成都理工大学中英合作办学
CDUT Sino-British Collaborative Education

OXFORD BROOKES UNIVERSITY

Advanced Object-Oriented Programming Coursework REPORT

Coursework Title:	**Numberle**
Surname:	**XIMENG**
First Name:	**LIANG**
Student Number:	**202018010111**
Teacher Name:	**Gore Jiang**
Module Code:	**CHC 6186**
Module Name:	**Advanced Object Oriented Programming**
Date Submitted:	**13th, May, 2024**

Table of Contents

Chapter 1 Implementation of code

1.1 Numberle_Model Package

1.1.1 Code Structure

The whole code follow MVC structure. The classes and interface about model is in the ‘Numberle_Model ’ package. The view class is in the ‘Numberle_View ’ package. The controller class is in the ‘Numberle_Controller ’ package.

1.1.2 NumberleModel class

```
package Numberle_Model;

import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;
import javax.script.ScriptException;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;

21 usages
public class NumberleModel extends Observable implements INumberleModel {
    15 usages
    private String targetNumber;
    2 usages
    private StringBuilder currentGuess;
    9 usages
    private int remainingAttempts;
    7 usages
    private boolean gameWon;
    6 usages
    private boolean shouldValidateEquation;
    2 usages
    private boolean shouldDisplayTarget;
    1 usage
    private boolean shouldRandomizeTarget;
    4 usages
    private List<String> equations = new ArrayList<>();
    10 usages
    private Set<Character> unusedDigitsAndSymbols = new HashSet<>();
    7 usages
    private Set<Character> presentButIncorrectlyPlaced = new HashSet<>();
    7 usages
    private Set<Character> correctlyPlaced = new HashSet<>();
    6 usages
    private Set<Character> missingFromEquation = new HashSet<>();
```

```java
/**
 * Constructs a new NumberleModel with specified settings for equation validation,
 * displaying the target equation, and randomizing the target equation.
 *
 * @param shouldValidateEquation If true, the input equations are validated against mathematical rules and displayed
 * @param shouldDisplayTarget If true, displays the target equation.
 * @param shouldRandomizeTarget If true, the target equation is generated randomly; otherwise,use default equation
 */
4 usages
public NumberleModel(boolean shouldValidateEquation, boolean shouldDisplayTarget, boolean shouldRandomizeTarget) {
    this.shouldValidateEquation = shouldValidateEquation;
    this.shouldDisplayTarget = shouldDisplayTarget;
    this.shouldRandomizeTarget = shouldRandomizeTarget;
    // Generate the initial target equation based on whether randomization is enabled
    targetNumber = generateTargetEquation();
    if (shouldRandomizeTarget) {
        targetNumber = generateTargetEquation();
    } else {
        // Fixed default equation
        targetNumber = "3+4*1=7";
    }
    if (shouldDisplayTarget) {
        System.out.println("Target equation set to: " + targetNumber);
    }
}

5 usages
public boolean invariant(){
    return(targetNumber != null && targetNumber.length() == 7)&&
            (remainingAttempts >= 0 && remainingAttempts <= MAX_ATTEMPTS)&&
            (!gameWon || isGameOver())&&
            ( !shouldValidateEquation || isMathematicallyCorrect(targetNumber))&&
            hasNoIntersection(unusedDigitsAndSymbols, presentButIncorrectlyPlaced) &&
            hasNoIntersection(unusedDigitsAndSymbols, correctlyPlaced) &&
            hasNoIntersection(unusedDigitsAndSymbols, missingFromEquation) &&
            hasNoIntersection(presentButIncorrectlyPlaced, correctlyPlaced) &&
            hasNoIntersection(presentButIncorrectlyPlaced, missingFromEquation) &&
            hasNoIntersection(correctlyPlaced, missingFromEquation);
}


/**
 * Determines if two sets of characters have no intersection.
 * This method checks whether there is any common element between two character sets.
 *
 * @param set1 The first set of characters to be compared.
 * @param set2 The second set of characters to be compared.
 * @return true if there is no common character between set1 and set2; otherwise, false.
 */
6 usages
private boolean hasNoIntersection(Set<Character> set1, Set<Character> set2) {
    // to check whether the two sets intersect, and return false if they do
    boolean noIntersection = true;

    // Loop through each character in set
    for (char c : set1) {
        if (set2.contains(c)) {
            noIntersection = false;
            break;
        }
    }
    return noIntersection;
}


/**
 * Initializes the game state to default values, ensuring the game is ready to start anew.
 * This method sets the initial conditions for a new game round.
 *
 * @invariant invariant()
 *
 * @requires true
 *           // Pre-condition: No specific requirements since this method is used to initialize or reset the game.
 *
 * @ensures \old(targetNumber).equals(targetNumber) &&
 *          currentGuess.length() == 7 &&
 *          currentGuess.toString().trim().isEmpty() &&
 *          remainingAttempts == MAX_ATTEMPTS &&
 *          !gameWon
 *
 */
```

```
     */
    1 usage
    @Override
    public void initialize() {
        assert targetNumber != null && targetNumber.length() == 7 :
                "Pre-condition failed: targetNumber should be properly set before re-initialization";
        currentGuess = new StringBuilder("       ");
        remainingAttempts = MAX_ATTEMPTS;
        gameWon = false;
        assert !gameWon : "Game won flag should be reset to false";
        setChanged();
        notifyObservers();
//        System.out.println(targetNumber);
        assert invariant() : "Invariant check failed after initialization";
    }


    /**
     * Processes the input by checking its validity against several rules: length, character content,
     * correct number of equals signs, and mathematical correctness. Updates game state accordingly.
     * @invariant invariant()
     *
     * @requires input != null && input.length() == 7;
     *           "Pre-condition: Input must be exactly 7 characters long and non-null."
     *
     * @ensures \old(remainingAttempts) > 0;
     *          "Pre-condition: There should be at least one remaining attempt before processing input."
     *
     * @ensures \result == true <==> (!\old(gameWon) && remainingAttempts > 0 && isInputValid(input));
     *          "Post-condition: True is returned only if the game was not won previously, there are remaining attempts,
     *          and the input was valid according to game rules."
     *
     * @ensures \result == false <==> input.length() != 7 || input has illegal characters ||
     *          input has != 1 equals sign || !isMathematicallyCorrect(input) || \old(remainingAttempts) <= 0;
     *          "Post-condition: False is returned if the input is invalid due to wrong length, illegal characters,
     *          incorrect number of equals signs, mathematical incorrectness, or no remaining attempts."
     *


     * @ensures (remainingAttempts == \old(remainingAttempts) - 1) || (\old(remainingAttempts) == 0);
     *          "Post-condition: Remaining attempts should decrease by one unless they were already zero."
     *
     */
    2 usages
    @Override
    public boolean processInput(String input) {
        assert invariant() : "Invariants check failed at start of processInput";

        // Check if input is null and the length
        if (input == null || input.length() != 7) {
            setChanged();
            notifyObservers( arg: "Invalid Input: The equation must be 7 characters long.");
            System.out.println("Invalid Input: The equation must be 7 characters long.");
            return false;
        }

        assert input.length() == 7 : "Input length must be exactly 7 characters";

        // Check digit and operator
        for (char c : input.toCharArray()) {
            if (!Character.isDigit(c) && !"+-*/=".contains(String.valueOf(c))) {
                if(shouldValidateEquation) {
                    setChanged();
                    notifyObservers( arg: "Invalid Input: The equation contains illegal characters.");
                    System.out.println("Invalid Input: The equation contains illegal characters.");
                    return false;
                }
            }
        }
        // Check equal sign
        int equalsCount = 0;
        for (char c : input.toCharArray()) {
            if (c == '=') {
                equalsCount++;
            }
        }
```

```java
int equalsCount = 0;
for (char c : input.toCharArray()) {
    if (c == '=') {
        equalsCount++;
    }
}
if (equalsCount != 1) {
    if(shouldValidateEquation) {
        setChanged();
        notifyObservers( arg: "Invalid Input: The equation must contain exactly one equals sign (=).");
        System.out.println("Invalid Input: The equation must contain exactly one equals sign (=).");
        return false;
    }
}else{
    // Check at least one operator
    if (!input.matches( regex: ".*[\\+\\-\\*/].*")) {
        if(shouldValidateEquation) {
