Project 3 (Autocomplete Me) Checklist

Prologue

Project goal: write a program to implement *autocomplete* for a given set of n strings and nonnegative weights, ie, given a prefix, find all strings in the set that start with the prefix, in descending order of weight

Files:

- `project3.pdf` ↗ (project description)
- `project3.zip` ↗ (starter files for the exercises/problems, `report.txt` file for the project report, and test data files)

Exercises

Exercise 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()`	constructs a die
`void roll()`	rolls the die
`int value()`	returns the face value of the die
`boolean equals(Die that)`	returns `true` if *this* die has the same face value as *that*, and `false` otherwise
`int compareTo(Die that)`	returns the signed difference between the face values of *this* die and *that*
`String toString()`	returns a string representation of the face value, ie, ⚀, ⚁, ⚂, ⚃, ⚄, or ⚅

```
>_ ~/workspace/project3

$ java edu.umb.cs210.p3.Die 5 3 3
*   *
  *
*   *
false
true
true
false
```

Exercises

Die.java

```java
package edu.umb.cs210.p3;

import stdlib.StdOut;
import stdlib.StdRandom;

// A data type representing a six-sided die.
public class Die implements Comparable<Die> {
    private int value; // face value

    // Construct a die.
    public Die() {
        ...
    }

    // Roll the die.
    public void roll() {
        ...
    }

    // Face value of the die.
    public int value() {
        ...
    }

    // Does the die have the same face value as that?
    public boolean equals(Object that) {
        if (this == that) return true;
        if (that == null) return false;
        if (this.getClass() != that.getClass()) return false;
        Die thatDie = (Die) that;
        ...
    }

    // A negative integer, zero, or positive integer depending on
    // whether this die’s value is less than, equal to, or greater
    // than the that die’s value.
```

Exercises

Die.java

```java
    public int compareTo(Die that) {
        ...
    }

    // A string representation of the die giving the current
    // face value.
    public String toString() {
        ...
    }

    // Test client. [DO NOT EDIT]
    public static void main(String[] args) {
        int x = Integer.parseInt(args[0]);
        int y = Integer.parseInt(args[1]);
        int z = Integer.parseInt(args[2]);
        Die a = new Die();
        a.roll();
        while (a.value() != x) {
            a.roll();
        }
        Die b = new Die();
        b.roll();
        while (b.value() != y) {
            b.roll();
        }
        Die c = new Die();
        c.roll();
        while (c.value() != z) {
            c.roll();
        }
        StdOut.println(a);
        StdOut.println(a.equals(b));
        StdOut.println(b.equals(c));
        StdOut.println(a.compareTo(b) > 0);
        StdOut.println(b.compareTo(c) > 0);
    }
}
```

Exercises

Exercise 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)`	constructs a new location given its name, latitude, and longitude
`double distanceTo(Location that)`	returns the great-circle distance[†] between *this* location and *that*
`boolean equals(Location that)`	returns `true` if *this* location the same as *that*, and `false` otherwise
`int compareTo(Location that)`	returns -1, 0, or 1 depending on whether the distance of *this* location to the origin (Parthenon, Athens, Greece @ 37.971525, 23.726726) is less than, equal to, or greater than the distance of *that* location to the origin
`String toString()`	returns the string representation of the location, in `"loc (lat, lon)"` format

[†] See Exercise 1 of Project 1 for formula

Exercises

```
>_ ~/workspace/project3

$ java edu.umb.cs210.p3.Location 4 40.6769 117.2319
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)
true
```

Exercises

Location.java

```java
package edu.umb.cs210.p3;

import stdlib.StdOut;

import java.util.Arrays;

// An immutable type representing a location on Earth.
public class Location implements Comparable<Location> {
    private final String loc; // location name
    private final double lat; // latitude
    private final double lon; // longitude

    // Construct a new location given its name, latitude, and
    // longitude.
    public Location(String loc, double lat, double lon) {
        ...
    }

    // The great-circle distance between this location and that.
    public double distanceTo(Location that) {
        ...
    }

    // Is this location the same as that?
    public boolean equals(Object that) {
        if (...) return true;
        if (...) return false;
        if (...) return false;
        Location thatLocation = (Location) that;
        ...
    }

    // -1, 0, or 1 depending on whether the distance of this
    // location to the origin (Parthenon, Athens, Greece @
    // 37.971525, 23.726726) is less than, equal to, or greater
    // than the distance of that location to the origin.
```

