Exercise 5: Multiple inheritance, Agents

Hand-out: 14 May 2004
Due: 28 May 2004

Please solve this exercise alone.

**Summary: Inheritance, Agents**

*Name clashes under multiple inheritance*

Name clashes may occur in case of multiple inheritance. To solve them, Eiffel has five feature adaptation clauses:

- **rename**: to change the name of a parent feature.
- **export**: to modify the visibility of a parent feature.
- **undefine**: to make a parent feature deferred.
- **redefine**: to redefine the signature and/or the body of a parent feature.
- **select**: to select a parent feature.

*Deferred classes and features*

A feature is either deferred or effective.

To effect an inherited feature (deferred in the parent) is to make it effective. No need for a redefine clause.

Like a feature, a class is either deferred or effective.

A class is deferred if it has at least one deferred feature (possibly coming from an ancestor) that it does not effect. It is effective otherwise.

A deferred class may not be instantiated.

BUT: A deferred class may have assertions (in particular, a deferred routine may have a precondition and a postcondition, and the class may have a class invariant).

*Agents*

An agent expression is of the form:

```
agent your_function (?, u, v)
```

where ? corresponds to an open argument (i.e. set at the time of any call to the agent) and 

```
u and v correspond to close arguments (i.e. set at the time of the agent’s definition).
```

It is also possible to specify a target, as follows:

```
agent some_object.some_routine (?, u, v)
```

where *some_object* is the target.
In the above example, the target is closed (i.e. it is \textit{some\_object}.) The target may also be open, like in:

\texttt{my\_employee\_list.for\_all(\texttt{agent \{EMPLOYEE\}.is\_married})}

\texttt{\{EMPLOYEE\}}” is the open target.

1. Multiple and repeated inheritance

Consider the following class diagram:

Here are the texts of classes \textit{DRIVER}, \textit{SWISS\_DRIVER}, and \textit{FRENCH\_DRIVER}:

```
class DRIVER

feature -- Access

  driver\_license: DRIVER\_LICENSE
  -- License of current driver

  violation\_count: INTEGER
  -- Number of times the driver has violated
  -- the highway code

feature -- Basic operations

  pay\_fee is
  -- Pay fee depending on the infringement done.
    require
    at\_least\_one\_violation: violation\_count >= 1
    do
    -- Do something here.
end

pay\_fee++

pay\_tax++

pay\_swiss\_fee

pay\_swiss\_tax

pay\_fee+

pay\_tax+

pay\_swiss\_fee

pay\_swiss\_tax

pay\_french\_fee

pay\_french\_tax
```

SWISS\_FRENCH\_DRIVER
pay_tax is -- Pay road tax.
  do -- Do something here.
  end

invariant

  violation_count_is_positive: violation_count >= 0
end

class

  SWISS_DRIVER

inherit

  DRIVER redefine
    pay_fee, pay_tax,
end

feature -- Basic operations

  pay_fee is -- Pay fee depending on the infringement done.
    do Precursor {DRIVER}
      pay_swiss_fee
    end

  pay_swiss_fee is -- Pay swiss fee depending on the infringement done.
    do
      -- Do something more here.
    end

  pay_tax is -- Pay road tax.
    do Precursor {DRIVER}
      pay_swiss_tax
    end

  pay_swiss_tax is -- Pay swiss tax depending on `tax_payed`.
    do
      -- Do something more here.
    end

feature -- Basic queries

  tax_payed: BOOLEAN -- Has the swiss driver already payed the annual tax?
end
class

   FRENCH_DRIVER

inhibit

   DRIVER
       redefine
           pay_fee,
           pay_tax
       end

feature -- Basic operations

   pay_fee is
       -- Pay fee depending on the infringement done.
       do
           Precursor {DRIVER}
           pay_french_fee
       end

   pay_french_fee is
       -- Pay french fee depending on infringement done.
       do
           -- Do something else here.
       end

   pay_tax is
       -- Pay road tax.
       do
           Precursor {DRIVER}
           pay_french_tax
       end

   pay_french_tax is
       -- Pay french tax depending on `current_highway'.
       do
           -- Do something more here.
       end

feature -- Basic queries

   current_highway: INTEGER
       -- The highway the driver is using currently.

end

To do

- Why, in Eiffel, any case of multiple inheritance is also a case of repeated inheritance?
- Write a possible implementation of class SWISS_FRENCH_DRIVER.

Hint

There are at least two solutions for class SWISS_FRENCH_DRIVER.
To hand in
Hand in the answer to the question on the previous page and the text of class
SWISS_FRENCH_DRIVER.

2. Deferred classes, Assertion inheritance
Consider the following classes A and B:

defered class
    A

feature – Access
    index: INTEGER
    -- Index

feature -- Element change
    change_index is
        -- Change `index'.
        require
        valid_index: index > 2
        deferred
        ensure
        index_changed: index > 10
    end

end

class
    B

inherit
    A
        redefine
        change_index
    end

feature -- Element change
    change_index is
        -- Change `index'.
### To do

- What is the main difference between Java/C# interfaces and Eiffel deferred classes?
- Classes $A$ and $B$ would not compile. Explain why.
- Correct the text of classes $A$ and $B$ to make them compile. Explain why the precondition and postcondition of class $B$ are “useless” in this case.

**Hint**

Look at assertions.

### To hand in

Hand in your answer to the above questions and the text of classes $A$ and $B$.

### 3. Agents

#### To do

- Write a class `EMPLOYEE` with at least the following information: name, age, salary, marital status, and gender. (Don’t forget the creation procedure and relevant setter procedures.)
- Write a class `CLIENT` (the root class), which creates a list of `EMPLOYEE`es with a few `EMPLOYEE`es in it and print the employees’ information using agents. You should display the following information:
  - Answer to the question: “Are all employees married?”
  - Answer to the question: “Are all employees women?”
  - Print the name, age, salary, marital status, and gender of all employees.

**Hint**

- You may need to add a few output features to the class `EMPLOYEE`.

### To hand in

Hand in the text of classes `EMPLOYEE` and `CLIENT`.

```eiffel
to do

require
  valid_index: index > 5
do
  -- Do something.
ensure
  index_changed: index > 7
end

end
```