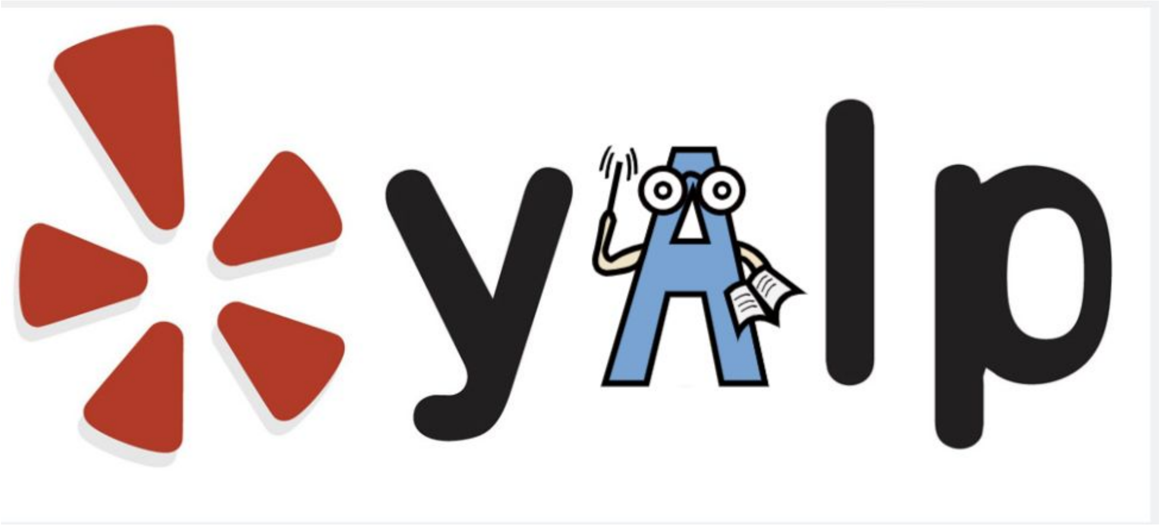


Your One-Stop Navigator for Restaurants, Shops, and Many More!

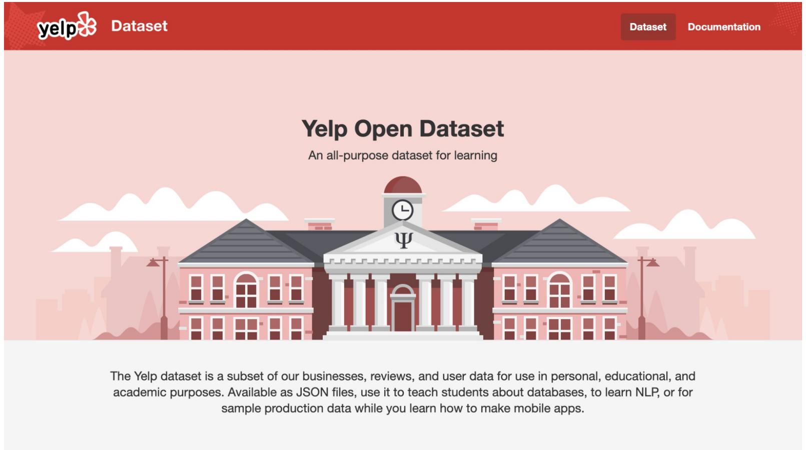
Overview

- A “yellow pages” like searching tool for finding businesses near a specific location
- Users enter their city, latitude and longitude, maximum distance, lowest / highest rating, and desired categories of businesses
- Users get back a list of businesses satisfying their criteria, sorted in the way they want (by name, by distance, or by rating)
- Our program backend helps users find businesses efficiently using **QuadTree, Binary Tree, Hashmap, ArrayList**, and other data structures

Data Source

We use the **Yelp Open Dataset**

- Covers over **1.2 million businesses** in over **600 North American cities**
- Contains info about businesses' names, ratings (stars), types, cities, latitudes/longitudes
- Available in .JSON format (which we will parse in our program)



