Prova di esame del 1 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. Ricordiamo che una classe C che è antenato sia di A che di B viene definita "*antenato comune minimo*" di A e B se non esiste un'altra classe X diversa da C che è antenato sia di A che di B ed è anche discendente di C. Si definisce **ereditarietà multipla** il caso di una classe discendente di due classi distinte A e B solo quando l'antenato comune minimo di A e B …
   a. … è unico e corrisponde ad *ANY*
   b. … esiste
   c. … esiste ed è unico
   d. … esiste ed è diverso sia da A che da B

2. 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*

3. Se il nome di una feature *f* compare nella clausola di dichiarazione **create** in una classe *C* allora *f*…
   a. … può essere usata per creare istanze di *C*
   b. … deve necessariamente 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*

4. …
   a. Una classe generica è una classe che ha uno o più parametri generici
   b. Il parametro di una classe generica può essere istanziato sia da una classe non-generica che da una derivazione di una classe generica
   c. Derivazioni diverse di una stessa classe generica sono sempre conformi l'una all'altra
   d. La genericità viene usata per parametrizzare una classe e l'ereditarietà viene usata per specializzare una classe

5. …
   a. Un'assegnazione o un passaggio di argomenti sono polimorfici se la variabile di destinazione (*target variable*) e l'espressione sorgente (*source expression*) hanno tipi diversi
   b. Il polimorfismo è la capacità degli oggetti di cambiare il loro tipo a tempo di esecuzione (*run-time*)
   c. Un'entità è polimorfica se durante l'esecuzione può riferirsi ad oggetti di tipi diversi
   d. Una struttura dati è polimorfica se può contenere riferimenti a oggetti di tipi diversi

SOLUZIONE:

1. Ricordiamo che una classe C che è antenato sia di A che di B viene definita "*antenato comune minimo*" di A e B se non esiste un'altra classe X diversa da C che è antenato sia di A che di B ed è anche discendente di C. Si definisce **ereditarietà multipla** il caso di una classe discendente di due classi distinte A e B solo quando l'antenato comune minimo di A e B …
   a. … è unico e corrisponde ad *ANY*
   b. … esiste
   c. … esiste ed è unico
   d. … esiste ed è diverso sia da A che da B

2. 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*

3. Se il nome di una feature *f* compare nella clausola di dichiarazione **create** in una classe *C* allora *f*…
   a. … può essere usata per creare istanze di *C*
   b. … deve necessariamente 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*

4. …
   a. Una classe generica è una classe che ha uno o più parametri generici
   b. Il parametro di una classe generica può essere istanziato sia da una classe non-generica che da una derivazione di una classe generica
   c. Derivazioni diverse di una stessa classe generica sono sempre conformi l'una all'altra
   d. La genericità viene usata per parametrizzare una classe e l'ereditarietà viene usata per specializzare una classe

5. …
   a. Un'assegnazione o un passaggio di argomenti sono polimorfici se la variabile di destinazione (*target variable*) e l'espressione sorgente (*source expression*) hanno tipi diversi
   b. Il polimorfismo è la capacità degli oggetti di cambiare il loro tipo a tempo di esecuzione (*run-time*)
   c. Un'entità è polimorfica se durante l'esecuzione può riferirsi ad oggetti di tipi diversi
   d. Una struttura dati è polimorfica se può contenere riferimenti a oggetti di tipi diversi

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 *has* (*a_value*) che ritorna *true* se la lista contiene e *false* altrimenti. Le linee vuote sono solo indicative e non corrispondono alle linee di codice effettivamente necessarie.

