Solution 4: Object creation

ETH Zurich

1 Creating objects in Traffic

Listing 1: Class *OBJECT_CREATION*

```
note
  description: "Creation class (Assignment 4)"
  date: "$Date$"
  revision: "$Revision$"

class
  OBJECT_CREATION

inherit

  TOURISM

feature -- Explore Paris
  explore
      -- Create new objects for Paris.
    do
      Paris.display
      add_passenger
      add_tram
      add_landmark
      add_free_moving
      add_line
      add_bus
    end

  add_passenger
      -- Add a new passenger to Paris.
    local
      p: TRAFFIC_PASSENGER
    do
      create p.make_with_route (Route3, 8.5)
      p.set_reiterate (True)
      Paris.put_passenger (p)
      p.go
    end

  add_tram
      -- Add a new tram to Paris.
    local
```

```
    t: TRAFFIC_TRAM
  do
    create t.make_with_line (Line1)
    Paris.put_tram (t)
    t.start
  end

add_landmark
    -- Add a landmark for station "Gare de Lyon" to Paris.
  local
    l: TRAFFIC_LANDMARK
  do
    create l.make (Station_gare_de_lyon.location, "Gare de Lyon", "train_station.png")
    Paris.put_landmark (l)
  end

add_free_moving
    -- Add a free moving to Paris.
  local
    pr: TRAFFIC_POINT_RANDOMIZER
    fm: TRAFFIC_FREE_MOVING
  do
    create pr.make (Paris.center, Paris.radius)
    pr.generate_point_array (10)
    create fm.make_with_points (pr.last_array, 15.7)
    Paris.put_free_moving (fm)
    fm.start
  end

tourist_line: TRAFFIC_LINE
    -- Tourist bus line.

add_line
    -- Create 'tourist_line' and add it to Paris.
  local
    t: TRAFFIC_TYPE_BUS
    c: TRAFFIC_COLOR
  do
    create t.make
    create tourist_line.make_with_terminal ("Tourist line", t, Station_gare_de_lyon)
    tourist_line.extend (Station_st_michel_notre_dame)
    tourist_line.extend (Station_champs_de_mars_tour_eiffel_bir_hakeim)
    tourist_line.extend (Station_charles_de_gaulle_etoile)
    tourist_line.extend (Station_palais_royal_musee_du_louvre)

    create c.make_with_rgb (255, 0, 127)
    tourist_line.set_color (c)

    Paris.put_line (tourist_line)
  end

add_bus
```

```
      -- Add a bus to Paris.
    local
      b: TRAFFIC_BUS
    do
      create b.make_with_line (tourist_line)
      b.set_reiterate (True)
      Paris.put_bus (b)
      b.start
    end
end
```

2 It's Logic!

Semi-strict (**and then**, **or else**, **implies**): evaluates the operands in order. The right side will be skipped if the result is already defined after evaluating the left side. Use it when the order of evaluation is important.

Strict (**and**, **or**): evaluation order is not specified. Compiler could:

- evaluate both sides at the same time (e.g. multi-processor optimization)
- evaluate the cheaper side first and skip the more expensive one if the result is already defined (e.g. single-processor optimization)

Use it when the order of evaluation doesn't matter.

Examples:

- $x >= 0$ **and** $x < 100$

  Strict operator is preferred, because the order of evaluation does not matter.

- *file.exists* **and then** *file.is_executable*

  We can only check executability of existing files, hence the semi-strict operator should be used.

- *list.before* **or** *list.after*

  Strict operator is preferred, because the order of evaluation does not matter.

- *string* = *Void* **or else** *string.is_empty*

  We can only call a query if the target is not *Void*, hence the semi-strict operator should be used.

