Coursework

Numberle Game

[Advanced Object-Oriented Programming]

by

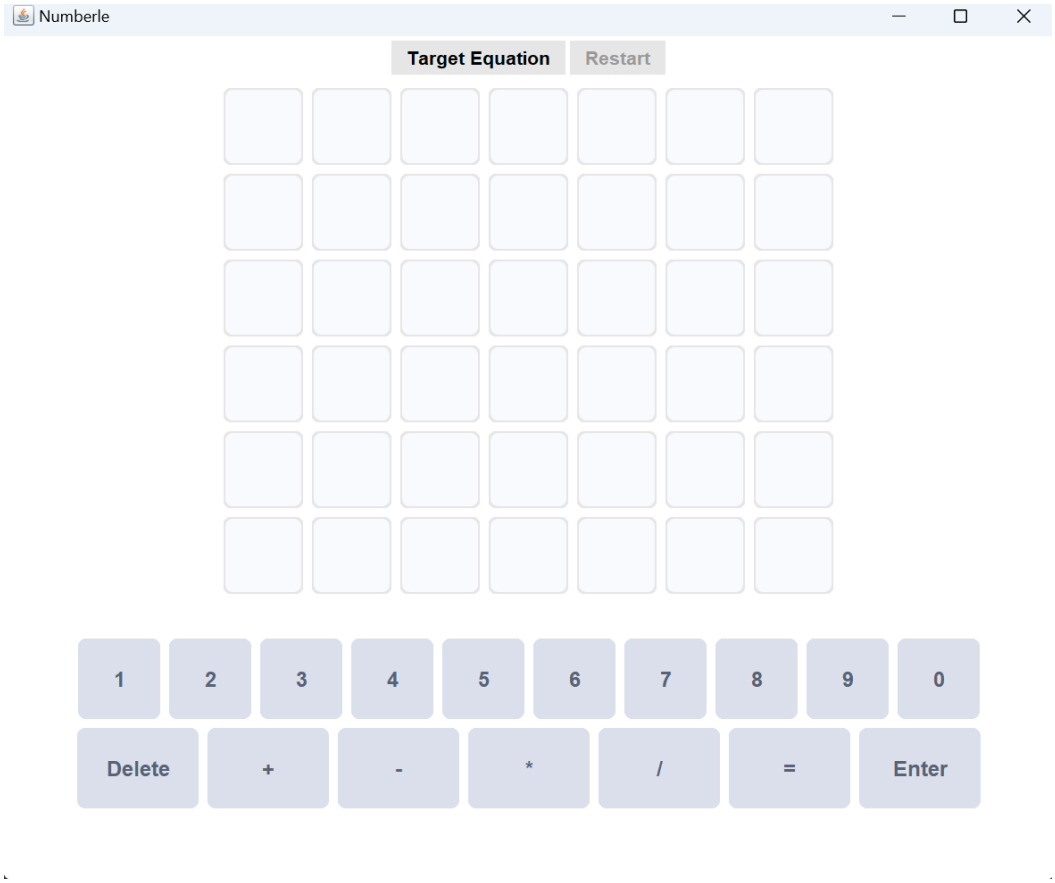
Name: Ryan

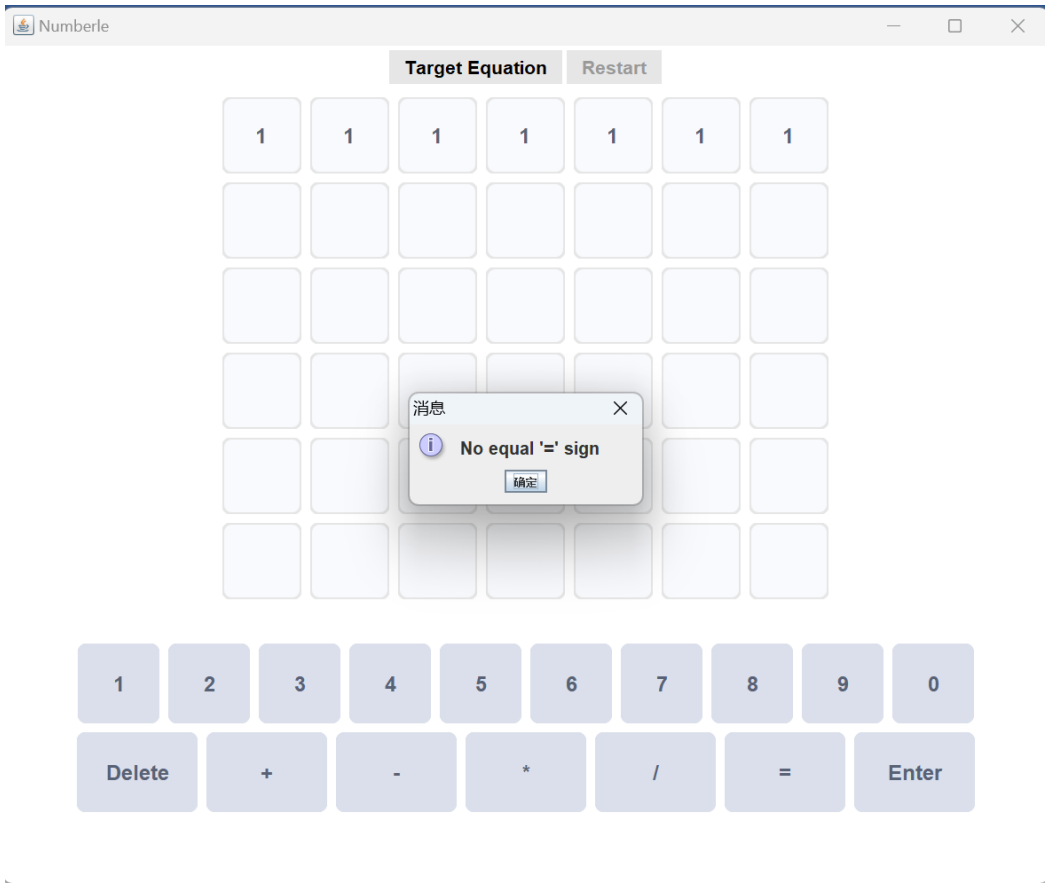
Student ID: 202018010315

Group: L6C6

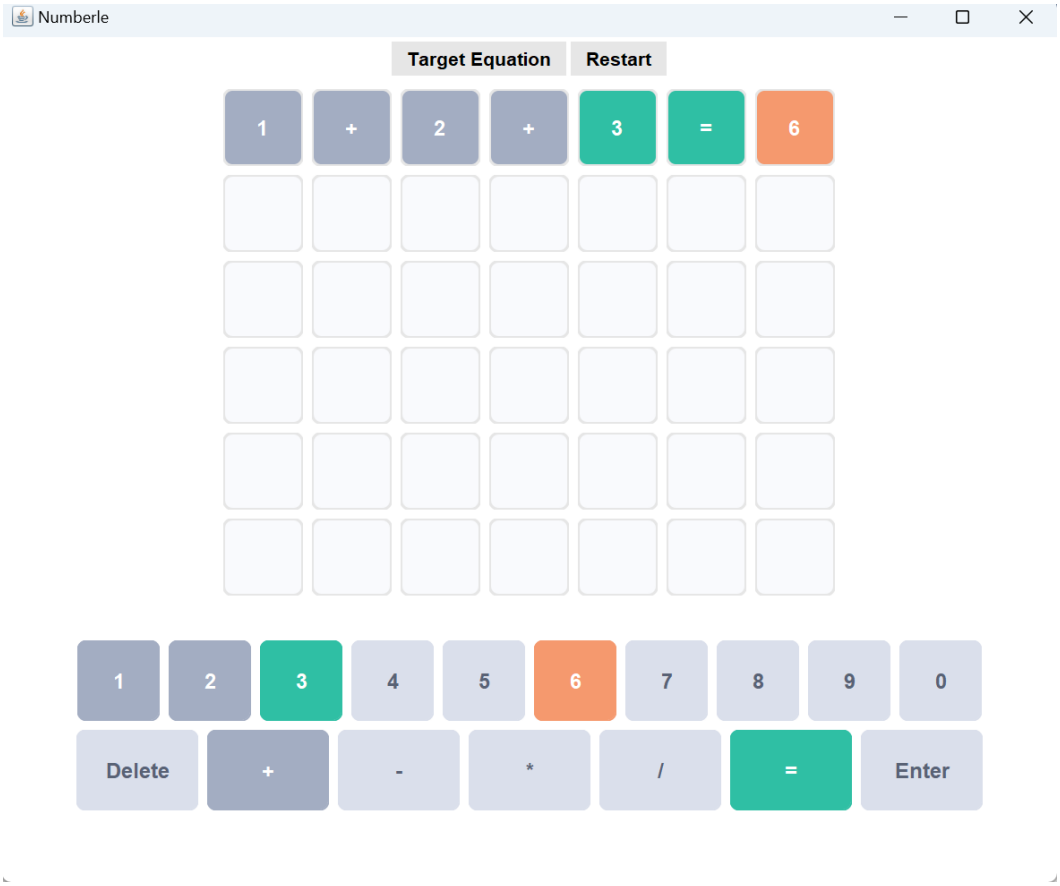
2024/5/13

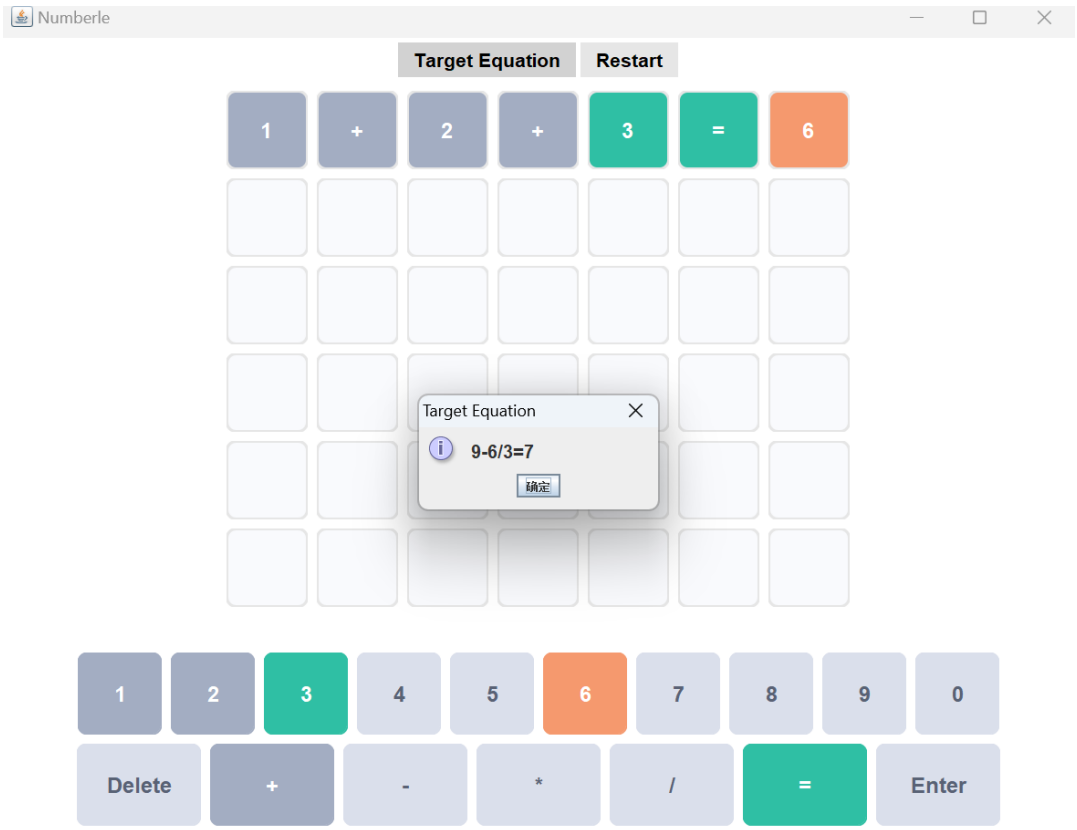
GUI:

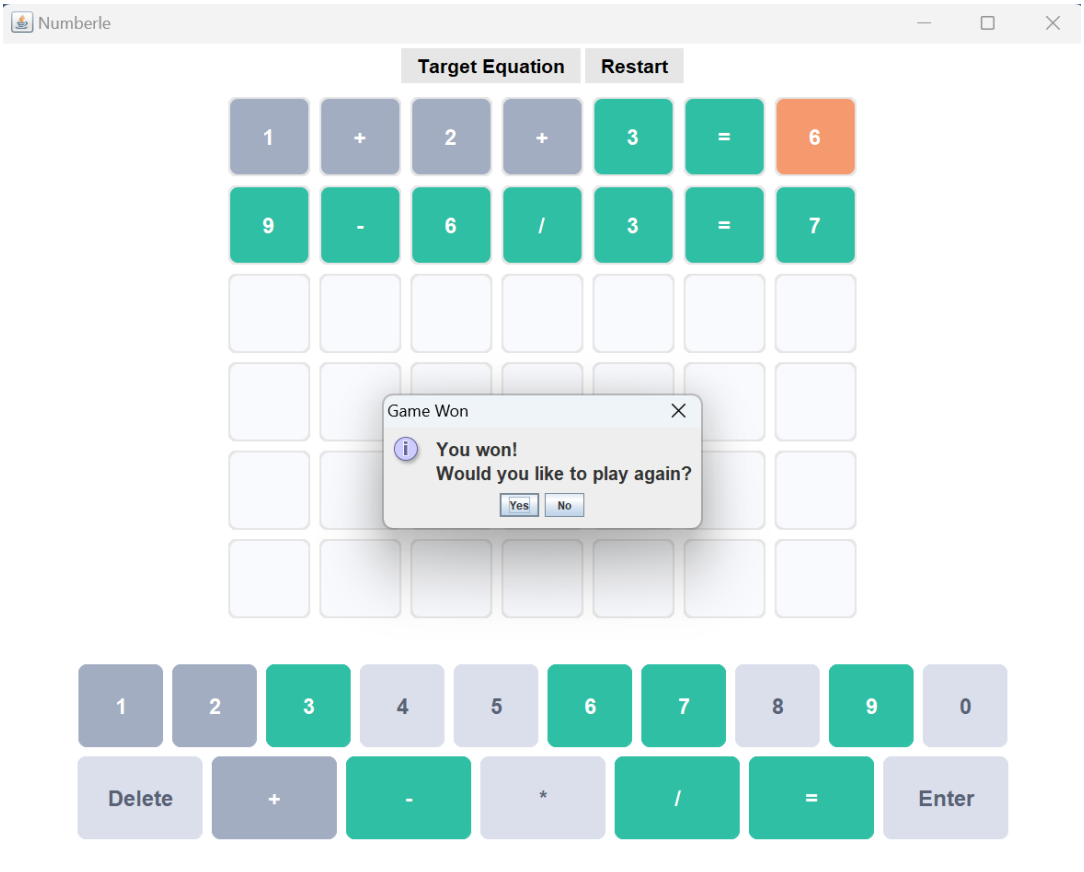

Numberle
Target Equation
Restart
1
2
3
4
5
6
7
8
9
0
Delete
+
-
*
/
=
Enter

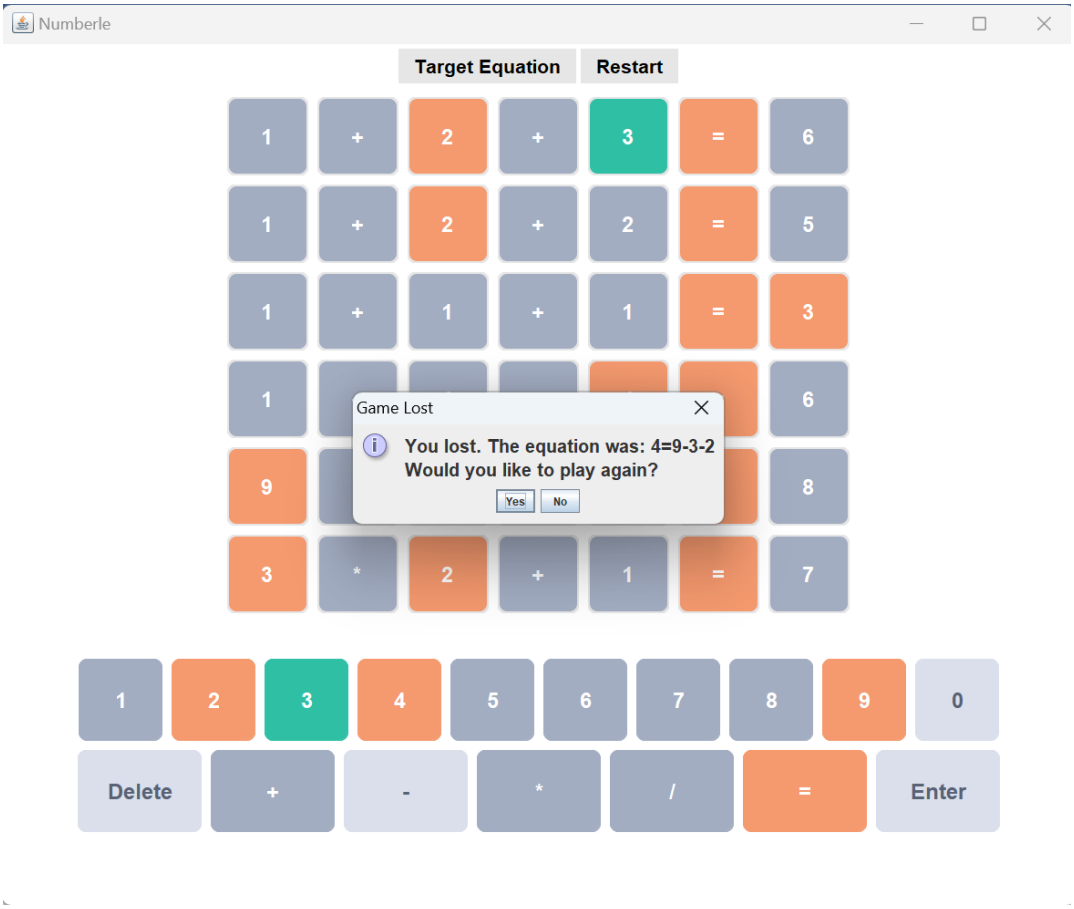

Numberle
Target Equation
Restart
1
1
1
1
1
1
1
消息
No equal '=' sign
确定
1
2
3
4
5
6
7
8
9
0
Delete
+
-
*
/
=
Enter


Numberle
Target Equation
Restart
1 1 1 1 1
消息
The length of the equation must be 7
确定
1 2 3 4 5 6 7 8 9 0
Delete + - * / = Enter


Numberle
Target Equation
Restart
1 + 2 + 3 = 6
1 2 3 4 5 6 7 8 9 0
Delete + - * / = Enter


Numberle
Target Equation
Restart
1 + 2 + 3 = 6
Target Equation
9-6/3=7
确定
1 2 3 4 5 6 7 8 9 0
Delete + - * / = Enter


Numberle
Target Equation
Restart
1 + 2 + 3 = 6
9 - 6 / 3 = 7
Game Won
You won!
Would you like to play again?
Yes
No
1 2 3 4 5 6 7 8 9 0
Delete + - * / = Enter


Numberle
Target Equation
Restart
1 + 2 + 3 = 6
1 + 2 + 2 = 5
1 + 1 + 1 = 3
1 6
9 8
3 * 2 + 1 = 7
Game Lost
You lost. The equation was: 4=9-3-2
Would you like to play again?
Yes
No
1 2 3 4 5 6 7 8 9 0
Delete + - * / = Enter

