

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. Un comando è una feature che non può restituire un risultato
 - b. Il risultato ritornato da una query è fornito sempre da una funzione
 - c. Il risultato ritornato da un comando è fornito da una procedura o da un attributo
 - d. Una query è una feature che può restituire un risultato oppure non restituirlo

2. Se la classe *C* **non** esporta l'attributo *x* verso una classe *T* allora ...
 - a. Un'istanza di *T* può assegnare il valore di *x* in un'istanza di *C* solo in modo diretto
 - b. Un'istanza di *T* non può assegnare in alcun modo il valore di *x* in un'istanza di *C*
 - c. Un'istanza di *T* non può leggere il valore di *x* direttamente da un'istanza di *C*
 - d. Un'istanza di *T* può ottenere il valore di *x* da un'istanza di *C* solo mediante un'apposita feature

3. Se la classe *C* esporta la feature *x* **solo** verso la classe *A* allora ...
 - a. Nessuna istanza di *A* può invocare in modo qualificato la feature *x* di un'istanza di *C*
 - b. Una qualunque istanza di *A* può invocare in modo qualificato la feature *x* di un'istanza di *C*
 - c. Solo un'istanza di *A* può invocare in modo non qualificato la feature *x* di un'istanza di *C*
 - d. Nessuna istanza di *C* può invocare in modo qualificato la feature *x* di un'istanza di *C*

4. La ridefinizione di una *feature* che è una funzione può...
 - a. ... cambiare il contratto
 - b. ... cambiare l'implementazione
 - c. ... cambiare la lista dei tipi degli argomenti
 - d. ... cambiare il tipo del risultato

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. Una struttura dati è polimorfica se può contenere riferimenti a oggetti di tipi diversi
 - d. Un'entità è polimorfica se durante l'esecuzione può riferirsi ad oggetti di tipi diversi

SOLUZIONE:

1. ...
 - a. Un comando è una feature che non può restituire un risultato
 - b. Il risultato ritornato da una query è fornito sempre da una funzione
 - c. Il risultato ritornato da un comando è fornito da una procedura o da un attributo
 - d. Una query è una feature che può restituire un risultato oppure non restituirlo

2. Se la classe *C* **non** esporta l'attributo *x* verso una classe *T* allora ...
 - a. Un'istanza di *T* può assegnare il valore di *x* in un'istanza di *C* solo in modo diretto
 - b. Un'istanza di *T* non può assegnare in alcun modo il valore di *x* in un'istanza di *C*
 - c. Un'istanza di *T* non può leggere il valore di *x* direttamente da un'istanza di *C*
 - d. Un'istanza di *T* può ottenere il valore di *x* da un'istanza di *C* solo mediante un'apposita feature

3. Se la classe *C* esporta la feature *x* **solo** verso la classe *A* allora ...
 - a. Nessuna istanza di *A* può invocare in modo qualificato la feature *x* di un'istanza di *C*
 - b. Una qualunque istanza di *A* può invocare in modo qualificato la feature *x* di un'istanza di *C*
 - c. Solo un'istanza di *A* può invocare in modo non qualificato la feature *x* di un'istanza di *C*
 - d. Nessuna istanza di *C* può invocare in modo qualificato la feature *x* di un'istanza di *C*

4. La ridefinizione di una *feature* che è una funzione può...
 - a. ... cambiare il contratto
 - b. ... cambiare l'implementazione
 - c. ... cambiare la lista dei tipi degli argomenti
 - d. ... cambiare il tipo del risultato

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. Una struttura dati è polimorfica se può contenere riferimenti a oggetti di tipi diversi
 - d. Un'entità è polimorfica se durante l'esecuzione può riferirsi ad oggetti di tipi diversi

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 (i : INTEGER)
    -- crea l'elemento
  do
    value := i
  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

feature -- operazioni fondamentali

has (*a_value*: INTEGER): BOOLEAN
 -- La lista contiene `a_value`?

