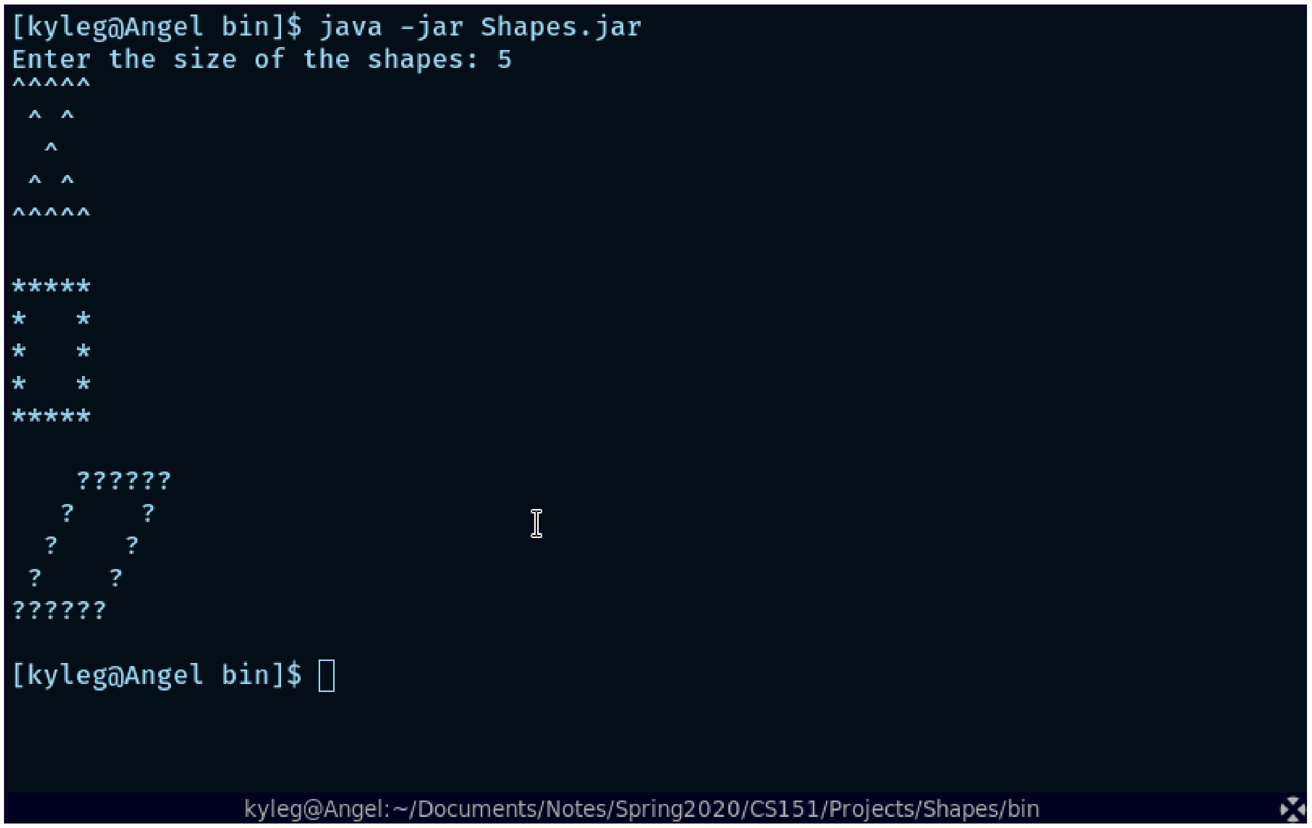
Project 7
Kyle Guarco

For more information on the solution, please read **README.txt** *in the project submission.*

Page 1	Output
Pages 2-3	TestShape.java
Pages 4-6	Buffer.java
Pages 7-8	Drawable.java
Pages 9-10	Vector2.java
Pages 11-12	VectorBuilder.java

Output



```
[kyleg@Angel bin]$ java -jar Shapes.jar
Enter the size of the shapes: 5
^^^^^
 ^ ^
  ^
 ^ ^
^^^^^

*****
*   *
*   *
*   *
*****

    ??????
   ?    ?
  ?    ?
 ?    ?
??????

[kyleg@Angel bin]$ 
```