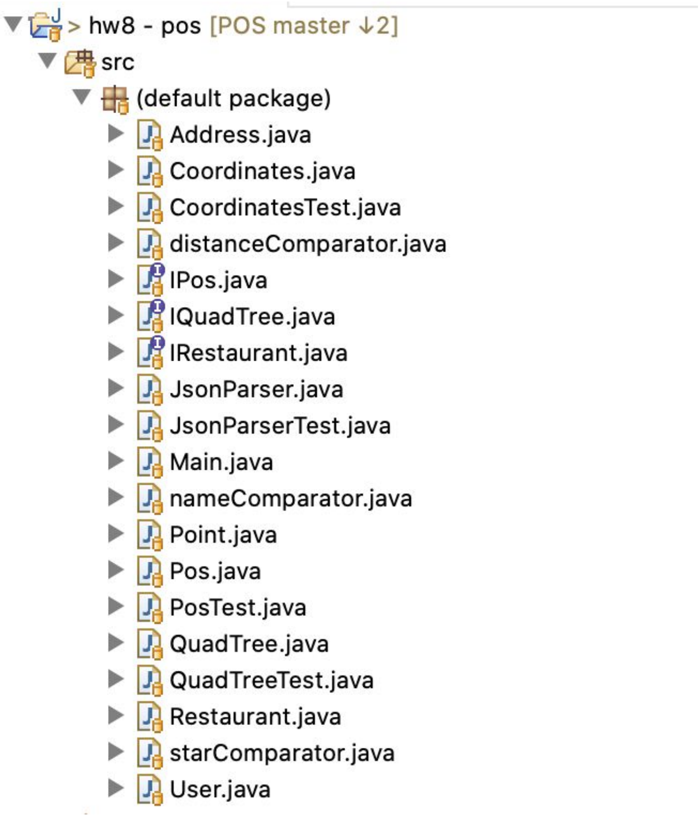
Yelp Open Dataset

An all-purpose dataset for learning

The Yelp dataset is a subset of our businesses, reviews, and user data for use in personal, educational, and academic purposes. Available as JSON files, use it to teach students about databases, to learn NLP, or for sample production data while you learn how to make mobile apps.

https://www.yelp.com/dataset

High-level Design & Class Diagram



hw8 - pos [POS master ↓2]

src

(default package)

Address.java

Coordinates.java

CoordinatesTest.java

distanceComparator.java

IPos.java

IQuadTree.java

IRestaurant.java

JsonParser.java

JsonParserTest.java

Main.java

nameComparator.java

Point.java

Pos.java

PosTest.java

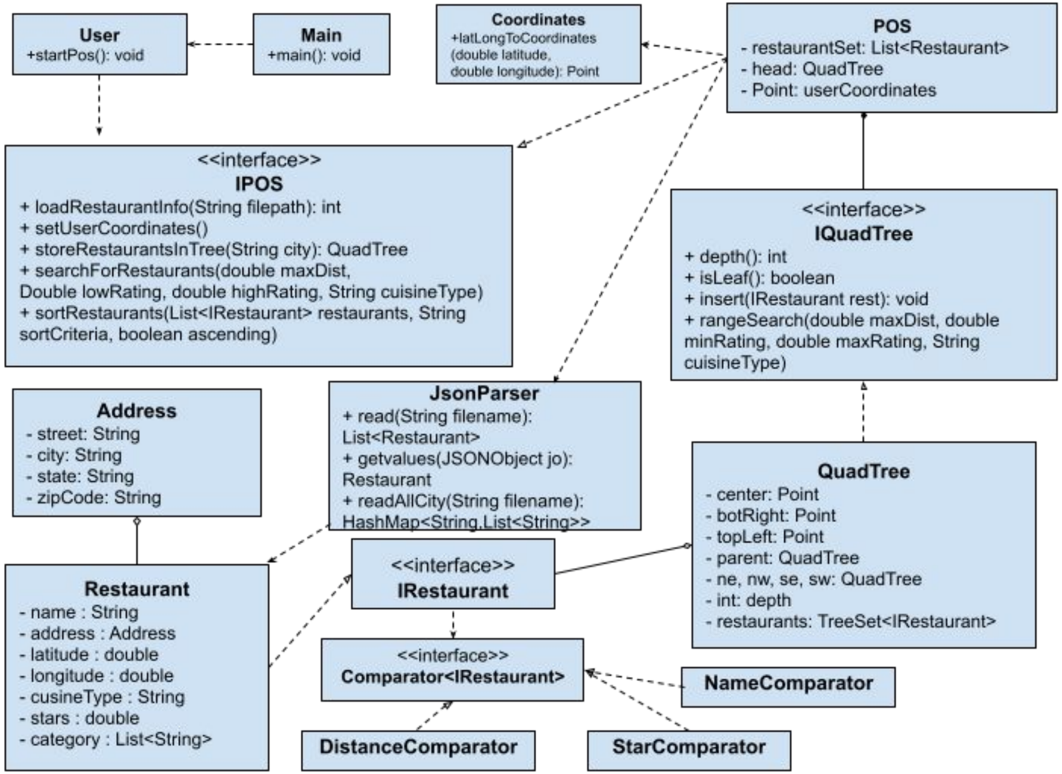
QuadTree.java

QuadTreeTest.java

Restaurant.java

starComparator.java

User.java



User
+startPos(): void

Main
+main(): void

Coordinates
+latLongToCoordinates
(double latitude,
double longitude): Point

POS
- restaurantSet: List<Restaurant>
- head: QuadTree
- Point: userCoordinates

<<interface>>
IPOS
+ loadRestaurantInfo(String filepath): int
+ setUserCoordinates()
+ storeRestaurantsInTree(String city): QuadTree
+ searchForRestaurants(double maxDist,
Double lowRating, double highRating, String cuisineType)
+ sortRestaurants(List<IRestaurant> restaurants, String
sortCriteria, boolean ascending)

<<interface>>
IQuadTree
+ depth(): int
+ isLeaf(): boolean
+ insert(IRestaurant rest): void
+ rangeSearch(double maxDist, double
minRating, double maxRating, String
cuisineType)

Address
- street: String
- city: String
- state: String
- zipCode: String

JsonParser
+ read(String filename):
List<Restaurant>
+ getvalues(JSONObject jo):
Restaurant
+ readAllCity(String filename):
HashMap<String,List<String>>

QuadTree
- center: Point
- botRight: Point
- topLeft: Point
- parent: QuadTree
- ne, nw, se, sw: QuadTree
- int: depth
- restaurants: TreeSet<IRestaurant>

Restaurant
- name : String
- address : Address
- latitude : double
- longitude : double
- cusineType : String
- stars : double
- category : List<String>