local

temp, *pre_temp*: INT_LINKABLE

do

from

temp := *first_element*;

pre_temp := Void

invariant

not Result implies (*pre_temp* /= Void **implies** *pre_temp.value* /= *a_value*)

until

(*temp* = Void) or **Result**

loop

if *temp.value* = *a_value* **then**

Result := **True**

end

pre_temp := *temp*

temp := *temp.next*

end

end

get_item (*a_value*: INTEGER): INT_LINKABLE

-- Ritorna l'elemento contenente `a_value`, se esiste

local

temp, *pre_temp*: INT_LINKABLE

do

from

temp := *first_element*;

pre_temp := Void

invariant

Result = Void **implies** (*pre_temp* /= Void **implies** *pre_temp.value* /= *a_value*)

until

(*temp* = Void) or (**Result** /= Void)

loop

if *temp.value* = *a_value* **then**

Result := *temp*

end

pre_temp := *temp*

temp := *temp.next*

end

ensure

(**Result** /= Void) **implies** **Result.value** = *a_value*

end

invariant

count >= 0

last_element /= Void **implies** *last_element.next* = Void

count = 0 **implies** (*first_element* = *last_element*) **and** (*first_element* = Void)

count = 1 **implies** (*first_element* = *last_element*) **and** (*first_element* /= Void)

count > 1 **implies** (*first_element* /= *last_element*) **and** (*first_element* /= Void) **and** (*last_element* /= Void) **and** **then** (*first_element.next* /= Void)

end

Aggiungere alla classe `INT_LINKED_LIST` una feature `insert_before` (`new`, `target: INTEGER`) che inserisce un nuovo elemento di valore `new` prima dell'elemento di valore `target`, se quest'ultimo è presente. Altrimenti il nuovo elemento viene inserito all'inizio della lista.

Non è ammesso modificare la classe `INT_LINKABLE`.

feature -- operazioni fondamentali

`insert_before` (`new`, `target: INTEGER`)

-- Inserisce elemento con valore `new` prima di elemento con valore `target`, se quest'ultimo esiste

-- Altrimenti inserisce elemento con valore `new` all'inizio della lista

local

do

ensure

end

SOLUZIONE:

Qua sotto una prima versione che usa la feature già esistente *has()*:

feature -- operazioni fondamentali

insert_before (*new*, *target*: *INTEGER*)

-- Inserisce elemento con valore *new* prima di elemento con valore *target*, se quest'ultimo esiste

-- Altrimenti inserisce elemento con valore *new* all'inizio della lista

local

new_item: *INT_LINKABLE*

previous_item: *INT_LINKABLE*

do

create *new_item*.*make* (*new*)

if *has* (*target*) **then**

if (*target* = *first_element*.*value*) **then**

new_item.*link_to* (*first_element*)

first_element := *new_item*

else -- la lista ha almeno 2 elementi e contiene *target* dalla seconda posizione in avanti

from

previous_item := *first_element*

until

previous_item.*next* = *target*

loop

previous_item := *prev_item*.*next*

end

new_item.*insert_after* (*temp*)

end

else -- la lista non contiene *target*

if *count* = 0 **then**

first_element := *new_item*

last_element := *first_element*

else

new_item.*link_to* (*first_element*)

first_element := *new_item*

end

end

count := *count* + 1

end

ensure

count = **old** *count* + 1

not (**old** *has*(*target*)) **implies** *first_element*.*value* = *new*

old *has*(*target*) **implies** *get_item*(*new*).*next*.*value* = *target*

end

Un'implementazione alternativa più efficiente si ottiene senza usare le feature già esistenti:

feature -- operazioni fondamentali

insert_before (*new*, *target*: *INTEGER*)

-- Inserisce elemento con valore *new* prima di elemento con valore *target*, se quest'ultimo esiste

-- Altrimenti inserisce elemento con valore *new* all'inizio della lista

local

new_item, *temp*: *INT_LINKABLE*

do

create *new_item*.*make* (*new*)

if *count* = 0 **then**

first_element := *new_item*

last_element := *first_element*

else

if (*target* = *first_element*.*value*) **then**

new_item.*link_to* (*first_element*)

first_element := *new_item*

else -- la lista contiene almeno un elemento e il primo elemento non ha il *target*

from

temp := *first_element*

until

(*temp*.*next* = **Void**) **or else** (*temp*.*next*.*value* = *target*)

loop

temp := *temp*.*next*

end

if *temp*.*next* /= **Void** **then**

new_item.*insert_after* (*temp*)

else -- la lista non contiene *target*

new_item.*link_to* (*first_element*)

first_element := *new_item*

end

end

end

count := *count* + 1

ensure

count = **old** *count* + 1

not (**old** *has*(*target*)) **implies** *first_element*.*value* = *new*

old *has*(*target*) **implies** *get_item*(*new*).*next*.*value* = *target*

end

Esercizio 3) [10 punti]

Il software di una stazione metereologica gestisce un sistema (implementato dalla classe *DATI_TEMPO*) che fornisce le misurazioni dei valori di temperatura, umidità e pressione e tre diversi sistemi per la loro visualizzazione, implementati da tre classi *VISUALIZZA_DATI_CORRENTI*, *VISUALIZZA_DATI_STATISTICI*, *VISUALIZZA_DATI_PREVISIONE*, che visualizzano rispettivamente il valore appena misurato, alcune statistiche sui valori misurati, una previsione del valore futuro.

La classe *root* dell'attuale implementazione è la seguente:

```

class
  STAZIONE_METEREOLOGICA

create
  gestisci_misurazioni

feature {NONE}
  fornitore_dati : DATI_TEMPO
    -- sistema che fornisce i valori correnti.

  visore_dati_correnti : VISUALIZZA_DATI_CORRENTI
    -- sistema che visualizza i valori correnti
  visore_dati_statistici : VISUALIZZA_DATI_STATISTICI
    -- sistema che visualizza le statistiche sui valori misurati
  visore_dati_previsione : VISUALIZZA_DATI_PREVISIONE
    -- sistema che visualizza una previsione del valore futuro

  gestisci_misurazioni
    -- esegue tutto il sistema
  do
    create fornitore_dati
    create visore_dati_correnti
    create visore_dati_statistici
    create visore_dati_previsione
  from
  until
    True
  loop
    if fornitore_dati.ha_nuovi_dati then
      visore_dati_correnti.aggiorna(fornitore_dati.valore_temperatura,
                                   fornitore_dati.valore_umidita,
                                   fornitore_dati.valore_pressione)
      visore_dati_statistici.aggiorna(fornitore_dati.valore_temperatura,
                                      fornitore_dati.valore_umidita,
                                      fornitore_dati.valore_pressione)
      visore_dati_previsione.aggiorna(fornitore_dati.valore_temperatura,
                                       fornitore_dati.valore_umidita,
                                       fornitore_dati.valore_pressione)
    end
  end
end
end
end
end

```

La classe che realizza il sistema che fornisce le misurazioni ha l'interfaccia descritta qua sotto. Le attuali implementazioni delle features non sono rilevanti.

class*DATI_TEMPO***feature** {*STAZIONE_METEREOLOGICA*, *VISUALIZZA_DATI_CORRENTI*, *VISUALIZZA_DATI_STATISTICI*, *VISUALIZZA_DATI_PREVISIONE*}*valore_temperatura* : *REAL*

-- ritorna il valore della temperatura ottenuto da un sensore esterno

valore_umidita : *REAL*

-- ritorna il valore dell'umidità ottenuto da un sensore esterno

valore_pressione : *REAL*

-- ritorna il valore della pressione ottenuto da un sensore esterno

ha_nuovi_dati : *BOOLEAN*

-- qualcuno dei valori misurati e' cambiato

end

Le classi che implementano i sistemi di visualizzazione sono descritte qua sotto.

class*VISUALIZZA_DATI_CORRENTI***feature** {*STAZIONE_METEREOLOGICA*}*aggiorna* : (*temperatura*: *REAL*; *umidita*: *REAL*; *pressione*: *REAL*)

-- visualizza i valori secondo le caratteristiche di questa classe

do**print** ("%N La temperatura attuale è: ")**print** (*temperatura*)**print** ("%N L'umidità attuale è: ")**print** (*umidita*)**print** ("%N La pressione attuale è: ")**print** (*pressione*)**end****end****class***VISUALIZZA_DATI_STATISTICI***feature** {*NONE*}*temperatura_precedente*: *REAL*

-- precedente valore ricevuto per la temperatura

umidita_precedente: *REAL*

-- precedente valore ricevuto per l'umidita'

pressione_precedente: *REAL*

-- precedente valore ricevuto per la pressione

media (*precedente*: *REAL*; *attuale*: *REAL*): *REAL*

-- fa la media dei due valori passati

do**Result** := (*precedente* + *attuale*) / 2**end**

feature {*STAZIONE_METEREOLOGICA*}

aggiorna : (*temperatura*: *REAL*; *umidita*: *REAL*; *pressione*: *REAL*)
 -- visualizza i valori secondo le caratteristiche di questa classe

do

print ("%N La temperatura media è: ")

print (*media* (*temperatura_precedente*, *temperatura*))

temperatura_precedente := *temperatura*

print ("%N L'umidità media è: ")

print (*media* (*umidita_precedente*, *umidita*))

umidita_precedente := *umidita*

print ("%N La pressione media è: ")