CLI:

```
"C:\Program Files\Java\jdk1.8.0_131\bin\java.exe" ...
Welcome to Numberle Game!
You have six chances in total.
The equation you enter must satisfy the following conditions:
1. When calculating, players can use numbers (0-9) and arithmetic signs (+ - * / =).
2. The length of the equation must be 7.
3. Must have equal sign '='.
4. There must be at least one sign +-*/ and multiple math symbol in a row is not allowed
5. Letters are not allowed.
---------------------------------------
Target Equation: 1+2*3=7
Remaining attempts: 6
Please enter your guess: 5+6=11
The length of the equation must be 7
Enter your guess again: 45+2+65
No equal '=' sign
Enter your guess again: 2+2+2=6
Hints:
2+2+2=6
Green indicates that numbers or math symbols are exist and are in the correct position
Orange indicates that numbers or math symbols are exist but not in the correct position
Gray represents numbers or math symbols not exist in the target equation
You have not used these numbers and symbols yet: [*, -, /, 0, 1, 3, 4, 5, 7, 8, 9]
---------------------------------------
Remaining attempts: 5
Please enter your guess: 5+6=9+8
The left side is not equal to the right side
Enter your guess again: 1+2*3=7
1+2*3=7
---------------------------------------
You won!
Do you want to play again? (Y/N)
y
```

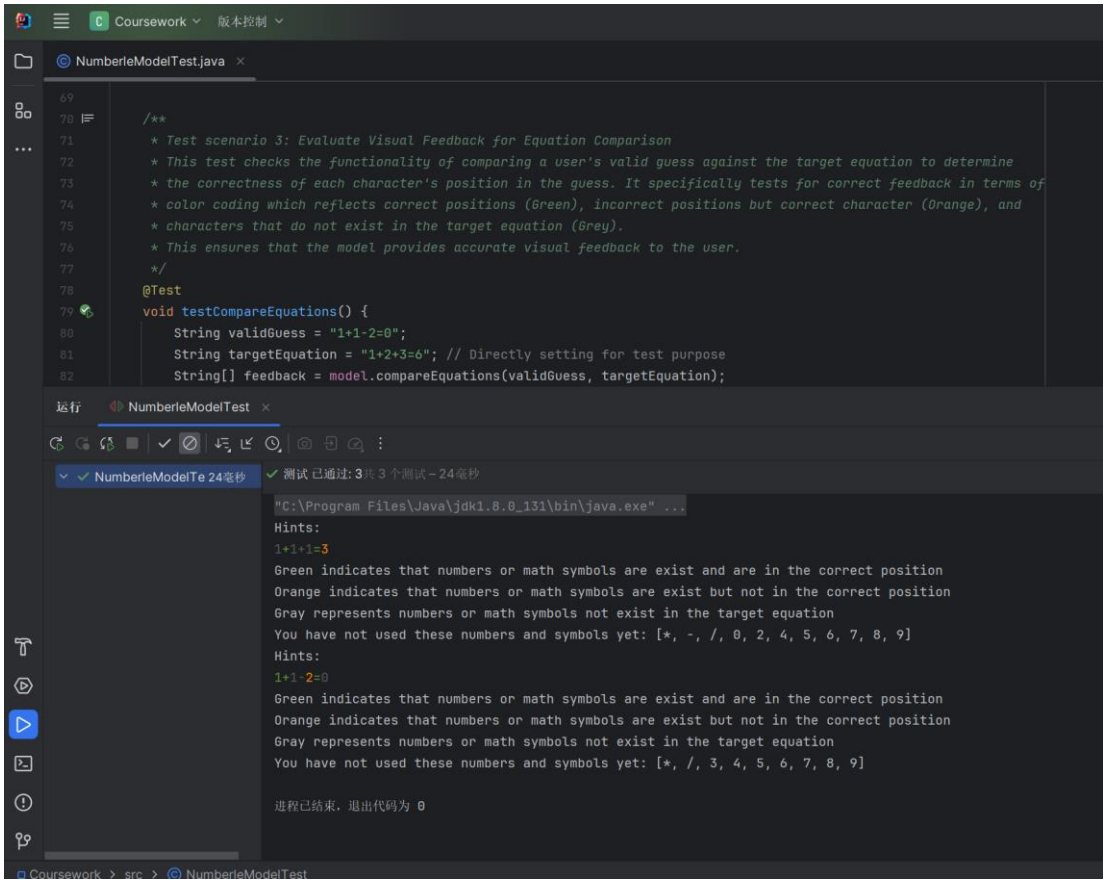
Target Equation: 1+2*3=7

Guess: 2+2+2=6

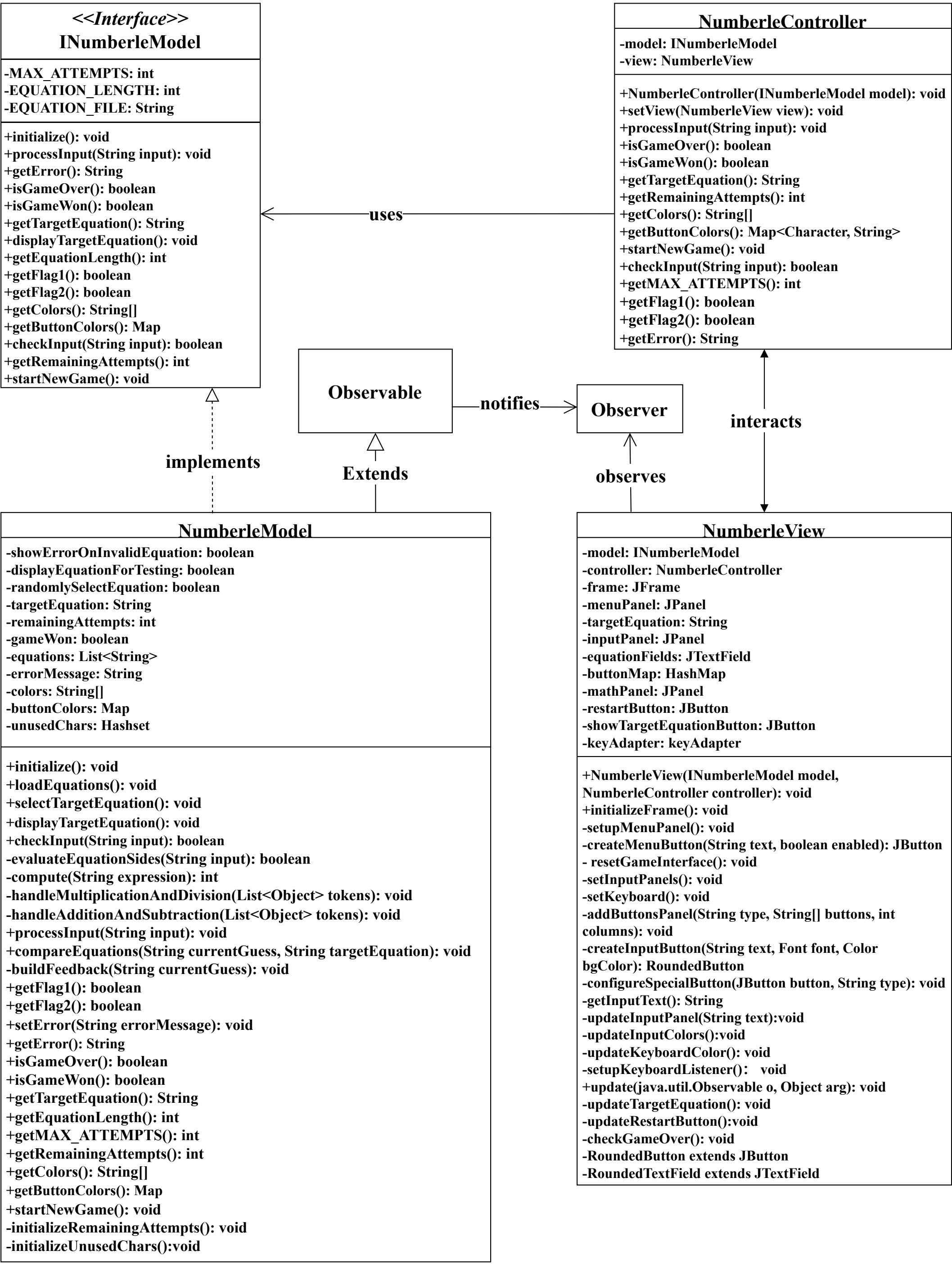
Only second number of 2 (correct location) will be green.

```
---------------------------------------
Remaining attempts: 1
Please enter your guess: 1+1+1=3
Hints:
1+1+1=3
Green indicates that numbers or math symbols are exist and are in the correct position
Orange indicates that numbers or math symbols are exist but not in the correct position
Gray represents numbers or math symbols not exist in the target equation
You have not used these numbers and symbols yet: [*, -, /, 0, 4, 5, 6, 7, 8, 9]
---------------------------------------
You lost. The answer was: 4=3-1+2
Do you want to play again? (Y/N)
```

JUnit:


