**PROJECT APPROACH**

Throughout the project, the classes were extended and modified to handle the new shape types and implement the required functionality. The lexer and parser were used to parse the scene definition file and generate the corresponding objects representing the scene and its images. The DrawingPanel class provided the graphical display of the scene, and the Main class coordinated the entire process.

**The steps of approach are listed below in detail:**

* **HollowPolygon Program:** The initial program provided was the HollowPolygon class, which defined a hollow polygon shape. This class served as the starting point for the project.
* **Lexer and Parser**: The next step was to implement the lexical analyzer (Lexer) and parser (Parser) classes to read and analyze the scene definition file. The Lexer class was responsible for tokenizing the input file, while the Parser class parsed the tokens according to the grammar rules.
* **Scene and Image Classes**: The Scene class was implemented to represent the scene that contains multiple images. It provided methods to create the drawing window and add images to the scene. The Image class was also created as an abstract base class for different image types.
* **Extending Image Classes**: The Polygon\_ class was introduced as an abstract base class for all polygon-shaped images. The RightTriangle class was implemented by extending HollowPolygon to represent a right triangle shape. The Parallelogram, RegularPolygon, and Isosceles classes were then implemented by extending HollowPolygon to handle the respective shape types.
* **DrawingPanel and Main Classes**: The DrawingPanel class was created to define a panel for drawing the images. It used the Graphics object to draw the images on the panel. The Main class was implemented as the entry point of the program. It instantiated the Parser, parsed the scene definition file, created the Scene object, and called the draw method to display the scene.

**A discussion of lessons that I have learnt from the above project is presented below:**

**Lessons Learnt:**

**Modular Design:** A modular design approach was followed by breaking down the project into smaller classes and components. This allowed for easier understanding, maintenance, and extensibility of the codebase.

**Lexer and Parser Implementation**: Implementing the lexer and parser was crucial for parsing the scene definition file and generating the corresponding objects. This project reinforced the importance of proper error handling, such as throwing lexical and syntax errors, to provide meaningful feedback to users.

**Inheritance and Polymorphism**: The use of inheritance and polymorphism was demonstrated through the abstract base classes (Image and Polygon\_) and their subclasses (HollowPolygon, RightTriangle, etc.). This approach enabled code reuse, flexibility, and the ability to handle different shapes within a unified framework.

**Graphics and GUI Programming:** The project involved working with graphical elements and GUI programming using Java's AWT and Swing libraries. This allowed for the creation of a visual representation of the scene and the display of various shapes within a window.

**2.PROJECT IMPROVEMENTS**

**Following the project , I suggest the following Improvements that could be made to better the project.**

* **Error Handling and Validation:** error handling mechanisms could be enhanced by providing more detailed error messages, including specific line numbers and descriptions of errors encountered during parsing. Additionally, consider implementing input validation to ensure that the scene definition file adheres to the grammar rules.
* **Enhanced User Interface:** Improving the user interface by adding features such as buttons or menus to allow users to interact with the program, such as loading different scene definition files or modifying the scene dynamically.
* **Additional Shape Types**: Expanding the project by implementing support for more shape types, such as circles, ellipses, or custom polygons. This can be achieved by extending the base classes and implementing the corresponding drawing logic for each new shape.
* **Optimizations and Performance:** When rendering and optimizing the process to enhance performance, especially when dealing with a large number of images or complex scenes , techniques like caching, efficient redrawing, and utilizing hardware acceleration should be considered for smoother graphics rendering.
* **Documentation and Comments**: comprehensive documentation and comments should be provided throughout the codebase to improve code readability and make it easier for others (or future developers) to understand and maintain the project.
* **Unit Testing**: unit tests should be implemented to verify the correctness of the individual components, classes, and methods. This can help identify and fix issues early on and ensure the overall stability and reliability of the project.

**TEST CASES**

**1.parallelogram**

import java.awt.\*;

*// Abstract base class for all polygon classes*

abstract class Polygon\_ extends Image {

private int vertexCount;

private Polygon polygon;

*// Constructor that initializes color and vertexCount of a polygon*

public Polygon\_(Color *color*, int *vertexCount*) {

super(color);

this.vertexCount = vertexCount;

}

*// Creates a polygon object given its vertices*

public void createPolygon(int[] *x\_points*, int[] *y\_points*) {

polygon = new Polygon(x\_points, y\_points, vertexCount);

}

*// Draws polygon using polygon object*

@Override

public void draw(Graphics *graphics*) {

colorDrawing(graphics);

drawPolygon(graphics, polygon);

}

*// Abstract method to be implemented by subclasses*

*// Draws the polygon using the provided graphics object and polygon*

abstract public void drawPolygon(Graphics *graphics*, Polygon *polygon*);

}

*// Subclass representing a hollow polygon*

class HollowPolygon extends Polygon\_ {

*// Constructor that calls super constructor*

public HollowPolygon(Color *color*, int *vertexCount*) {

super(color, vertexCount);

