Compito di *Fondamenti della Programmazione: Metodi Evoluti* - A.A. 2012-13
Prova di esame del 24 settembre 2013

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 può esistere solo a tempo di esecuzione (*run-time*)
   b. Una funzione non deve modificare alcun oggetto
   c. Una query è sempre una funzione
   d. Una procedura può ritornare valori come risultato della sua esecuzione

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

3. …
   a. Un loop deve sempre definire un invariante altrimenti il programma non compila correttamente
   b. La clausola *from* del loop non può essere vuota altrimenti il programma non compila correttamente
   c. L'invariante del loop deve essere vero anche immediatamente dopo l'ultima iterazione
   d. Se la clausola *until* del loop è falsa si esce dal loop

4. …
   a. La variante del loop deve incrementare ad ogni iterazione del 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. …
   a. Il parametro di una classe generica può essere istanziato sia da una classe non-generica che da una derivazione di una classe generica
   b. Se *C* è una classe *deferred* allora non possono esistere nel programma entità di tipo statico *C*
   c. La genericità viene usata per parametrizzare una classe e l'ereditarietà viene usata per specializzare una classe
   d. Se *C* è una classe che usa la *feature* di un'altra classe S allora C è *supplier* (fornitore) di S

SOLUZIONE:

1. …
   a. Una classe può esistere solo a tempo di esecuzione (*run-time*)
   b. Una funzione non deve modificare alcun oggetto
   c. Una query è sempre una funzione
   d. Una procedura può ritornare valori come risultato della sua esecuzione

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

3. …
   a. Un loop deve sempre definire un invariante altrimenti il programma non compila correttamente
   b. La clausola *from* del loop non può essere vuota altrimenti il programma non compila correttamente
   c. L'invariante del loop deve essere vero anche immediatamente dopo l'ultima iterazione
   d. Se la clausola *until* del loop è falsa si esce dal loop

4. …
   a. La variante del loop deve incrementare ad ogni iterazione del 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

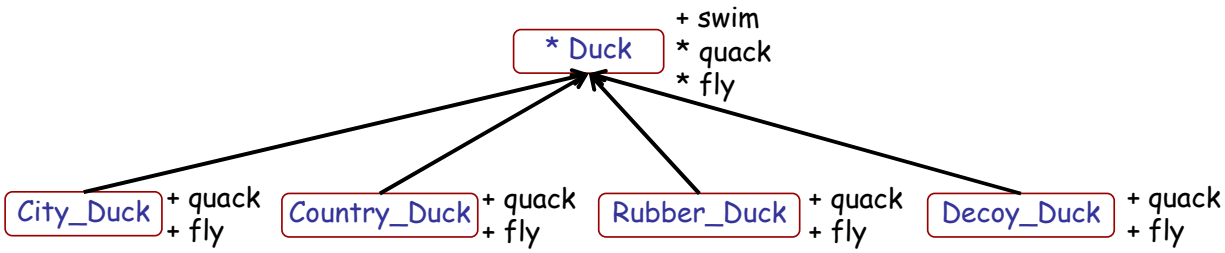
5. …
   a. Il parametro di una classe generica può essere istanziato sia da una classe non-generica che da una derivazione di una classe generica
   b. Se *C* è una classe *deferred* allora non possono esistere nel programma entità di tipo statico *C*
   c. La genericità viene usata per parametrizzare una classe e l'ereditarietà viene usata per specializzare una classe
   d. Se *C* è una classe che usa la *feature* di un'altra classe S allora C è *supplier* (fornitore) di S

Esercizio 2) [10 punti]
Il software usato da un gioco per la caccia alle anatre modella anatre di città (*city_duck*), anatre di campagna (*country_duck*), anatroccoli per vasca da bagno (*rubber_duck*) e anatre-esca di legno (*decoy_duck*). Il software ha una specifica classe per ognuno di questi tipi di oggetti. Ognuno di questi tipi di oggetti è in grado di far qua-qua (*quack*) e di nuotare (*swim*). Quando si chiede di far qua-qua agli anatroccoli per vasca da bagno questi devono però squittire mentre le anatre-esca non possono emettere alcun suono. Sia le anatre di città che quelle di campagna sanno volare (*fly*) mentre né gli anatroccoli per vasca da bagno né le anatre-esca possono davvero volare.

Nell'attuale implementazione ognuna di queste classi è definita come sottoclasse di una classe astratta *Duck* e non vengono sfruttati al massimo i vantaggi derivanti dall'utilizzo di un approccio object-oriented.

Il diagramma sotto disegnato presenta l'attuale struttura di ereditarietà delle classi mostrando:
1. la struttura di ereditarietà delle classi (cioè le relazioni superclasse – sottoclasse)
2. quali classi e *feature* sono *deferred* (*), *effective* (+) o *redefined* (++).

Si noti che il metodo *fly* è definito nella classe astratta per consentire a qualunque tipo di anatra di rispondere correttamente all'invocazione del metodo stesso senza doversi interrogare sul suo tipo per evitare errori a run-time.



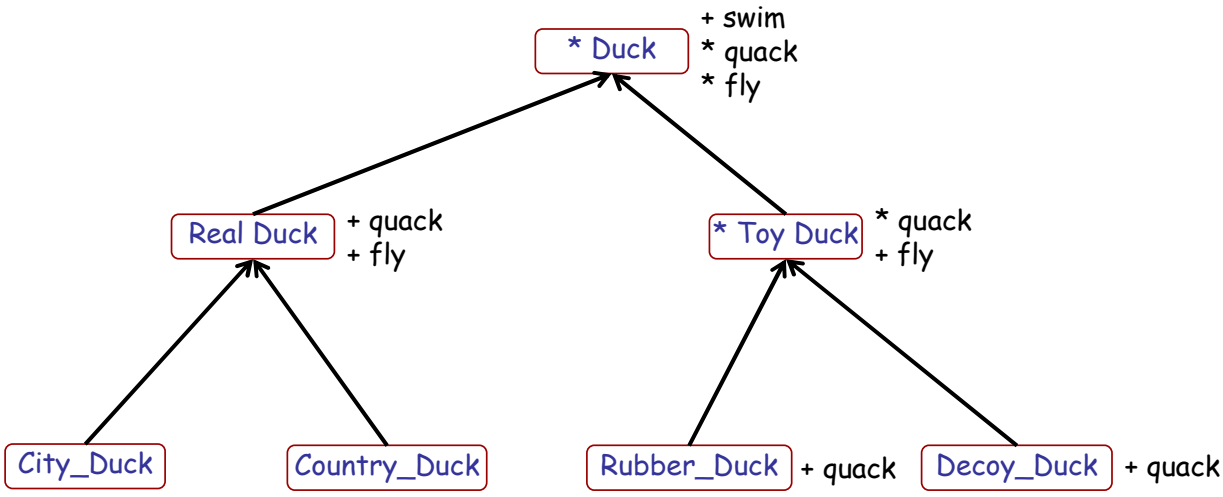
Disegnare un nuovo diagramma che definisca una nuova struttura di ereditarietà che consenta di:
- ottenere un migliore riuso del codice già scritto
- ottenere una migliore manutenibilità del codice immaginando che in futuro verranno introdotte altre varianti di anatre che sono in grado di far qua-qua, nuotare e volare come le anatre di città e quelle di campagna

Non è necessario scrivere alcun codice.

SOLUZIONE:

La soluzione introduce due sottoclassi per mettere a fattor comune ciò che è condiviso da più tipi di anatre. Nello specifico:
(1) la possibilità reale di volare e di far quack per le anatre di città e quelle di campagna – rappresentata dalla classe *Real_Duck* (che implementa una volta sola per tutte le sue sottoclassi i metodi *fly* e *quack* e rende possibile l'aggiunta di nuovi tipi di anatre in grado di volare e far quack senza dover aggiungere codice),
(2) l'incapacità di volare per gli anatroccoli da vasca da bagno e per le anatre-esca – rappresentata dalla classe *Toy_Duck* (che implementa una volta sola per tutte le sue sottoclassi il metodo *fly* in modo da impedire alle anatre di questo tipo di volare ma consente alle sue sottoclassi di implementare il metodo *quack* in modo da consentire loro di emettere il verso appropriato: uno squittio per gli anatroccoli per vasca da bagno, nessun verso per le anatre-esca).

Si ottiene quindi il diagramma qua sotto:


* Duck
+ swim
* quack
* fly
Real Duck
+ quack
+ fly
* Toy Duck
* quack
+ fly
City_Duck
Country_Duck
Rubber_Duck
+ quack
Decoy_Duck
+ quack

Esercizio 3) [10 punti]
Completare i contratti (pre-condizioni, post-condizioni, invarianti di classe) della classe *CRUISE_CONTROL* sotto descritta che modella un sistema per il controllo della velocità di crociera di un veicolo in modo da rispecchiare la seguente specifica informale:

1. Il sistema di controllo della velocità di crociera (cruise control = CC) è in uno dei seguenti stati: *spento*, *acceso*, *abilitato* (cioè il controllo della velocità è attivo), *disabilitato* (cioè il controllo della velocità non è attivo)
2. La velocità può essere assegnata o riassegnata in qualunque momento CC è acceso. Lo spegnimento di CC causa la de-assegnazione della velocita'
3. In stato *abilitato* la pressione sui freni (*brake*) causa la disabilitazione di CC. In stato *disabilitato* il comando riprendi velocità (*recover_speed*) causa l'abilitazione con velocità pari all'ultima velocità assegnata.
4. In stato *abilitato* la pressione dell'acceleratore (*increase_gas*) causa la disabilitazione di CC. Il rilascio dell'acceleratore (*decrease_gas*) causa l'abilitazione con velocità pari all'ultima velocità assegnata.

