Prova di esame del 24 settembre 2020

Esercizio 1) [10 punti]
Marcare le affermazioni che si ritengono vere. Ogni domanda può avere un qualunque numero naturale di affermazioni vere. Vengono **assegnati** 0.5 punti sia per ogni affermazione *vera che viene marcata* che per ogni affermazione *falsa che viene lasciata non marcata*. Analogamente vengono **sottratti** 0.5 punti sia per ogni affermazione *falsa che viene marcata* che per ogni affermazione *vera che viene lasciata non marcata*.

1. …
   a. Una funzione è una query implementata mediante memorizzazione
   b. Un attributo è una query implementata mediante memorizzazione
   c. Una routine è l'implementazione di una feature mediante computazione
   d. Un attributo è l'implementazione di una feature mediante computazione

2. Nell'istruzione **inspect** …
   a. È obbligatoria almeno una clausola *when*
   b. La variabile testata da *inspect* deve essere di tipo *INTEGER* oppure *BOOLEAN*
   c. È obbligatoria la clausola *else*
   d. I valori testati dalle clausole *when* devono essere tra loro disgiunti

3. Sia *f* una feature *effective* introdotta nella classe *A* e sia *B* una classe distinta che eredita da A. Allora …
   a. … sicuramente *f* è ancora *effective* in *B*
   b. … sicuramente le istanze di *B* possono usare *f*
   c. … per usare *f* in *B* si deve necessariamente attuare il suo *rename*
   d. … è possibile attuare un *rename* di *f* in *B*

4. La ridefinizione di una *feature* che è una funzione può…
   a. Cambiare l'implementazione purché vengano rispettati i contratti ereditati
   b. Cambiare i contratti purché vengano rispettati quelli ereditati
   c. Cambiare arbitrariamente la lista dei tipi degli argomenti
   d. Cambiare arbitrariamente il tipo del risultato

5. Se *f* è l'unico nome di feature nella clausola di dichiarazione **create** in una classe *C* allora *f*…
   a. … può essere usata in alternativa a *default_create* per creare istanze di *C*
   b. … deve essere usata per creare istanze di *C*
   c. … non può essere usata se non per creare istanze di *C*
   d. … può essere invocata per le istanze di *C*

SOLUZIONE:

1. …
   a. Una funzione è una query implementata mediante memorizzazione
   b. Un attributo è una query implementata mediante memorizzazione
   c. Una routine è l'implementazione di una feature mediante computazione
   d. Un attributo è l'implementazione di una feature mediante computazione

2. Nell'istruzione **inspect** …
   a. È obbligatoria almeno una clausola *when*
   b. La variabile testata da inspect deve essere di tipo *INTEGER* oppure *BOOLEAN*
   c. È obbligatoria la clausola *else*
   d. I valori testati dalle clausole *when* devono essere tra loro disgiunti

3. Sia *f* una feature *effective* introdotta nella classe *A* e sia *B* una classe distinta che eredita da A. Allora …
   a. … sicuramente *f* è ancora *effective* in *B*
   b. … sicuramente le istanze di *B* possono usare *f*
   c. … per usare *f* in *B* si deve necessariamente attuare il suo *rename*
   d. … è possibile attuare un *rename* di *f* in *B*

4. La ridefinizione di una *feature* che è una funzione può…
   a. Cambiare l'implementazione purché vengano rispettati i contratti ereditati
   b. Cambiare i contratti purché vengano rispettati quelli ereditati
   c. Cambiare arbitrariamente la lista dei tipi degli argomenti
   d. Cambiare arbitrariamente il tipo del risultato

5. Se *f* è l'unico nome di feature nella clausola di dichiarazione **create** in una classe *C* allora *f*…
   a. … può essere usata in alternativa a *default_create* per creare istanze di *C*
   b. … deve essere usata per creare istanze di *C*
   c. … non può essere usata se non per creare istanze di *C*
   d. … può essere invocata per le istanze di *C*

