3 Methods for Complex Class Hierarchies; Libraries

Etudes

3.1 Etude

Work out the problems 14.5, 14.6, and 15.10 in the textbook. ∎

3.2 Etude

Here is an HtDP data definition:

```
;; A BST is one of
;; — empty
;; — Node

;; A Node is (make-node Number BST BST)
(define-struct node (value left right))

;; we expect the value to be a whole number
```

The BST (also known as the binary search tree) has the property that all values in the left sub-tree are smaller than the value of the node, and all values of the right subtree are larger than the value of the node. (We will not allow the same number to be a value of more than one node in the tree.)

1. Define the Java class hierarchy that represents a BST.
2. Design the method that counts the number of *Node*s in a *BST*.
3. Design the method that adds the values of all nodes in the BST.
4. Design the method that determines whether a given number is one of the node values in the BST.
5. Design the method that inserts a new node with the given value into the right place in the BST. If there already exists a node with the given value, the method produces the BST that looks the same as the original one.

∎

Main Assignment — Part 1

We continue working with the classes that help us set up the recording of TV shows. Our recording program keeps track of all of our requests. To be able to do that, we first need to design the data representation of a list of recordings.

3.3 Problem

Design the necessary data, then design the methods that allow us to answer the following questions:

1. What is the total recording time for all shows we want to record?
2. Is the list of recordings ordered by the starting times for the recordings?
3. Can we record all the shows on the list that is ordered by the starting times for the recordings? We cannot do so if some of the recording requests overlap in time.

   You may need several methods here! Think carefully about the design. Remember, one task — one method.
4. Produce a list of all recording requests for the given station.
5. **Challenge - optional**

   - We would like to have a list of all stations whose shows we plan to record. Produce such a list.
   - The TV recording system has a list of all stations we can record. Are there any recording requests for shows from stations that are not on this list?

   For a partial credit you may design the wish list of the methods you will need (purpose statements and the headers), without implementing the method bodies.

∎

Main Assignment — Part 2

We continue designing classes that help us draw the city map and its attractions. For this problem you do not need any of the classes from the previous assignment, other than the class *Place* that represents a location on the map.

3.4 Problem

Develop the data definition to represent a route through the city. We are especially interested in being able to locate specific intersections of streets, or named squares, plazas. etc. and to deal with city streets.

1. Design the class *Xing* that represents an intersection on a city map. It should include a name and the location information.

2. Next we need to define the class that represents a segment of the street that connects two intersections. For each street segment we need to know the name of the street and its starting and ending intersection.

   Design the class *Road* to represent one street segment.

3. Finally, we need a list of street segments that may represent either a routing in the directions generated by a map program, or the list may represent one street in the city.

   Design the classes to represent a list of *Road*s — call it a *Route*.

4. Design the method that determines the distance each *Road* covers.

5. Design the method that computes the length of a *Route*.

6. A *Route* provided by a map program must have the street segments connected to each other — i.e. next segment must start where the previous one ended. Design the method *isRoute* that determines whether a *Route* represents valid map directions.

7. A street in a city not only consists of adjacent street segments, but additionally, every segment has the same street name. Design the method *isStreet* that determines whether a *Route* represents a city street.

8. **Optional**

Design the methods and classes that will draw each of the intersections as a black dot and will draw the streets as well. You do not need to add the names to the map — simple dots and black lines are fine. *Note: If you do this part, make it a separate program. You should still use the Beginner ProfessorJ language here.*

Main Assignment — Part 3

During the communist regime in the former USSR the censorship prevented people from having access to many books. It led to the development of an underground publishing chain known as *samizdat*. A person who would acquire a forbidden book would make several copies (typing on a typewriter with a copying paper) and distribute them to the trusted friends. Each person could only make a certain number of copies - depending on the typewriter they owned. The friends who received the copies would, in turn, do the same and distribute more copies. Our program will manage such *samizdat* dissemination chain.

3.5 Problem

Design the data representation for the *samizdat* chain. Make sure you include the person's name and the capacity (the number of copies the person can produce). Some of the chain members do not own typewriters, so they are not able to generate new copies of the book.

1. Design the classes needed to represent the information about the samizdat distribution. Include a class diagram. Make examples that correspond to the samizdat organization shown in the figure 1 — the capacity for each member is shown below the name, as is the list of people who receive copies from that member.

2. Design the method *count* that counts the number of members of the *samizdat* chain.

3. Design the method *totalCapacity* that determines the number of copies that can be produced by the chain. Include in the count the copies that the members of the chain receive.

```
Jon
3
|
Paul---------------->Pete
5                    3
|                    |
Dan-->Al-->Rick      Hal-->Jack
0     0    2         0     1
           |               |
           Joe-->Dave      Josh
           0     0         0
```

Figure 1: A sample Samizdat distribution system.

4. Design the method *excessCapacity* that determines how many books can be produced by the chain that are not already slated for distribution to existing members.

5. Design the method *friends* that produces a list of the names of all people who get the book from this chain.

∎