Your Own Wardrobe - Development

Development Techniques used:

- Object-oriented programming - Java
- GUI building - JButton, JComboBox, JTextField
- Sorting
- Parameter passing
- ArrayLists
- Filtering
- Data Validation
- Exception Handling
- Reading images as byte arrays
- MySQL Database
- Database queries

The product is a digital wardrobe built as a Java program connected to a MYSQL database. Its primary functions are:

1. the addition and deletion of items from the database
2. viewing the wardrobe
3. receiving outfit suggestions based upon the user's interpretation of the weather outside

Program's structure:

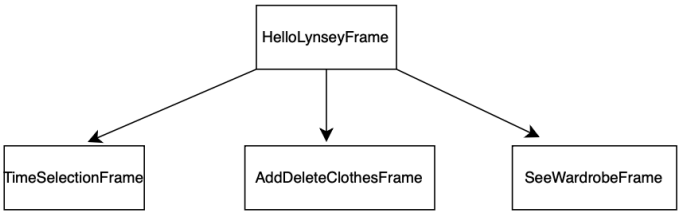

Fig 1. Program's basic format

Overall, the program consists of 8 java classes corresponding to user interfaces built in Java and Swing and out of which 4 of them are directly operating with the database through queries.

Object Oriented Programming

HelloLynseyFrame - Overall structure of a GUI

The *HelloLynseyFrame* class serves as a menu for the user upon deciding on whether they want to see their wardrobe, add or delete clothes to it or receive an outfit suggestion for the day. The main purpose of this class is to establish the general connection within the 3 main features of the program, giving the user a comprehensive representation of them.

```
        // Create "Create Outfit" button
        JButton createOutfitButton = new JButton("Suggest Outfit");
        createOutfitButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Open the TimeSelectionFrame when "Create Outfit" button is clicked
                new TimeSelectionFrame(connection);
            }
        });
        buttonPanel.add(createOutfitButton);

        // Create "Edit Wardrobe" button
        JButton editWardrobeButton = new JButton("Add/Delete Clothes");
        editWardrobeButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Open the AddDeleteClothesFrame when "Edit Wardrobe" button is clicked
                new AddDeleteClothesFrame(connection);
            }
        });
        buttonPanel.add(editWardrobeButton);

        // Create "See Wardrobe" button
        JButton seeWardrobeButton = new JButton("Show Wardrobe");
        seeWardrobeButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Open the WardrobeFrame when "See Wardrobe" button is clicked
                new SeeWardrobeFrame(connection);
            }
        });
        buttonPanel.add(seeWardrobeButton);
```

Fig 2. HelloLynseyFrame.java - Addition and connection to the other classes (Appendix#C)

This structure is used further in the AddDeleteClothesFrame.java which would provide the user with 2 buttons (one to add and one to delete clothes). By adding an ActionListener to each button, the frame of the specific button will be displayed by creating a new instance of it and passing the 'connection' object to it.

Adding Clothes - Functionality

The ImageSelectionFrame.java class opens when the user chooses to add clothes within the database. Compared to the previous interface, the ImageSelectionFrame has direct access to the database through the setConnection() method, using the INSERT query to add new items to the database.

```
JFileChooser fileChooser = new JFileChooser(); // Moved fileChooser declaration here

browseButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        FileNameExtensionFilter filter = new FileNameExtensionFilter("JPEG Images", "jpg", "jpeg");
        fileChooser.setFileFilter(filter);
        int returnVal = fileChooser.showOpenDialog(ImageSelectionFrame.this);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            selectedFile = fileChooser.getSelectedFile(); // Assigning the selected file to selectedFile
            ImageIcon icon = new ImageIcon(selectedFile.getPath());
            Image img = icon.getImage();
            Image newImg = img.getScaledInstance(100, 100, Image.SCALE_SMOOTH);
            ImageIcon newIcon = new ImageIcon(newImg);
            imageLabel.setIcon(newIcon);
        }
    }
});
```

Fig 3. ImageSelectionFrame.java - browseButton functionality and filters-Appendix#C

By initialising the fileChooser and using it within the ActionListener of the browseButton, the dialog would provide access to browse and select files from the user's file system.

Inside the `actionPerformed` method, the filter is added through the `FileNameExtensionFilter` shown in the file chooser, accepting only images with .jpg and .jpeg extensions to be added to the database. If the user selects a file and clicks "Open" the `showOpenDialog()` method would return `JFileChooser.APPROVE_OPTION`, indicating that the selection was successful and therefore displaying it on the interface through the getter method `fileChooser.getSelectedFile()`.

```
JButton uploadButton = new JButton("Upload");
uploadButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Upload button clicked");
        String weather = (String) weatherComboBox.getSelectedItem();
        String preferenceText = preferenceField.getText();
        if (!preferenceText.isEmpty()) { // Checking if preference rate field is not empty
            int preference = Integer.parseInt(preferenceText);
            if (selectedFile != null && isValidPreference(preference)) { // Checking if a file is selected and preference is valid
                try {
                    // Read the selected image file
                    FileInputStream fis = new FileInputStream(selectedFile);
                    byte[] imageData = new byte[(int) selectedFile.length()];
                    fis.read(imageData);
                    fis.close();

                    // Insert the image into the database
                    String query = "INSERT INTO clothes (image, weather_type, preference_rate) VALUES (?, ?, ?)";
                    PreparedStatement statement = connection.prepareStatement(query);
                    statement.setBytes(1, imageData);
                    statement.setString(2, weather);
                    statement.setInt(3, preference);
                    statement.executeUpdate();

                    JOptionPane.showMessageDialog(ImageSelectionFrame.this, "Image uploaded successfully!");
                } catch (IOException | SQLException ex) {
                    ex.printStackTrace();
                    JOptionPane.showMessageDialog(ImageSelectionFrame.this, "Failed to upload image.");
                }
            } else {
                JOptionPane.showMessageDialog(ImageSelectionFrame.this, "Please select an image and enter a preference rate between 1 and 5.");
            }
        } else {
            JOptionPane.showMessageDialog(ImageSelectionFrame.this, "Please enter a preference rate.");
        }
    }
});
buttonPanel.add(uploadButton);
```

Fig 4. ImageSelectionFrame.java - uploadButton functionality -Appendix#C

The upload function of the ImageSelectionFrame allows the user to add the item together with its weather type and preference rate to the database.

As the button is pressed the "Upload button clicked" is printed to the console therefore retrieving the:

- weather_type from the `weatherComboBox` as a String through: `String weather = (String) weatherComboBox.getSelectedItem()`
- the preference_rate from the `preferenceField` through: `String preferenceText = preferenceField.getText()`

- the image file which is read as a byte array(MIT, 2024) from the `selectedFile` using: `byte[] imageData = new byte[(int) selectedFile.length()]` and a `FileInputStream;`

Following the retrievals, the `INSERT` query adds the image data, weather type and preference rate into the `clothes` database, using the three inputs as parameters within the statement. The statement is therefore executed through the `statement.executeUpdate()` which, if successful, would display "Image uploaded successfully!".

Adding Clothes - Exception Handling

```
                JOptionPane.showMessageDialog(ImageSelectionFrame.this, "Image uploaded successfully!");
            } catch (IOException | SQLException ex) {
                ex.printStackTrace();
                JOptionPane.showMessageDialog(ImageSelectionFrame.this, "Failed to upload image.");
            }
        } else {
            JOptionPane.showMessageDialog(ImageSelectionFrame.this, "Please select an image and enter a preference rate between 1 and 5.");
        }
    } else {
        JOptionPane.showMessageDialog(ImageSelectionFrame.this, "Please enter a preference rate.");
    }
}
});
```

Fig 5. ImageSelectionFrame.java - Exception Handling (Appendix#C)

In the case of an error during the upload process (`IOException or SQLException`) there will be a stack trace of the exception printer within the console and the following error messages will be displayed:

- “Failed to upload image” -when the image cannot be uploaded successfully,
- “Please select an image and enter a preference rate between 1 and 5” - when both the image and the preference rate fields are empty
- “Please enter a preference rate” - if only the preference rate field is empty

The weather type does not have an exception handle and it will just retrieve the default “Hot” from the JComboBox.

Adding Clothes - Validation

```
        private void validateInput() {
            String text = preferenceField.getText();
            try {
                int value = Integer.parseInt(text);
                if (value < 1 || value > 5) {
                    // Show error message and clear the field
                    JOptionPane.showMessageDialog(ImageSelectionFrame.this, "Preference rate must be between 1 and 5.");
                    preferenceField.setText("");
                }
            } catch (NumberFormatException ex) {
                // Show error message and clear the field
                JOptionPane.showMessageDialog(ImageSelectionFrame.this, "Preference rate must be a number between 1 and 5.");
                preferenceField.setText("");
            }
        }
    });
}

private boolean isValidPreference(int preference) {
    return preference >= 1 && preference <= 5;
}
```

Fig 6. ImageSelectionFrame.java - Validation process -Appendix#C

The validation method checks the input entered in the preferenceField. By retrieving the String from the `preferenceField`, it is then parsed into an integer using `Integer.parseInt(text)`. If the parse is successful, it checks whether the integer is between 1 and 5. If the value is not within the range, it displays the error message "Preference rate must be between 1 and 5." emptying the field. If the parsing failed due to a `NumberFormatException` meaning the input was not a number, it will therefore display "Preference rate must be a *number* between 1 and 5.", emptying the field.

Adding Clothes - Database Connectivity

```
public static void main(String[] args) {
    // Establish connection to the database
    Connection connection = null;
    try {
        // Set your database URL, username, and password
        String url = "jdbc:mysql://localhost:3306/clothing_store";
        String username = "root";
        String password = "pcCervantes12345";
        // Establish connection
        connection = DriverManager.getConnection(url, username, password);
    } catch (SQLException e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(null, "Failed to connect to the database.");
    }

    // Create an instance of ImageSelectionFrame
    if (connection != null) {
        new ImageSelectionFrame(connection);
    }
}
```

Fig 7. ImageSelectionFrame.java - Database connectivity - Appendix#C

The ImageSelectionFrame has a direct connection to the MySQL database which is therefore established with the specified URL, username, and password. If the connection is successful, the connection object is initialised with the connection, however, if it catches an SQLException, it prints

the stack trace and displays the error message “Failed to connect to the database.” using `JOptionPane.showMessageDialog`.

Delete Clothes - Functionality

The DeleteClothesFrame.java displays a graphical user interface used to delete items from the database.

```
try {
    Statement statement = connection.createStatement();
    ResultSet resultSet = statement.executeQuery("SELECT id, image FROM clothes");
    while (resultSet.next()) {
        int id = resultSet.getInt("id");
        byte[] imageData = resultSet.getBytes("image");
        if (imageData != null) { // Add null check
            ImageIcon imageIcon = new ImageIcon(imageData);
            Image scaledImage = imageIcon.getImage().getScaledInstance(100, 100, Image.SCALE_SMOOTH);
            ImageIcon scaledIcon = new ImageIcon(scaledImage);
            JButton imageButton = new JButton(scaledIcon);
            imageButton.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    int confirm = JOptionPane.showConfirmDialog(null, "Are you sure you want to delete this item?", "Confirm Deletion", JOptionPane.YES_NO_OPTION);
                    if (confirm == JOptionPane.YES_OPTION) {
                        try {
                            PreparedStatement deleteStatement = connection.prepareStatement("DELETE FROM clothes WHERE id = ?");
                            deleteStatement.setInt(1, id);
                            int rowsAffected = deleteStatement.executeUpdate();
                            if (rowsAffected > 0) {
                                JOptionPane.showMessageDialog(null, "Clothing item deleted successfully.");
                                dispose();
                                new DeleteClothesFrame(connection);
                            } else {
                                JOptionPane.showMessageDialog(null, "Failed to delete clothing item.");
                            }
                        } catch (SQLException ex) {
                            ex.printStackTrace();
                            JOptionPane.showMessageDialog(null, "Error deleting clothing item.");
                        }
                    }
                }
            });
            clothesPanel.add(imageButton);
        }
```

Fig 8. DeleteClothesFrame.java - Overall action - Appendix#C

The try-catch block inside of the constructor creates a statement using the connection to execute a SELECT query. It uses a while loop to iterate through the table and extract the id and image for each row, creating a JButton with an ImageIcon with each item’s image in the case where `image!=null`. It further adds an ActionListener to the JButton to handle the delete action.

```
public void actionPerformed(ActionEvent e) {
    int confirm = JOptionPane.showConfirmDialog(null, "Are you sure you want to delete this item?", "Confirm Deletion", JOptionPane.YES_NO_OPTION);
    if (confirm == JOptionPane.YES_OPTION) {
        try {
            PreparedStatement deleteStatement = connection.prepareStatement("DELETE FROM clothes WHERE id = ?");
            deleteStatement.setInt(1, id);
            int rowsAffected = deleteStatement.executeUpdate();
```

Fig 9. DeleteClothesFrame.java - Delete action - Appendix#C

When a Jbutton representing a clothing item is clicked, it requests a confirmation message dialog from the user before officially deleting the item.

After the user confirms the deletion, it performs a DELETE query, removing the clothing item from the database based upon its id. If the deletion is successful, "Clothing item deleted successfully." will be displayed further creating a new instance of the `DeleteClothesFrame` with the updated data, else it will display an error message.

SeeWardrobe - Functionality

The SeeWardrobeFrame represents a graphical user interface window which displays and edits the clothes stored within the database.

```
List<Clothes> clothesList = new ArrayList<>();
try {
    PreparedStatement statement = connection.prepareStatement("SELECT image, preference_rate, worn FROM clothes");
    ResultSet resultSet = statement.executeQuery();
    while (resultSet.next()) {
        byte[] imageData = resultSet.getBytes("image");
        if (imageData != null) {
            ImageIcon imageIcon = new ImageIcon(imageData);
            Image scaledImage = imageIcon.getImage().getScaledInstance(100, 100, Image.SCALE_SMOOTH);
            ImageIcon scaledIcon = new ImageIcon(scaledImage);
            int preferenceRate = resultSet.getInt("preference_rate");
            boolean worn = resultSet.getBoolean("worn"); // Retrieve worn status from database
            clothesList.add(new Clothes(scaledIcon, preferenceRate, worn, imageData));
        }
    }
```

Fig 10. SeeWardrobeFrame.java - Retrieval from database- Appendix#C

The try-catch block retrieves the data from the database, constructing Clothes objects to represent each item of clothing in a process similar to the one from DeleteClothesFrame.

SeeWardrobe - Toggle-like functionality of worn button

```
// Add action listener to handle button click
wornButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Toggle worn state when button is clicked
        clothes.setWorn(wornButton.isSelected());
        try {
            // Update the database to mark the item as worn or not worn
            PreparedStatement updateStatement = connection.prepareStatement("UPDATE clothes SET worn = ? WHERE image = ?");
            updateStatement.setBoolean(1, clothes.isWorn());
            updateStatement.setBytes(2, clothes.getImageData());
            updateStatement.executeUpdate();
        } catch (SQLException ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(SeeWardrobeFrame.this, "Failed to update wardrobe.");
        }

        // Change background color based on button state
        if (wornButton.isSelected()) {
            wornButton.setBackground(Color.RED);
        } else {
            wornButton.setBackground(null);
        }
    }
});
```

Fig 11. SeeWardrobeFrame.java - wornButton functionality- Appendix#C

As the actionPerformed method is invoked, the worn state of the clothes object based upon the wornButton status will act as a toggle. This will further use the UPDATE query in order to change the boolean value of the item's 'worn' column therefore changing the wornButton's colour based upon its selection state visually indicating whether the clothes are worn or not.

SeeWardrobe - Sorting process

In order to sort the clothesList based on the preference rate of each clothing item, this line has been used:

```
// Sort clothesList based on preference rate
clothesList.sort((c1, c2) -> c2.getPreferenceRate() - c1.getPreferenceRate());
```

Fig 12. SeeWardrobeFrame.java - Sorting algorithm- Appendix#C

The `clothesList.sort(...)` invokes the sort method on the list to reorder the elements of the list based on the order determined through comparison.

`(c1, c2) -> c2.getPreferenceRate()-c1.getPreferenceRate()` is a comparator interface used by the sorting algorithm to determine the lists' order.

`c2.getPreferenceRate() - c1.getPreferenceRate()` expression calculates the difference between the preference rates of the `c1` and `c2` parameters which represent two items from the list therefore determine the order of elements in the sorted list. (Nero, 2019)

Outfit suggestion - TimeSelection Frame - Functionality

The `TimeSelectionFrame`, allows users to select the current weather condition from a set of options.

```
// Create label with "What is the weather outside?"
JLabel label = new JLabel("What is the weather outside?", SwingConstants.CENTER);
label.setFont(new Font("Arial", Font.PLAIN, 16));

// Add label to JFrame
add(label, BorderLayout.NORTH);

// Create panel for weather selection
JPanel weatherPanel = new JPanel(new GridLayout(3, 2, 10, 10));

// Create weather options
String[] weatherOptions = {"Hot", "Warm", "Okay-ish", "Cold", "Freezing"};

// Create weather radio buttons
ButtonGroup weatherGroup = new ButtonGroup();
for (String weather : weatherOptions) {
    JRadioButton weatherButton = new JRadioButton(weather);
    weatherGroup.add(weatherButton);
    weatherPanel.add(weatherButton);
}

// Add weather selection panel to JFrame
add(weatherPanel, BorderLayout.CENTER);
```

Fig 13. SelectTimeFrame.java - functionality - Appendix#C

The radio buttons represent the weather options (“Hot”, “Warm”, “Okay-ish”, “Cold”, “Freezing”) which, as a `for` loop iterates over each option, a JRadioButton named weatherButton is created for each weatherOptions array.

Outfit suggestion - OutfitSuggestionFrame - Functionality

This OutfitSuggestionFrame class creates a JFrame to display outfit suggestions based on the selected weather condition from the TimeSelectionFrame.

```java
public OutfitSuggestionFrame(Connection connection, String selectedWeather) {
    this.connection = connection;

    setTitle("Outfit Suggestions for " + selectedWeather);
    setSize(600, 400);
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

    // Create a panel to display outfit suggestions
    JPanel outfitPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 10));

    try {
        // Query the database to retrieve outfit suggestions based on selected weather,
        // sorted by preference rate from highest to lowest and then by worn status
        PreparedStatement statement = connection.prepareStatement("SELECT image, preference_rate, worn FROM clothes WHERE weather_type = ? ORDER BY preference_rate DESC, worn ASC");
        statement.setString(1, selectedWeather);
        ResultSet resultSet = statement.executeQuery();
```

Fig 14. OutfitSuggestionFrame.java - constructor and database extraction- Appendix#C

The constructor takes two parameters: Connection connection for the database connection and String selectedWeather for the selected weather condition. Within the try-catch block, the SELECT query is used to retrieve the `image`, `preference_rate`, and `worn` columns from the `clothes` table, filtering by the selected weather condition, ordering the items from the highest to the lowest preference rate(descending) and their worn status(ascending). As it iterates through the results set, each item will have its preference level and image shown as well as a JLabel(wornLabel) with a red flag emoji to indicate whether the clothing item is worn.

The worn label is given through the if statement:

```java
            // Add "🚩worn" label if the clothing item is selected as worn
            if (worn) {
                outfitItemPanel.add(wornLabel, BorderLayout.NORTH);
            }

            outfitPanel.add(outfitItemPanel);
        }
```

Fig 15. OutfitSuggestionFrame.java - worn JLabel if statement- Appendix#C

Therefore, this class would provide a graphical interface to view outfit suggestions based on the selected weather condition fetched from the database, displaying outfit images, preference rates, and worn status (indicated by a red flag emoji) in a scrollable panel.

MYSQL Database

Data Definition Language for the clothes table in the clothing_store database

DDL for clothing_store.clothes

```
CREATE TABLE `clothes` (
  `id` int NOT NULL AUTO_INCREMENT,
  `image` mediumblob,
  `weather_type` varchar(20) DEFAULT NULL,
  `preference_rate` int DEFAULT NULL,
  `worn` tinyint(1) DEFAULT '0',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=58 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Fig 16. MySQL Database clothes table DDL

Database.java - Connecting the database to the programme

```
package GUIS;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class Database {
    private static final String JDBC_URL = "jdbc:mysql://localhost:3306/clothing_store";
    private static final String USERNAME = "root";
    private static final String PASSWORD = "pcCervantes12345";

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);
    }

    public static void insertData(Connection connection, String imagePath, String weatherType, int
        String sql = "INSERT INTO your_table (image_path, weather_type, preference_level) VALUES (
        try (PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setString(1, imagePath);
            statement.setString(2, weatherType);
            statement.setInt(3, preferenceLevel);
            statement.executeUpdate();
        }
    }

    public static String getJdbcUrl() {
        return JDBC_URL;
    }
}
```

Fig 17. Database.java

This class is used to encapsulate the operations related to the database such as establishing the connections between the interfaces and the database itself and further inserting, removing or editing data within the clothes table of the clothing_store database.

Libraries used

1. javax.swing.
2. java.awt.event
3. java.awt
4. java.sql
5. java.util.Enumeration
6. javax.swing.event.DocumentEvent
7. javax.swing.filechooser.FileNameExtensionFilter
8. java.io
9. java.util.ArrayList
10. java.util.List

WordCount: 1338

Bibliography

1. (No date) *Byte arrays*. Available at: https://web.mit.edu/barnowl/share/gtk-doc/html/glib/glib-Byte-Arrays.html (Accessed: 07 April 2024).
2. Nero, J.C.B.R. del and Nero, R. del (2019) *Sorting with comparable and comparator in Java*, *InfoWorld*. Available at: https://www.infoworld.com/article/3323403/java-challengers-5-sorting-with-comparable-and-comparator-in-java.html (Accessed: 08 April 2024).