General programming problems

Often coding interviews involve problems which do not involve complicated algorithms or data structures - these are straight programming problems

These are problems you should nail

All they need is practice

They test your ability to work through details and get the edge cases right

The bad thing about them is that they involve lots of cases which you need to work through - they tend to be frustrating

The great thing about them is that if you are organised and systematic in your thought process it is reasonably straightforward to get them right

Let’s start practising…

We’ll solve 8 general programming problems here - they all use arrays or simple data structures which you create

None of them require any detailed knowledge of standard algorithms

All of these are solvable from first principles

In this class we’ll solve these using Java, however you can use any language to get to the solution

Check whether a given string is a palindrome

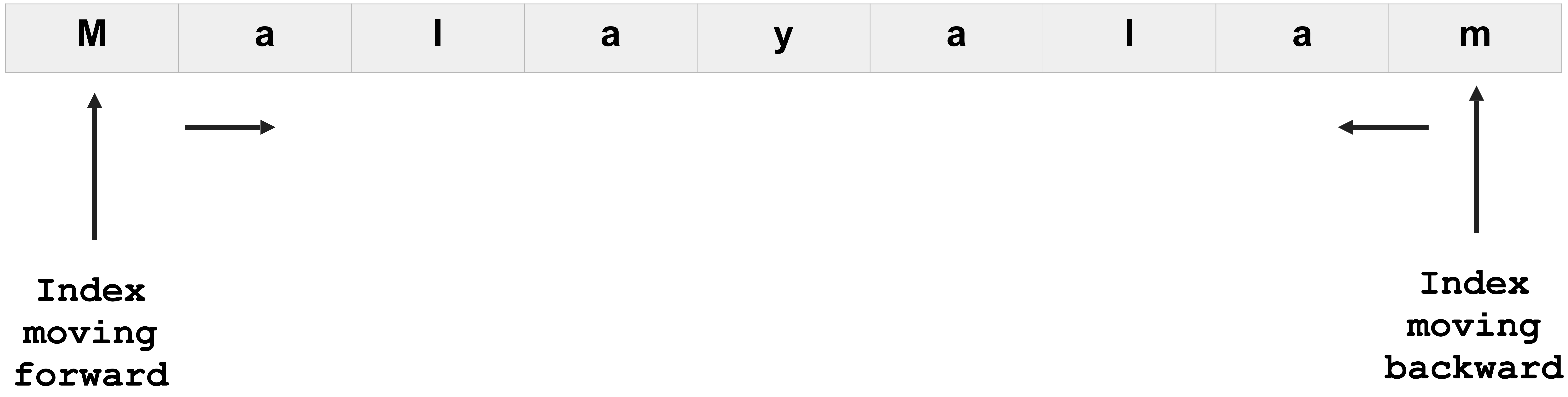
Palindromes are strings which read the same when read forwards or backwards

You can reverse all the letters in a palindrome and get the original string

Examples of palindromes are: “madam”, “refer”, “lonely tylenol”

Note:
1. The string can have spaces, ignore spaces in the palindrome check, the spaces can all be collapsed.
2. the check is case in-sensitive i.e. Malayalam with uppercase M is also a palindrome

