10.2-OLD The Flight Example

Description

+++ Det følgende skal måske stå i kap 5 eller et andet sted. Vi bruger det vel også i Transaction, og vel også i Account
OLM: enig – men ret så omfattende
Nemmere hvis du kommentarer med blå baggrund og tiny? ?

```
class Date(year: var integer, month: var integer, day: var integer): Value
   setDate(y: var integer, m: var integer, d: var integer):
      year := y
      month := m
      day := d
   asString -> s: var String:
      s := year + "." + month + "." + day

class TimeOfDay(th: var Time.Hours): Value
   - :
      in p: var TimeOfDay
      out r: var Time.Hours
      r.magnitude := th.magnitude - p.th

class AnyTime(d: var Date, td: var TimeOfDay(0 hours)): Value
   - :
      in p: var AnyTime(Date, TimeOfDay)
      out r: var Time.Hours
      r.magnitude := td.th.magnitude - p.td.th
```

In order to define the period between two values of `TimeOfDay` or `AnyTime`, the operator '`-`' (minus) is defined for both `TimeOfDay` and `AnyTime`.

These classes are in turn defined based upon the dimension `Time` defined as follows.

```
Time: obj Dimension
   class Hours: Unit{...}
   class Minutes: Unit{...}
   class Seconds: Unit{...}
   ...
```

Every dimension is defined by the units in which values in a dimension are given; for `Time` we have a.o. `Hours`, `Minutes`, and `Seconds` as shown above. The value is held by the `magnitude`.

```
class Dimension:
   class Unit: Value
     magnitude: var float
     ...
   ...
```

We will not go into details here, but for each unit of a dimension, values may be annotated by a string that denote the unit. For the unit `Hours` in `Time` this string is "`hours`", so the expression `TimeOfDay(0 hours)` makes a `TimeOfDay` value with unit `Hours` and the `magnitude` `0`.

According to Webster, a clock is for measuring Time, and a clock may actually hold a Time.Hour value. But is reset to zero at midnight. Unless using am/pm!?
Time24? +++ linken til hvordan er gjort i Java (https://www.baeldung.com/java-8-date-time-intro)

Flight Routes and Flights +++ skal blive 10.2.1

In the following we illustrate nested classes by an example in the domain of flights and flight routes. An airline company like

© Copyright by Ole Lehrmann Madsen and Birger Møller-Pedersen 2023.

SAS has a timetable with all their flight routes with flights, e.g. SK SK 1926 from Aarhus in Denmark to Oslo in Norway, and SK 1927 from Oslo to Aarhus.

Time tables, routes and flights are obvious represented by objects. They are used for two different purposes: as a basis for bookings (e.g as in the chapter on travel bookings, +++) and as a basis for a website that shows the status of flights at a specific airport, i.e. whether flights are scheduled (time of departure), cancelled, expected time of arrival, has arrived (and at what time), etc.

Flypladserne har et sådant system, eller det vil sige f.eks. Avinor i Norge,
https://www.avinor.no/konsern/tjenester/flydata/flydata-i-xml-format

Men uanset, så har SAS en hjemmeside hvor denne type information præsenteres:
https://www.sas.no/travel/flightstatus?date=2024-06-04&departureStation=OSL&arrivalStation=&

A timetable has an entry for each of the different routes, and in the model each entry is represented by an object of class `Route`:

```
timeTable: obj  -- alternative just Routes
   entries: obj OrderedList(Route)
   createFlights(d: var Date(0,0,0)):
      entries.scan
         current.createFlight(d)
```

Each `Route` object has attributes that represent the name of the route, the source and destination airports, the scheduled departure and arrival time, and scheduled flying time The flights of a route is represented by a list of `Flight` objects for each `Route` object.

```
class Route(name, origin, destination: var String):
   scheduledDepartureTime: var TimeOfDay
   scheduledArrivalTime: var TimeOfDay
   scheduledFlyingTime -> sft: var
 Time.Hours:
      sft :=  scheduledArrivalTime - scheduledDepartureTime
   setTime(dt: var TimeOfDay(0 hours),
           at: var TimeOfDay(0 hours))
      scheduledDepartureTime := dt
      scheduledArrivalTime := at

   flights: obj
 OrderedList(Flight)
   createFlight(d: var Date(0,0,0)):
      flights.insert(Flight(d, scheduledDepartureTime, scheduledArrivalTime))
```

The `scheduledFlyingTime` is only used on the website for making bookings, as this is an important information when booking, while `scheduledDepartureTime` and `scheduledArrivalTime` are also used for showing flight status, at airports or at the flight status website of the airline company. The reason that `scheduledDepartureTime` and `scheduledArrivalTime` are not modelled as parameters of `Route` is that these times may be changed at different setups of the time table.

