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

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

```java
$ java Die 5 3 4
```

Dices a, b, and c:

```
* * *
* * *
* * *
* * *
* * *
```

```java
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:

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

† See Problem 1 of Exercise 1 for formula.

```java
$ 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) (29.0698, 75.7312)
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)
```

```java
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:

```java
$ java com/point3D
```

Wonders[2] == XYZ (27.175, 78.0419)? true
Exercise 3 (Comparable Data Types)

**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\(^1\) 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 y-coordinate

static Comparator<Point3D> zOrder() returns a comparator to compare two points by their z-coordinate

\(^1\) The Euclidean distance between the points \((x_1, y_1, z_1)\) and \((x_2, y_2, z_2)\) is given by \(\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}\).