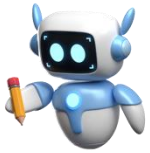
HDC – HGP - Assignment 02
UI Design Document



Student Name: Anriel Almeida
Student Number: 3168178

1. Using an application such as draw.io, canva or similar, create a wireframe of the layout of your UI, including explanation of what containers you used and why, as well as the naming convention used for each of your components.

Wireframe and explanation:

Containers :Vbox (vbMain) stacked on top of Grid Plane

VBox (**vbMain**)

- Acts as the **main outer layout container**.
- **Explaination**: VBox stacks children vertically from top to bottom, which I used for stacking the **MenuBar** at the top, and the **GridPane** layout below it. In this way it is structured appropriately. used to hold the menu bar (which has special feature). Menu bar will drop down to show menu items.

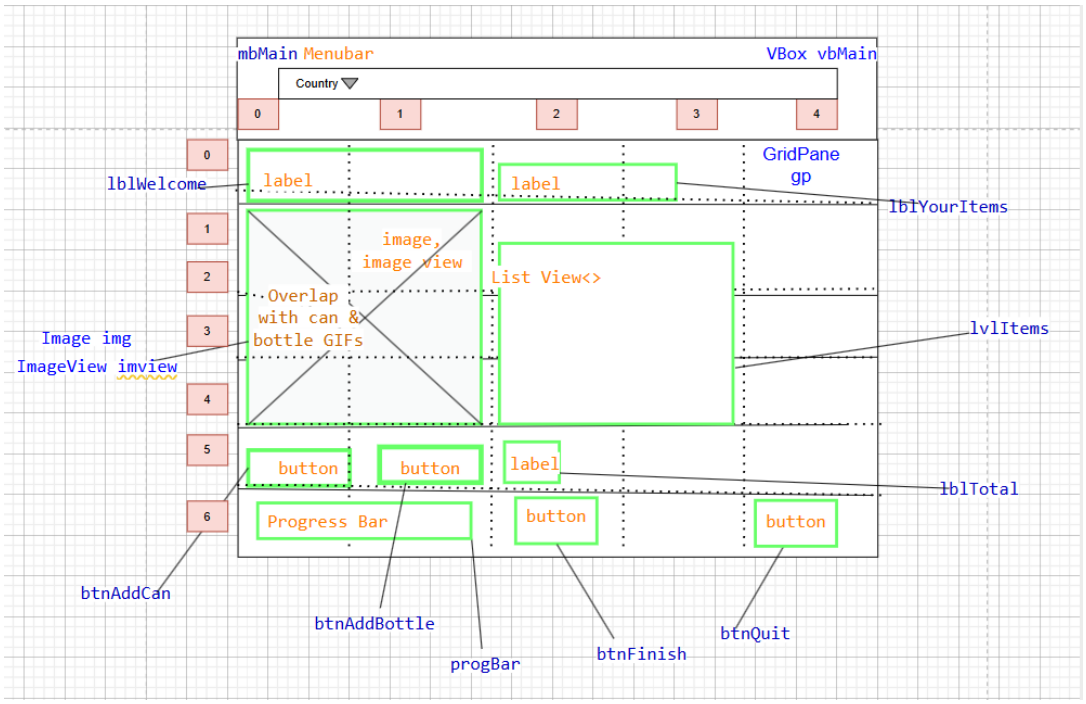
GridPane (**gp**)

- Main container for arranging interactive components like buttons, labels, images, ListView, and progress bar.
- **Explanation**: GridPane allows the exact positioning of UI elements using rows and columns. Since there were many elements grid plane gave me more control over the positioning. Also knowing that the GIFs would be overlapping on the image I choose Grid plane for smooth transitioning. Used for controlling elements to create visual structure, consistency and visual flow of the whole process. Symmetry was also focused on between the 1) add buttons and 2) image and ListView

Component naming convention follows Hungarian notation:

- **Elements**: Prefixes indicate type (btn for Button, lbl for Label, lbl, lv for ListView, progBar for ProgressBar, imv for ImageView, mb for MenuBar, mnu for Menu, mi for MenuItem).
- **Containers**: Prefixes indicate layout type (vb for VBox, bp for BorderPane)
- **Camel case**: Used for some identifiers (btnAddBottle , miCountryFrance)
- **According to function**: (lblWelcome: Label displaying the welcome message), (btnAddCan: Button to add a can item), (lvItems: ListView showing the list of items), (progBar: ProgressBar indicating the export progress), (imv: ImageView displaying images related to recycling actions), (mbMain: Main MenuBar containing menus, mnuCountry: Menu for selecting the country), (miCountryGermany: MenuItem for selecting Germany.)

Accurately named labels and buttons make the users feel in control.

