מחלקות החבילה *Renderer*

1. מבנה המחלקה *ImageWriter* (מימוש מלא)

```
package renderer;

import java.awt.Color;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;

public class ImageWriter
{
	private int _imageWidth;
	private int _imageHeight;

	private int _Ny, _Nx;

	final String PROJECT_PATH = System.getProperty("user.dir");

	private BufferedImage _image;
	private String _imageName;

	// ***************** Constructors ********************** //
	public ImageWriter(String imageName, int width, int height,
				int Ny, int Nx)
	{

		_Nx = Nx;
		_Ny = Ny;

		_imageWidth = width;
		_imageHeight = height;

		_imageName = imageName;
		_image = new BufferedImage(_imageWidth, _imageHeight,
						BufferedImage.TYPE_INT_RGB);
	}
```

```
public ImageWriter(ImageWriter imageWriter)
{
      _Nx = imageWriter._Nx;
      _Ny = imageWriter._Ny;

      _imageWidth = imageWriter.getWidth();
      _imageHeight = imageWriter.getHeight();

      _imageName = imageWriter._imageName;

      _image = new BufferedImage(_imageWidth, _imageHeight,
                                  BufferedImage.TYPE_INT_RGB);
}

// ***************** Getters/Setters ********************** //
public int getWidth()  { return _imageWidth; }
public int getHeight() { return _imageHeight; }

public int getNy() { return _Ny;  }
public int getNx() { return _Nx;  }

public void setNy(int _Ny) { this._Ny = _Ny;  }
public void setNx(int _Nx) { this._Nx = _Nx;  }

// ***************** Operations ******************** //
public void writeToimage()
{
      File outFile = new File(PROJECT_PATH + "/" + _imageName + ".jpg");

            try  {
                  ImageIO.write(_image, "jpg", outFile);
            }
             catch (IOException e) {
                  e.printStackTrace();
            }
}
```

```
public void writePixel(int xIndex, int yIndex, int r, int g, int b)
{
           int rgb = new Color(r, g, b).getRGB();
           _image.setRGB(xIndex, yIndex, rgb);
}

public void writePixel(int xIndex, int yIndex, int[] rgbArray)
{
      int rgb = new Color(rgbArray[0], rgbArray[1], rgbArray[2]).getRGB();
      _image.setRGB(xIndex, yIndex, rgb);

}

public void writePixel(int xIndex, int yIndex, Color color)
{
      _image.setRGB(xIndex, yIndex, color.getRGB());
}
```

```
// Testing image writer
package tests;

import java.util.Random;
import org.junit.Test;

import renderer.ImageWriter;

public class ImageWriterTest
{

@Test
public void writeImageTest()
{
        ImageWriter imageWriter = new ImageWriter("Image writer test", 500, 500, 1, 1);
        Random rand = new Random();
        for (int i = 0; i < imageWriter.getHeight(); i++){
                for (int j = 0; j < imageWriter.getWidth(); j++)
                {
                        if (i % 25 == 0 || j % 25 == 0 || i == j || i == imageWriter.getWidth() - j)
                                        imageWriter.writePixel(j, i, 0, 0, 0);  // Black
                        else
                        if(i >= 200 && i <= 300 && j >= 200 && j <= 300)
                                imageWriter.writePixel(j, i, 255, 0, 0); // Red
                        else
                        if(i >= 150 && i <= 350 && j >= 150 && j <= 350)
                                imageWriter.writePixel(j, i, 0, 255, 0);  // Green
                        else
                        if(i >= 100 && i <= 400 && j >= 100 && j <= 400)
                                imageWriter.writePixel(j, i, 0, 0, 255); // Blue
                        else
                        if(i >= 50 && i <= 450 && j >= 50 && j <= 450)
                                imageWriter.writePixel(j, i, 255, 255, 0); // Yellow
                        else
                                imageWriter.writePixel(j, i, rand.nextInt(255), rand.nextInt(255),
                                                                rand.nextInt(255)); // Random
                }
         }

        imageWriter.writeToimage();

}

}
```

2. מבנה המחלקה Renderer :

```
package renderer;

import java.awt.Color;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

import elements.LightSource;
import geometries.FlatGeometry;
import geometries.Geometry;
import primitives.Point3D;
import primitives.Ray;
import primitives.Vector;
import scene.Scene;

public class Render
{
private Scene _scene;
private ImageWriter _imageWriter;

private final int RECURSION_LEVEL = 3;

// ***************** Constructors ********************** //
public Render(ImageWriter imageWriter, Scene scene);

// **************** Operations ******************* //
public void renderImage();


private Entry<Geometry, Point3D> findClosesntIntersection(Ray ray);


public void printGrid(int interval);


public void writeToImage();


private Color calcColor(Geometry geometry, Point3D point, Ray ray);
```

```
private Color calcColor(Geometry geometry, Point3D point,
                        Ray inRay, int level);  // Recursive

private Ray constructRefractedRay(Geometry geometry, Point3D point,
                                  Ray inRay);

private Ray constructReflectedRay(Vector normal, Point3D point,
                                  Ray inRay);

private boolean occluded(LightSource light, Point3D point,
                         Geometry geometry);

private Color calcSpecularComp(double ks, Vector v, Vector normal,
                       Vector l, double shininess, Color lightIntensity);

private Color calcDiffusiveComp(double kd, Vector normal, Vector l,
                                Color lightIntensity);

private Map<Geometry, Point3D> getClosestPoint(Map<Geometry,
                               List<Point3D>> intersectionPoints);

private Map<Geometry, List<Point3D>> getSceneRayIntersections(Ray ray);

private Color addColors(Color a, Color b);

}
```