In this exercise you were given four methods to define, focusing on transforming values passed in to the methods through parameters to produce return values, often using for loops. There were two versions of this exercise – though the scenario was a little different in both cases the structure of the solution was the same in both.

METHOD 1

The first method was described as follows:

```
/*
* DESCRIBING A PLACE
*
* Given:
*     a String, the name of place,
*     a HashMap<String,Point2D> containing a place name to location mapping, and
*     a HashMap<String,Integer> containing a place name to population mapping,
* return a String that describes that place as in the example below.
*
* You may assume that if there is an entry for String in either of the two HashMaps,
* then there is an entry for that String in both.
*
* If there is no entry for the String in either HashMap, return the string
*     "Requested place is unknown: <placename>."
* where <placename> is replaced by the requested place name.
*
* Suppose we know the following about Tonawanda City:
*
*          Tonawanda City is at 43.020335,-78.880315, according to
*          http://www.itouchmap.com/latlong.html
*
*          The population of Tonawanda City is 15,130
*          (https://en.wikipedia.org/wiki/Tonawanda_(city),_New_York)
*
* Assuming that "Tonawanda City" has an entry in both HashMaps, reflecting this
* information, the String returned must be:
*
*     "Tonawanda City has latitude 43.020335, longitude -78.880315, and has a population
*      of 15130 persons."
*
* For example, assuming that "Omicron Persei 8" does not exist in either HashMap, the
* String returned must be:
*
*     "Requested place is unknown: Omicron Persei 8."
*/
```

The provided method stub was:

```
public String description(String name,
                          HashMap<String,Point2D> location
                          HashMap<String, Integer> population) {
      String answer = "";
      return answer;
}
```

To define this method we need to look up the given name in the each of the two HashMaps, and if found create an answer String that contains the requested information. If it is not found, the "Request place is unknown: ..." String must be returned.

We can start with a skeletal implementation containing an if-else statement:

```
public String description(String name,
                          HashMap<String,Point2D> location,
                          HashMap<String, Integer> population) {
      String answer = "";
      if (location.containsKey(name)) {




      }
      else {
            answer = "Requested place is unknown: "+name+".";
      }
      return answer;
}
```

Then we retrieve the location and population associated with the name:

```
public String description(String name,
                          HashMap<String,Point2D> location,
                          HashMap<String, Integer> population) {
      String answer = "";
      if (location.containsKey(name)) {
            Point2D loc = location.get(name);
            int pop = population.get(name);


      }
      else {
            answer = "Requested place is unknown: "+name+".";
      }
      return answer;
}
```

Finally we construct the answer String:

```
public String description(String name,
                          HashMap<String,Point2D> location,
                          HashMap<String, Integer> population) {
      String answer = "";
```

```
            if (location.containsKey(name)) {
                  Point2D loc = location.get(name);
                  int pop = population.get(name);
                  answer = name + " has latitude "+loc.getX()+
                        ", longitude "+loc.getY()+
                        ", and has a population of "+pop+" persons.";
            }
            else {
                  answer = "Requested place is unknown: "+name+".";
            }
            return answer;
      }
```

METHOD 2

The second method was given to you:

```
/*
* FIND THE CLOSEST PLACE
*
* Given:
*     a Point2D, representing a location, and
*     a HashMap<String,Point2D>, containing a place name to location mapping,
* return a String that gives the place name of the closest place to that location from
* the HashMap.
*
* You may assume that the HashMap has at least one entry - this means that there is
* in fact a closest place!
*
* HINT: Use a figure larger than the Earth's equatorial circumference to seed the
* closest distance.
* https://www.space.com/17638-how-big-is-earth.html
*
* HINT: Remember that you can obtain the set of keys for which the HashMap has entries
* by calling the keySet() method on the HashMap.
*
* HINT: Keep track of both the shortest distance you've come across so far and the
* name of that place as you iterate through the loop.
*/
```

This method stub was given as a starting point:

```
      public String closestToPoint(Point2D p, HashMap<String, Point2D> location) {
            String closest = "";
            return closest;
      }
```

Taking the last hint to heart we declare two variables to keep track of the name of the closest city as well as the distance from the given point to that city:

```
      public String closestToPoint(Point2D p, HashMap<String, Point2D> location) {
            String closest = "";
```

```
        double distance =

        return closest;
    }
```

But what should we initialize distance to? There's a hint for that too: use a large enough value that no two locations on the Earth can be further apart. For example:

```
    public String closestToPoint(Point2D p, HashMap<String, Point2D> location) {
        String closest = "";
        double distance = 25000.0;

        return closest;
    }
```

Next we loop through all the keys in the HashMap, because we have to consider the distance of each place to the given point. The keys are the Strings in the location HashMap:

```
    public String closestToPoint(Point2D p, HashMap<String, Point2D> location) {
        String closest = "";
        double distance = 25000.0;
        for (String name : location.keySet()) {

        }
        return closest;
    }
```

Inside the loop we consider each city name; we get the location of the name,

```
    public String closestToPoint(Point2D p, HashMap<String, Point2D> location) {
```

```
        String closest = "";
        double distance = 25000.0;
        for (String name : location.keySet()) {
                Point2D loc = location.get(name);




        }
        return closest;
}
```

Then we compute the distance from the given point p to this location:

```
public String closestToPoint(Point2D p, HashMap<String, Point2D> location) {
        String closest = "";
        double distance = 25000.0;
        for (String name : location.keySet()) {
                Point2D loc = location.get(name);
                double x = p.distance(loc);




        }
        return closest;
}
```

If this computed distance is less than what we're already remembered that means we've found a place that's closer to the point, so we want to update our shortest distance and the name of the closest location:

```
public String closestToPoint(Point2D p, HashMap<String, Point2D> location) {
        String closest = "";
        double distance = 25000.0;
        for (String name : location.keySet()) {
                Point2D loc = location.get(name);
                double x = p.distance(loc);
                if (x < distance) {
                        distance = x;
                        closest = name;
                }
        }
        return closest;
}
```

After the loop completes closest will contain the name of the place closest to the given point.

