Laurea in Informatica – Programmazione ad oggetti – Appello d'Esame 26.1.2015

Nome..... Cognome..... Matricola.....

È VIETATO l'uso di QUALSIASI oggetto diverso dalla penna. Dove previsto scrivere la risposta CHIARAMENTE nell'apposito spazio.

Quesito 1

Si consideri il seguente modello di realtà concernente l'app FitApp© per la registrazione di allenamenti sportivi.

(A) Definire la seguente gerarchia di classi.

1. Definire una classe base polimorfa astratta `Allenamento` i cui oggetti rappresentano un allenamento memorizzabile in FitApp. Ogni `Allenamento` è caratterizzato dalla durata in minuti. La classe è astratta in quanto prevede i seguenti **metodi virtuali puri**:
   - un metodo di "clonazione": `Allenamento* clone()`.
   - un metodo `int calorie()` con il seguente contratto puro: `a->calorie()` ritorna il numero di calorie consumate durante l'allenamento `*a`.
2. Definire una classe concreta `Ciclismo` derivata da `Allenamento` i cui oggetti rappresentano un allenamento di ciclismo. Ogni oggetto `Ciclismo` è caratterizzato dalla distanza totale percorsa in Km e dal numero di Km percorsi in salita. La classe `Ciclismo` implementa i metodi virtuali puri di `Allenamento` come segue:
   - per ogni puntatore `p` a `Ciclismo`, `p->clone()` ritorna un puntatore ad un oggetto `Ciclismo` che è una copia di `*p`.
   - per ogni puntatore `p` a `Ciclismo`, `p->calorie()` ritorna il numero di calorie dato dalla formula $200K^2/D + 100S$, dove K è la distanza totale percorsa in Km nell'allenamento `*p`, D è la durata in minuti dell'allenamento `*p`, S è il numero di Km percorsi in salita nell'allenamento `*p`.
3. Definire una classe concreta `Corsa` derivata da `Allenamento` i cui oggetti rappresentano un allenamento di corsa. Ogni oggetto `Corsa` è caratterizzato dalla distanza totale percorsa in Km e dal numero di Km percorsi su fondo sterrato. La classe `Corsa` implementa i metodi virtuali puri di `Allenamento` come segue:
   - per ogni puntatore `p` a `Corsa`, `p->clone()` ritorna un puntatore ad un oggetto `Corsa` che è una copia di `*p`.
   - per ogni puntatore `p` a `Corsa`, `p->calorie()` ritorna il numero di calorie dato dalla formula $600K^2/D$, dove K è la distanza totale percorsa in Km nell'allenamento `*p`, D è la durata in minuti dell'allenamento `*p`.
4. Definire una classe concreta `Nuoto` derivata da `Allenamento` i cui oggetti rappresentano un allenamento di nuoto. Ogni oggetto `Nuoto` è caratterizzato dal numero di vasche nuotate e dallo stile di nuoto (si assumano due soli stili possibili: libero e rana). La classe `Nuoto` implementa i metodi virtuali puri di `Allenamento` come segue:
   - per ogni puntatore `p` a `Nuoto`, `p->clone()` ritorna un puntatore ad un oggetto `Nuoto` che è una copia di `*p`.
   - per ogni puntatore `p` a `Nuoto`, se N è il numero di vasche nuotate nell'allenamento `*p`, allora `p->calorie()` ritorna $35N$ calorie.

(B) Definire una classe `FitApp` i cui oggetti rappresentano una installazione dell'app. Un oggetto di `FitApp` è quindi caratterizzato da un vector di oggetti di tipo `const Allenamento*` che contiene tutti gli allenamenti memorizzati dall'app. La classe `FitApp` rende disponibili i seguenti metodi:

1. Un metodo `vector<Ciclismo> salita(double)` con il seguente comportamento: una invocazione `a.salita(perc)` ritorna un vector di oggetti `Ciclismo` contenente tutti e soli gli allenamenti di ciclismo memorizzati nella FitApp `a` la cui percentuale di Km percorsi in salita sulla distanza totale percorsa è $>$ `perc`.
2. Un metodo `vector<Allenamento*> calorie(int)` con il seguente comportamento: una invocazione `a.calorie(x)` ritorna un vector contenente dei puntatori a **copie** di tutti e soli gli allenamenti memorizzati nella FitApp `a` che : (i) hanno comportato un consumo di calorie $>$ `x`; **ed inoltre** (ii) se sono allenamenti di nuoto allora sono a stile libero.
3. Un metodo `void insert(Corsa*)` con il seguente comportamento: una invocazione `a.insert(p)` inserisce il nuovo allenamento di corsa `p` nella FitApp `a` solamente se l'allenamento di corsa `*p` stabilisce il nuovo record (cioè il nuovo massimo assoluto) di numero di Km percorsi su fondo sterrato tra tutti gli allenamenti di corsa già memorizzati in `a`; se `*p` non stabilisce il nuovo record allora viene sollevata l'eccezione "`NoInsert`" di tipo `string`.
4. **(Opzionale)** Usando **obbligatoriamente una lambda espressione**, definire un metodo `int conta(double)` con il seguente comportamento: una invocazione `a.conta(limite)` ritorna il numero di allenamenti memorizzati nella FitApp `a` con durata in minuti $>$ `limite`.