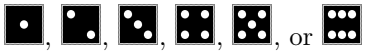
Objectives. Implement simple data types that:

1. Conform to a given API.
2. Are comparable.
3. Support alternate orderings.
4. Are iterable.

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

method	description
`Die()`	construct a die
`void roll()`	roll the die
`int value()`	face value of the die
`boolean equals(Die that)`	does the die have the same face value as *that*?
`int compareTo(Die that)`	the signed difference between the face values of this die and *that*
`String toString()`	a string representation of the current face value of the die, ie, 

```
$ java Die 5 3 3
*   *
  *
*   *
false
true
2
0
```

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

method	description
`Location(String loc, double lat, double lon)`	construct a new location given its name, latitude, and longitude values
`double distanceTo(Location that)`	the great-circle distance† between this location and *that*
`boolean equals(Location that)`	is this location the same as *that*?
`int compareTo(Location that)`	-1, 0, or 1 depending on whether the distance of this location to the origin is less than, equal to, or greater than the distance of *that* location to the origin, where the origin is the center of the universe, ie, UMass Boston (42.3134, -71.0384)
`String toString()`	a string representation of the location, in `"loc (lat, lon)"` format

† See Problem 4 of Homework 1 for formula

```
$ java Location 5 40.6769 117.2319
Chichen Itza (Mexico) (20.6829, -88.5686)
Christ the Redeemer (Brazil) (22.9519, -43.2106)
Machu Picchu (Peru) (-13.1633, -72.5456)
The Colosseum (Italy) (41.8902, 12.4923)
Petra (Jordan) (30.3286, 35.4419)
The Great Wall of China (China) (40.6769, 117.2319)
Taj Mahal (India) (27.175, 78.0419)
true
```

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

method/class	description
`Point3D(double x, double y, double z)`	construct a point in 3D given its coordinates
`double distance(Point3D that)`	the Euclidean distance† between this point and *that*
`int compareTo(Point3D that)`	-1, 0, or 1 depending on whether this point's Euclidean distance to the origin is less than, equal to, or greater than *that* point's Euclidean distance to the origin, where the origin is $(0, 0, 0)$
`static class XOrder`	a comparator for comparing points based on their x-coordinates
`static class YOrder`	a comparator for comparing points based on their y-coordinates
`static class ZOrder`	a comparator for comparing points based on their z-coordinates
`String toString()`	a string representation of the point, in `"(x, y, z)"` format

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

```
$ java Point3D
5
-3 1 6
0 5 8
-5 -7 -3
-2 4 7
-6 8 6
(-3.0, 1.0, 6.0)
(0.0, 5.0, 8.0)
(-5.0, -7.0, -3.0)
(-2.0, 4.0, 7.0)
(-6.0, 8.0, 6.0)
(-3.0, 1.0, 6.0)
(-2.0, 4.0, 7.0)
(-5.0, -7.0, -3.0)
(0.0, 5.0, 8.0)
(-6.0, 8.0, 6.0)
(-6.0, 8.0, 6.0)
(-5.0, -7.0, -3.0)
(-3.0, 1.0, 6.0)
(-2.0, 4.0, 7.0)
(0.0, 5.0, 8.0)
(-5.0, -7.0, -3.0)
(-3.0, 1.0, 6.0)
(-2.0, 4.0, 7.0)
(0.0, 5.0, 8.0)
(-6.0, 8.0, 6.0)
(-5.0, -7.0, -3.0)
(-3.0, 1.0, 6.0)
(-6.0, 8.0, 6.0)
(-2.0, 4.0, 7.0)
(0.0, 5.0, 8.0)
```

Problem 4. (*Iterable Binary Strings*) Implement an immutable, iterable data type `BinaryStrings` in `BinaryStrings.java` to systematically iterate over length-n binary strings. The data type must support the following API:

method	description
`BinaryStrings(int n)`	construct an iterable `BinaryStrings` object given the length of binary strings needed
`Iterator<String> iterator()`	an iterator for binary strings of a given length

```
$ java BinaryStrings 4
0000
0001
0010
0011
```

```
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
```

Problem 5. (*Iterable Primes*) Implement an immutable, iterable data type `Primes` in `Primes.java` to systematically iterate over the first n primes. The data type must support the following API:

method	description
`Primes(int n)`	construct an iterable `Primes` object given the number of primes needed
`Iterator<Integer> iterator()`	an iterator for the given number of primes

```
$ java Primes 10
2
3
5
7
11
13
17
19
23
29
```

Files to Submit

1. `Die.java`
2. `Location.java`
3. `Point3D.java`
4. `BinaryStrings.java`
5. `Primes.java`

Before you submit:

- Make sure your programs meet the input and output specifications by running the following command on the terminal:

  ```
  $ python3 run_tests.py -v [<problems>]
  ```

  where the optional argument `<problems>` lists the problems (`Problem1`, `Problem2`, etc.) you want to test, separated by spaces; all the problems are tested if no argument is given.

- Make sure your programs meet the style requirements by running the following command on the terminal:

  ```
  $ check_style <program>
  ```

  where `<program>` is the `.java` file whose style you want to check.