METHOD 3

The third method was described as follows:

```
/*
* FIND THE LARGEST PLACE WITHIN RADIUS
*
* Given:
*     a Point2D, a location,
*     a double, a radius,
*     a HashMap<String,Point2D> containing a place name to location mapping, and
*     a HashMap<String,Integer> containing a place name to population mapping,
* return a String that gives the place name of the most populous place within the
* given radius to the given location.
*
* If there is no place within the radius of the given location, return an empty
* String, "".
*
* HINT: Remember that you can obtain the set of keys for which the HashMap has entries
* by calling the keySet() method on the HashMap.
*
* HINT: Keep track of both the largest population you've come across so far and the
* name of that place as you iterate through the loop.
*/
```

Here's our starting point.

```
public String largestToPoint(Point2D pt, double radius,
              HashMap<String, Point2D> location,
              HashMap<String,Integer> population) {
       String largest = "";
       return largest;
}
```

The structure of this method will be similar to that of the other one. We'll start by introducing a variable to keep track of the largest population we've come across so far:

```
public String largestToPoint(Point2D pt, double radius,
              HashMap<String, Point2D> location,
              HashMap<String,Integer> population) {
       String largest = "";
       int pop = 0;
```

```
        return largest;
    }
```

Then we loop through the names in the HashMap of locations:

```
    public String largestToPoint(Point2D pt, double radius,
            HashMap<String, Point2D> location,
            HashMap<String,Integer> population) {
        String largest = "";
        int pop = 0;
        for (String name : location.keySet()) {









        }
        return largest;
    }
```

Because we are looking for the most populous place within a given radius of the given point, we get the location of name, compute the distance from the point to the name's location, and check whether it lies within the given radius:

```
    public String largestToPoint(Point2D pt, double radius,
            HashMap<String, Point2D> location,
            HashMap<String,Integer> population) {
        String largest = "";
        int pop = 0;
        for (String name : location.keySet()) {
            Point2D loc = location.get(name);
            double x = pt.distance(loc);
            if (x < radius) {









            }
        }
        return largest;
    }
```

If it does, we check whether this place has larger population than we've already found:

```
    public String largestToPoint(Point2D pt, double radius,
```

```
                HashMap<String, Point2D> location,
                HashMap<String,Integer> population) {
        String largest = "";
        int pop = 0;
        for (String name : location.keySet()) {
                Point2D loc = location.get(name);
                double x = pt.distance(loc);
                if (x < radius) {
                        int p = population.get(name);
                        if (p > pop) {


                        }
                }
        }
        return largest;
}
```

If it does, we update both the name of the population of the largest place we've seen so far:

```
public String largestToPoint(Point2D pt, double radius,
                HashMap<String, Point2D> location,
                HashMap<String,Integer> population) {
        String largest = "";
        int pop = 0;
        for (String name : location.keySet()) {
                Point2D loc = location.get(name);
                double x = pt.distance(loc);
                if (x < radius) {
                        int p = population.get(name);
                        if (p > pop) {
                                largest = name;
                                pop = p;
                        }
                }
        }
        return largest;
}
```

After the loop completes largest will contain the name of the most populous place within the radius of the given point.

METHOD 4

The fourth method was described as follows:

```
/*
* FIND THE LONGEST DISTANCE BETWEEN A HASHMAP OF PLACES
*
* Given:
*     a HashMap<String,Point2D> containing a place name to location mapping
* return a double, containing the maximum distance between any two points in the
```

```
* HashMap.
*
* For example:
* Tonawanda City has latitude 43.020335, longitude -78.880315, and
* Davis Hall has latitude 43.00277, longitude -78.78731, and the distance between them
* is  0.09464913760831883.  If these were the two most distant points in the HashMap
* the method should return 0.09464913760831883.
*
* If there are fewer than two entries in the HashMap, return 0.0.
*
* HINT: Remember that you can obtain the set of values for which the HashMap has keys
* by calling the values() method on the HashMap.
*
* HINT: Use a loop within a loop, a so-called nested loop, to check all pairs of
* points in the HashMap.
*/
```

We begin with a basic starting point:

```
public double longestDistance(HashMap<String, Point2D> location) {
	double distance = 0.0;
	return distance;
}
```

In this method we have to do pairwise comparisons between all the locations to find the two that are furthest apart. To do that we need a ***nested loop*** (one loop inside another):

```
public double longestDistance(HashMap<String, Point2D> location) {
	double distance = 0.0;
	for (Point2D loc1 : location.values()) {
		for (Point2D loc2 : location.values()) {



		}
	}
	return distance;
}
```

The nested structure means that for each loc1 in the outer loop we compute its distance to every loc2 in the inner loop. Both loc1 and loc2 are drawn from the locations in the HashMap.

If we find a pair of locations that are further apart than any we've found so far then we update distance to reflect that:

```
public double longestDistance(HashMap<String, Point2D> location) {
	double distance = 0.0;
	for (Point2D loc1 : location.values()) {
		for (Point2D loc2 : location.values()) {
```

```
                        if (loc1.distance(loc2) > distance) {
                            distance = loc1.distance(loc2);
                        }
                    }
                }
                return distance;
        }
```

After the loop completes distance will contain the largest distance between any pair of places in the HashSet.