I. ASSIGNMENT OF MEMBERS

1.ASSIGNMENT

|  |  |  |  |
| --- | --- | --- | --- |
| Nguyễn Thế An  20210006 | Hồ Ngọc Ánh  20214877 | Lê Tuấn Anh  20214874 | Mai Hoàng Anh  20205142 |

* Nguyễn Thế An:
  + List.java
  + Queue.java
  + Diagram: 15%
  + StackDisplayController: 15%
  + QueueDisplayController.java: 60%
  + ListDisplayController:10%
  + Report :80%
* Hồ Ngọc Ánh
  + Stack.java
  + Diagram 10%
  + MainmenuScreen
  + GeneralDisplayController
  + StackDisplayController:70%
  + QueueDisplayController:25%
  + ListDisplayController :75%
  + Report:20%
* Lê Tuấn Anh
  + Diagram 75%
  + ListDisplayController :15%
  + Slide 50%
* Mai Hoàng Anh
  + StackDisplayController : 15%
  + QueueDisplayController:15%
  + Slide 50%

2.About idea:

- About the GUI , we get the idea from the link: <https://visualgo.net/en/list>

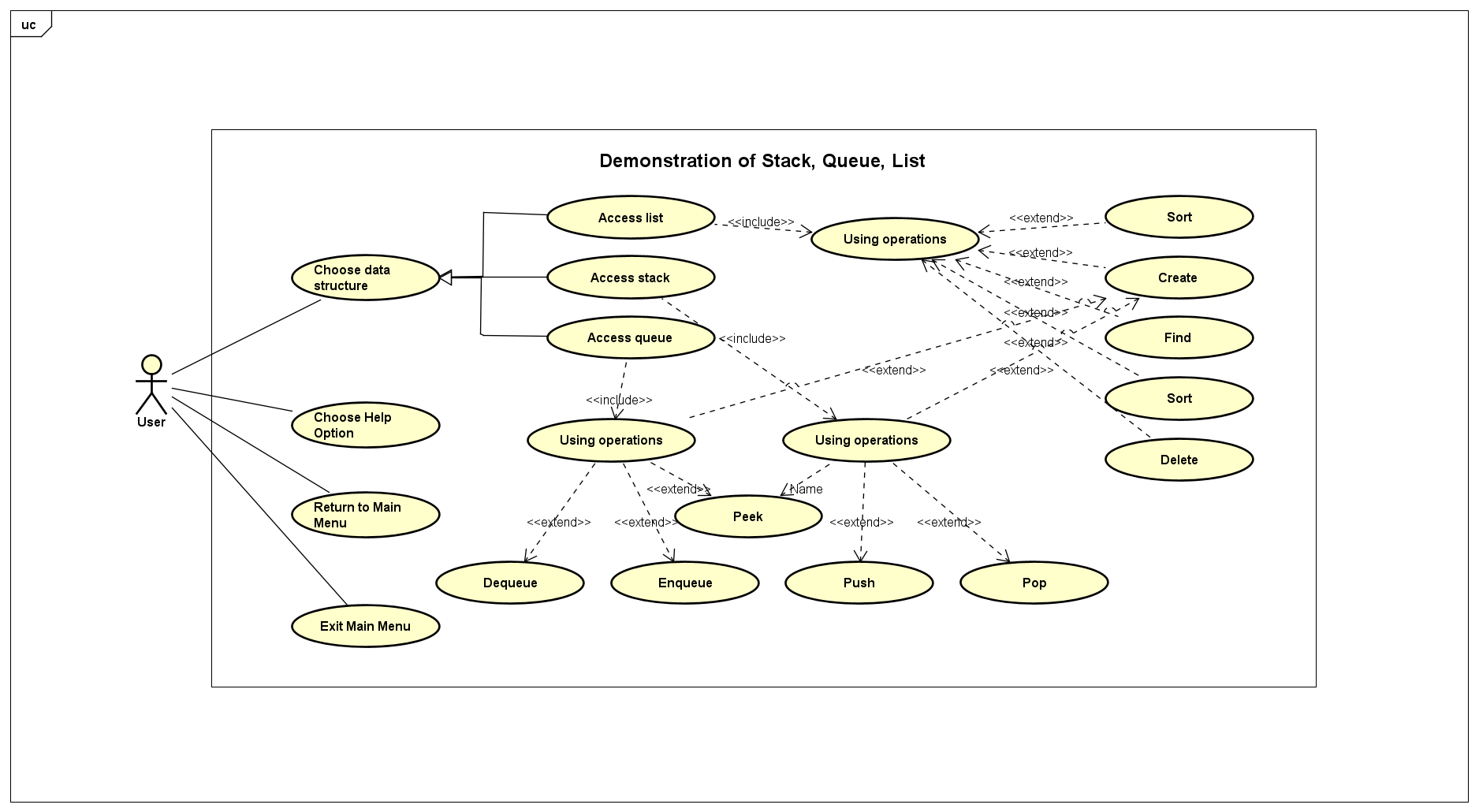
- <https://code.makery.ch/library/javafx-tutorial/>