            setChanged();
            notifyObservers( arg: "Invalid Input: The equation must contain at least one operator (+, -, *, /).")
            System.out.println("Invalid Input: The equation must contain at least one operator (+, -, *, /).");
            return false;
        }
    }else {// Verify whether the left side and right side is equal
        if (!isMathematicallyCorrect(input)) {
            if(shouldValidateEquation) {
                setChanged();
                notifyObservers( arg: "Incorrect Equation: The left side is not equal to right side.");
                System.out.println("Incorrect Equation: The left side is not equal to right side.");
                return false;

            }

        }
    }
}
// Ensure there are remaining attempts before decrementing
assert remainingAttempts > 0 : "Remaining attempts should be greater than zero before decrementing";
remainingAttempts--;
assert remainingAttempts >= 0 : "Remaining attempts should not go below zero";
```

```
ArrayList<String> answerList = new ArrayList<>();
assert answerList.isEmpty() : "Answer list should be empty before processing new input";

// Compare the input with the target number and generate feedback
if (input.equals(targetNumber)) {
    gameWon = true;
    for(int i = 0; i < input.length(); i++) {
        answerList.add("Green");
        System.out.print("Green ");
    }
    System.out.println();
} else {
    for (int i = 0; i < input.length(); i++) {
        char c = input.charAt(i);
        if (i < targetNumber.length() && c == targetNumber.charAt(i)) {
            answerList.add("Green");
            System.out.print("Green!");
        } else if (targetNumber.contains(String.valueOf(c))) {
            answerList.add("Orange");
            System.out.print("Orange! ");
        } else {
            answerList.add("Gray");
            System.out.print("Gray! ");
        }
    }
}
setChanged();
notifyObservers(answerList);
System.out.println();
updateSets(input, targetNumber);

// Check if the game is over and update observers accordingly
if (isGameOver()) {
    remainingAttempts = INumberleModel.MAX_ATTEMPTS;
    setChanged();
    notifyObservers(gameWon ? "Game Won" : "Game Over");
} else {
    System.out.println(getFeedbackMessage());
    setChanged();
```

```
        setChanged();
        notifyObservers(gameWon ? "Game Won" : "Game Over");
    } else {
        System.out.println(getFeedbackMessage());
        setChanged();
        notifyObservers( arg: "Try Again");
    }
    assert invariant() : "Invariants check failed at end of processInput";
    return true;

}
```

```java
/**
 * Constructs and returns a feedback message that describes the state of the current guesses in terms of
 * unused digits and symbols, present but incorrectly placed, correctly placed, and missing elements from equation.
 * @invariant invariant()
 *
 * @invariant unusedDigitsAndSymbols != null && presentButIncorrectlyPlaced != null &&
 *                 correctlyPlaced != null && missingFromEquation != null;
 *
 * @requires true;
 *           "No specific preconditions. Method should always be callable."
 *
 * @ensures \result.contains("Unused digits and symbols:") &&
 *          \result.contains("Present but incorrectly placed:") &&
 *          \result.contains("Exist and correctly placed:") &&
 *          \result.contains("Not in the equation:");
 *          "Post-condition: The result must contain specific sections describing the state of the guesses."
 *
 */
1 usage
public String getFeedbackMessage() {
    assert invariant() : "Invariant check failed at the start of getFeedbackMessage";
    StringBuilder feedback = new StringBuilder();
    feedback.append("Unused digits and symbols: ").append(unusedDigitsAndSymbols).append("\n");
    feedback.append("Present but incorrectly placed: ").append(presentButIncorrectlyPlaced).append("\n");
    feedback.append("Exist and correctly placed: ").append(correctlyPlaced).append("\n");
    feedback.append("Not in the equation: ").append(missingFromEquation).append("\n");
    assert invariant() : "Invariant check failed after getting feedback";
    return feedback.toString();
}
```

```java
/**
 * Updates the sets that track the placement and usage of characters based on the current input compared to the targ
 * This function modifies various sets to reflect the correct placement, presence, and absence of characters from in
 *
 * @param input The player's current guess input.
 * @param target The target or correct equation that needs to be guessed.
 */
1 usage
private void updateSets(String input, String target) {
    for (char c : "0123456789+-*/=".toCharArray()) {
        unusedDigitsAndSymbols.add(c);
    }

    // Compare input and target, upadate
    for (int i = 0; i < input.length(); i++) {
        char inputChar = input.charAt(i);
        char targetChar = target.charAt(i);

        if (inputChar == targetChar) {
            // If character matches the target in the correct position, add to correctlyPlaced and
            // remove from unusedDigitsAndSymbols.
            correctlyPlaced.add(inputChar);
            unusedDigitsAndSymbols.remove(inputChar);
        } else {
            if (target.indexOf(inputChar) >= 0) {
                // If the character is present in the target but not at the correct position,
                // add it to presentButIncorrectlyPlaced set and remove from unusedDigitsAndSymbols
                presentButIncorrectlyPlaced.add(inputChar);
                unusedDigitsAndSymbols.remove(inputChar);
            } else {
                // If the character is not present in the target at all,
                // add it to missingFromEquation set
                missingFromEquation.add(inputChar);
            }
        }
    }

    // Ensure that no characters are intersected between correctlyPlaced and presentButIncorrectlyPlaced
    presentButIncorrectlyPlaced.removeIf(correctlyPlaced::contains);
```

```
    // Finally, ensure unusedDigitsAndSymbols does not contain any characters that have been placed or are missing
    unusedDigitsAndSymbols.removeAll(correctlyPlaced);
    unusedDigitsAndSymbols.removeAll(presentButIncorrectlyPlaced);
    unusedDigitsAndSymbols.removeAll(missingFromEquation);
}

/**
 * Evaluates if a mathematical expression is correct by comparing its left and right sides.
 *
 * @param expression The mathematical expression to evaluate.
 * @return true if the left side of the equation equals the right side; otherwise, false.
 */
2 usages
private boolean isMathematicallyCorrect(String expression) {
    try {
        ScriptEngine engine = new ScriptEngineManager().getEngineByName( shortName: "JavaScript");

        // Split the expression around the equals sign and evaluate both sides
        String[] parts = expression.split( regex: "\\s*=\\s*", limit: 2);
        return String.valueOf(engine.eval(parts[0])).equals(String.valueOf(engine.eval(parts[1])));
    } catch (ScriptException | ArrayIndexOutOfBoundsException e) {
        return false;
    }
}


/**
 * Generates a target equation either by selecting randomly from a file or using a default value.
 *
 * @return The target equation as a string.
 */
2 usages
private String generateTargetEquation() {
    final String fileName = "equations.txt";
    BufferedReader reader = null;
    String targetEquation = null;

    try {
        reader = new BufferedReader(new FileReader(fileName));
        // // Read all lines from the file and add them to the list after trimming white spaces
        String line;
        while ((line = reader.readLine()) != null) {
            equations.add(line.trim());
        }

        if (!equations.isEmpty()) {
            Random rand = new Random();
            targetEquation = equations.get(rand.nextInt(equations.size()));
        }
    } catch (IOException e) {
        System.err.println("Error reading file: " + fileName);
        e.printStackTrace();
    } finally {
        // Ensure the reader is closed properly
        try {
            if (reader != null) {
                reader.close();
            }
        } catch (IOException e) {
            System.err.println("Error closing file reader for: " + fileName);
            e.printStackTrace();
        }
    }
```

```
        // Return a default equation if no equations were loaded from the file
        return targetEquation != null  ? targetEquation : "3+4*1=7";
    }

    1 usage
    public boolean isShouldDisplayTarget() { return shouldDisplayTarget; }


    5 usages
    @Override
    public boolean isGameOver() { return remainingAttempts <= 0 || gameWon; }

    2 usages
    @Override
    public boolean isGameWon() { return gameWon; }

    3 usages
    @Override
    public String getTargetNumber() { return targetNumber; }

    1 usage
    @Override
    public StringBuilder getCurrentGuess() { return currentGuess; }

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

    2 usages
    @Override
    public void startNewGame() { initialize(); }
}
```