Setting up the time table is done by a sequence of actions that generate `Route` objects, set the values of their attributes and insert them into the `timeTable`:

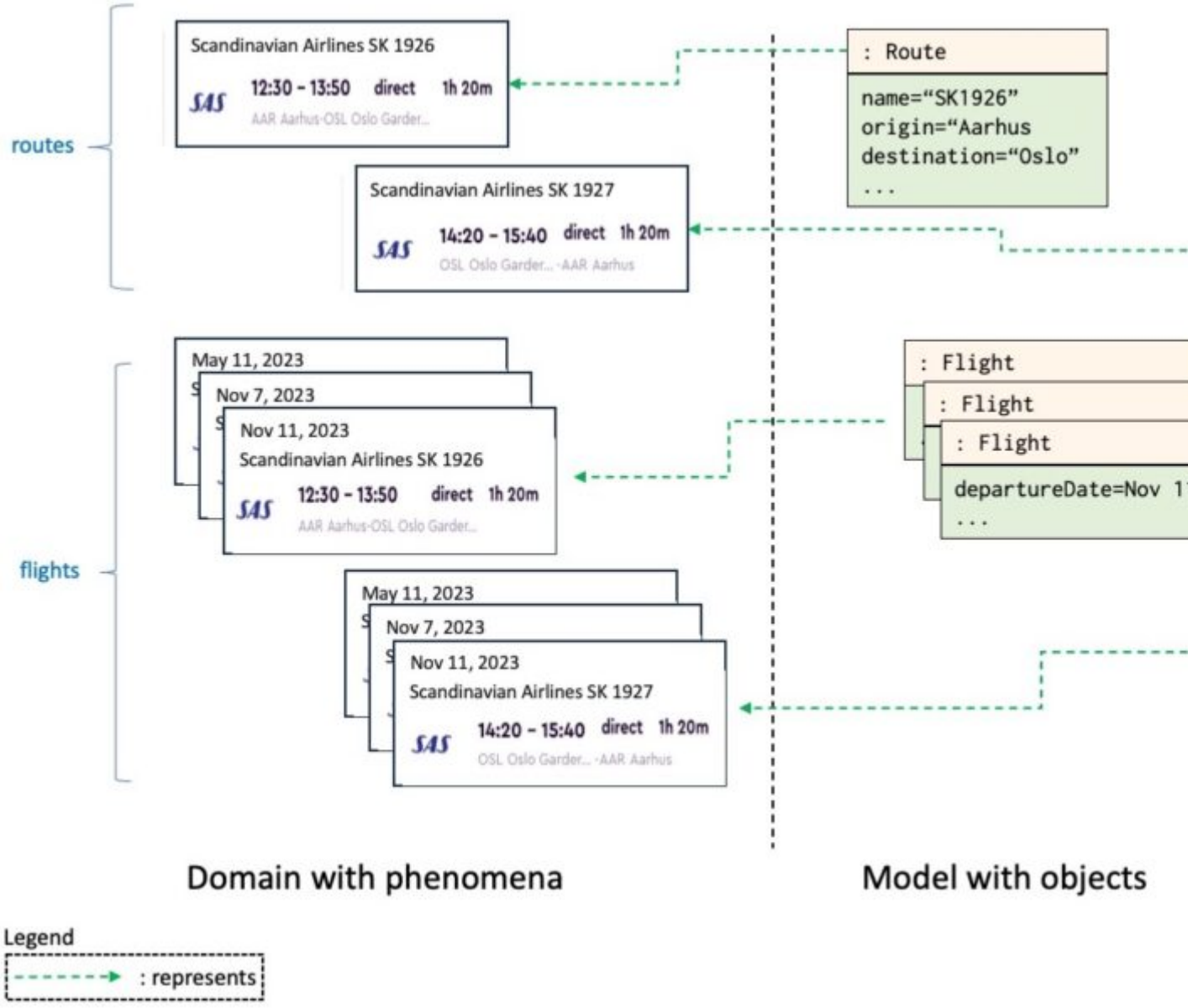
```
[redacted]
Sk1926.setTime(TimeOfDay([redacted] hours), TimeOfDay(13.50 hours))
```


