CMSC 330 Project 1

The first programming project involves extending the Java skeleton program that it is provided in the attached `.zip` file. That skeleton program displays a scene of graphic images contained in a scene definition file. The grammar for that scene definition file is shown below:

```
scene → SCENE IDENTIFIER number_list images END '.'

images → image images | image

image → right_triangle | rectangle

right_triangle →
  RIGHT_TRIANGLE COLOR number_list AT number_list HEIGHT NUMBER WIDTH
  NUMBER ';'

rectangle →
  RECTANGLE_ COLOR number_list AT number_list HEIGHT NUMBER WIDTH
  NUMBER ';'

number_list → '(' numbers ')'

numbers → NUMBER | NUMBER ',' numbers
```

In the above grammar, terminal symbols are upper case names or character literals shown in blue and nonterminal symbols are lower case names shown in red. EBNF metacharacters are shown in black. Tokens can be separated by any number of spaces. Identifiers and keywords are strings of alphabetic characters. Identifiers are case sensitive. Numbers are unsigned integers.

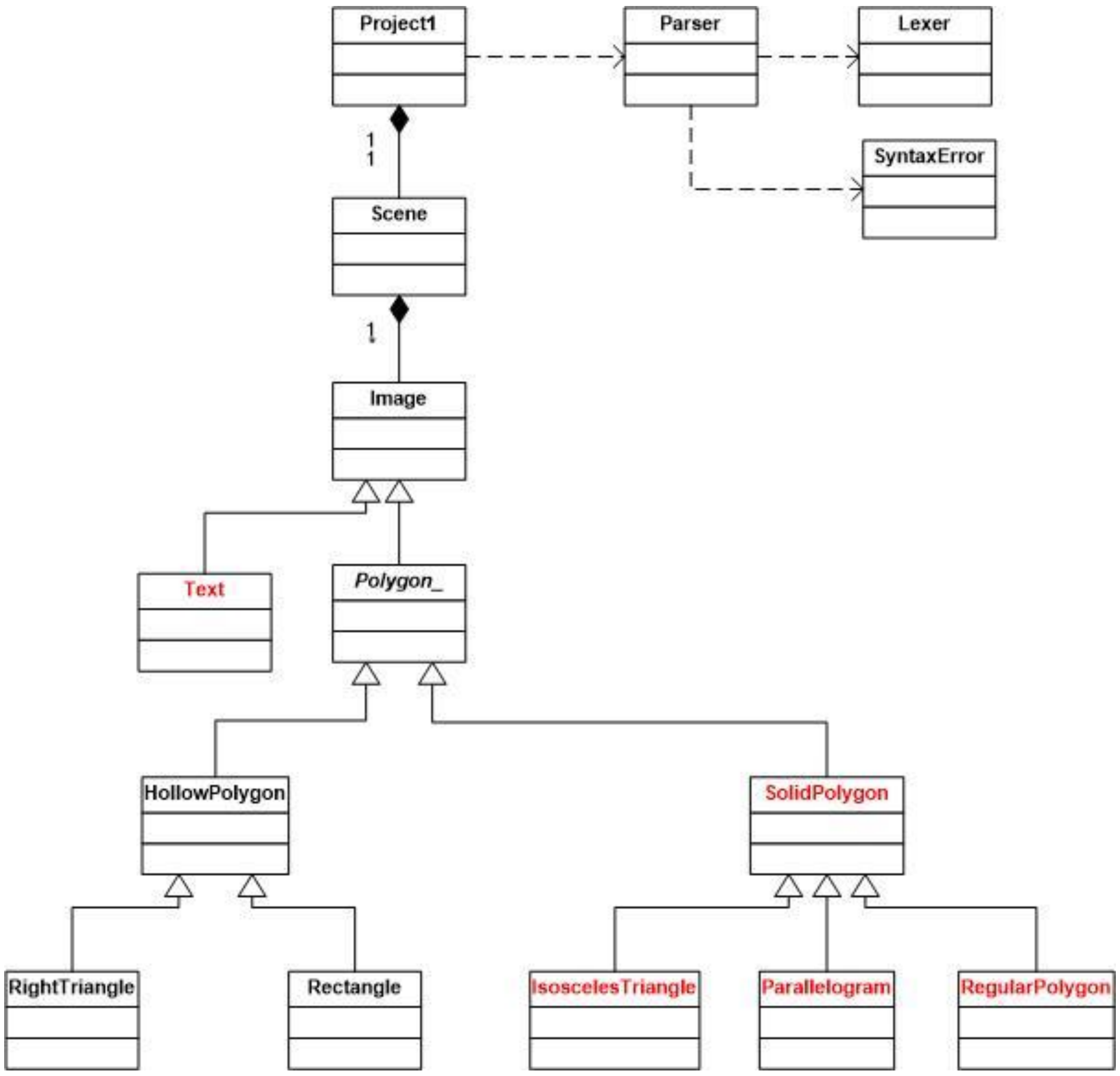
That program reads in the scene definition file that defines the image objects in a scene and creates those objects, inserts them into the scene and displays that scene.