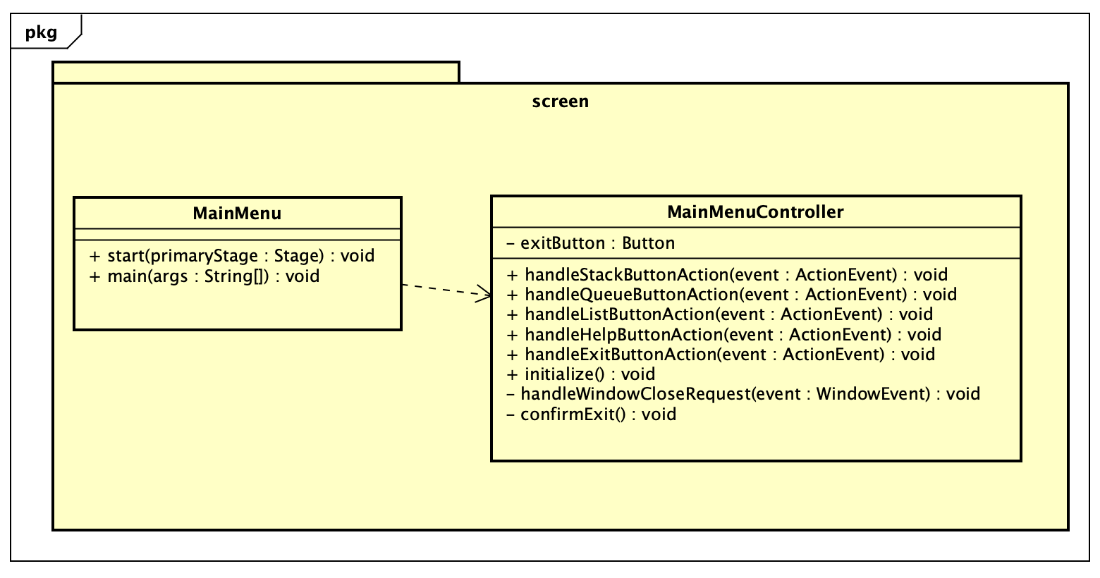
- <https://o7planning.org/11009/javafx>

II. MINI-PROJECT DESCRIPTION

1. Introduction

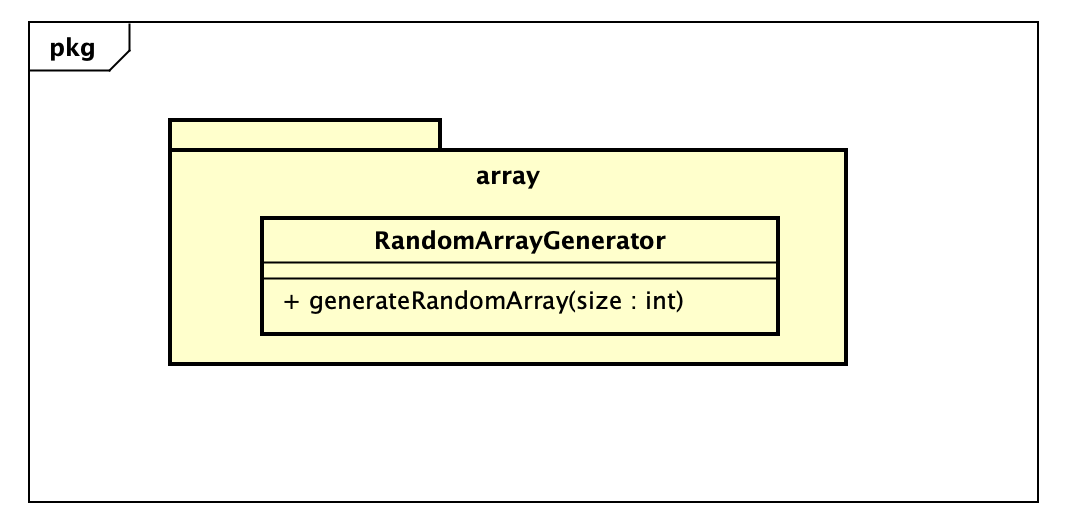
The mission of our project is to build a Java program that visualizes the basic operations of queue, list and stack.

