Computer Laboratory 6

CSCI 1913: Introduction to Algorithms,
Data Structures, and Program Development

1 Essential information

- This assignment is due Tuesday November 1st at noon and will be turned in on gradescope
- The assignment is intended to be done in pairs, although you're allowed to work on your own if you want.
- You can start the assignment before your lab period, although I recommend not getting too far – the lab TAs often have useful hints, and I don't want you wasting too much time struggling with something that the TAs might announce in-class.
- For more rules see the lab rules document on canvas.
- **By function count this is a long lab** This is a bit deceptive, many of the functions are about 1 line long, and 3 of them are nearly identical. Don't let the function count intimidate you. That said, it's also important **You don't lose too much time on any one problem** – make sure you're using the TA support available in-lab and asking for help when you get stuck. The TAs are around to help you with syntax translation problems you might face. (And they will do their best if you get stuck on the geometry)

2 Introduction

In this lab we will create some simple procedural art. Procedural art is a form of artistic expression in which the artist creates artwork by writing code. When run, these "art programs", in turn, create digital art (music files, images, etc.) Our code will be focused on relatively simple visual art created by simple geometric shapes.

Since this is still a quite early java lab – we won't be building a novel framework for procedural art – but instead extending an existing codebase with some new geometric utilities. This will give you a lot of practice reading and using java objects in different configurations (Arrays of objects, objects in other objects, etc.)

From an educational standpoint, in this lab you will:

- Practice *reading* code written by others and thinking about how to use it.
- Practice creating and using java objects, calling methods, and thinking about how objects relate to each other.

- Get experience with two common situations that tend to cause trouble: arrays of objects and objects stored inside other objects.
- Build your expertise with Java syntax and object-oriented programming.
- Get practice and feedback on your java code style (this will be graded!)
- See one method developed in three different contexts (instance method, static method on same class using private variables, static method on different object.)
- Allow you to express yourself creatively in Java!

While this lab may not be as *computationally* difficult as some earlier labs in this class, it focuses on important *concepts and syntax*. As you work on this lab, therefore, don't just "edit until it works". Instead, focus on working deliberately – make sure you understand the syntax and concepts behind everything you do. If you do find yourself getting stuck, you are of course free to get help from course staff or your lab partner. More importantly however: after you do resolve the issue, make sure to come back after debugging and review what the correct syntax for a given task is!

On the other hand – if you do find this to be an easier lab than most

1. pat yourself on the back – you've really got the nuances of object-oriented syntax down,
2. take this moment to focus on developing your code style – think actively about how to make your code maximally readable! and
3. Take this as a moment to REALLY show off – the last part of this assignment is very open-ended to give everyone a chance to explore java and create something fun and interesting! (With permission) we intend to post our favorite results to the class as a celebration of our creativity!

3 Software environment setup

This lab will be done using a java programming environment. We recommend the IntelliJ IDE running on your own personal computer. See lab 6 for instructions about how to set this up.

A few reminders:

- Source code goes in the `src` folder in java. Java is VERY particular about where files go, and what they are called.
- Lab06 has syntax guides if you still find translating to java difficult.
- Do not have any folders inside src – these are called packages and will make your code fail the autograder.
- Download *every* provided file and move them all to the src folder before starting.
- Code style elements like comments and javadocs are **important** and **easier to do as you go**. Don't write hard-to-read code and think "I can fix it later".

4 Files

This lab will involve the following **provided** files. You will need to read these files, but should not need to change them:

- `Point.java` – A simple java class representing a 2d point.
- `Circle.java` – A simple java class representing a circle.
- `ShapeDrawer.java` – A more complicated java class providing a toolkit for creating 2d drawings
- `ShapeUtilsTester.java` – A java program that tests the ShapeUtils class you will have to write.

This lab also involves the following files **you will edit or create**:

- `Triangle.java` – A simple java class representing a triangle. (A version of this file is provided – you will have to update this)
- `ShapeUtils.java` – A java class with many static functions that perform various geometric computations on java objects
- `DrawTree.java` – A Java program that creates an image fitting a few key requirements.
- `tree.png` – A copy of the type of image you code can generate.