1.1.3 INumberleModel class

```java
package Numberle_Model;

public interface INumberleModel {
    9 usages
    int MAX_ATTEMPTS = 6;

    1 usage  1 implementation
    void initialize();
    2 usages  1 implementation
    boolean processInput(String input);
    5 usages  1 implementation
    boolean isGameOver();
    2 usages  1 implementation
    boolean isGameWon();
    3 usages  1 implementation
    String getTargetNumber();
    1 usage  1 implementation
    StringBuilder getCurrentGuess();
    3 usages  1 implementation
    int getRemainingAttempts();
    2 usages  1 implementation
    void startNewGame();
    1 usage  1 implementation
    public boolean isShouldDisplayTarget() ;
}
```

1.1.4 NumberleModelTest class

```
package Numberle_Model;

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.*;

class NumberleModelTest {
    13 usages
    private INumberleModel instance;

    @BeforeEach
    void setUp() {
        instance = new NumberleModel( shouldValidateEquation: true,  shouldDisplayTarget: true,  shouldRandomizeTarget: false);
    }

    @AfterEach
    void tearDown() {instance = null;}

    @Test
    /**
     * Tests the initialization of the NumberleModel to ensure it properly resets the game state.
     * This method checks that after initialization:
     * - The number of remaining attempts is set to the maximum defined by the model.
     * - The game is not in a "won" state.
     *
     * * @invariant invariant()
     *
     * @requires instance != null
     *   // Pre-condition: The instance of NumberleModel must be non-null and properly instantiated.
     *
     * @ensures instance.getRemainingAttempts() == INumberleModel.MAX_ATTEMPTS
     *   // Post-condition: After initialization, the remaining attempts should be reset to the maximum allowed attempts
     *
     * @ensures !instance.isGameWon()
     *   // Post-condition: After initialization, the game should not be in a won state.
     *
     */


     *
     */
    void testInitialize() {
        instance.initialize();
        assertEquals(INumberleModel.MAX_ATTEMPTS, instance.getRemainingAttempts());
        assertFalse(instance.isGameWon());
    }
```

```java
@Test
/**
 * This test checks the behavior of the NumberleModel when it processes various forms of invalid inputs.
 * The model should correctly reject these inputs and should not decrement the number of remaining attempts.
 *
 * @invariant \forall String input; model.isInputInvalid(input) => !model.processInput(input)
 * @invariant invariant()
 *
 * @requires model != null && model.isInitialized()
 *           // The model must be non-null and properly initialized before testing.
 *
 * @ensures \old(model.getRemainingAttempts()) == model.getRemainingAttempts()
 *          // Ensures that the number of attempts remain unchanged after processing invalid input.
 *
 */
void testProcessInvalidInput() {
    instance.initialize();
    String[] invalidInputs = {
            "1++2=3",
            "123asds",
            "1234567",
            "123*-67",
            "12+-=67",
            "1+2+3=1",
            "1/3+1=2"
    };

    // Test each invalid input and assert the expected outcomes
    for (String input : invalidInputs) {
        boolean result = instance.processInput(input);
        assertFalse(result, message: "Expected false for invalid input: " + input);
        assertEquals(INumberleModel.MAX_ATTEMPTS, instance.getRemainingAttempts(),
                message: "Attempt count should not decrease for invalid input");
    }
}
```

```java
    @Test
    /**
     * This test checks NumberleModel when it processes a series of valid inputs that should lead to a win.
     * The model should correctly accept these inputs and should recognize the win correctly.
     *
     * @invariant invariant()
     *
     * @invariant model.isGameWon() == (\exists String input; model.isWinningInput(input); model.processInput(input))
     *
     * @requires model != null && model.getTargetNumber() != null
     *           // The model must be non-null and properly initialized before testing.
     *
     * @ensures model.isGameWon()
     *          // Ensures that the game is won after the winning input is processed.
     *
     * @ensures \old(model.getRemainingAttempts()) == model.getRemainingAttempts()
     *          // Ensures that the number of attempts remain unchanged after processing winning input.
     */
    void testWinGame() {
        instance.initialize();
        String[] trialInputs = {
                "1+2*3=7", "1-1*1=0", "3+1*1=4", "3+1*4=7", "3+1-1=3"
        };
        String winningInput = "3+4*1=7";
        for (String input : trialInputs) {
            boolean result = instance.processInput(input);
            assertTrue(result, message: "Processing valid input should return true: " + input);
        }

        boolean winResult = instance.processInput(winningInput);
        assertTrue(winResult, message: "Processing winning input should return true");
        assertTrue(instance.isGameWon(), message: "Game should be won after correct guess");
        assertEquals(INumberleModel.MAX_ATTEMPTS, instance.getRemainingAttempts(),
                message: "Attempt count should remain the same when game is won");
    }
}
```

