Prova di esame del 22 luglio 2016

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. Se la classe *C* esporta la feature *x* verso *NONE* allora …
   a. Nessuna istanza può invocare in modo qualificato la feature *x* di un'istanza di *C*
   b. Una qualunque istanza può invocare in modo qualificato la feature *x* di un'istanza di *C*
   c. Solo un'istanza di *C* può invocare in modo qualificato la feature *x* di un'altra istanza di *C*
   d. Solo la stessa istanza di *C* può invocare in modo non qualificato la feature *x* di sé stessa

2. Sia f una feature *effective* introdotta nella classe *A*, siano *B* e *C* due classi che ereditano direttamente da *A* (cioè non attraverso altre classi) e che non modificano *f* in alcun modo. Sia *X* una classe che eredita direttamente sia da *B* che da *C*. Allora …
   a. … per usare f in X si deve attuare *rename* di almeno una delle due versioni ereditate
   b. … per usare f in X si deve attuare *rename* di entrambe le versioni ereditate
   c. … si può usare f in X anche senza modificare quanto ereditato
   d. … non si può in alcun caso usare f in X

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

4. …
   a. Se la clausola *until* del loop è falsa si esce dal loop
   b. Se la clausola *until* del loop è vuota il programma non compila correttamente
   c. L'invariante del loop deve essere vero anche immediatamente prima della prima iterazione
   d. La variante del loop deve sempre essere un intero positivo

5. Sia f una feature *effective* introdotta nella classe *A*, siano *B* e *C* due classi che ereditano direttamente da *A* (cioè non attraverso altre classi). Assumiamo che *B* attui solo un *redefine* di *f* mentre *C* non modifica *f* in alcun modo. Sia *X* una classe che eredita direttamente sia da *B* che da *C*. Allora si può usare *f* in *X* se…
   a. … almeno una delle due versioni ereditate viene *renamed*
   b. … almeno una delle due versioni ereditate viene *undefined*
   c. … esattamente una delle due versioni ereditate viene *deferred*
   d. … entrambe le versioni ereditate vengono *renamed*

SOLUZIONE:

1. Se la classe *C* esporta la feature *x* verso *NONE* allora …
   a. Nessuna istanza può invocare in modo qualificato la feature *x* di un'istanza di *C*
   b. Una qualunque istanza può invocare in modo qualificato la feature *x* di un'istanza di *C*
   c. Solo un'istanza di *C* può invocare in modo qualificato la feature *x* di un'altra istanza di *C*
   d. Solo la stessa istanza di *C* può invocare in modo non qualificato la feature *x* di sé stessa

2. Sia f una feature *effective* introdotta nella classe *A*, siano *B* e *C* due classi che ereditano direttamente da *A* (cioè non attraverso altre classi) e che non modificano *f* in alcun modo. Sia *X* una classe che eredita direttamente sia da *B* che da *C*. Allora …
   a. … per usare f in X si deve attuare *rename* di almeno una delle due versioni ereditate
   b. … per usare f in X si deve attuare *rename* di entrambe le versioni ereditate
   c. … si può usare f in X anche senza modificare quanto ereditato
   d. … non si può in alcun caso usare f in X

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

4. …
   a. Se la clausola *until* del loop è falsa si esce dal loop
   b. Se la clausola *until* del loop è vuota il programma non compila correttamente
   c. L'invariante del loop deve essere vero anche immediatamente prima della prima iterazione
   d. La variante del loop deve sempre essere un intero positivo

5. Sia f una feature *effective* introdotta nella classe *A*, siano *B* e *C* due classi che ereditano direttamente da *A* (cioè non attraverso altre classi). Assumiamo che *B* attui solo un *redefine* di *f* mentre *C* non modifica *f* in alcun modo. Sia *X* una classe che eredita direttamente sia da *B* che da *C*. Allora si può usare *f* in *X* se…
   a. … almeno una delle due versioni ereditate viene *renamed*
   b. … almeno una delle due versioni ereditate viene *undefined*
   c. … esattamente una delle due versioni ereditate viene *deferred*
   d. … entrambe le versioni ereditate vengono *renamed*

Esercizio 2) [10 punti]