© Copyright by Ole Lehrmann Madsen and Birger Møller-Pedersen 2023.

```
timeTable[redacted]
...
SK1927: obj Route("SK1927", "OSL", "AAR")
SK1927.setTime(TimeOfDay(14.20 hours), TimeOfDay(15.40 hours))

[redacted]
...
```

For a given route (e.g. SK1926) the `flights` attribute of the corresponding `Route` object keeps track of the flights of this route. This is illustrated below. For each route there is a `Route` object, and for each `Route` object there are a number of `Flight` objects. +++ kunne illustrere flights-attributten.



So far we have not modelled where the class `Flight` is defined. It could be a class along with the class `Route`, however, the class `Flight` that represent the flights of a given route is only meaningful in the context of the given route. As we have defined the class `Route`, so that each object of class `Route` represents a specific route, the class `Flight` is therefore defined in the context of class `Route`. A specific `Route` object will thereby have its own class `Flight` of objects representing flights on this route. Another Route object will have another class `Flight`.

+++ as mentioned A class is defined in the context of another class by *nesting* its description in the description of the

© Copyright by Ole Lehrmann Madsen and Birger Møller-Pedersen 2023.

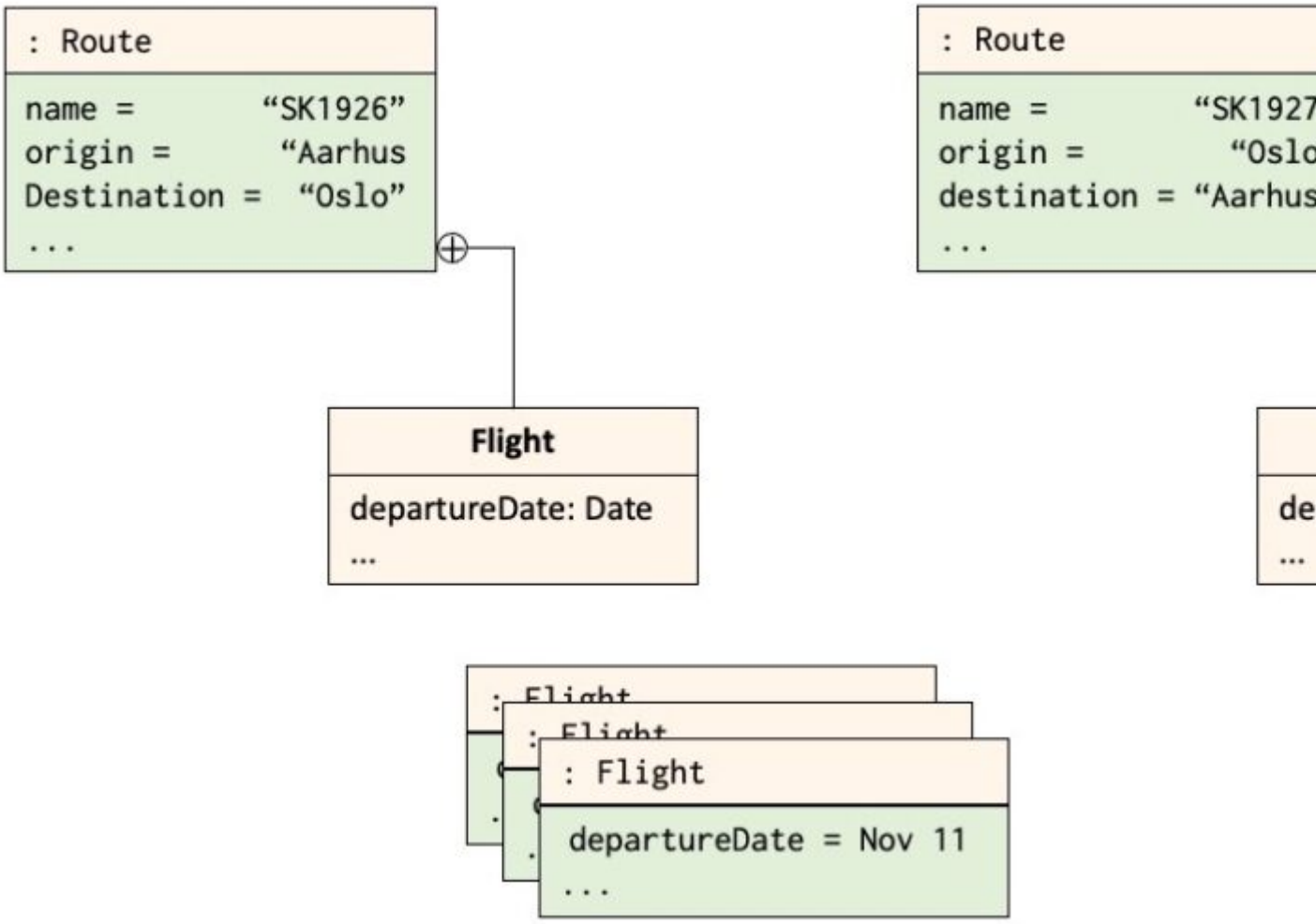
context class.

```
class Route(name, origin, destination: ref String):
   scheduledDepartureTime: var TimeOfDay
   scheduledArrivalTime: var TimeOfDay
