Revision History

Date	Version	Description	Author
5th June	V 1.0	ProjectPhaseTwo	Sana Shamma

Software Construction Project – Phase Two

I. Specification, Design, and Implementation of the Client/Server

A. Puzzle Parser Class

PuzzleParser.java > PuzzleParser > parsePuzzleData(String)

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class PuzzleParser {

    // Private constructor to prevent instantiation
    private PuzzleParser() {
    }

    /*
     * @param filePath the path of the file containing the puzzle data
     * @return a 2D array of strings representing the parsed puzzle data
     * @throws IOException if an I/O error occurs while reading the file
     */
    public static String[][] parsePuzzleData(String filePath) throws IOException {
        // Create a BufferedReader to read the puzzle data from the file
        BufferedReader reader = new BufferedReader(new FileReader(filePath));

        // Determine the number of rows and columns in the file
        int numRows = countRows(filePath);
        int numCols = countCols(filePath);

        // Create a 2D array to store the puzzle data
        String[][] puzzle = new String[numRows][numCols];

        // Reset the reader to the beginning of the file
        reader.close();
        reader = new BufferedReader(new FileReader(filePath));

        // Loop through each line in the file and parse the puzzle data
        int row = 0;
        String line;
        while ((line = reader.readLine()) != null) {
            String[] cells = line.split(regex: " ");
            for (int col = 0; col < cells.length; col++) {
                puzzle[row][col] = cells[col];
            }
            row++;
        }

        // Close the reader and return the parsed puzzle data
        reader.close();
        return puzzle;
    }

```

Figure 1 – Parser Class Part1

PuzzleParser.java > PuzzleParser > countRows(String)

```java

    /**
     * @param filePath the path of the file to count the rows in
     * @return the number of rows in the file
     * @throws IOException if an I/O error occurs while reading the file
     */
    private static int countRows(String filePath) throws IOException {
        // Create a BufferedReader to read the puzzle data from the file
        BufferedReader reader = new BufferedReader(new FileReader(filePath));

        // Count the number of rows in the file
        int numRows = 0;
        while (reader.readLine() != null) {
            numRows++;
        }

        // Close the reader and return the number of rows
        reader.close();
        return numRows;
    }

    /**
     * @param filePath the path of the file to count the columns in
     * @return the number of columns in the file
     * @throws IOException if an I/O error occurs while reading the file
     */
    private static int countCols(String filePath) throws IOException {
        // Create a BufferedReader to read the puzzle data from the file
        BufferedReader reader = new BufferedReader(new FileReader(filePath));

        // Determine the number of columns in the file
        int numCols = 0;
        String line;
        while ((line = reader.readLine()) != null) {
            int count = line.split(regex: " ").length;
            if (count > numCols) {
                numCols = count;
            }
        }

        // Close the reader and return the number of columns
        reader.close();
        return numCols;
    }

}

```

Figure 2 – Parser Class Part2

Revision History

Date	Version	Description	Author
5th June	V 1.0	ProjectPhaseTwo	Sana Shamma

B. Server Class

```java
import java.io.IOException;
import java.io.OutputStream;
import java.net.InetSocketAddress;
// import java.util.Arrays;

import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;
import com.sun.net.httpserver.HttpServer;

public class PuzzleServer {
    // Method Signature:
    /**
     * Method name: main
     * Starts the puzzle server on a specific port.
     *
     * @param args command line arguments
     * @throws IOException if an I/O error occurs while starting the server
     */
    Run | Debug
    public static void main(String[] args) throws IOException {
        // Create an HTTP server on port 8000
        HttpServer server = HttpServer.create(new InetSocketAddress(port:8000), backlog:0);

