Programming and Data Structures
Exam #2

Programming Project

1. Define a generic class Pair with two generic types as described in the UML below:

Pair<E1, E2>	Generic class with two generic types
-first: E1	Data member of type E1
-second: E2	Data member of type E2
+Pair(E1 f, E2 s)	Initialize first to f and second to s
+getFirst(): E1	Returns first
+getSecond(): E2	Returns second
+setFirst(E1 f): void	Sets first to f
+setSecond(E2 s): void	Sets second to s
+equals(Object obj): boolean	Returns true if two pairs have identical first and second members
+toString(): String	Returns the members first and second between parentheses and separated by a comma (first, second)

2. Define a class **ShapeAL** that has the following UML diagram:

ShapeAL	
-points: ArrayList<Pair<Integer, Integer>>	Array list of the points of the shape. Each point is a pair of two integers (x coordinate and y coordinate). Use java.util.ArrayList
+ShapeAL()	Creates an empty list points with capacity 10
+add(Pair<Integer, Integer> p): void	Adds the point p to the list of points
+isClosed(): boolean	Returns true if the first point of the shape is identical to the last point
+containsPoint(Pair<Integer, Integer> p)	Returns true if **p** is found in the list of points. This method should be **recursive, use linear search, and use an iterator** to visit the elements of the list
+toString(): String	Returns the list of points of the shape as a string

3. Define a class ShapeLL that has the following UML diagram:

ShapeLL	
-points: LinkedList<Pair<Integer, Integer>>	Linked list of the points of the shape. Each point is a pair of two integers (x coordinate and y coordinate). Use java.util.LinkedList
+ShapeLL()	Creates an empty list points

`+add(Pair<Integer, Integer> p): void`	Adds the point p to the list of points
`+isClosed(): boolean`	Returns true if the first point of the shape is identical to the last point
`+containsPoint(Pair<Integer, Integer> p)`	Returns true if p is found in the list of points of the shape. This method should be **recursive, use linear search, and use an iterator** to visit the elements of the list
`+toString(): String`	Returns the list of points of the shape as a string

4. Define a class **`Test`** to test the classes.
    a. Create an instance of class **`ShapeAL`** and name it **`crescent`**.
    b. Create an instance of class **`ShapeLL`** and name it **`hexagon`**.
    c. Add the following points to the shape **`crescent`**: (30, 50), (25, 40), (25, 30), (30, 20), and (40,10)
    d. Add the following points to the shape **`hexagon`**: (50, 60), (40, 40), (50, 20), 70, 20), (90, 40), (70, 60), and (50, 60)
    e. Display the points of **`crescent`**. Search for the point (50, 60) in **`crescent`** by calling **`containsPoint()`** and display an appropriate message. Check if **`crescent`** is closed by calling **`isClosed()`** and display a message.
    f. Display the points of **`hexagon`**. Search for the point (50, 60) in **`hexagon`** by calling **`containsPoint()`** and display an appropriate message. Check if **`hexagon`** is closed by calling **`isClosed()`** and display a message.

A sample run is provided below for testing.

Submit the files **`Pair.java`**, **`ShapeAL.java`**, **`ShapeLL.java`**, and **`Test.java`**. Javadoc comments are not required.

```
Shape Crescent: [(30, 50),(25, 40),(25, 30),(30, 20),(40, 10)]
Shape crescent does not contain the point (50, 60)
Shape crescent is open.
Shape Hexagon: [(50, 60),(40, 40),(50, 20),(70, 20),(90, 40),(70, 60),(50,
60)]
Shape hexagon contains the point (50, 60)
Shape hexagon is closed.
```