Assignment 6: Aliens

ETH Zurich

Due: 9. November 2007

Goals

• Refactor code to use routines.

• Apply your knowledge about inheritance, polymorphism, and dynamic binding.

1 Refactoring code

To do

In last week’s assignment you wrote some features that added trams and buildings to the map. You solved both tasks in a stepwise fashion: You first added trams (or buildings) for a single line (road) and then added another feature that did the same for all lines (all roads). In this week’s assignment your task is to refactor your code from last week, so that you omit code repetition.

1. Feature generate_trams_for_line8 and feature generate_trams have strong similarities: while generate_trams_for_line8 creates trams for every stop of Line8, generate_trams does the same for all tram lines. This observation suggests that feature generate_trams_for_line8 could be generalized to solve this task for any tram line, so that it then can be used by generate_trams. Adapt the code in this way and don’t forget to add appropriate contracts. Also adapt the code of explore_on_click to use the improved interface.

2. Do the same for generate_buildings_along_road and generate_buildings. There are slight adaptations:

• generate_buildings_along_road did not check for collisions with line connections, but the refactored code should always do this (also when buildings are generated for a single road).

• Generalize your code in a way such that not only roads can have buildings along it, but also line connections. Hint: TRAFFIC_LINE_CONNECTION and TRAFFIC_ROAD_CONNECTION both inherit their feature polypoints from TRAFFIC_CONNECTION.

To hand in

Hand in class LOOPINGS with your refactored code.
2 Aliens in Paris

Aliens are taking over Paris! In this task you will implement a special kind of TRAFFIC_PASSENGER: a TRAFFIC_ALIEN. Aliens have the following behavior: they appear in an arbitrary line vehicle (a tram or a bus) and disappear again after 4 minutes (in simulated time) of traveling with it to reappear in another bus or tram.

To do

1. Download http://se.ethz.ch/teaching/2007-F/eprog-0001/exercises/assignment_6.zip and extract it in traffic/example. You should now have a new directory traffic/example/ev_assignment_6 with assignment_6.ecf directly in it (it is important that the location corresponds to the description here!).

2. Open and compile this new project.

3. There is a bug in the postcondition Result_positive of feature duration in class TRAFFIC_TIME. Please replace the line with

   `Result_positive_or_zero : Result.is_positive or Result.is_zero`

4. Write a class TRAFFIC_ALIEN that has the described behavior. To do this you will need to inherit from TRAFFIC_PASSENGER and redefine its feature move. move should (instead of calculating a position as done in TRAFFIC_PASSENGER) take the location of the vehicle the alien is traveling with as its position (with a little offset to be able to see both the alien and the vehicle). Hints:

   - You will need to provide a creation procedure that gives a valid initial position, origin, and destination to the alien and creates the poly_cursor for it. We recommend you to create a coordinate object at (0,0) as position and use the same object as destination and origin. The poly_cursor needs a list of coordinates with which it works. Create an object of type `DS_ARRAYED_LIST [TRAFFIC_COORDINATE]` and add the object referenced through position twice. Then create the poly_cursor using the list by passing it as an argument.

   - You will need to generate random numbers, so that your aliens randomly pick their transport vehicle every 4 minutes. The class RANDOM provides a random number generator. The following code illustrates its usage (for more information see http://www.eiffelroom.com/article/random_numbers):

     ```eiffel
     local
     t : TIME
     random: RANDOM
     s : INTEGER
     do
       create t.make_now  -- Create a time object for the seed
       s := t.hour       -- Milliseconds since midnight
       s := s * 60 + t.minute
       s := s * 60 + t.second
       s := s * 1000 + t.milli_second
       create random.set_seed (s)  -- Create the random number generator
       with s as seed
       random.start  -- RANDOM is an infinite list: with
       i := random.item \ 100  -- start you set it to a first valid position
       -- Access the first random number: \ operator. Make it a number between 0..99
       random.forth  -- Advance the generator to get a new number
     end
     ```
• The alien should only finish its traveling when there are no vehicles in the city. Otherwise they go on traveling forever.

• Feature `time.actual_time` provides access to the simulated time of traffic (check the code of `move` in class `TRAFFIC_MOVING` on its usage).

• Feature `map` provides access to Paris if the alien is added to the map.

• Decide carefully on what inherited features you reuse.

• Make conscious decisions on when to create new objects and when to modify existing ones.

5. Test your `TRAFFIC_ALIEN` class by creating instances of it in feature `explore_on_click` of class `ALIEN_TEST`. You will also need to create trams and busses that your aliens can use.

To hand in

Hand in class `TRAFFIC_ALIEN` and `ALIEN_TEST`.

3 Inherited Fraction

Your task is to implement class `FRACTION` which represents fractions of the form \( \frac{\text{numerator}}{\text{denominator}} \). `FRACTION` inherits from `NUMERIC`. The class `FRACTION_TEST` shown in Listing 1 should work with your class without any changes.

`NUMERIC` is a deferred class providing the following features:

- `one`: like `Current`
- `zero`: like `Current`
- `divisible` (other: like `Current`): `BOOLEAN`
- `exponentiable` (other: like `Current`): `BOOLEAN`
- infix `"+"`(other: like `Current`): like `Current`
- infix `"−"`(other: like `Current`): like `Current`
- infix `"/"`(other: like `Current`): like `Current`
- infix `"∗"`(other: like `Current`): like `Current`
- prefix `"+"`: like `Current`
- prefix `"−"`: like `Current`

`like Current` in the feature signatures above states that the return type or argument type of a feature is the same as the type of `Current`. In the case of `FRACTION` this means that all the `like Currents` in fact denote the type `FRACTION`. 
To do

1. Create a new project with the root class `FRACTION_TEST`.

2. Copy and paste the class text in the root class from http://se.ethz.ch/teaching/2007-F/eprog-0001/exercises/fraction_test.e.

3. Add a new class `FRACTION` that inherits from `NUMERIC`, and implement the missing features. The fraction should always be reduced (gekürzt). To reduce a fraction, you can use a Greatest Common Divisor (GCD) algorithm, for example the Euclidian algorithm available at http://se.ethz.ch/teaching/2007-F/eprog-0001/exercises/gcd.txt. Other hints:
   - In Eiffel, integer division is done with ‘\//’, integer remainder (modulo) with ‘\%
   - Do not forget the contracts. `FRACTION` has an obvious invariant.
   - The feature `exponentiable` should always return `False`.
   - The feature `divisible` should only return `True` if the division is valid (i.e. does not result in a division through zero).
   - The feature `zero` returns an identity object for addition and subtraction, so that for fractions `f` and `zero` (`f + zero).is_equal(f)` and (`f - zero).is_equal(f)
   - The feature `one` returns an identity object for multiplication and division, so that for fractions `f` and `one` (`f * one).is_equal(f)` and (`f / one).is_equal(f)

Listing 1: Class `FRACTION_TEST`

class FRACTION_TEST
create make

feature -- Initialization
a, b, c: FRACTION
make is
  -- Creation procedure.
do
create a.make (1, 2)
create b.make (3, 4)
io.put_string ("Calculating with fractions:" + "%N%N")
io.put_string ("a: " + a.out)
io.put_new_line
io.put_string ("b: " + b.out)
io.put_new_line
c := a + b
io.put_string ("a + b: " + c.out)
io.put_new_line

c := a - b
io.put_string ("a - b: " + c.out)
io.put_new_line

c := a * b
io.put_string ("a * b: " + c.out)
io.put_new_line

c := a / b
io.put_string ("a / b: " + c.out)
io.put_new_line
To hand in

Hand in the source code of your class *FRACTION*. 