        // Register handlers for each puzzle type
        server.createContext(path:"/SudokuBlankPuzzle", new PuzzleHandler(puzzleFile:"SudokuBlankPuzzle.txt"));
        server.createContext(path:"/SudokuSolvedPuzzle", new PuzzleHandler(puzzleFile:"SudokuSolvedPuzzle.txt"));
        server.createContext(path:"/HexSudokuBlankPuzzle", new PuzzleHandler(puzzleFile:"HexSudokuBlankPuzzle.txt"));
        server.createContext(path:"/HexSudokuSolvePuzzle", new PuzzleHandler(puzzleFile:"HexSudokuSolvePuzzle.txt"));

        // Start the server
        server.start();
        System.out.println(x:"Server started on port 8000");
```

Figure 3 – Server Class Part1

```java
    private static class PuzzleHandler implements HttpHandler {
        private String puzzleFile;
        /**
         * Method name: PuzzleHandler
         * Constructs a PuzzleHandler with the specified puzzle file.
         *
         * @param puzzleFile the path of the file containing the puzzle data
         */
        public PuzzleHandler(String puzzleFile) {
            this.puzzleFile = puzzleFile;
        }
        /**
         * Method name: handle
         * Handles an HTTP request and sends the puzzle data as the response.
         *
         * @param exchange the HTTP exchange object representing the request and response
         * @throws IOException if an I/O error occurs while handling the request
         */
        public void handle(HttpExchange exchange) throws IOException {
            // Parse the puzzle data from the file
            String[][] puzzle = PuzzleParser.parsePuzzleData(puzzleFile);
            // Encode the puzzle data as a string with a custom delimiter
            String puzzleString = encodePuzzleAsString(puzzle, delimiter:"; ");
            // Set the response headers
            exchange.sendResponseHeaders(rCode:200, puzzleString.length());
            // Get the output stream of the response
            OutputStream responseStream = exchange.getResponseBody();
            // Write the puzzle data to the output stream
            responseStream.write(puzzleString.getBytes());
            // Close the output stream
            responseStream.close();
        }
```

Figure 4 – Server Class Part2

Revision History

Date	Version	Description	Author
5th June	V 1.0	ProjectPhaseTwo	Sana Shamma

```java
        /**
         * Method name: encodePuzzleAsString
         * Encodes the puzzle data as a string with a custom delimiter.
         *
         * @param puzzle    the puzzle data as a 2D array of strings
         * @param delimiter the delimiter string used to separate puzzle elements
         * @return the encoded puzzle data as a string
         */
        private String encodePuzzleAsString(String[][] puzzle, String delimiter) {
            StringBuilder encodedPuzzle = new StringBuilder();
            for (int i = 0; i < puzzle.length; i++) {
                String[] row = puzzle[i];

                // Join the elements of the row with the delimiter
                encodedPuzzle.append(String.join(delimiter, row));

                // Add a newline character after each row except the last one
                if (i != puzzle.length - 1) {
                    encodedPuzzle.append(delimiter);
                }
                encodedPuzzle.append(str:"\n");
            }
            return encodedPuzzle.toString();
        }
    }
}
```

Figure 5 – Server Class Part3

Revision History

Date	Version	Description	Author
5th June	V 1.0	ProjectPhaseTwo	Sana Shamma

Properties of the code (Three Pillars):

Class	Principle	Line	Comment
PuzzleServer	Safe from bugs	19, 54	The code is well-structured and follows best practices for handling I/O operations and HTTP server implementation.
		6,7,8	The implementation follows established standards and best practices for HTTP server development using the com.sun.net.httpserver package, which is widely recognized for its reliability and safety.
	Easy to understand	34, 54, 82	The code is well-organized into separate methods and classes, making it modular and easy to comprehend.
		Throw all the code	Meaningful variable and method names are used to enhance code readability.
		Throw all the code	Comments have been added to describe the purpose and functionality of each method and class.
	Ready for change	24, 25, 26, 27	The code separates the logic for handling different puzzle types into distinct contexts, making it easy to add or modify puzzle types in the future.
		34	The use of the PuzzleHandler class as a handler for HTTP requests allows for encapsulation and flexibility in handling different puzzle data sources.
		54, 82	The code demonstrates separation of concerns by separating the puzzle parsing and encoding functionality into separate methods, which can be modified independently if needed.

Table 1 – PuzzleServer Class

Revision History

Date	Version	Description	Author
5th June	V 1.0	ProjectPhaseTwo	Sana Shamma

C. Client Class

1. PuzzleClinet Class

PuzzleServer.java | PuzzleClientv1.java X

PuzzleClientv1.java

```java
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;

public class PuzzleClientv1 {
    private static final String SUDOKU_PUZZLE_URL = "http://localhost:8000/SudokuBlankPuzzle";
    private static final String HEXSUDOKU_PUZZLE_URL = "http://localhost:8000/HexSudokuBlankPuzzle";

    private JFrame frame;
    private JPanel buttonPanel;
    private JPanel finishPanel;
    private JButton sudokuButton;
    private JButton hexSudokuButton;
    private JPanel puzzlePanel;
    private JTextField[][] puzzleFields;
    private PuzzleState puzzleState;
    private JFrame solutionFrame;

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new PuzzleClient().createAndShowGUI());
    }
```

Figure 1 – Client Class Part1

PuzzleClientv1.java > PuzzleClientv1 > createAndShowGUI()

```java
    /**
    Creates and shows the graphical user interface (GUI) for the Puzzle Client.
    Initializes the puzzle state, sets up the frame, buttons, panels, and event listeners.
    @param None
    @return None
    @throws None
    */
    private void createAndShowGUI() {
        puzzleState = new PuzzleState();
        frame = new JFrame(title:"Puzzle Client");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Create the buttons
        sudokuButton = new JButton(text:"Sudoku");
        hexSudokuButton = new JButton(text:"HexSudoku");

        // Add action listeners to the buttons
        sudokuButton.addActionListener(e -> fetchAndDisplayPuzzle(SUDOKU_PUZZLE_URL, puzzleSize:9));
        hexSudokuButton.addActionListener(e -> fetchAndDisplayPuzzle(HEXSUDOKU_PUZZLE_URL, puzzleSize:16));

        // Create the button panel and add the buttons
        buttonPanel = new JPanel();
        buttonPanel.add(sudokuButton);
        buttonPanel.add(hexSudokuButton);

        // Create the puzzle panel and puzzle fields
        puzzlePanel = new JPanel();
        puzzleFields = new JTextField[0][0];
```

Figure 2 – Client Class Part2

Revision History

Date	Version	Description	Author
5th June	V 1.0	ProjectPhaseTwo	Sana Shamma

PuzzleServer.java PuzzleClientv1.java 1 X

PuzzleClientv1.java > PuzzleClientv1

```java
/**
Saves the user input from the puzzle fields to a file.
Validates the input, updates the puzzle state, and writes the user input and puzzle state to a file.
@param None
@return None
@throws IOException if an error occurs while saving the user input
*/

private void saveUserInput() {
    String[][] userInput = new String[puzzleFields.length][puzzleFields[0].length];

    for (int i = 0; i < puzzleState.getPuzzle().length; i++) {
        for (int j = 0; j < puzzleState.getPuzzle()[0].length; j++) {
            String input = puzzleFields[i][j].getText();

            boolean isValidInput = isValidInput(input);
            if (!isValidInput) {
                String errorMessage = getErrorMessage(input);
                showErrorMessage(errorMessage);
                return;
            }

            userInput[i][j] = input;
            puzzleState.updateCell(i, j, input); // Update the puzzle state
        }
    }

```

Figure 3 – Client Class Part3

PuzzleServer.java PuzzleClientv1.java 1 X

PuzzleClientv1.java > PuzzleClientv1