print (*media* (*pressione_precedente*, *pressione*))

pressione_precedente := *pressione*

end

end

class

VISUALIZZA_DATI_PREVISIONE

feature {*NONE*}

temperatura_precedente: *REAL*

-- precedente valore ricevuto per la temperatura

umidita_precedente: *REAL*

-- precedente valore ricevuto per l'umidita'

pressione_precedente: *REAL*

-- precedente valore ricevuto per la pressione

media (*precedente*: *REAL*; *attuale*: *REAL*): *REAL*

-- fa la media dei due valori passati

do

Result := (*precedente* + *attuale*) / 2

end

previsione (*precedente*: *REAL*; *attuale*: *REAL*): *REAL*

-- fa la previsione del valore futuro sulla base della media e dei valori passati

local

delta: *REAL*

do

delta := (*attuale* - *precedente*)

Result := *media* (*precedente* + *attuale*) + *delta*

end

SOLUZIONE:

L'attuale implementazione soffre di due problemi.

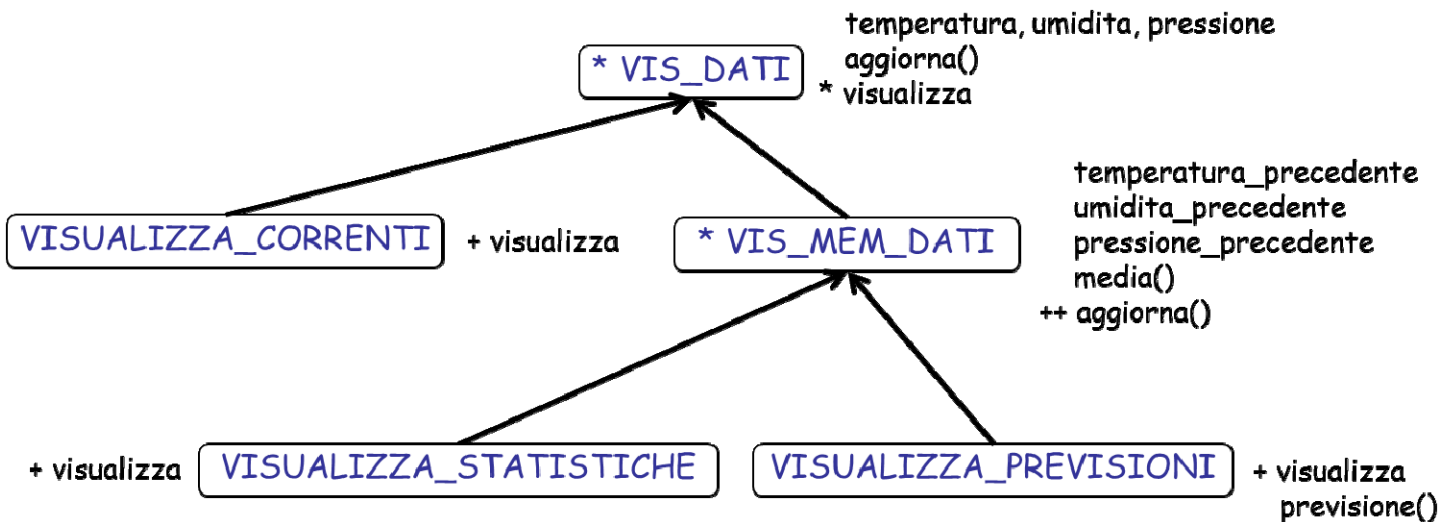
Un primo problema è che le varie classi che fanno la visualizzazione non sfruttano in alcun modo i loro comportamenti comuni.

Per risolvere tale problema i comportamenti comuni vanno raggruppati in superclassi *deferred* da cui le specifiche classi che fanno la visualizzazione ereditano. Si veda più sotto l'implementazione

Un secondo è che la classe *root* è accoppiata alle specifiche classi che fanno la visualizzazione e quindi l'inserimento di una nuova modalità di visualizzazione o la cancellazione di una modalità esistente si riflettono sulla necessità di cambiare la classe *root*.

Per risolvere tale problema è necessario ricorrere all'uso del *design pattern* di *publisher/subscriber*.

Per risolvere il primo problema bisogna introdurre una strutturazione, nelle classi di visualizzazione, che metta a fattor comune i comportamenti condivisi. Si noti, nel diagramma di ereditarietà delle classi raffigurato qua sotto, l'introduzione di due classi *deferred* che realizzano appunto quest'obiettivo. Si ricordi che '*' indica classi o feature *deferred*, '+' l'implementazione di feature *deferred*, '++' il cambiamento di un'implementazione ereditata.