<<interface>>
IRestaurant

<<interface>>
Comparator<IRestaurant>

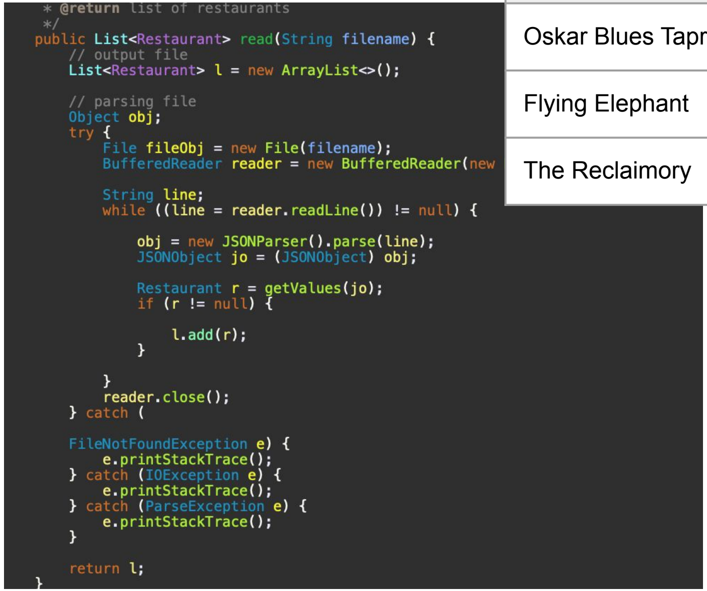
NameComparator

DistanceComparator

StarComparator

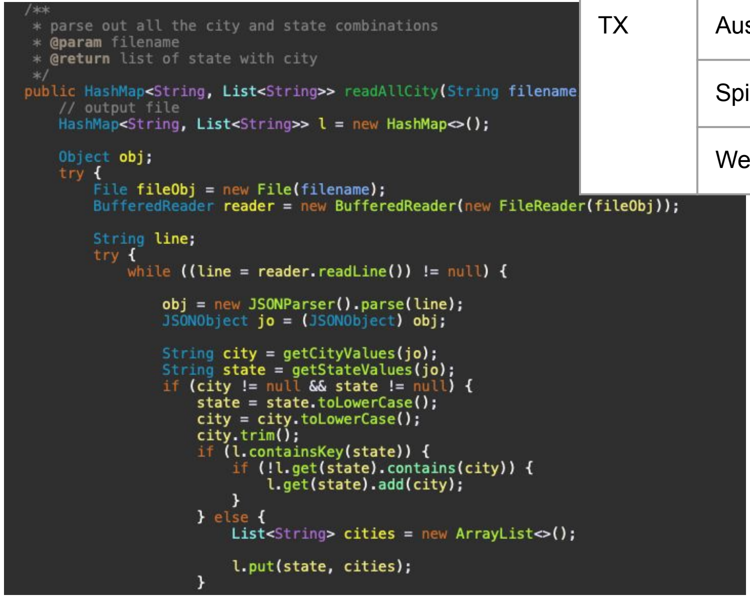
Data Structure Part I: Parser

Creates a **List** of Restaurant objects and a **HashMap** of state mapped to a list of cities



```
 * @return list of restaurants
 */
public List<Restaurant> read(String filename) {
    // output file
    List<Restaurant> l = new ArrayList<>();

    // parsing file
    Object obj;
    try {
        File fileObj = new File(filename);
        BufferedReader reader = new BufferedReader(new

        String line;
        while ((line = reader.readLine()) != null) {

            obj = new JSONParser().parse(line);
            JSONObject jo = (JSONObject) obj;

            Restaurant r = getValues(jo);
            if (r != null) {

                l.add(r);
            }

        }
        reader.close();
    } catch (

    FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (ParseException e) {
        e.printStackTrace();
    }

    return l;
}
```

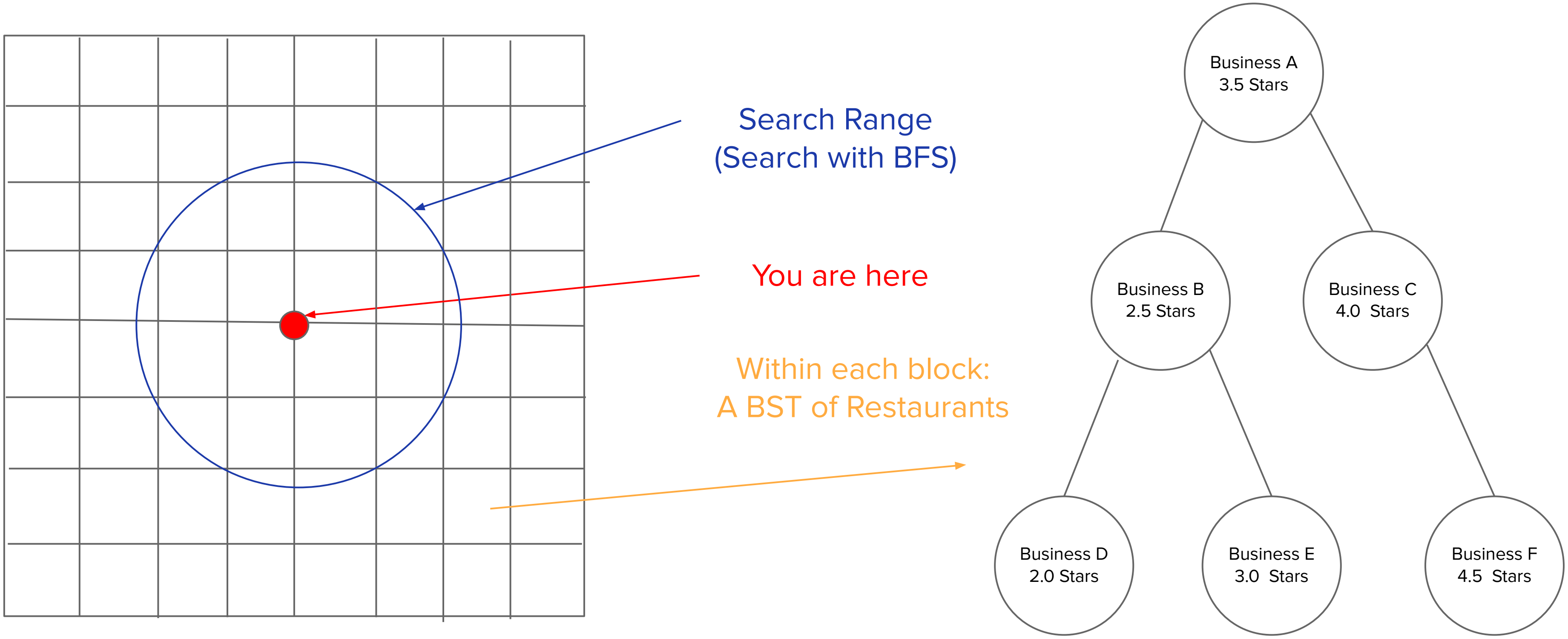
Spots
Oskar Blues Taproom
Flying Elephant
The Reclaimory



```
/**
 * parse out all the city and state combinations
 * @param filename
 * @return list of state with city
 */
public HashMap<String, List<String>> readAllCity(String filename
    // output file
    HashMap<String, List<String>> l = new HashMap<>();

    Object obj;
    try {
        File fileObj = new File(filename);
        BufferedReader reader = new BufferedReader(new FileReader(fileObj));

        String line;
        try {
            while ((line = reader.readLine()) != null) {

                obj = new JSONParser().parse(line);
                JSONObject jo = (JSONObject) obj;

                String city = getCityValues(jo);
                String state = getStateValues(jo);
                if (city != null && state != null) {
                    state = state.toLowerCase();
                    city = city.toLowerCase();
                    city.trim();
                    if (l.containsKey(state)) {
                        if (!l.get(state).contains(city)) {
                            l.get(state).add(city);
                        }
                    } else {
                        List<String> cities = new ArrayList<>();

                        l.put(state, cities);
                    }
```

