Testing Report for Project 3

**PointTest**

1. toString- We plug in a point and test whether the output of the string function matches the expected output of "(4.1,4.1,4.1)". It does, so this one checks out.
2. Equals- We test whether the equals method works by testing three cases. One case is two different points entirely; one uses an object that is not a point, and one uses the same points. The first two should prove false and our assertFalse statements prove that. The last one should return true and by using assertEquals, we can see that they are indeed equal.
3. addVector-tests whether when we add the vectors together do we get the same value as the method. We test the method and use the assertEquals to compare it to the manually calculated one and they are both true.
4. Distance- this is the same as the addvector where we are finding a numerical distance between two points. Doing it manually and putting it in the method should come up with the same result and it checks out using our assertEquals.

**Point2DTest**

1. toString- the only method we need to modify, it needs to be able to show the point as expected. We use the assertEquals to test whether the p1.toString() will return the desired output of "(4.1,4.1)" and it does, so we are good.

**LIneTest**

1. toString- We initialize a line using a point and a vector and see if the method’s display of this line matches our display of the line. We see that the assertEquals confirms that the method’s output is "(4.1x+4.1,4.1y+4.1,4.1z+4.1)".
2. equals- We test whether the lines are equal to each other. We see that there are 4 cases. The same line, the not even a line, the line with different starting points but the same slope and the two different lines. The not even a line, line with different starting points but same slope, and the two different lines should give false. They do all do. However, the last test where they are indeed the same line returns false when it should return true. I have no clue why.
3. isOnLine- This method tests whether the point is on the line. Using one point that is not on the line and one point that is, we see that the assertEquals corresponds with when we do indeed have a point on the line and when we don’t.
4. isParallel- We test whether lines are parallel or not by initializing 3 lines and comparing them. The first two lines are indeed not parallel and this is confirmed using our assertEquals and our method matches its result. We can also see that our method matches the assertEquals value for when the lines are indeed parallel.
5. Intersection- We test whether the lines are intersecting. When they are parallel, they do not intersect and by using assertEquals, we can see the null parameter is returned by the method when the lines are not intersecting. Otherwise, we see the method return the point of intersection when the two lines do intersect in correspondence with the assertEquals.

**Line2DTest**

1. toString- We test the two conditions. One, when the line is vertical and two, when the line can be represented in slope intercept form. We see that the when the line is vertical, the method does indeed represent the string correctly. We also see that when the line is able to be represented in slope-intercept form correctly by the method when we compare it to the equation using the assertEquals.
2. Intersection-this tests whether the points intersect. Again, when the lines are parallel they do not intersect and this is verified by assertEquals returning the same value as the null that our method gives. When they do intersect, we again see that the point that it returns matches with the actual intersection point.

**VectorTest**

1. toString- We plug in a vector and test whether the output of the string function matches the expected output of "<4.1,4.1,4.1>". It does, so this one checks out too.
2. Equals- We test whether the equals method works by testing three cases. One case is two different vectors entirely; one uses an object that is not a vector, and one uses the same vectors. The first two should prove false and our assertFalse statements prove that. The last one should return true and by using assertEquals, we can see that they are indeed equal.
3. Magnitude- We test whether the calculated magnitude matches up with the method’s magnitude. It does and it works.
4. Unitvector-We test whether the manual unitvector and the method unitvector return the same value using assertEquals. It does and it works.
5. Sum- We test whether the manual sumvector and the sumvector method return the same value using assertEquals. They do.
6. scaleVector- we test whether vector with the multiplier is equal to the method’s way of calculating the scale vector. It appears to equal each other.
7. dotProduct-We test whether the calculated dotproduct is equal to the method’s calculation of the dotProduct. It does.
8. crossProduct-We test whether the calculated crossproduct vector is equal to the method’s way of calculating the crossproduct. It works and equals each other.
9. angle-We test whether the manual angle and the method angle return the same value using angle. The almighty assertEquals method does return the same values.
10. isOrthoganol-We test first whether the orthoganol method returns false when the vectors are not orthoganol and this does hold true. Then, we test whether the orthoganol method returns true when the vectors are orthoganol. They do. The assertEquals method shows it.
11. isParallel- We test first whether the parallel method returns false when the vectors are not parallel and this does hold true. Then, we test whether the parallel method returns true when the vectors are parallel. The AssertEquals method shows this.

**PlaneTest**

1. toString- We initialize a plane using a point and a vector and then compare it’s representation by the toString method with our manually calculated ways using assertEquals and it shows that they are indeed the same.
2. Equals- We test 4 things for this. We test the different planes entirely, the same plane, the same plane with different starting points, and the not a plane at all but a superman. The 1st and last should return false and this checks out using the assertEquals method. The middle two should return true and this again returns true using the assertEquals method.
3. Contains- this tests whether the point is on the plane. We test two scenarios where the point is on the plane and the point is not on the plane. By comparing our method’s calculations to what we know is true, we see that the method does indeed correctly return whether the point is on the plane or not.
4. isParallel- We test whether the planes are parallel by using testing two planes that are parallel and two planes that are not. We see that it does correctly return when the planes are parallel and when they are not using the assertEquals method.
5. isOrthoganol- We test whether the planes are Orthoganol by testing two planes that are orthoganol and two that are not. We see that when they are orthoganol, the assertEquals verifies that the methods displays and this and that when they are not, the assertEquals also displays this.