Requirements

- Implement a user-defined function (ST_CONTAINS) for Spark such that, when given a string that represents a rectangle and a string that represents a point, return whether that point is contained within the rectangle
- Implement a user-defined function (ST_WITHIN) for Spark such that, when given two strings that represent points and a range distance value, return whether the distance between the two points is within the given range distance

Design

- One of the easiest ways to deal with this problem is to model it using objects (ie. Object-Oriented Programming). This allows for us to break down a concept from real-word models to conceptual self-contained coding models.
- Recommended Scala Classes:
    - Point
        - represents a point in two-dimensional space (x and y coordinates)
        - coordinates are given as floating point numbers to 4 decimal places (ex. 43.1234). We can store these values as Doubles.
        - Note: Although the assignment will eventually look at using latitudes and longitudes of NYC cab rides, we shouldn't restrict Points to being within those bounds, as the test data provided in this assignment has illegal values (ex. latitudes of -91)
    - Rectangle
        - can be represented as two vertices (Points) in two-dimensional space
        - Since the rectangle is being represented as a four-ple of coordinates (x1, y1, x2, y2), we can model it as an object that contains two Point
        - Note: Since we're going to be dealing with containment of a Point in a Rectangle, we should be cognizant of the order of which the Points are stored, such that when we retrieve this data, we can do so in a consistent manner. The easiest way to do this is to ensure that the first vertex has the smaller of the x and y coordinates.
- For both Point and Rectangle classes it is recommended to use a factory method to create objects.

The factory method delegates construction of an object to another method (usually static). In Scala, this can be done with a companion object. The following is a sample layout for the Point class and companion object:

```
// We make the constructor private to limit access to it and make it
a
```

```
// simple data class (no logic in constructor, holds values)
class Point private(var x: Double, var y: Double) {
  // This is where all instance functions live
  ...
}

// This is the companion object - allows for static method calls
object Point {
  def fromString(pointsString: String): Point = {
    // This method parses a string of two coordinates and
    // returns a Point
    ...
  }

  def apply(x: Double, y: Double): Point = {
    // More Scala "sugar", this method will call the Point
    // constructor to return a Point.

    // It can be called either by Point.apply(x,y) or Point(x,y)

    // In other classes (ex. Rectangle), we can manipulate our
    // input parameters (ex. find min and max values for coordinates)
    // such that we have all of our logic outside of the constructor
    ...
  }
}
```

- Once we have implemented the basic classes, we can move on to implementing the functions that we need:
    - ST_CONTAINS
        - Preferred to implement as a method in the Rectangle class, using the following signature:

```
def contains(point: Point): Boolean = { ... }
```

    - ST_WITHIN
        - Preferred to implement as a method in the Point class, using the following signature:

```
def within(point: Point, rangeAllowed: Double): Boolean = { … }
```

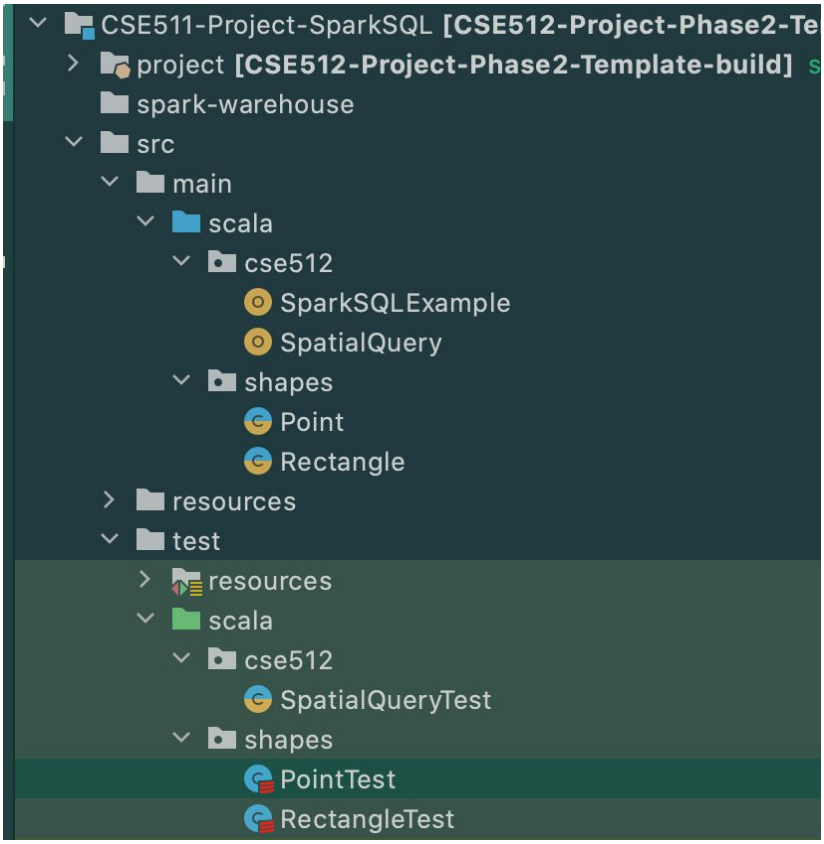
- Once these methods are implemented in the classes, we can call them from the User-Defined Function in Spark

```
    spark.udf.register(
      "ST_Contains",
      (queryRectangle: String, pointString: String) =>

Rectangle.fromString(queryRectangle).contains(Point.fromString(pointS
tring)))
```

FAQ

- Why bother with classes and why not just implement everything in the `spark.udf.register` method?
    - Testability - by breaking down the components into the smaller elements we can really easily unit test and debug our logic. We can do so much quicker than trying to run Spark, hopefully push some data in and maybe catch our errors.
    - In the build.sbt file, you'll notice that scalatest libraries are included, so they want us to test our code (good practice in the real-world also dictates that you should do this too)
- How do I write tests for my class?
    - Create the src/test/scala directory in the codebase, then create a test class, ex. PointTest



    - In the above, I've separated the shapes classes into a separate package and created tests for Point, Rectangle, and eventually SpatialQuery classes. I didn't

create one for SparkSQLExample as this is a class we don't touch and is used as the test runner when we submit

- Example Test (PointTest):

```
import org.scalatest.FunSuite

class PointTest extends FunSuite {

  test("creates a point") {
    val point = Point(10, 20)
    assert(point.x == 10)
    assert(point.y == 20)
  }

  ... more tests ...
}
```

- Tests can be run individually in IntelliJ, you can also run an entire test class, entire package of tests, and all tests (using the `sbt test` command)

Tasks - For Part 1

- Get everything installed and running in IntelliJ
- Create the base shape class Point
  - Add a static method to create a Point from a string
  - Document the class and methods using ScalaDoc
  - Add tests to validate all behaviors
- Create the base shape class Rectangle
  - Add a static method to create a Rectangle from a string
  - Document the class and methods using ScalaDoc
  - Add tests to validate all behaviors
- Create a contains method in the Rectangle class (signature above)
  - Document the method using ScalaDoc
  - Add tests to validate
- Create a within method in the Point class (signature above)
  - Document the method using ScalaDoc
  - Add tests to validate
- Implement the UDFs in the SpatialQuery class
  - Add tests to validate (this one is a little tricky, but I can show more)
- Run the example input arguments against SparkSQLExample
  - If the output values are the same as the exampleanswer, you're good!
- Submit code for automated marking (each one of us needs to do this)
- Submit source code from repo (one of us needs to do this)
- Complete and submit the Systems Documentation Report