BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

K. K. BIRLA GOA CAMPUS
FIRST SEMESTER 2024-2025

CS F213 (Object Oriented Programming) – Lab 11 Assignment, Date: 29/10/24

Problem Title: Geometric Figures Management Using Maps

Objective:

You are tasked with developing a Java application that manages a collection of geometric figures. You will implement three classes:

1. **Figure**: Represents a geometric figure.
2. **FigureManager**: Manages the collection of figures using Maps.
3. **L11_Q1_Main**: Contains the `main` method, reads figures from terminal input into an array.

Requirements

Figure Class

Define a concrete class **`Figure`** with the following specifications:

- **Properties**:
  - `String name`: The type of figure. Valid values are `"Triangle"`, `"Rectangle"`, or `"Rhombus"`.
  - `double dim1`, `double dim2`: Dimensions of the figure.
    - For a **Triangle**:
      - `dim1`: Base
      - `dim2`: Height
    - For a **Rectangle**:
      - `dim1`: Width
      - `dim2`: Height
    - For a **Rhombus**:
      - `dim1`: Length of the first diagonal
      - `dim2`: Length of the second diagonal
- **Constructor**:
  - `Figure(String name, double dim1, double dim2)`: Initializes the figure with the given name and dimensions.
- **Methods**:

- `double getArea()`: Calculates and returns the area of the figure based on its type (`name`).
    - **Triangle**: Area = `(base * height) / 2`
    - **Rectangle**: Area = `width * height`
    - **Rhombus**: Area = `(diagonal1 * diagonal2) / 2`

FigureManager Class

Define a class **`FigureManager`** that manages an array of `Figure` objects.

- **Constructor**:
    1. `FigureManager(Figure[] figures)`: Initializes the manager with an array of figures.
- **Methods**:
    1. **Count Figures by Type**
        - **Signature**: `void countFigures()`
        - **Description**:
            - Uses a `Map<String, Integer>` to find the number of figures of each type present in the array.
            - Prints the counts for each figure type: `"Triangle"`, `"Rectangle"`, and `"Rhombus"`.
    2. **Find Largest Areas**
        - **Signature**: `void findLargestAreas()`
        - **Description**:
            - Uses a `Map<String, Double>` to find the largest area among figures of each type.
            - If a figure type is not present, store `-1` for that type.
            - Prints the largest area for each figure type.
    3. **Count Figures by Area**
        - **Signature**: `void countFiguresByArea()`
        - **Description**:
            - Uses a `Map<Double, Integer>` to count how many figures have the same area.
            - Specifically, prints the number of figures that have the areas `20.0` and `30.0`.
    4. **Replace Figures Based on Precedence**
        - **Signature**: `void replaceFigures()`
        - **Description**:
            - Uses a `Map<Double, Figure>` to store figures keyed by their area.
            - If an area key already exists, replace the existing figure based on the following precedence rules:
                - A **Triangle** can replace only another **Triangle**.
                - A **Rectangle** can replace both a **Rectangle** and a **Triangle**.
                - A **Rhombus** can replace all types of figures.

        - After processing, prints the figure type that has the area `20.0` and `30.0`.
        - If there is no figure with an area of `20.0` or `30.0`, prints `"Key not present."`

5. **Calculate Average Areas by Figure Type**
    - **Signature**: `void calculateAverageAreas()`
    - **Description**:
        - Calculates and prints the average area for each figure type (`"Triangle"`, `"Rectangle"`, `"Rhombus"`) present in the array.
        - Uses Maps to accumulate the total area and count of figures for each type.
        - Handles cases where a figure type might not be present.

L11_Q1_Main Class (This class is provided with the assignment)

Define a class `L11_Q1_Main` containing the `main` method.

- **Functionality**:
    - Reads the number of figures (**n**) from the terminal.
    - Reads **n** figures from terminal input, each specified as `name`, `dim1`, and `dim2`.
    - Stores the figures in an array of `Figure` objects.
    - Creates an instance of `FigureManager` with the array.
    - Calls each method in `FigureManager` to demonstrate their functionality.

Example 1:

Input: The input given below corresponds to 5 figures. The first line contains the number of figures in the array. The remaining lines contain dim1 and dim2 of each figure.

```
5
Triangle 5 8
Rectangle 5 6
Triangle 6 10
Rectangle 8 7
Rhombus 8 7
```

Output: The output corresponds to the output of the five methods in the `FigureManager` class.

```
Figure counts:
Triangles: 2
Rectangles: 2
Rhombuses: 1
Largest areas:
Triangle: 30.0
Rectangle: 56.0
```

```
Rhombus: 28.0
Figures with area 20.0: 1
Figures with area 30.0: 2
Figure with area 20.0: Triangle
Figure with area 30.0: Rectangle
Average areas:
Triangle: 25.0
Rectangle: 43.0
Rhombus: 28.0
```

Explanation:
First, the `countFigures` method prints the number of Triangles, Rectangles and Rhombuses in the array. Next, the `findLargestAreas` method prints the largest area for a Triangle, a Rectangle and a Rhombus in the array. The third output corresponds to the output of the `countFiguresByArea` method that prints the number of figures with areas 20.0 and 30.0. The fourth output corresponds to the output of the `replaceFigures` method that replaces figures corresponding to an area using the given precedence rules. The final output corresponds to the `calculateAverageAreas` method that prints the average areas of Triangle, Rectangle and Rhombus.

Instructions:

Follow the steps given below to complete the OOP lab problem.

Step 1: Read the Lab Question

Read and understand the Lab problem given above.

Step 2: Give execute permission to the executable files

Use the following command to give execute permission to the executables.

```
:~$ chmod +x RunTestCase CreateSubmission
```

Step 3: See the input and the expected output

Use the following command to see the input and the expected output for test case T1. Note that L11 refers to Lab 11, Q1 refers to Question 1 and T1 refers to test case 1.

```
:~$ ./RunTestCase L11 Q1 YourBITSId T1
```

Use your own (13 character) BITS Id in place of YourBITSId in the above command. Type your BITS Id in upper case (capital letters). Ensure that you enter your BITS Id in 202XA7PSXXXXG format.

Run the command from within the folder containing the executables and the java files.

Step 4: Modify the solution java file

Modify the solution Java file(s) to solve the question given above. Repeat Step 3 until all the test cases are passed. The test cases are numbered T1, T2, etc.

Step 5: Passing evaluative test cases

There are evaluative test cases whose expected output is hidden. The evaluative test cases are numbered ET1, ET2, etc. The lab marks will be based on the evaluative test cases. Use the following command to check whether your solution passes the evaluative test cases.

```
:~$ ./RunTestCase L11 Q1 YourBITSId ET1
```

Ensure that your *final* solution passes all the evaluative test cases ET1, ET2, etc. The expected output of the evaluative test cases are *hidden*.

Step 6: Create submission zip file

Use the following command to create the submission zip file.

```
:~$ ./CreateSubmission L11 YourBITSId
```

The first command line argument must correspond to the lab number and the second command line argument must be your 13 character BITS id.

The above command will create a zip file. The command will also list the evaluative test cases that your solution program has passed.

Step 7: Upload submission zip file

Upload the submission zip file created in Step 6. Do not change the name of the zip file or modify any file inside the zip file. You will not be awarded marks if the zip file is tampered with in any manner.