1.2 Numberle_View Package

1.2.1 NumberleView class

```
package Numberle_View;

import Numberle_Controller.NumberleController;

import Numberle_Model.INumberleModel;
import Numberle_Model.NumberleModel;

import javax.swing.*;
import javax.swing.border.Border;
import javax.swing.border.EmptyBorder;
import java.awt.*;
import java.util.*;

8 usages
public class NumberleView implements Observer {
    14 usages
    private INumberleModel model;
    18 usages
    private NumberleController controller;
    22 usages
    private final JFrame frame = new JFrame( title: "Numberle");
    4 usages
    private final StringBuilder input = new StringBuilder();
    12 usages
    private final JTextField[][] fields = new JTextField[INumberleModel.MAX_ATTEMPTS][7];

    7 usages
    private int lineNumber;
    13 usages
    private int currentPosition = 0;
    9 usages
    private HashMap<String, JButton> buttonMap = new HashMap<>();
    3 usages
    private boolean lastFlag1 = false;
    3 usages
    private boolean lastFlag2 = false;
    3 usages
    private boolean lastFlag3 = false;

2 usages
public NumberleView(INumberleModel model, NumberleController controller) {
    this.model = model;
    this.controller = controller;
    controller.startNewGame();
    ((NumberleModel) this.model).addObserver( o: this);
    this.controller.setView(this);
    update((NumberleModel) this.model,  arg: null);
    initializeFrame();
}
```

```java
public NumberleView(INumberleModel model, NumberleController controller,boolean lastFlag1,boolean lastFlag2,boolean lastFlag3) {
    this.model = model;
    this.controller = controller;
    controller.startNewGame();
    ((NumberleModel) this.model).addObserver( o: this);
    this.controller.setView(this);
    update((NumberleModel) this.model,  arg: null);
    this.lastFlag1=lastFlag1;
    this.lastFlag2=lastFlag2;
    this.lastFlag3=lastFlag3;
    initializeFrame();
}
```

```java
public NumberleView() {
    initializeFrame();
}

/**
 * Initializes the main frame of the application with all user interface components.
 */
3 usages
public void initializeFrame() {
    // Set default close operation and frame dimensions, and disable resizing
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize( width: 800,  height: 800);
    frame.setResizable(false);

    // Create top panel and set layout
    JPanel topPanel = new JPanel(new BorderLayout());
    String answer=null;
    if(controller.isShouldDisplayTarget()) {
        answer = "The anwser equation: " + controller.getTargetWord();
    }
    JLabel topLabel = new JLabel(answer);
    topLabel.setHorizontalAlignment(SwingConstants.CENTER);
    topPanel.add(topLabel, BorderLayout.CENTER);

    // Adding an icon to the top panel
    JLabel iconLabel = new JLabel(new ImageIcon( filename: "E:\\大四下\\AOOP\\cw1\\202018010111-coursework\\icon(1).png"));
    iconLabel.setHorizontalAlignment(JLabel.LEFT);
    iconLabel.setVerticalAlignment(JLabel.TOP);
    topPanel.add(iconLabel, BorderLayout.WEST);
    frame.add(topPanel, BorderLayout.NORTH);

    // Keyboard panel setup
    JPanel keyboardPanel = new JPanel();
    keyboardPanel.setLayout(new BorderLayout());
```

```
// Keyboard panel setup
JPanel keyboardPanel = new JPanel();
keyboardPanel.setLayout(new BorderLayout());

// Center panel setup for display
JPanel center = new JPanel();
center.setLayout(new BoxLayout(center, BoxLayout.X_AXIS));
center.add(new JPanel());
frame.add(center, BorderLayout.CENTER);

// Display panel configuration
JPanel displayPanel = new JPanel();
displayPanel.setLayout(new GridLayout( rows: 6,  cols: 7,  hgap: 5,  vgap: 5));
displayPanel.setBorder(BorderFactory.createEmptyBorder( top: 15,  left: 15,  bottom: 15, right: 15));

// Text field setup with custom fonts and borders
int textFieldColumns = 3;
Font boldFont = new Font( name: "Arial", Font.BOLD,  size: 18);
Border roundedBorder = BorderFactory.createLineBorder(Color.GRAY,  thickness: 2,  rounded: true);
Border marginBorder = new EmptyBorder( top: 5,  left: 5,  bottom: 5,  right: 5);
Border compoundBorder = BorderFactory.createCompoundBorder(roundedBorder, marginBorder);
```

```
for (int i = 0; i < INumberleModel.MAX_ATTEMPTS; i++) {
    for (int j = 0; j < 7; j++) {
        fields[i][j] = new JTextField();
        JTextField textField = new JTextField();
        fields[i][j].setFont(boldFont);
        textField.setColumns(textFieldColumns);
        fields[i][j].setBorder(compoundBorder);
        fields[i][j].setEditable(false);
        fields[i][j].setHorizontalAlignment(JTextField.CENTER);
        displayPanel.add(fields[i][j]);
    }
}
center.add(displayPanel);
center.add(new JPanel());
frame.add(center, BorderLayout.CENTER);

// Keyboard panel layout with numeric and operation keys
keyboardPanel.setLayout(new GridLayout( rows: 2, INumberleModel.MAX_ATTEMPTS,  hgap: 5,  vgap: 5));
keyboardPanel.setBorder(BorderFactory.createEmptyBorder( top: 0,  left: 10,  bottom: 10,  right: 10));

JPanel numberPanel = new JPanel(new GridLayout( rows: 1,  cols: 10,  hgap: 5,  vgap: 5));
String[] numberKeys = {"1", "2", "3", "4", "5", "6", "7", "8", "9", "0"};

for (String key : numberKeys) {
    JButton button = new JButton(key);
    buttonMap.put(key, button);
    button.setPreferredSize(new Dimension( width: 100,  height: 60));
    button.addActionListener(e -> {
        if (currentPosition < 7) {
            fields[lineNumber][currentPosition].setText(key);
            currentPosition++;
        }
    });
    numberPanel.add(button);
}
```

```
    JPanel operationPanel = new JPanel(new GridLayout( rows: 1,  cols: 7,  hgap: 5,  vgap: 5));
    String[] operationKeys = {"Back", "+", "-", "*", "/", "=", "Restart","Set", "Enter"};
    for (String key : operationKeys) {
        JButton button = new JButton(key);
        buttonMap.put(key, button);
        button.addActionListener(e -> {
            if (currentPosition <= 7) {
                switch (key) {
                    case "Back":
                        if (currentPosition > 0) {
