentMode = 1;
private static List<String> comp;
```

Full Breakdown:

- `private final static List<String> operators = Arrays.asList("+", "-", "*", "/");`: Creates a list with all fundamental operations in Math (Addition, Substraction, Multiplication, Division)
- `private static Random rand = new Random();`: Creates an Object from Java's Random class, to be able to create random numbers later on in the project.
- `private static String currentTask;`: Initializes the String variable currentTask, to be able to store the current task that will be displayed on the GUI.
- `private static int currentResult;`: Initializes the Integer variable currentResult, to be able to store the current result and access it, in order to compare it to the user's input.
- `private static final int amountPanels = 4;`: Sets the amount of Subpanels the main Panel will have to 4.
- `private static List<Color> colors = Arrays.asList(Color.WHITE, Color.GREEN, Color.YELLOW, Color.RED);`: Creates a List of color objects of Java's class Color, to be able to change the color of some UI components, by accessing this list.
- `private static int currentMode = 1;`: Sets the selected mode, when the project is opened, to 1, which corellates to the easy mode.
- `private static List<String> comp;`: Initializes the comp variable, used to gain and store the results of the function, which is used to compare the user's input to the tasks solution.

GUI

```
public static void GUI() {

        // Create and set up the window.
        JFrame frame = new JFrame("MathTester");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(600, 350);
        frame.setResizable(false);

        // Create Panel for Components
        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new GridLayout(amountPanels, 1));
        frame.add(mainPanel);
```

```
        // Create SubPanels for different use cases
        JPanel[][] subPanels = new JPanel[amountPanels][1];
        JPanel menuPanel = subPanels[0][0] = new JPanel();
        JPanel outputPanel = subPanels[1][0] = new JPanel();
        JPanel buttonPanel = subPanels[2][0] = new JPanel();
        JPanel textPanel = subPanels[3][0] = new JPanel();

        buttonPanel.setLayout(new GridLayout(1, 2));
        textPanel.setLayout(new GridLayout(1, 1));
        outputPanel.setLayout(new GridLayout(1, 2));
        mainPanel.add(menuPanel);
        mainPanel.add(outputPanel);
        mainPanel.add(buttonPanel);
        mainPanel.add(textPanel);

        // Create Menu Bar
        Border menuBorder = new LineBorder(Color.black);
        JMenuBar modeMenu = new JMenuBar();
        modeMenu.setBorder(menuBorder);
        menuPanel.add(modeMenu);

        // Create Items and separators for Menu
        JMenuItem easyMode = new JMenuItem("Easy");
        JMenuItem mediumMode = new JMenuItem("Medium");
        JMenuItem hardMode = new JMenuItem("Hard"); // add Action
Listeners + funcitons
        easyMode.setBackground(colors.get(1));
        menuPanel.setBackground(colors.get(1));
        modeMenu.add(easyMode);
        modeMenu.add(mediumMode);
        modeMenu.add(hardMode);

        // Create Buttons
        JButton[][] buttons = new JButton[1][2];
        JButton submitButton = buttons[0][0] = new JButton("Submit");
        JButton quitButton = buttons[0][1] = new JButton("Quit");
        buttonPanel.add(submitButton);
        buttonPanel.add(quitButton);

        // Create Entry Textfield
        JTextField[][] textFields = new JTextField[2][1];
        JTextField entry = textFields[0][0] = new JTextField("");
        textPanel.add(entry);

        // Create Output Label
        JLabel[][] labels = new JLabel[1][2];
        JLabel calcLabel = labels[0][0] = new JLabel(currentTask);
        JLabel outputLabel = labels[0][1] = new JLabel("Solve the
equation.");
        outputPanel.add(calcLabel);
        outputPanel.add(outputLabel);

        // Add Action Listeners to Buttons
        quitButton.addActionListener(new ActionListener() {
```

```
            public void actionPerformed(final ActionEvent e) {
                System.exit(0);
            }
        });

        submitButton.addActionListener(new ActionListener() {
            public void actionPerformed(final ActionEvent e) {
                comp = compareResult(entry.getText());
                calcLabel.setText(comp.get(0));
                outputLabel.setText(comp.get(1));
                entry.setText("");
            }
        });