La classe *INT_LINKABLE* modella un elemento di una lista che può rappresentare valori interi. La sua implementazione è la seguente:

```
class
   INT_LINKABLE
create
   make

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

   next : INT_LINKABLE
      -- Il successivo elemento della lista

feature -- operazioni fondamentali
   make (a_value : INTEGER)
      -- crea l'elemento
      do
         value := a_value
      ensure
         value = a_value
      end

   link_to (other: 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 di esso
      require
         other /= Void
      do
         link_to (other.next)
         other.link_to (Current)
      ensure
         other.next = Current
         other.next.next = old other.next
      end

end
```

La classe *INT_LINKED_LIST* modella una lista di interi. Una parte della sua implementazione è fornita qua sotto:

```
class
   INT_LINKED_LIST

feature -- accesso
   first_element: INT_LINKABLE
        -- Il primo elemento della lista

   last_element: INT_LINKABLE
        -- L'ultimo elemento della lista

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

Aggiungere alla classe *INT_LINKED_LIST* una feature *value_follows* (*a_value, target*: *INTEGER*): *BOOLEAN* che restituisce vero se e solo se la lista contiene un elemento di valore *a_value* dopo la prima occorrenza di *target* (non necessariamente subito dopo).

```
feature -- operazioni fondamentali
   value_follows (a_value, target: INTEGER): BOOLEAN
        -- La lista contiene `a_value' dopo la prima occorrenza di `target'?

      require
         ____________________________________________________________________________
         ____________________________________________________________________________
      local
         ____________________________________________________________________________
         ____________________________________________________________________________
      do
         ____________________________________________________________________________
         ____________________________________________________________________________
         ____________________________________________________________________________
         ____________________________________________________________________________
         ____________________________________________________________________________
         ____________________________________________________________________________
         ____________________________________________________________________________
         ____________________________________________________________________________
         ____________________________________________________________________________
         ____________________________________________________________________________
      ensure
         ____________________________________________________________________________
         ____________________________________________________________________________
end
```

SOLUZIONE:

Una possibile soluzione usa una feature separata *get_element*, che restituisce la prima occorrenza di *target*, in modo da ottenere una soluzione più semplice. È ovviamente possibile includere la ricerca della prima occorrenza di *target* direttamente all'interno della feature *value_follows*.

```
feature -- operazioni fondamentali
  get_element (a_value: INTEGER): INT_LINKABLE
       -- Ritorna il primo elemento contenente `a_value', se esiste
     local
       current_element, temp: like first_element
     do
       from
         current_element := first_element;
         temp := Void
       invariant
         Result = Void implies (temp /= Void implies temp.value /= a_value)
       until
         (current_element = Void) or (Result /= Void)
       loop
         if current_element.value = a_value then
           Result := current_element
         end
         temp := current_element
         current_element := temp.next
       end
     ensure
       (Result /= Void) implies Result.value = a_value
     end

feature -- operazioni fondamentali
  value_follows (a_value, target: INTEGER): BOOLEAN
       -- La lista contiene `a_value' dopo la prima occorrenza di `target'?
     local
       current_element, temp: like first_element
     do
       from
         current_element := get_element(target)
         temp := Void
       invariant
         not Result implies (temp /= Void implies temp.value /= a_value)
       until
         (current_element = Void) or Result
       loop
         if current_element.value = a_value then
           Result := True
         end
         temp := current_element
         current_element := current_element.next
       end
     end
```

Esercizio 3) [10 punti]

Completare i contratti (pre-condizioni, post-condizioni, invarianti di classe, invarianti e varianti di *loop*) della classe *CAR* sotto descritta che modella automobili. **Non è ammesso** né cambiare l'interfaccia della classe né le implementazioni fornite. Non c'è relazione tra il numero di righe vuote ed il numero di contratti da scrivere. Non è necessario conoscere altro codice all'infuori di quello fornito.

```
class
   CAR

create
   make

feature {NONE}  -- creazione
   make
      -- crea un'automobile di default
      require
         ____________________________________________________________________________
         ____________________________________________________________________________
      do
         create doors.make — la feature make crea una LINKED_LIST con zero elementi
      ensure
         ____________________________________________________________________________
         ____________________________________________________________________________
      end

feature {ANY}   -- accesso
   is_convertible : BOOLEAN
      -- L'automobile è una cabriolet? Il valore di creazione di default è False.
   doors : LINKED_LIST [CAR_DOOR]
      -- Le porte dell'automobile. Il numero delle porte deve essere 0, 2, o 4. Il valore di
creazione di default è 0.
   colors : COLOR
      -- Il colore dell'automobile. Void se non specificato. Il valore di creazione di
default è Void.