                            fields[lineNumber][currentPosition - 1].setText("");
                            currentPosition--;
                        }
                        break;
                    case "Enter":
                        for (int i = 0; i < currentPosition; i++) {
                            input.append(fields[lineNumber][i].getText());
                        }

                        controller.processInput(input.toString());
                        input.setLength(0);


                        break;
                    case "+":
                    case "-":
                    case "*":
                    case "/":
                    case "=":
                        if (currentPosition < 7) {
                            fields[lineNumber][currentPosition].setText(key);
                            currentPosition++;
                        }
                }
            }
        });
    });
    operationPanel.add(button);
}
buttonMap.get("Restart").setEnabled(false);
buttonMap.get("Restart").addActionListener(e -> {
    restartGame();
});

frame.add(displayPanel, BorderLayout.CENTER);

keyboardPanel.add(numberPanel, BorderLayout.NORTH);
keyboardPanel.add(operationPanel, BorderLayout.CENTER);

frame.add(keyboardPanel, BorderLayout.SOUTH);

frame.setVisible(true);


// Action listener for settings button to handle game options
buttonMap.get("Set").addActionListener(e -> {
    Object[] options = {"Display errors", "Display target equation", "Random generate"};

    // Create a checkbox panel for user options
    JPanel panel = new JPanel();
    JCheckBox[] checkBoxes = new JCheckBox[options.length];

    // Loop through options to create checkboxes for each one
    for (int i = 0; i < options.length; i++) {
        checkBoxes[i] = new JCheckBox((String) options[i]);

        // Set the checkbox state based on previous user choices
        checkBoxes[i].setSelected(i == 0 ? lastFlag1 : (i == 1 ? lastFlag2 : lastFlag3));

        // Add each checkbox to the panel
        panel.add(checkBoxes[i]);
    }
```

```
        // Display a confirmation dialog with the checkbox panel
        int result = JOptionPane.showConfirmDialog( parentComponent: null, panel,
                title: "Select Options", JOptionPane.OK_CANCEL_OPTION, JOptionPane.PLAIN_MESSAGE);

        // Handle the user's selection from the dialog
        if (result == JOptionPane.OK_OPTION) {
            frame.dispose();

            // Retrieve the selection states from checkboxes and update lastFlag variables
            boolean flag1 = checkBoxes[0].isSelected();
            boolean flag2 = checkBoxes[1].isSelected();
            boolean flag3 = checkBoxes[2].isSelected();
            lastFlag1 = flag1;
            lastFlag2 = flag2;
            lastFlag3 = flag3;
            NumberleModel numberleModel = new NumberleModel(flag1, flag2, flag3);
            NumberleController controller = new NumberleController(numberleModel);
            controller.startNewGame();
            ((NumberleModel) this.model).addObserver( o: this);
            this.controller.setView(this);
            update((NumberleModel) this.model,  arg: null);
            new NumberleView(numberleModel,controller,flag1,flag2,flag3);
        }
    });
}
```

```
@Override
/**
 * Handles updates from the observable model, processing various types of messages and updating the UI accordingly.
 *
 * @param o The observable object.
 * @param arg The argument passed by the notifyObservers in Observable, expected to be either a String or ArrayList.
 */
public void update(Observable o, Object arg) {
    // Check if the update argument is a string, which is used for status messages
    if (arg instanceof String) {
        String message = (String) arg;

        // Handle different types of messages received from the Observable
        switch (message) {
            case "Invalid Input":
                JOptionPane.showMessageDialog(frame, message,  title: "Game Won", JOptionPane.INFORMATION_MESSAGE);
                currentPosition = input.length();
                break;
            case "Game Won":
                if (controller.isGameWon()) {
                    int option = JOptionPane.showConfirmDialog(frame,
                            message: "Congratulations! You won! " + controller.getTargetWord()
                                    + "\nDo you want to play again?",  title: "Game Over", JOptionPane.YES_NO_OPTION);
                    if (option == JOptionPane.YES_OPTION) {
                        restartGame();
                    } else {
                        disableInputComponents();
                        ((NumberleModel) this.model).deleteObserver( o: this);
                        System.out.println("User doesn't want to play again. Exiting the game.");
                    }
                } else {
                    int option = JOptionPane.showConfirmDialog(frame,  message: "Sorry, you lost. The word was "
                            + controller.getTargetWord()
                            + "\nDo you want to play again?",  title: "Game Over", JOptionPane.YES_NO_OPTION);

                    if (option == JOptionPane.YES_OPTION) {
                        restartGame();
```

```
            if (option == JOptionPane.YES_OPTION) {
                restartGame();
            } else {
                disableInputComponents();
                ((NumberleModel) this.model).deleteObserver( o: this);
                System.out.println("User doesn't want to play again. Exiting the game.");
            }
        }

        break;
    case "Game Over":
        if (controller.isGameWon()) {
            int option = JOptionPane.showConfirmDialog(frame, message: "Congratulations! You won! "
                    + controller.getTargetWord()
                    + "\nDo you want to play again?", title: "Game Over", JOptionPane.YES_NO_OPTION);
            if (option == JOptionPane.YES_OPTION) {
                restartGame();
            } else if(option == JOptionPane.NO_OPTION){
                disableInputComponents();
