10.2.1 Booking of flights

In order to support booking of flights we need that `FlightRoute` is extended with `scheduledFlightTime`, as this one of the criteria people use when booking:

```
class FlightRoute(flightNumber, origin, destination: var String):
   scheduledDepartureTime: var TimeOfDay
   scheduledArrivalTime: var TimeOfDay
   flights: obj OrderedList(Flight)
   -"-
   scheduledFlightTime -> sft: var Time.Hours:
      sft :=  scheduledArrivalTime - scheduledDepartureTime
```

Objects of class `Flight` are created as soon as it is possible to make bookings on this flight, typically some months before scheduled departure. At that time the `departureDate` is set. For this purpose, the class `FlightRoute` has the method `createFlight`:

```
class FlightRoute(flightNumber, origin, destination: var String):
   -:-
   createFlight(d: var Date):
      f: ref Flight
      f := Flight(d)
      f.departureTime := scheduledDepartureTime
      f.arrivalTime := scheduledArrivalTime
      flights.insert(f)
```

Booking is based upon choosing origin and destination airports, together with a date. In practise the airline will provide options for the given date plus/minus a couple of day; the following simply gives the flights at just one date.

For simplicity we have excluded the handling of seats, but we do that in section . Given origin and destination airports, and a date, we define a method `flightsForBooking` that delivers a list of the actual flights. The list delivered by this method will form the basis for a website where the `Flight` information is displayed together with e.g. price, in a form that makes it possible to select one of the flights and reserve seats.

```
flightsForBooking(from, to: var String, d: var Date)
      -> flights: ref OrderedList(Flight):
   flights := timeTable.flightsFromToAt(from, to, d)
```

This is based on a method `flightsFromToAt` in the `timeTable`:

```
timeTable: obj
   -"-
   flightsFromToAt(from, to: var String, d: var Date)
         -> flights: ref OrderedList(Flight):
      routesFromTo(from, to).scan
         current.flights.scan
            if (current.date = d) :then flights.insert(current)
```

which in turn is based on a method `routesFromTo`, also in `timeTable`:

```
timeTable: obj
   -"-
   routesFromTo(from, to: var String)
         -> routes: ref OrderedList(FlightRoute):
      entries.scan
         if (current.origin = from and current.destination = to)
            :then routes.insert(current)
```

Note that the call `routesFromTo(from, to)` in `flightsFromToAt` delivers an ordered list of references to `FlightRoute` objects. As an `OrderedList` has a `scan` method, the statement

```
   routesFromTo(from, to).scan
      current.flights.scan
         if (current.date = d) :then flights.insert(current)
```

describes the invocation of a singular method object with `routesFromTo(from, to).scan` as super method. The singular method has one statement which in turn is a singular method with `current.flights.scan` as super method. In the innermost scanning of the `flights` list, each element (`current`), where `date = d`, is inserted in `flights`.

So, in summary, the method `routes` finds all the `FlightRoute` objects that matches the origin/destination airports and delivers this as a list. For each of the `FlightRoute` objects in this list, scanning the `flights` list finds the list of `Flight` objects with the right departure date.