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

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

>_ ~/workspace/exercise3

```
$ 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
```

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

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

† See Problem 1 of Exercise 1 for formula.

>_ ~/workspace/exercise3

```
$ 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
```

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

Point3D

`Point3D(double x, double y, double z)`	constructs a point in 3D given its `x`, `y`, and `z` coordinates
`double distance(Point3D other)`	returns the Euclidean distance† between this point and `other`
`String toString()`	returns a string representation of this point
`int compareTo(Point3D other)`	returns a comparison of this point with `other` based on their respective distances to the origin $(0, 0, 0)$
`static Comparator<Point3D> xOrder()`	returns a comparator to compare two points by their x-coordinate
`static Comparator<Point3D> yOrder()`	returns a comparator to compare two points by their x-coordinate
`static Comparator<Point3D> zOrder()`	returns a comparator to compare two points by their x-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/exercise3

```
$ 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)
```

Files to Submit

1. `Die.java`
2. `Location.java`
3. `Point3D.java`

Before you submit your files, make sure:

- You do not use concepts outside of what has been taught in class.
- Your code is adequately commented, follows good programming principles, and meets any specific requirements such as corner cases and running times.