5 Instructions

Before beginning you should:

1. Setup an IntelliJ package
2. Download the provided files and place them in the src folder.
3. Create empty versions of the two required java files, and ensure that your names are in a comment at the top of the file (as this is required for code-style)
4. Update the comment at the top of the Triangle class to include your name.
5. Comment code out in ShapeUtilsTester until it shows no errors – you will need to progressively uncomment it as you implement more functions (but java will not run AT ALL if there are ANY syntax errors.)
6. Skim the formal requirements sections and the java syntax sections. (You'll want to look over everything before you get started)
7. Read, and review the provided classes – you will be expected to use these files, and the provided source code *is your guide* for doing that. You may find it useful to take quick notes so you know what methods are available, and roughly what they are called. Remember – your goal isn't to read the **CODE** your goal is to read function names and javadocs.
8. Only then should you *think* about doing any novel programming.

6 Formal Requirement: Code reading

The first required task here is to read the provided java classes and make sure you know how to use them. The steps for this, as presented in class, are as follows:

- Open the file
- (If your editor supports it) "fold" all the code – you **should not need to read the code** (sometimes that can even make understanding how to use a class *harder!*)
- Read only the function names/parameters/return types and java docs
- Make a few informed guesses if things are not fully clear, and don't be afraid to ask for more info if you need to.
- Write some testing code for yourself to make sure you understand anything that isn't clear enough. Don't forget the **power** of playing around with the code to see what works.

Be aware that code-reading is an IMPORTANT SKILL, and one that many consider harder than code writing! Even though this requirement has no formal "hand-in" – give it your best effort and practice just like the code-writing requirements! Many of the most common problems in this lab stem directly from failure at this requirement.

7 Formal Requirements: ShapeUtils

The ShapeUtils class has eight required functions. Many of these can be done in only a few lines with the right syntax. You can do these in any order – and you may not find that the provided order here is the "easiest" order to take things.

1. `public static double distance(Point p1, Point p2)`
2. `public static Point getCenter(Point[] points)`
3. `public static Triangle[] makeFakePoly(Point[] points)`
4. `public static double getArea(Circle c)`
5. `public static double getArea(Triangle t)` (Note – this is actually detailed in the next section it's listed here for completeness)
6. `public static boolean isIn(Triangle t, Point p)`
7. `public static boolean isIn(Circle c, Point p)`
8. `public static boolean isThereOverlap(Circle c1, Circle c2)`

7.1 distance

The distance function takes two points and computes the distance between them:

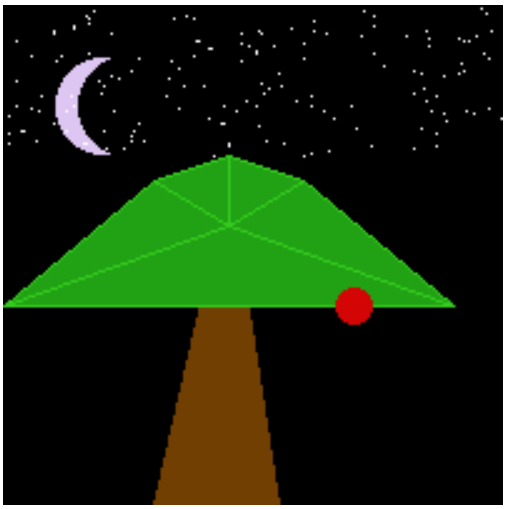
$$distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

7.2 getCenter(Point array)

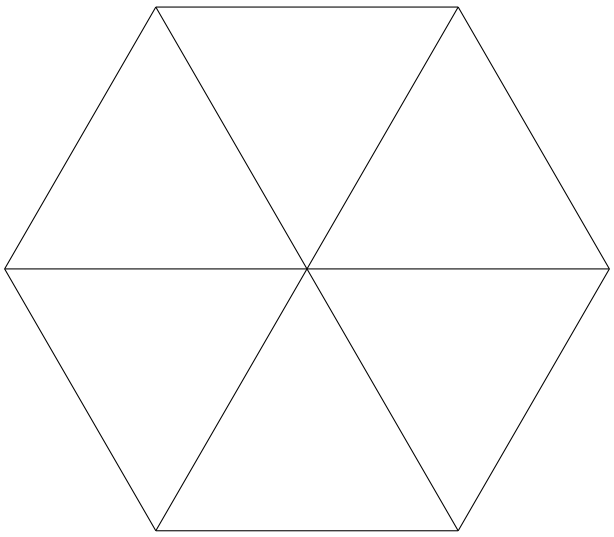
The getCenter function should take an array of points and return a point that represents the "center" of those points. This can be computed by taking the average x value of each point in the array, and the average y value of each point in the array, and making a new point from those two averages. If an empty array is given the point $(0, 0)$ should be returned.