Le due classi *deferred* *VIS_DATI* e *VIS_MEM_DATI* definiscono i comportamenti comuni, rispettivamente, per la visualizzazione dei dati e per la gestione dei dati necessari per statistiche e previsioni. E' stata separata la funzione di ricezione dei nuovi dati (*aggiorna*) da quella di visualizzazione dei dati correnti (*visualizza*), poiché tali due funzioni cambiano in modi differenti con i vari comportamenti di visualizzazione.

deferred class

VIS_DATI

feature {NONE}

temperatura : REAL

-- contiene il valore della temperatura ricevuto

umidita : REAL

-- contiene il valore dell'umidità ricevuto

pressione : REAL

-- contiene il valore della pressione ricevuto

visualizza

-- visualizza i valori secondo le caratteristiche di questa classe

deferred

end

feature {*STAZIONE_METEREOLOGICA*}

aggiorna (*temperatura_ricevuta*: *REAL*; *umidita_ricevuta*: *REAL*; *pressione_ricevuta*: *REAL*)

-- registra i valori ricevuti e ne attiva la visualizzazione

do

temperatura := *temperatura_ricevuta*

umidita := *umidita_ricevuta*

pressione := *pressione_ricevuta*

visualizza

end

end

deferred class

VIS_MEM_DATI

inherit

VIS_DATI

redefine

aggiorna

end

feature {*NONE*}

temperatura_precedente : *REAL*

-- contiene il valore della temperatura precedentemente ricevuto

umidita_precedente : *REAL*

-- contiene il valore dell'umidita precedentemente ricevuto

pressione_precedente : *REAL*

-- contiene il valore della pressione precedentemente ricevuto

media (*precedente* : *REAL*; *attuale* : *REAL*) : *REAL*

-- ritorna la media dei due valori passati

do

Result := (*precedente* + *attuale*) / 2

end

feature {*STAZIONE_METEREOLOGICA*}

aggiorna (*temperatura_ricevuta*: *REAL*; *umidita_ricevuta*: *REAL*; *pressione_ricevuta*: *REAL*)

-- registra i valori ricevuti e ne attiva la visualizzazione

do

Precursor (*temperatura_ricevuta*, *umidita_ricevuta*, *pressione_ricevuta*)

temperatura_precedente := *temperatura*

umidita_precedente := *umidita*

pressione_precedente := *pressione*

end

end

A questo punto le tre classi *VISUALIZZA_DATI_CORRENTI*, *VISUALIZZA_DATI_STATISTICI*, *VISUALIZZA_DATI_PREVISIONE*, implementano soltanto gli aspetti univoci per ognuna di esse.

```
class
  VISUALIZZA_DATI_CORRENTI

inherit
  VIS_DATI

feature {NONE}

  visualizza
    -- visualizza i valori secondo le caratteristiche di questa classe
  do
    print ("%N La temperatura attuale è: ")
    print (temperatura)
    print ("%N L'umidità attuale è: ")
    print (umidita)
    print ("%N La pressione attuale è: ")
    print (pressione)
  end

end
```

```
class
  VISUALIZZA_DATI_STATISTICI

inherit
  VIS_MEM_DATI

feature {NONE}

  visualizza
    -- visualizza i valori secondo le caratteristiche di questa classe
  do
    print ("%N La temperatura media è: ")
    print (media (temperatura_precedente, temperatura))
    print ("%N L'umidità media è: ")
    print (media (umidita_precedente, umidita))
    print ("%N La pressione media è: ")
    print (media (pressione_precedente, pressione))
  end

end
```

class*VISUALIZZA_DATI_PREVISIONE***inherit***VIS_MEM_DATI***feature** {*NONE*}*previsione (precedente: REAL; attuale: REAL): REAL**-- fa la previsione del valore futuro sulla base della media e dei valori passati***local***delta: REAL***do***delta := (attuale – precedente)***Result** := *media (precedente + attuale) + delta***end***visualizza**-- visualizza i valori secondo le caratteristiche di questa classe***do****print** (“%N La temperatura prevista è: ”)**print** (*previsione (temperatura_precedente, temperatura)*)**print** (“%N L’umidità prevista è: ”)**print** (*previsione (umidita_precedente, umidita)*)**print** (“%N La pressione prevista è: ”)**print** (*previsione (pressione_precedente, pressione)*)**end****end**