3.1 Designing Methods: Simple Classes

In this lab we will focus on designing methods for classes and class hierarchies of increasing complexity. Make sure you understand each part before moving on to the next one. If you are having difficulties at any time, do ask questions and make sure you understand how things work.

In the last lab you have defined class `Restaurant` that corresponds to the following class diagram into *Beginner ProfessorJ*:

```
+--------------+
| Restaurant   |
+--------------+
| String name  |
| String kind  |
| int avgPrice |
+--------------+
```

Remember the examples we have used:

- Chinese restaurant Blue Moon with average price per dinner $15
- Japanese restaurant Kaimo with average price per dinner $20
- Mexican restaurant Cordita with average price per dinner $12

1. Design the following methods for the `Restaurant` class.
    - Method `isKind` that determines whether this restaurant is the kind we are looking for (e.g. Chinese, French, etc.)
    - Method `cheaperThan` that determines whether this restaurant is cheaper than the given one, measured by the average price per dinner
    - Method `canEat` that determines whether a party of some given number of people will be able to afford a dinner in this restaurant, given the total amount they are willing to spend.

      (E.g. 3 people cannot eat at Blue Moon if they have only $40 to spend.)

3.2 Designing Methods: Classes with Containment

Look at the data definition for the following class diagram that you have defined in the previous lab:

```
+--------------+
| Restaurant2  |
+--------------+
| String name  |
| String kind  |
| int avgPrice |
| CartPt loc   |--+
+--------------+  |
                  v
              +--------+
              | CartPt |
              +--------+
              | int x  |
              | int y  |
              +--------+
```

1. Design the method `countBlocks` that computes the number of blocks we have to walk from this restaurant to the given one. Assume that the location represents the intersection of the numbered streets and numbered avenues on a city map similar to the midtown Manhattan.

Note: You may need to use the `Math.abs(int)` method that produces the absolute value of the given integer.

3.3 Designing Methods: Unions of Classes

In the previous lab you have designed the class hierarchy that represents the following kinds of pets:

- **cats** where we record whether it is a short-hair cat of a long-hair cat
- **dogs** where we record the breed (e.g. Husky, Labrador, etc., or Mutt — describing an unknown breed)
- **gerbils** where we need to know whether it is a male of female

still keeping track of the name of the animal and of its owner.

1. Design the method `isOwner` that determines whether this animal's owner has the given name.
2. Design the method `sameName` that determines whether two pets have the same name.

3.4 Designing Methods: Self-Referential Class Hierarchies

We will work with a list of restaurants. The code in the file **restaurant-list.java** defines the class hierarchy represented by the following class diagram:

```
            +------+
            | ILoR |<--------------+
            +------+               |
              / \                  |
              ---                  |
               |                   |
    --------------------           |
    |                  |           |
+-------+    +------------------+  |
| MtLoR |    | ConsLoR          |  |
+-------+    +------------------+  |
+-------+  +-| Restaurant first |  |
           | | ILoR rest        |--+
           | +------------------+
           v
+--------------+
| Restaurant   |
+--------------+
| String name  |
| String kind  |
| int avgPrice |
+--------------+
```

1. Design the method `count` that counts the number of restaurants in a list of restaurants.
2. Design the method `averageDinner` that computes the average of the average prices for all restaurants in a list of restaurants. For the `rlist3` in the file **restaurant-list.java** the method should produce $14.
3. Design the method `cheapList` that produces a list of all restaurants that have the average dinner price below the given limit.
4. Design the method `sort` that produces a list of all restaurants sorted by their average dinner prices.