                // Delete observer
                ((NumberleModel) this.model).deleteObserver( o: this);

            }
        } else {
            int option = JOptionPane.showConfirmDialog(frame, message: "Sorry, you lost. The word was "
                    + controller.getTargetWord()
                    + "\nDo you want to play again?", title: "Game Over", JOptionPane.YES_NO_OPTION);

            if (option == JOptionPane.YES_OPTION) {
                restartGame();
            } else if(option == JOptionPane.NO_OPTION){
                disableInputComponents();
                ((NumberleModel) this.model).deleteObserver( o: this);
            }
        }
        lineNumber = INumberleModel.MAX_ATTEMPTS - controller.getRemainingAttempts();
        break;
```

```
    case "Try Again":
        JOptionPane.showMessageDialog(frame, message: message + ", Attempts remaining: "
                + controller.getRemainingAttempts(), title: "Try Again", JOptionPane.INFORMATION_MESSAGE);
        lineNumber = INumberleModel.MAX_ATTEMPTS - controller.getRemainingAttempts();
        break;
    case "Invalid Input: The equation must contain exactly one equals sign (=).":
        JOptionPane.showMessageDialog(frame,
                message: "Invalid Input: The equation must contain exactly one equals sign (=).",
                title: "Try Again", JOptionPane.INFORMATION_MESSAGE);
        break;
    case "Incorrect Equation: The left side is not equal to right side.":
        JOptionPane.showMessageDialog(frame,
                message: "Incorrect Equation: The left side is not equal to right side.",
                title: "Try Again", JOptionPane.INFORMATION_MESSAGE);
        break;
    case "Invalid Input: The equation must be 7 characters long.":
        JOptionPane.showMessageDialog(frame,
                message: "Invalid Input: The equation must be 7 characters long.",
                title: "Try Again", JOptionPane.INFORMATION_MESSAGE);
        break;
    case "Invalid Input: The equation must contain at least one operator (+, -, *, /).":
        JOptionPane.showMessageDialog(frame,
                message: "Invalid Input: The equation must contain at least one operator (+, -, *, /).",
                title: "Try Again", JOptionPane.INFORMATION_MESSAGE);
        break;
```

```
    } else if (arg instanceof ArrayList) {
        // Handle updates that include color feedback for the input fields
        ArrayList<String> list = (ArrayList) arg;
        for (int i = 0; i < list.size(); i++) {
            String s = list.get(i);
            updateFieldAndButtonColor(i, s);
        }
        currentPosition = 0;
        buttonMap.get("Restart").setEnabled(true);
    }


}
```

```
5 usages
private void restartGame() {
    frame.dispose();
    NumberleModel numberleModel = new NumberleModel( shouldValidateEquation: true,
            shouldDisplayTarget: true,  shouldRandomizeTarget: true); // Assuming default flags for new game
    NumberleController controller = new NumberleController(numberleModel);
    controller.startNewGame();
    ((NumberleModel) this.model).addObserver( o: this);
    this.controller.setView(this);
    update((NumberleModel) this.model,  arg: null);
    new NumberleView(numberleModel, controller);
    frame.repaint();
}

/**
 * Disables some components of the GUI
 */
4 usages
private void disableInputComponents() {
    // Disable all text fields
    for (int i = 0; i < INumberleModel.MAX_ATTEMPTS; i++) {
        for (int j = 0; j < 7; j++) {
            fields[i][j].setEnabled(false);
        }
    }
    // Disable all buttons except for "Restart" and "Set"
    for (String key : buttonMap.keySet()) {
        JButton button = buttonMap.get(key);
        if (!key.equals("Restart") && !key.equals("Set")) {
            button.setEnabled(false);
        }
    }
}
```

```java
    /**
     * Updates the color of the specified field and its corresponding button based on the game feedback.
     * This method changes the background and text color to visually indicate the state of the guess.
     *
     * @param index The index of the field and button to update.
     * @param colorName The name of the color to apply, which determines the feedback visual.
     */
    1 usage
    private void updateFieldAndButtonColor(int index, String colorName) {
        JTextField textField = fields[lineNumber][index];
        JButton button = buttonMap.get(textField.getText());

        switch (colorName) {
            case "Gray":
                setFieldAndButtonColor(textField, button, new Color( r: 175,  g: 175,  b: 175), Color.WHITE);
                break;
            case "Orange":
                setFieldAndButtonColor(textField, button, new Color( r: 245,  g: 164,  b: 71), Color.WHITE);
                break;
            case "Green":
                setFieldAndButtonColor(textField, button, new Color( r: 35,  g: 206,  b: 110), Color.WHITE);
                break;
            default:
                break;
        }
    }
```

```java
    /**
     * Sets the background and foreground colors for a given text field and its corresponding button.
     * This method ensures that the visual state of both the text field and the button reflects the game's feedback,
     * but prevents changes if the button is already set to a winning state (Green).
     *
     * @param textField The text field whose color needs to be updated.
     * @param button The button corresponding to the text field.
     * @param backgroundColor The new background color to set.
     * @param foregroundColor The new foreground color to set.
     */
    3 usages
    private void setFieldAndButtonColor(JTextField textField,JButton button,Color backgroundColor,Color foregroundColor){
        if (!button.getBackground().equals(new Color( r: 35,  g: 206,  b: 110))) {
            button.setBackground(backgroundColor);
            button.setForeground(foregroundColor);
        }
        textField.setBackground(backgroundColor);
        textField.setForeground(foregroundColor);
    }
}
```