kyleg@Angel:~/Documents/Notes/Spring2020/CS151/Projects/Shapes/bin

TestShape.java

```
import java.util.Scanner;

public class TestShape
{
      public static void main(String[] args)
      {
            Scanner stdin = new Scanner(System.in);

            // Ask the user to enter the size of all the buffers.
            System.out.print("Enter the size of the shapes: ");
            int size = stdin.nextInt();

            // We'll just use one buffer object for the sake of memory, OK?
            Buffer buffer;

            // Make an hourglass appear!
            buffer = new Buffer('^', size, size);
            buffer.render(new Hourglass());
            System.out.println(buffer);

            // Make a square appear!
            buffer = new Buffer('*', size, size);
            buffer.render(new Square());
            System.out.println(buffer);

            // Make a slightly slanted square appear!
            buffer = new Buffer('?', size * 2, size);
            buffer.render(new SlantRight());
            System.out.println(buffer);

            stdin.close();
      }
}

class Hourglass implements Drawable
{
      @Override
      public Vector2[] draw(Buffer buffer)
      {
            int width = buffer.getWidth(), height = buffer.getHeight();
`
            VectorBuilder line_1 = new VectorBuilder()
                  .plot(Vector2.ZERO)
                  .plot(width, height);
            VectorBuilder line_2 = new VectorBuilder()
                  .plot(0, height)
                  .plot(width, 0);

            return connect(line_1.merge(line_2).points());
```

```
	}
}

class Square implements Drawable
{
	@Override
	public Vector2[] draw(Buffer buffer)
	{
		int width = buffer.getWidth(), height = buffer.getHeight();

		return connect(new VectorBuilder()
			.plot(Vector2.ZERO)
			.plot(width, 0)
			.plot(width, height)
			.plot(0, height)
			.points());
	}
}

class SlantRight implements Drawable
{
	@Override
	public Vector2[] draw(Buffer buffer)
	{
		int width = buffer.getWidth(), height = buffer.getHeight();
		int middle = width / 2;

		Vector2[] verticies = new VectorBuilder()
			.plot(middle, 0)
			.plot(width, 0)
			.plot(middle, height)
			.plot(0, height)
			.points();

		return connect(verticies);
	}
}
```

Buffer.java

```
/**
* This is, what I will call, a text buffer. This class holds an array of empty
* characters in memory. <p>
*
* This class will have points passed to it, points that exist in the text
* buffer. It will print those points into the text buffer using a specified
* character to draw with. <p>
*
* In addition, buffers can be drawn on top of other buffers! Just pass
* a buffer to another buffer and all the points on it will be printed into
* the other buffer instance. For example: {@code buffer.render(otherbuffer)}
* will render {@code otherbuffer} onto {@code buffer}.
*
* @author Kyle Guarco
*/
public final class Buffer implements Drawable
{
      /* The width and height of the buffer */
      private int width, height;
      /** The character that will be used to draw lines */
      private char draw;
      /** The buffer itself. This is where the characters will render. */
      private char[][] buffer;
      /** Used for the buffer drawing feature. */
      private Vector2[] points;

      public Buffer(char draw, int width, int height)
      {
            this.draw = draw;
            this.width = width;
            this.height = height;

            this.buffer = new char[height][width];

            // The buffer musn't be empty. This is so shapes can render
properly.
            for (int y = 0; y < height; y++)
                  for (int x = 0; x < width; x++)
                        buffer[y][x] = ' ';
      }

      public Buffer(char draw, Vector2 dimensions)
      {
            this(draw, dimensions.x, dimensions.y);
      }

      @Override
      public String toString()
      {
            // We need to properly convert each line of the
            // buffer into a string for some reason.
            StringBuilder display = new StringBuilder();

            // Append each line of the text buffer to the final string
            for (int i = 0; i < height; i++)
                  display.append(new String(buffer[i]) + '\n');

            return display.toString();
      }

      @Override
      public Vector2[] draw(Buffer buffer)
      {
            // Avoid a NullPointerException by just creating an empty array.
            if (points == null)
                  return new Vector2[0];
            // If the width or height of the buffer are larger, don't draw
            if (width < buffer.getWidth() || height < buffer.getHeight())
                  return new Vector2[0];

            return points;
      }

      /**
      * This function draws characters at all the specified points.
      *
      * This doesn't return anything, but instead accesses the buffer directly.
      *
      * @param drawable Any drawable element
      * @param x_ofs
      * @param y_ofs
      */
      public void render(Drawable drawable, int x_ofs, int y_ofs)
      {
            this.points = drawable.draw(this);

            for (Vector2 point : points)
            try {
                  buffer[point.y + y_ofs][point.x + x_ofs] = draw;
            } catch (ArrayIndexOutOfBoundsException e) {
                  System.out.println("ERROR: Can't Draw " + point);
            }
      }

      /**
      * This function draws characters at all the specified points.
      * This doesn't return anything, but instead accesses the buffer directly.
      *
      * @param drawable Any drawable element
      */
      public void render(Drawable drawable)
```

```
	{
		render(drawable, 0, 0);
	}

	/** @return The width, not accounting for the newline at the end */
	public int getWidth()
	{
		return width - 1;
	}

	/** @return The height of the buffer */
	public int getHeight()
	{
		return height - 1;
	}
}
```