State	City
TX	Austin
	Spicewood
	West Lake

Data Structure Part II: QuadTree

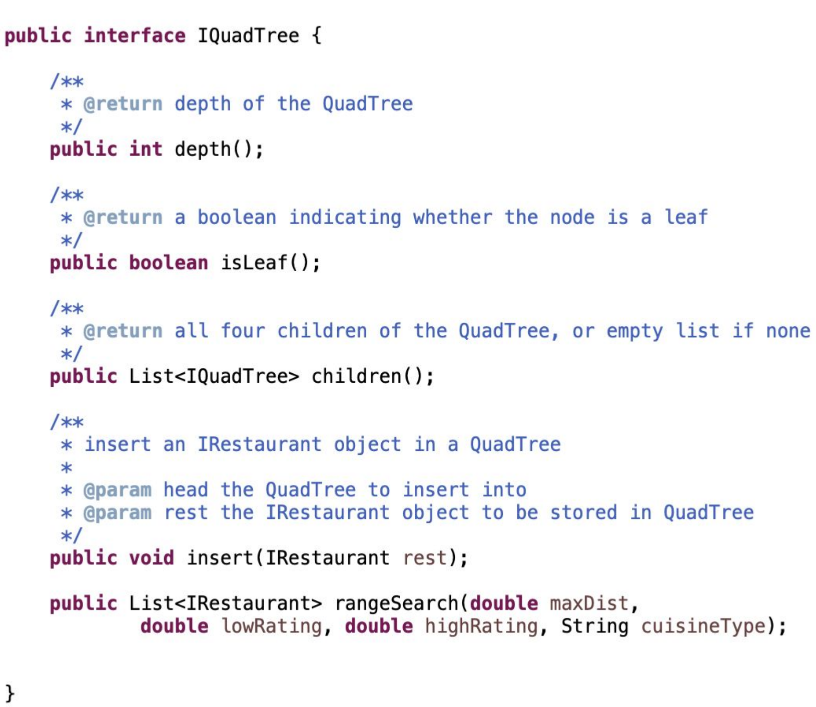


Search Range
(Search with BFS)