        // Add Action Listeners to MenuBar
        easyMode.addActionListener(new ActionListener() {
            public void actionPerformed(final ActionEvent e) {
                menuPanel.setBackground(colors.get(1));
                easyMode.setBackground(colors.get(1));
                mediumMode.setBackground(colors.get(0));
                hardMode.setBackground(colors.get(0));
                modeSelect("easy");
                calcLabel.setText(currentTask);
                outputLabel.setText("Switched to Easy-Mode");
            }
        });

        mediumMode.addActionListener(new ActionListener() {
            public void actionPerformed(final ActionEvent e) {
                menuPanel.setBackground(colors.get(2));
                easyMode.setBackground(colors.get(0));
                mediumMode.setBackground(colors.get(2));
                hardMode.setBackground(colors.get(0));
                modeSelect("medium");
                calcLabel.setText(currentTask);
                outputLabel.setText("Switched to Medium-Mode");
            }
        });

        hardMode.addActionListener(new ActionListener() {
            public void actionPerformed(final ActionEvent e) {
                menuPanel.setBackground(colors.get(3));
                easyMode.setBackground(colors.get(0));
                mediumMode.setBackground(colors.get(0));
                hardMode.setBackground(colors.get(3));
                modeSelect("hard");
                calcLabel.setText(currentTask);
                outputLabel.setText("Switched to Hard-Mode");
            }
        });

        // Display the window.
        frame.setVisible(true);
    }
```

Full Breakdown:

```
JFrame frame = new JFrame("MathTester");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(600, 350);
frame.setResizable(false);
```

1. Creates the Frame, used to display all components
2. Tells Java to exit the program, in case the Frame is closed
3. Sets the size of the Frame to the dimensions 600x350
4. Prevents the user from being able to resize the Frame, to keep all components visible

```
JPanel mainPanel = new JPanel();
mainPanel.setLayout(new GridLayout(amountPanels, 1));
frame.add(mainPanel);
```

1. Creates a Panel
2. Sets the Panels Layout, so each component will fill up it's assigned space
3. Adds the Panel to the Frame, so other components can be displayed on it

```
JPanel[][] subPanels = new JPanel[amountPanels][1];
JPanel menuPanel = subPanels[0][0] = new JPanel();
JPanel outputPanel = subPanels[1][0] = new JPanel();
JPanel buttonPanel = subPanels[2][0] = new JPanel();
JPanel textPanel = subPanels[3][0] = new JPanel();
```

1. Creates a list of Panels in a 2D-List, so each Subpanel can be placed in it's space in the Layout respectively
2. Creates the menuPanel for the Menu Bar
3. Creates the outputPanel for Labels
4. Creates the buttonPanel, for Buttons
5. Creates the textPanel, for the Entry Field

```
buttonPanel.setLayout(new GridLayout(1, 2));
textPanel.setLayout(new GridLayout(1, 1));
outputPanel.setLayout(new GridLayout(1, 2));
mainPanel.add(menuPanel);
mainPanel.add(outputPanel);
mainPanel.add(buttonPanel);
mainPanel.add(textPanel);
```

1. Sets the buttonPanels Layout, so each component will fill up it's assigned space
2. Sets the textPanels Layout, so each component will fill up it's assigned space
3. Sets the outputPanels Layout, so each component will fill up it's assigned space
4. Adds the menuPanel to the mainPanel, so it can be displayed
5. Adds the outputPanel to the mainPanel, so it can be displayed
6. Adds the buttonPanel to the mainPanel, so it can be displayed
7. Adds the textPanel to the mainPanel, so it can be displayed

```
Border menuBorder = new LineBorder(Color.black);
JMenuBar modeMenu = new JMenuBar();
modeMenu.setBorder(menuBorder);
menuPanel.add(modeMenu);
```

1. Creates a Border for the Menu Bar, which is black
2. Creates the MenuBar
3. Links the Border to the Menu Bar
4. Adds the Menu Bar to it's Panel, so it can be displayed