You are to modify the program so that it will parse and display the additional images defined by the expanded grammar shown below with the additions to the grammar highlighted in yellow:

```
scene → SCENE IDENTIFIER number_list images END '.'

images → image images | image

image →
  right_triangle | rectangle | parallelogram | regular_polygon | isosceles
  | text

right_triangle →
  RIGHT_TRIANGLE COLOR number_list AT number_list HEIGHT NUMBER WIDTH
  NUMBER ';'

rectangle →
  RECTANGLE_ COLOR number_list AT number_list HEIGHT NUMBER WIDTH
  NUMBER ';'

parallelogram →
  PARALLELOGRAM COLOR number_list AT number_list number_list OFFSET
  NUMBER ';'
```

```
regular_polygon →
  REGULAR_POLYGON COLOR number_list AT number_list SIDES NUMBER RADIUS
  NUMBER ';'

isosceles →
  ISOSCELES COLOR number_list AT number_list HEIGHT NUMBER WIDTH
  NUMBER ';'

text →
  TEXT COLOR number_list AT number_list STRING ';'

number_list → '(' numbers ')'

numbers → NUMBER | NUMBER ',' numbers
```

The UML diagram for the whole project is shown below:



The classes shown in black are included in the skeleton project. You must complete the project by writing those classes shown in red, modifying the `Parser` class so that it will parse the expanded grammar, modifying the `Lexer` class to handle string tokens, and modifying the `Tokens` enumerated type to add all the new tokens.. Below is a description of each of the five classes that you must write:

The `Text` class must contain a constructor that is supplied the color that defines the text color, a point that specifies the text location and a string containing the text to be displayed. It must also contain a `draw` function because it is extends the abstract class `Image`. The `draw` function must draw the text using the method `drawString` in `Graphics` class.

The `SolidPolygon` class must contain a constructor that is passed the number of vertices in the polygon and its color. It must define the method `drawPolygon` because it is extends the abstract class `Polygon_`. It should call the `fillPolygon` method of the `Graphics` class to draw a solid polygon.

The `IsoscelesTriangle` class must have a constructor that is supplied the color of the triangle, a point that specifies the location of the top vertex, and the height and width of the triangle. It must allocate the arrays of *x* and *y* coordinates that defines the triangle and it must compute their values.

