The following classes are implemented for the Hydrograph Simulator (In top to bottom order):

1. **ZoomInZoomOutFunctionality:**
   This class is developed to provide the zoom-in and zoom-out functionality when we use the openCV module.
   Apart from the last 'reconstructWindow' method, we can use this idea for developing other modules' zoom-in & zoom-out if any module provides only static image view functionality.
   Here, two zoom scale factors are used for the x and y directions so that in case of extra exploration in one dimension other should not get affected, i.e., provide relaxation where one dimension requires zoom but not the other.
   For finding the next coordinates in both zoom-in and zoom-out, the default incremental zoom percentage is hard-coded as 10%, But by using other data structures, we can use a stepping zoom percentage as we observe in the photo viewer application (Where there are stepping values set. Example 100% -> 110% -> 121% -> 133% ...)
   The fact implemented is Linear interpolation or extrapolation of 10% length from the mouse pointer to all four directions.
   The 'setImageWindowDimensions' method is there to set view window dimensions as per our requirement (by default is fixed at 950x650).

2. **ProcessSurfaceFlowAndFinalGraphOutput:**
   This class is developed for the final computation of surface flow and shows results.
   The 'startCalculating' is there for the computation process. In this method, code computes surface flow over triangular cells by a zero-inertia model where flow exchange between cells across a common boundary is approximated by Manning's equation.
   The triangulation is done especially using 'pA,' which will do triangulation for those nodes which are present inside the closed boundary, hence catchment boundary segments play here great roles, all those nodes which were selected outside the catchment boundary now will no longer be part of triangles, but since intersection points are already made so there is no error in the formation of triangles along with ridge line or valley lines And also no new nodes will be created automatically.

In order to create a neighbor triangle cells track, three types of indicator numbers are used, which represent as follows: 1 -> There exists a neighbor triangle, 0 -> No flow boundary, -1 -> water transfer to valley segment. These numbers are used, and the corresponding index number for the triangular cell or valley segment is entered.

We are assuming triangular cells are in a horizontal plane hence all distances are that way, and its height from the datum point is the average of the z-coordinate of its nodes. The next Idea used here is that simulates the surface flow over the triangular cells and stores the value of the water amount in the valley segments after each time interval. The first version is to develop the unit hydrograph of the catchment. Coming versions can include the rainfall rate as per the provided from the user inputs. We require topological sorting of valley segments from upstream to downstream so that water transfer can be made easily by following these segments' order and will require just transfer from one node to another node in order.

Transfer of water from one valley node to another made using Kinetic wave approximation. By using this method, we will require to have a memory complexity of O(partitionOnTime*partitionOnLength*NoOfValleySegments* TotalNoOfSimulationIntervals), and the same goes for the time complexity. However, we can't optimize on time from this method, but we can certainly do for memory complexity. Let's do this in the following matrices way, matB is useful for the computation of one segment at a time for one simulation interval after that, these results will be saved in matA1 and matA2. Matrices matA1 are initialized for all valley nodes, And matA2 are initialized for all valley segments. Another method used works as follows, Valley segments are of uniform length, and the time required for full transfer of the water from a segment is equal to the one simulation time interval. This is less accurate but can be used for verification purposes where we can find a peak discharge value.

At last ‘deleteFilesFromLocalStorage’ method is for deleting all the files from local storage which were formed during operations.

3. VisualiseCatchmentAreaAndProceedForFinal:

This class is for refining the imputed data and showing the geometry formed from these data.

First of all, All nodes and segments are initialized as per the requirement by setting the active/inactive bit along with stored data. If some nodes are overlapping, then we have to eliminate them, and appropriate action has to be carried out. If the node is inactive, don't put it in the new list, And maintain a

separate list through which we can refine segments node indexing. For this, pre-sum has to be maintained so that subtracting this number from the original node number will give the correct order of the active node number index. Eliminated nodes already get adjusted in the segment's formation, so don't need any special attention here.

Now next operation is to find the intersection point between the segments of high-priority segments and low-priority segments. This is done by iterating over high-priority segments and then checking against low-priority segments whether there is any possibility of a point of intersection. Priority is as follows(high to low): Catchment segments, Valley segments, Ridge segments, and Contour segments.

