10.2.3 Status of Flights

Description

In order to provide status on flights, we extend the first definition of class `Flight`:

```
   class Route(name, origin, destination: ref String):
      scheduledDepartureTime: var TimeOfDay
      scheduledArrivalTime: var TimeOfDay
      class Flight(departureDate: var Date):
         seats: ...
         departureTime: var TimeOfDay
            -- before take off: estimated departure time
            -- after take off: actual departure time
         arrivalTime -> at: var TimeOfDay:
            -- before take off: estimated arrival time
            -- after take off: actual arrival time
            at := departureTime + scheduledFlyingTime
         flyingTime: -> ft: var Time.Hours
            ft := arrivalTime - departureTime
         delayed: var Boolean
         delayDeparture(newTime: var
Time):
            -- this is called in case the departure is delayed
            delayed := True
            departureTime := newTime
         cancelled: var Boolean
         cancel:
            cancelled := True
         hasArrived: var Boolean
         hasTakenOff: var Boolean
         delay -> period: var Time.Hours:
            period := arrivalTime – scheduledArrivalTime
```

The scheduled arrival time is common to all flights on this route, while the arrival time of a flight is represented by a method the compute its arrival time by

```
arrivalTime -> at: var TimeOfDay:
   at := departureTime + scheduledFlyingTime
```

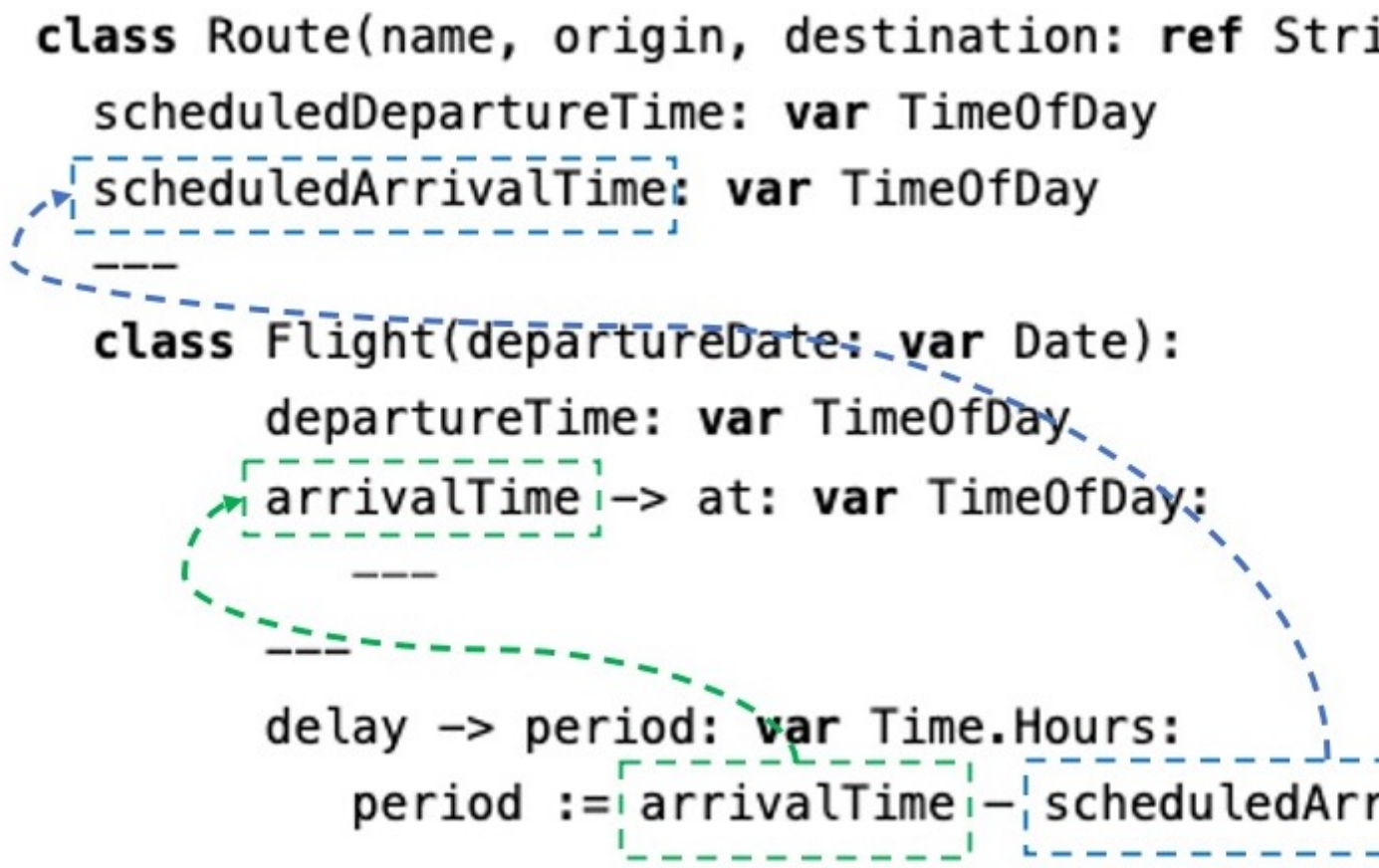
where `scheduledFlyingTime` is the property of the enclosing `Route` object and therefore visible in `Flight`).

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 `DepartureTime` to represent the estimated departure time, `delayed` is set to `True`.

In the event of cancellation, the method `cancel` is called.

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 actual delay is computed from the estimated arrival time (set while flying) and the scheduled arrival time.

The following illustration 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 `scheduledarrivalTime` (in the enclosing `Route`) from the local `Flight` property `ArrivalTime`:



+++

Status of flights are provided in two different ways, either given the origin/destination airport at a given date, or given the name of the route, e.g. SK1926.

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

```
class Flight(departureDate: var Date):
   ---
   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 +
         " expected arrival time: " + (DepartureTime + 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      if (not hasArrived) :then
         info := info + " expected at: " + arrivalTime
      :else
```

```
         info := info + " arrived at: " + ArrivalTime +
                 " delayed: " + delay
```

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
```

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

The following method delivers the list of flights for giving the status of all flights with a given origin 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)
```

It is left as a simple exercise to make the method that delivers the list of flights for giving the status of all flights with a given destination airport at a given date.

Svar:

```
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)
```

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(+++,+++, +++)).scan
   current.departureStatus.print
fromAirport("OSL", Date(+++, +++)).scan
   current.arrivalStatus.print

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

Given the Route name, airport, and a given date

The following method produces the list of flights given a certain route, from a given airport at a certain date:

```
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)
```

It is left as a simple exercise to make the method that produces the list of flights given a certain route, to a given airport at a certain date:

```
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)
```

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

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