Coursework
版本控制
NumberleModelTest.java
/**
* Test scenario 3: Evaluate Visual Feedback for Equation Comparison
* This test checks the functionality of comparing a user's valid guess against the target equation to determine
* the correctness of each character's position in the guess. It specifically tests for correct feedback in terms of
* color coding which reflects correct positions (Green), incorrect positions but correct character (Orange), and
* characters that do not exist in the target equation (Grey).
* This ensures that the model provides accurate visual feedback to the user.
*/
@Test
void testCompareEquations() {
String validGuess = "1+1-2=0";
String targetEquation = "1+2+3=6"; // Directly setting for test purpose
String[] feedback = model.compareEquations(validGuess, targetEquation);
运行
NumberleModelTest
NumberleModelTe 24毫秒
测试 已通过: 3共 3 个测试 – 24毫秒
"C:\Program Files\Java\jdk1.8.0_131\bin\java.exe" ...
Hints:
1+1+1=3
Green indicates that numbers or math symbols are exist and are in the correct position
Orange indicates that numbers or math symbols are exist but not in the correct position
Gray represents numbers or math symbols not exist in the target equation
You have not used these numbers and symbols yet: [*, -, /, 0, 2, 4, 5, 6, 7, 8, 9]
Hints:
1+1-2=0
Green indicates that numbers or math symbols are exist and are in the correct position
Orange indicates that numbers or math symbols are exist but not in the correct position
Gray represents numbers or math symbols not exist in the target equation
You have not used these numbers and symbols yet: [*, /, 3, 4, 5, 6, 7, 8, 9]
进程已结束，退出代码为 0
Coursework > src > NumberleModelTest

