CSC-223
Lab 8
Work in a group of 1, 2, or 3.

Geometry Fact Computer: Angle and Triangle Identifier

After much effort, we have reached a point in our Geometry Fact Computer that we can begin computing more figure-based facts. In this assignment, we will compute all angles and all triangles for an input geometry figure. We will leverage all the other functionality you have written, but of note is the return of your equivalence classes implementation.

An Example

Our overarching goal is to compute all angles and triangles in a geometry figure. Consider the now 'classic' example in Figure 1. While feasible, it is still difficult for an observer to answer "How many triangles can you identify in the figure?". As an extra challenge, now consider the task of identifying all angles in Figure 1. An angle such as $\angle BAC$ is obvious while an 'equivalent' angle $\angle DAE$ may not be identified.

In identifying all angles in a geometry figure, we need to establish some constraints and requirements. We are interested in computing all angles with measure greater than $0°$. This means we will not consider angles such as $\angle DAB$ in which the two rays of the angle overlap. However, we will compute straight angles (angles with measure 180°) such as $\angle ACE$ and $\angle DXC$. We will store our computed angles in an equivalence class structure in which angles such $\angle DAE \equiv \angle BAE \equiv \angle BAC \equiv \angle DAC$.

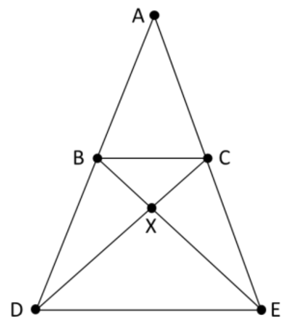

Figure 1: A geometry figure with 44 angles, 25 unique angles, and 12 triangles.

In total, Figure 1 contains `12` triangles (ΔABC, ΔBCX, ΔBXD, ΔCXE, ΔEXD, ΔADE, ΔADC, ΔAEB, ΔBCE, ΔDBC, ΔDCE, ΔDBE), `25` equivalence classes (unique angles), and `44` angles.

What You Need to Do: Implementation Details

You will implement code in the files described below. Your first task (after reading this document) is to review what code has been provided and what functionality must be implemented.

`Angle`

Angle equality will be point-based. For example, $\angle BAC$ is equal to $\angle CAB$ in Figure 1. However, $\angle BAC$ is equivalent to $\angle DAE$ but $\angle BAC$ is *not* equal to $\angle DAE$ since the endpoints of the angle do not align.

`AngleStructureComparator`

In order to properly represent angles in an equivalence class structure, we must define a `Comparator` class. Our implementation of `compare` will determine *structural equality* of angles. However, our definition will deviate from normal implementation of compare as it will return one of the values described in Table 1.

Table 1: How to interpret structural comparison of angles: `compare(left, right)`.

Value	Description
`Integer.MAX_VALUE`	Two angles are structurally equivalent when they share the same vertex and their corresponding rays overlap; otherwise, two angles are considered *structurally incomparable*. This is an error value indicating two angles (`left` and `right`) are structurally incomparable. For example in Figure 1, $\angle BDE$ is not structurally comparable to $\angle BDX$ even though they share the same vertex and one ray.
`1`	When angles `left` and `right` are structurally comparable and both rays for the `left` angle are *greater than or equal* in length to the corresponding rays in the `right` angle. For example, $\angle DAE$ is structurally larger than $\angle DAC$ in Figure 1.
`-1`	Similarly, when angles `left` and `right` are structurally comparable and both rays for the `left` angle are *less than or equal* in length to the corresponding rays in the `right` angle. For example, $\angle DAC$ is structurally smaller than $\angle DAE$ in Figure 1.
`0`	Consider two angles such as `left` = $\angle DAC$ and `right` = $\angle BAE$. The two angles are structurally comparable with vertex `A`, but one ray in the `left` angle (AD) is larger than its corresponding ray in the `right` angle (AB) while `left` ray (`AC`) is smaller than its corresponding ray in the `right` angle (`AE`). In this case, we cannot clearly state that one angle is structurally 'smaller' or larger' than another.

`AngleLinkedEquivalenceClass`

This class will inherit (`AngleLinkedEquivalenceClass extends LinkedEquivalenceClass<Angle>`) from your prior implementation of `LinkedEquivalenceClass` and override *only applicable methods* to account for our comparator values from Table 1. Note how our inheritance relationship specifies `Angle` as our generic type. This means `AngleLinkedEquivalenceClass` will only use `Angle` objects and thus easily integrate an `AngleStructureComparator` into its functionality.

The only other issue we must consider in our implementation is our choice of canonical angle. For our implementation, the canonical angle will always be the *smallest structural angle*. For example, we observe the equivalence class of four angles $\angle DAE \equiv \angle BAE \equiv \angle BAC \equiv \angle DAC$ in Figure 1; our choice of canonical angle will be $\angle BAC$ since $\angle BAC$ is structurally smaller than the other three angles.

`AngleEquivalenceClass`

Similar to our `AngleLinkedEquivalenceClass`, the `AngleEquivalenceClasses` will inherit from `EquivalenceClasses` (`AngleEquivalenceClasses extends EquivalenceClasses<Angle>`). This class will inherit from your prior implementation of `EquivalenceClasses` and override *only applicable method(s)* to account for our use of `AngleLinkedEquivalenceClass`.

`AngleIdentifier` and `TriangleIdentifier`

Your task with these two classes is to develop an algorithm to compute all angles and triangles in a geometry figure.

Testing

Two tests classes have been provided as well. *Your tests must go beyond what is provided and use other figures*. In writing other tests, please clean up the provided functionality by, for example, mitigating redundancy of methods by moving some of the functionality to a shared testing class (e.g., have these test classes inherit from the same class). Our goal here is to provide a framework to test with more extensive figures.

Commenting

Comment well. See old lab instructions for details.

Submitting: Source Code

For this lab, you will demonstrate your source code to the instructor, in person. Be ready to demonstrate (1) successful execution of all junit tests and (2) the github repository which includes commented source code (see above) and a clear commit history with meaningful commit messages *from all group members*.