



# Discussion 5: Project 3 Exercises

Kyle Hackett



# Helpful Resources

<http://tutorials.jenkov.com/java-collections/comparator.html>

<https://www.geeksforgeeks.org/binary-search/>

<https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/>



**Comparable 6-Sided Die**  
**Comparable Geolocation**  
**Comparable 3D Point**



**Exercise 1.** (*Comparable Six-sided Die*) Implement a comparable data type called `Die` that represents a six-sided die and supports the following API:

Die	
<code>Die()</code>	constructs a die
<code>void roll()</code>	rolls this die
<code>int value()</code>	returns the face value of this die
<code>boolean equals(Die other)</code>	returns <code>true</code> if this die is the same as <code>other</code> , and <code>false</code> otherwise
<code>int compareTo(Die other)</code>	returns a comparison of this die with <code>other</code> , by their face values
<code>String toString()</code>	returns a string representation of this die

```
$ java Die 5 3 4  
Dice a, b, and c:
```

```
*      *
```

```
      *
```

```
*      *
```

```
*
```

```
      *
```

```
        *
```

```
*      *
```

```
*      *
```

```
a.equals(b)      = false
```

```
b.equals(c)      = false
```

```
a.compareTo(b)   = 2
```

```
b.compareTo(c)   = -1
```

```
*
```

```
*
```

```
*
```

```
      *
```

```
        *
```

```
*
```

```
*
```

```
          *
```

```
        *
```

```
        *
```

```
        *
```

```
        *
```



**Exercise 2.** (*Comparable Geo Location*) Implement an immutable data type called `Location` that represents a location on Earth and supports the following API:

Location	
<code>Location(String name, double lat, double lon)</code>	constructs a new location given its name, latitude, and longitude
<code>double distanceTo(Location other)</code>	returns the great-circle distance <sup>†</sup> between this location and <code>other</code>
<code>boolean equals(Object other)</code>	returns <code>true</code> if this location is the same as <code>other</code> , and <code>false</code> otherwise
<code>String toString()</code>	returns a string representation of this location
<code>int compareTo(Location other)</code>	returns a comparison of this location with <code>other</code> based on their respective distances to the origin, Parthenon (Greece) @ 37.971525, 23.726726

$$d = 6359.83 \arccos(\sin(x_1) \sin(x_2) + \cos(x_1) \cos(x_2) \cos(y_1 - y_2)).$$

```
>_ ~/workspace/project3
```

```
$ java Location 2 XYZ 27.1750 78.0419
```

```
Seven wonders, in the order of their distance to Parthenon (Greece):
```

```
  The Colosseum (Italy) (41.8902, 12.4923)
```

```
  Petra (Jordan) (30.3286, 35.4419)
```

```
  Taj Mahal (India) (27.175, 78.0419)
```


```
  Christ the Redeemer (Brazil) (22.9519, -43.2106)
```

```
  The Great Wall of China (China) (40.6769, 117.2319)
```

```
  Chichen Itza (Mexico) (20.6829, -88.5686)
```

```
  Machu Picchu (Peru) (-13.1633, -72.5456)
```

```
wonders[2] == XYZ (27.175, 78.0419)? true
```



**Exercise 3.** (*Comparable 3D Point*) Implement an immutable data type called `Point3D` that represents a point in 3D and supports the following API:

Point3D	
<code>Point3D(double x, double y, double z)</code>	constructs a point in 3D given its <code>x</code> , <code>y</code> , and <code>z</code> coordinates
<code>double distance(Point3D other)</code>	returns the Euclidean distance <sup>†</sup> between this point and <code>other</code>
<code>String toString()</code>	returns a string representation of this point
<code>int compareTo(Point3D other)</code>	returns a comparison of this point with <code>other</code> based on their respective distances to the origin $(0, 0, 0)$
<code>static Comparator&lt;Point3D&gt; xOrder()</code>	returns a comparator to compare two points by their $x$ -coordinate
<code>static Comparator&lt;Point3D&gt; yOrder()</code>	returns a comparator to compare two points by their $y$ -coordinate
<code>static Comparator&lt;Point3D&gt; zOrder()</code>	returns a comparator to compare two points by their $z$ -coordinate



† The Euclidean distance between the points  $(x_1, y_1, z_1)$  and  $(x_2, y_2, z_2)$  is given by  $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$ .

```
>_ ~/workspace/project3
```

```
$ java Point3D
How many points? 3
Enter 9 doubles, separated by whitespace: -3 1 6 0 5 8 -5 -7 -3
Here are the points in the order entered:
  (-3.0, 1.0, 6.0)
  (0.0, 5.0, 8.0)
  (-5.0, -7.0, -3.0)
Sorted by their natural ordering (compareTo)
  (-3.0, 1.0, 6.0)
  (-5.0, -7.0, -3.0)
  (0.0, 5.0, 8.0)
Sorted by their x coordinate (xOrder)
  (-5.0, -7.0, -3.0)
  (-3.0, 1.0, 6.0)
  (0.0, 5.0, 8.0)
Sorted by their y coordinate (yOrder)
  (-5.0, -7.0, -3.0)
  (-3.0, 1.0, 6.0)
  (0.0, 5.0, 8.0)
Sorted by their z coordinate (zOrder)
  (-5.0, -7.0, -3.0)
  (-3.0, 1.0, 6.0)
  (0.0, 5.0, 8.0)
```