Project Overview: Collinear Points Finder

The goal of this project is to identify every maximal line segment (a straight line drawn between two endpoints) in a two-dimensional plane that contains 4 or more collinear points. In doing so, you will need to create and use several classes that work together using a sorting-based approach. The project is split into distinct components, each responsible for a specific part of the overall task.

Components and Their Method Signatures

1. Point Class

This class represents a point in the plane, defined by its x and y coordinates. Each point is immutable and must be validated to ensure that both coordinates are non-negative.

- **Constructor:**
  - **Signature:** *Point(x, y)*
  - **Description:** Creates a new point with the specified x and y coordinates. Both coordinates must be non-negative integers.
- **draw Method:** (optional)
  - **Signature:** *draw()*
  - **Description:** Displays the point graphically. This method will typically interact with a drawing library.
- **drawTo Method:** (optional)
  - **Signature:** *drawTo(anotherPoint)*
  - **Description:** Draws a line segment from the current point to another specified point.
- **toString Method:**
  - **Signature:** *toString()*
  - **Description:** Returns a string representation of the point (for example, in the format "(x, y)").
- **compareTo Method:**
  - **Signature:** *compareTo(anotherPoint)*
  - **Description:** Compares this point with another point. The comparison is based first on the y-coordinate, and if these are equal, then on the x-coordinate. This method returns an

integer that indicates the order.

- **slopeTo Method:**
  - **Signature:** *slopeTo(anotherPoint)*
  - **Description:** Calculates the slope between this point and another point. Special cases include:
    - If the two points are identical, it returns a designated value for a degenerate line segment.
    - If the line formed is vertical, it returns a value indicating positive infinity.
    - If the line is horizontal, it returns positive zero.
- **slopeOrder Method:**
  - **Signature:** *slopeOrder()*
  - **Description:** Returns a comparator (or comparable function) that can be used to sort a list of points based on the slopes they make with this point.

2. LineSegment Class (Code provided)

This class represents a line segment defined by two endpoints. It ensures that both endpoints are valid (i.e., not null and not the same point).

- **Constructor:**
  - **Signature:** *LineSegment(point1, point2)*
  - **Description:** Creates a new line segment with the two given endpoints. It validates that neither of the endpoints is null and that they are distinct.
- **toString Method:**
  - **Signature:** *toString()*
  - **Description:** Returns a string representation of the line segment, typically showing the endpoints in order (from the smallest to the largest point).
- **hashCode Method:**
  - **Signature:** *hashCode()*
  - **Description:** (Not supported in this project) – Students are informed that this method is intentionally not implemented.

3. FastCollinearPoints Class

This class is responsible for detecting all line segments that contain 4 or more collinear points. It uses a sorting-based approach where each point is treated as an origin to sort the other points by the slopes they form with it.

- **Constructor:**
  - **Signature:** *FastCollinearPoints(points)*
  - **Description:** Accepts an array or list of points as input. It first performs a defensive copy of the input, validates that no point is null, and ensures there are no duplicate points. Then, it uses the sorting-based algorithm to find all line segments that include 4 or more collinear points.
- **numberOfSegments Method:**
  - **Signature:** *numberOfSegments()*
  - **Description:** Returns the number of line segments found during the processing. This is typically an integer.
- **segments Method:**
  - **Signature:** *segments()*
  - **Description:** Returns an array or list containing all the line segments that were identified. Each segment should be reported only once, from the smallest to the largest point in the sorted order.

4. Solution Class (Java) / Main Function (Python)

This component serves as the entry point of the application and is responsible for reading input, instantiating the appropriate classes, and printing the detected segments.

- **Main Entry Point:**
  - **Signature (Java):** *main(args)*
  - **Signature (Python):** *main()*
  - **Description:**
    - Reads the number of points and the coordinates for each point from input.
    - Creates the necessary point objects.
    - Instantiates the FastCollinearPoints class with these points.
    - Retrieves and prints the detected line segments.
    - Handles exceptions gracefully by printing error messages if any input validations fail.