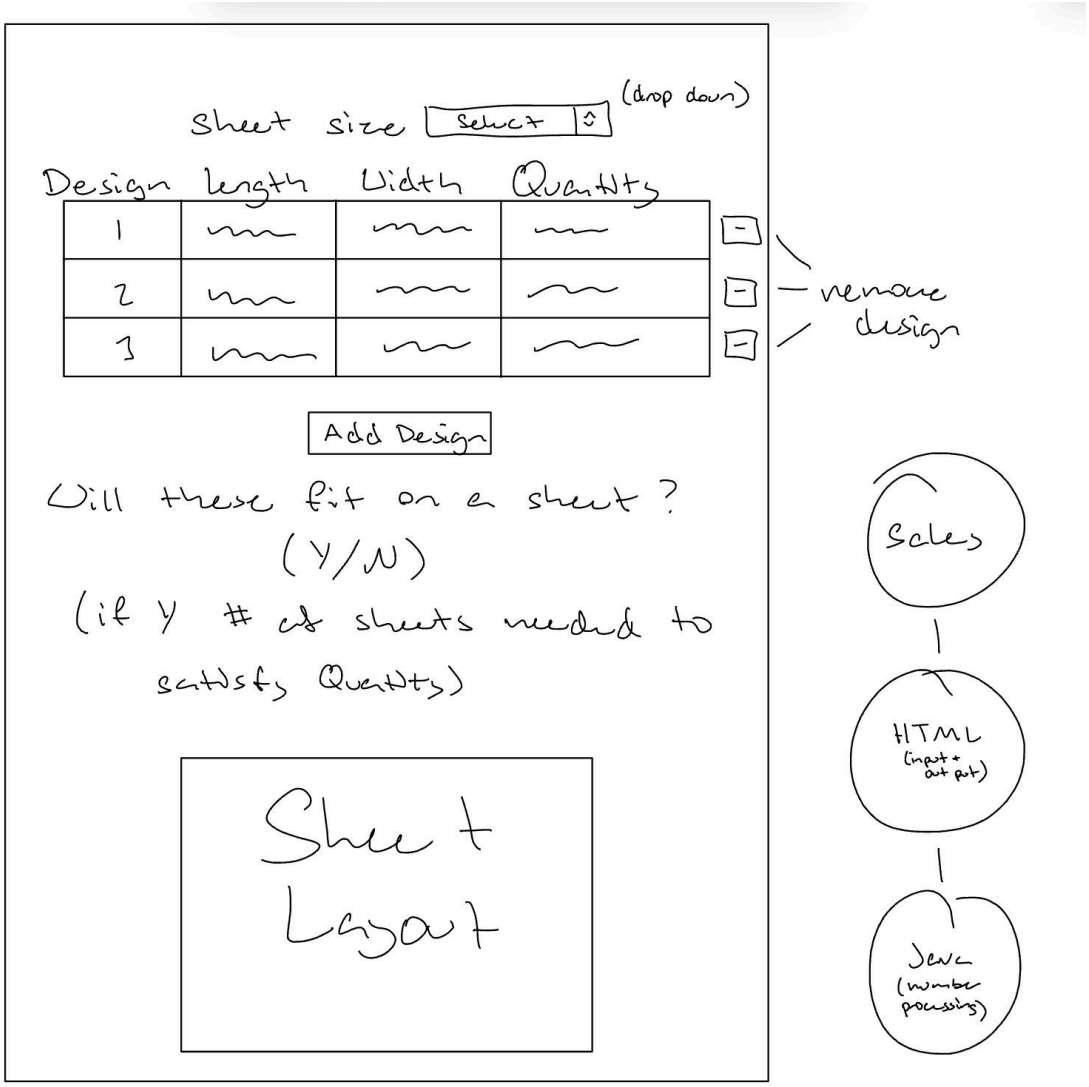
Criterion B: Solution Overview

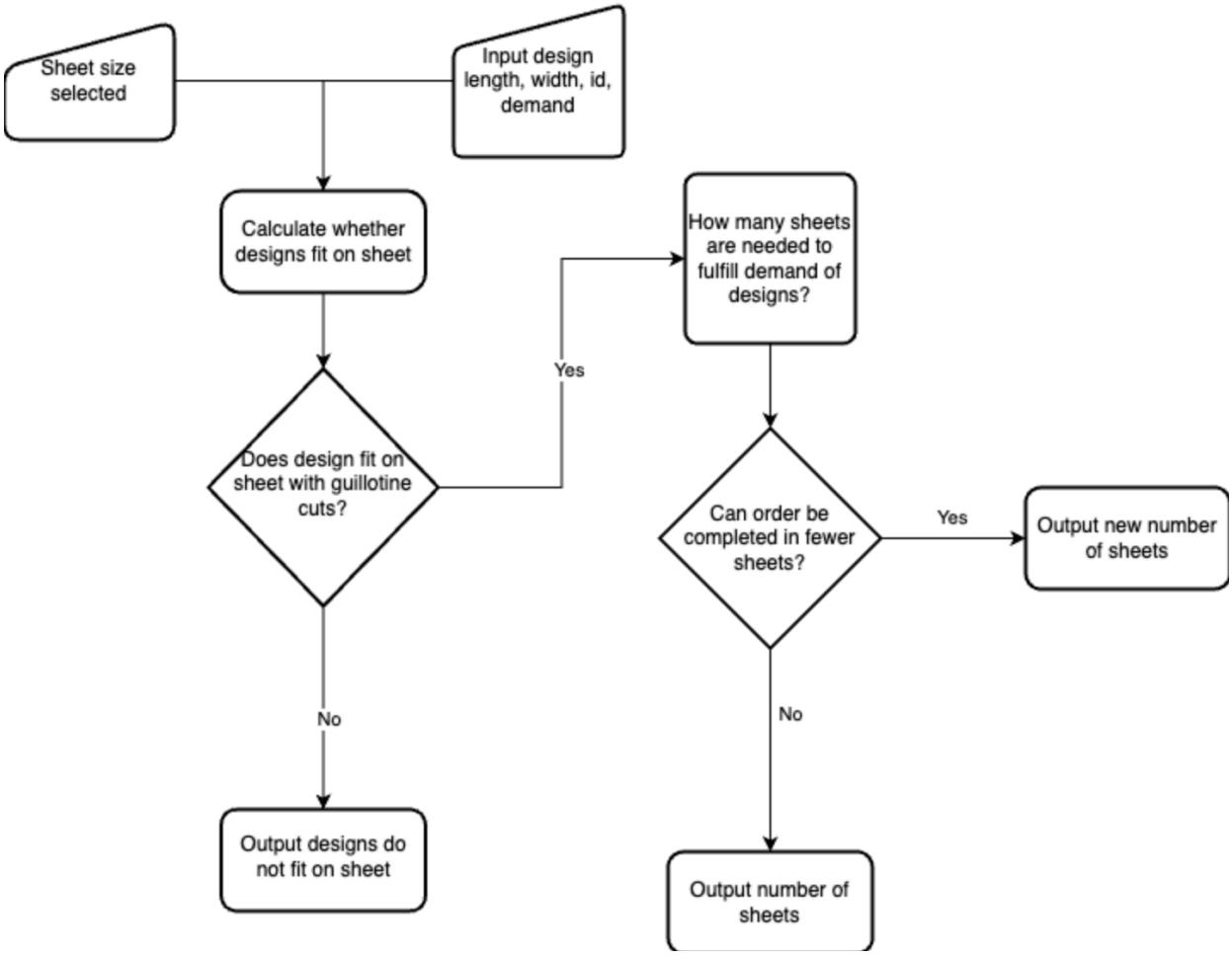
Development Planning

Create a class for the sheets that designs are being placed on, and a class for the designs each with instance variables for their attributes. A third class will be used for the logic and algorithm of the project, involving the recursion that attempts to place all the designs. Since there are two ways to initially cut a sheet once a design is placed and two orientations for each design to go in, the recursive method will have to account for these four possibilities with each design. Some way to pick an ideal design, like the largest area or filling the height or width of a sheet will also need to be added to the class. Finally, a GUI that has an option for sheet size, a place that shows each design being placed on the sheet, a way to add designs, a section for the yes/no response, and a layout of designs upon successful placement of every design.

