# 7 Shortest Path; Accumulators

## 7.1 Problem

In the Lab 7 you were working on the following problem:

> Finally, design the methods that will produce a list of stations we need to go through to travel from this station to the given station. Include the origin and destination in the list. Produce an empty list of stations, if there is no way of getting there from here.

1. Design the data definition of the routing from one station to another, so that it includes fields that record not only the station we travel through, but also the length of the route up to this station. It would be appropriate to call is a *Routing*, because we no longer can add arbitrary stations to the list, only those that are connected by a train route.

   For example, for the route a - 3 - c - 4 - s - 6 - p, the new routing would be recorded as 0 *a*, 3 *c*, 7 *s*, 13 *p*. A route from a to a is recorded as 0 *a*.

   On possible data definition would be:
   A Routing is one of
   - empty
   - structure: length, Station, Routing

2. Add a method *extendRouting* to the classes that represent the routing that adds a new station to the routing and increments the length of the routing by the specified distance.

3. Add a method to the classes that represent the routing that determines whether this routing is better than some other, according to the following criteria:

   - any routing is better than an empty one
   - a nonempty routing is better that some other routing, if its length is shorter

4. Design a new variant of your solution of the lab problem in which you produced a list from one station to another. The new variant produces a routing instead of just a list of stations. Use the number of stops as your measure of the length.

5. Modify the solution above, so it produces the shortest routing.
6. Design the methods that now produce the shortest routing measured by the time of travel.
7. Abstract over the two methods, by defining an interface for a function object that computes the length of travel between two stations.

   Remember, that the design recipe for abstractions asks you to implement the solutions to the original problem using the new abstraction. Do so for the two ways of measuring the length of the routing. Add another variant that measures the length of travel between two stations using the Manhattan distance between the locations of the stations. ∎

### 7.2 Problem

Repeat all of the previous tasks, using the data definitions from Part b of Lab 7. You may find it helpful to design a method that produces a list of zero, one, or two neighbors of a given station. ∎

### 7.3 Problem: Challenge

Make an example of a transit system where it is possible to return to the same station. Design a class hierarchy that represents this transit system as a list of stations, where each station has a list of neighbors. The neighbor information includes the route, the time to travel, and the name of the next station.

Design the method that produces the shortest path for this transit system. You will need to keep track of the routes to the stations already visited, as well as a list of stations you should visit next.

You may use either one of the three measures for computing the length of a route. ∎

### 7.4 Problem

In the Maze game from Exercise Set 5 you computed the maximum amount the girl can win. Produce a list of moves (left or right) the girl should do to win the maximum amount.∎

## Part 2

### 7.5 Problem

Return to the problem 3 from homework 3 where you designed the shape class hierarchy. Modify the examples to test cases in Java 1.4 and run them as a project in Eclipse. ∎

### 7.6 Problem

Run the Maze game in Eclipse. Again, modify the test cases as needed.

The instructions and the classes you need for representing the World will be provided separately. You will learn to use Eclipse in the lab on March 8th. ∎