```java
    String gameType = (puzzleFields.length == 9) ? "Sudoku" : "HexSudoku";
    String outputFilename = gameType + "Output.txt";
    File outputFile = new File(outputFilename);

    if (outputFile.exists()) {
        outputFile.delete();
    }

    try (BufferedWriter writer = new BufferedWriter(new FileWriter(outputFilename))) {
        for (String[] row : userInput) {
            writer.write(String.join(delimiter:" ", row));
            writer.newLine();
        }

        writer.write(str:"-----");
        writer.newLine();

        for (String[] row : puzzleState.getPuzzle()) {
            writer.write(String.join(delimiter:" ", row));
            writer.newLine();
        }

        showMessage(message:"User input saved successfully!");
    } catch (IOException e) {
        showErrorMessage(errorMessage:"An error occurred while saving the user input.");
    }
}
```

Figure 4 – Client Class Part4

Revision History

Date	Version	Description	Author
5th June	V 1.0	ProjectPhaseTwo	Sana Shamma

PuzzleServer.java | PuzzleClientv1.java 1 X

PuzzleClientv1.java > PuzzleClientv1

```java
    /**
    Checks if the given input is valid.
    @param input the input string to validate
    @return true if the input is valid, false otherwise
    @throws None
    */
    private boolean isValidInput(String input) {
        return input.matches(regex:"[1-9]|1[0-6]|-");
    }

    /**
    Gets the error message for the given input.
    @param input the input string
    @return the error message corresponding to the input
    @throws None
    */
    private String getErrorMessage(String input) {
        if (input.isEmpty()) {
            return "Input cannot be empty.";
        } else {
            return "Invalid input. Please enter a digit from 1 to 9 or A to G, or '-' for empty cells.";
        }
    }
```

Figure 5 – Client Class Part5

PuzzleServer.java | PuzzleClientv1.java 1 X

PuzzleClientv1.java > PuzzleClientv1

```java
    /**
    Displays an error message dialog box.
    @param errorMessage the error message to display
    @return None
    @throws None
    */
    private void showErrorMessage(String errorMessage) {
        JOptionPane.showMessageDialog(frame, errorMessage, title:"Invalid Input", JOptionPane.ERROR_MESSAGE);
    }

    /**
    Displays a message dialog box.
    @param message the message to display
    @return None
    @throws None
    */
    private void showMessage(String message) {
        JOptionPane.showMessageDialog(frame, message, title:"Message", JOptionPane.INFORMATION_MESSAGE);
    }
```

Figure 6 – Client Class Part6

Revision History

Date	Version	Description	Author
5th June	V 1.0	ProjectPhaseTwo	Sana Shamma

```java
    /**
    Fetches a puzzle from a specified URL and displays it on the UI.
    @param url the URL to fetch the puzzle from
    @param puzzleSize the size of the puzzle (e.g., 9 for Sudoku, 16 for HexSudoku)
    @return None
    @throws IOException if an error occurs while fetching the puzzle
    */
    private void fetchAndDisplayPuzzle(String url, int puzzleSize) {
        try {
            URL puzzleUrl = new URL(url);
            HttpURLConnection connection = (HttpURLConnection) puzzleUrl.openConnection();
            connection.setRequestMethod(method:"GET");

            int responseCode = connection.getResponseCode();
            if (responseCode == HttpURLConnection.HTTP_OK) {
                InputStreamReader InputStreamReader = new InputStreamReader(connection.getInputStream());
                BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
                String line;

                String[][] puzzle = new String[puzzleSize][puzzleSize];
                int row = 0;

                while ((line = bufferedReader.readLine()) != null) {
                    String[] values = line.split(regex:" ");
                    puzzle[row] = values;
                    row++;
                }

                bufferedReader.close();
                inputStreamReader.close();

                // Display the puzzle on the UI
                displayPuzzle(puzzle, puzzleSize);
            } else {
                showErrorMessage("Error fetching puzzle: " + responseCode);
            }
        } catch (IOException e) {
            showErrorMessage(errorMessage:"An error occurred while fetching the puzzle.");
        }
    }
```

Figure 7 – Client Class Part7