https://github.com/FOMOKN/CHC6186-Advanced-Object-Oriented-Programming



```java
// NumberleModel.java
// Imports necessary Java utility and IO classes
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;

// The main model class for the Numberle game implementing the observer pattern
public class NumberleModel extends Observable implements INumberleModel {
    private final boolean showErrorOnInvalidEquation = true; // Flag to display
an error if input is invalid
    private final boolean displayEquationForTesting = true; // Flag to display
the equation for testing
    private final boolean randomlySelectEquation = true; // Flag to choose
equations randomly or use a fixed one
    private String targetEquation; // Stores the current target equation
    private int remainingAttempts; // Counter for remaining guesses
    private boolean gameWon; // Indicates if the game has been won
    private List<String> equations = new ArrayList<>(); // List to store all
valid equations
    private String errorMessage = ""; // Store the different errorMessage when
user input the invalid equation
    private String[] colors; // Different colors for numbers and math symbols
    private Map<Character, String> buttonColors; // Color coding for the virtual
keyboard buttons
    private Set<Character> unusedChars = new HashSet<>(); // Set of numbers and
symbols

    /**
     * Initializes the game by loading equations, selecting a target equation,
and resetting attempts.
     * @ invariant equations.size() > 0 : "There must be at least one equation
available at all times."
     * @ ensures gameWon == false && remainingAttempts == MAX_ATTEMPTS
     *   && unusedChars.containsAll(Arrays.asList('0', '1', '2', '3', '4', '5', '
6', '7', '8', '9',
     *   '+', '-', '*', '/', '='))
     *   && targetEquation.length() == EQUATION_LENGTH
     */
    @Override
    public void initialize() {
        loadEquations();
        selectTargetEquation();
        initializeRemainingAttempts(); //Initialize the number of remaining
attempts
        initializeUnusedChars(); // Initialize the set of unused characters
        gameWon = false;
        setChanged(); // Set the flag indicating that the model has changed
        notifyObservers(); // Notify observers that the model has changed
    }

    /**
     * Loads equations from a specified file.
     * @ invariant !equations.isEmpty() : "Equations list should not be empty
after loading."
     * @ ensures \forall String eq; eq \in equations; eq.length() ==
EQUATION_LENGTH
     */
    public void loadEquations() {
```

```
        try (BufferedReader reader = new BufferedReader(new FileReader(
EQUATION_FILE))) {
            String line;
            while ((line = reader.readLine()) != null) {
                assert line.length() == EQUATION_LENGTH : "Equation length is
incorrect";
                equations.add(line);
            }
        } catch (IOException e) {
            e.printStackTrace(); // Print stack trace for debugging
        }
    }

    /**
     * Selects the target equation for the game, either randomly or a
predetermined one for testing.
     * @ requires !equations.isEmpty() : "Equations list must not be empty"
     * @ ensures targetEquation != null && equations.contains(targetEquation)
     */
    public void selectTargetEquation() {
        assert !equations.isEmpty() : "Equations list must not be empty";
        if (randomlySelectEquation) {
            Random rand = new Random();
            int index = rand.nextInt(equations.size());
            targetEquation = equations.get(index);
        } else {
            targetEquation = equations.get(0);  // Set to a fixed equation
        }
    }

    /**
     * Display the target equation according to flag 2.
     */
    @Override
    public void displayTargetEquation() {
        if (displayEquationForTesting) {
            System.out.println("Target Equation: " + targetEquation);
        }
    }

    /**
     * Validates the user's input against a set of rules to ensure it forms a
valid equation.
     * @param input The user's input equation.
     * @ invariant targetEquation.length() == EQUATION_LENGTH;
     * @ requires input != null;
     * @ ensures \result == (\result ? evaluateEquationSides(input) : false);
     */
    @Override
    public boolean checkInput(String input) {
        assert input != null : "Input should not be null";  // Assert to ensure
input is not null

        // Various checks on the input string to ensure it is a valid equation
        // 1. Check if the length is not 7
        if (input.length() != getEquationLength()) {
            setError("The length of the equation must be 7");
            return false;
        }
```

```
        // 2. Check if the input contains an equal sign
        else if (!input.contains("=")) {
            setError("No equal '=' sign");
            return false;
        }

        // 3. Check if the input contains at least one -+*/ symbol
        else if (!input.matches(".*[-+*/].*")) {
            setError("There must be at least one sign +-*/");
            return false;
        }

        // 4. Check if the input contains consecutive -+*/ symbols
        else if (input.matches(".*[-+*/]{2,}.*")) {
            setError("Multiple math symbol in a row");
            return false;
        }

        // 5. Check if the input contains any characters other than digits, -+*/=
        else if (input.matches(".*[^0-9-+*/=].*")) {
            setError("Illegal math symbols detected");
            return false;
        }

        // 6. Split the expression by the first equal sign only.
        // If it contains more than one equals sign, it will be handled in
evaluate method.
        else {
           return evaluateEquationSides(input);
        }
    }

    /**
     * Evaluates the equation sides to check if they are equal.
     * @param input User input containing the equation.
     * @return true if both sides of the equation evaluate to the same result,
false otherwise.
     */
    private boolean evaluateEquationSides(String input) {
        int indexOfEqual = input.indexOf('=');
        String leftPart = input.substring(0, indexOfEqual);
        String rightPart = input.substring(indexOfEqual + 1);

        try {
            int leftResult = compute(leftPart);
            int rightResult = compute(rightPart);

            // Handle division by zero specially. 8/0=9/0 can be validated.
            if (leftResult == Integer.MAX_VALUE && rightResult == Integer.
MAX_VALUE) {
                return true; // Both sides are division by zero
            }

            if (leftResult != rightResult) {
                setError("The left side is not equal to the right side");
                return false;
            }
            return true;
```

```
        } catch (Exception e) {
            setError("Calculation error. Please enter a valid equation");
            return false;
        }
    }

    /**
     * Evaluates a mathematical expression and returns the calculated result.
     * @param expression The mathematical expression to evaluate.
     * @return The result of the expression as an integer.
     */
    private int compute(String expression) {
        // Arithmetic expressions are evaluated using BODMAS.
        // BODMAS stands for "Brackets, Orders (exponents), Division and
multiplication, Addition and Subtraction."
        // This means that operations within brackets are performed first,
followed by any exponents.
        // Then division and multiplication (from left to right).
        // And finally addition and subtraction (from left to right).
        try {
            expression = expression.replaceAll("\\s+", "");
            List<Object> tokens = new ArrayList<>();

            // Handle parentheses. An aspect that can be expanded in the future

            // Tokenize the expression
            StringBuilder numberBuffer = new StringBuilder();
            for (char ch : expression.toCharArray()) {
                if (Character.isDigit(ch)) {
                    numberBuffer.append(ch);
                } else {
                    if (numberBuffer.length() > 0) {
                        tokens.add(Integer.parseInt(numberBuffer.toString()));
                        numberBuffer = new StringBuilder();
                    }
                    tokens.add(ch);
                }
            }
            if (numberBuffer.length() > 0) {
                tokens.add(Integer.parseInt(numberBuffer.toString()));
            }

            // Handle exponentiation. An aspect that can be expanded in the
future

            // Handle multiplication and division
            handleMultiplicationAndDivision(tokens);

            // Handle addition and subtraction
            return handleAdditionAndSubtraction(tokens);
        } catch (Exception e) {
            // If any exception is encountered
            throw new RuntimeException("Error in expression evaluation: " + e.
getMessage(), e);
        }
    }

    /**
     * Handles multiplication and division in the list of tokens.
```

```
     * @param tokens List of tokens including numbers and operators.
     */
    private void handleMultiplicationAndDivision(List<Object> tokens) {
        for (int i = 0; i < tokens.size(); i++) {
            if (tokens.get(i) instanceof Character) {
                char operator = (Character) tokens.get(i);
                if (operator == '*' || operator == '/') {
                    int left = (Integer) tokens.get(i - 1);
                    int right = (Integer) tokens.get(i + 1);
                    if (operator == '/' && right == 0) {
                        tokens.set(i - 1, Integer.MAX_VALUE); // Use MAX_VALUE to
 represent division by zero
                        tokens.remove(i);
                        tokens.remove(i);
                        i--;
                        continue;
                    }
                    int result = operator == '*' ? left * right : left / right;
                    tokens.set(i - 1, result);
                    tokens.remove(i);
                    tokens.remove(i);
                    i--;
                }
            }
        }
    }

    /**
     * Computes the final result from the list of tokens, supporting leading and
unary plus and minus.
     * @param tokens List of tokens including numbers and operators.
     * @return Final calculation result as an integer.
     */
    private int handleAdditionAndSubtraction(List<Object> tokens) {
        // Initialize result and starting index
        int result;
        int i;

        // Handle leading plus or minus. For example, +5+6=11 or -1-2=-3 is valid
 equation
        if (tokens.get(0) instanceof Character && (tokens.get(0).equals('+') ||
tokens.get(0).equals('-'))) {
            // Start processing from the next operator after the unary plus
            if (tokens.get(0).equals('-')) {
                result = -(Integer) tokens.get(1); // Apply unary minus
            } else {
                result = (Integer) tokens.get(1); // Unary plus doesn't change
the number
            }
            i = 2; // Start processing from the next operator after the unary
minus
        } else {
            result = (Integer) tokens.get(0); // Normal case, start with the
first number
            i = 1; // Start processing from the first operator
        }

        // Process the rest of the tokens
        for (; i < tokens.size(); i += 2) {
```

```
            if (i + 1 < tokens.size()) { // Ensure there is a number following
the operator
                char operator = (Character) tokens.get(i);
                int right = (Integer) tokens.get(i + 1);
                if (operator == '+') {
                    result += right;
                } else if (operator == '-') {
                    result -= right;
                }
            }
        }

        return result;
    }

    /**
     * Processes the user's guess by comparing it against the target equation and
 updates the game state.
     * Decrements the remaining attempts and notifies observers of any changes.
     * @param input The user's guess to process.
     */
    @Override
    public void processInput(String input) {
        assert targetEquation != null : "Target equation must not be null";
        assert targetEquation.length() == EQUATION_LENGTH : "Target equation
length is incorrect";
        compareEquations(input, targetEquation);
        remainingAttempts--;
        setChanged();
        notifyObservers();
    }

    /**
     * Compares the user's guess against the target equation and provides visual
feedback.
     * Determines which characters are correctly placed, present but misplaced,
or absent.
     * Updates the color feedback accordingly.
     * @param currentGuess The user's current guess.