   scheduledFlyingTime -> sft: var Time.Hours
:
      sft :=  scheduledArrivalTime - scheduledDepartureTime
   setTime: -- as above +++
   flights: obj OrderedList(Flight)

   class Flight(departureDate: var Date):
      seats: ...
      estimatedDepartureTime: var TimeOfDay
      estimatedArrivalTime -> e: var
 Time.Hours:
         e := estimatedDepartureTime + scheduledFlyingTime
      departureTime: var TimeOfDay  -- actual
      arrivalTime: var TimeOfDay    -- actual
      actualFlyingTime: -> aft: var Time.Hours
         aft := arrivalTime - departureTime
      delay -> period: var Time.Hours:
         period := arrivalTime – scheduledArrivalTime
      delayed: var Boolean
      delayDeparture(newTime: var Time):
      -- this is called in case the departure is delayed
         delayed := True
         estimatedDepartureTime := newTime
      hasArrived: var Boolean
      hasTakenOff: var Boolean
```

This is illustrated with circle-enclosed ‘+’ annotated relation between objects of class Route and the class Flight:

© Copyright by Ole Lehrmann Madsen and Birger Møller-Pedersen 2023.



Route objects with nested Flight classes and corresponding objects

A `Flight` object has attributes representing the actual departure time, the actual arrival time and the actual flying time. For the purpose of showing status of the flight in case the departure time is delayed, it also has the attribute `estimatedDepartureTime` and the method `estimatedArrivalTime`.