```
JMenuItem easyMode = new JMenuItem("Easy");
JMenuItem mediumMode = new JMenuItem("Medium");
JMenuItem hardMode = new JMenuItem("Hard");
easyMode.setBackground(colors.get(1));
menuPanel.setBackground(colors.get(1));
modeMenu.add(easyMode);
modeMenu.add(mediumMode);
modeMenu.add(hardMode);
```

1. Creates an Item for the Menu Bar, which displays the String "Easy"
2. Creates an Item for the Menu Bar, which displays the String "Medium"
3. Creates an Item for the Menu Bar, which displays the String "Hard"
4. Sets the Item's color which displays "Easy" to green
5. Sets the Panels color, in which the Menu Bar is located, to green
6. Adds the Item that displays "Easy" to the Menu Bar
7. Adds the Item that displays "Medium" to the Menu Bar
8. Adds the Item that displays "Hard" to the Menu Bar

```
JButton[][] buttons = new JButton[1][2];
JButton submitButton = buttons[0][0] = new JButton("Submit");
JButton quitButton = buttons[0][1] = new JButton("Quit");
buttonPanel.add(submitButton);
buttonPanel.add(quitButton);
```

1. Creates a list of Buttons in a 2D-List, so each Button can be placed in it's space in the Layout respectively

2. Creates a Button, which displays "Submit"
3. Creates a button, which displays "Quit"
4. Adds the Button displaying "Submit" to it's Panel
5. Adds the Button displaying "Quit" to it's Panel

```
JTextField[][] textFields = new JTextField[2][1];
JTextField entry = textFields[0][0] = new JTextField("");
textPanel.add(entry);
```

1. Creates a list of Text Fields in a 2D-List, so each Text Field can be placed in it's space in the Layout respectively
2. Creates a Text Field
3. Adds the Text Field to it's Panel

```
JLabel[][] labels = new JLabel[1][2];
JLabel calcLabel = labels[0][0] = new JLabel(currentTask);
JLabel outputLabel = labels[0][1] = new JLabel("Solve the equation.");
outputPanel.add(calcLabel);
outputPanel.add(outputLabel);
```

1. Creates a list of Labels in a 2D-List, so each Label can be placed in it's space in the Layout respectively
2. Creates a Label, displaying the current Task
3. Creates a Label stating the objective of the user
4. Adds the Label, displaying the current Task, to it's Panel
5. Adds the Label, stating the objective of the user, to it's Panel

Adding Functionality

```
quitButton.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
        System.exit(0);
    }
});
```

- Adds an Action Listener to the Quit-Button
- If the Button is clicked, the program will exit

```
submitButton.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
        comp = compareResult(entry.getText());
        calcLabel.setText(comp.get(0));
        outputLabel.setText(comp.get(1));
        entry.setText("");
```

```
    }
});
```

- Adds an Action Listener to the Submit-Button
- If the Button is clicked, the program will:
    1. Call a function to compare the user input with the current solution and store a set of outputs to display
    2. Display the outputs it retrieved
    3. Set the Text Field's content to an empty String

```
easyMode.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
        menuPanel.setBackground(colors.get(1));
        easyMode.setBackground(colors.get(1));
        mediumMode.setBackground(colors.get(0));
        hardMode.setBackground(colors.get(0));
        modeSelect("easy");
        calcLabel.setText(currentTask);
        outputLabel.setText("Switched to Easy-Mode");
    }
});

mediumMode.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
        menuPanel.setBackground(colors.get(2));
        easyMode.setBackground(colors.get(0));
        mediumMode.setBackground(colors.get(2));
        hardMode.setBackground(colors.get(0));
        modeSelect("medium");
        calcLabel.setText(currentTask);
        outputLabel.setText("Switched to Medium-Mode");
    }
});

hardMode.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
        menuPanel.setBackground(colors.get(3));
        easyMode.setBackground(colors.get(0));
        mediumMode.setBackground(colors.get(0));
        hardMode.setBackground(colors.get(3));
        modeSelect("hard");
        calcLabel.setText(currentTask);
        outputLabel.setText("Switched to Hard-Mode");
    }
});
```

- Adds Action Listeners to each item in the Menu Bar
- If an Item in the Menu Bar is clicked, the program will:
    1. Set the Panel in which the Menu Bar is located to the Items assigned color

2. Set the clicked Item's color, to it's assigned color
3. Set the color of the Items that weren't clicked to white
4. Select the Mode associated with the Item that has been clicked
5. Calls the modeSelect method and passes it'S String
6. Display the new current Task
7. Display the Mode it has switched to