7.3 makeFakePoly(Point array)

Our graphics toolkit (like many) does not *explicitly* support drawing any polygon more complicated than a triangle. While this might sound like a limitation, it's because you can draw any polygon by drawing a bunch of triangles. As example of this can be seen in the "top" of the tree image here:



Here's a more "mathematical" drawing showing how we can make a hexagon with 6 triangles:



The purpose of the makeFakePoly is to create arrays of triangles that can be used to draw more complicated polygons. The function takes an array of points. You can assume this array has at least 3 points in it. The function should return an array of triangles.

Important – there are MANY ways to do this which would work in general. Our tests, however, are designed to look for one specific solution approach, which is outlined here. If you want to pass tests you will have to be careful to follow this pseudocode

1. Create an array for outputting triangles. Ultimately this will need to be the same length as the input point array.
2. Compute the "center point" using the previous function.
3. for each array index i create a new triangle using $p1 = points[i], p2 = points[i+1], p3 = center$ (For the final position, $p1 = points[N-1], p2 = points[0], p3 = center$)
4. return the new array

So if we gave the function the points $[(1,1),(1,2),(2,2),(2,1)]$ it should return the 4 triangles (in this order):
$(1,1) - -(1,2) - -(1.5,1.5)$
$(1,2) - -(2,2) - -(1.5,1.5)$
$(2,2) - -(2,1) - -(1.5,1.5)$
$(2,1) - -(1,1) - -(1.5,1.5)$

7.4 getArea(Circle)

The getArea function takes a circle and returns it's area. The area of a circle can be computed by the following equation:

$$A = \pi r^2$$

7.5 getArea(Triangle)

This function is outlined in the next major requirements section. You should wait to implement it with the other versions discussed then.

7.6 isIn(Triangle, Point)

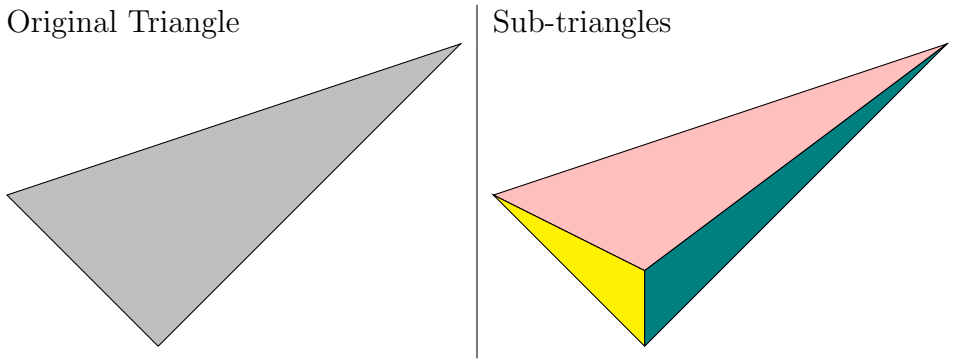
The isIn function for a triangle asks if a point is within a triangle. This is a surprisingly hard problem, with many mathematical/geometric solutions. We recommend the following approach:

- Compute the area of the triangle
- Make 3 new triangles, in each one a different point from the original triangle has been replaced with the input point. we call these the "sub-triangles"
- Add the area of the 3 sub-triangles

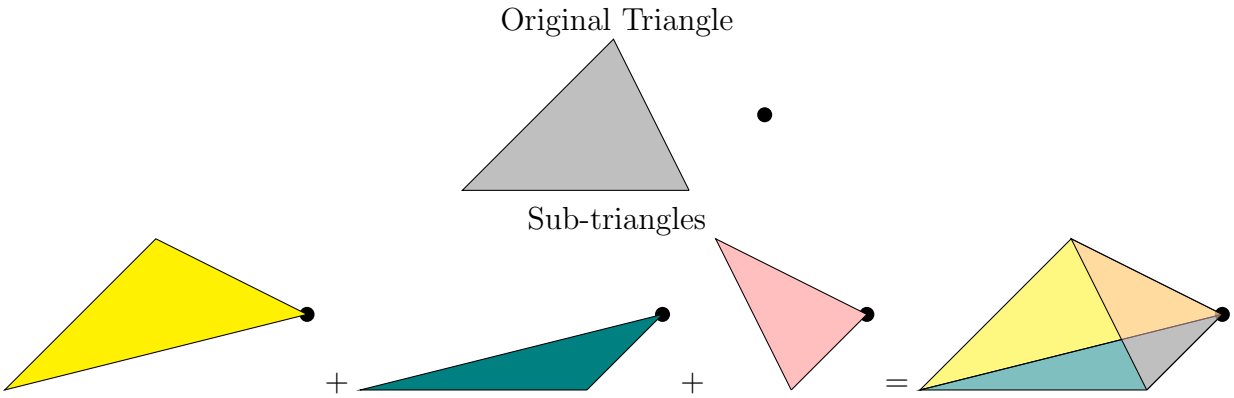
- If the area of the 3 sub-triangles equals the are of the input triangle, the point is in the triangle, otherwise it is not.

