This program makes use of 10 threads. 1 thread is used as the main thread. Three threads were used for checking the 9 columns, so each of these 3 threads in turn handles 3 columns, since 3 columns x 3 threads gives 9 total columns that are checked, the amount of columns that need to be checked. Three threads were used for checking the 9 rows, so each of these threads, similar to the column threads, handles the checking of 3 rows since 3 rows x 3 threads gives 9 total rows that can be checked, the amount of rows that need to be checked. The final 3 threads were used for checking the 9 3x3 subgrids, so each thread, also similar to the row and column threads, handles 3 3x3 subgrids, since 3 3x3 subgrids x 3 threads gives 9 total subgrids, the amount of subgrids that need to be checked. Using 9 threads in this way allowed me to evenly spread out the tasks that needed to be done, ensuring that each thread only had 3 tasks assigned to them, which is why I went with the use of 9 threads and assigned each thread 3 tasks based on a grouping.

With regards to the organization of the project itself, there are separate classes that handle the checking of columns, rows, and grids. This separation was done in order to make it easier to read and understand how each row, grid, and column was being inspected by allowing me to use more specific names for methods, variables, etc. What the threads do essentially as they scan their columns, rows, and grids, is store any potential errors inside a container class I created called ErrorAndSuggestionContainer, while also marking what values were missing or present in their row, column, or grid in a shared array. The threads also add these objects of this container class into a list of all errors detected that is shared amongst all of the threads. When all of these helper threads are done, the main thread uses the SodokuValidator base class to reduce all of these potential errors into the actual error. More specifically, this is done by using the shared array that the threads updated with missing items and present items in their row, column, or grid, to determine the value that the row, column, and grid of the potential error are all missing; if there is no common missing value in either the row, column, and grid of this potential error, then this potential error is not the actual error, and the number that this potential error conflicted with (its duplicate sibling) will need to be investigated for being an actual error by following the same steps as before. Only when the grid, column, and row of a potential error are found to be missing the same thing does that potential error get marked as the actual error, and the solution becomes that common missing value, else, that potential error gets marked as clean (not a potential error).