1.3 Numberle_Controller Package

1.3.1 NumberleController class

```java
package Numberle_Controller;

import Numberle_Model.INumberleModel;
import Numberle_View.NumberleView;

// Controller.NumberleController.java
11 usages
public class NumberleController {
    9 usages
    private INumberleModel model;
    1 usage
    private NumberleView view;

    3 usages
    public NumberleController(INumberleModel model) { this.model = model; }

    4 usages
    public void setView(NumberleView view) { this.view = view; }

    1 usage
    public boolean processInput(String input) { return model.processInput(input); }

    1 usage
    public boolean isGameOver() { return model.isGameOver(); }

    2 usages
    public boolean isGameWon() { return model.isGameWon(); }

    5 usages
    public String getTargetWord() { return model.getTargetNumber(); }

    public StringBuilder getCurrentGuess() {
        return model.getCurrentGuess();
    }

    3 usages
    public int getRemainingAttempts() { return model.getRemainingAttempts(); }



    4 usages
    public void startNewGame() { model.startNewGame(); }
    1 usage
    public boolean isShouldDisplayTarget() {
        return model.isShouldDisplayTarget();
    }
}
```

1.4 CLIApp Class

```java
import Numberle_Model.INumberleModel;
import Numberle_Model.NumberleModel;

import java.util.Scanner;

public class CLIApp {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        boolean flag1 = false;
        boolean flag2 = false;
        boolean flag3 = false;
        boolean isValidInput = false;
        while (!isValidInput) {
            System.out.println("Enter 0 or 1 for each flag:");
            System.out.print("Validate Equation (0/1): ");
            int validateEquation = readFlag(scanner);
            System.out.print("Display Target (0/1): ");
            int displayTarget = readFlag(scanner);
            System.out.print("Randomize Target (0/1): ");
            int randomizeTarget = readFlag(scanner);

            if (validateEquation == 0 || validateEquation == 1 &&
                    displayTarget == 0 || displayTarget == 1 &&
                    randomizeTarget == 0 || randomizeTarget == 1) {

                flag1 = validateEquation == 0 ? false : true;
                flag2 = displayTarget == 0 ? false : true;
                flag3 = randomizeTarget == 0 ? false : true;
                isValidInput = true;
            } else {
                System.out.println("Invalid input. Please enter 0 or 1.");
            }
        }
        INumberleModel model = new NumberleModel(flag1, flag2, flag3);
```

```
model.startNewGame();
System.out.println("Welcome to Numberle - CLI Version. Guess the equation in 6 tries.");
System.out.println("You have " + model.getRemainingAttempts()
        + " attempts to guess the right equation. Good luck!");

while (!model.isGameOver()) {
    System.out.print("Enter your guess (7 characters long equation, e.g., '3+5*2=7'): ");
    String input = scanner.nextLine();


    model.processInput(input);

    // 根据游戏状态更新输出
    if (model.isGameOver()) {
        if (model.isGameWon()) {
            System.out.println("Congratulations! You win! You've guessed the equation correctly: "
                    + model.getTargetNumber());
        } else {
            System.out.println("Game Over! You lost! You've run out of attempts. The correct equation was: "
                    + model.getTargetNumber());
        }
        break;
    } else {
        System.out.println("Try again. You have " + model.getRemainingAttempts() + " attempts left.");
    }
}

scanner.close();
```

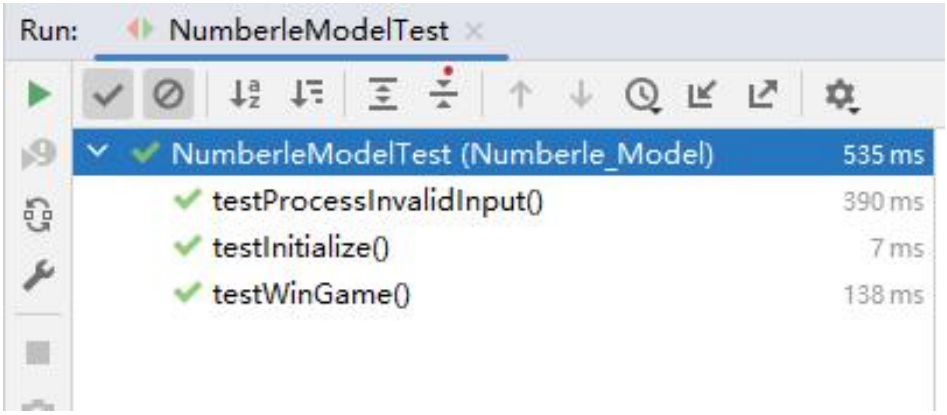
```
    3 usages
    private static int readFlag(Scanner scanner) {
        while (true) {
            try {
                int input = Integer.parseInt(scanner.nextLine().trim());
                if (input == 0 || input == 1) {
                    return input;
                } else {
                    System.out.println("Invalid input. Please enter 0 or 1.");
                }
            } catch (NumberFormatException e) {
                System.out.println("Invalid input. Please enter 0 or 1.");
            }
        }
    }
}
```