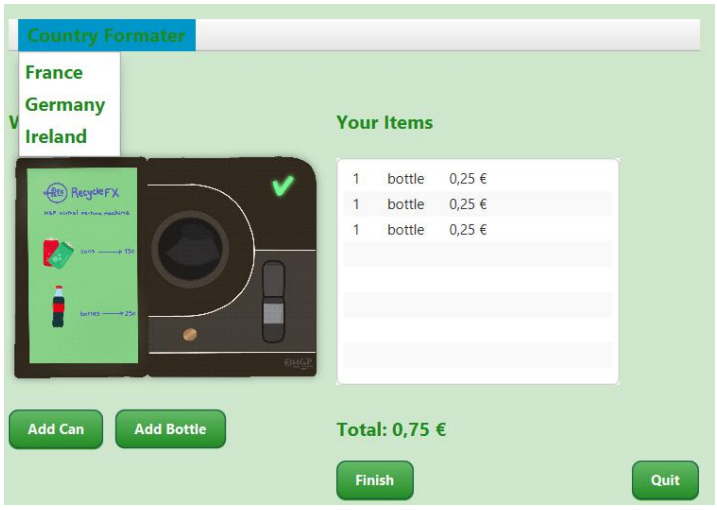


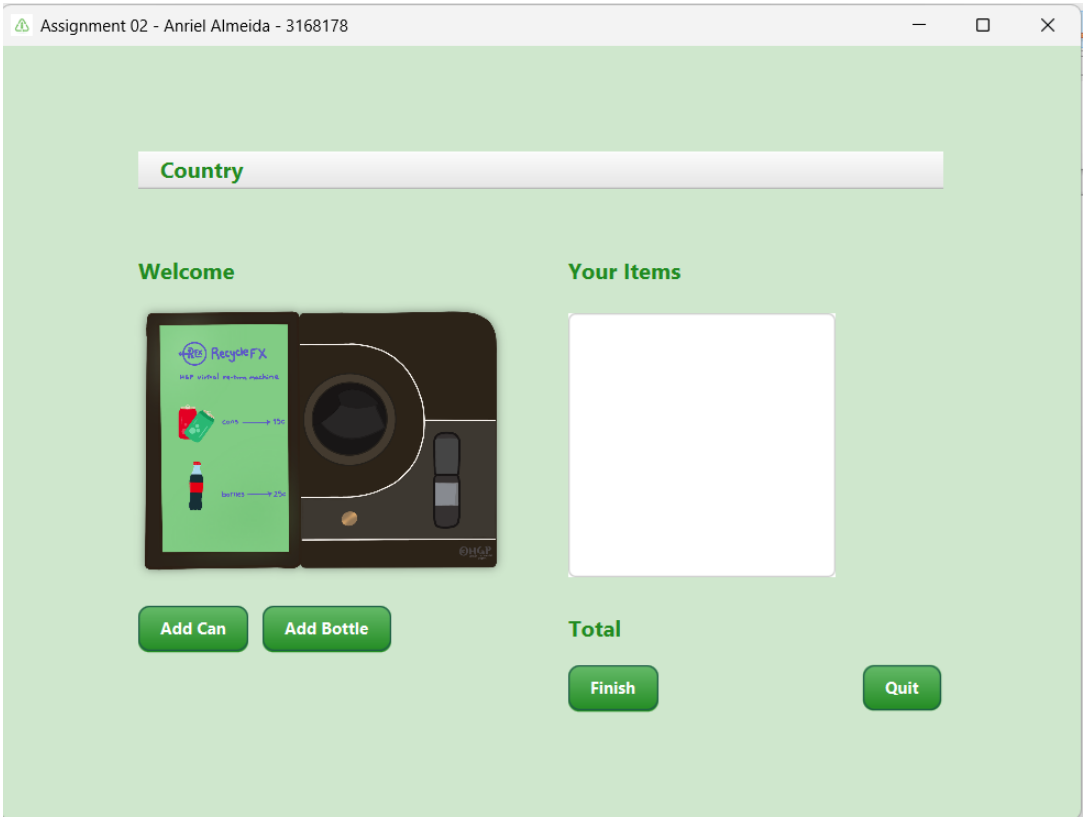
2. Include screenshots of your Main User Interface. Why does your UI look the way it does? What design decisions did you make for this interface? Justify your decisions with a foundation of UI Design Principles.

Screenshots and explanation:

I chose to go with a forest green colour as my app is environmental based and to contrast it, I used a light green background colour with doesn't clash with the GIF image colours. And to maintain consistency I styled the buttons , menu items, labels and the elements on focus / hover in similar shades.

The font I used Segoe UI, is because I wanted an easy to read and professional font.





1. MenuBar (**mbMain**)

- **Location**: Top of the window – according to accepted convention and expected location for navigation menus, increasing *predictability, void Ambiguity and experience tunes us to look for expected features like menu bar in expected places on top of the screen.*
- **Colour**: Standard light grey with forest green text ensures *high contrast*, aiding *readability* and *accessibility*. Following the forest green consistent theme with labels and buttons for a cohesive and aesthetic look.
- **Size**: Default size to maintain *consistency* and prevent distraction from main functionality that is calculating the voucher price and generating vouchers.
- **Style**: Minimalist design following the *principle of simplicity*, ensuring that users are not overwhelmed.
- **Justification**: Users are trained to find navigation at the top – leveraging **familiarity** and **consistency** principles.

2. Label Welcome (**lblWelcome**)

- **Location**: Positioned at the top of the main panel – immediately visible, acts as a *first point of interaction.*
- **Colour**: Forest green text on a light green background to ensure *high contrast* and visual appeal.
- **Size**: Slightly larger than body text – to signal its importance without overpowering other components.
- **Style**: Bold and clear font, following the **visibility** and **hierarchy** principle.
- Justification: Accurately named labels and buttons make the users feel in control.

3. ListView of Items (**lvItems**)

- **Location**: Left-center of the grid – placed for *easy scanning*, users read from left to right.
- **Colour**: Light background with black text to ensure *accessibility* for users with low vision.
- **Size**: Large enough to display 5+ items without scrolling – supports *efficiency of use*.
- **Style**: 1)Uses default list styling for *consistency* and *familiar interaction*.

2) **Symmetry: ImageView, ListView** are styles symmetrically. Symmetrical designs as reduce perceived complexity., Enhances aesthetics and balance.

4. ImageView (**imv**)

- **Location**: Center-right of the layout to balance text-heavy elements on the left and visual content on the right.
- **Colour**: The image displayed adjusts based on user actions, offering *visual feedback*.
- **Size**: Proportioned to not overwhelm the ListView but large enough to convey meaning.
- **Style**: 1) CSS styling:.image-view {

-fx-effect: dropshadow(gaussian, rgba(0,0,0,0.2), 10, 0.5, 0, 0);

-fx-background-radius: 10;

}

Drop shadow used for 3-D effect and increase aesthetic appeal.

