1. **ALGORITHM**: *spatialIntersectionPoint*
2. **INPUT**: Point2D objects *pointLhs* and *pointRhs*
3. **OUTPUT**: Point2D object *intersectionPoint*
4. **BEGIN**
5. Point2D *intersectionPoint*;
6. **if** checkIntersection(*pointLhs*, *pointRhs*) = TRUE **then**
7. selectFirst(*pointLhs, pointRhs, object, status*);
8. **while** *object != none* ***and*** *status = endOfNone* **do**
9. **if** *object = both* **then**
10. Point2D.addPoint(*intersectionPoint*, *pointLhs.currentPoint*);
11. **endif**
12. selectNext(*pointLhs, pointRhs, object, status*);
13. **endwhile**
14. **endif**
15. **return** *intersectionPoint*;
16. **END** *spatialIntersectionPoint*

The *spatialIntersectionPoint* algorithm takes in two Point2D objects *pointLhs* and *pointRhs* and computes the spatial intersection between them and returns a Point2D object *intersectionPoint* containing this intersection.

In line 6, we first check with a simple boolean function whether the two objects have an intersection. If so, then we proceed to lines 7-13.

First, we call the selectFirst function (line 7) passing the two point objects, which would in turn update the value of *object* with *none, both, first* or *second,* and the value of *status* with *endOfNone, endOfFirst, endOfSecond* or *endOfBoth.*

Only if the value of *object* is *both* would the line 10 execute. What this means is that the current pointers for both the Point2D objects under consideration have the same simple points (same (x,y) coordinates). If so, add this point to the *intersectionPoint* Point2D object being calculated.

The *selectNext* function call updates the respective pointer(s) of the object(s) according to the value in *object.* This is captured by line 12.

We do this as long as we don’t reach the end of either of the objects, which is captured by line 8.

After the complete traversal of the either/both of the two objects, we return the calculated *intersectionPoint* object (line 15) containing the required intersection points.