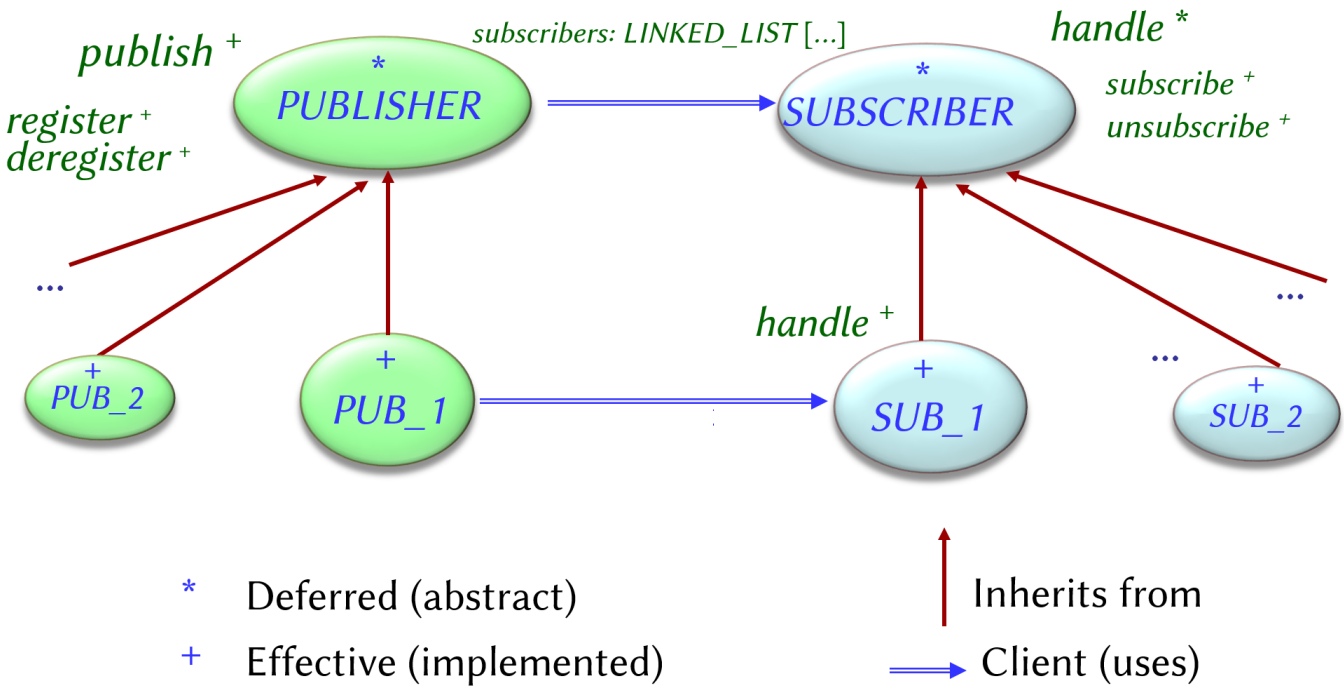
```
feature -- operazioni fondamentali
   has (a_value: INTEGER): BOOLEAN
-- La lista contiene `a_value'?
     require
        _____________________________________________________________________________
        _____________________________________________________________________________
     local
        _____________________________________________________________________________
        _____________________________________________________________________________
     do
        _____________________________________________________________________________
        _____________________________________________________________________________
        _____________________________________________________________________________
        _____________________________________________________________________________
        _____________________________________________________________________________
        _____________________________________________________________________________
        _____________________________________________________________________________
        _____________________________________________________________________________
        _____________________________________________________________________________
        _____________________________________________________________________________
        _____________________________________________________________________________
        _____________________________________________________________________________
        _____________________________________________________________________________
        _____________________________________________________________________________
        _____________________________________________________________________________
        _____________________________________________________________________________
        _____________________________________________________________________________
     ensure
        _____________________________________________________________________________
        _____________________________________________________________________________
end
```

SOLUZIONE:

```
feature -- operazioni fondamentali
   has (a_value: INTEGER): BOOLEAN
      -- La lista contiene `a_value'?
   local
      current_element, previous_element: like first_element
   do
      from
         current_element := first_element
         previous_element := Void
      invariant
         not Result implies
              (previous_element /= Void implies previous_element.value /= a_value)
         Result implies
              (previous_element /= Void implies previous_element.value = a_value)
      until
         (current_element = Void) or Result
      loop
         if current_element.value = a_value then
            Result := True
         end
         previous_element := current_element
         current_element := current_element.next
   end
```

Esercizio 3) [10 punti]

Con riferimento al pattern **Observer** descritto a lezione e ricordato nel diagramma di classi sottostante.



Scrivere l'implementazione delle feature **register** e **deregister** e **publish** per la classe *PUBLISHER* e **subscribe** e **unsubscribe** per la classe SUBSCRIBER.

```
register (s: SUBSCRIBER)
```

end

deregister (s: *SUBSCRIBER*)

end

publish

end

```
subscribe (p: PUBLISHER)
```

```
end
```

```
unsubscribe (p: PUBLISHER)
```

```
end
```

SOLUZIONE:

Assumiamo un contesto ***void safe***. Nella classe *PUBLISHER* :

```
register (s: SUBSCRIBER)
   -- Iscrive s nella lista dei sottoscrittori
require
   not_registered: not subscribers.has(s)
do
   subscribers.extend(s)
ensure
   subscribers.has(s)
end


deregister (s: SUBSCRIBER)
   -- Cancella s dalla lista dei sottoscrittori
require
   registered: subscribers.has(s)
do
   subscribers.remove(s)
ensure
   not subscribers.has(s)
end


publish
   -- Invia a tutti i sottoscrittori nella lista gli argomenti args legati all'evento
do
   from
      subscribers.start
   until
      subscribers.after
   loop
      subscribers.item.handle (args)
      subscribers.forth
   end
end
```

Versione alternativa che usa gli agenti

```
publish
   -- Invia a tutti i sottoscrittori nella lista gli argomenti args legati all'evento
do
      subscribers.do_all (agent {SUBSCRIBER}.handle(args))
end
```

Nella classe *SUBSCRIBER* :

```
subscribe (p: PUBLISHER)
   -- Chiedo a p di inserirmi nella lista dei suoi sottoscrittori
do
   p.register(Current)
end


unsubscribe (p: PUBLISHER)
   -- Chiedo a p di cancellarmi dalla lista dei suoi sottoscrittori
do
   p.deregister(Current)
end
```