Classroom 2

ETH Zurich

Date: 10. January 2006

The classroom exercise intends to help you self-evaluate your knowledge and skills and lets us gain knowledge about the performance of our students. The setup resembles the situation you will encounter during the fall exam. The assistants will be happy to clarify any problems with the formulation of the tasks, but will not solve the tasks for you. This exercise will be corrected and graded by your assistant; the grade will not have any influence on the fall exam or the testate.

In this paper, the number of empty lines reserved for your answers is not a hint on the number of lines that you should fill in.

Please solve this exercise alone.

1 Pairs... and Pairs of Pairs

Consider class *PAIR* [*G*, *H*] given below, which represents pairs of elements.

1.1 A client for class PAIR

Fill in the source of class *CLIENT_OF_PAIR* so that:

- its feature *pair_of_integers* returns a pair made of the two integers that it receives as arguments
- its feature *pair_of_pairs_of_strings* returns a pair whose first element is a pair made of the first two strings that it receives as arguments, and whose second element is a pair made of the third and fourth strings that the routine receives as arguments

1.2 A client for class CLIENT_OF_PAIR

Fill in the source of class *ROOT_CLASS* so that its *make* feature creates a pair of pairs of strings, where the first pair consists of the strings "a" and "b", and the second of the strings "c" and "d". Then *make* must print the concatenation of the members of the pair of pairs of strings. In other words, the output of *make* must be "abcd".

```
   class
 2   PAIR [G, H]

 4 create
     make
 6
   feature -- Initialization
 8
     make (f: G; s: H) is
10           -- Create a new pair with first member ‘f’ and second member ‘s’.
         do
12           first  := f
             second := s
14       ensure
             first_set :  first  = f
16           second_set: second = s
         end
18

20 feature -- Access

22   first :  G
       -- First member of the pair
24
     second: H
26     -- Second member of the pair

28 end
```

```
   class
 2   CLIENT_OF_PAIR

 4 feature -- Basic operations

 6   pair_of_integers (i1, i2: INTEGER): .................................. is
           -- Pair made of the two integers ‘i1’ and ‘i2’
 8       do

10           ...............................................................................

12           ...............................................................................

14           ...............................................................................

16           ...............................................................................
         end
18
     pair_of_pairs_of_strings (s1, s2, s3, s4: STRING):
                   .................................................. is
20           -- Pair consisting of two pairs of strings
             -- (‘s1’ and ‘s2’ form one pair, ‘s3’ and ‘s4’ form another pair, and these 2 pairs
                  are also grouped in a pair.)
22       local

24           ...............................................................................

26           ...............................................................................
         do
```

```
28      ..............................................................................

30      ..............................................................................

32      ..............................................................................

34      ..............................................................................

36      ..............................................................................
    end
```

```
   class
 2   ROOT_CLASS

 4 create
     make
 6
   feature
 8
     make is
10         -- Creates a pair of pairs of strings using the strings "a", "b", "c", "d"
           -- and then prints the concatenation of the members of each element of this pair.
12     local
         cp: CLIENT_OF_PAIR
14
         ..........................................................................
16
         ..........................................................................
18     do
         ..........................................................................
20
         ..........................................................................
22
         ..........................................................................
24
         ..........................................................................
26
         ..........................................................................
28
         ..........................................................................
30       print   (................   +   ...............   +   ...............   +   ...............   )
       end
32
   end
```

2 Inversion of Linked List

Consider the following classes *SINGLE_LINKED_LIST* [*G*] and *SINGLE_CELL* [*G*] implementing a single linked list. The head of the list (first element of the list) is stored in the attribute *first* of the class *SINGLE_LINKED_LIST* [*G*]. Attribute *next* of class *SINGLE_CELL* [*G*] delivers the next cell (instance of the class *SINGLE_CELL* [*G*]). Calling *next* on the last cell (instance of the class *SINGLE_CELL* [*G*]) will return a **Void** reference.

```
   class
 2   SINGLE_LINKED_LIST [G]

 4 feature -- Access

 6   first : SINGLE_CELL [G]
         -- Head element of the list, 'Void' if the list is empty
 8
   feature -- Basic operations
10
     invert is
12       -- Invert the order of the elements of the list .
         -- E.g. the list [6, 2, 8, 5] should be become [5, 8, 2, 6].
14     local

16       ..............................................................................
       do
18
         ..............................................................................
20
         ..............................................................................
22
         ..............................................................................
24
         ..............................................................................
26
         ..............................................................................
28
         ..............................................................................
30
         ..............................................................................
32
         ..............................................................................
34
         ..............................................................................
36
         ..............................................................................
38
         ..............................................................................
40
         ..............................................................................
42
         ..............................................................................
44
         ..............................................................................
46     end
   end
48
   class
50   SINGLE_CELL [G]

52 feature -- Access

54   next: SINGLE_CELL [G]
         -- Reference to the next generic list cell of a list
56
```

```
58 feature -- Element change

60   set_next (an_element: SINGLE_CELL [G]) is
          -- Set 'next' to 'an_element'.
62     ensure
          next_set: next = an_element
64
   end
```

Implement the feature *invert* of class *SINGLE_LINKED_LIST* [*G*], so that it inverts the order of the elements in the list. If we have e.g. the list [6, 2, 8, 5] (with 6 being the first element of the list and 5 the last element) inverting it should result in [5, 8, 2, 6]. Do not create objects of type *SINGLE_CELL* [*G*] and also do not introduce any new feature in class *SINGLE_LINKED_LIST* [*G*] and *SINGLE_CELL* [*G*].

