8 Stacks, Queues, Binary Search Trees: Traversals and Visitors

Portfolio Problems

Use the queue you have designed in Lab 8 to represent a queue of cars going across the canvas.

1. Design the class Car that represents one car of a random color. The new car starts on the left of the Canvas and it moves right on each tick. Of course, the car can be drawn on the given Canvas.

2. Now design a queue of cars. When the car reaches the right side of the Canvas, it is replaced by a new car that starts on the left. So, you need to remove the car from the front of the queue and add a new one at the end.

3. Design a world of moving cars. If you wish, you may have more than one line of traffic.

4. *Optional:* Add a class that represents a *chicken* or a *frog* that is trying to cross the street. You know what can happen!

Use the *idraw* package.

Pair Programming Assignment

8.1 Problem

Work out the Exercises 34.11 - 34.15.

8.2 Problem

Traversals over Binary Search Trees

Start with the code given in the **BST.zip** file. You should have the following files:

- *Book.java* our good old `Book` class that includes two `Comparator`s.
- *ABST.java* an `abstract` class that represents a generic binary search tree.

- *Leaf.java* an `abstract` class that represents a leaf of a generic binary search tree.
- *Node.java* an `abstract` class that represents a node of a generic binary search tree.
- *Algorithms.java* a class that contains methods that traverse over a generic binary search tree, relying on the `Traversal` interface.
- *Examples.java* that contains several examples of binary search trees of books and some sample tests.
- *ABSTvisitor.java* a visitor interface for a generic binary search tree that will allow us to define a number of methods easily.

In this problem you will work with the `Traversal` interface and see both its advantages and its shortcomings. The next problem deals with the tree visitor and illustrates its advantages.

A. Run the project. Build additional examples of binary search trees using the comparison by price.

B. Add tests similar to those already shown for the new data you have defined.

C. The class `Node` implements the methods `getFirst` and `getRest` in a very strange way. As you can see, some of the tests fail. Design the correct implementation of these methods.

D. In the `Algorithms` class design the method `totalPrice` that uses the hooks provided by the `Traversal` interface and computes the total price of all books in a binary search tree.

E. In the `Algorithms` class design the method `makeString` that uses the hooks provided by the `Traversal` interface and produces a `String` of all data in the binary search tree. You may add some *separators* between the individual data items, such as new line, comma, or semicolon.

8.3 Problem

Traversals are OK is you only want to see all data items in the tree, one at a time, is the order specified by the `Comparator`. But you loose a lot of information about the tree structure. Try to design the method that computes

the heights of the tree — the maximum number of generations of children, using the hooks provided by the `Traversal` interface.

A. Look at the `ABSTvisitor` class and at the class `CountNodes`. Add test cases in the `Examples` class for the additional trees you have defined earlier.

B. Design the class `ComputeHeight` that implements the `ABSTvisitor` by defining methods that compute the heights of the binary search tree.

C. Design the class `Contains`that implements the `ABSTvisitor` by defining methods that determine whether the given item matches one of the data items in the binary search tree.

   Hint: Look at the lecture notes for ideas.

   Note: The binary search tree is already equipped with a method that determines whether two items have matching values.