     * @param targetEquation The target equation to compare against.
     * @return An array of color codes indicating feedback for each character in
the guess.
     */
    public String[] compareEquations(String currentGuess, String targetEquation
) {
        StringBuilder feedback = new StringBuilder();
        colors = new String[EQUATION_LENGTH];
        buttonColors = new HashMap<>();
        Map<Character, Integer> targetCounts = new HashMap<>();
        Map<Character, Integer> guessCounts = new HashMap<>();

        // Count each character in the target equation to manage character
frequencies
        for (char c : targetEquation.toCharArray()) {
            targetCounts.put(c, targetCounts.getOrDefault(c, 0) + 1);
        }

        // Remove used characters from the unused character set
        for (char c : currentGuess.toCharArray()) {
```

```
            unusedChars.remove(c);
        }

        // Check if the current guess exactly matches the target equation
        if (currentGuess.equals(targetEquation)) {
            String ANSI_RESET = "\033[0m"; // ANSI color code. Default Color
            String ANSI_GREEN = "\033[32m";   // Green color for correct guesses
            gameWon = true;
            Arrays.fill(colors, "Green");
            for (char c : currentGuess.toCharArray()) {
                buttonColors.put(c, "Green");
            }
            feedback.append(ANSI_GREEN).append(currentGuess).append(ANSI_RESET);
            System.out.println(feedback);
        } else {
            // Process exact matches
            for (int i = 0; i < EQUATION_LENGTH; i++) {
                char guessChar = currentGuess.charAt(i);
                if (guessChar == targetEquation.charAt(i) && targetCounts.get(
guessChar) > 0) {
                    colors[i] = "Green";
                    buttonColors.put(guessChar, "Green");
                    targetCounts.put(guessChar, targetCounts.get(guessChar) - 1
); // Decrease in count
                    guessCounts.put(guessChar, guessCounts.getOrDefault(guessChar
, 0) + 1);
                }
            }
            // Then handle partial matches and non-existent characters
            for (int i = 0; i < EQUATION_LENGTH; i++) {
                if (colors[i] == null) { // Only processed if the position is not
 marked green
                    char guessChar = currentGuess.charAt(i);
                    if (targetCounts.getOrDefault(guessChar, 0) > 0) {
                        colors[i] = "Orange";
                        buttonColors.putIfAbsent(guessChar, "Orange");
                        targetCounts.put(guessChar, targetCounts.get(guessChar
) - 1);
                        guessCounts.put(guessChar, guessCounts.getOrDefault(
guessChar, 0) + 1);
                    } else {
                        colors[i] = "Grey";
                        buttonColors.putIfAbsent(guessChar, "Grey");
                    }
                }
            }
            // Build feedback based on the determined colors
            buildFeedback(currentGuess);
        }
        return colors; // Return the array of color feedback for each character
    }

    /**
     * Builds the visual feedback for the user's guess by appending ANSI color
codes to each character.
     * Each character in the feedback string is colored.
     * The method also prints hints and a summary of characters that have not
been used yet.
     * @param currentGuess The current guess string whose characters are
```

```
 evaluated against the target equation.
     */
    private void buildFeedback(String currentGuess) {
        StringBuilder feedback = new StringBuilder();
        for (int i = 0; i < EQUATION_LENGTH; i++) {
            char guessChar = currentGuess.charAt(i);
            // ANSI color code. Default Color
            String ANSI_RESET = "\033[0m";
            if (colors[i].equals("Green")) {
                // Green color for correct guesses
                String ANSI_GREEN = "\033[32m";
                feedback.append(ANSI_GREEN).append(guessChar).append(ANSI_RESET);
            } else if (colors[i].equals("Orange")) {
                // Existing but not in the right place, use orange color
                String ANSI_ORANGE = "\033[38;5;208m";
                feedback.append(ANSI_ORANGE).append(guessChar).append(ANSI_RESET
);
            } else {
                // Not existed in the target equation, use gray color
                String ANSI_WHITE = "\033[90m";
                feedback.append(ANSI_WHITE).append(guessChar).append(ANSI_RESET);
            }
        }

        System.out.println("Hints:");
        System.out.println(feedback);
        System.out.println("Green indicates that numbers or math symbols are
exist and are in the correct position");
        System.out.println("Orange indicates that numbers or math symbols are
exist but not in the correct position");
        System.out.println("Gray represents numbers or math symbols not exist in
the target equation");
        System.out.println("You have not used these numbers and symbols yet: " +
unusedChars);
    }

    /**
     * Sets the error message to be displayed for invalid inputs.
     * @param error The error message to set.
     */
    public void setError(String error) {
        this.errorMessage = error;
    }

    /**
     * Retrieves the current error message.
     * @return The current error message if any.
     */
    @Override
    public String getError() {
        return this.errorMessage;
    }

    /**
     * Checks if the game is over, either because all attempts are used or the
correct equation has been guessed.
     * @return true if the game is over, false otherwise.
     */
    @Override
```

```
    public boolean isGameOver() {
        return remainingAttempts <= 0 || gameWon;
    }

    /**
     * Checks if the game has been won.
     * @return true if the game is won, false otherwise.
     */
    @Override
    public boolean isGameWon() {
        return gameWon;
    }

    @Override
    public boolean getFlag1() {
        return showErrorOnInvalidEquation; // Return the value of flag_1
    }

    @Override
    public boolean getFlag2() {
        return displayEquationForTesting; // Return the value of flag_2
    }

    @Override
    public String getTargetEquation() {
        return targetEquation;
    }

    @Override
    public int getEquationLength() {
        return EQUATION_LENGTH;
    }

    @Override
    public int getMAX_ATTEMPTS() {
        return MAX_ATTEMPTS; // Return the maximum attempts
    }

    @Override
    public int getRemainingAttempts() {
        return remainingAttempts;
    }

    @Override
    public String[] getColors() {
        return colors;
    }

    @Override
    public Map<Character, String> getButtonColors() {
        return buttonColors;
    }

    @Override
    public void startNewGame() {
        initialize();
    }

    private void initializeRemainingAttempts() {
```

```
        remainingAttempts = MAX_ATTEMPTS; // Initialize remaining attempts to
maximum attempts
    }

    private void initializeUnusedChars() {
        unusedChars.addAll(Arrays.asList('0', '1', '2', '3', '4', '5', '6', '7',
'8', '9', '+', '-', '*', '/', '='));
    }

}
```

```java
import java.util.List;
import java.util.Map;

/**
 * Interface for the Numberle game model.
 * Defines the essential operations and properties that the model must implement.
 */
public interface INumberleModel {
    int MAX_ATTEMPTS = 6; // The maximum number of attempts allowed in the game
    int EQUATION_LENGTH = 7; // The fixed length of the mathematical equations to
 guess
    String EQUATION_FILE = "equations.txt";  // File path for loading potential
target equations

    /**
     * Initializes or resets the game to its starting state.
     */
    void initialize();

    /**
     * Processes the user's input guess and updates the game state accordingly.
     * @param input the player's guessed equation as a String
     */
    void processInput(String input);

    /**
     * Retrieves the current error message if any validation fails.
     * @return current error message
     */
    String getError();

    /**
     * Checks if the game is over either through winning or exhausting all
attempts.
     * @return true if the game is over, false otherwise
     */
    boolean isGameOver();

    /**
     * Checks if the game has been won.
     * @return true if the game is won, false otherwise
     */
    boolean isGameWon();

    /**
     * Retrieves the target equation for the current game.
     * @return the target equation as a String
     */
    String getTargetEquation();

    /**
     * Returns the fixed length of the equations used in the game.
     * @return the length of equations
     */
    int getEquationLength();

    /**
     * Returns the fixed length of the equations used in the game.
     * @return the length of equations
```

```
     */
    int getMAX_ATTEMPTS();

    /**
     * Provides the current color feedback for each character in the last guess.
     * @return an array of color codes as Strings, corresponding to the feedback
for each character
     */
    String[] getColors();

    /**
     * Provides the color states of buttons in the GUI.
     * @return a map of characters to their corresponding color codes
     */
    Map<Character, String> getButtonColors();

    /**
     * Retrieves the status of the first flag concerning error handling on
invalid inputs.
     * @return true if errors on invalid inputs should be shown, false otherwise
     */
    boolean getFlag1();

    /**
     * Retrieves the status of the second flag concerning display the target
equation.
     * @return true if the target equation should be shown, false otherwise
     */
    boolean getFlag2();

    /**
     * Display the target equation if the flag 2 is true.
     */
    void displayTargetEquation();

    /**
     * Validates the user's input to check if it conforms to the rules of forming
 equations.
     * @param input the player's input equation to validate
     * @return true if the input is valid, false if invalid
     */
    boolean checkInput(String input);

    /**
     * Retrieves the number of remaining attempts the player has to guess the
equation.
     * @return the number of remaining attempts
     */
    int getRemainingAttempts();

    /**
     * Starts a new game by reinitializing the game model.
     */
    void startNewGame();
}
```

```java
// NumberleView.java
import javax.swing.*;
import java.awt.*;
import java.util.Observer;
import java.awt.event.*;
import java.util.HashMap;
import java.util.Map;
import java.awt.geom.RoundRectangle2D;

/**
 * The view component of the MVC pattern for the Numberle game. This class handles the user interface,
 * including setup and updates to the visual components based on changes in the game state.
 */
public class NumberleView implements Observer {
    private final INumberleModel model;
    private final NumberleController controller;
    private final JFrame frame = new JFrame("Numberle");
    private JPanel menuPanel; // Panel on the west side for game info
    private String targetEquation; // String target equation when visible
    private JPanel inputPanel; // Panel for the input text boxes
    private JTextField[][] equationFields; // Grid of text fields for equation input
    private Map<String, JButton> buttonMap = new HashMap<>(); // Map of buttons
    private JPanel mathPanel; // Panel for number and operator buttons (keyboard)
    private JButton restartButton; // Button to restart the game
    private JButton showTargetEquationButton; // Button to show target equation
    private KeyAdapter keyAdapter; // Adapter for keyboard input

    /**
     * Constructs a NumberleView instance with associated model and controller.
     *
     * @param model The game model.
     * @param controller The controller that mediates between model and view.
     */
    public NumberleView(INumberleModel model, NumberleController controller) {
        this.controller = controller;
        this.model = model;
        this.controller.startNewGame();
        ((NumberleModel)this.model).addObserver(this);
        initializeFrame();
        this.controller.setView(this);
        update((NumberleModel)this.model, null); // Initial update to configure UI elements
    }

    /**
     * Initializes and displays the main game window.
     */
    public void initializeFrame() {
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(1200, 1000); // Resizing the window
        frame.setBackground(Color.white); // Setting the Layout

        setupMenuPanel();
        setInputPanels(); // Set the text box
        setKeyboard(); // Set the keyboard
        setupKeyboardListener(); // Set up Keyboard Listener
```