```java
    /**
    Displays the puzzle on the UI, allowing user interaction.
    @param puzzle The puzzle grid represented as a 2D array of strings.
    @param puzzleSize The size of the puzzle grid.
    @return None
    @throws None
    */
    private void displayPuzzle(String[][] puzzle, int puzzleSize) {
        puzzlePanel.removeAll();
        puzzlePanel.setLayout(new GridLayout(puzzleSize, puzzleSize));

        puzzleFields = new JTextField[puzzleSize][puzzleSize];
        for (int i = 0; i < puzzleSize; i++) {
            for (int j = 0; j < puzzleSize; j++) {
                String cellValue = puzzle[i][j];

                JTextField textField = new JTextField(cellValue, columns:2);
                textField.setHorizontalAlignment(JTextField.CENTER);
                textField.setEditable(b:true);

                puzzleFields[i][j] = textField;
                puzzlePanel.add(textField);
            }
        }
        finishPanel.setVisible(aFlag:true);

        frame.revalidate();
        frame.repaint();
    }
```

Figure 8 – Client Class Part8

Revision History

Date	Version	Description	Author
5th June	V 1.0	ProjectPhaseTwo	Sana Shamma

PuzzleServer.java PuzzleClientv1.java 1 X

PuzzleClientv1.java > PuzzleClientv1

```java
    /**
    Displays the solution of the puzzle on a separate frame.
    @return None
    @throws None
    */
    private void showSolution() {
        String[][] solution = puzzleState.getPuzzle();
        int puzzleSize = solution.length;

        solutionFrame = new JFrame(title:"Solution");
        solutionFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        solutionFrame.setLayout(new GridLayout(puzzleSize, puzzleSize));

        for (int i = 0; i < puzzleSize; i++) {
            for (int j = 0; j < puzzleSize; j++) {
                String cellValue = solution[i][j];

                JTextField textField = new JTextField(cellValue, columns:2);
                textField.setHorizontalAlignment(JTextField.CENTER);
                textField.setEditable(b:false);

                solutionFrame.add(textField);
            }
        }

        solutionFrame.setSize(width:400, height:400);
        solutionFrame.setVisible(b:true);
    }
}
```

Figure 9 – Client Class Part9

Puzzle Game	Version: 1.0
Sudoku & Hex Sudoku	Issue Date: 5th June 2023
Document Identifier: 1	Author: GroupD

2. IPuzzleState Class

```java
interface IPuzzleState {
    String[][] getPuzzle();
    void setPuzzle(String[][] puzzle);
    String getPuzzleData();
    void updateCell(int row, int column, String value);
}
```

Figure 1 – IPuzzleState Class Part1

3. PuzzleState Class

```java
class PuzzleState implements IPuzzleState{
    private String[][] puzzle;
    private StringBuilder puzzleData;

    public PuzzleState() {
        puzzle = new String[0][0];
        puzzleData = new StringBuilder();
    }

    public String[][] getPuzzle() {
        return puzzle;
    }

    public void setPuzzle(String[][] puzzle) {
        this.puzzle = puzzle;
        updatePuzzleData();
    }

    public String getPuzzleData() {
        return puzzleData.toString();
    }

    public void updateCell(int row, int column, String value) {
        puzzle[row][column] = value;
        updatePuzzleData();
    }