For making final refined segments of our interest, absolute in nature, i.e., no further distinction between the valley or ridge or catchment or contour nodes. Nodes will be finally added in the order: Catchment Nodes + Valley Nodes + Ridge Nodes + Contour Nodes. And then final segments formed are as follows, Full transfer segments (these are made from valley segments), No transfer segments (these are ridge and catchment segments), And all segments (these are above two categories along with contour segments).

The final geometry is shown using the graph object module, and before proceeding to the next page, the outlet node is mandatory to select, flow through this node will be shown in the hydrograph.

4. **ElevationValueInputFromWindow:**

   This is a sub-window for getting elevation values from the user during the catchment boundary points collection.

   It is compulsory to provide an elevation along with the node selected, hence if we put an errored elevation value, this window will not be closed until we enter a valid elevation value.

5. **CommonSubWindowProperties:**

   Some properties, methods, or variables which are common to some sub-windows are listed in this class so that they can be used by inheriting this class.

   Here, we create a copy of the contour image so that later if we want to reject the new changes made, we can delete the modified image and make available this image at that position to make an appearance as if nothing got saved.

6. **ValleyLinesWindow:**

   From inputs of nodes to segment formation, handled here in this class for the valley lines.

   Here, nodes are selected from the contour nodes, and then the corresponding elevation value is inserted and saved. After getting these valley nodes, we can view the nodes to form corresponding segments. Menu list provided for selecting endpoints of the segment. This way, we can select the network of valley segments that are connected throughout the catchment area.

7. **ElevationValueORvalleyNode:**

   This class is for sub-window when it comes to the elevation value input for the ridge line, we can either select one of the three nearest valley nodes as per the point selected on the contour map as a node in the ridge line or provide an elevation value for that selected point on the contour map.

   We have to select either option, and there is no way to close the window, even by a cross icon on the top right.

8. **RidgeLinesWindow:**

   This class is for handling stuff related to ridgelines, from getting nodes to combining these continuously into segments.

   The elevation of nodes is from the window implemented in the 'ElevationValueORvalleyNode' class. When getting the elevation value is under process, it hinders all other operations in order to make the process synchronous.

9. **ContourPointsWindow:**

   This class is for handling stuff related to contour lines, from getting nodes to combining these into segments.

   All points on the same contour line within the catchment area have to select from the contour map in one go, then the corresponding elevation value of the contour line has to be inserted. This way, all the contour lines present in the contour map within the catchment area have to be mapped.

10. PointsCollectionWindow:

After selecting the scale factor, we have to get the basic geometry of the catchment area, for that, we have four main inputs: Catchment boundary,
valley lines, ridge lines, and contour lines. When these four will combine, we will have a full geometry of the catchment area.

Along with just the landing page for the above four main categories, We also defined the catchment boundary's points collection through the openCV window's design in this class, and its elevation value is from the window defined in the 'ElevationValueInputFromWindow' class. Segments are formed by joining the nodes in the order of selection, and the first catchment node will be used for the completion of the catchment loop and hence the final segment of the catchment boundary.

The provision for transferring the mobility to only newly-created windows. Hence we can't use the button or any other functionality in the root window.

11. InitialInformationCollectionWindow:

This class implements the basic window, where the contour image will be selected from the local storage, and scale points and corresponding scale values will be taken as input.

Copy of the selected image made with a custom name of the file in the same folder of the local storage. Only two points can be selected for scale, and the openCV window will be closed immediately after selecting the second point. The scale value is taken from the entry widget.

12. App:

This class creates a landing root window for the simulator.

"Welcome to the Hydrograph Simulator!"

Related to nodes and segments files generated during operations:

All the node points, irrespective of type(catchment, valley, ridge, or contour node), are stored initially in the integral value by getting its value from mouse handling of the contour map image and multiplying with the zoom scale factor, and elevation value is appended to these x,y coordinates.
And each segment is comprised of four integers “Integer1 Integer2 Integer3 Integer4”, Where 1&2 are representing the end node’s index in the node list, And 3&4 represents which of the node list it belongs to (i.e., catchment, valley, ridge, or contour node list). Various numbering is provided as follows Catchment node list -> 0, Valley node list -> 1, Ridge node list -> 2, and Contour node list -> 3.
After the refinement process, scale factor from the initial inputs multiplied to x and y coordinates.