Figure/Ground: We distinguish objects (recycling bin) to pop out and come to foreground from their background.

Helps in highlighting primary UI element (recycling bin).

- 2) **Symmetry: ImageView, ListView** are styles symetrically. Symmetrical designs as reduce perceived complexity., Enhances aesthetics and balance.

5. Buttons (e.g., **btnAddCan**, **btnAddBottle vs btnFinish,btnQuit**)

- **Location**: btnAddCan**,** btnAddBottle placed under the image grouped close together styled similarly following principle of proximity

btnFinish,btnQuit :Aligned below the ListView and image

Follows Visual Hierarchy: For logical flow: *select item → see total → take action*. (Principle of continuity: The eye follows continuous lines or patterns rather than abrupt changes. Helps in smooth navigation paths and content flow.)

- **Colour**: For all buttons following principle of similarity as they belong to the same app.

Forest green was chosen for aesthetics as a base colour because it is an environmental themed app.

- **Size**: Larger than labels or list items to afford clicking/tapping – enhances *affordance*.
- **Style**: Uses consistent button style (background color, hover effect, padding) – following

the *consistency* principle.

- **Principle:** 1)**Similarity:** Visually similar elements (color, shape) are grouped together,

Using this for btnAddCan**,** btnAddBottle to make users think that they belong to the same category.

2)Proximity: Items close together are perceived as a group, this affects how users interpret layout and organization.

3) **Symmetry:** btnAddCan**,** btnAddBottle are styles symmetrically. Symmetrical designs as reduce perceived complexity., Enhances aesthetics and balance.

4) **User Control**: Accurately named labels and buttons make the users feel in control.

6. ProgressBar (**progBar**)

- **Location**: Bottom of the main panel.
- **Colour**: Darker forest green fill – ensures visibility while matching the environmental theme.Using inline styling progBar.setStyle("-fx-accent: #1e7a1e;");
- **Size**: Using progBar.prefWidthProperty().bind(primaryStage.widthProperty().divide(3.5)); Fits under the image and ends with sufficient space before the Finish button.
- **Style**: Animated fill – provides *immediate feedback* to the user when exporting data.
- **Principle: 1) Consistency:** Similarly, coloured to fit theme

2) User feedback: to wait for the voucher to be processed

7. Country Menu (**mnuCountry**)

- **Location**: Inside the MenuBar, logically grouped with its menu items.
- **Colour**: Matches menu color scheme – ensures *visual consistency*.
- **Size**: Compact dropdown – doesn't require screen space unless interacted with.
- **Style**: Standard dropdown style to follow *platform conventions*.
- **Principle: 1) Consistency:** Similarly coloured to fit theme

2) Visual feedback: Hover effect in CSS styling

8. Label Total (`lblTotal`)

- **Location**: Positioned below list view
- **Colour**: Forest green text on a light green background to ensure *high contrast* and visual appeal. Uses consistent label styling as it belongs to the same app.
- **Size**: 16px
- **Style**: 'Segoe UI', sans-serif and clear font.
- **Principle: 1) similar and consistent to other labels**

2) Match between system and real world: Intuitive to user as, it is below the vertical stack of list of added items acting like the real world recycle machine or other calculating machine where the

total is given below the list of mathematical operations. In the sense when mathematic addition operations are performed numbers are added to a list and the total is calculated at the bottom.

3)Numbers Formatting to help uses read numbers easily

3. Briefly explain the structure of your application, including what methods are included and what their functionality is.

Explanation:

1. Application Architecture
   - **Extends Application Class**: Follows (init, start, stop)
   - **The Pattern**: Separates UI components, event handling, logic

2. Component Organization

   - **Class-Level Components**: UI elements declared at class level for global accessibility
   - **Centralized Initialization**: Components are instantiated in the constructor
   - **Event Handling**: Event listeners configured in the init() method
   - **Layout Design**: UI layout structured in the start() method

3. Key Structural Elements

   - **Main Window**: Primary Stage containing a Scene with VBox as the main layout
   - **Layout Hierarchy**:
     - VBox (main container) - (MenuBar + GridPane)
     - GridPane for precise component positioning with defined column/row structure
     - Components arranged in (welcome area, image display, item list, controls)

4. Functional Structure

   - Methods for processing user interactions (addCan, addBottle)
   - Methods for generating vouchers and calculating totals
   - Methods for exporting data to external files
   - Use of Task Object for background thread for long task
   - Methods for resetting and updating the interface

5. Error Handling

   - **Exception Management**: Try-catch blocks for volatile operations
   - **User Feedback**: Alert dialogs for communicating status and errors
   - Application continues functioning even if non-critical components fail

6. Special Features

   - Currency formatting based on locale selection
   - Quit button
   - Drop down menu

METHODS:-

constructor()

public RecyclingFX()

- Initializes all elements such as labels, buttons, list views, images, progress bar, and menu items, instantiate components using keyword 'new'.
- Loads a default image and sets styles and visibility for components. eg.progBar.setVisible(false); , progBar.setStyle("-fx-accent: #1e7a1e;");
- Prepares locale and currency formatter (special feature).

init()

public void init()

- Registers **event handlers** for buttons and menu items.
    - btnAddCan and btnAddBottle - add items and update image, list, and total.
    - btnFinish -triggers voucher export and progress bar.
    - btnQuit - exits the application.
    - Menu items (France, Germany, Ireland) - change the currency format.

start(Stage primaryStage)

public void start(Stage primaryStage)

- Sets up the primary stage (window title, size, and icon).
- Creates the main layout using VBox and GridPane.
- Adds all components to the layout.
- Adds the menu bar and sets spacing, alignment, and padding.

main()

launches the JavaFX application.

Custom Methods

AddCan() -Loads a GIF for a can., Adds a can entry to the list view. , Updates the total and its display.

addBottle()-Similar to addCan() but for bottles. , Loads a different GIF and updates total accordingly.

