Prova di esame del 19 luglio 2019

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

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

3. Ridefinire (*redefine*) una feature *f* ereditata implica …
   a. … cambiare il nome della feature senza necessariamente cambiarne l'implementazione
   b. … cambiare l'implementazione della feature senza necessariamente cambiarne il nome
   c. … cambiare sia il nome che l'implementazione della feature
   d. … portare lo stato della feature a *effective*

4. Una struttura di dati di tipo *dispenser* (cioè che fornisce su richiesta un elemento da elaborare) la cui politica di gestione fornisce come elemento da elaborare l'elemento meno recentemente inserito in essa è una…
   a. … pila o *stack*
   b. … coda
   c. … array
   d. … tabella *hash*

5. L'istanza di una classe *C* che esporta la feature *x* verso sé stessa …
   a. Può invocare la feature *x* di sé stessa solo in modo qualificato
   b. Può invocare la feature *x* di sé stessa
   c. Può invocare la feature *x* di altre istanze di *C* solo in modo qualificato
   d. Non può in alcun modo invocare la feature *x* di altre istanze di *C*

SOLUZIONE:

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

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

3. Ridefinire (*redefine*) una feature *f* ereditata implica …
   a. … cambiare il nome della feature senza necessariamente cambiarne l'implementazione
   b. … cambiare l'implementazione della feature senza necessariamente cambiarne il nome
   c. … cambiare sia il nome che l'implementazione della feature
   d. … portare lo stato della feature a *effective*

4. Una struttura di dati di tipo *dispenser* (cioè che fornisce su richiesta un elemento da elaborare) la cui politica di gestione fornisce come elemento da elaborare l'elemento meno recentemente inserito in essa è una…
   a. … pila o *stack*
   b. … coda
   c. … array
   d. … tabella *hash*

5. L'istanza di una classe *C* che esporta la feature *x* verso sé stessa …
   a. Può invocare la feature *x* di sé stessa solo in modo qualificato
   b. Può invocare la feature *x* di sé stessa
   c. Può invocare la feature *x* di altre istanze di *C* solo in modo qualificato
   d. Non può in alcun modo invocare la feature *x* di altre 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

   count: INTEGER
        -- Il numero di elementi della lista

feature -- spostamenti cursore
   start
        -- Sposta `active_element' al primo elemento
      do
         active_element:= first_element
      ensure
         active_element = first_element
      end

   last
        -- Sposta `current_element' all'ultimo elemento
      do
         active_element:= last_element
      ensure
         active_element = last_element
      end

   forth
        -- Sposta `active_element' al successivo elemento, se esiste
      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 *get_element* (*a_value*) che ritorna il primo elemento contenente *a_value* se esiste e **Void** altrimenti. Le linee vuote sono solo indicative e non corrispondono alle linee di codice effettivamente necessarie.

```
feature -- operazioni fondamentali
   get_element (a_value: INTEGER): detachable INT_LINKABLE
-- Ritorna il primo elemento contenente `a_value'
     require
        _____________________________________________________________________
        _____________________________________________________________________
     local
        _____________________________________________________________________
        _____________________________________________________________________
     do
        _____________________________________________________________________
        _____________________________________________________________________
        _____________________________________________________________________
        _____________________________________________________________________
        _____________________________________________________________________
        _____________________________________________________________________
        _____________________________________________________________________
        _____________________________________________________________________
        _____________________________________________________________________
        _____________________________________________________________________
        _____________________________________________________________________
        _____________________________________________________________________
        _____________________________________________________________________
        _____________________________________________________________________
        _____________________________________________________________________
        _____________________________________________________________________
     ensure
        _____________________________________________________________________
        _____________________________________________________________________
end
```

SOLUZIONE:

```
feature -- operazioni fondamentali
   get_element (a_value: INTEGER): detachable INT_LINKABLE
-- Ritorna il primo elemento contenente `a_value'
   local
      current_element, previous_element: like first_element
   do
      from
         current_element := first_element
         previous_element := Void
      invariant
         previous_element /= Void implies previous_element.value /= a_value
      until
         current_element = Void or else current_element.value = a_value
      loop
         previous_element := current_element
         current_element := current_element.next
      end
      Result := current_element
   ensure
      Result /= Void implies Result.value = a_value
      Result = Void implies not has(a_value)
   end
```

Esercizio 3) [10 punti]

Sia data la seguente classe per rappresentare persone

```
class
   PERSONA

create
   make_complete

feature
   id: INTEGER
   nome: STRING
   eta: INTEGER
   stipendio: INTEGER

   make_complete (p_nome: STRING; p_eta: INTEGER; p_stipendio: INTEGER)
      do
         nome := p_nome
         eta := p_eta
         stipendio := p_stipendio
      end

   set_nome (nuovo_nome: STRING)
      require   nuovo_nome /= Void
      do
         nome := nuovo_nome
      ensure
         nome = nuovo_nome
      end

   set_eta (nuova_eta: INTEGER)
      require
         nuova_eta > 0
      do
         eta := nuova_eta
      ensure
         eta = nuova_eta
      end

   set_stipendio (nuovo_stipendio: INTEGER)
      require
         nuovo_stipendio > 0
      do
         stipendio := nuovo_stipendio
      ensure
         stipendio = nuovo_stipendio
      end

   incrementa_stipendio (aumento: INTEGER)
      require
         aumento > 0
      do
         set_stipendio (stipendio + aumento)
      ensure
         stipendio = old stipendio + aumento
      end
```

```
   stampa
      do
         print ("nome: ");
         print (nome);
         print (";  ")
         print ("eta: ");
         print (eta);
         print (";  ")
         print ("stipendio: ");
         print (stipendio);
         print ("%N")
      end

end
```

Si consideri un programma che dichiara la seguente feature:

```
elenco: LINKED_LIST [ PERSONA ]
```

e poi riempie `elenco` con varie istanze di persone in numero non noto.

Si scriva il codice per stampare tutte le persone in `elenco` e per aumentare lo stipendio di ogni persona in `elenco` di una quantità parametrica, utilizzando il meccanismo degli agenti.

```
stampa_elenco
begin
   ____________________________________________________________________________

   ____________________________________________________________________________

   ____________________________________________________________________________

   ____________________________________________________________________________

   ____________________________________________________________________________

   ____________________________________________________________________________

   ____________________________________________________________________________

   ____________________________________________________________________________

   ____________________________________________________________________________

   ____________________________________________________________________________

   ____________________________________________________________________________

end
```

```
aumenta_paga (valore: INTEGER)
begin
    _____
    _____
    _____
    _____
    _____
    _____
    _____
    _____
    _____
    _____
    _____
end
```

SOLUZIONE:

Ricordiamo che la classe generica *LINKED_LIST* [*G*] di Eiffel possiede la feature `do_all` che permette di applicare una procedura, passata come argomento, a tutti gli elementi della lista.

Bisogna quindi utilizzare tale feature ed il meccanismo degli agenti con bersaglio aperto per passare la procedura necessaria ad implementare le funzioni richieste.

La soluzione per stampare tutte le persone in `elenco` è la seguente:

```
stampa_elenco
   do
      elenco.do_all (agent {PERSONA}.stampa)
   end
```

La soluzione per aumentare lo stipendio a tutte le persone in `elenco` è la seguente:

```
aumenta_paga (valore: INTEGER)
   do
      elenco.do_all (agent {PERSONA}.incrementa_stipendio(valore))
   end
```