Prova di esame del 22 febbraio 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. …
   a. Una classe *deferred* non può ereditare da una classe *effective*
   b. Una classe *C* non può ereditare da due classi differenti *B1* e *B2*, se sia *B1* che *B2* hanno una stessa classe *A* come antenato comune
   c. Ad un entità di tipo statico *C* si può assegnare un oggetto di un tipo che è un antenato di *C*
   d. In una classe *C* una feature *f* ereditata dalla classe *A* può essere ridefinita soltanto se *f* è *deferred* in A

2. Una query …
   a. … può essere usata come procedura di creazione
   b. … può essere implementata come una *routine*
   c. … può apparire nelle precondizioni e postcondizioni di una qualunque routine
   d. … può apparire nell'invariante di classe

3. …
   a. Una funzione è una query implementata mediante memorizzazione
   b. Un attributo è una query implementata mediante memorizzazione
   c. Una routine è l'implementazione di una feature mediante computazione
   d. Un attributo è l'implementazione di una feature mediante computazione

4. …
   a. Il risultato ritornato da una query è fornito sempre da una funzione
   b. Il risultato ritornato da una query è fornito sempre da un attributo
   c. Il risultato ritornato da una query è fornito da una funzione o da un attributo
   d. Il risultato ritornato da un comando è fornito da una procedura o da un attributo

5. …
   a. Se *C* è una classe *deferred* allora non possono esistere nel programma entità di tipo statico *C*
   b. Se *C* è una classe che usa la *feature* di un'altra classe S allora C è *supplier* (fornitore) di S
   c. La classe *C* può essere contemporaneamente un *supplier* (fornitore) ed un *client* (cliente)
   d. Se il tipo di una variabile è un tipo espanso allora il valore della variabile a tempo di esecuzione (*run-time*) è un oggetto

SOLUZIONE:

1. …
   a. Una classe *deferred* non può ereditare da una classe *effective*
   b. Una classe *C* non può ereditare da due classi differenti *B1* e *B2*, se sia *B1* che *B2* hanno una stessa classe *A* come antenato comune
   c. Ad un entità di tipo statico *C* si può assegnare un oggetto di un tipo che è un antenato di *C*
   d. In una classe *C* una feature *f* ereditata dalla classe *A* può essere ridefinita soltanto se *f* è *deferred* in A

2. Una query …
   a. … può essere usata come procedura di creazione
   b. … può essere implementata come una *routine*
   c. … può apparire nelle precondizioni e postcondizioni di una qualunque routine
   d. … può apparire nell'invariante di classe

3. …
   a. Una funzione è una query implementata mediante memorizzazione
   b. Un attributo è una query implementata mediante memorizzazione
   c. Una routine è l'implementazione di una feature mediante computazione
   d. Un attributo è l'implementazione di una feature mediante computazione

4. …
   a. Il risultato ritornato da una query è fornito sempre da una funzione
   b. Il risultato ritornato da una query è fornito sempre da un attributo
   c. Il risultato ritornato da una query è fornito da una funzione o da un attributo
   d. Il risultato ritornato da un comando è fornito da una procedura o da un attributo

5. …
   a. Se *C* è una classe *deferred* allora non possono esistere nel programma entità di tipo statico *C*
   b. Se *C* è una classe che usa la *feature* di un'altra classe S allora C è *supplier* (fornitore) di S
   c. La classe *C* può essere contemporaneamente un *supplier* (fornitore) ed un *client* (cliente)
   d. Se il tipo di una variabile è un tipo espanso allora il valore della variabile a tempo di esecuzione (*run-time*) è un oggetto