ClickFinish()- Disables the button, checks if the list is empty. , If items exist, shows a progress bar, writes data to a voucher file, and resets the UI., Handles errors gracefully and shows appropriate alerts.

updateCurrency(Locale locale)-Updates the NumberFormat based on the selected country. Refreshes item prices and total in the selected currency format.

alertVoucher()-Displays an informational alert when trying to export with no items.

4. In the context of JavaFX, explain what a Thread and a Task are, how they work together in a JavaFX application, and why an extra thread is needed to update a progress bar during a long-running operation instead of using the main UI thread. Explanation should be in your own words.

Explanation:

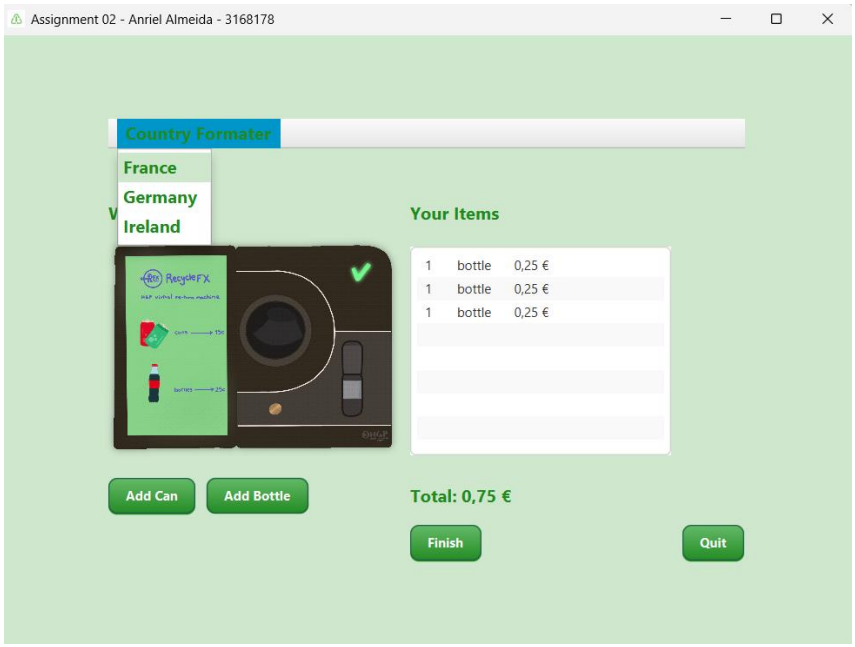
A thread is a unit of execution in a program. JavaFX has one main thread on which different tasks run on it. When a task is too long the program doesn't work well, freezes and becomes unresponsive, because the thread is too busy.

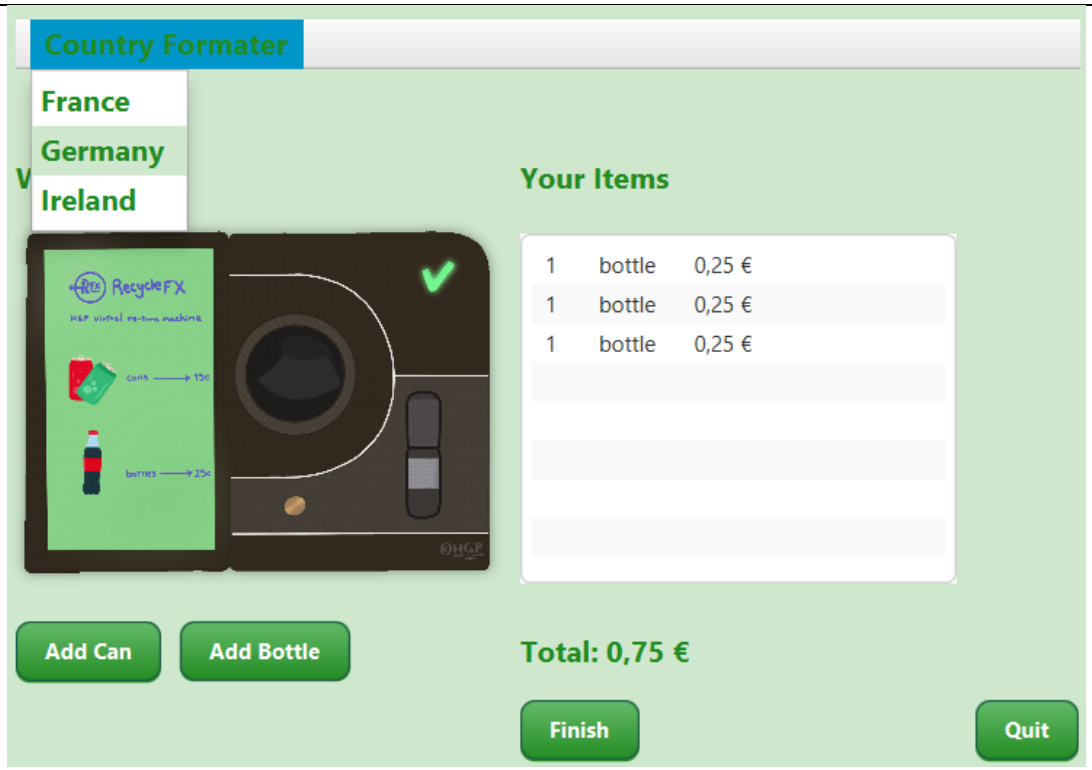
A task object is needed to be created for a different thread (runs in the background) than the main thread (free to perform other tasks). To do that you have to create a Task object, so the task object is used to execute the long task in the background, so the main thread is not compromised.
An extra thread is needed to update the progress bar because threads perform task extremely quickly and for the progress bar to be visible at a rate where the user can see it we need to slow it down. So, this makes it a long task and therefore needs a separate thread to run it instead of compromising the main thread which is free to other tasks.