Esercizio 2) [10 punti]
Siamo in un contesto ***void safe***. La classe *INT_LINKABLE* modella il generico elemento di una lista di valori interi. La sua implementazione è la seguente:

```
class
   INT_LINKABLE
create
   set_value

feature -- accesso
   value : INTEGER
      -- L'intero memorizzato in questo elemento.

   next : detachable INT_LINKABLE
      -- Il successivo elemento della lista.

feature {NONE} -- assegnazione
   set_value (a_value : INTEGER)
      -- Assegna l'intero memorizzato in questo elemento.
      do
         value := a_value
      ensure
         value = a_value
      end

feature {NONE} -- manipolazione
   link_to (other: detachable INT_LINKABLE)
      -- Collega questo elemento con `other'.
      do
         next := other
      ensure
         next = other
      end

   link_after (other: INT_LINKABLE)
      -- Inserisce questo elemento dopo `other' conservando quello che c'era dopo.
      do
         link_to (other.next)
         other.link_to (Current)
      ensure
         next = old other.next
         other.next = Current
      end

end
```

Sempre in un contesto ***void safe***, la classe *INT_LINKED_LIST* modella una lista di interi. La sua attuale implementazione è solo quella fornita qua sotto:

```
class
   INT_LINKED_LIST

feature -- Accesso
   first_element: detachable INT_LINKABLE
        -- Il primo elemento della lista.

   last_element: detachable INT_LINKABLE
        -- L'ultimo elemento della lista.
```

```
   active_element: detachable INT_LINKABLE
        -- L'elemento corrente della lista, cioè il cursore
        -- Può essere Void anche se la lista non è vuota.

   count: INTEGER
        -- Il numero di elementi della lista.

feature -- Spostamento del cursore
   start
        -- Sposta il cursore al primo elemento.
      do
        active_element:= first_element
      ensure
        active_element = first_element
      end

   last
        -- Sposta il cursore all'ultimo elemento.
      do
        active_element:= last_element
      ensure
        active_element = last_element
      end

   forth
        -- Sposta l'elemento corrente, se esiste, all'elemento successivo.
      do
        if attached active_element as ae then
          active_element:= ae.next
        end
      ensure
        attached old active_element as ae implies active_element = ae.next
      end

invariant
   count >= 0
   attached last_element as le implies le.next = Void
   count = 0 implies (first_element = last_element) and (first_element = Void)
     and (first_element = active_element)
   count = 1 implies (first_element = last_element) and (first_element /= Void)
     and attached active_element as ae implies (ae = first_element)
   count = 2 implies (first_element /= last_element) and (first_element /= Void)
     and (last_element /= Void)
     and attached active_element as ae implies (ae = first_element or ae = last_element)
     and attached first_element as fe implies (fe.next = last_element)
   count > 2 implies (first_element /= last_element) and (first_element /= Void)
     and (last_element /= Void)
     and attached first_element as fe implies (fe.next /= last_element)

end
```

Aggiungere alla classe *INT_LINKED_LIST* una feature *insert_before* (*a_value, target*) che inserisce un elemento contenente *a_value* subito prima della prima occorrenza dell'elemento contenente *target*, se presente nella lista, e lo inserisce all'inizio della lista altrimenti. L'elemento atttivo non viene modificato. Le linee vuote sono solo indicative e non corrispondono alle linee di codice effettivamente necessarie.

```
feature -- Inserimento singolo vincolato
   insert_before (a_value, target: INTEGER)
-- Aggiunge `a_value' subito prima della prima occorrenza di `target', se esiste,
-- altrimenti lo aggiunge all'inizio della lista. Non modifica `active_element'.

   require

   _____________________________________________________________________________

   _____________________________________________________________________________

   local

   _____________________________________________________________________________

   _____________________________________________________________________________

   do

   _____________________________________________________________________________

   _____________________________________________________________________________

   _____________________________________________________________________________

   _____________________________________________________________________________

   _____________________________________________________________________________

   _____________________________________________________________________________

   _____________________________________________________________________________

   _____________________________________________________________________________

   _____________________________________________________________________________

   _____________________________________________________________________________

   _____________________________________________________________________________

   _____________________________________________________________________________

   _____________________________________________________________________________

   _____________________________________________________________________________

   _____________________________________________________________________________

   ensure

   _____________________________________________________________________________

   _____________________________________________________________________________

   end
```