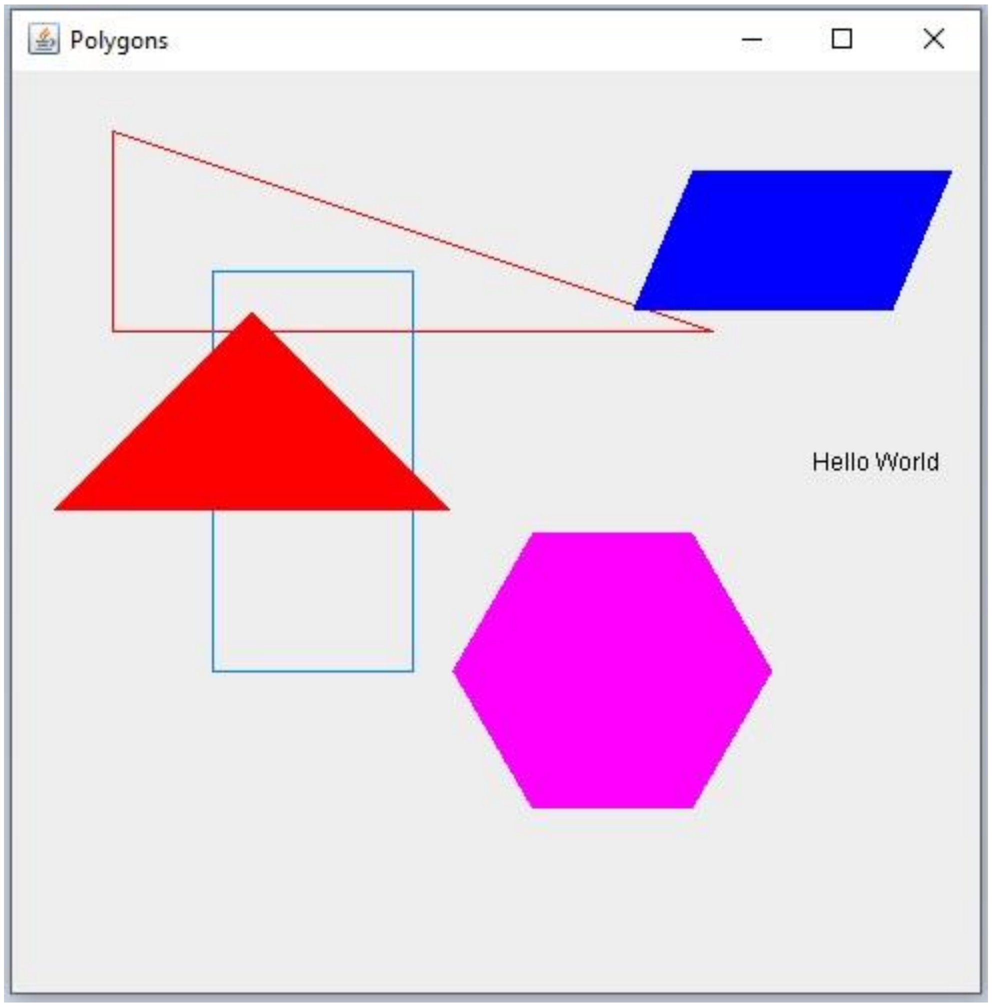
The `Parallelogram` class must have a constructor that is supplied the color of the parallelogram, two points that specifies the location of the upper left and lower right vertices in addition to an *x* offset value that specifies the *x* distance between the upper and lower left vertices. It must allocate the arrays of *x* and *y* coordinates that defines the parallelogram and it must compute their values.

The `RegularPolygon` class must contain a constructor that is supplied the color of the polygon, the number of sides, a point that specifies the location of its center, and the radius, which defines the distance between the center and each of the vertices. It must allocate the arrays of *x* and *y* coordinates that defines the regular polygon and it must compute their values.

Below is a sample of a scene definition file that would provide input to the program:

```
Scene Polygons (500, 500)
  RightTriangle Color (255, 0, 0) at (50, 30) Height 100 Width 300;
  Rectangle Color (0, 128, 255) at (100, 100) Height 200 Width 100;
  Isosceles Color (255, 0, 0) at (120, 120) Height 100 Width 200;
  Parallelogram Color (0, 0, 255) at (340, 50) (440, 120) Offset 30;
  RegularPolygon Color(255, 0, 255) at (300, 300) Sides 6 Radius 80;
  Text Color(0, 0, 0) at (400, 200) "Hello World";
End.
```

Notice that the token names are all upper case and may contain underscores whereas the corresponding lexemes at in title case without underscores. For example the token `RIGHT_TRIANGLE` has lexeme `RightTriangle`. The lexical analyzer in the skeleton handles the necessary conversion. Shown below is the scene that should be produced when the program is provided with the above scene definition.



The deliverables for this project include the following:

1. A `.zip` file containing all the source code correctly implementing all required functionality.
    a. All the `.java` files provided in the skeleton should included regardless of whether they required any changes
    b. All new files must include a header with the student name, date, project and a description of what the file contains
    c. All modified files must include a header with the same information
2. A Word or PDF file that contains the following:
    a. A discussion of how you approached the project
    b. A test plan that contains test cases that include all additional images and test any new functionality. For each test case, the output produced should be included.
    c. A discussion of lessons learned from the project and any improvements that could be made