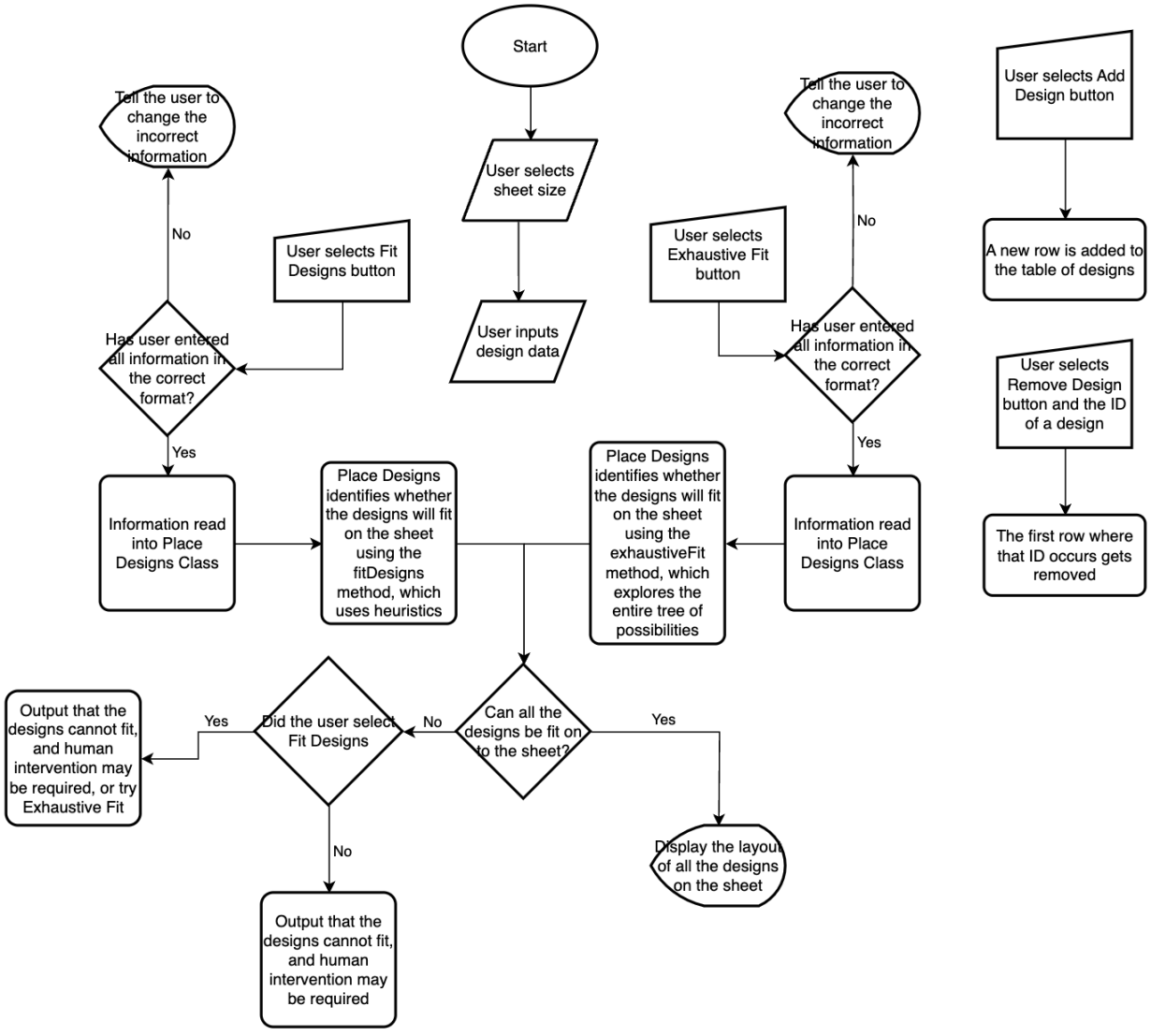
Sketch (Original Idea of GUI layout)