3 Temperature application

Listing 2: Class *TEMPERATURE*

```
note
  description: "Temperatures."
  author: ""
  date: "$Date$"
  revision: "$Revision$"

class
  TEMPERATURE
```

```
create
  make_celsius, make_kelvin

feature -- Initialization
  make_celsius (v: INTEGER)
      -- Create with Celsius value 'v'.
    require
      above_absolute_zero: v >= - Celsius_zero
    do
      celsius_value := v
    ensure
      celsius_value_set: celsius_value = v
    end

  make_kelvin (v: INTEGER)
      -- Create with Kelvin value 'v'.
    require
      above_absolute_zero: v >= 0
    do
      celsius_value := v - Celsius_zero
    ensure
      kelvin_value_set: kelvin_value = v
    end

feature -- Access
  celsius_value: INTEGER
      -- Value in Celsius scale.

  kelvin_value: INTEGER
      -- Value in Kelvin scale.
    do
      Result := celsius_value + Celsius_zero
    end

  Celsius_zero: INTEGER = 273
      -- The zero of the Celsius scale in Kelvin scale.

feature -- Measurement
  average (other: TEMPERATURE): TEMPERATURE
      -- Average temperature between 'Current' and 'other'.
    do
      create Result.make_celsius ((celsius_value + other.celsius_value) // 2)
    end

invariant
  above_absolute_zero: kelvin_value >= 0
end
```

Listing 3: Class *APPLICATION*

```
note
  description : "Temperature application root class"
```

```
  date : "$Date$"
  revision : "$Revision$"

class
  APPLICATION

create
  execute

feature {NONE} -- Initialization

  execute
      -- Run application.
    local
      t1, t2, t3: TEMPERATURE
    do
      Io.put_string ("Enter the first temperature in Celsius: ")
      Io.read_integer
      create t1.make_celsius (Io.last_integer)
      Io.put_string ("The first temperature in Kelvin is: ")
      Io.put_integer (t1.kelvin_value)
      Io.new_line

      Io.put_string ("Enter the second temperature in Kelvin: ")
      Io.read_integer
      create t2.make_kelvin (Io.last_integer)
      Io.put_string ("The second temperature in Celsius is: ")
      Io.put_integer (t2.celsius_value)
      Io.new_line

      t3 := t1.average (t2)
      Io.put_string ("The average in Celsius is: ")
      Io.put_integer (t3.celsius_value)
      Io.new_line
      Io.put_string ("The average in Kelvin is: ")
      Io.put_integer (t3.kelvin_value)
      Io.new_line
    end

end
```

4 Ein ticket für alles

Listing 4: Class *CUSTOMER*

```
note
  description: "Public transportation system customers."

class
  CUSTOMER

create
```

```
  make

feature {NONE} -- Initialization
  make (fname, lname: STRING; bdate: DATE)
      -- Create a customer with a name 'fn' 'ln' and birth date 'bd'.
    require
      fname_exists: fname /= Void
      lname_exists: lname /= Void
      bdate_exists: bdate /= Void
    do
      first_name := fname
      last_name := lname
      birth_date := bdate
    ensure
      first_name_set: first_name = fname
      last_name_set: last_name = lname
      birth_date_set: birth_date = bdate
    end

feature -- Access
  first_name: STRING
      -- First name.

  last_name: STRING
      -- Last name.

  birth_date: DATE
      -- Birth date.

feature -- Measurement
  age: INTEGER
      -- Age (years).
    local
      today: DATE
    do
      create today.make_now
      Result := today.relative_duration (birth_date).year
    end

  has_discount: BOOLEAN
      -- Is the customer eligible for discount on the seasonal ticket?
    do
      Result := age < 25
    end

feature -- Output
  info: STRING
      -- String representation of the information about the customer.
    do
      Result := "First name: " + first_name + "%N" +
      "Last name: " + last_name + "%N" +
      age.out + " years old%N" +
```

```
      "Eligible for discount: " + has_discount.out + "%N"
    end

invariant
  first_name_exists: first_name /= Void
  last_name_exists: last_name /= Void
  birth_date_exists: birth_date /= Void
  info_exists: info /= Void
end
```

Listing 5: Class *APPLICATION*

```
note
  description : "ticket application root class"

class
  APPLICATION

create
  execute

feature {NONE} -- Initialization

  execute
      -- Run application.
    local
      name: STRING
      birth_date: DATE
      pos: INTEGER
      c: CUSTOMER
    do
      Io.put_string ("Enter full name: ")
      Io.read_line
      name := Io.last_string.twin

      Io.put_string ("Enter birth date as mm/dd/yyyy: ")
      Io.read_line
      create birth_date.make_from_string_default (Io.last_string)

      pos := name.index_of (' ', 1)
      create c.make (name.substring (1, pos - 1), name.substring (pos + 1, name.count),
          birth_date)

      Io.new_line
      Io.put_string (c.info)
    end

end
```