feature {ANY}   -- modifica dell'oggetto
   set_convertible (status : BOOLEAN)
      require
         ____________________________________________________________________________
         ____________________________________________________________________________
      do
         is_convertible := status
      ensure
         ____________________________________________________________________________
         ____________________________________________________________________________
      end

   set_doors (new_doors : ARRAY [CAR_DOOR])
      require
         ____________________________________________________________________________
         ____________________________________________________________________________
      local
         door_index : INTEGER
      do
         doors.wipe_out -- la feature wipe_out cancella tutti gli elementi della LINKED_LIST
         if new_doors /= Void then
            from
               door_index := 1
            invariant
               ____________________________________________________________________
               ____________________________________________________________________
```

```
            until
               door_index > new_doors.count
            loop
               doors.extend (new_doors[door_index])
               door_index := door_index + 1
            variant
               ____________________________________________________________________________
               ____________________________________________________________________________
            end
         end
      ensure
         ____________________________________________________________________________
         ____________________________________________________________________________
      end

   set_color (new_color : COLOR)
      require
         ____________________________________________________________________________
         ____________________________________________________________________________
      do
         color := new_color
      ensure
         ____________________________________________________________________________
         ____________________________________________________________________________
      end

invariant
____________________________________________________________________________________
____________________________________________________________________________________
```

SOLUZIONE:

```
class
   CAR

create
   make

feature {NONE}  -- creazione
   make
      -- crea un'automobile di default
      require
      do
         create doors.make -- la feature make crea una LINKED_LIST con zero elementi
      ensure
         not is_convertible -- assicura la correttezza del valore di creazione dell'attributo
         doors /= Void and then doors.count = 0 -- assicura la correttezza del valore di
creazione dell'attributo
         color = Void -- assicura la correttezza del valore di creazione dell'attributo
      end

feature {ANY}   -- accesso
   is_convertible : BOOLEAN
      -- L'automobile è una cabriolet? Il valore di creazione di default è False.
   doors : LINKED_LIST [CAR_DOOR]
      -- Le porte dell'automobile. Il numero delle porte deve essere 0, 2, o 4. Il valore di
creazione di default è 0.
   colors : COLOR
      -- Il colore dell'automobile. Void se non specificato. Il valore di creazione di
default è Void.
```

```
feature {ANY}  -- modifica dell'oggetto
   set_convertible (status : BOOLEAN)
      require
      do
         is_convertible ≔ status
      ensure
         is_convertible = status
      end

   set_doors (new_doors : ARRAY [CAR_DOOR])
      require   -- Le due versioni in alternativa sono da accoppiare nell'ordine
corrispondente con le versioni in alternativa per le postcondizioni
         v_1: new_doors /= Void implies (new_doors.count = 0 or new_doors.count = 2 or
new_doors.count = 4)
         v_2: new_doors /= Void and then (new_doors.count = 0 or new_doors.count = 2 or
new_doors.count = 4)
      local
         door_index : INTEGER
      do
         doors.wipe_out -- la feature wipe_out cancella tutti gli elementi della LINKED_LIST
         if new_doors /= Void then
            from
               door_index ≔ 1
            invariant
               doors.count = door_index − 1
               1 ≤ door_index
               door_index ≤ new_doors.count + 1
            until
               door_index > new_doors.count
            loop
               doors.extend (new_doors[door_index])
               door_index ≔ door_index + 1
            variant
               new_doors.count − doors.count -- equivalentemente: new_doors.count −
doors.index + 1
            end
         end
      ensure  -- Le due versioni in alternativa sono da accoppiare con le versioni in
alternativa per le precondizioni nell'ordine corrispondente
         v_1: (new_doors = Void implies doors.count = 0) or (new_doors /= Void implies
doors.count = 0 or doors.count = 2 or doors.count = 4)
         v_2: doors.count = 0 or doors.count = 2 or doors.count = 4
      end

   set_color (new_color : COLOR)
      require
      do
         color ≔ new_color
      ensure
         color = new_color
      end

invariant
     doors /= Void and then (doors.count = 0 or doors.count = 2 or doors.count = 4)
   -- la precedente formulazione è migliore dei due contratti separati:
      -- doors /= Void
      -- doors.count = 0 or doors.count = 2 or doors.count = 4
   -- la cui corretta esecuzione dipende dal verificare il primo contratto prima del secondo
```