double math is often difficult. Since this math is done with double values – we can't actually guarantee that two numbers which mathematically *should* be equal, will be equal. Therefore you should return true if the two areas are within 0.00001 of each other. This can easily be checked by seeing if the absolute value of the difference is less than 0.00001.

A visual example explaining the geometry As a quick explanation of this method, consider the following two examples:



Case 1: In this case, the point is contained in the triangle, so the 3 "sub-triangles" just split-the original triangle into 3 parts. It's visually clear that the sub-triangles would have the same total area as the original triangle.



Case 2: The point is outside the original triangle. In this case the three "sub-triangles" have to stretch outside of the original triangle to get to the point Additionally, due to the structure of the triangles, the triangles overlap (the orange and grey parts of the final image are overlaps with the pink triangle) As can be seen, the three sub-triangles in this case will have a much larger total area.

7.7 isIn(Circle, Point)

This function takes a circle and checks if a point is in the circle. Compared to the triangle case, this is a lot easier, simply compute the distance from the center of the circle to the target point. If that distance is less-than or equal to the radius of the circle, the point is in the circle.

7.8 isThereOverlap

The isThereOverlap function takes two circles and asks if the two circles overlap. This problem *sounds* like tricky geometry, but it can actually be solved quite easily. Simply compare the distance between the center of the two circles with the sum of their radii. If the distance between the centers is less than the sum of the radii, then the circles overlap. (If the geometry here isn't clear I recommend drawing a few circles)

8 Formal Requirements: Area of a Triangle

This section is intended to give you exposure to the differences in how you write code in three different contexts by having you implement the same behavior (area of a triangle) in three different functions.

- Instance method (private-access non-static)
  (Triangle class method `public double getArea()`)
- Same-class function (private-access static)
  (Triangle class function `public static double getArea(Triangle t)`)
- Other-class function (public-access only, static)
  (ShapeUtils function) `public static double getArea(Triangle t)`

8.1 Instance method

On the Triangle class fill-in the empty function `public double getArea()`

The area of a triangle can be computed from it's points with the following equations where $p1, p2, p3$ are the triangle points

$$t1 = p1.x * (p2.y - p3.y)$$

$$t2 = p2.x * (p3.y - p1.y)$$

$$t3 = p3.x * (p1.y - p2.y)$$

$$area = 0.5 * abs(t1 + t2 + t3)$$

(or all-in one)

$$area = 0.5 * abs(p1.x * (p2.y - p3.y) + p2.x * (p3.y - p1.y) + p3.x * (p1.y - p2.y))$$

Note – in this method you will have direct access to the three private point variables `p1`, `p2`, `p3`. You will not, however, have access to the point-variable's private x and y values. You will have direct access to p1, p2, p3 because this is not a static method – therefore this will always have to be run on *some* Triangle, and that triangle's p1, p2, p3 variables can be used directly.

(An example for clarity)

```
double xsum = p1.getX() + p2.getX() + p3.getX();
```

8.2 same-class static method

On the triangle class fill-in the empty function `public static double getArea(Triangle t)`

The area equation is, of course, the same here. The only difference is how you access the p1, p2, and p3 variables. This method runs in an *Static context*, this means that, unlike the previous function, it isn't called like `someTriangle.getArea()` it would have to be called on the class itself `Triangle.getArea(someTriangle)`. This also means that you only have *direct* access to static properties and cannot just write `p1.getX()` (since there isn't a clear definition of what "p1" means in this case).

To update your code for this context you will need to modify it to use the `t` triangle variable to get the p1, p2, and p3 values. Note – since this is still in the Triangle class you can still access the private variables. Make sure you understand *why* you need to make these changes when moving from an instance method to a static one.