You are here

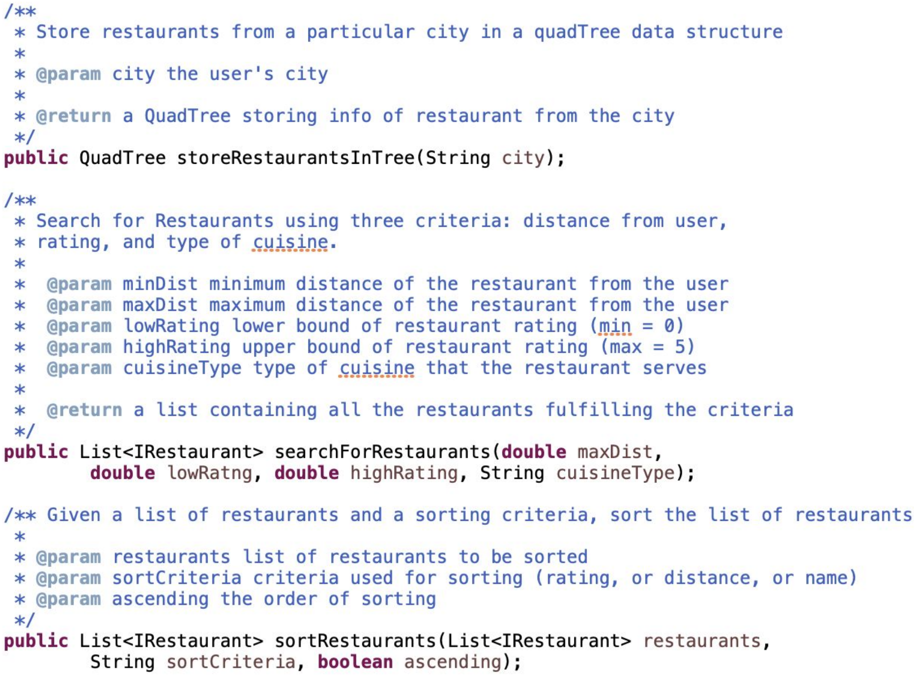
Within each block:
A BST of Restaurants

QuadTree Interface



```java
public interface IQuadTree {

    /**
     * @return depth of the QuadTree
     */
    public int depth();

    /**
     * @return a boolean indicating whether the node is a leaf
     */
    public boolean isLeaf();

    /**
     * @return all four children of the QuadTree, or empty list if none
     */
    public List<IQuadTree> children();

    /**
     * insert an IRestaurant object in a QuadTree
     *
     * @param head the QuadTree to insert into
     * @param rest the IRestaurant object to be stored in QuadTree
     */
    public void insert(IRestaurant rest);

    public List<IRestaurant> rangeSearch(double maxDist,
            double lowRating, double highRating, String cuisineType);

}
```

IQuadTree.java



```java
/**
 * Store restaurants from a particular city in a quadTree data structure
 *
 * @param city the user's city
 *
 * @return a QuadTree storing info of restaurant from the city
 */
public QuadTree storeRestaurantsInTree(String city);

/**
 * Search for Restaurants using three criteria: distance from user,
 * rating, and type of cuisine.
 *
 *  @param minDist minimum distance of the restaurant from the user
 *  @param maxDist maximum distance of the restaurant from the user
 *  @param lowRating lower bound of restaurant rating (min = 0)
 *  @param highRating upper bound of restaurant rating (max = 5)
 *  @param cuisineType type of cuisine that the restaurant serves
 *
 *  @return a list containing all the restaurants fulfilling the criteria
 */
public List<IRestaurant> searchForRestaurants(double maxDist,
        double lowRatng, double highRating, String cuisineType);

/** Given a list of restaurants and a sorting criteria, sort the list of restaurants
 *
 * @param restaurants list of restaurants to be sorted
 * @param sortCriteria criteria used for sorting (rating, or distance, or name)
 * @param ascending the order of sorting
 */
public List<IRestaurant> sortRestaurants(List<IRestaurant> restaurants,
        String sortCriteria, boolean ascending);
```

IPos.java

Program Demo

Let's say…

- You are a devout Christian and you would like to find a church near the Texas Capitol in Austin, TX (30.274716965679016, -97.7403750974195)
- You are a tourist staying at the St. Julien Hotel in Boulder, CO (40.016007699518845, -105.28282644458191) You want to find a decent bar for a good night of fun
- You arrived at the Haymarket Station in Boston, MA (42.36287579459132, -71.05826687116254) and you realized that you want some good bread at a local bakery