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

    /**
     * Sets up the north panel which contains two buttons.
     *  Button 1: show target equation.
     *  Button 2: restart the game
     */
    private void setupMenuPanel() {
        menuPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
        menuPanel.setBackground(Color.WHITE); // Set background color

        // Create button 1
        // When flag 2 is true, button 1 is enabled.
        showTargetEquationButton = createMenuButton("Target Equation", controller
.getFlag2());
        showTargetEquationButton.addActionListener(e -> {
            JLabel messageLabel = new JLabel(targetEquation);
            messageLabel.setFont(new Font("SansSerif", Font.BOLD, 20));
            JOptionPane.showMessageDialog(frame, messageLabel,
                    "Target Equation", JOptionPane.INFORMATION_MESSAGE);
        });

        // Create button 2
        restartButton = createMenuButton("Restart", false);
        restartButton.addActionListener(e -> {
            controller.startNewGame(); // Call the controller's startNewGame
method when the button is clicked
            inputPanel.requestFocusInWindow();
            resetGameInterface();
        });

        // Add buttons to menuPanel
        menuPanel.add(showTargetEquationButton);
        menuPanel.add(restartButton);
        // Add the Start New Game button to the interface
        frame.add(menuPanel,BorderLayout.NORTH); // Add the panel to the north
side of the window
    }

    private JButton createMenuButton(String text, boolean enabled) {
        JButton button = new JButton(text);
        button.setEnabled(enabled);
        button.setFont(new Font("SansSerif", Font.BOLD, 21));
        button.setBorderPainted(false);
        button.setFocusPainted(false);
        button.setOpaque(true);
        button.setForeground(Color.BLACK);
        button.setBackground(new Color(230, 230, 230));
        button.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseEntered(java.awt.event.MouseEvent evt) {
                button.setBackground(new Color(210, 210, 210));
            }
            public void mouseExited(java.awt.event.MouseEvent evt) {
                button.setBackground(new Color(230, 230, 230));
            }
        });
        return button;
```

```
    }

    /**
     * Resets the game interface to its initial state.
     */
    private void resetGameInterface() {
        Color inputBackgroudColor = Color.WHITE;  // Setting the default
background color
        Color keyboardBackgroudColor = new Color(218, 223, 235);

        for (int row = 0; row < 6; row++) {
            for (int col = 0; col < 7; col++) {
                equationFields[row][col].setText("");  // Clear the text of all
input fields
                equationFields[row][col].setBackground(inputBackgroudColor);
                equationFields[row][col].setForeground(new Color(89, 98, 117));
            }
        }

        for (JButton button : buttonMap.values()) {
            button.setBackground(keyboardBackgroudColor);
            button.setForeground(new Color(89, 98, 117));
        }
    }

    /**
     * Sets up the input panel for entering guesses.
     */
    private void setInputPanels() {
        // The input square panel, 6 rows and 7 columns
        inputPanel = new JPanel(new GridLayout(6, 7, 10, 10)); // Spacing of 2
        inputPanel.setBorder(BorderFactory.createEmptyBorder(10, 245, 50, 245));
        inputPanel.setBackground(Color.WHITE); // Set background color
        inputPanel.requestFocusInWindow();
        equationFields = new JTextField[6][7];

        Font textFieldFont = new Font("SansSerif", Font.BOLD, 24); // Set up the
size and font
        for (int row = 0; row < 6; row++) {
            for (int col = 0; col < 7; col++) {
                equationFields[row][col] = new RoundedTextField();
                equationFields[row][col].setEditable(false);
                equationFields[row][col].setHorizontalAlignment(JTextField.CENTER
); // Horizontal centering of text
                equationFields[row][col].setFont(textFieldFont);
                equationFields[row][col].setBackground(new Color(249, 250, 253));
                equationFields[row][col].setForeground(new Color(89, 98, 117));
                inputPanel.add(equationFields[row][col]);
            }
        }
        frame.add(inputPanel, BorderLayout.CENTER);
    }

    /**
     * Sets up the keyboard panel for number and operator input.
     */
    private void setKeyboard() {
        // Initialize the math panel with extra vertical space between button
rows
```

```
        mathPanel = new JPanel();
        mathPanel.setLayout(new BoxLayout(mathPanel, BoxLayout.Y_AXIS)); // Use
BoxLayout for vertical stacking
        mathPanel.setBorder(BorderFactory.createEmptyBorder(0, 80, 80, 80));
        mathPanel.setBackground(Color.WHITE);

        // Add number buttons panel
        addButtonsPanel("number", new String[]{"1", "2", "3", "4", "5", "6", "7"
, "8", "9", "0"}, 10);

        // Add an empty panel as a vertical spacer
        JPanel spacerPanel = new JPanel();
        spacerPanel.setPreferredSize(new Dimension(800, 10)); // Set the height
to 10 for vertical spacing
        spacerPanel.setOpaque(false); // Make the spacer panel transparent
        mathPanel.add(spacerPanel);

        // Add operator buttons panel
        addButtonsPanel("operator", new String[]{"Delete", "+", "-", "*", "/",
"=", "Enter"}, 7);

        frame.add(mathPanel, BorderLayout.SOUTH);
    }

    /**
     * Adds a panel of buttons with the specified labels and layout configuration
.
     * @param type The type of buttons, e.g., "number" or "operator".
     * @param buttons An array of strings representing the button labels.
     * @param columns The number of columns for the grid layout.
     */
    private void addButtonsPanel(String type, String[] buttons, int columns) {
        JPanel buttonsPanel = new JPanel();
        buttonsPanel.setLayout(new GridLayout(1, columns, 10, 10));
        buttonsPanel.setPreferredSize(new Dimension(800, 90));
        buttonsPanel.setBackground(Color.WHITE);

        Font buttonFont = new Font("SansSerif", Font.BOLD, 24);
        Color backgroundColor = new Color(218, 223, 235);

        for (String buttonText : buttons) {
            JButton button = createInputButton(buttonText, buttonFont,
backgroundColor);
            if ("Delete".equals(buttonText) || "Enter".equals(buttonText)) {
                configureSpecialButton(button, buttonText);
            } else {
                button.addActionListener(e -> updateInputPanel(getInputText() +
button.getText()));
            }
            buttonsPanel.add(button); // Add the button directly to the button
panel
            buttonMap.put(buttonText, button);
        }
        mathPanel.add(buttonsPanel);
    }

    /**
     * Creates a round button with specified text, font, and background color.
     * @param text The text to display on the button.
```

```
     * @param font The font of the button text.
     * @param bgColor The background color of the button.
     * @return Button The newly created round button.
     */
    private RoundedButton createInputButton(String text, Font font, Color bgColor
) {
        RoundedButton button = new RoundedButton(text);
        button.setEnabled(true);
        button.setFont(font);
        button.setBackground(bgColor);
        button.setForeground(new Color(89, 98, 117)); // Customize text color
        return button;
    }

    /**
     * Configures the action listener for special buttons like 'Delete' and '
Enter'.
     * @param button The button to configure.
     * @param type The type of the button which determines its functionality.
     */
    private void configureSpecialButton(JButton button, String type) {
        button.addActionListener(e -> {
            if ("Delete".equals(type)) {
                String currentText = getInputText();
                if (!currentText.isEmpty()) {
                    updateInputPanel(currentText.substring(0, currentText.length
() - 1));
                }
            } else if ("Enter".equals(type)) {
                if (controller.checkInput(getInputText())) {
                    controller.processInput(getInputText());
                    updateInputColors();
                    updateKeyboardColor();
                    checkGameOver();
                } else {
                    if (controller.getFlag1()) {
                        JLabel errorLabel = new JLabel(controller.getError());
                        errorLabel.setFont(new Font("SansSerif", Font.BOLD, 20));
                        JOptionPane.showMessageDialog(frame, errorLabel);
                    }
                }
            }
        });
    }