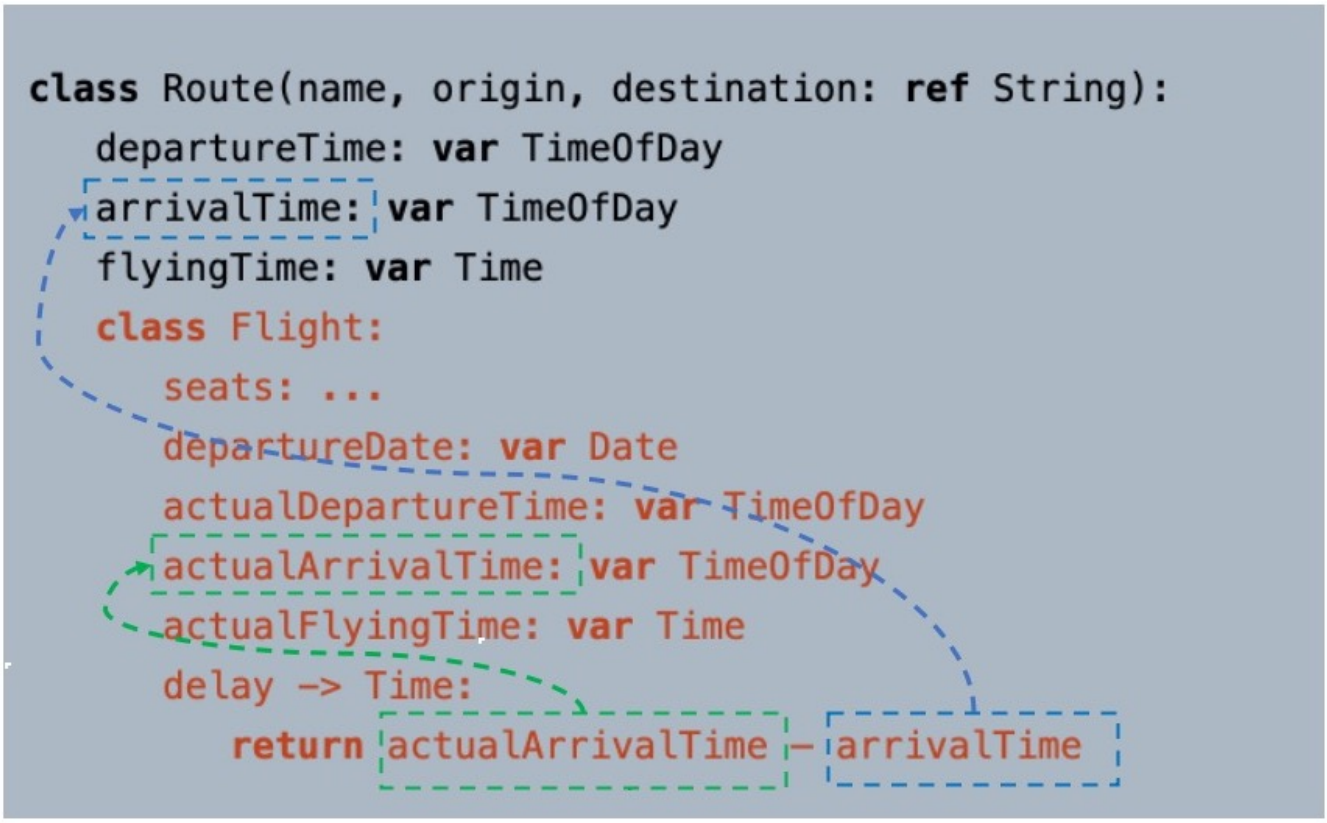
Objects of class `Flight` are created as soon as it is possible to make bookings on this flight, typically some months before scheduled departure. At that time the `departureDate` is set. The `estimatedDepartureTime` is set to the `scheduledDepartureTime` from its `Route` object, but this may be changed according to possible delays.

If the flight is delayed, the method `delayDeparture` of the actual `Flight` object is called, with the new time as parameter. In addition to setting `estimatedDepartureTime`, `delayed` is set to `True`.

At take off, `departureTime` is set to the time of take off, `hasTakenOff` is set to `True`, and `hasArrived` is set to `False`. While flying the attribute `arrivalTime` is set based on weather condition and the landing condition of the destination airport. It is therefore assumed that this is set based on real time information from the plane. At arrival, `hasArrived` is set to `True` and `hasTakenOff` to `False`.

The following shows how nesting is used to compute the delay. By nesting the `Flight` class in the `Route` class, the attributes of `Route` are directly visible in class `Flight`. The method `delay` (in class `Flight`) may therefore compute the delay of the flight by subtracting the `arrivalTime` (in the enclosing `Route`) from the local `Flight` property `ArrivalTime`
: +++ ændre figuren

© Copyright by Ole Lehrmann Madsen and Birger Møller-Pedersen 2023.