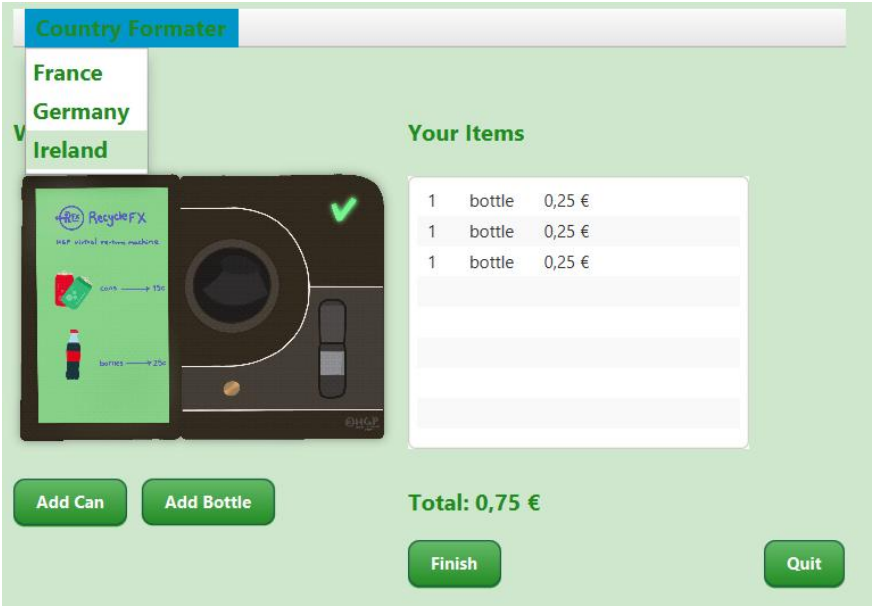
Oracle , 2014. *Java Platform, Standard Edition 8 API Specification: Class Thread* . [online] Available at: https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html (Accessed: 9 May 2025).

5. EXTRA FEATURE: Explain what your extra feature is, why you chose it, how it works etc. Include any relevant screenshots of your extra feature in action. Remember you must complete this section in detail to receive marks for your extra feature.

Screenshots and explanation:







1. **Dropdown Menu (Country Selection)**

I chose a drop-down menu because I wanted the user to choose from a list of at least 3 different countries and this was the most cleanest way to go about doing that,

The dropdown menu allows the user to **select a euro using country**, (specifically France, Germany, Ireland) which updates how the prices of bottles and cans are displayed (i.e., changes the **currency symbol** and formatting).

Components Involved:

1. MenuBar – the top bar that holds menus.
2. Menu – a dropdown list of options, like "Country".

3. MenuItem – the actual clickable options in a Menu.

2. **Currency Formatter**
I chose this feature because I was looking for a way to increase my user base.

Purpose:

To **dynamically format the prices** in the ListView (lvItems) and the lblTotal label to reflect the **selected country's currency format**.

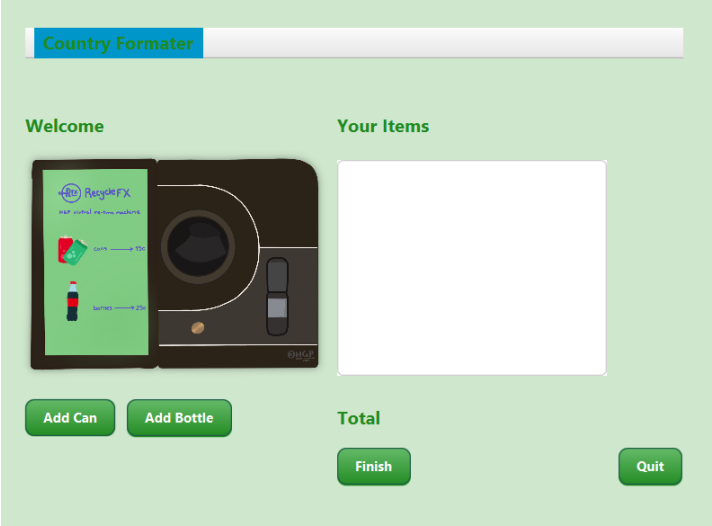
when a user selects “France,” updateCurrency(Locale.FRANCE) is called

The way currency is formatted in different countries. Germany uses: 1.234,56 € euro symbol is written behind “,” used instead of .
France uses: 1 234,56 € euro symbol is written behind and “,” used instead of “.” uses space instead of, for thousand
Ireland uses: €1,234.56 uses US format with euro in the front / decimal used instead of “,”

The method- updateCurrency(Locale): Changes how prices are displayed &Updates ListView and lblTotal

3.Another extra feature of mine is the **Quit button** which can quit the application – using method Platform.*exit*());

- I chose it because it is easy for the person to close the app and follow the logic flow of the design from adding can/ bottle to seeing the total, generating voucher and then quitting. Its easy to find the button and quickly close the app.



6. Describe the interactive elements in your UI. How did you make sure they are intuitive and easy to use?

Explanation:

Interactive element List and Description:
I made them intuitive by **naming them is a user friendly short and meaningful way that it is easy to understand**. This makes it intuitive for the user. And easy to use.

1. Buttons
AddCan/ Add bottle : Adds can worth 0.15/0.25 euro to the total. When clicked triggers gif to show can /bottle added to machine animation and adds 0.15/0.25 euro to the total, keeps adding to total as it is clicked. Also, it will recalculate using the locale based currency (due to special feature). It is intuitive that they have similar functions since they are grouped closer together and styled similarly

1) btnFinish/ btnQuit:
Starts a background task to simulate processing, then exports voucher info to a file and clears the UI./Uses Platform.exit()to close the app.
The finish buttons and quit button are spaced away indicating that they aren't closely linked. The quit button is place at the end corner of the page at the end of the process flow. The finish button is placed after the total is got so that you can click the finish button next.