PALINDROME ALGORITHM

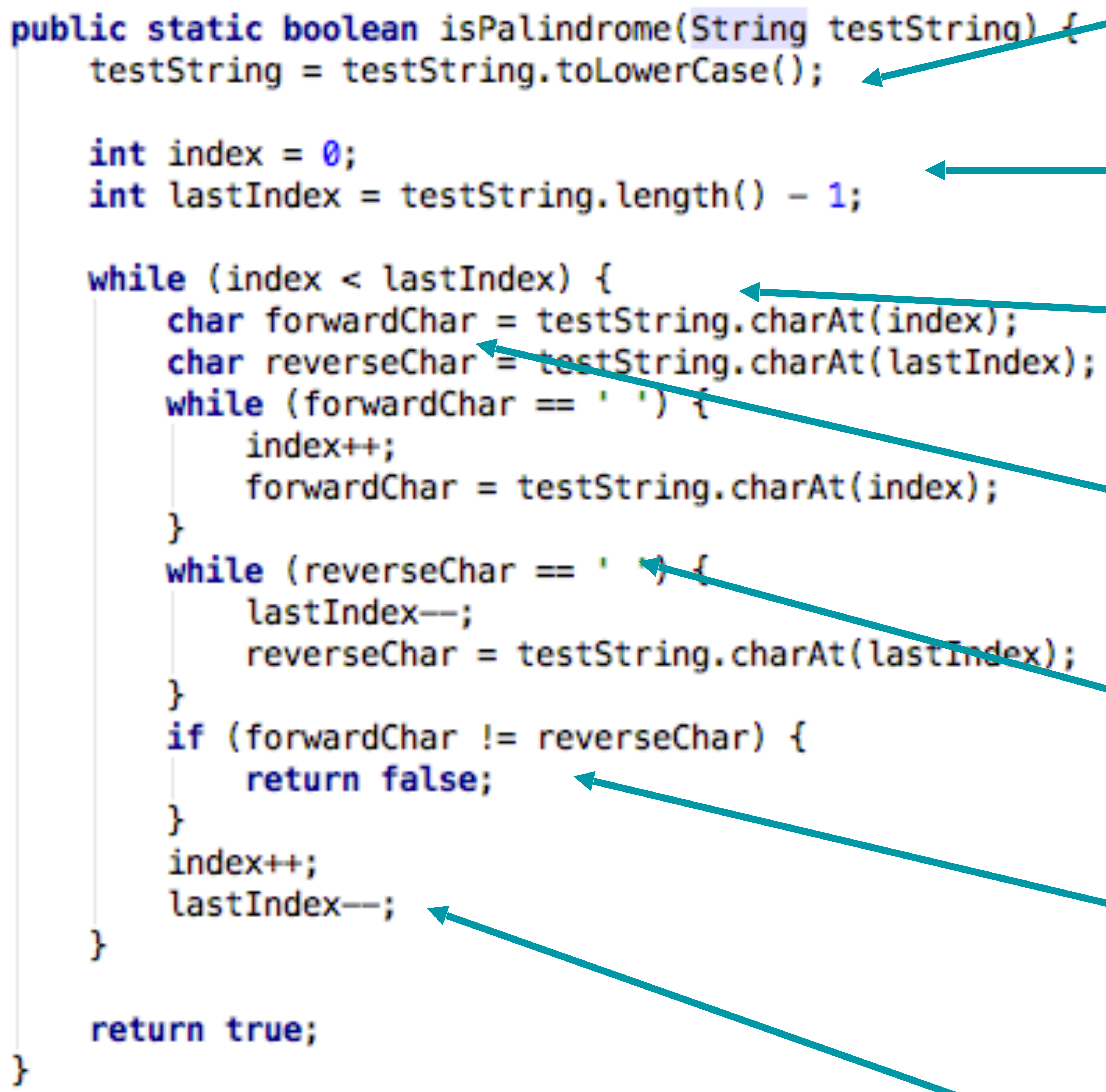


SET UP 2 INDICES, ONE MOVING FROM THE BEGINNING TOWARDS THE CENTER AND ANOTHER MOVING FROM THE END TOWARDS THE CENTER

COMPARE THE CHARACTERS FOR EVERY INDEX

REMEMBER TO ACCOUNT FOR WHITE SPACES, THEY SHOULD BE IGNORED

CHECK FOR PALINDROME



```java
public static boolean isPalindrome(String testString) {
    testString = testString.toLowerCase();

    int index = 0;
    int lastIndex = testString.length() - 1;

    while (index < lastIndex) {
        char forwardChar = testString.charAt(index);
        char reverseChar = testString.charAt(lastIndex);
        while (forwardChar == ' ') {
            index++;
            forwardChar = testString.charAt(index);
        }
        while (reverseChar == ' ') {
            lastIndex--;
            reverseChar = testString.charAt(lastIndex);
        }
        if (forwardChar != reverseChar) {
            return false;
        }
        index++;
        lastIndex--;
    }

    return true;
}
```