1.5 GUIApp Class

```java
import Numberle_Controller.NumberleController;
import Numberle_Model.INumberleModel;
import Numberle_Model.NumberleModel;
import Numberle_View.NumberleView;

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

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

    1 usage
    public static void createAndShowGUI() {
        INumberleModel model = new NumberleModel( shouldValidateEquation: true, shouldDisplayTarget: false, shouldRandomizeTarget: true);
        NumberleController controller = new NumberleController(model);
        while (!controller.isGameOver()) {
        }
        NumberleView view = new NumberleView(model, controller);
    }
}
```

Chapter 2 Testing

2.1 Testing Result



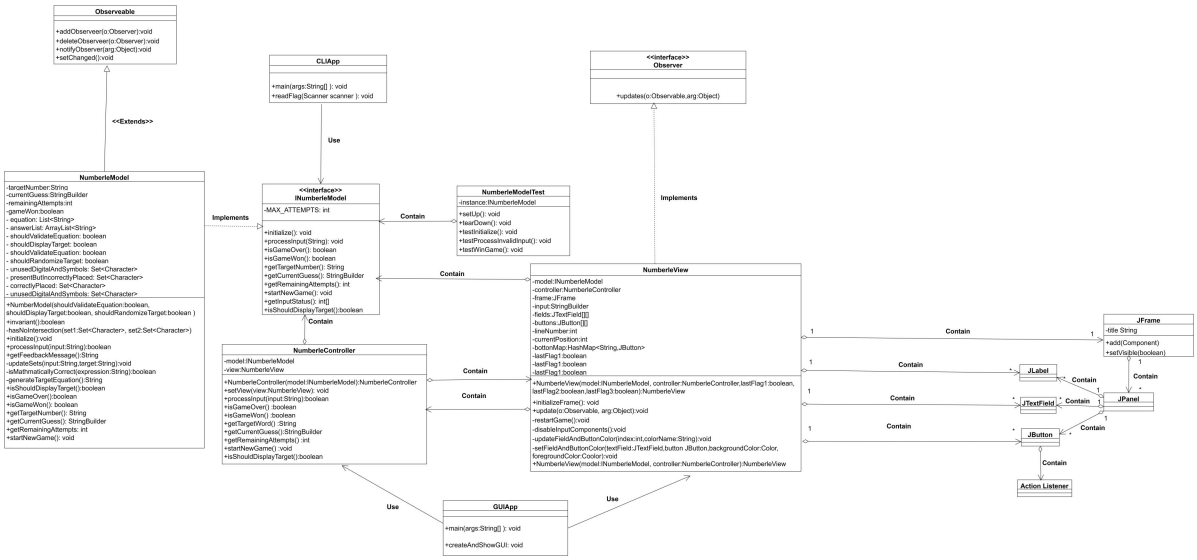
Tests passed: 3 of 3 tests – 535 ms

```
E:\java8\bin\java.exe ...
Target equation set to: 3+4*1=7
Invalid Input: The equation must be 7 characters long.
Invalid Input: The equation contains illegal characters.
Invalid Input: The equation must contain exactly one equals sign (=).
Invalid Input: The equation must contain exactly one equals sign (=).
Incorrect Equation: The left side is not equal to right side.
Incorrect Equation: The left side is not equal to right side.
Incorrect Equation: The left side is not equal to right side.
Target equation set to: 3+4*1=7
Target equation set to: 3+4*1=7
Orange! Green!Gray! Green!Orange! Green!Green!
Unused digits and symbols: [-, /, 0, 4, 5, 6, 8, 9]
Present but incorrectly placed: [1, 3]
Exist and correctly placed: [7, *, +, =]
Not in the equation: [2]

Orange! Gray! Orange! Green!Green!Green!Gray!
Unused digits and symbols: [/, 4, 5, 6, 8, 9]
Present but incorrectly placed: [3]
Exist and correctly placed: [1, 7, *, +, =]
Not in the equation: [0, 2, -]

Green!Green!Orange! Green!Green!Green!Orange!
Unused digits and symbols: [/, 5, 6, 8, 9]
Present but incorrectly placed: [4]
Exist and correctly placed: [1, 3, 7, *, +, =]
Not in the equation: [0, 2, -]

Green!Green!Orange! Green!Orange! Green!Green!
Unused digits and symbols: [/, 5, 6, 8, 9]
Present but incorrectly placed: [4]
Exist and correctly placed: [1, 3, 7, *, +, =]
Not in the equation: [0, 2, -]
```

```
Orange! Gray! Orange! Green!Green!Green!Gray!
Unused digits and symbols: [/, 4, 5, 6, 8, 9]
Present but incorrectly placed: [3]
Exist and correctly placed: [1, 7, *, +, =]
Not in the equation: [0, 2, -]

Green!Green!Orange! Green!Green!Green!Orange!
Unused digits and symbols: [/, 5, 6, 8, 9]
Present but incorrectly placed: [4]
Exist and correctly placed: [1, 3, 7, *, +, =]
Not in the equation: [0, 2, -]

Green!Green!Orange! Green!Orange! Green!Green!
Unused digits and symbols: [/, 5, 6, 8, 9]
Present but incorrectly placed: [4]
Exist and correctly placed: [1, 3, 7, *, +, =]
Not in the equation: [0, 2, -]

Green!Green!Orange! Gray! Green!Green!Orange!
Unused digits and symbols: [/, 5, 6, 8, 9]
Present but incorrectly placed: [4]
Exist and correctly placed: [1, 3, 7, *, +, =]
Not in the equation: [0, 2, -]

Green Green Green Green Green Green Green


Process finished with exit code 0
```

Chapter 3 Design of Class Diagram

3.1 Class Diagram



Chapter 4 GitHub

https://github.com/XimengLiang/Aoop_cw