}

*// Draws the hollow polygon using the provided graphics object and polygon*

@Override

public void drawPolygon(Graphics *graphics*, Polygon *polygon*) {

*// Set the color of the polygon*

graphics.setColor(getColor());

*// Draw the outline of the polygon*

graphics.drawPolygon(polygon);

}

}

*// Class representing a parallelogram shape*

class Parallelogram extends HollowPolygon {

*// Constructor that calls super constructor*

public Parallelogram(Color *color*, Point *point*, Point *secondPoint*, int *offset*) {

super(color, 4);

int[] x\_points = { point.x, secondPoint.x, secondPoint.x - offset, point.x - offset };

int[] y\_points = { point.y, secondPoint.y, secondPoint.y, point.y };

createPolygon(x\_points, y\_points);

}

}

**2.Isosceles**

import java.awt.\*;

*// Abstract base class for all polygon classes*

abstract class Polygon\_ extends Image {

private int vertexCount;

private Polygon polygon;

*// Constructor that initializes color and vertexCount of a polygon*

public Polygon\_(Color *color*, int *vertexCount*) {

super(color);

this.vertexCount = vertexCount;

}

*// Creates a polygon object given its vertices*

public void createPolygon(int[] *x\_points*, int[] *y\_points*) {

polygon = new Polygon(x\_points, y\_points, vertexCount);

}

*// Draws polygon using polygon object*

@Override

public void draw(Graphics *graphics*) {

colorDrawing(graphics);

drawPolygon(graphics, polygon);

}

*// Abstract method to be implemented by subclasses*

*// Draws the polygon using the provided graphics object and polygon*

abstract public void drawPolygon(Graphics *graphics*, Polygon *polygon*);

}

*// Subclass representing a hollow polygon*

class HollowPolygon extends Polygon\_ {

*// Constructor that calls super constructor*

public HollowPolygon(Color *color*, int *vertexCount*) {

super(color, vertexCount);

}

*// Draws the hollow polygon using the provided graphics object and polygon*

@Override

public void drawPolygon(Graphics *graphics*, Polygon *polygon*) {

*// Set the color of the polygon*

graphics.setColor(getColor());

*// Draw the outline of the polygon*

graphics.drawPolygon(polygon);

}

}

*// Class representing an isosceles shape*

class Isosceles extends HollowPolygon {

*// Constructor that calls super constructor*

public Isosceles(Color *color*, Point *point*, int *height*, int *width*) {

super(color, 3);

int[] x\_points = { point.x, point.x - width/2, point.x + width/2 };

int[] y\_points = { point.y, point.y + height, point.y + height };

createPolygon(x\_points, y\_points);

}

}

**3. regular\_polygon**

import java.awt.\*;

*// Abstract base class for all polygon classes*

abstract class Polygon\_ extends Image {

private int vertexCount;

private Polygon polygon;

*// Constructor that initializes color and vertexCount of a polygon*

public Polygon\_(Color *color*, int *vertexCount*) {

super(color);

this.vertexCount = vertexCount;

}

*// Creates a polygon object given its vertices*

public void createPolygon(int[] *x\_points*, int[] *y\_points*) {

polygon = new Polygon(x\_points, y\_points, vertexCount);

}

*// Draws polygon using polygon object*

@Override

public void draw(Graphics *graphics*) {

colorDrawing(graphics);

drawPolygon(graphics, polygon);

}

*// Abstract method to be implemented by subclasses*

*// Draws the polygon using the provided graphics object and polygon*

abstract public void drawPolygon(Graphics *graphics*, Polygon *polygon*);

}

*// Subclass representing a hollow polygon*

class HollowPolygon extends Polygon\_ {

*// Constructor that calls super constructor*

public HollowPolygon(Color *color*, int *vertexCount*) {

super(color, vertexCount);

}

*// Draws the hollow polygon using the provided graphics object and polygon*

@Override

public void drawPolygon(Graphics *graphics*, Polygon *polygon*) {

*// Set the color of the polygon*

*//graphics.setColor(getColor());*

*// Draw the outline of the polygon*

graphics.drawPolygon(polygon);

}

}

*// Class representing a regular polygon shape*

class RegularPolygon extends HollowPolygon {

*// Constructor that calls super constructor*

public RegularPolygon(Color *color*, Point *center*, int *sides*, int *radius*) {

super(color, sides);

int[] x\_points = new int[sides];

int[] y\_points = new int[sides];

double angle = 2 \* Math.PI / sides;

for (int i = 0; i < sides; i++) {

double x = center.x + radius \* Math.cos(i \* angle);

double y = center.y + radius \* Math.sin(i \* angle);

x\_points[i] = (int) Math.round(x);

y\_points[i] = (int) Math.round(y);

}

createPolygon(x\_points, y\_points);

}

}