COMPSCI 2ME3 (Fall 2025)

Assignment 1: Implementing Geometric Elements in Java

Due: Friday, October 10th 11:59pm

This assignment asks you to implement various geometric elements with provided descriptions in Java. This assignment worths **10%** toward your final grade and will be graded out of **100 points**.

This assignment is to be done **individually**. Turning in a solution that **does not reflect your individual work and understanding** is considered **academic dishonesty** (see syllabus for full academic dishonesty policies). However, you **are encouraged** to seek help from the course staff using proper communication channels (any assignment related questions should be asked **in the dedicated channel in the class Teams group**, but do not share your code directly in the channel). Start your work early so you will be able to ask questions and reach out for help.

GitHub Set Up and Submit Instructions

Follow these steps for setting up your directory for this assignment:

1. Open the GitHub Classroom link for this assignment: https://classroom.github.com/a/opDtglQH
2. Click the Accept this assignment button.
3. You should see a page saying that "Your assignment repository has been created," or something to that effect. There will be a repository url that looks something like this:

   ```
   https://github.com/mac-compsci2me3/assignment_name-your_github_name.
   ```

   Click that link.
4. Click the big green Code button to clone the repository for a copy on your own computer.

You will save all the files you create and changes you make to the files as part of this assignment inside this directory. You will submit your assignment by pushing your work to GitHub and remember to also "save" your work periodically by making commits and pushing your progress to GitHub as you work.

Instructions

For this assignment, please implement the classes described in the following, using Java programming language:

`1. AbstractElement:` This should be an abstract class that serves as a base for other geometric element classes (e.g., point, line, circle). It keeps track of the total number of instances created for all geometric elements.

- **Attributes:**
  - `numberOfInstances`: This attribute should track the number of instances that created from this geometric element.
- **Methods:**
  - `getNumOfInstances()`: Returns the total number of instances of this geometric element.

`2. CollisionDetector:` This should be an interface for detecting collisions or intersections between different geometric elements. Any class implementing this interface must provide functionality and actual implementation to check intersections with other geometric elements.

- **Methods:**
  - `intersect(Point)`: Checks if this geometric element intersects with a given `Point`.
  - `intersect(LineSeg)`: Checks if this geometric element intersects with a given `LineSeg`.
  - `intersect(Rectangle)`: Checks if this geometric element intersects with a given `Rectangle`.
  - `intersect(Circle)`: Checks if this geometric element intersects with a given `Circle`.

`3. Point:` This should be a concrete class that represent a 2D point with x and y coordinates, extending `AbstractElement` class and implementing `CollisionDetector` interface. It is capable of detecting intersections with other geometric elements and tracks the number of `Point` instances created.

- **Attributes:**
  - `x`: The x-coordinate of the point.
  - `y`: The y-coordinate of the point.
  - `numberOfInstances`: Tracks the number of `Point` instances created.
- **Constructors:**
  - `Point()`: Default constructor that sets the point at (0, 0).
  - `Point(x, y)`: Constructor that sets the point at the given (x, y) coordinates.
- **Methods:**
  - `getX()`: Returns the x-coordinate of the point.
  - `getY()`: Returns the y-coordinate of the point.
  - `getNumOfInstances()`: Returns the total number of `Point` instances.
  - Intersection methods as specified by `CollisionDetector`.

`4. LineSeg:` This should be a concrete class that represents a line segment defined by two points (begin and end), which extends `AbstractElement` class and implements `CollisionDetector` interface. This class can detect intersections with other geometric elements and maintains a count of all `LineSeg` instances created.

- **Attributes:**
  - `begin`: The starting point of the line segment.
  - `end`: The ending point of the line segment.
  - `numberOfInstances`: Tracks the number of `LineSeg` instances created.

- **Constructors:**
  - `LineSeg()`: Default constructor that initializes the line segment.
  - `LineSeg(begin, end)`: Constructor that initializes the line segment with given start and end points.
- **Methods:**
  - `getBegin()`: Returns the starting point.
  - `getEnd()`: Returns the ending point.
  - `getNumOfInstances()`: Returns the total number of `LineSeg` instances.
  - Intersection methods as specified by `CollisionDetector`.

`5. Rectangle:` This should be a concrete class that represents a rectangle defined by its left, right, top, and bottom boundaries (i.e., four numbers), and hence the four corners of this rectangle will be (left, top), (left, bottom), (right, top), and (right, bottom). This class should extend `AbstractElement` class and implement `CollisionDetector` interface. The class can determine intersections with other geometric elements and keeps track of the number of `Rectangle` instances created.