SOLUZIONE:

```
feature -- Inserimento singolo vincolato
   insert_before (a_value, target: INTEGER)
-- Aggiunge `a_value' subito prima della prima occorrenza di `target', se esiste
-- altrimenti lo aggiunge all'inizio della lista. Non modifica `active_element'.
local
   current_element, new_element: like first_element
do
   create new_element.set_value (a_value)
   if count = 0 then
      first_element := new_element
      last_element := first_element
   else
      if attached first_element as fe and then fe.value = target then
         new_element.link_to (first_element)
         first_element := new_element
      else -- la lista contiene almeno un elemento e il primo element non è `target'
         from
            current_element := first_element
         until
            attached current_element as ce implies ce.next = Void or else
                                          (attached ce.next as cen implies cen.value = target)
         loop
            current_element := ce.next
         end
         if attached current_element as ce then
            if ce.next = Void then
            -- la lista non contiene `target'
               new_element.link_to (first_element)
               first_element := new_element
            else
               new_element.link_after (ce)
            end
         end
      end
   end
   count := count + 1
ensure
   uno_in_piu: count = old count + 1
   valore_aggiunto: has (a_value)
   in_testa se_non_presente: not (old has (target)) implies
                                (attached first_element as fe implies fe.value = a_value)
end
```

Versione alternativa con due cursori che scorrono la lista

```
feature -- Inserimento singolo vincolato
   insert_before_with_2_cursors (a_value, target: INTEGER)
-- Aggiunge `a_value' subito prima della prima occorrenza di `target', se esiste
-- altrimenti lo aggiunge all'inizio della lista. Non modifica `active_element'.
local
   previous_element, current_element, new_element: like first_element
do
   create new_element.set_value (a_value)
   if count = 0 then
      first_element := new_element
      last_element := first_element
   else -- la lista contiene almeno un elemento
      from
         previous_element := Void
         current_element := first_element
      until
         attached current_element as ce implies ce.value = target
      loop
         previous_element := current_element
         current_element := current_element.next
      end
      if current_element = Void then
         -- la lista non contiene `target'
         new_element.link_to (first_element)
         first_element := new_element
      else -- `current_element' contiene `target'
         if previous_element = Void then
            new_element.link_to (first_element)
            first_element := new_element
         else
            previous_element.link_to (new_element)
            new_element.link_to (current_element)
         end
      end
   end
   count := count + 1
ensure
   uno_in_piu: count = old count + 1
   valore_aggiunto: has (a_value)
   in_testa se_non_presente: not (old has (target)) implies
                                (attached first_element as fe implies fe.value = a_value)
end
```

Esercizio 3) [10 punti]

Discutere le caratteristiche del meccanismo di *genericità* nei linguaggi orientati agli oggetti e come viene attuato in Eiffel.

SOLUZIONE:

La genericità è un meccanismo dei linguaggi orientati agli oggetti che consente di definire classi con argomenti non completamente specificati, cioè appunto "generici".

Un esempio tipico è quello di una classe che definisce una struttura dati di tipo "lista". Con il meccanismo di genericità si può definire la classe "lista" prescindendo dal tipo di oggetti gestiti, cioè definire caratteristiche e operazioni di manipolazione per una lista di oggetti "generici".

La classe così definita è appunto una classe generica, che possiede cioè nella sua definizione uno o più parametri di tipo non specificato. Solo nel momento del suo effettivo utilizzo, cioè quando si dichiara una variabile come del tipo definito dalla classe, il tipo del parametro viene specificato, rendendo così la classe completamente utilizzabile.

In Eiffel la definizione di una classe generica avviene indicando un nome di tipo non definito nella definizione della classe, come nel seguente esempio in cui si assume che G sia un nome di tipo non definito nel sistema

```
class CLASSE_GENERICA [ G ]
feature
        un_comando (x : G )
                ...
        una_query : G
                ...
end
```

L'utilizzo in Eiffel di tale classe generica avviene quando viene definita una variabile col tipo di questa classe, sostituendo il parametro (generico) con un nome di tipo esistente, come ad esempio:

```
class UNA_MIA_CLASSE
feature
        una_mia_variabile : CLASSE_GENERICA [ NOME_DI_TIPO_ESISTENTE ]
                ...
end
```