    private void updatePuzzleData() {
        puzzleData.setLength(newLength:0);
        for (String[] row : puzzle) {
            for (String value : row) {
                puzzleData.append(value).append(str:" ");
            }
            puzzleData.append(str:"\n");
        }
    }
```

Figure 1 – PuzzleState Class Part1

Puzzle Game	Version: 1.0
Sudoku & Hex Sudoku	Issue Date: 5th June 2023
Document Identifier: 1	Author: GroupD

Properties of the code (Three Pillars):

Class's Name	Line	Comment
PuzzleClientv1	33,86, and 258 17, 86, and 184 Throw all the code	**Easy to understand:** **Proper organization:** The code is organized into methods, making it easier to follow the flow of execution. Each method has a specific purpose, such as creating the GUI, handling user input, fetching puzzles, and displaying solutions. **Descriptive variable and method names:** The variable and method names are mostly descriptive, making it easier to understand their purpose and functionality. For example, puzzleFields, saveUserInput(), fetchAndDisplayPuzzle() **Comments:** There are comment in the code, which can make it easier for others (or even yourself) to understand the code's intent, especially in complex sections
	32,85,138, 148,162,172, 183,224,253 [8 – 19] 20 8 & 9 127	**Ready to change:** The code is structured into different methods, which helps separate concerns and improve code modularity. This approach makes it ready to change. Code with global variable is less ready for change because a change needs to be made in many places. By making variables private, it easier to change the implementation of the class without affecting the users of the class or breaking their code. Constructors can provide flexibility by allowing users of the class to specify different values for the object's attributes when creating a new instance of the class. The final keyword can help make the code ready to change by providing a clear and explicit specification of the intended behaviour of variables. This can make it easier to modify the code without introducing unexpected behavior or breaking existing code. A meaningful error message can make it easier to modify the code in the future by providing a clear and consistent specification of the intended behavior. It can also make it easier to debug errors that may occur during modifications
	20 [8 - 9]	**Free from bug:** Adding a constructor to a class can make the code safer from bugs by ensuring that objects are always initialized in a valid state. Since '**SUDOKU_PUZZLE_URL' and ''HEXSUDOKU_PUZZLE_URL'** parameters do not change within the method, we add final keyword to indicate that they are not mutable. This can prevent accidental changes to these variables.

Table 2 – PuzzleClient Class

Puzzle Game	Version: 1.0
Sudoku & Hex Sudoku	Issue Date: 5th June 2023
Document Identifier: 1	Author: GroupD

II. Testing of The Client/Server

A. Parser Class

```
src > main > java > phase2 > PuzzleParser.java > PuzzleParser > parsePuzzleData(String)
14    // Partition: Parsing puzzle data from a file
15    // Subdomain:
16    // 1. Valid file path with correct puzzle data
17    // 2. Valid file path with empty puzzle data
18    // 3. Invalid file path
19    // Test Cases:
20    // 1. Valid file path with 9rows and 9 columns puzzle data-->return expected puzzle (cover Subdomain 1 )
21    // 2. Valid empty file path with 0 rows and 0 columns-->return 0 (cover Subdomain 2 )
22    // 3. Invalid file path with a spelling mistake--> IOException (cover Subdomain 3 )
23    public static String[][] parsePuzzleData(String filePath) throws IOException {···
53
54    // Partition: Counting the number of rows in a file
55    // Subdomain:
56    // 1. Empty file
57    // 2. File with a single row
58    // 3. File with multiple rows
59    // Test Cases:
60    // 1. Empty file (0 rows)-->return 0 (cover Subdomain 1 )
61    // 2. File with a single row and Five columns-->return 1 (cover Subdomain 2 )
62    // 3. File with Seven rows and one columns-->return 7 (cover Subdomain 3 )
63    static int countRows(String filePath) throws IOException {···
77
```

Figure 1 – Parser Class Part1

```
78    // Partition: Counting the number of columns in a file
79    // Subdomain:
80    // 1. Empty file
81    // 2. File with a single row and multiple columns
82    // 3. File with multiple rows and one columns
83    // Test Cases:
84    // 1. Empty file (0 columns)-->return 0 (cover Subdomain 1 )
85    // 2. File with a single row and Five columns-->return 5 (cover Subdomain 2 )
86    // 3. File with Seven rows and one columns-->return 1 (cover Subdomain 3 )
87    static int countCols(String filePath) throws IOException {···
```

Figure 2 – Parser Class Part2

Puzzle Game	Version: 1.0
Sudoku & Hex Sudoku	Issue Date: 5th June 2023
Document Identifier: 1	Author: GroupD

B. Client Part

C. Server Class