Class: Quicksort Demo

Fields:

- ARRAY_SIZE (int): The number of colored bars.
- hue (int[]): The array that will be sorted.
- palette (Color[]): Colors in spectral order.
- canvas (Canvas): The panel that displays the colored bars.
- g (GraphicsContext): A graphics context for drawing on the canvas.
- startButton (Button): The button that starts and stops the demo.
- runner (Runner): The thread that runs the recursion.
- running (volatile boolean): Set to true while recursion is running; set to false as a signal to the thread to abort.

Methods:

- main(String[] args): The entry point of the program.
- start(Stage stage): Sets up the GUI and event-handling, fills the palette array with colors in spectral order, and displays the initial contents of the canvas.
- drawSorted(): Redraws the entire canvas with colors in sorted order.
- setHue(int index, int colorNumber): Changes one of the values in the hue array and redraws the corresponding vertical line on the canvas in the new color.
- doStartOrStop(): Called when the user clicks the Start button, starts or stops the recursion thread based on the current state.
- delay(int millis): Inserts a delay and checks the value of the signaling variable running.
- quickSortStep(int lo, int hi): The basic non-recursive QuickSortStep algorithm that rearranges elements of the hue array based on a pivot value, with smaller items to the left and bigger items to the right.
- quickSort(int lo, int hi): The recursive quickSort algorithm that sorts the hue array into increasing order using quickSortStep.
- ThreadTerminationException: An exception thrown to terminate the recursion when the user aborts the sort.

Class: Runner (extends Thread)

Fields:

None

Methods:

- Runner(): Constructs a Runner object and sets it as a Daemon thread.
- run(): The method executed when the thread starts. Randomizes the hue array, calls quickSort to sort the array, and handles thread termination.

```
+--------------+
| start()      |
+--------------+
       |
       V
+--------------+
| drawSorted() |
+--------------+
       |
       V
+---------------------+
| doStartOrStop()      |
+---------------------+
       |
  +----V----+
  | running | (true)
  +----+----+
       |
       V
+---------------------+
| startRunnerThread()   |
+---------------------+
       |
  +----V----+
  | running | (false)
  +----+----+
       |
       V
+---------------------+
| stopRunnerThread()    |
+---------------------+
       |
       V
+---------------------+
| delay(int millis)     |
+---------------------+
       |
  +----V----+
  | running | (false)
  +----+----+
       |
       V
+---------------------+
| ThreadTerminationException |
+---------------------+
       |
       V
+---------------------+
| delay(int millis)     |
+---------------------+
      |
  +----V----+
  | running | (true)
  +----+----+
      |
      V
+----------------------+
| quickSortStep(int lo, |
| int hi)              |
+----------------------+
      |
      V
+----------------------+
| quickSort(int lo,     |
| int
```

To test the above program, you can follow these steps:

1. Compile the program and make sure there are no compilation errors.
2. Run the program.
3. The program window will appear with a canvas showing vertical bars in sorted order and a "Start!" button at the bottom.
4. Click the "Start!" button to start the quicksort algorithm.
5. Observe the bars on the canvas as they change positions during the sorting process. A black bar will mark the empty space.
6. If you want to stop the sorting process before it finishes, click the "Start!" button again.
7. The program will restore the bars to their sorted order.
8. Repeat steps 4-7 to test the start and stop functionality multiple times.
9. Close the program window to end the test.

During the test, observe the following behaviors:
- The bars should change positions and gradually move towards their sorted order.
- The sorting process should be visually displayed on the canvas.
- Clicking the "Start!" button should start and stop the sorting process.
- When the sorting process is stopped, the bars should be restored to their sorted order.

By performing these steps and observing the expected behaviors, you can verify if the program is working correctly.