1. General Design  
   The design requirement:  
   - On the main menu: title of the application, options for user to choose between the three types of data structure, help menu and quit  
   • User must select a type of data structure before getting into the demonstration  
   • The help menu show basic usage and aim of the project  
   • Quit button exits the application. Remember to ask for confirmation  
   - In the demonstration  
   • The main demonstration shows options for user to choose: create, insert, sort, find and delete elements in the structure. You can choose to put it in a dropdown list or separate buttons. Remember to process input if needed.  
   • After user has picked a choice, demonstrate it on the screen. For simplication, you do not need to implement revert or slow down function, but put the speed of demonstration at average.  
   • Always have a Back button for user to return to main menu at any time
   1. Use case diagram  
        
      Base on the requirements, we developed multiple use case as you can see in the above figure.
   2. General class diagram  
      A yellow paper with black lines and a few squares

      Description automatically generated  
      Packages:  
      - Package “screen”: stores the main menu GUI and the main menu controller.  
      - Package “display”: stores the controller for the display of stack, queue and list. In this package there are 3 smaller packages for the display of each datatype.  
      - Package “datastructures”: stores our own implementations of 3 data structures: stack, queue and list. There are smaller packages for each type.  
      - Package “array”: stores the random array generator.
2. Specific design
   1. Package “screen”  
        
      This is where we store the main menu and the main menu controller.  
      Each handle button operations handles the operation as its name.
   2. Package “display”  
      A screenshot of a computer program

      Description automatically generated  
      Controllers for stack, queue and list display.  
      The ListDisplayController, QueueDisplayController and the StackDisplayController are inherited from the GeneralDisplayController.  
      In the GeneralDisplayController, there will be some attributes and operations that all three remain controllers have in common, for example: handleBackButtonAction, drawArray,… Except for StackDisplayController, it will override the drawArray method because we draw the stack in vertical way.  
      All the attributes is for creating the GUI.  
      Each operations handle the button as its name.
   3. Package “datastructures”  
      A screenshot of a computer

      Description automatically generated  
      This is where we store our data structures:  
      Queue and Stack will inherited from List because they share the attributes and some operations in common.  
      - List:  
      + Attributes:  
      • elements: we use an array to store the elements in a list  
      • size: the size of a list  
      • maxSize: max size of a list, the default value is 10.  
      + Operations:  
      • Constructor: List  
      • add: add an elements  
      • get: get the element at that position  
      • size: return size of that list  
      • remove: remove an element from the list  
      • contains: check if the list contains that element or not  
      • clear: empty that list  
      • find: return the position of an element  
      • print: print that list  
      • isFull: check if that list is full or not  
      • isEmpty: check if that list is empty or not  
      • toArray: change that list to an array  
      • sort: sort that list with bubble sort algorithm  
      • insertAtIndex: insert an element at that index  
      - Stack: inherited from list  
      + Attributes: same as list  
      + Operations:  
      • peek: return the top element of that stack  
      • push: push an element to the top  
      • pop: remove an element from the top  
      - Queue: inherited from list  
      + Attributes: same as list  
      + Operations:  
      • peek: return the element at the front of queue  
      • peekBack: return the element at the back of queue  
      • enqueue: add an element to the back  
      • dequeue: remove an element from the front
   4. Package “array”

  
This operation generates a random array of the size given.