(An example for clarity)

```
double xsum = t.p1.getX() + t.p2.getX() + t.p3.getX();
```

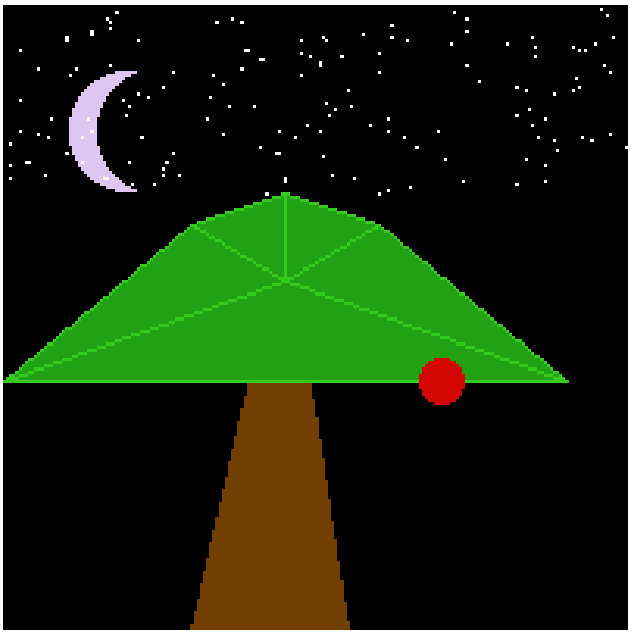
8.3 Other class static method

For our final function add a method `public static double getArea(Triangle t)` to the ShapeUtils class. Like the last function this runs in a static context and must be given a triangle to operate on. Unlike the last function this is on a different class, so it can only use public properties. To access the points you will need to use the public getter methods (`getP1()`, `getP2()`, `getP3()`) on the triangle class. Again – make sure you understand *why* you need to make these changes when the function is in a different class.

(An example for clarity)

```
double xsum = t.getP1().getX() + t.getP2().getX() + t.getP3().getX();
```

9 Draw Tree



The DrawTree class should have one method: `public static void main(String[] args)`. This program should use the ShapeDrawer, as well as the Point, Triangle, and Circles classes to draw a simple tree figure. This should be similar to the image above, but it does not need to be identical. The image file needs to be saved to the file "tree.png". If you name the file something else (or forget to tell the shapeDrawer to draw it) the autograder will not accept it.

Your tree picture must:

- Must have a clearly tree-trunk and tree-top. The trunk and top should be different colors. The trunk can be drawn as a single triangle as in the example above, but the tree top must be drawn using more than one shape.
- Must have a sun or moon. (A crescent moon is a fun little challenge to draw with the shapes we have, there's a trick to making it look quite nice!)
- Must have some "tree decoration" in our example that's the red circle representing some fruit, but you can feel free to go bigger and do a flower, or a star, or a bird or anything else that seems appropriate.
- At some point in the main method you must use the ShapeUtils class in a *meaningful way* (I.E. something we couldn't just delete) We recommend doing this by using the makeFakePoly function to make one or more interesting non-basic shapes for your drawing.
- The image itself should be at least 100 pixels wide by 100 pixels tall. You should seek to make a larger picture than this, but 100-by-100 is the minimum we'll allow
- you will need to change colors from the default draw colors (you won't need to use *our* colors, but you will need to use more than just the defaults)

  - To do this you will need to use the java.awt.color class, feel free to search that to see how to use it.
  - The built-in java color class defines a few common/simple colors, but also allows you to create new colors using the RGB color space.
  - Intellij has a tool built-in to help you pick numbers representing colors visually for RGB, as does google (if you google search "rgb color picker" you can find a tool to help find colors)

Beyond these requirements you are free to do what you wish. Make sure this class runs correctly and submit your tree.png to demonstrate that you've run the code correctly.

If you find yourself finishing early, you are *encouraged* to explore a bit – a few ideas:

- try adding clouds, stars, or "sun rays" (lines or triangles showing coming off the sun)
- Give the sun (or moon) a face!
- Create stars or clouds – try to do this randomly, rather than programming all the details yourself.
- fractal trees. Use a recursive function to make the tree have branches, and those branches have branches etc!
- Draw a whole orchard! Or a friendly house for the tree?

Note – we may give extra credit for drawings that we find particularly impressive either artistically (it looks very good), or technically(there are some complicated bits of code going into your output such as randomized functions and/or recursive drawing functions).

10 Java Syntax Guide

Most of the required java syntax has been in readings and/or in lecture before now, but there are a few minor details I wanted to include as a reminder.