MAKE THE COMPARISON CASE INSENSITIVE BY CONVERTING THE ENTIRE STRING TO THE SAME CASE

START AT BOTH ENDS OF THE STRING TO START COMPARING CHARACTER BY CHARACTER

THE WHILE LOOP CONTINUES TILL THE INDICES MEET OR PASS EACH OTHER

GET THE CHARACTERS FROM THE BEGINNING AND END OF THE STRING TO COMPARE THEM

IGNORE WHITE SPACES IN CHECKING FOR A PALINDROME, WE SKIP OVER ALL WHITE SPACES AND DO NOT COMPARE THEM

THE MAIN COMPARISON LOGIC, RETURN FALSE IF WE FIND A SINGLE MISMATCH

INCREMENT THE INDEX MOVING FORWARD AND DECREMENT THE INDEX MOVING BACKWARD

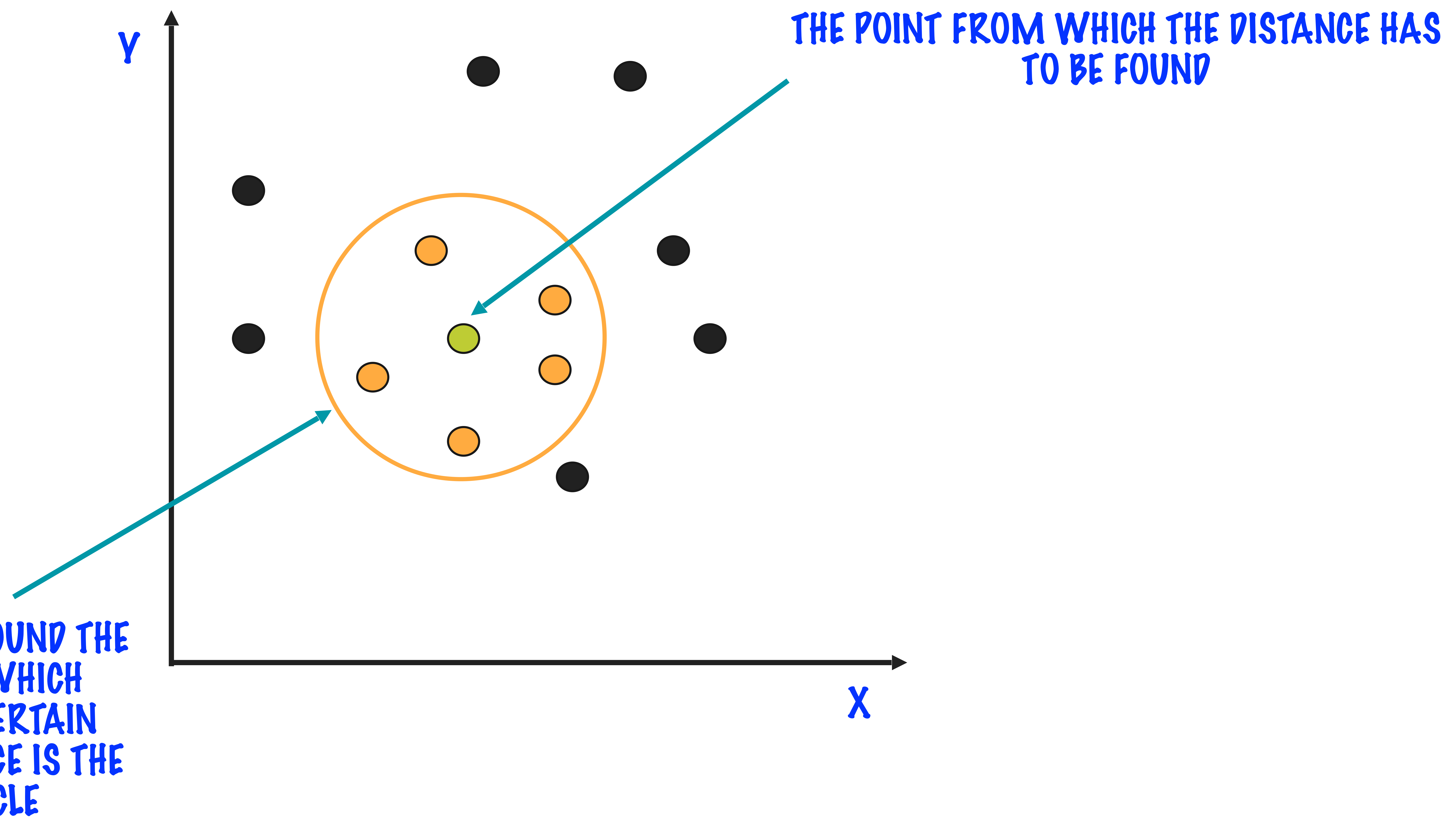
Find all points within a certain distance of another point

Given a list of points (x, y coordinates of the point) find all points which are within a certain distance of another point. The distance and the central point is specified as an argument to the function which computes the list of points in range