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
    CRUISE_CONTROL

feature {ANY}-- accesso
    cruise_speed : INTEGER
        -- La velocità di crociera che è stata definita. Vale 0 quando non è stata assegnata.
    current_speed : INTEGER
        -- La velocità corrente.

feature {ANY}-- stato
    is_CC_on : BOOLEAN
        -- Il sistema CC è acceso?
    is_CC_enabled : BOOLEAN
        -- Il sistema CC è abilitato?
        deferred
        ensure
            Result = (cruise_speed /= 0)
        end

feature {ANY}-- eventi
    brake : BOOLEAN
        -- Sono stati premuti i freni?
    increase_gas : BOOLEAN
        -- E' stato premuto l'acceleratore?
    decrease_gas : BOOLEAN
        -- E' stato rilasciato l'acceleratore?

feature {ANY}-- operazioni fondamentali
    set_speed
        -- Assegna la velocità di crociera
        require
            _____________________________________________________________________
            _____________________________________________________________________
        deferred
        ensure
            _____________________________________________________________________
            _____________________________________________________________________
        end
```

recover_speed
-- Riprendi la velocità di crociera
require
_____
_____
deferred
ensure
_____
_____
end

enable_CC
-- Abilita il sistema CC
require
_____
_____
deferred
ensure
_____
_____
end

disable_CC
-- Disabilita il sistema CC
require
_____
_____
deferred
ensure
_____
_____
end

switch_on
-- Accende il sistema CC
require
_____
_____
deferred
ensure
_____
_____
end

switch_off
-- Spegne il sistema CC
require
_____
_____
deferred
ensure
_____
_____
end

invariant
_____
_____

SOLUZIONE:

```
deferred class
    CRUISE_CONTROL

feature {ANY}-- accesso
    cruise_speed : INTEGER
        -- La velocità di crociera che è stata definita. Vale 0 quando non è stata assegnata.
    current_speed : INTEGER
        -- La velocità corrente.

feature {ANY}-- stato
    is_CC_on : BOOLEAN
        -- Il sistema CC è acceso?
    is_CC_enabled : BOOLEAN
        -- Il sistema CC è abilitato?
        deferred
        ensure
            Result = (cruise_speed /= 0)
        end

feature {ANY}-- eventi
    brake : BOOLEAN
        -- Sono stati premuti i freni?
    increase_gas : BOOLEAN
        -- E' stato premuto l'acceleratore?
    decrease_gas : BOOLEAN
        -- E' stato rilasciato l'acceleratore?

feature {ANY}-- operazioni fondamentali
    set_speed
        -- Assegna la velocità di crociera
        require
            is_CC_on
        deferred
        ensure
            cruise_speed /= 0
        end

    recover_speed
        -- Riprendi la velocità di crociera
        require
            is_CC_on
            not is_CC_enabled
            cruise_speed /= 0
        deferred
        ensure
            is_CC_enabled
            current_speed = cruise_speed
        end
```

```
    enable_CC
        -- Abilita il sistema CC
        require
            is_CC_on
            not is_CC_enabled
            cruise_speed /= 0
        deferred
        ensure
            is_CC_enabled
            current_speed = cruise_speed
        end

    disable_CC
        -- Disabilita il sistema CC
        require
            is_CC_enabled
        deferred
        ensure
            not is_CC_enabled
        end

    switch_on
        -- Accende il sistema CC
        require
            not is_CC_on
        deferred
        ensure
            is_CC_on
        end

    switch_off
        -- Spegne il sistema CC
        require
            is_CC_on
        deferred
        ensure
            not is_CC_on
            cruise_speed = 0
            not is_CC_enabled
        end

invariant
    current_speed ≥ 0
    cruise_speed ≥ 0
    decrease_gas implies (is_CC_enabled and current_speed = cruise_speed)
    is_CC_enabled implies not (brake or increase_gas)
-- all'ultimo contratto potevano essere sostituiti i due contratti:
    -- brake implies not is_CC_enabled
    -- increase_gas implies not is_CC_enabled
```

N.B.
I contratti evidenziati in giallo non erano necessari per la versione modificata dell'esercizio che è stata effettivamente svolta durante la prova d'esame e non sono quindi stati considerati ai fini della valutazione.