    /**
     * Retrieves the text from the input fields for the current guess.
     * This method compiles the text from each JTextField in the row
corresponding to the current attempt.
     * @return A string composed of the characters currently entered the input
fields.
     */
    private String getInputText() {
        // Get the content of the text box panel from the current guess row
        int currentGuess = controller.getMAX_ATTEMPTS() - controller.
getRemainingAttempts();
        int startRow = currentGuess % 6; // Row number corresponding to the
current guess count
```

```
        StringBuilder sb = new StringBuilder();
        for (int col = 0; col < 7; col++) { // Since there are 7 columns in each
row
            JTextField textField = equationFields[startRow][col];
            String text = textField.getText();
            sb.append(text);
        }
        return sb.toString();
    }

    /**
     * Updates the text fields in the input panel based on the current user input
.
     * This method sets each JTextField in the row for the current guess to
display the appropriate characters.
     *
     * @param text The current string to display in the input fields.
     */
    private void updateInputPanel(String text) {
        inputPanel.requestFocusInWindow();
        // Determine the current attempt and calculate the starting row
        int currentGuess = controller.getMAX_ATTEMPTS() - controller.
getRemainingAttempts();
        int startRow = currentGuess % 6;  // Assuming the currentGuess is zero-
based

        // Calculate the starting and ending indices for this row
        int startIndex = 0;  // Always starts at the first column of the row
        int endIndex = 7;    // Ends at the last column of the row (since there
are 7 columns)

        // Update the text fields in the specified row
        JTextField[] rowFields = equationFields[startRow];
        for (int i = startIndex; i < endIndex; i++) {
            if (i < text.length()) {
                rowFields[i].setText(String.valueOf(text.charAt(i)));
            } else {
                rowFields[i].setText("");  // Clear the field if there is no
character to display
            }
        }
    }

    /**
     * Updates the background colors of the text fields based on feedback from
the game model.
     * Colors are used to indicate the correctness of each character in the guess
.
     */
    private void updateInputColors() {
        String[] colors = controller.getColors(); // Get an array of colors in
the model
        int currentGuess = controller.getMAX_ATTEMPTS() - controller.
getRemainingAttempts() -1;
        int startRow = currentGuess % 6; // Calculate the line number
corresponding to the current attempt

        JTextField[] rowFields = equationFields[startRow]; // Get all text fields
 of the current line
```

```
        for (int i = 0; i < rowFields.length; i++) {
            JTextField textField = rowFields[i];
            textField.setForeground(Color.WHITE);
            if (i < colors.length) {
                switch (colors[i]) {
                    case "Green":
                        textField.setBackground(new Color(47, 191, 164));
                        break;
                    case "Orange":
                        textField.setBackground(new Color(245, 153, 110));
                        break;
                    case "Grey":
                        textField.setBackground(new Color(163, 173, 194));
                        break;
                    default:
                        textField.setBackground(Color.WHITE);
                        break;
                }
            } else {
                textField.setBackground(Color.WHITE);
            }
        }
    }

    /**
     * Updates the colors of the keyboard buttons based on their usage in the
game.
     * This function uses the feedback from the model to highlight keys that have
 been used correctly, incorrectly, or are close.
     */
    private void updateKeyboardColor() {
        Map<Character, String> colors = controller.getButtonColors();
        for (Map.Entry<String, JButton> entry : buttonMap.entrySet()) {
            String key = entry.getKey();
            JButton button = entry.getValue();
            if (key.length() == 1 && colors.containsKey(key.charAt(0))) {
                Color currentColor = button.getBackground();
                String newColorCode = colors.get(key.charAt(0));
                button.setForeground(Color.WHITE);

                // Check if the current color is already green, if so, do not
make any changes
                if (!currentColor.equals(new Color(47, 191, 164))) {
                    switch (newColorCode) {
                        case "Green":
                            button.setBackground(new Color(47, 191, 164));
                            break;
                        case "Orange":
                            button.setBackground(new Color(245, 153, 110));
                            break;
                        case "Grey":
                            button.setBackground(new Color(163, 173, 194));
                            break;
                        default:
                            button.setBackground(Color.WHITE);
                            break;
                    }
                }
            }
```

```
        }
    }

    /**
     * Sets up the listener for physical keyboard inputs.
     * This method allows the view to respond to keystrokes, updating the game
state accordingly.
     */
    private void setupKeyboardListener() {
        keyAdapter = new KeyAdapter() {
            @Override
            public void keyPressed(KeyEvent e) {
                char keyChar = e.getKeyChar();
                String currentText = getInputText();
                if (e.getKeyCode() == KeyEvent.VK_BACK_SPACE) {
                    // Handling the backspace button
                    if (!currentText.isEmpty()) {
                        updateInputPanel(currentText.substring(0, currentText.
length() - 1));
                    }
                } else if (e.getKeyCode() == KeyEvent.VK_ENTER) {
                    // Handling the Enter button
                    if (controller.checkInput(currentText)) {
                        controller.processInput(currentText);
                        updateInputColors();
                        updateKeyboardColor();
                        checkGameOver();
                    } else {
                        if (controller.getFlag1()) {
                            JLabel errorLabel = new JLabel(controller.getError
());
                            errorLabel.setFont(new Font("SansSerif", Font.BOLD,
20));
                            JOptionPane.showMessageDialog(frame, errorLabel);
                        }
                    }
                } else {
                    // Handling number and operator buttons
                    if (Character.isDigit(keyChar) || "+-*/=".indexOf(keyChar
) != -1) {
                        updateInputPanel(currentText + keyChar);
                    }
                }
            }
        };

        // Adding a key listener to the component
        inputPanel.addKeyListener(keyAdapter);
        // Set focus to receive keyboard input
        inputPanel.setFocusable(true);
        // Set focus to inputPanel
        inputPanel.requestFocusInWindow();
    }

    @Override
    public void update(java.util.Observable o, Object arg) {
        updateRestartButton();
        updateTargetEquation();
    }
```

```
    /**
     * Updates the visibility and text of the target equation.
     */
    private void updateTargetEquation() {
        targetEquation = controller.getTargetEquation();
    }

    /**
     * Enables or disables the restart button based on game state.
     */
    private void updateRestartButton() {
        // Update logic for the restart button: Unable to start a new game when
the user has not completed the first attempt
        // Once the user has completed the first attempt, a new game can be
started at any time
        restartButton.setEnabled(controller.getMAX_ATTEMPTS() - controller.
getRemainingAttempts() >= 1);
    }

    /**
     * Checks if the game is over and prompts the user accordingly.
     */
    private void checkGameOver() {
        // Check if the game is over
        if (controller.isGameOver()) {
            String messageText;
            String title;

            if (controller.isGameWon()) {
                // Win the game
                messageText = "<html>You won!<br>Would you like to play again?</
html>";
                title = "Game Won";
            } else {
                // loss the game
                messageText = "<html>You lost. The equation was: " + controller.
getTargetEquation() +
                        "<br>Would you like to play again?</html>";
                title = "Game Lost";
            }

            // Create a JLabel with custom font for the message
            JLabel message = new JLabel(messageText);
            message.setFont(new Font("Sans Serif", Font.BOLD, 20));
            Object[] options = {"Yes", "No"};
            // A confirmation dialog box is displayed asking if the user wants to
 restart the game
            int response = JOptionPane.showOptionDialog(frame, message, title,
JOptionPane.YES_NO_OPTION,
                    JOptionPane.INFORMATION_MESSAGE, null, options, options[0]);

            if (response == JOptionPane.YES_OPTION) {
                controller.startNewGame();  // The user selects yes to restart
the game
                resetGameInterface();  // Reset the game interface
            } else {
                frame.dispose();  // User selects No to close the game window
            }
```

```
        }
    }

    /**
     * Custom text field with rounded corners.
     */
    private static class RoundedTextField extends JTextField {
        private Shape shape;

        public RoundedTextField() {
            setOpaque(false);
        }
        protected void paintComponent(Graphics g) {
            Graphics2D g2 = (Graphics2D) g;
            g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.
VALUE_ANTIALIAS_ON);
            g2.setColor(getBackground());
            g2.fillRoundRect(0, 0, getWidth()-1, getHeight()-1, 20, 20);
            super.paintComponent(g2);
        }
        protected void paintBorder(Graphics g) {
            Graphics2D g2 = (Graphics2D) g;
            g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.
VALUE_ANTIALIAS_ON);
            // Define the border width
            int borderWidth = 3;
            g2.setStroke(new BasicStroke(borderWidth));
            g2.setColor(new Color(230, 230, 230));
            g2.drawRoundRect(0, 0, getWidth()-1, getHeight()-1, 20, 20);
        }
        public boolean contains(int x, int y) {
            if (shape == null || !shape.getBounds().equals(getBounds())) {
                shape = new RoundRectangle2D.Float(0, 0, getWidth()-1, getHeight
()-1, 20, 20);
            }
            return shape.contains(x, y);
        }
    }

    /**
     * Custom button rounded corners.
     */
    private static class RoundedButton extends JButton {
        private Shape shape;

        public RoundedButton(String text) {
            super(text);
            setContentAreaFilled(false); // Make the button transparent
            setFocusPainted(false); // Remove the focus border
        }
        protected void paintComponent(Graphics g) {
            Graphics2D g2 = (Graphics2D) g;
            g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.
VALUE_ANTIALIAS_ON);
            g2.setColor(getBackground());
            g2.fillRoundRect(0, 0, getWidth()-1, getHeight()-1, 20, 20);
            super.paintComponent(g2);
        }
        protected void paintBorder(Graphics g) {
```