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
      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_all* (*a_value*) che rimuove tutti gli elementi della lista che contengono *a_value*, se esistono, completando la parziale implementazione fornita nel seguito. Se l'*active_element* lo contiene, allora *active_element* dev'essere aggiornato all'elemento che lo segue, se esiste, altrimenti a quello che lo precede.

```
feature - - operazioni fondamentali
   remove_all (a_value: INTEGER)
-- Rimuove tutti gli elementi della lista che contengono `a_value', se esistono
-- 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_all (a_value: INTEGER)
-- Rimuove tutti gli elementi della lista che contengono `a_value', se esistono
-- 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
         if current_element /= Void then
            -- l'elemento `current_element' contiene `a_value'
            from
            until current_element = Void
            loop
               if current_element.value = a_value then
                  if current_element = active_element then
                     remove_active
                  else
                     count := count - 1
                  end
                  if current_element = first_element then
                     first_element := current_element.next
                  end
                  if current_element = last_element then
                     last_element := pre_current
                  end
                  if pre_current /= Void then
                     pre_current.next := current_element.next
                  end
               else
                  -- l'elemento `current_element' NON contiene `a_value'
                  pre_current := current_element
               end
            current_element := current_element.next
            end
         end
      ensure
         (count = old count) or (count = old count - 1)
      end
```

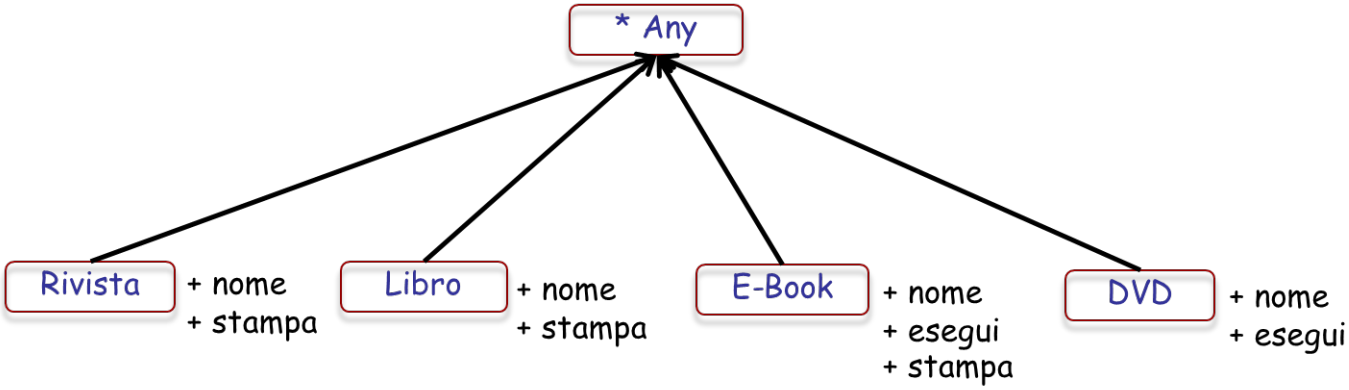
Una versione alternativa che rinuncia ad usare il **loop** proposto dal testo è la seguente:

```
feature - - operazioni fondamentali
   remove_all (a_value: INTEGER)
-- Rimuove tutti gli elementi della lista che contengono `a_value', se esistono
-- 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
         loop
            if current_element.value = a_value then
               if current_element = active_element then
                  remove_active
               else -- La lista ha almeno due elementi
                  if current_element = first_element then
                     first_element := first_element.next
                  elseif current_element = last_element then
                     last_element := pre_current
                     last_element.link_to(Void)
                  else
                     pre_current.link_to(current_element.next)
                  end
                  count := count - 1
               end
            end
            pre_current := current_element
            current_element := current_element.next
         end
      ensure
         (count = old count) or (count = old count − 1)
      end
```

Esercizio 3) [10 punti]

Il software usato da un editore multimediale modella libri, riviste, DVD e libri digitali (*e-book*). Il software ha una specifica classe per ognuno di questi tipi di oggetti. I libri, le riviste e i libri digitali possono essere stampati e quindi il software fornisce un comando **stampa** per ognuna delle relative classi. Un DVD non può essere stampato ma può essere eseguito e quindi il software fornisce un comando **esegui** per la relativa classe. Anche un libro digitale può essere eseguito e quindi anche per la relativa classe il software fornisce un comando **esegui**. Ogni istanza di un tipo di oggetto ha un suo nome e quindi il software fornisce un attributo **nome** di tipo **stringa**.