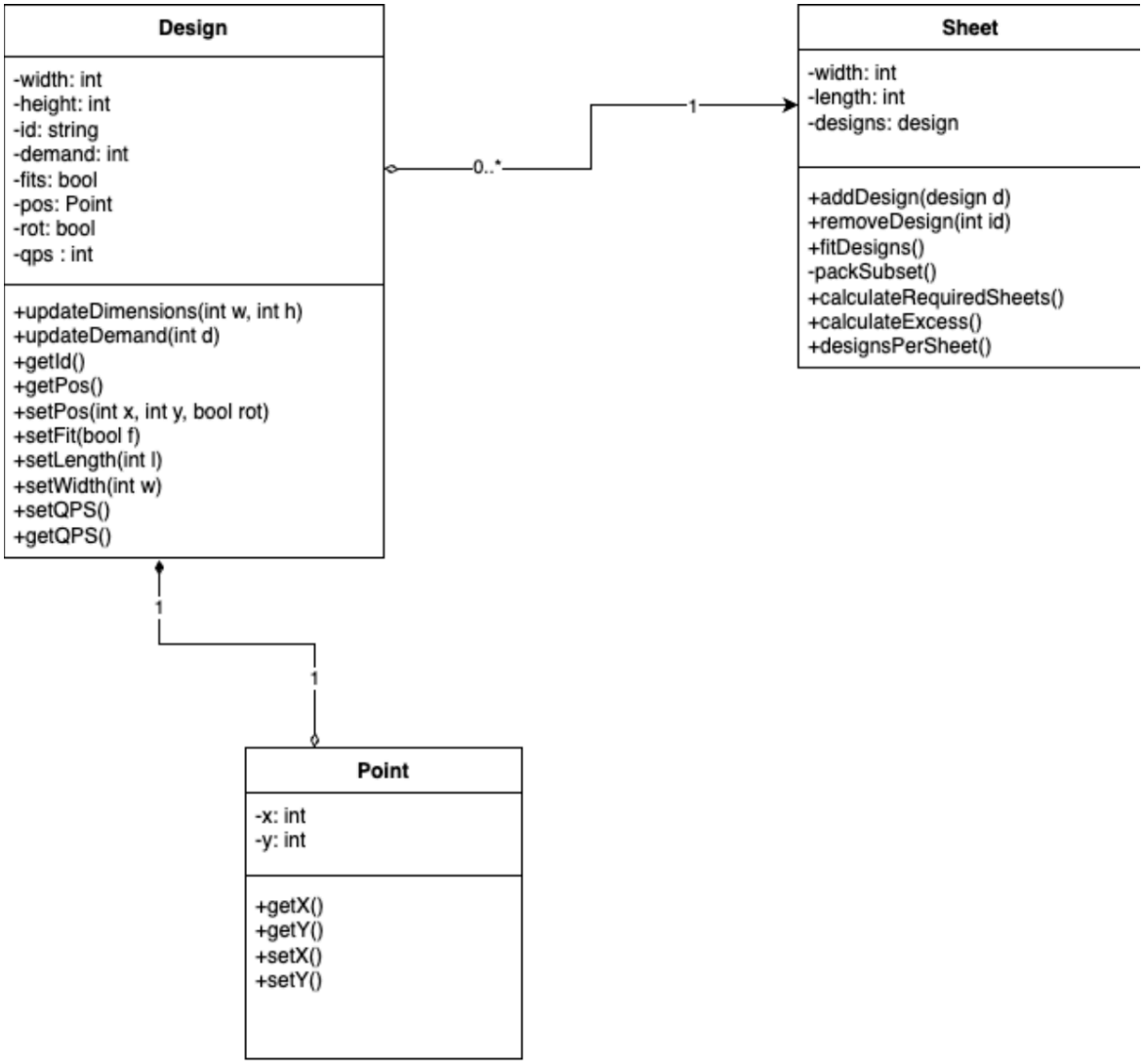
Flowchart of program (Original version)



