Tile Designer Project

Objective: You will write a program involving OOP principles that will act as a tile designer program. Tiles will take on the various shapes:

- Quadrilaterals (Rectangle, Square, trapezoid)
- Ellipse
- Circle
- Donut (inner and outer radius)
- Half-Circle

The shapes will have the following attributes colour, length, width, radius, inner/outer radius, etc.

You need to find out the area and perimeter / circumference of each tile. The area will be used to calculate the cost of each tile.

Tiles will be placed onto a floor that resembles a 2D array. Shapes will fit into each cell of the table below. You will use a 2D array to store tile objects. Each cell below will have a maximum length and width. Tiles must fit into the cell or else they will not be allowed to be placed there.

Tile 1	Tile 2	Tile 3
Tile 4	Tile 5	Tile 6
Tile 7	Tile 8	Tile 9

There will be a class called ROOM that will store the 2D array that represents the floor layout. It will also provide functionality to calculate the price of all of the tiles.

The user must be able to:

Initial Menu

- State how big the room is (columns and rows of tiles)
- State the max dimensions (length and width) of a tile
- Enter the price per square foot (all measurements are in feet)

Once the room is initialized, the user will then be given a new menu to do the following:

- Give an option to randomly assign a tile and appropriate dimensions to each cell in the 2D array (floor).
- Create a new tile (give a list of the options to choose from and ask for the parameters. Do not accept it if the dimensions do not fit in with the length/width requirements. If the tile is acceptable, ask the user where to place it: column and row number. Remember that when a customer specifies a col/row that they do not start at index 0 like a programmer would. Do not place a tile down if there is already a tile there.
- Delete an existing tile (specify the row/column). Give an error message if the row/column doesn't exist or if the tile area is empty already.
- Display the whole floor layout with the tile info printed inside.
- Calculate the total cost (or cost so far if not all of the tiles are placed)
- Calculate the total area of all of the tiles placed so far.
- Clear the array of tiles to start over again

Tile Designer Project

- Allow the user to pay for the tiles once every cell is filled with a tile
- Once the user has paid, you can ask the user to create another room or exit
- Have an option throughout that if the user enters in a Q, the program will end

The user should be able to perform these operations in any order, any number of times. Rooms will be composed of tiles of the following types: (one shape per array cell).

Each shape should inherit from an *abstract* tile class. The tile class should have a getArea() function, which returns a double, and a toString() function which returns details about the shape in question in the following form:
"***ShapeName[instanceVariable1:value, instanceVariable2:value, ...]***".

We want the text to be minimal so that it will fit on the screen so we can truncate the text. For example, if you use toString() on a square object that has a side length of 50 cm, and is red the output would look like this:

[SQ s:10 c:Red]

A rectangle that is blue and has a length of 30 cm and a width of 80cm would look like this:

[REC L:30 W:80 c:blue]

Use this following format for all other shapes when they are printed to represent the 2D array

EXAMPLE:

If the user specifies a room that is 2x2, initially the floor will look like this if it is printed without any tiles placed:

[EMPTY] [EMPTY]

[EMPTY] [EMPTY]

If you add a square, the program should find out the dimensions of the shape, and the colour. Then the program should ask you where you want to put it. If the user specifies row 1 column 1, then the room will look like this when you print it:

[SQ s:10 c:red] [EMPTY]

[EMPTY] [EMPTY]

BONUS:

To achieve a level 4-/4/4+ under *Application* you will must improve upon the basic of the program:

Level 4-/4 ideas:

→ prompt the user appropriately if the tile location is already occupied when you try to place a new tile. Do not allow them to do so.

→ search the 2D array for a specific tile and return where it is located (remember that to the customer the indexes do not start at 0. They start at 1.

Tile Designer Project

→ have an option to print out an entire row or column only of tiles

→ make sure that all tiles printed line up perfectly. To make it line up everytime figure out the length of the longest String and use String.format() to make sure they all get the exact same space. This will require some research.

Example (instead of this):

[SQ s:10 c:red] [EMPTY]

[EMPTY] [EMPTY]

Example (it should look like this):

[SQ s:10 c:red] [EMPTY]

[EMPTY] [EMPTY]

Level 4+/4++ ideas:

Determine if an entire row or column is empty

Sort an entire row or column by colour, area, perimeter / circumference (Arrays have a build in sort function but there is a special way to sort the objects based on attributes.

OVERVIEW OF CLASSES

- Tile (abstract class): should contain attributes like cost per square foot and other members that will be inherited by the other tile shapes
- For the specific tiles below make sure to follow an appropriate class hierarchy based on HAS-A and IS-A relationships. The tiles below should keep track of their own colour, area, perimeter, toString() method, and cost.
    - Quadrilaterals (Rectangle, Square, trapezoid)
    - Ellipse
    - Circle
    - Donut (inner and outer radius)
    - Half-Circle

- Room class that contains a 2D array that stores the tile objects. It should contain methods to add, remove, and the other functions that are described in the menu.

- You will create a separate class file that includes black box testing. One file that will bypass the menu and allow the programmer to create all objects and directly call the methods. All features must be tested. Have pauses in between so that the output can be analyzed. To do this you can ask for an input with text saying "Press Enter to show the next case". You should also include text that shows exactly what your test case was.

You will include a UML diagram in your report that shows the relationship between all ten classes.

Tile Designer Project

Categories	**Level 1** **(50 – 59%)**	**Level 2** **(60 – 69%)**	**Level 3** **(70 – 79%)**	**Level 4** **(80 – 100%)**
Communication Code Design Program Header Comments Variable names Method names Indentation / 10	Variable names and method names along with a description of methods and any required parameters are missing crucial elements and do not communicate a clear plan	Variable names and method names along with a description of methods and any required parameters is communicated with limited clarity	Variable names and method names along with a description of methods and any required parameters is communicated with only minor omissions or errors	Variable names and method names along with a description of methods and any required parameters is communicated clearly
Thinking Program Testing Black Box Testing /10	Program testing was missing crucial elements.	Program testing was insufficient to conclude that the program runs properly	Program testing missed covered a considerable number of possible cases and was summarized in a logical way	Program testing was thorough, and summarized in a logical and succinct way
Application Program Execution Error Checking / 25	Program doesn't run properly, or the output has serious errors.	Program runs properly. Output has errors. The provided client code will not run properly, but student provided client does run properly	Program runs properly. Output is correct with only minor errors. The provided client code will run properly	Program runs properly as is. The output is identical or superior to the sample provided. The provided client code will run properly with an additional feature.
Knowledge UML Diagram Programming Concepts <u>Focus:</u> Inheritance Polymorphism Abstract Class / 25	Few of the programming concepts from the unit are used properly	Some of the programming concepts from the unit are used properly	Most of the programming concepts from the unit are used properly	Many or all of the programming concepts from the unit are used properly to maximize the efficiency of the code