Esercizio Funzione

Si assumano le seguenti specifiche (**NON È CODICE DA SCRIVERE**) di una generica libreria grafica.

A. `Component` è una classe astratta i cui oggetti, detti componenti, hanno una rappresentazione grafica che può essere mostrata sul display. La classe `Component` rende disponibile un metodo virtuale e costante `bool hasFocus()` con il seguente comportamento: una invocazione `c.hasFocus()` ritorna `true` se la componente `c` detiene il focus del display, altrimenti ritorna `false`.

B. `Container` è una sottoclasse concreta di `Component` i cui oggetti sono componenti detti contenitori che possono contenere altre componenti. La classe `Container` rende disponibile un metodo virtuale `void setHeight(double)` con il seguente comportamento: una invocazione `c.setHeight(d)` imposta l'altezza `d` in cm per il contenitore `c`. La classe `Container` rende inoltre disponibile un metodo virtuale `void setWidth(double)` con il seguente comportamento: una invocazione `c.setWidth(d)` imposta la larghezza `d` in cm per il contenitore `c`.

C. `Window` è una sottoclasse di `Container` i cui oggetti rappresentano generiche finestre. La classe `Window` rende disponibile un metodo `void hide()` con il seguente comportamento: una invocazione `w.hide()` nasconde la finestra `w` se `w` è visibile sul display, altrimenti, cioè se `w` è nascosta, lancia un oggetto eccezione di un tipo `Hidden` dotato di costruttore di default. La classe `Window` rende inoltre disponibile un metodo virtuale e costante `bool hasMenu()` con il seguente comportamento: una invocazione `w.hasMenu()` ritorna `true` se alla finestra `w` è stato impostato un menu, altrimenti ritorna `false`. La classe `Window` fornisce l'overriding dei metodi virtuali `void setHeight(double)` e `void setWidth(double)` specializzandoli per la classe `Window`.

D. `Frame` è una sottoclasse di `Window` i cui oggetti rappresentano finestre grafiche con titolo e bordo (dette frame). La classe `Frame` rende disponibile un metodo virtuale `void setTitle(string)` con il seguente comportamento: una invocazione `f.setTitle(s)` imposta alla stringa `s` il titolo del frame `f`. La classe `Frame` fornisce l'overriding dei metodi virtuali `void setHeight(double)` e `void setWidth(double)` specializzandoli per la classe `Frame`.

Definire una funzione `void fun(const Component&, vector<const Window*>&)` con il seguente comportamento: in ogni invocazione `fun(c,v)`:

– se `c` è un frame a cui è stato impostato un menu allora imposta alla stringa "`menu`" il titolo del frame `c` e inserisce un puntatore a `c` nel vettore `v`;
– se `c` è una generica finestra visibile sul display allora nasconde la finestra `c`;
– se `c` è un contenitore che detiene il focus del display allora imposta altezza e larghezza di `c` entrambe a 3cm;
– in tutti gli altri casi, esce normalmente senza provocare alcun effetto, in particolare quindi senza lanciare alcuna eccezione.

SOLUZIONE

```
void fun(const Component& c, vector<const Window*>& w){
        NB: Capisco che non ci vorrà il "for" dal fatto che:
        (1) non c'è scritto nel testo "per ogni puntatore contenuto" etc. etc.
        (2) si capisce vedendo dalla firma e da come sono scritti i punti
        che mi "basta" avere il parametro Component& "c" come frame, etc. etc.

        // SOLUZIONE "NORMALE" - TOLGO TUTTI I CONST (ma lo deve rimettere..)

        Frame* f = dynamic_cast<Frame*>(const_cast<Component*>)(&c));
        // Non mettiamo *c perché il parametro non è un iteratore
        // ed è un riferimento

        if(f && f->hasMenu()) {
                f->setTitle("menu");
                v.push_back(dynamic_cast<const Frame*>(*f));
        }

        // ALTRIMENTI - tengo tutto const

        const Frame* f = dynamic_cast<const Frame*>(&c);
        // Non mettiamo *c perché il parametro non è un iteratore
        // ed è un riferimento

        if(f && f->hasMenu()) {
                f->setTitle("menu");
                v.push_back(*f);
        }

        try{
                //Finestra visibile sul display, nascondi
                const Window* w = dynamic_cast<const Window*>(&c));
                w->hide();
        }
        catch(Hidden h){
                std::error("Finestra nascosta, quindi non visibile!");
        }

        const Component* co = dynamic_cast<const Component*>(&c);
        // Non mettiamo *c perché il parametro non è un iteratore
        // ed è un riferimento

        if(c && c->hasFocus()) {
                c->setHeigth(3);
                c->setWidth(3);
        }
}
```