- Classes like java.lang.Math and java.util.Random are available for you to use. If you can't remember what functions are available, try looking up the official java documentation.
  - `https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html`
  - `https://docs.oracle.com/javase/8/docs/api/java/util/Random.html` (remember – you will need to import the Random class)
- We will be requiring javadocs (written by you) in this assignment. As a reminder these use a two-star comment placed before the function:

```
/**
 * compute the sum of numbers between a minimum (inclusive)
 * and maximum value (inclusive) value.
*/
public static int sumOfNums(int min, int max) {...}
```

Remember – documentation like this does not need to be super long, but it should explain what the function is, what it does, and if unclear – what it's parameters and return type mean.

- Static, or class, methods have the static modifier – these can be called without making objects. If a function or variable does not have the static method, however, it is an instance-property. This means that any function or variable not listed with "static" – is designed to be used with objects of a given type and cannot be used directly.
- As an example, the Point method `getX()` does not have the static modifier, so you can't use `Point.getX()` to call the function, instead you would need to reference a point object directly (perhaps using a variable) such as: `pointVar.getX()` – as you need to run getX on one specific object.
- Static methods can be called directly from a class. As an example, the distance function you will write in ShapeUtils can be called from any other class like: `ShapeUtils.distance(point1, point2)` Within the ShapeUtils class, however, you can call distance directly: `distance(point1, point2)`.
- Constructors – you can tell that a function is a constructor if it's name matches the class name (and it lists no return type) There's a special syntax for calling constructors. Example:

```
public class Foo {
    public Foo(int x) {...} // this is a constructor
    public static void (String[] args) {
            // and here we can see it being used-- note the syntax:
            // new <classname>(<parameters>)
      Foo variable = new Foo(7);
    }
}
```

More information on any and all of these topics can be found in the zybooks textbook, or by asking for assistance from your lab TA. Remember – the purpose of this lab is to get comfortable with this object-oriented syntax – it's OK if you spend some time wrapping your head around this syntax and it's rules. Just make sure you know these rules by the deadline and you're fine!

11 Java Autograder Notes

The java autograder is much fussier than the python autograder. This is mostly due to Java's nature as a compiled language. While we could often *partially* test python files that had errors in some functions – for your code to be testable your code must compile, have all the correct function definitions, and all the correct names. Even once you get the tests *running* you should also be aware that the autograder for java does not give as clear feedback as the python autograder when things go wrong. As such **you are expected to be testing you code on your own**. Do not use the autograder as a debugging tool.

Due to some strange things in the autograder, we have a few specific requirements for your code:

1. The code you submit must be valid and compileable java code. (We cannot test code with syntax errors. If we can't test it, we can't give you autograder credit for it)
2. The code you submit must have no package statement in it. (We have to do some weird things with package structures to get the code to compile correctly.) **YOU WILL GET 0 points if your code has a package statement!** Your IDE may have put one in without you noticing so you should check for this. If you see code like `package com.whatever;` at the top of your file you need to delete it.
3. The code you submit must match the provided function signature (name, parameter types, and modifiers) EXACTLY. Any mis-match, no matter how minor, may cause the test code to not compile, which would prevent testing.

As a general rule, if you started with the template file on canvas and didn't modify anything other than the function bodies you should be fine this time. Likewise, if your code can be run with the provided testing code (with no modifications to this testing code) you should be fine. If you do run into testing issues, do not hesitate to reach out to course staff to help with debugging.

12 Submission

For this lab you need to turn in:

- `ShapeUtils.java`
- `Triangle.java`
- `DrawTree.java`
- `tree.png`

Grading on this assignment will follow the given general rubric:

- 50 points autograder – This is split somewhat chaotically across the methods.
- 10 points code style:
  - Your name (and the name of any collaborators) should be in a comment at the top of the file
  - Correctly formatted JavaDoc (this is the `/** ... */` comment blocks placed before each function that describes the function
  - Good variable names
  - Consistent tabbing (while java does not require this the same way python did – we **do** require it. Reading and grading poorly tabbed code is a painful experience)
  - Outside of these explicit requirements, as normal, we will generally be looking for (and giving feedback on) anything that makes the code harder/more painful to read, and will take away points if it's particularly bad.

- 10 points for the DrawTree and tree.png
- 30 points for the following functions:
  - 4 The three versions of the triangle getArea functions
  - 2 points distance(Point)
  - 5 getCenter(Point array)
  - 5 makeFakePoly(Point array)
  - 2 getArea(Circle)
  - 5 isIn(Triangle)
  - 2 isIn(Circle)
  - 5 isThereOverlap(Circle, Circle)