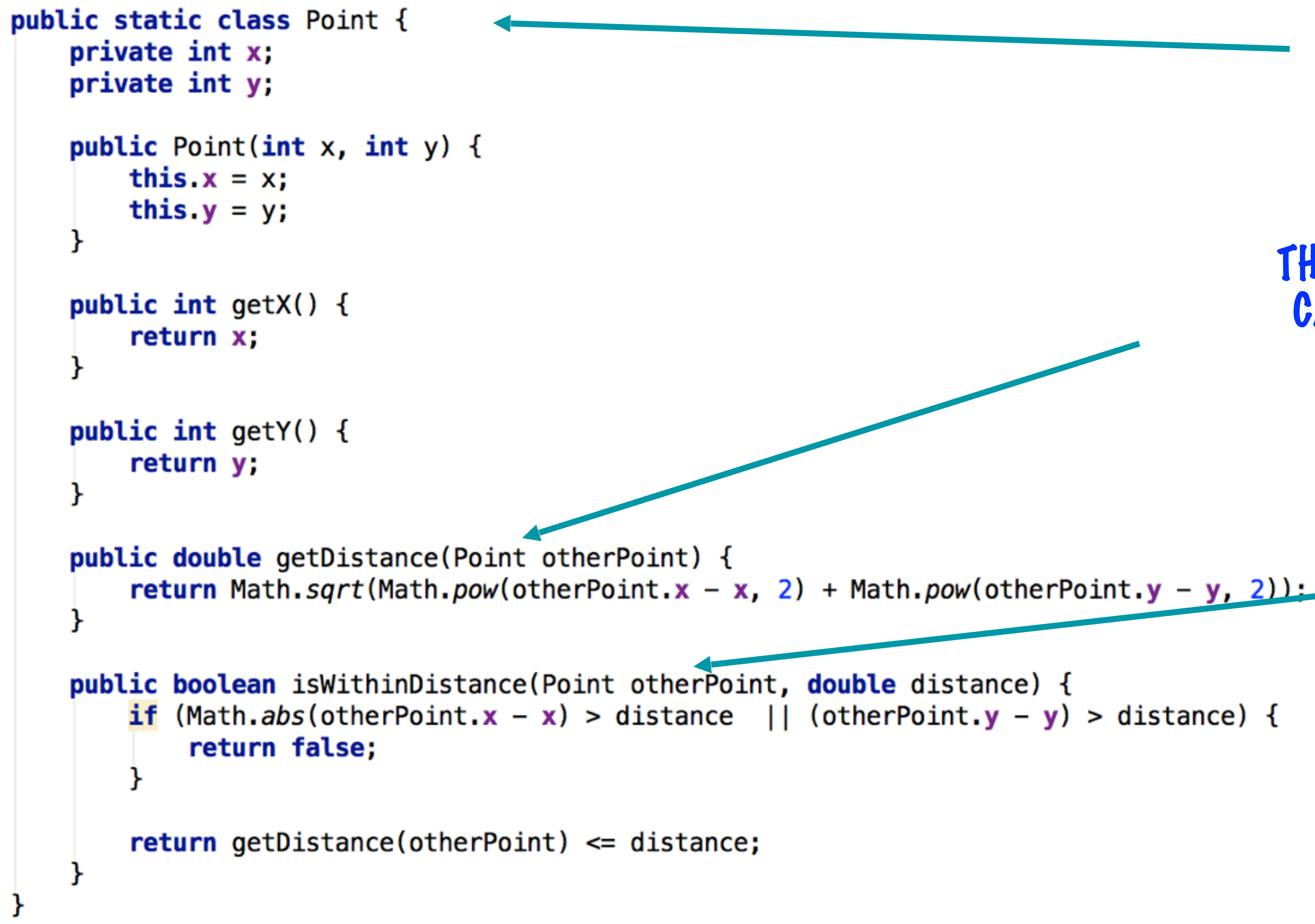
Example, all points within a distance 10 of (0.0) will include the point at (3, 4) but not include the point at (12, 12)

HINT: If you are using an OO programming language set up an entity which represents a point and contains methods within it to find the distance from another point

FINDING POINTS WITHIN A CERTAIN DISTANCE


Y
THE POINT FROM WHICH THE DISTANCE HAS TO BE FOUND
X
IMAGINING A CIRCLE AROUND THE CENTRE WILL SHOW WHICH POINTS LIE WITHIN A CERTAIN DISTANCE - THAT DISTANCE IS THE RADIUS OF THE CIRCLE