Exercises

Location.java

```java
    public int compareTo(Location that) {
        ...
    }

    // A string representation of the location, in
    // "loc (lat, lon)" format.
    public String toString() {
        ...
    }

    // Test client. [DO NOT EDIT]
    public static void main(String[] args) {
        int rank = Integer.parseInt(args[0]);
        double lat = Double.parseDouble(args[1]);
        double lon = Double.parseDouble(args[2]);
        Location[] wonders = new Location[7];
        wonders[0] = new Location("The Great Wall of China (China)",
                                  40.6769, 117.2319);
        wonders[1] = new Location("Petra (Jordan)", 30.3286, 35.4419);
        wonders[2] = new Location("The Colosseum (Italy)", 41.8902, 12.4923);
        wonders[3] = new Location("Chichen Itza (Mexico)", 20.6829, -88.5686);
        wonders[4] = new Location("Machu Picchu (Peru)", -13.1633, -72.5456);
        wonders[5] = new Location("Taj Mahal (India)", 27.1750, 78.0419);
        wonders[6] = new Location("Christ the Redeemer (Brazil)",
                                  22.9519, -43.2106);
        Arrays.sort(wonders);
        for (Location wonder : wonders) {
            StdOut.println(wonder);
        }
        Location loc = new Location("", lat, lon);
        StdOut.println(wonders[rank].equals(loc));
    }
}
```

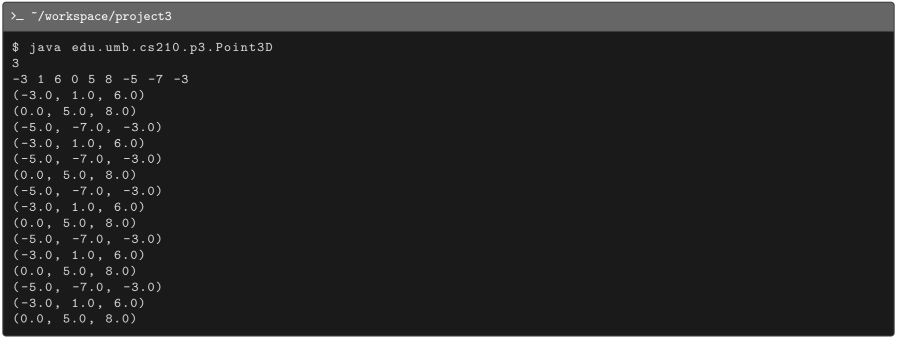
Exercises

Exercise 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	Description
`Point3D(double x, double y, double z)`	constructs a point in 3D given its x, y, and z coordinates
`double distance(Point3D that)`	returns the Euclidean distance[†] between *this* point and *that*
`int compareTo(Point3D that)`	returns -1, 0, or 1 depending on whether *this* point's Euclidean distance to the origin $(0, 0, 0)$ is less than, equal to, or greater than *that* point's Euclidean distance to the origin
`String toString()`	returns a string representation of the point, in `"(x, y, z)"` format
`static Comparator<Point3D> xOrder()`	returns an x-coordinate comparator
`static Comparator<Point3D> yOrder()`	returns a y-coordinate comparator
`static Comparator<Point3D> zOrder()`	returns a z-coordinate comparator

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

Exercises



~/workspace/project3

```
$ java edu.umb.cs210.p3.Point3D
3
-3 1 6 0 5 8 -5 -7 -3
(-3.0, 1.0, 6.0)
(0.0, 5.0, 8.0)
(-5.0, -7.0, -3.0)
(-3.0, 1.0, 6.0)
(-5.0, -7.0, -3.0)
(0.0, 5.0, 8.0)
(-5.0, -7.0, -3.0)
(-3.0, 1.0, 6.0)
(0.0, 5.0, 8.0)
(-5.0, -7.0, -3.0)
(-3.0, 1.0, 6.0)
(0.0, 5.0, 8.0)
(-5.0, -7.0, -3.0)
(-3.0, 1.0, 6.0)
(0.0, 5.0, 8.0)
```

Exercises

Point3D.java

```java
package edu.umb.cs210.p3;

import stdlib.StdIn;
import stdlib.StdOut;

import java.util.Arrays;
import java.util.Comparator;

// An immutable data type representing a 3D point.
public class Point3D implements Comparable<Point3D> {
    private final double x; // x coordinate
    private final double y; // y coordinate
    private final double z; // z coordinate

    // Construct a point in 3D given its coordinates.
    public Point3D(double x, double y, double z) {
        ...
    }

    // The Euclidean distance between this point and that.
    public double distance(Point3D that) {
        ...
    }

    // -1, 0, or 1 depending on this point's Euclidean
    // distance to the origin (0, 0, 0) is less than,
    // equal to, or greater than that point's Euclidean
    // distance to the origin.
    public int compareTo(Point3D that) {
        ...
    }

    // An x-coordinate comparator.
    public static Comparator<Point3D> xOrder() {
        ...
    }
```