```
frame.setVisible(true);
```

- Displays the program

Adding Modes

```
private static void modeSelect(String mode) {
    switch (mode) {
        case "easy":
            currentMode = 1;
            break;

        case "medium":
            currentMode = 2;
            break;

        case "hard":
            currentMode = 3;
            break;

        default:
            System.exit(50);
            break;
    }

    taskHandler();
}
```

- The Method modeSelect takes in a String when calles
- modeSelect then sets the new Mode according to the input
- If something unexpected calls this method, it will exit the program with the exit code 50 (which is not the appropriate exit code for this case)
- After the mode is set, a new task is created with the newly applied mode

Creating equations

```
private static void taskHandler() {
    int firstNumber = rand.nextInt(100 * currentMode);
    int secondNumber = rand.nextInt(100 * currentMode);
```

```
    String operator;

    if (firstNumber % secondNumber == 0 && secondNumber != 0) {
        operator = operators.get(rand.nextInt(4));
    } else {
        operator = operators.get(rand.nextInt(3));
    }

    if ((operator == "*" || operator == "/") && (firstNumber > 15 ||
secondNumber > 15)) {
        String newFirstNumber = String.valueOf(firstNumber);
        String newSecondNumber = String.valueOf(secondNumber);
        if (newFirstNumber.length() > currentMode ||
newSecondNumber.length() > currentMode) {
            firstNumber = Integer.parseInt(newFirstNumber.substring(0,
currentMode));
            secondNumber = Integer.parseInt(newSecondNumber.substring(0,
currentMode));
        }
    }

    currentTask = String.valueOf(firstNumber).concat(" ")
            .concat(operator)
            .concat(" ")
            .concat(String.valueOf(secondNumber))
            .concat(" = ");

    switch (operator) {
        case "+":
            currentResult = firstNumber + secondNumber;
            break;

        case "-":
            currentResult = firstNumber - secondNumber;
            break;

        case "*":
            currentResult = firstNumber * secondNumber;
            break;

        case "/":
            currentResult = firstNumber / secondNumber;
            break;

        default:
            currentResult = 0;
            System.exit(50);
            break;
    }
}
```

- The taskHandler's purpose is to create new tasks

- When called, it will generate a random number for both variables in the task
- If dividing the number won't lead to a whole number or if the divisor is 0, the taskHandler will refrain from using divison as the operation for the task
- The taskHandler also takes into account that numbers greater than 15, will make division and multiplication and therefore:
    1. Checks whether the number is longer than the current Mode which limit the lenght of a number as a String from 1-3 (depending on the current mode)
    2. If the number is indeed longer than the current Mode allows, it will cut it to the appropriate size

Check Userinput

```
private static List<String> compareResult(String number) {
      int userResult;
      try {
          userResult = Integer.valueOf(number);
      } catch (Exception ValueError) {
          List<String> misinputError = Arrays.asList(currentTask, "Error:
Input could not be parsed as an Integer.");
          return misinputError;
      }

      if (userResult == currentResult) {
          String lastResult = String.format("Your Last answer %d was
correct.",
                  Integer.parseInt(String.valueOf(currentResult)));
          taskHandler();
          List<String> checkAssist = Arrays.asList(currentTask,
lastResult);
          return checkAssist;
      } else {
          List<String> checkAssist = Arrays.asList(currentTask, "Wrong
Answer. Try Again.");
          return checkAssist;
      }
```

- The method compareResult also takes in a String, which is provided by what the user placed in the Text Field before submitting
- This method will try to convert the users input into an Integer and, if failed, return an error message to the Labels
- If the Integer can be parsed it will be evaluated, whether it corresponds to the current Result of the task
- If the user's input correlates to the current result, an output will be prepared, for the user to see that his last input was correct and a new task will be created, so it can be displayed on the output Labels
- If the answer is incorrect, the user will be told to try again, by returning said information as a String to the output Labels

Main Function

```
public static void main(String[] args) {
    taskHandler();

    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            GUI();
        }
    });
}
```

- The main method, used to initialize the program, will:
    1. Create a new task by calling the taskHandler
    2. Schedule a threat for the application to run
    3. Call the GUI, to display the application