GET POINTS WITHIN A DISTANCE - THE POINT CLASS



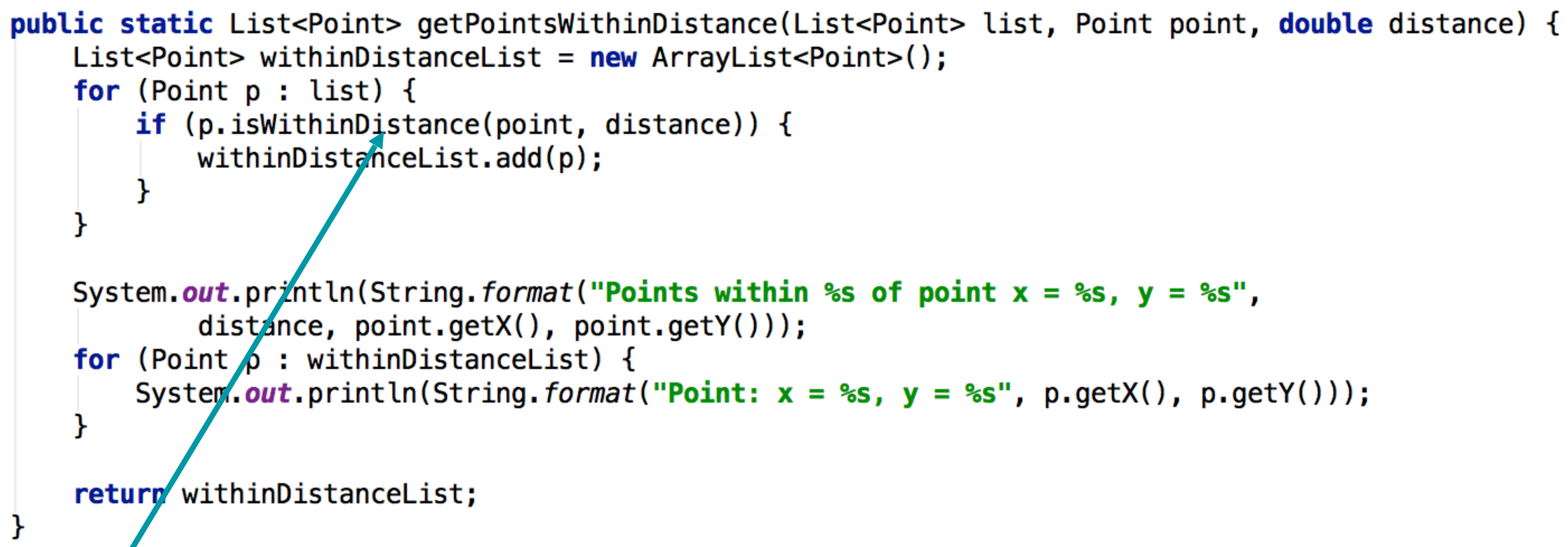
```java
public static class Point {
    private int x;
    private int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public double getDistance(Point otherPoint) {
        return Math.sqrt(Math.pow(otherPoint.x - x, 2) + Math.pow(otherPoint.y - y, 2));
    }

    public boolean isWithinDistance(Point otherPoint, double distance) {
        if (Math.abs(otherPoint.x - x) > distance  || (otherPoint.y - y) > distance) {
            return false;
        }

        return getDistance(otherPoint) <= distance;
    }
}
```

SET UP A CLASS TO REPRESENT POINT, THIS PROVIDES AN ENCAPSULATION WHICH HOLDS THE X AND Y COORDINATES

THIS CAN ALSO HOLD THE HELPER METHODS TO CALCULATE DISTANCE FROM ANOTHER POINT

USE A FEW SHORTCUTS WHEN CALCULATING WHETHER THE POINT IS WITHIN A CERTAIN DISTANCE, IF THE POINT EXCEEDS THE DISTANCE ON ANY ONE OF THE AXIS IT MEANS IT WILL NOT BE IN RANGE SO WE CAN RETURN FALSE RIGHT AWAY

GET POINTS WITHIN A DISTANCE - THE CODE



```java
public static List<Point> getPointsWithinDistance(List<Point> list, Point point, double distance) {
    List<Point> withinDistanceList = new ArrayList<Point>();
    for (Point p : list) {
        if (p.isWithinDistance(point, distance)) {
            withinDistanceList.add(p);
        }
    }

    System.out.println(String.format("Points within %s of point x = %s, y = %s",
            distance, point.getX(), point.getY()));
    for (Point p : withinDistanceList) {
        System.out.println(String.format("Point: x = %s, y = %s", p.getX(), p.getY()));
    }

    return withinDistanceList;
}
```

GO THROUGH ALL THE POINTS AND FIND THE ONES WHICH ARE WITHIN THE DISTANCE SPECIFIED

THANKS TO THE HELPER METHOD THE CODE HERE IS SHORT AND VERY READABLE