- **Attributes:**
  - `left`: The left boundary of the rectangle.
  - `right`: The right boundary of the rectangle.
  - `top`: The top boundary of the rectangle.
  - `bottom`: The bottom boundary of the rectangle.
  - `numberOfInstances`: Tracks the number of `Rectangle` instances created.
- **Constructors:**
  - `Rectangle()`: Default constructor.
  - `Rectangle(left, right, top, bottom)`: Constructor that initializes the rectangle with the specified boundaries.
- **Methods:**
  - `getLeft()`: Returns the left boundary.
  - `getRight()`: Returns the right boundary.
  - `getTop()`: Returns the top boundary.
  - `getBottom()`: Returns the bottom boundary.
  - `getNumOfInstances()`: Returns the total number of `Rectangle` instances.
  - Intersection methods as specified by `CollisionDetector`.

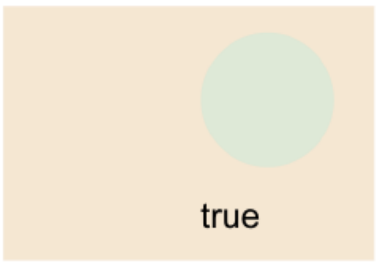
`6. Circle:` This should be a concrete class that represents a circle defined by a center point and a radius. The class should extend `AbstractElement` class and implement `CollisionDetector` interface. This class is capable of checking intersections with other geometric elements and keeps track of the number of `Circle` instances created.

- **Attributes:**
  - `center`: The center point of the circle.
  - `radius`: The radius of the circle.
  - `numberOfInstances`: Tracks the number of `Circle` instances created.
- **Constructors:**

  - `Circle()`: Default constructor.
  - `Circle(center, radius)`: Constructor that initializes the circle with the given center point (`Point` type) and radius (a number).
- **Methods:**
  - `getCenter()`: Returns the center of the circle.
  - `getRadius()`: Returns the radius.
  - `getNumOfInstances()`: Returns the total number of `Circle` instances.
  - Intersection methods as specified by `CollisionDetector`.

Some additional notes:

- Put your implemented `AbstractElement, CollisionDetector, Rectangle, LineSeg, Point, Circle` in separate java files and placed under the `src` folder in the directory (you should have a total of 6 java files in your `src` folder).
- You can consider using `float` type for all numbers associated with the geometric elements (e.g., `x, y, center, radius`).
- You can assume all geometric elements will be created in a valid way, e.g., radius of a circle will be positive, the beginning point and the ending point of a line segment will not coincide (so you don't need to handle exception cases for this assignment yet).
- When implement the intersection methods, you can assume all objects created from `Circle` and `Rectangle` are solid, like the examples in the following:



  in the example on the left above, `rectangle.intersect(circle)` and `circle.intersect(rectangle)` should return **true**. If two objects connect at the edges or corners, which is the example on the right, `rectangle.intersect(circle)` and `circle.intersect(rectangle)` should return **false**.
- A simple example of test code `testElements.java` is provided in on Avenue. You can also add your own test cases for testing various intersection cases for different geometric elements.
- Make sure your code **can compile and run** in your submission. During grading, TA's manual intervention to fix any compilation errors will cause a penalty.
- Make sure **all six java files** are included in your repository `(AbstractShape.java, CollisionDetector.java, Point.java, LineSeg.java, Rectangle.java, Circle.java)`.
- Make sure you **commit and push your changes** when you complete your assignment.

Bonus

*The bonus part will **ONLY** be considered part of your grading if all of your intersect methods and geometric elements classes are correctly implemented. From a grade point of view, it is more efficient to focus on your work from the main instructions than rushing the bonus step.*

You can get an extra 10 points if there is no redundant implementation for `intersect` methods. For example, intersection of rectangle and circle does not need to be implemented twice in `Rectangle.intersect(Circle)` and `Circle.intersect(Rectangle)`. Find a way to avoid redundancy.

Grading Scheme

This assignment is graded according to several dimensions, classified into four categories: 1) program basic setups, 2) the correctness of implementation, and 3) coding style. The following table show a summary of these dimensions with their associated marks:

Category	Dimensions	Points
Program Basics	All Java files are set up correctly in the project folder.	6
	All required content is set up correctly based on the provided instructions and requirements.	50
	Project can compile and run without errors.	10
Implementation Correctness	All attributes and methods are implemented correctly (e.g., numberOfInstances, intersect methods). We will run various test cases to test the implementation of these methods (similar to ones we provided in the starter code).	30
Coding Style	Codes are clear and readable, following good naming convention in Java, code blocks have appropriate indentation, etc.	4
Extra	If all implementations are correct and no duplicate code used in the implementation of the intersect methods.	10
	Total Points for A1:	**100 + 10**