L'attuale struttura di ereditarietà delle classi è la seguente



Ridefinire tale struttura (anche introducendo nuove classi, se necessario), motivandone i vantaggi e disegnando il nuovo diagramma di ereditarietà che mostra quali classi e *feature* sono *deferred* (*), *effective* (+) o *redefined* (++).

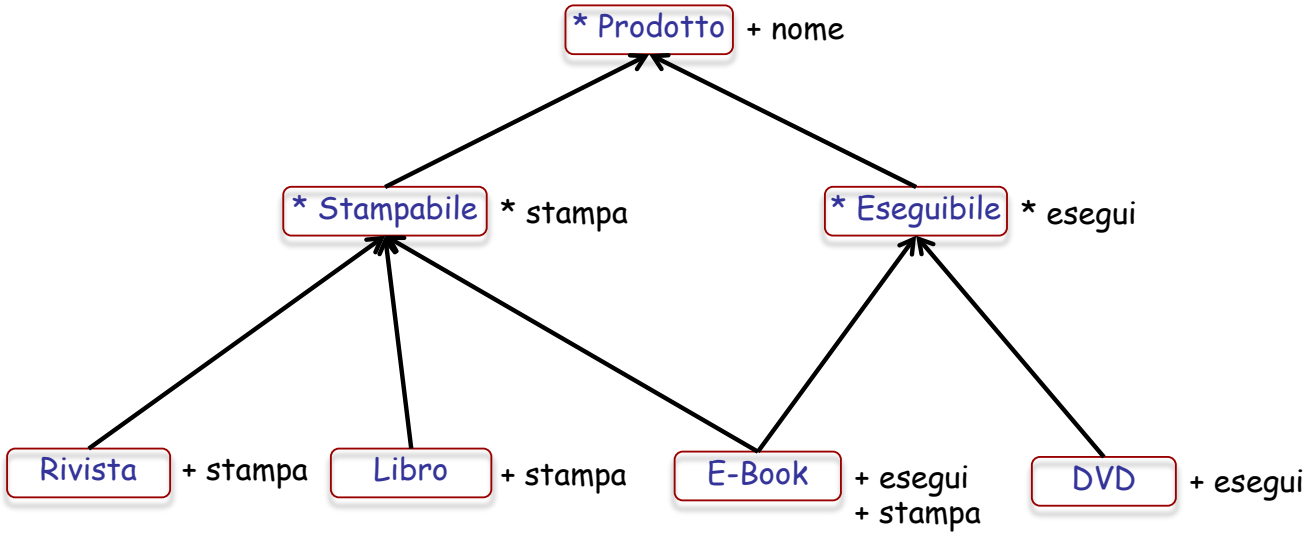
Non è necessario scrivere alcun codice.

SOLUZIONE:

L'osservazione fondamentale derivata dall'analisi delle feature presenti nelle classi è che vi sono due insiemi di prodotti che esibiscono feature comuni ma distinte. In particolare, vi è un insieme di prodotti che possono essere stampati e un insieme di prodotti che possono essere eseguiti.

Limitandosi a questa osservazione si può allora introdurre una superclasse *deferred* per rappresentare ciò che è comune a tutti i prodotti e due sue sottoclassi (anch'esse *deferred*) per rappresentare ciò che è comune a tutti i prodotti stampabili e a tutti i prodotti eseguibili.

Sulla base di questa osservazione si ottiene una prima ristrutturazione mostrata nel seguito



Un ulteriore ristrutturazione può essere prodotta assumendo la condivisione di caratteristiche tra il libro digitale e il libro, da un lato, e tra il libro digitale e il dvd, dall'altro. Si ottiene in tal caso il nuovo diagramma qui sotto, in cui la classe E-Book viene definita come sottoclasse che eredita da entrambe le classi Libro e DVD