2)Menu items: The menu bar has a drop-down menu for country to choose formatting and has the list of country names to choose from. Menu bar is places on top for the user to easily identify but is set away from the main features which are at the center of the app. The menu is styled minimalistically to avoid distraction. So, users are intuitively focused on the important functions instead.

3) ListView (lvItems):

Displays a dynamic list of added items (cans and bottles).

- Updates dynamically when user clicks **Add Can** or **Add Bottle** .
- Uses locale-specific formatting for prices (e.g., 0,15 € instead of €0.15).

Display along with the changing total label act like the real life version of the recycle machine screen so the user intuitively expects that adding the can / bottle will increase total label.

4) Total Label: (Not directly interactive but responds to user action)

Dynamically calculates the total of the items added to the list

Intuitive to user as, it is below the vertical stack of list of added items acting like the real world recycle machine or other calculating machine where the total is given below the list of mathematical operations

5) Progress Bar (progBar)
Shows progress during the voucher export simulation, appears temporarily while the background task runs. Binds to the progress property of the Task used during export.
It is to provide a feed back to the user to wait as a longer task is in progress. So, the user can intutively know to wait and that the process is in progress, as the progress bar is being filled.

6) Alert Dialogs
1)Information Alert comes on when Export completes successfully
message shown is "Voucher exported successfully!"
2)Information Alert comes on when user tries to export an empty list
message shown is "There are no items to export."
3)Error Alert comes on when there’s an issue writing the voucher file and displays exception message

7) Image Viewer (ImageView imv) :

Displays animated GIFs when a can or bottle is added.

- Updates dynamically based on which button was clicked:
- recycleCan.GIF for cans
- recycleBottle.GIF for bottles

- Resets to default image (recycle1.PNG) after export/reset.

Intuitive for the user to expect the next time when the button is pressed or than when say the can is pressed and generates can related GIF to intuitively expect the other button to create a bottle related GIF.

7. What do you envision to be the user base for this application? Justify your response. Describe the typical user of this application.

Explanation:

User base will be
1) General public – someone who is owns and can use money (euros) to buy can and bottles .
Justification- Someone who wants to calculate the amount they will receive by a voucher for their purchase.
2) Environmental conscious: People who care about the environment and recycling
Justification- recycling decreases environmental pollution and can encourage more users to recycle and track habits if they have an incentive.
3) People of Low income: Students and people of lower income who depend on vouchers for extra money. Poorer educational background people are expected so the naming convention is easy to understand.
Justification- can be used to calculate vouchers before purchase for money saved.
4) Above the age of 18: People who are above the drinking age as most of the cans and bottles purchased are alcoholic based.
Justification- encourages collection, segregation and proper disposal for repeated buyers
5) People who use euro: Especially Germans, French and Irish. Non english speakers are expected so the app is designed with simple naming of labels and buttons.

Justification: as I have specifically designed the currency formatter for these 3 countries.

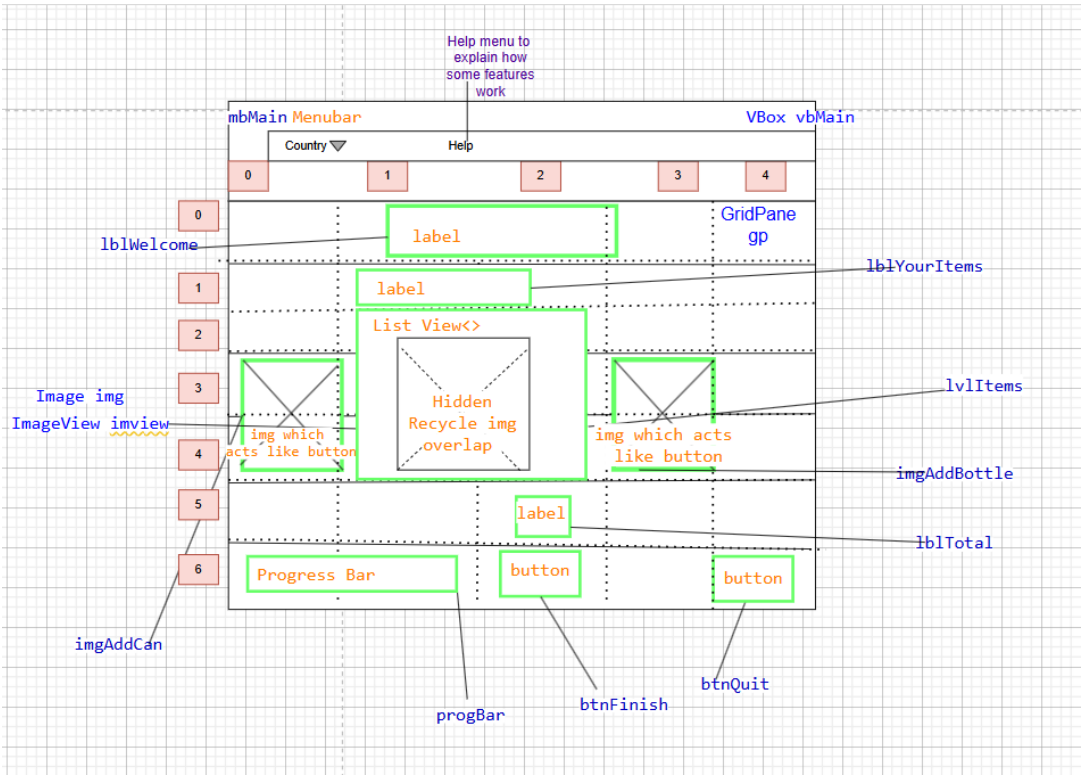
Typical User Profile/ Persona

- **Name**: Emily, age 32
- **Occupation**: Student and Part-time worker
- **Lives:** Ireland
- **Tech Skill Level**: Moderate
- **Device Used**: Laptop or desktop (JavaFX-based)
- **Goals**:
    - Calculate amount of money saved by vouchers
    - Track her own recycling habits
    - Encouraged to be environmentally conscious
- **Needs**:
    - Simple and intuitive UI
    - Clear labeling and icons
    - Visual categorization of waste

- Motivation to recycle by voucher incentive

8. How would you redesign this application? Draw a new wireframe detailing what you believe the ideal design to be for the RecyclingFX application.

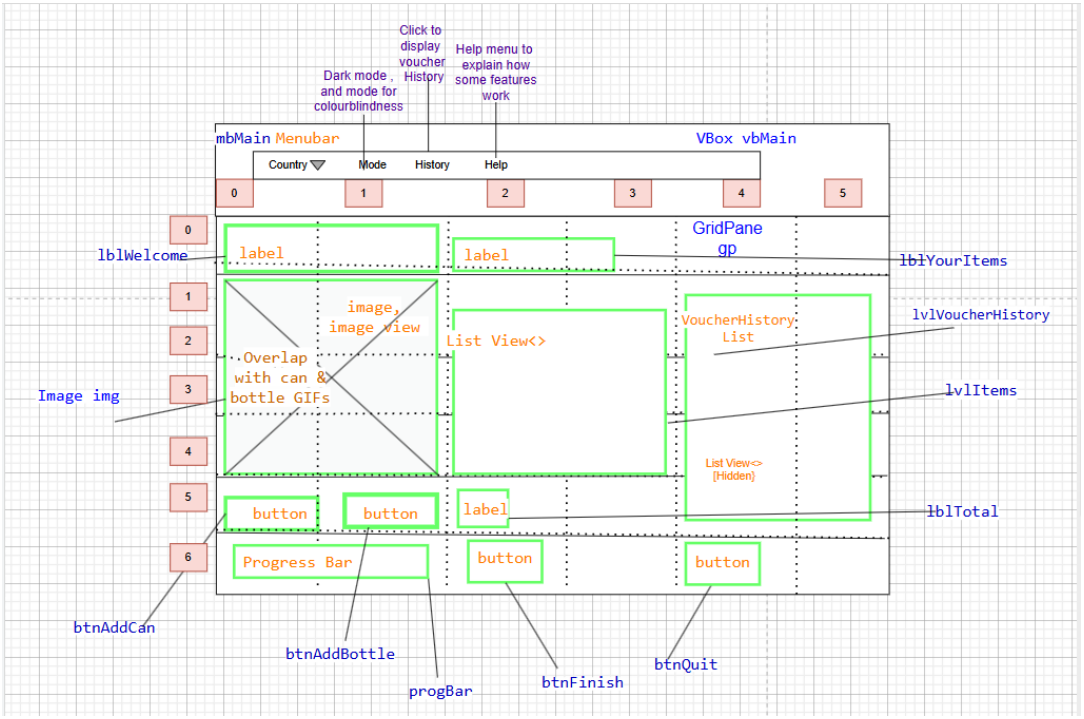
Wireframe and explanation:



I would redesign the application by
1) adding the list view to the center – because it is the main feature
2) instead of add can and bottle buttons- I would add a can and bottle image to the side of the list view. This is so the app can cross language barriers
3) Add the Recycle GIFs which is hidden on top of list view, to only appear when the respective buttons are clicked. When the image button is clicked the GIF will appear on top of the list view, play and then go back to being hidden.
And then you will see the item added to the list.
In this way all the features will still be present but will have the List view and total as the focus.
Can use a method like
imgRecycle.setVisible(false); // Hides the image but keeps its layout space
lvItems.setVisible(true);

9. If you were to continue working on improving this application, what enhancements or updates would you add and why? Briefly explain what you would incorporate in future work, justify your reasoning.

Explanation:



I would add more menu options like:
1) A mode option: which has 2 menu items of dark mode
and mode for colour blindness – this would increase the number of types of users using the app. Dark mood would reduce strain on the eyes and save battery life of user's devices and could give a user more control over the user experience
-CSS file styling will be used to switch between options
2) A History option:
Click to display voucher History in a list form, to track voucher history.
A hidden panel will appear at the left-hand side gp.add(historyItems, 4, 1, 1, 4);

- This could help the user keep track of past history and increase the functionality of the app

3) Help option: Help menu to explain how some features work through dialog/ alerts

- Help users who are not able to understand some of the features get instructions to use the app. This will increase the type of users using the app and enhance the user experience for first time users.

If I were to continue improving this application, I would implement several key enhancements to improve functionality, accessibility, and user experience:

1. Accessibility Enhancements in the menu bae

- **Display Mode Options:** Implement a comprehensive Mode menu with:
  - **Dark Mode:** Reduce eye strain during extended use, decrease battery consumption on, and provide a modern aesthetic alternative that many users prefer.
  - **Color Blindness Modes:** Include specific color schemes optimized for various types of color vision deficiency, ensuring the application is accessible to the population with color vision impairments.
  - **Implementation:** Create a CSS theme styling system that allows switching between visual modes while maintaining UI consistency and readability.

2. Transaction History Management in the menu bar

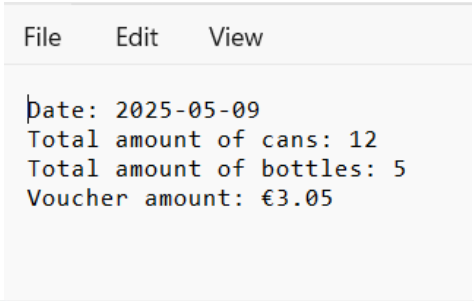
- **History Tracking System:** Develop a history feature that:
  - Maintains a record of all recycling vouchers
  - Displays transaction history in a hidden side panel.
  - **Implementation:** This would be integrated as a sliding panel (gp.add(historyItems, 4, 1, 1, 4)) that dynamically appears when requested.

