Object-Oriented Programming Final Project

Instructor: Sir Jhun Brian Andam

Project Proposal: Randomizer

Group 1

ANCHETA, JOSHUA GIL B

ARSULA, JOMARI F

BLANCO, Noel jhumel G

CABASAG, Claire D

MAGLINTE, Prince Dave M

RAGURO, Ruelyn P

SAROL, Kent O

Task Description and Features

The group has chosen the Randomizer Application as our project theme. The application is designed to simplify tasks that require randomization–such as group randomization, or getting one random output–. The application will be a useful and versatile tool for students and teachers alike. Additionally, the application will offer intuitive user interfaces and multiple output formats. The following main features will be included in the randomizer application:

Random grouping, it will automatically divide a list of inputs–like names or numbers– into random groups. This feature will be useful for team assignments, study groups, or even event planning. Moreover, the group size or number of groups will be adjustable to your liking.

Shuffling, it will rearrange a list of items–e.g. names, numbers, or tasks– into a completely random order. Therefore, it will be perfect for creating unbiased task assignments, presentation orders, or seating arrangements.

Random output generation, as the name suggests, will generate random outputs based on the user input. Moreover, the randomizer application will be capable of generating single or multiple outputs at once.

The Randomizer application will consist of the following components:

First, we have the input module. The input module will allow the users to add inputs manually. Moreover, it will include a simple and user-friendly interface for entering and managing data.

Second, the package will consist of a processing module that is implemented by using randomization logic for shuffling. This will ensure fairness and true randomness in group creation and output generation.

Third, the group will be putting settings and preferences in the application to allow the users to configure the tool based on their needs.

Finally, the output module will display the results on-screen.

The randomizer application aims to make randomization tasks quick, easy, efficient, and fair to users who will be using it. Moreover, with an intuitive interface, users can effortlessly input their criteria and preferences, ensuring a personalized experience.

Class Hierarchy and Methods

1. Base Class: Randomizer

- Attributes**:**
    - inputs (List[str | int]): Stores the user-provided inputs.
    - results (List[str | int]): Stores the outputs after processing.
- Methods**:**
    - add_input(input_data: str | int): Adds a single input to the list.
    - add_inputs(input_list: List[str | int]): Adds multiple inputs to the list.
    - clear_inputs(): Clears all inputs.
    - get_results() -> List[str | int]: Returns the results.

---

2. Subclass: GroupRandomizer (inherits Randomizer)

- Attributes:
    - group_size (int): Number of items per group.
    - number_of_groups (int): Number of groups to create.
- Methods:

- set_group_size(size: int): Sets the desired group size.
- set_number_of_groups(count: int): Sets the desired number of groups.
- generate_groups() -> List[List[str | int]]: Randomly divides inputs into groups based on group_size or number_of_groups.

---

3. Subclass: ShuffleRandomizer (inherits Randomizer)

- Methods:
  - shuffle_items() -> List[str | int]: Randomly shuffles the order of the inputs and updates results.

---

4. Subclass: OutputGenerator (inherits Randomizer)

- Attributes:
  - output_type (str): The type of output to generate (e.g., "number", "letter", "custom").
  - constraints (Dict[str, Any]): Any constraints like range or character set.
- Methods:
  - set_output_type(type: str): Sets the type of output to generate.
  - set_constraints(constraints: Dict[str, Any]): Configures generation constraints.
  - generate_output(count: int = 1) -> List[Any]: Generates one or more random outputs based on the type and constraints.

---

5. Helper Class: Preferences

- Attributes:
  - allow_duplicates (bool): Determines if duplicates are allowed in results.
- Methods:

- toggle_duplicates(allow: bool): Enables or disables duplicate results.

---

6. Subclass: ResultDisplay

- Methods:
  - display_on_screen(results: List[Any]): Displays results in a user-friendly format.

UI Layout and Functionalities

INPUT USER: This Input field is for the user input

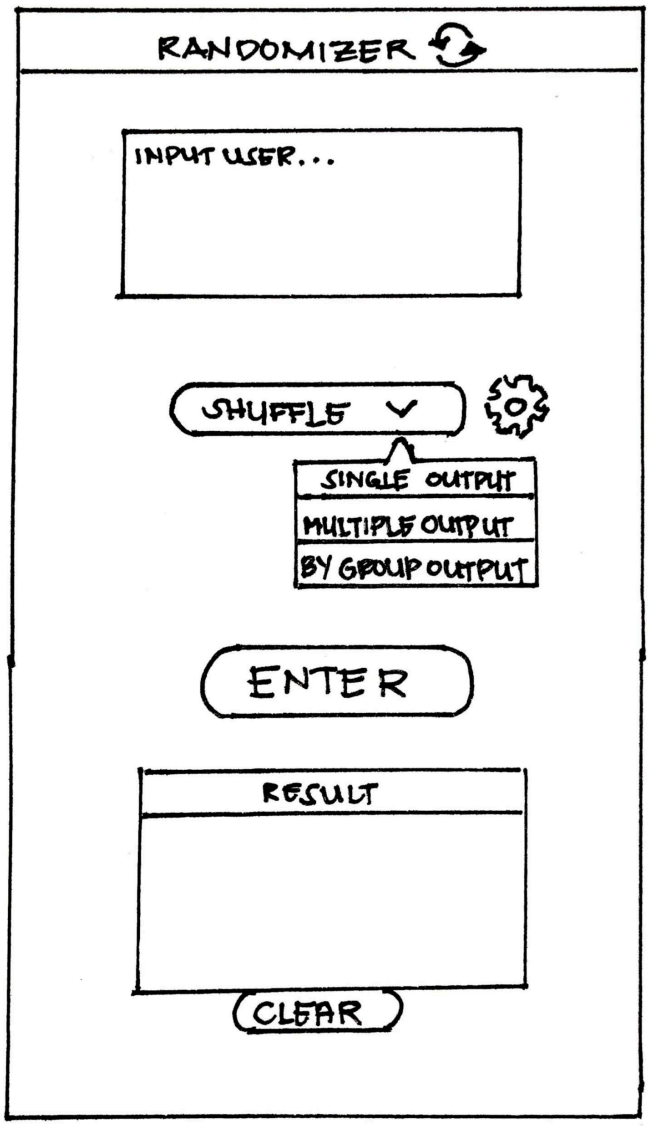
SHUFFLE: The "Shuffle" button when clicked will display a drop down button namely "Single output" if the user wants to only have a single output result, "Multiple Output" if the user wants Many output to be displayed, and the "By Group output" if the user wants the display to be outputted by groups. that the user will select according to preferred number of output

SETTING ICON: Is for allowing duplicates in the variables or not

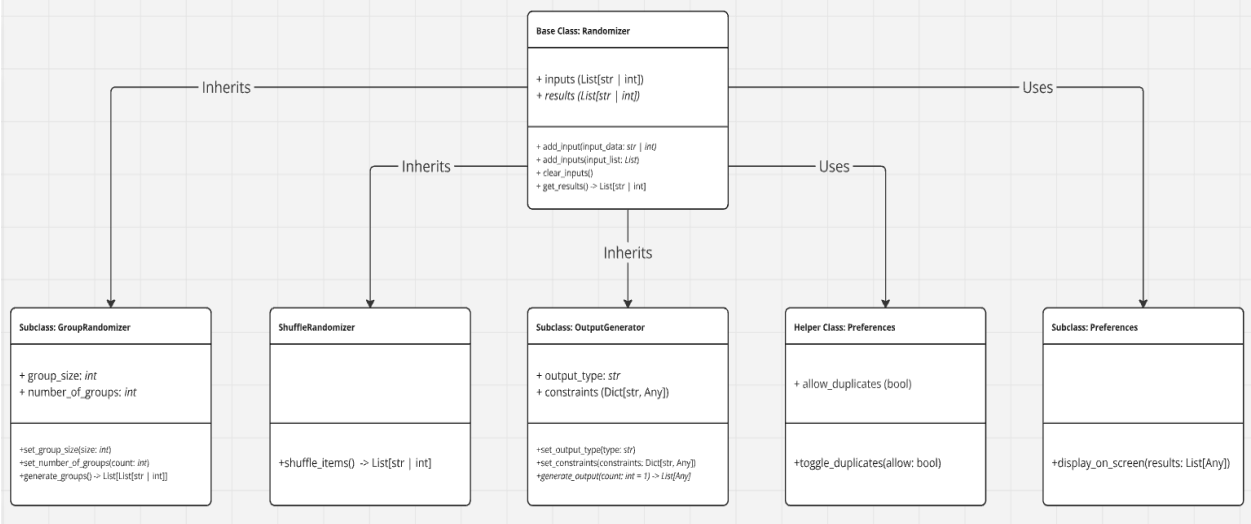
ENTER BUTTON: will enter the variable inputted by the user and start the program

RESULT FIELD: this is where the result will display

CLEAR: to clear the displayed output in the result field to be use again in the next task



UML Diagram


Base Class: Randomizer
+ inputs (List[str | int])
+ results (List[str | int])
+ add_input(input_data: str | int)
+ add_inputs(input_list: List)
+ clear_inputs()
+ get_results() -> List[str | int]
Inherits
Uses
Inherits
Uses
Inherits
Subclass: GroupRandomizer
+ group_size: int
+ number_of_groups: int
+set_group_size(size: int)
+set_number_of_groups(count: int)
+generate_groups() -> List[List[str | int]]
ShuffleRandomizer
+shuffle_items() -> List[str | int]
Subclass: OutputGenerator
+ output_type: str
+ constraints (Dict[str, Any])
+set_output_type(type: str)
+set_constraints(constraints: Dict[str, Any])
+generate_output(count: int = 1) -> List[Any]
Helper Class: Preferences
+ allow_duplicates (bool)
+toggle_duplicates(allow: bool)
Subclass: Preferences
+display_on_screen(results: List[Any])