~~When it comes to where the `Flight` objects should be held, in practice this will probably be in a data base, together withe the `Route` objects. From this data base it will then be possible to extract the following sets/lists. In the following we will just make a simple object structure that can be used for both showing status of flights and for booking of flights.~~

For the purpose of creating `Flight` objects for a route, the class `Route` will have the method `createFlight`:

```
class Route(name, origin, destination: ref String):
   -- as above
   createFlight(d: var Date):
      f: ref Flight
      f := Flight(d)
      f.estimatedDepartureTime := scheduledDepartureTime
      flights.insert(f)
```

Showing status of flights

Status of flights are provided in two different ways, either given the source/destination airport at a given date, or given the name of the route, e.g. SK1926. In the first case the pair (airport origin/destination, date) shall extract a list of `Flight` objects from the data base. In the second case the extract is based upon the tuple (route name, origin/destination airport, date). For booking the tuple (source airport, destination airport, data) is used for the extraction.+++ måske i et senere afsnit. OLM: tror man skal holde sig til en af tilfældene så det blir så simplest som muligt.

In the following we will show how it would be done based upon the object structure given by the objects and classes introduced above: `timeTable` with a list of `Route` objects, and each `Route` object with a list of `Flight` objects. We define two of status methods of Flight that deliver strings that should be displayed on the status website (+++ nævne XML/HTML – ***OLM***: diskussion af et real system synes jeg man kan samle sidst som et discussion afsnit – har gjort det med 10.3): status on departure, and status on arrival.

The method departureStatus defined below is called before take off of the flight:

Synes stadig ikke man skal have både estimated DT of DT

© Copyright by Ole Lehrmann Madsen and Birger Møller-Pedersen 2023.

Og tror departureStatus kan gøres simplere med hjælpemetoder.
Meget lange eksempler nedenfor – list lost ?

```
class Flight:
   departureDate: var Date
   seats: ...
   estimatedDepartureTime: var TimeOfTime
   estimatedArrivalTime -> e: var TimeOfDay:
      e := estimatedDepartureTime + scheduledFlyingTime
   departureTime: var TimeOfDay -- actual/estimated
   arrivalTime: var TimeOfDay   -- actual/estimated
   scheduledDepartureTime -> sdt: TimeOfDay:
      sdt := this(Route).scheduledDepartureTime
   flyingTime: -> ft: var Time.Hours
      ft := arrivalTime - departureTime
   delay -> period: var Time.Hours:
      period:= arrivalTime – scheduledArrivalTime
   delayed: var Boolean
   delayDeparture(newTime: var Time):
      -- this is called in case the departure is delayed
      delayed := True
      estimatedDepartureTime := newTime
   hasTakenOff: var Boolean
   hasArrived: var Boolean

   departureStatus -> info: var String:
      -- this is called before the flight has taken off
      info := ("Flight " + name + " at: " + departureDate.asString)
      if departureDelayed :then
         info := info + "Estimated departure time: "+ estimatedDepartureTime +
         [redacted]
 + flyingTime)
      :else
         info := info + " On schedule: " +
                 F2S(scheduledDepartureTime.t.magnitude)
```

The next method is called after take off:

```
class Flight(departureDate: var Date):
  +++...

   arrivalStatus -> info: var String:
      -- this is called when the flight has taken off
      info := ("Flight " + name + " at: " + departureDate.asString)
      info := info + "Departed at: "+ departureTime
      [redacted]
      :else
         [redacted]
```

Based upon the entries in the time table, flight status is provided by:

```
showFlightStatus:
   timeTable.scanTimeTable
      r: ref Route
      r := current
      r.scan
         if (not hasTakenOff) :then
            currentFlight.departureStatus.print
         :else
            currentFlight.arrivalStatus.print
         newline
```


© Copyright by Ole Lehrmann Madsen and Birger Møller-Pedersen 2023.

