

Prova di esame del 21 giugno 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. ...
 - a. Ad un entità di tipo statico *C* si può assegnare un oggetto di un tipo che è un antenato di *C*
 - b. Ad un entità di tipo statico *C* non si può assegnare un oggetto di un tipo che è un antenato di *C*
 - c. Ad un entità di tipo statico *C* si può assegnare un oggetto di un tipo che è un discendente di *C*
 - d. Ad un entità di tipo statico *C* non si può assegnare un oggetto di un tipo che è un discendente di *C*
2. Se *A* è una classe *deferred* allora:
 - a. la definizione di *A* non può contenere una clausola *create*
 - b. la definizione di *A* può contenere una clausola *create* purché per le entità di tipo statico *A* non si invochi mai l'istruzione *create*
 - c. la definizione di *A* non può contenere feature *effective*
 - d. la definizione di *A* deve contenere almeno una feature *deferred*
3. 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*
4. 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 *deferred*
5. 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*

SOLUZIONE:

1. ...
 - a. Ad un entità di tipo statico *C* si può assegnare un oggetto di un tipo che è un antenato di *C*
 - b. Ad un entità di tipo statico *C* non si può assegnare un oggetto di un tipo che è un antenato di *C*
 - c. Ad un entità di tipo statico *C* si può assegnare un oggetto di un tipo che è un discendente di *C*
 - d. Ad un entità di tipo statico *C* non si può assegnare un oggetto di un tipo che è un discendente di *C*

2. Se *A* è una classe *deferred* allora:
 - a. la definizione di *A* non può contenere una clausola *create*
 - b. la definizione di *A* può contenere una clausola *create* purché per le entità di tipo statico *A* non si invochi mai l'istruzione *create*
 - c. la definizione di *A* non può contenere feature *effective*
 - d. la definizione di *A* deve contenere almeno una feature *deferred*
3. 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*
4. 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 *deferred*
5. 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*

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. 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

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_active` che rimuove l'elemento corrente (cioè quello accessibile mediante `active_element`) completando la parziale implementazione fornita nel seguito

[illegible]

SOLUZIONE:

```

feature -- operazioni fondamentali
  remove_active
    -- Rimuove elemento accessibile mediante `active_element`

  require
    count > 0
  local
    current_element, pre_current: INT_LINKABLE
  do
    if count = 1 then
      first_element := Void
      active_element := Void
      last_element := Void
    else
      from
        current_element := first_element
        pre_current := Void
      until
        (current_element = active_element)
      loop
        pre_current := current_element
        current_element := current_element.next
      end
    -- qui `current_element` coincide con `active_element`
    if current_element = first_element then
      -- `current_element` cioè `active_element` è il primo elemento della lista
      first_element := first_element.next
      active_element := first_element
    elseif current_element = last_element then
      -- `current_element` cioè `active_element` è l'ultimo elemento della lista
      last_element := pre_current
      last_element.link_to(Void)
      active_element := last_element
    else
      -- `current_element` cioè `active_element` è elemento intermedio della lista
      pre_current.link_to(current_element.next)
      active_element := current_element.next
    end
    count := count - 1
  end
ensure
  count = old count - 1
  old active_element = old first_element implies active_element = first_element
  old active_element = old last_element implies active_element = last_element
  old active_element /= old last_element
    implies active_element = old active_element.next
end

```

Esercizio 3) [10 punti]

Completare i contratti (pre-condizioni, post-condizioni, invarianti di classe) della classe *STOPWATCH_CONTROL* sotto descritta che modella un sistema software per controllare un cronometro conta-secondi in modo da rispecchiare la seguente specifica informale:

1. Si può operare sul cronometro attraverso il sistema software mediante i seguenti comandi: *avvio*, *arresto*, *giro*, *riprendi*, *azzerà*. In corrispondenza di tali comandi il sistema software emette analoghi comandi verso il cronometro con la seguente semantica:
 - o *avvio*: fa partire il cronometro dal valore corrente del tempo
 - o *arresto*: ferma il cronometro al valore corrente del tempo
 - o *giro*: blocca il visore del cronometro sul valore corrente del tempo mentre il cronometro continua a cronometrare
 - o *riprendi*: riporta il visore del cronometro a visualizzare il valore corrente del tempo
 - o *azzerà*: riporta il valore del visore al valore iniziale di 0 secondi
2. Il sistema software (e quindi il cronometro) può essere in uno di questi stati:
 - o *fermo*: il cronometro **non** sta misurando il tempo che scorre
 - o *in_moto*: il cronometro sta misurando il tempo che scorre
3. I comandi *avvio* e *azzerà* possono essere dati solo quando il sistema è nello stato *fermo*
4. I comandi *arresto*, *riprendi*, *giro* possono essere dati solo quando il sistema è nello stato *in_moto*

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

STOPWATCH_CONTROL

feature {ANY} accesso

tempo_del_visore : *INTEGER*

-- contiene il valore da visualizzare sul visore del cronometro.

tempo_cronometrato : *INTEGER*

-- contiene il valore corrente del tempo col cronometro.

feature {ANY} stato

fermo : *BOOLEAN*

-- il cronometro non sta misurando il tempo che scorre.

in_moto: *BOOLEAN*

-- il cronometro sta misurando il tempo che scorre.

feature {ANY} -- operazioni sul cronometro

avvio

-- Fa partire il cronometro dal valore corrente del tempo

require

deferred

ensure

end

arresto

-- Ferma il cronometro al valore corrente del tempo

require

deferred

ensure

end

giro

-- Blocca il visore del cronometro sul valore corrente del tempo mentre il cronometro continua a cronometrare

require

deferred

ensure

end

riprendi

-- Riporta il visore del cronometro a visualizzare il valore corrente del tempo

require

deferred

ensure

end

azzera

-- Riporta il visore al valore iniziale di 0 secondi

require

deferred

ensure

end

invariant

SOLUZIONE:

deferred class

STOPWATCH_CONTROL

feature {ANY} accesso

tempo_del_visore : *INTEGER*

-- contiene il valore da visualizzare sul visore del cronometro.

tempo_cronometrato : *INTEGER*

-- contiene il valore corrente del tempo col cronometro.

feature {ANY} stato

fermo : *BOOLEAN*

-- il cronometro non sta misurando il tempo che scorre.

in_moto: *BOOLEAN*

-- il cronometro sta misurando il tempo che scorre.

feature {ANY} -- operazioni sul cronometro

avvio

-- Fa partire il cronometro dal valore corrente del tempo

require

fermo

deferred

ensure

in_moto

end

arresto

-- Ferma il cronometro al valore corrente del tempo

require

in_moto

deferred

ensure

fermo

end

giro

-- Blocca il visore del cronometro sul valore corrente del tempo mentre il cronometro continua a cronometrare

require

in_moto

deferred

ensure

in_moto

tempo_del_visore = **old** tempo_cronometrato

end

riprendi

-- Riporta il visore del cronometro a visualizzare il valore corrente del tempo

require

in_moto

deferred

ensure

in_moto

tempo_del_visore = tempo_cronometrato

end

```
azzera
-- Riporta il visore al valore iniziale di 0 secondi
require
  fermo
deferred
ensure
  fermo
  tempo_del_visore = 0
end
```

invariant

```
fermo XOR in_moto: (fermo and not in_moto) or (not fermo and in_moto)
consistenza tempi in moto: in_moto implies tempo_del_visore <= tempo_cronometrato
consistenza tempi fermo: fermo implies tempo_del_visore = tempo_cronometrato
tempo_cronometrato significativo: tempo_cronometrato >= 0
tempo_visore significativo: tempo_visore >= 0
```