```
            Graphics2D g2 = (Graphics2D) g;
            g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.
 VALUE_ANTIALIAS_ON);
            // Define the border width
            int borderWidth = 2;
            g2.setStroke(new BasicStroke(borderWidth));
            g2.drawRoundRect(0, 0, getWidth()-1, getHeight()-1, 20, 20);
        }
        public boolean contains(int x, int y) {
            if (shape == null || !shape.getBounds().equals(getBounds())) {
                shape = new RoundRectangle2D.Float(0, 0, getWidth()-1, getHeight
 ()-1, 20, 20);
            }
            return shape.contains(x, y);
        }
    }
 }
```

```java
import java.awt.*;
import java.util.Map;

/**
 * Controller class for the Numberle game.
 * Mediates interactions between the model and the view.
 */
public class NumberleController {
    private INumberleModel model;
    private NumberleView view;

    /**
     * Constructor to initialize the controller with a model.
     * @param model The model object that handles the game logic.
     */
    public NumberleController(INumberleModel model) {
        this.model = model;
    }

    /**
     * Sets the view for this controller.
     * @param view The view that displays the game UI.
     */
    public void setView(NumberleView view) {
        this.view = view;
    }

    /**
     * Processes the user's input through the model.
     * @param input The equation guess from the user.
     */
    public void processInput(String input) {
        model.processInput(input);
    }

    /**
     * Checks if the game is over.
     * @return true if the game is over, otherwise false.
     */
    public boolean isGameOver() {
        return model.isGameOver();
    }

    /**
     * Checks if the game has been won.
     * @return true if the game has been won, otherwise false.
     */
    public boolean isGameWon() {
        return model.isGameWon();
    }

    /**
     * Retrieves the target equation from the model.
     * @return The target equation.
     */
    public String getTargetEquation() {
        return model.getTargetEquation();
    }
```

```
    /**
     * Gets the number of remaining attempts from the model.
     * @return The number of remaining attempts.
     */
    public int getRemainingAttempts() {
        return model.getRemainingAttempts();
    }

    /**
     * Retrieves the color feedback for each character of the last guess from the
 model.
     * @return An array of color codes as strings.
     */
    public String[] getColors() {
        return model.getColors();
    }

    /**
     * Retrieves the color states of buttons in the GUI from the model.
     * @return A map of characters to their corresponding color codes.
     */
    public Map<Character, String> getButtonColors() {
        return model.getButtonColors();
    }

    /**
     * Starts a new game by reinitializing the model.
     */
    public void startNewGame() {
        model.startNewGame();
    }

    /**
     * Gets the maximum number of attempts allowed from the model.
     * @return The maximum number of attempts.
     */
    public int getMAX_ATTEMPTS() {
        return model.getMAX_ATTEMPTS();
    }

    /**
     * Retrieves the status of the first flag concerning error handling on
invalid inputs.
     * @return true if errors on invalid inputs should be shown, false otherwise.
     */
    public boolean getFlag1() {return model.getFlag1();}

    /**
     * Retrieves the status of the second flag concerning display the target
equation.
     * @return true if the target equation should be shown, false otherwise
     */
    public boolean getFlag2() {return model.getFlag2();}

    /**
     * Validates the user's input to check if it conforms to the rules of forming
 equations.
     * @param input The player's input equation to validate.
     * @return true if the input is valid, false if invalid.
```

```
     */
    public boolean checkInput(String input) {
        return model.checkInput(input);
    }

    /**
     * Retrieves the current error message if any validation fails.
     * @return Current error message.
     */
    public String getError() {return model.getError();}
}
```

```java
import java.util.Scanner;

class CLIApp {

    private INumberleModel model; // Game model

    private Scanner scanner; // Scanner object for reading user input

    public CLIApp() {
        model = new NumberleModel();
        scanner = new Scanner(System.in);
    }

    public void startGame() {
        System.out.println("Welcome to Numberle Game!");
        System.out.println("You have six chances in total."); // Some tips for
players
        System.out.println("The equation you enter must satisfy the following
conditions:");
        System.out.println("1. When calculating, players can use numbers (0-9)
and arithmetic signs (+ - * / =). ");
        System.out.println("2. The length of the equation must be 7.");
        System.out.println("3. Must have equal sign '='.");
        System.out.println("4. There must be at least one sign +-*/ and multiple
 math symbol in a row is not allowed");
        System.out.println("5. Letters are not allowed.");
        System.out.println("----------------------------------------");

        model.startNewGame();
        model.displayTargetEquation();

        while (!model.isGameOver()) { // A while loop. Players are allowed to
enter a valid equation six times
            System.out.println("Remaining attempts: " + model.
getRemainingAttempts());
            System.out.print("Please enter your guess: ");
            String input = scanner.nextLine();

            while (!model.checkInput(input)) { // Check whether is a valid input
.
                if (model.getFlag1()) {
                  System.out.println(model.getError()); // Display specific
error message.
                }
                System.out.print("Enter your guess again: ");
                input = scanner.nextLine();
            }

            model.processInput(input); // Determining whether a player has won

            System.out.println("----------------------------------------");
        }

        if (model.isGameWon()) { // Check if the game is won
            System.out.println("You won!"); // Print victory message
        } else {
            System.out.println("You lost. The answer was: " + model.
getTargetEquation());
        }
```

```
            System.out.println("Do you want to play again? (Y/N)");
            String playAgain = scanner.nextLine();

            if (playAgain.equalsIgnoreCase("Y") ||playAgain.equalsIgnoreCase("y")) {
                startGame(); // Start a new game loop
            } else {
                System.out.println("Bye");
            }
        }

        public static void main(String[] args) {
            CLIApp game = new CLIApp(); // Create a CLI object
            game.startGame(); // Start the game
    }
}
```

```java
import javax.swing.*;

public class GUIApp {
    public static void main(String[] args) {

        javax.swing.SwingUtilities.invokeLater(
                new Runnable() {
                    public void run() {
                        createAndShowGUI();
                    } // Create and show the graphical user interface
                }
        );
    }

    public static void createAndShowGUI() {
        INumberleModel model = new NumberleModel(); // Create a model of Numberle
        NumberleController controller = new NumberleController(model); // Create
a controller of Numberle
        NumberleView view = new NumberleView(model, controller); // Create a view
 of Numberle and pass the model and controller
    }
}
```

```java
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

/*
 * Author: Ryan
 * Date: 2024/4/24
 */
class NumberleModelTest {
    NumberleModel model;

    @BeforeEach
    void setUp() {
        model = new NumberleModel();
        model.initialize(); // Initialize the game state before each test
    }

    @AfterEach
    void tearDown() {
        model = null; // Clean up after each test
    }

    /**
     * Test scenario 1: Validate Error Handling for Invalid Inputs
     * This test verifies that the model correctly handles various types of
invalid inputs
     * by returning the appropriate error messages.
     */
    @Test
    void testInvalidInputHandling() {
        // Input too short
        String tooShortInput = "1+1=2";
        assertFalse(model.checkInput(tooShortInput),
                "Input that is too short should be rejected");
        assertEquals("The length of the equation must be 7", model.getError(),
                "Error message should indicate the equation is too short");

        // Input without an equals sign
        String noEqualsInput = "123+456";
        assertFalse(model.checkInput(noEqualsInput),
                "Input without an equals sign should be rejected");
        assertEquals("No equal '=' sign", model.getError(),
                "Error message should indicate missing '=' sign");

        // Input with invalid characters
        String invalidCharactersInput = "1^1+1=2";
        assertFalse(model.checkInput(invalidCharactersInput),
                "Input with invalid characters should be rejected");
        assertEquals("Illegal math symbols detected", model.getError(),
                "Error message should indicate invalid characters are not allowed
");
    }

    /**
     * Test scenario 2: Verify Correct Input and Game Logic
     * This test ensures that a valid correct guess updates the game state
appropriately,
```

```
     * decrementing the remaining attempts.
     */
    @Test
    void testCorrectInputAndGameLogic() {
        // Set a valid equation but not equals to the target equation for
controlled testing
        String validGuess = "1+1+1=3";
        assertTrue(model.checkInput(validGuess), "Correct and valid input should
be accepted");
        model.processInput(validGuess);
        assertFalse(model.isGameWon(), "Game should be won if correct guess is
processed");
        assertEquals(model.getRemainingAttempts(), NumberleModel.MAX_ATTEMPTS - 1
,
                "Remaining attempts should decrement by one after processing a
guess");
    }

    /**
     * Test scenario 3: Evaluate Visual Feedback for Equation Comparison
     * This test checks the functionality of comparing a user's valid guess
against the target equation to determine
     * the correctness of each character's position in the guess. It specifically
 tests for correct feedback in terms of
     * color coding which reflects correct positions (Green), incorrect positions
 but correct character (Orange), and
     * characters that do not exist in the target equation (Grey).
     * This ensures that the model provides accurate visual feedback to the user.
     */
    @Test
    void testCompareEquations() {
        String validGuess = "1+1-2=0";
        String targetEquation = "1+2+3=6"; // Directly setting for test purpose
        String[] feedback = model.compareEquations(validGuess, targetEquation);
        assertEquals("Green", feedback[0], "Color feedback for correct position
should be 'Green'");
        assertEquals("Green", feedback[1], "Color feedback for correct position
should be 'Green'");
        assertEquals("Grey", feedback[2], "Color feedback for not exist should be
 'Grey'");
        assertEquals("Grey", feedback[3], "Color feedback for not exist should be
 'Grey'");
        assertEquals("Orange", feedback[4], "Color feedback for incorrect
position should be 'Orange'");
        assertEquals("Green", feedback[5], "Color feedback for correct position
should be 'Green'");
        assertEquals("Grey", feedback[6], "Color feedback for not exist should be
 'Grey'");
    }

}
```