Exercises

Point3D.java

```java
    // Helper x-coordinate comparator.
    private static class XOrder implements Comparator<Point3D> {
        // -1, 0, or 1 depending on whether p1's x-coordinate
        // is less than, equal to, or greater than p2's
        // x-coordinate.
        public int compare(Point3D p1, Point3D p2) {
            ...
        }
    }

    // A y-coordinate comparator.
    public static Comparator<Point3D> yOrder() {
        ...
    }

    // Helper y-coordinate comparator.
    private static class YOrder implements Comparator<Point3D> {
        // -1, 0, or 1 depending on whether p1's y-coordinate
        // is less than, equal to, or greater than p2's
        // y-coordinate.
        public int compare(Point3D p1, Point3D p2) {
            ...
        }
    }

    // A z-coordinate comparator.
    public static Comparator<Point3D> zOrder() {
        ...
    }

    // Helper z-coordinate comparator.
    private static class ZOrder implements Comparator<Point3D> {
        // -1, 0, or 1 depending on whether p1's z-coordinate
        // is less than, equal to, or greater than p2's
        // z-coordinate.
```

Exercises

Point3D.java

```java
        public int compare(Point3D p1, Point3D p2) {
            ...
        }
    }

    // A string representation of the point, as "(x, y, z)".
    public String toString() {
        ...
    }

    // Test client. [DO NOT EDIT]
    public static void main(String[] args) {
        StdOut.print("How many points?: ");
        int n = StdIn.readInt();
        Point3D[] points = new Point3D[n];
        StdOut.printf("Enter %d doubles, separated by whitespace: ", n*3);
        for (int i = 0; i < n; i++) {
            double x = StdIn.readDouble();
            double y = StdIn.readDouble();
            double z = StdIn.readDouble();
            points[i] = new Point3D(x, y, z);
        }
        StdOut.println("\nHere are the points in the order entered.");
        for (Point3D point : points) {
            StdOut.println(point);
        }
        Arrays.sort(points);
        StdOut.println("Sorted by their natural ordering (compareTo).");
        for (Point3D point : points) {
            StdOut.println(point);
        }
        Arrays.sort(points, Point3D.xOrder());
        StdOut.println("Sorted by their x value (xOrder).");
        for (Point3D point : points) {
            StdOut.println(point);
        }
```

Exercises

Point3D.java

```
        Arrays.sort(points, Point3D.yOrder());
        StdOut.println("Sorted by their y value (yOrder).");
        for (Point3D point : points) {
            StdOut.println(point);
        }
        Arrays.sort(points, Point3D.zOrder());
        StdOut.println("Sorted by their z value (zOrder).");
        for (Point3D point : points) {
            StdOut.println(point);
        }
    }
}
```

Problems

The guidelines for the project problems that follow will be of help only if you have read the description of the project and have a general understanding of the problems involved. It is assumed that you have done the reading.

Problems

Problem 1. (*Autocomplete Term*) Implement an immutable comparable data type `Term` that represents an autocomplete term and has the following API:

Method	Description
`Term(String query)`	initializes a term with the given query string and zero weight
`Term(String query, long weight)`	initializes a term with the given query string and weight
`int compareTo(Term that)`	compares the terms in lexicographic order by query
`static Comparator<Term> byReverseWeightOrder()`	returns a comparator for comparing terms in descending order by weight
`static Comparator<Term> byPrefixOrder(int r)`	returns a comparator for comparing terms in lexicographic order but using only the first r characters of each query
`String toString()`	returns a string representation of the term

Hints

⇝ Instance variables

- ⇝ Query string, `String query`
- ⇝ Query weight, `long weight`

Problems

- `Term(String query)` and `Term(String query, long weight)`
  - Initialize instance variables to appropriate values
- `int compareTo(Term that)`
  - Return a negative, zero, or positive integer based on whether `this.query` is smaller, equal to, or larger than `that.query`
- `static Comparator<Term> byReverseWeightOrder()`
  - Return an object of type `ReverseWeightOrder`
- `ReverseWeightOrder` :: `int compare(Term v, Term w)`
  - Return a -1, 0, or +1 based on whether `v.weight` is smaller, equal to, or larger than `w.weight`