Drawable.java

```
/**
* This interface allows for the implementation of drawable elements
* onto a {@code Buffer}. Anything that implements this interface can
* be drawn, as long as it has points to draw.
*
* @author Kyle Guarco
*/
public interface Drawable
{
      /**
      * Called by the buffer, and passes itself to the shape, so that
      * it may properly draw itself.
      *
      * @param buffer The buffer instance
      * @return The points that are to be printed into the buffer
      */
      Vector2[] draw(Buffer buffer);

      /**
      * A helper function for drawables. Allows for the connection
      * of verticies, creating shapes that have sides!
      *
      * @param verticies The verticies for the shape
      * @return All the points required for drawing
      */
      default Vector2[] connect(Vector2[] verticies)
      {
            VectorBuilder points = new VectorBuilder();

            for (int i = 0; i < verticies.length; i++)
            {
                  boolean looping = i + 1 == verticies.length;

                  Vector2 p1 = verticies[i];
                  // If 'i' is already maxed, use the first point
                  // to finish the connection
                  Vector2 p2 = looping ?
                        verticies[0] :
                        verticies[i + 1];

                  // Usually, when you're drawing lines on a graph, you'd
calculate
                  // the slope and find all the points that way. However, this
isn't
                  // a real graph. We need to find both the X and Y distance
from
                  // the second vector.

                  // Also, negatives shouldn't exist!
                  points.plot(p1);

                  while (true)
                  {
                        // Since this isn't a real graph, we need to plot the
                        // points on it a different way. Compare X and Y and take
                        // the comparison's integer representation (-1, 0, 1) and
keep
                        // adding that on to the vector until it is equal with
p2.
                        int dX = p1.compareX(p2);
                        int dY = p1.compareY(p2);

                        // Add dX (difference in X) and dY to the current vector
                        p1 = new Vector2(p1.x + dX, p1.y + dY);
                        points.plot(p1);

                        // When dX and dY are 0, it means that both points are
                        // even with each other. This line has finished drawing!
                        if (dX == 0 && dY == 0)
                              break;
                  }

                  points.plot(p2);
            }

            return points.points();
      }
}
```

Vector2.java

```
public final class Vector2
{
	public static final Vector2 ZERO = new Vector2(0, 0);

	/* The constant values of the vector. */
	public final int x, y;

	public Vector2(int x, int y)
	{
		this.x = x;
		this.y = y;
	}

	@Override
	public String toString()
	{
		return String.format("Vector2(%d, %d)", x, y);
	}

	@Override
	public boolean equals(Object o)
	{
		if (!(o instanceof Vector2))
			return false;

		Vector2 vec = (Vector2) o;

		return x == vec.x && y == vec.y;
	}

	/**
	* Compares the current vector with the passed vector's X value
	* @param p2
	* @return
	* {@code 1} if the passed vector is greater <p>
	* {@code 0} if both vectors are equal <p>
	* {@code -1} if the passed vector is lesser
	*/
	public int compareX(Vector2 p2)
	{
		if (p2.x > x)
			return 1;
		else if (p2.x < x)
			return -1;

		return 0;
	}

	/**
	* Compares the current vector with the passed vector's Y value
```

```
	 * @param p2
	 * @return
	 * {@code 1} if the passed vector is greater <p>
	 * {@code 0} if both vectors are equal <p>
	 * {@code -1} if the passed vector is lesser
	 */
	public int compareY(Vector2 p2)
	{
		if (p2.y > y)
			return 1;
		else if (p2.y < y)
			return -1;

		return 0;
	}

	public int distance(Vector2 p2)
	{
		return (p2.y - y) / (p2.x - x);
	}

	public Vector2 add(Vector2 vec)
	{
		return new Vector2(vec.x + x, vec.y + x);
	}

	public Vector2 sub(Vector2 vec)
	{
		return new Vector2(vec.x - x, vec.y - x);
	}

	public Vector2 mul(Vector2 vec)
	{
		return new Vector2(vec.x * x, vec.y * y);
	}

	public Vector2 div(Vector2 vec)
	{
		return new Vector2(vec.x / x, vec.y / y);
	}
}
```

VectorBuilder.java

```java
import java.util.ArrayList;

/**
* This class uses the builder pattern to construct an array of points.
*
* @author Kyle Guarco
*/
public class VectorBuilder
{
      private ArrayList<Vector2> points;

      public VectorBuilder()
      {
            this.points = new ArrayList<>();
      }

      @Override
      public String toString()
      {
            StringBuilder result = new StringBuilder();

            for (Vector2 point : points)
                  result.append(point.toString() + '\n');
            return result.toString();
      }

      public VectorBuilder plot(Vector2 vec)
      {
            points.add(vec);

            return this;
      }

      public VectorBuilder plot(int x, int y)
      {
            points.add(new Vector2(x, y));

            return this;
      }

      public VectorBuilder merge(VectorBuilder... builders)
      {
            for (VectorBuilder builder : builders)
                  for (Vector2 vec : builder.points())
                        plot(vec);

            return this;
      }
```

```
	public Vector2[] points()
	{
		return points.toArray(new Vector2[points.size()]);
	}
}
```