Prova di esame del 30 luglio 2018

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. 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*

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. Rinominare (*rename*) 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. … portare lo stato della feature a *deferred*
   d. … portare lo stato della feature a *effective*

4. 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. … può essere invocata per le istanze di *C* solo per crearle
   d. … può essere invocata per le istanze di *C*

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 non 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. 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*

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. Rinominare (*rename*) 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. … portare lo stato della feature a *deferred*
   d. … portare lo stato della feature a *effective*

4. 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. … può essere invocata per le istanze di *C* solo per crearle
   d. … può essere invocata per le istanze di *C*

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 non 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]
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 (an_element: INT_LINKABLE)
            - - collega questo elemento con `an_element'
        do
            next := an_element
        ensure
            next = an_element
        end

    link_after (an_element: INT_LINKABLE)
            - - inserisce questo elemento dopo `an_element' conservando quello che c'era dopo di esso
        require
            an_element /= Void
        do
            link_to (an_element.next)
            an_element.link_to (Current)
        ensure
            an_element.next = Current
            an_element.next.next = old an_element.next
        end

end
```

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: INT_LINKABLE
            - - Il primo elemento della lista

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

```
    active_element: INT_LINKABLE
            -- L'elemento corrente della lista

    count: INTEGER
            -- Il numero di elementi della lista

feature -- operazioni fondamentali
    forth
            -- Sposta `active_element' al successivo elemento, se esiste
        do
            if active_element /= Void and then active_element.next /= Void then
                active_element:= active_element.next
            end
        end

    start
            -- Sposta `active_element' al primo elemento, se esiste
        do
            if first_element /= Void then
                active_element:= first_element
            end
        end

    last
            -- Sposta `current_element' all'ultimo elemento, se esiste
        do
            if last_element /= Void then
                active_element:= last_element
            end
        end

    remove_active
-- Rimuove elemento accessibile mediante `active_element'
-- Assegna ad `active_element' il suo successore, se esiste, altrimenti il suo predecessore
        require
            count > 0
-- RESTO DELL'IMPLEMENTAZIONE NON RILEVANTE
        end


invariant
    count >= 0
    last_element /= Void implies last_element.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 (first_element = active_element)
    count > 1 implies (first_element /= last_element) and (first_element /= Void) and (last_element /= Void) and (active_element
    /= Void) and then (first_element.next /= Void)

end
```

Aggiungere alla classe *INT_LINKED_LIST* una feature *remove_first* (*a_value*) che rimuove il primo elemento della lista che contiene *a_value*, completando la parziale implementazione fornita nel seguito. Se tale elemento è l'*active_element* allora questo dev'essere aggiornato a quello che lo seguiva, se esiste, altrimenti a quello che lo precedeva.

```
feature - - operazioni fondamentali
    remove_first (a_value: INTEGER)
- - Rimuove il primo elemento della lista che contiene `a_value'
- - Aggiorna `active_element', se necessario, al suo successore, se esiste, altrimenti al suo predecessore
        require
            count > 0
        local
            current_element, pre_current: INT_LINKABLE
        do
            from
                current_element := first_element
                pre_current := Void
            until
                (current_element = Void) or else (current_element.value = a_value)
            loop
                pre_current := current_element
                current_element := current_element.next
            end
            ____________________________________________________________________________________
            ____________________________________________________________________________________
            ____________________________________________________________________________________
            ____________________________________________________________________________________
            ____________________________________________________________________________________
            ____________________________________________________________________________________
            ____________________________________________________________________________________
            ____________________________________________________________________________________
            ____________________________________________________________________________________
            ____________________________________________________________________________________
            ____________________________________________________________________________________
            ____________________________________________________________________________________
        ensure
            ____________________________________________________________________________________
            ____________________________________________________________________________________
end
```

SOLUZIONE:

```
feature - - operazioni fondamentali
    remove_first (a_value: INTEGER)
- - Rimuove il primo elemento della lista che contiene `a_value'
- - Aggiorna `active_element', se necessario, al suo successore, se esiste, altrimenti al suo predecessore
        require
            count > 0
        local
            current_element, pre_current: INT_LINKABLE
        do
            from
                current_element := first_element
                pre_current := Void
            invariant
                current_element /= Void implies (current_element.value /= a_value implies
                                                 (pre_current.value /= Void implies pre_current.value /= a_value))
            until
                (current_element = Void) or else (current_element.value = a_value)
            loop
                pre_current := current_element
                current_element := current_element.next
            end
            if current_element /= Void then
                - - la lista contiene `a_value'
                if current_element = active_element then
                    remove_active
                else
                    if current_element = first_element then
                        - - `current_element' è il primo elemento della lista
                        first_element := first_element.next
                    elseif current_element = last_element then
                        - - `current_element' è l'ultimo elemento della lista
                        last_element := pre_current
                        last_element.link_to(Void)
                    else
                        - - `current_element' è elemento intermedio della lista
                        pre_current.link_to(current_element.next)
                    end
                    count := count - 1
                end
            end
        ensure
            (count = old count) or (count = old count – 1)
            old first_element.value:= a_value implies first_element = old first_element.next
        end
```

Esercizio 3) [10 punti]
Completare i contratti (pre-condizioni, post-condizioni, invarianti di classe) della classe *COFFEE_MACHINE* sotto descritta che modella il controllo di una macchina per il caffè a cialde. La macchina possiede un *serbatoio* dell'acqua, un *porta-cialde* che viene *aperto* e *chiuso* mediante due pulsanti, e due comandi: *espresso* e *lungo*. Le specifiche di funzionamento sono:

1. Il serbatoio dell'acqua può avere acqua o essere vuoto
2. Per inserire una cialda bisogna aprire il porta-cialde
3. I pulsanti avviano l'erogazione solo se il porta-cialde è chiuso e il serbatoio non è vuoto
4. Al termine dell'erogazione del caffè il porta-cialde viene automaticamente aperto

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.

```
deferred class
    COFFEE_MACHINE

feature {ANY}     stato
    vuoto : BOOLEAN
        - - il serbatoio dell'acqua è vuoto.
    cialda : BOOLEAN
        - - il porta-cialde contiene una cialda.
    aperto : BOOLEAN
        - - il porta-cialde è aperto.

feature {ANY}     - - operazioni sul cronometro
    apri
        - - Apre il porta-cialde
        require
            ____________________________________________________________________
            ____________________________________________________________________
        deferred
        ensure
            ____________________________________________________________________
            ____________________________________________________________________
        end

    chiudi
        - - Chiude il porta-cialde
        require
            ____________________________________________________________________
            ____________________________________________________________________
        deferred
        ensure
            ____________________________________________________________________
            ____________________________________________________________________
        end

    espresso
        - - Fa partire l'erogazione di un caffè espresso
        require
            ____________________________________________________________________
            ____________________________________________________________________
        deferred
        ensure
            ____________________________________________________________________
            ____________________________________________________________________
        end
```

lungo
- - Fa partire l'erogazione di un caffè lungo
require
_____
_____
deferred
ensure
_____
_____
end

invariant
_____
_____
_____

SOLUZIONE:

```
deferred class
    COFFEE_MACHINE

feature {ANY}     stato
    vuoto : BOOLEAN
        - - il serbatoio dell'acqua è vuoto.
    cialda : BOOLEAN
        - - il porta-cialde contiene una cialda.
    aperto : BOOLEAN
        - - il porta-cialde è aperto.

feature {ANY}     - - operazioni
    apri
        - - Apre il porta-cialde
        require
            not aperto
        deferred
        ensure
            aperto
        end

    chiudi
        - - Chiude il porta-cialde
        require
            aperto
        deferred
        ensure
            not aperto
        end

    espresso
        - - Fa partire l'erogazione di un caffè espresso
        require
            not vuoto
            cialda
            not aperto
        deferred
        ensure
            aperto
        end

    lungo
        - - Fa partire l'erogazione di un caffè lungo
        require
            not vuoto
            cialda
            not aperto
        deferred
        ensure
            aperto
        end

invariant
        - - nessun invariante
```