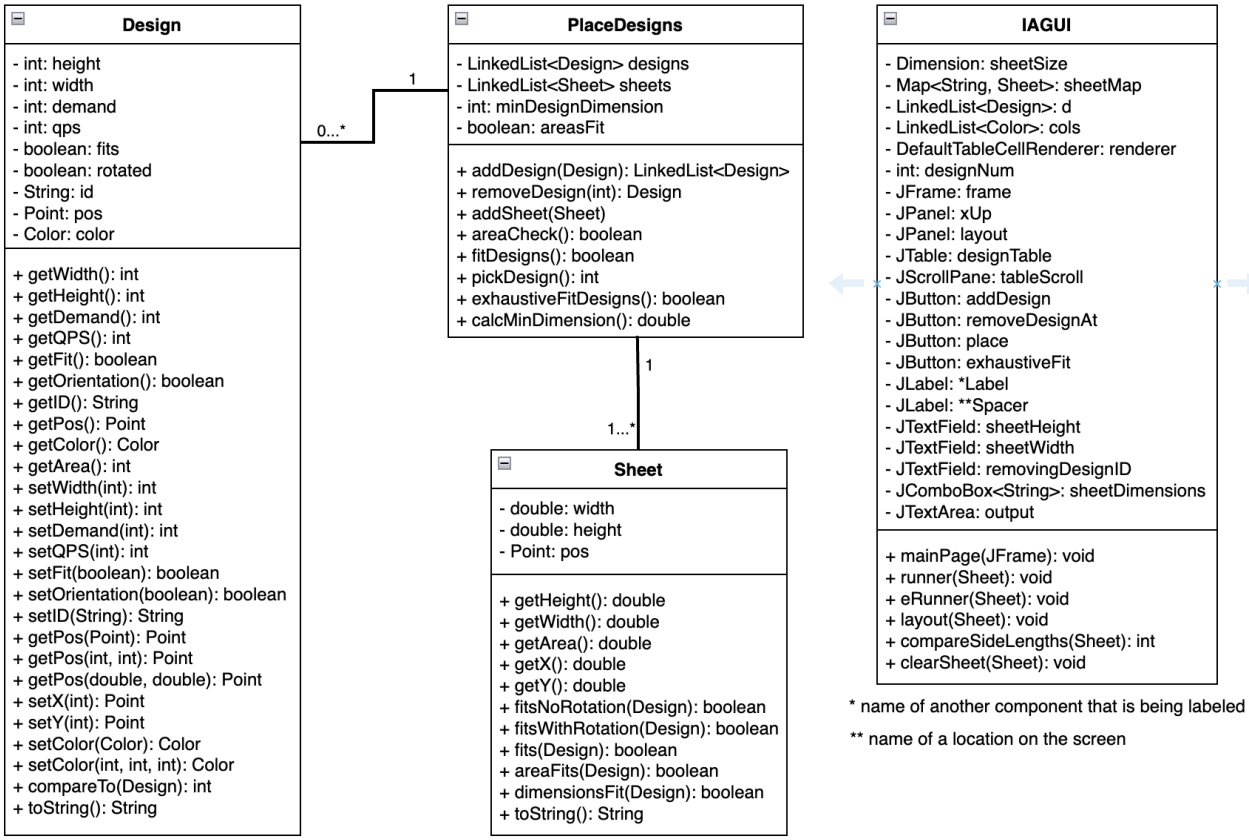
Flowchart of program (Updated)



UML Diagram of Sheet, Design, and Point classes (Original version)



UML Diagram of Classes (Updated)



Original Pseudocode showing Algorithmic Thinking

```
addDesign (id, length, width, demand)
        Design d = new Design(id, length, width, demand)
        MAP.put(id, d)
end method
calculateRequiredSheets()
        Iterator IT = MAP
        NOW = IT.next()
        MAX = NOW
        while(IT.hasNext())
                NOW = IT.next()
                if(NOW.getDemand()/NOW.getQPS() > MAX.getDemand()/MAX.getQPS())
                        MAX = NOW
                end if
        end loop
        return MAX.getDemand()/MAX.getQPS()
```

```
end method
calculateExcess()
        LinkedList XTRA
        Iterator IT = MAP
        NOW = IT.next()
        while(IT.hasNext())
                XTRA.add(NOW.getQPS()*SHEET.calculateRequiredSheets()-NOW.getDemand
        ())
        end loop
        return XTRA
end method
```

Test Case Chart

Actions/Processes to test	**Description**	**Solution/Test**
Ensure designs are added correctly The button is clicked when there are different numbers of designs in the table. Designs are also added after program has run	When the user wants to add a new design to the sheet, they have to click the "Add Design" button which displays a new row in the JTable for the user to input design information.	Multiple different values are entered. Once entered the Design is printed into the console, along with the entire LinkedList.
Algorithm correctly identifies whether designs fit on a sheet. 1 4x4 design. 1 8x3, 2 5x2. 1 11x8, 1 7x7, 1 5x7, 1 12x1, 1 19x1, 1 11x3. (each period ends a test)	This is the main part of my code. Deciding whether designs fit on the sheet is the most important part of the program. Algorithm fits all the given designs on the sheet with guillotines cut or outputs false.	Several data sets are compiled with some sets of designs that fit easily, some sets that don't fit, and some that barely fit.
Algorithm provides an accurate layout of the sheet if all the designs fit. 1 4x4. 1 11x8, 1 7x7, 1 5x7, 1 12x1, 1 19x1, 1 11x3.	If all the designs fit on the sheet, the algorithm can correctly lay out a possible solution for packing all the designs into the specified sheet size.	Several data sets are compiled that fit, and the final layout is assessed for accuracy. This means that no designs are overlapping or going off the sheet.

(each period ends a test)		
Remove design button removes the first row of a given ID. IDs both in the table and not are tested, as are numbers, emojis, and other unusual characters that may be used as IDs.	If the user wants to remove a design there is a place to enter an ID and a button which will remove that design's row from the JTable. Should there be several designs with the same ID, the Remove Design button will only remove the first one.	Several designs are added, then some are removed at random spots, and the final LinkedList is printed to the console.