This is based upon a `scanTimeTable` method:

```
timeTable: obj
   entries: obj OrderList(Route)
   scanTimeTable:
      current: ref Route
      entries.scan
          this(scanTimeTable).current := current
          inner(scanTimeTable)
   lookupRoute(routeId: var String) -> theRoute: ref
 Route:
         entries.scan
            if (current.name = routeId):then
                theRoute := current
                leave(LookupRoute)
```

which in turn is based upon a `scan` of routes:

```
class Route(name, origin, destination: ref String):
   -- as above
   scan:
       currentFlight: ref Flight
       flights.scan
          currentFlight := current
          inner(scan)

showFlightStatus:
   timeTable.scanTimeTable
      r: ref Route
      r := current
      r.scan
         if (not hasTakenOff) :then
            currentFlight.departureStatus.print
         :else
            currentFlight.arrivalStatus.print
         newline
```

For the purpose of providing status of flights we have two ways to ask for that: flights departing or arriving from a given airport at a given date, or flights of a given route at a given airport and date.

From/to a given airport, at a given date

```
fromAirport(ap: var String, d: var Date)
   -> flightList: ref OrderedList(Flight):
   routeList: obj OrderedList(Route)
   timeTable.entries.scan
      if (current.origin = ap :then
         routeList.insert(current)
   routeSet.scan
      current.flights.scan
         if (current.date = d) :then
            flightList.insert(current)

break+++

toAirport(ap: var String, d: var Date)
   -> flightList: ref OrderedList(Flight):
   routeList: obj OrderedList(Route)
   timeTable.entries.scan
      if (current.destination = ap :then routeList.insert(current)
   routeList.scan
      current.flights.scan
         if (current.date = d) :then flightList.insert(current)
```


© Copyright by Ole Lehrmann Madsen and Birger Møller-Pedersen 2023.

Before the list of `Flight` objects delivered by these two methods are used for producing the status website, the list delivered by `fromAirport` should be sorted according to departure time (in fact scheduled departure time, as this should be displayed together with the actual departure time), while the list of `Flight` objects delivered by `toAirport` should be sorted according to arrival time.

Given these two lists of `Flight` objects, the status website can produce the two strings delivered by the methods `departureStatus` and `arrivalStatus`.

```
fromAirport("OSL", Date(11,11, 1949)).scan
   current.departureStatus.print
fromAirport("OSL", Date(11,11, 1949)).scan
   current.arrivalStatus.print

fromAirport("ARR", Date(11,11, 1949)).scan
   current.departureStatus.print
fromAirport("ARR", Date(11,11, +++1949)).scan
   current.arrivalStatus.print
```

Given the Route name, airport, and a given date

For this case we just provide the code:

```
onRouteNameFrom(n: var String, from: var String d: var Date)
   -> flightList: ref OrderedList(Flight):
   theRoute: ref Route
   theRoute := timeTable.lookupRoute(n)
   if theRoute.origin = from :then
      theRoute.flights.scan
         if (current.date = d) :then flightList.insert(current)

++++ break

onRouteNameTo(n: var String, to: var String d: var Date)
   -> flightList: ref OrderedList(Flight):
   theRoute: ref Route
   theRoute := timeTable.lookupRoute(n)
   if theRoute.origin = to :then
      theRoute.flights.scan
         if (current.date = d) :then flightList.insert(current)

onRouteNameFrom("SK1926", "ARR" Date(11,11,1949)).scan
   current.departureStatus.print
onRouteNameTo("SK1926", "OSL" Date(11,11,1949)).scan
   current.arrivalStatus.print
```

Booking flights

It is not so obvious that the object structure that works for providing flight status also works for flight booking. The following shows that this is possible. Booking is based upon choosing origin and destination airports, together with a date. In practise the airline company will provide options for the given date plus/minus a couple of day; the following simply gives the flights at just on date:

```
flightsForBooking(from, to: var String, d: var Date)
   -> flightList: ref OrderList(Flight):
   routeList: obj OrderedList(Route)
   timeTable.entries.scan
      if (current.origin = from and current.destination = to)
         :then routeList.insert(current)
   routeList.scan
      current.flights.scan
         if (current.date = d) :then flightList.insert(current)
```

The list of `Flight` objects delivered by this method will form the basis for a website where the `Flight` information is

© Copyright by Ole Lehrmann Madsen and Birger Møller-Pedersen 2023.

displayed together with e.g. price, in a form that makes it possible to select one of the flights.

© Copyright by Ole Lehrmann Madsen and Birger Møller-Pedersen 2023.