3 Polymorphism and dynamic binding

Consider the inheritance hierarchy shown in Figure 1 and the corresponding class text shown in Listing 1.

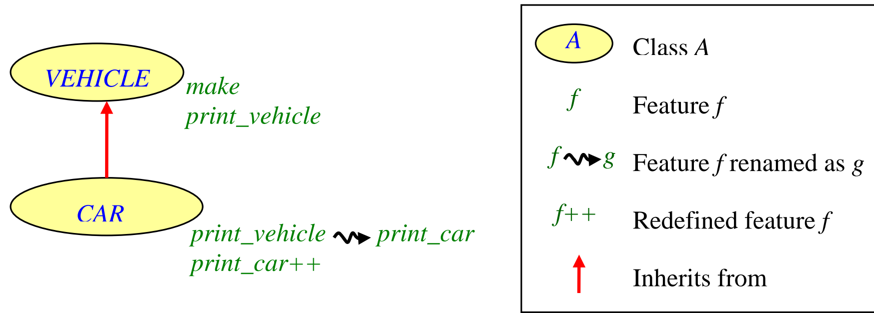

Figure 1: inheritance hierarchy

Listing 1: Classes *VEHICLE*, *CAR*

```
   class
 2   VEHICLE
   create
 4   make
   feature -- Initialization
 6   make is
        -- Initialize vehicle.
 8     do
          ...
10     end
   feature -- Output
12   print_vehicle is
        -- Print message.
14     do
        io.put_string ("This is a vehicle.")
16     end
   end
18
   class
20   CAR
   inherit
22   VEHICLE
       rename
24       print_vehicle as print_car
       redefine
26       print_car
       end
28 create
     make
30 feature -- Output
     print_car is
32      -- Print message.
```

```
    do
34    io.put_string ("This is a car.")
    end
36 end
```

Listing 2: Classes *SIMULATION*

```
   class
 2   SIMULATION
   create
 4   make
   feature {NONE} -- Initialization
 6   make is
         -- Initialize ' traffic '.
 8     do
         create traffic.make
10     end
   feature -- Access
12   traffic : LINKED_LIST [VEHICLE]
         -- Vehicles in traffic
14 feature -- Element change
     add_vehicle_to_traffic is
16       -- Extend 'traffic' with a vehicle.
       local
18         ...
       do
20         ...
       ensure
22       one_more: traffic.count = old traffic.count + 1
       end
24 invariant
       traffic_not_void : traffic /= Void
26 end
```

Class *SIMULATION*, as shown in Listing 2 has a list of *VEHICLE*s called *traffic* . As you can see command *add_vehicle_to_traffic* is incomplete. The goal of this exercise is to find three different ways to implement this command.

Example

Question

Complete the following code so that the message displayed at execution is: “This is a vehicle”, and explain why the code you wrote works (mention principles of object-oriented programming to explain).

```
   add_vehicle_to_traffic is
2     -- Extend 'traffic' with a vehicle.
   local
4     v: VEHICLE
   do
6
      ......................................................................................
8
      v. print_.............................................................................
10 ensure
      one_more: traffic . count = old traffic . count + 1
12 end
```

Answer:

```
   add_vehicle_to_traffic is
2     -- Extend 'traffic' with a vehicle.
    local
4     v: VEHICLE
    do
6     create v.make
      v. print_vehicle
8   ensure
      one_more: traffic . count = old traffic . count + 1
10  end
```

Explanation

The only feature we have at our disposal to display “This is a vehicle.” is the feature *print_vehicle* defined in class *VEHICLE*. Therefore the last line should be *v. print_vehicle* . For this code to be valid *v* needs to be created as an instance of type *VEHICLE*; thus the second line should be **create** *v.make*. Hence the above code.

To do:

Question

Complete the following code so that the message displayed at execution is: “This is a car”, and explain why the code you wrote works (mention principles of object-oriented programming to explain).

Note: There is exactly one instruction missing on the first dotted line, and a part of a feature name missing on the dotted segment.

```
   add_vehicle_to_traffic is
2        -- Extend 'traffic' with a vehicle.
     local
4        c: CAR
     do
6
         ..........................................................................
8         traffic.extend (c)

10       c.print_ ..................................................................
     ensure
12       one_more: traffic.count = old traffic.count + 1
     end
```

Explanation

....

....

....

....

....

....

....

....

....

....

Question

Complete the following code so that the message displayed at execution is: "This is a car", and explain why the code you wrote works (mention principles of object-oriented programming to explain).

Note: There is exactly one instruction missing on the first dotted line, and a part of a feature name missing on the dotted segment.

```
   add_vehicle_to_traffic is
 2    -- Extend 'traffic' with a vehicle.
    local
 4    v: VEHICLE
    do
 6
      ..............................................................................
 8     traffic.extend (v)

10    v.print_ .........................................................................
    ensure
12    one_more: traffic.count = old traffic.count + 1
    end
```

Explanation

....

....

....

....

....

....

....

....

....

....

....

Question

Complete the following code so that the message displayed at execution is: “This is a car”, and explain why the code you wrote works (mention principles of object-oriented programming to explain).

Note: There is exactly one instruction missing on the first dotted line, and a part of a feature name missing on the dotted segment.

```
   add_vehicle_to_traffic is
2        -- Extend 'traffic' with a vehicle.
     local
4        v: VEHICLE
         c: CAR
6    do

8        ..............................................................................
         v := c
10       traffic.extend (v)

12       v.print_ ......................................................................
     ensure
14       one_more: traffic.count = old traffic.count + 1
   end
```

Explanation

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .