CS102 – Algorithms and Programming II
Programming Assignment 3
Spring 2025

ATTENTION:

- Compress all of the Java program source files (.java) files into a single zip file.
- The name of the zip file should follow the below convention, where you replace the variables “Sec1”, “Surname” and “Name” with your actual section, surname and name:

  CS102_Sec1_Asgn3_Surname_Name.zip
- Upload the above zip file to Moodle by the deadline (if not significant points will be taken off). You will get a chance to update and improve your solution by consulting to the TAs and tutors during the lab. You have to make the preliminary submission before the lab day. After the TA checks your work in the lab, you will make your final submission. Even if your code does not change in between these two versions, you should make two submissions for each assignment.

GRADING WARNING:

- Please read the grading criteria provided on Moodle. The work must be done individually. Code sharing is strictly forbidden. We are using sophisticated tools to check the code similarities. The Honor Code specifies what you can and cannot do. Breaking the rules will result in disciplinary action.

Calculating Shapes

For this assignment, you will implement a Java console application that can store various 2D and 3D shapes based on their attributes to calculate areas and volumes where applicable.

You should implement the classes for the shapes in a hierarchical manner. An abstract *GeometricShape2D* class should include the abstract methods *float calculateArea()* and *void printInfo()*. The classes that extend GeometricShape2D will implement these methods: *calculateArea()* will calculate the area of the shape based on the type of the shape (rectangle, circle, triangle, etc.); *printInfo()* method will give information about the shape; printing its dimensions based on shape type (for example, width and height for rectangle, radius for circle, etc.) and the type of the shape. You may need to keep different variables for each type of shape to store the shape’s dimensions as float variables.

You should also implement an abstract *GeometricShape3D* class that extends the *GeometricShape2D* class. *GeometricShape3D* should include the abstract method *float calculateVolume()*, which will be used for calculating the volume based on the shape type and dimensions. Since *GeometricShape3D* inherits the parent class’s methods, any class that extends *GeometricShape3D* will need to implement the *calculateArea()* and *printInfo()* methods, as well as the *calculateVolume()* method.

You will implement the classes for the following shapes:

- Classes that extend *GeometricShape2D*:
    - Rectangle
    - Circle
    - Square
    - Ellipse
    - Equilateral Triangle
    - *MultiShape2D* (explained below)*

- Classes that extend *GeometricShape3D*:
    - Cuboid
    - Sphere
    - Cylinder
    - Cube
    - Pyramid

Try to utilize inheritance for your subclasses too. For example, Square is a special case of a Rectangle, so why not extend it to use the already implemented methods of the Rectangle class?

You will implement a *MultiShape2D* class that contains multiple *GeometricShape2D* objects in it. The area of this shape is calculated as the sum of the areas of the shapes included in this object. For *MultiShape2D*, the printInfo() method should list the shapes (with dimensions) included in this multi-shape. A multi-shape can include other multi-shapes in it; in this case, it is listed in the *printInfo()* method as a multi-shape with its total area.

For example, a multi-shape can list the following information using the *printInfo()* method:

- Circle, radius 3.
- Circle, radius 5.
- Multi-shape, area 200.
- Square, edge length 10.

A multi-shape initially has no shapes in it. Include a *void addShape(GeometricShape2D addedShape)* method to attach new shapes to a multi-shape. *MultiShape2D* class should also contain a method *void mergeShapes()*, which destroys all the shapes in this multi-shape to construct one square shape with an area equal to the sum of areas of the shapes included. For example, suppose the multi-shape has the following shapes in it:

- Rectangle, 3 by 5. (Area = 15)
- Square, edge length 5. (Area = 25)
- Multi-shape, area 60. (Area = 60)

After calling the *mergeShapes()* method, the same multi-shape will have the following shape only:

- Square, edge length 10. (Area = 100)

You may round the edge length of the square to the closest integer in case the total area of the shapes is not a perfect square. If there are no shapes added into the multi-shape, the mergeShapes method does nothing; i.e., it does not construct a square.

For this lab assignment, you are not allowed to use the Java Collections Framework, so you cannot use an ArrayList to keep *GeometricShape2D* of the multi-shape. You should store the shapes as an object array. The size of this array should be exactly equal to the number of shapes in the multi-shape. As we add a new shape using the *addShape* method, you should create a larger array to contain the newly added shape and carry any of the existing shapes into this larger array. Similarly, when we use the *mergeShapes* method, a new array with length one should be created.

You will also store all the shapes created in your application to operate on them; you need to keep them as object arrays. Make sure you utilize polymorphism to iterate over your different shape types using the common parent type. For example, you should keep your 2D shapes in an array of type MultiShape2D and your 3D shapes in an array of type MultiShape3D.

For this assignment, you should not use any of the classes that implement the Shape interface in Java; you should implement your own shape classes.

You should implement a menu-driven program that supports the following functionality:

1. **Create and Store a New Shape:** If the user selects this option, you will display the possible list of shapes. Then, based on the type of shape the user chooses, the program should input the required parameters. For example, if the selected shape is a square, you just input one length parameter, but if it is a cuboid, you need to input its width, height, and depth. If a multi-shape is selected, then you will allow the user to create and add new shapes to this multi-shape. For example, after creating a multi-shape, the user may add a couple of different circles and a square to it; however, in this option, we cannot create a new multi-shape in the multi-shape (we will use the second option for this purpose).

2. **Add an Existing Shape to a Multi-shape:** This option is enabled when there are at least two 2D shapes stored, and at least one of them is a multi-shape. The user should select which 2D shape to add to the selected multi-shape. To this end, you may first list the multi-shapes so that the user chooses which one to add the shape to; then, you can list the 2D shapes (except for the initially selected multi-shape) that the user can add to the selected multi-shape. Note that this option only applies to the 2D shapes and 3D shapes are not listed or added. Suppose our application contains one circle and a multi-shape; after adding the circle to the multi-shape, the application will only have the multi-shape, and the circle is now kept inside the multi-shape. With this option, we can also add multi-shapes to other multi-shapes.

3. **List All Shapes:** This option lists all the shapes, printing their information using the *printInfo* method. Then, allow the user to choose a specific shape to give more detail about it. If the user selects a 2D shape, this will print its area. If the selected shape is 3D, this will print both its area and volume. If the selected shape is a multi-shape, this will print the shapes contained in that multi-shape. While showing the details, any multi-shape within the multi-shape should also print its details. For example, suppose we have the following list when we first enter this option, showing the details for the multi-shape will be as follows:

```
2D Shapes:

[0] Circle, radius 5.
[1] Square, edge length 10.
[2] Circle, radius 2.
[3] Multi-shape, area 100.
[4] Multi-shape, area 12.

3D Shapes:

[5] Cuboid, 4 by 6 by 8.
[6] Sphere, radius 7.

Do you want details for a specific shape?
Enter shape index or -1 to return: 3

Multi-shape, area 100:
   -  Square, edge length 5.
   -  Square, edge length 4.
   -  Multi-shape, area 50:
         -  Square, edge length 5.
         -  Multi-shape, area 25:
               -  Square, edge length 5.
   -  Square, edge length 3.
```

You should indent the details of any shape within the multi-shapes so that it is easy to understand which shape is included in which multi-shape. You can print the information using the *printInfo* method for the shapes; no need to print area specifically for non-multi-shape objects.

4. **Merge Multi-shapes:** This option calls the *mergeShapes* method for all multi-shapes. After calling this method, all the multi-shapes will contain a single square. This will get rid of any nested multishapes. For example, for the multi-shape below:

```
Multi-shape, area 100:
   -  Square, edge length 5.
   -  Square, edge length 4.
   -  Multi-shape, area 50:
         -  Square, edge length 5.
         -  Multi-shape, area 25:
               -  Square, edge length 5.
   -  Square, edge length 3.
```

After calling the merge operation, only the top-level multi-shape will remain with a square of area equal to 100, and nested multi-shapes will be gone:

```
Multi-shape, area 100:
   -  Square, edge length 10.
```

5. **Edit Shape:** Editing an existing shape is only applicable to non-multi-shapes. If the user selects this option, you will list the available shapes for editing, and the user will select one of them. Then, the program should ask for the new dimensions of the corresponding shape to change its existing values.

Preliminary Submission: You will submit an early version of your solution before the final submission. This version should at least include the following:

- All the shape classes should be implemented.
- The logic for adding new shapes in the menu-driven program should be completed.
- Do not forget to test your implemented shape classes!

You will have time to complete your solution after you submit your preliminary solution. You can consult the TAs and tutors during the lab. Do not forget to make your final submission at the end. Even if you finish the assignment in the preliminary submission, you should submit for the final submission on Moodle.

Not completing the preliminary submission on time results in 50% reduction of this assignment's final grade.