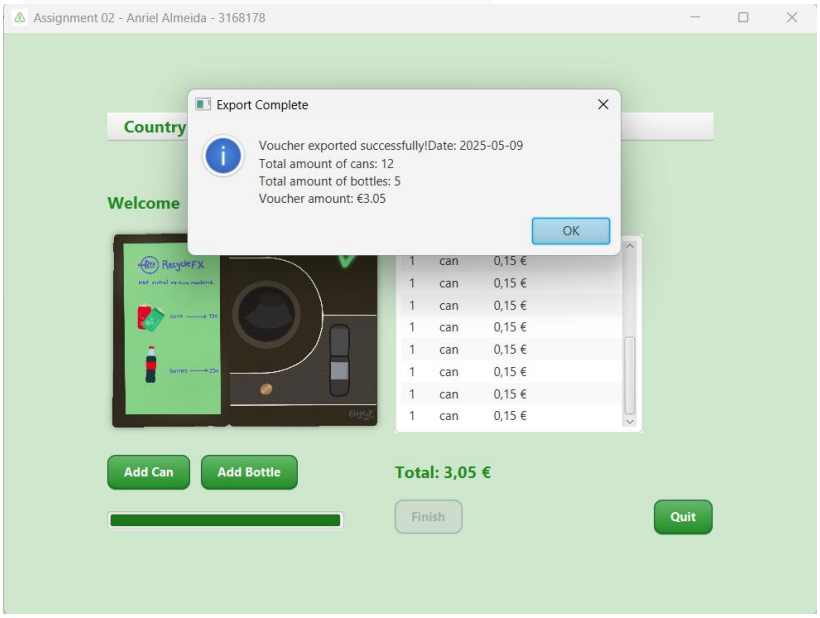
3. Help System in the menu bar

- **Contextual Help Features:**
  - Information that explains functions in the form of alert or dialog boxes
  - Step-by-step instructions for first-time users
  - **Implementation:** Use of JavaFX alert dialogs,

10. Describe the steps you followed to complete the file export task for part (f). In your user interface, add 5 bottles and 12 cans to the recycling machine and click finish to export the voucher as a text file. Provide screenshots of both the user interface and the exported file below. Include the exported file in your submission.

Explanation and screenshots:





1)Background Task Management

- The code uses Task<Void> to create a background task that doesn't block the UI
- The startExportTask() method creates and starts this background task
- When the task is completed, it calls succeeded() which uses Platform.runLater() to update the UI on the JavaFX Application Thread

- **2)Counting Items from ListView**

- The generateVoucherText() method iterates through the ListView's contents
- It counts cans and bottles separately using a for-each loop that checks if each item string contains "can" or "bottle"

3)Voucher Text Generation

- Gets the current date using java.time.LocalDate.now().toString()
- Formats the voucher text according to the specified format using String.format()

4) **File Writing**

- Uses BufferedWriter for file writing
- Creates a file named "Voucher#3168178.txt" (using the student number)
- Writes the formatted voucher text to the file
- Properly closes the writer to ensure data is flushed to disk

5)Exception Handling

- The file writing operation is wrapped in a try-catch block to handle possible I/O exceptions
- If an exception occurs, it prints an error message and stack trace for debugging

6)User Feedback

- Shows a success alert dialog to inform the user that the export completed successfully
- Includes the voucher text in the alert dialog so the user can see what was exported

7) Clear UI

- The clearUI() method clears the ListView of all items
- Resets the total amount to zero
- Updates the total label to show "Total" (without any amount)
- Resets the image to the default recycling image

8) ClearAfterExport()

- The clearAfterExport() method runs regardless of success or failure (in the finally block)
- Unbinds the progress bar from the task's progress property
- Resets the progress bar to zero
- Hides the progress bar
- Re-enables the Finish button for future use

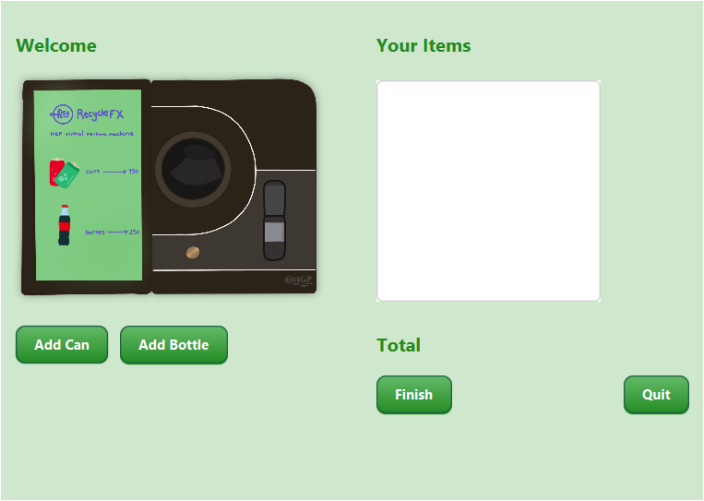
11. With the aid of a diagram, explain the Gestalt Principle of Proximity and its significance in interface design. Show examples of where this principle applies in your own UI.

Diagram, explanation and examples:

Depending on distance from 2 objects we belive that those elements are either grouped if close or not grouped if far
‣ Based on spacing- It appears like
in the first image you see three columns, and in the
second image you saw three rows

In my design you can see this principle at play in the addCan and add Bottle buttons vs the Finish and Quit buttons
- Since add can and bottle are closer together, they appear to be grouped together- you intuitively feel they have similar function.
- Finish and Quit buttons are further apart they appear not to be grouped r- you intuitively feel

they have different function.



12. If any part of your submission isn't working, please detail this below including relevant screenshots.

Screenshots and explanation:

Changing the css colour for menu when hovered/ focused