2.1 Understanding Data: Simple Classes

In this lab we will focus on understanding data definitions, the distinction between information and data, how information can be represented as data, how to interpret the information that some instance of data represents, and learn to encode the data definitions, as well as construct instances of data in a class based language (like Java).

Look at the following data definitions in the *Beginner Student HtDP* language:

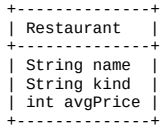
```
;; Sample data definitions -- simple classes of data

;; to represent a pet
;; A Pet is (make-pet String Num String)
(define-struct pet (name weight owner))

;; Examples of pets:
(define kitty (make-pet "Kitty" 15 "Pete"))
(define spot (make-pet "Spot" 20 "Jane"))
```

1. Draw the class diagram for this data definition.

2. Convert the data definition to the *Beginner ProfessorJ* language — including the examples of data.

   If you are comfortable with this material, you may omit the next two questions.

3. Convert the following class diagram into *Beginner ProfessorJ* language:

4. Convert the following information to data examples for your `Restaurant` class.

   - 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

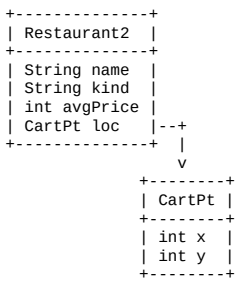
2.2 Understanding Data: Classes with Containment

Look at the following data definitions in the *Beginner Student HtDP* language:

```
;; to represent a pet
;; A Pet2 is (make-pet String Num Person)
(define-struct pet2 (name weight owner))

;; to represent a person - a pet's owner
;; A Person is (make-person String Num Boolean)
(define-struct person (name age male?))

;; Examples of person data:
(define pete (make-person "Pete" 15 true))
(define jane (make-person "Jane" 19 false))

;; Examples of pet2 data:
(define kitty2 (make-pet "Kitty" 15 pete))
(define spot2 (make-pet "Spot" 20 jane))
```

1. Draw the class diagram for this data definition.
2. Convert the data definition to the *Beginner ProfessorJ* language — including the examples of data.

   If you are comfortable with this material, you may omit the next two questions.
3. Convert the following class diagram into *Beginner ProfessorJ* language:



4. Make new examples for your `Restaurant2` class.

2.3 Understanding Data: Unions of Classes

The class of data that represent pets in the first part is not really sufficient. We have no idea what kind of pet the animal is. We would like to distinguish between 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

We need a data definition for pets that covers all these options. Of course, we still keep track of the name of the animal and of its owner.

1. Make examples (in English words) of at least one of each kind of pets.
2. Draw a class diagram for the class hierarchy that represents this information about pets.
3. Design data definitions for this data in the *Beginner ProfessorJ* language.
4. Convert your examples to data.

2.4 Representing Self-Referential Data

We want to trace your ancestry. Write down the name of your mother and your father, for each of them the name of their mother and father - as far as you can trace your ancestors. Write *unknown* when you no longer know the names. Organize your ancestor information into a tree-like structure - you are the root, your parents are the two branches, and each set of parents represents the two branches above their child. (You do not need to use actual names — feel free to make up the names of your ancestors — but go back to at least one great-grandparent.)

Design data definition that can be used to represent this information and then convert the information about your ancestry into data.

Follow the DESIGN RECIPE for data definitions:

- Is the information simple enough to be represented by a primitive data type?
- Are there several pieces of information that represent one entity? — if yes, design a class of data with a field for each piece of information.
- Is any of the fields itself a complex piece of data? — if yes, deal with designing classes for that field as a separate task.
- Are there several variants of data that should be known by a common name? — if yes, define an *interface* and have each variant implement this *interface*.
- Make sure you write down a comment explaining what each class of data (or each interface) represents.
- Make sure you make examples of every class you design.

2.5 Designing methods

Design the following methods for the classes that represent pets:

1. Method `weighsLessThan` that determines whether the pet weighs less than the given weight limit for flying in the passenger cabin of an airplane. (Each airline has their own limit.)
2. Method `sameOwner` that tells us whether the owner of the pet is the given person. Do this for first two variants of the `Pet` class.
3. Method `changeOwner` that produces a new `Pet` same as the original one, but with the owner changed to the given one. Do this for first two variants of the `Pet` class.