Problems

- `static Comparator<Term> byPrefixOrder(int r)`
  - Return an object of type `PrefixOrder`
- `PrefixOrder` :: Instance variable
  - Prefix length, `int r`
- `PrefixOrder` :: `PrefixOrder(int r)`
  - Initialize instance variable appropriately
- `PrefixOrder` :: `int compare(Term v, Term w)`
  - Return a negative, zero, or positive integer based on whether `a` is smaller, equal to, or larger than `b`, where `a` is a substring of `v` of length `min(r, v.query.length())` and `b` is a substring of `w` of length `min(r, w.query.length())`
- `String toString()`
  - Return a string containing the weight and query separated by a tab

Problems

Problem 2. (*Binary Search Deluxe*) Implement a library of static methods `BinarySearchDeluxe` with the following API:

Method	Description
`static int firstIndexOf(Key[] a, Key key, Comparator<Key> c)`	returns the index of the first key in $a[]$ that equals the search key, or -1 if no such key
`static int lastIndexOf(Key[] a, Key key, Comparator<Key> c)`	returns the index of the last key in $a[]$ that equals the search key, or -1 if no such key

Hints

- `static int firstIndexOf(Key[] a, Key key, Comparator<Key> c)`
  - Modify the standard binary search such that when `a[mid]` matches `key`, instead of returning `mid`, remember it in, say `index` (initialized to -1), and adjust `hi` appropriately
  - Return `index`
- `static int lastIndexOf(Key[] a, Key key, Comparator<Key> c)` can be implemented similarly

Problems

Problem 3. (*Autocomplete*) Create an immutable data type `Autocomplete` with the following API:

Method	Description
`Autocomplete(Term[] terms)`	initializes the data structure from the given array of terms
`Term[] allMatches(String prefix)`	returns all terms that start with the given prefix, in descending order of weight
`int numberOfMatches(String prefix)`	returns the number of terms that start with the given prefix

Hints

- Instance variable
  - Array of terms, `Term[] terms`
- `Autocomplete(Term[] terms)`
  - Initialize `this.terms` as a defensive copy (ie, a fresh copy and not an alias) of `terms`
  - Sort `terms` in lexicographic order

Problems

- `Term[] allMatches(String prefix)`
  - Use `BinarySearchDeluxe` and `Term.byPrefixOrder()` to obtain the first index `i` of occurrence of `prefix`
  - Find the number `n` of terms that match `prefix`
  - Construct an array `matches` containing `n` elements from `terms`, starting at index `i`
  - Sort `matches` in reverse order of weight and return the sorted array
- `int numberOfMatches(String prefix)`
  - Use `BinarySearchDeluxe` and `Term.byPrefixOrder()` to obtain the first index and last index of occurrence of `prefix`
  - Compute and return the number of terms that match `prefix`

Problems

The `data` directory contains sample input files for testing; for example

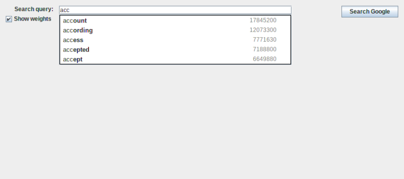
```
>_ ~/workspace/project3

$ more data/wiktionary.txt
10000
   5627187200  the
   3395006400  of
          ...  ...
       392402  wench
       392323  calves
```

The visualization client `AutocompleteGUI` takes the name of a file and an integer k as command-line arguments, provides a GUI for the user to enter queries, and presents the top k matching terms in real time

```
>_ ~/workspace/project3

$ java edu.umb.cs210.p3.AutocompleteGUI data/wiktionary.txt 5
```



Search query: acc

☑ Show weights

```
account     17845200
according   12073300
access       7771630
accepted     7188800
accept       6649880
```

Epilogue

Use the template file `report.txt` to write your report for the project

Your report must include:

- Time (in hours) spent on the project
- Difficulty level (1: very easy; 5: very difficult) of the project
- A short description of how you approached each problem, issues you encountered, and how you resolved those issues
- Acknowledgement of any help you received
- Other comments (what you learned from the project, whether or not you enjoyed working on it, etc.)

Epilogue

Before you submit your files:

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

  ```
  >_ ~/workspace/project3

  $ check_style <program>
  ```

  where `<program>` is the fully-qualified name of the program

- Make sure your code is adequately commented, is not sloppy, and meets any project-specific requirements, such as corner cases and running time
- Make sure your report uses the given template, isn't too verbose, doesn't contain lines that exceed 80 characters, and doesn't contain spelling mistakes

Epilogue

Files to submit:

1. `Die.java`
2. `Location.java`
3. `Point3D.java`
4. `Term.java`
5. `BinarySearchDeluxe.java`
6. `Autocomplete.java`
7. `report.txt`