

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

Esercizio 1 (NB: Per comodità di correzione, definire tutti i metodi inline)

Si consideri il seguente modello di realtà concernente il gestore P² di piani di telefonia mobile.

(A) Definire una classe di eccezioni *Anomalia* che rende disponibile un costruttore *Anomalia(char)* con il seguente comportamento:

Anomalia('v') costruisce una eccezione che rappresenta una anomalia relativa al traffico voce; *Anomalia('d')* costruisce una eccezione che rappresenta una anomalia relativa al traffico dati; *Anomalia('s')* costruisce una eccezione che rappresenta una anomalia relativa al traffico sms; *Anomalia('c')* costruisce una eccezione che rappresenta una anomalia relativa all'addebito di un costo.

(B) Definire la seguente gerarchia di classi.

1. Definire una classe base astratta *Scheda* i cui oggetti rappresentano una scheda SIM ricaricabile gestita da P². Ogni *Scheda* è caratterizzata da: un numero telefono che si suppone essere univoco e rappresentato mediante una stringa (non sono richiesti controlli di univocità e consistenza della stringa); il credito residuo in euro, che non può quindi essere negativo; il costo in € di spedizione di un sms.

- La classe *Scheda* è astratta in quanto prevede i seguenti metodi virtuali puri:
 - un metodo `void telefonata(unsigned int)` con il seguente contratto puro: `s->telefonata(n)` contabilizza sulla scheda `*s` una telefonata di durata `n` secondi; se lo stato del piano della scheda `*s` non permette una telefonata di `n` secondi allora viene sollevata una eccezione *Anomalia('v')*.
 - un metodo `void connessione(double)` con il seguente contratto puro: `s->connessione(k)` contabilizza sulla scheda `*s` una connessione di `k` MB; se lo stato del piano della scheda `*s` non permette una connessione di `k` MB allora viene sollevata una eccezione *Anomalia('d')*.
- La classe *Scheda* rende disponibile un metodo `void messaggioSMS(unsigned int)` con il seguente comportamento: per tutti i sottotipi di *Scheda*, `s->messaggioSMS(k)` contabilizza sulla scheda `*s` l'invio di `k` messaggi sms scalando dal credito disponibile su `*s` il costo di spedizione di un sms per ognuno dei `k` messaggi sms; se il costo dell'invio dei `k` messaggi sms eccede il credito disponibile su `*s` allora viene sollevata una eccezione *Anomalia('s')*.

2. Definire una classe concreta *Mensile* derivata da *Scheda* i cui oggetti rappresentano una scheda SIM ricaricabile con piano di tariffazione ad abbonamento mensile per telefonate e connessioni. Ogni oggetto *Mensile* è quindi caratterizzato da: una soglia mensile di secondi di telefonate incluse; il traffico totale di secondi di telefonate effettuate nel mese corrente; una soglia mensile di MB di connessioni incluse; il traffico totale di MB di connessioni effettuate nel mese corrente; il costo mensile di abbonamento. La classe *Mensile* rende disponibile un costruttore di default che costruisce un piano ad abbonamento mensile con le seguenti caratteristiche: numero di telefono "0", credito residuo 0 €, costo di spedizione di un sms 0.1 €, soglia mensile di telefonate 60000 secondi, soglia mensile di connessioni 2096 MB.

La classe *Mensile* implementa i metodi virtuali puri di *Scheda* come segue:

- per ogni puntatore `m` a *Mensile*, `m->telefonata(n)` aggiunge `n` secondi al traffico totale delle telefonate del mese corrente della scheda `*m`, e se questa telefonata provoca il superamento della soglia mensile di telefonate della scheda `*m`, allora viene sollevata una eccezione *Anomalia('v')*.
- per ogni puntatore `m` a *Mensile*, `m->connessione(k)` aggiunge `k` MB al traffico totale delle connessioni del mese corrente della scheda `*m`, e se questa connessione provoca il superamento della soglia mensile di connessioni della scheda `*m`, allora viene sollevata una eccezione *Anomalia('d')*.

3. Definire una classe concreta *Consumo* derivata da *Scheda* i cui oggetti rappresentano una scheda SIM ricaricabile con piano di tariffazione a consumo per telefonate e connessioni. Ogni oggetto *Consumo* è quindi caratterizzato da: il costo in € di 1 secondo di telefonata; il costo in € di 1 MB di connessione. La classe *Consumo* rende disponibile un costruttore di default che costruisce un piano a consumo di default con le seguenti caratteristiche: numero di telefono "0", credito residuo 0 €, costo di spedizione di un sms 0.2 €, costo di 1 secondo di telefonata 0.01 €, costo di 1 MB di connessione 0.1 €.

La classe *Consumo* implementa i metodi virtuali puri di *Scheda* come segue:

- per ogni puntatore `c` a *Consumo*, `c->telefonata(n)` addebita `n` secondi di telefonata sul credito residuo della scheda `*c`, e se questo addebito provoca il superamento del credito residuo allora viene sollevata una eccezione *Anomalia('v')*.
- per ogni puntatore `c` a *Mensile*, `m->connessione(k)` addebita `k` MB di connessione sul credito residuo della scheda `*c`, e se questo addebito provoca il superamento del credito residuo allora viene sollevata una eccezione *Anomalia('d')*.

(C) Definire una classe *P2* i cui oggetti rappresentano un centro di gestione di schede SIM ricaricabili di P². Un oggetto di *P2* è caratterizzato da un contenitore di elementi di tipo *Scheda** che memorizza tutte le schede SIM gestite dal centro. La classe *P2* definisce i seguenti metodi:

1. Un metodo `Consumo* cambioPiano(std::string)` con il seguente comportamento: in una invocazione `p2.cambioPiano("num")`, se il numero di telefono "num" è gestito dal centro `p2` e corrisponde ad una scheda SIM ricaricabile con piano di tariffazione ad abbonamento mensile allora cambia il piano di tariffazione tramutandolo in un piano di tariffazione a consumo di default che mantiene lo stesso numero "num" e preserva il credito residuo; se l'invocazione `p2.cambioPiano("num")` effettivamente provoca il cambio di piano allora viene ritornato un puntatore al nuovo piano di tariffazione a consumo, altrimenti viene ritornato il puntatore nullo.

- Un metodo `vector<Consumo> rimuoviConsumoZero()` con il seguente comportamento: una invocazione `p2.rimuoviConsumoZero()` rimuove dalle schede SIM gestite dal centro `p2` tutte le schede con piano di tariffazione a consumo che hanno un credito residuo pari a 0 €, e restituisce una vettore contenente una copia di tutte le schede con piano di tariffazione a consumo rimosse.
- Un metodo `double contabilizza()` con il seguente comportamento: una invocazione `p2.contabilizza()` provoca la contabilizzazione in tutte le schede SIM gestite dal centro `p2` con credito residuo positivo di una telefonata di 1 secondo, di una connessione di 1 MB e dell'invio di 1 sms, e restituisce il guadagno ottenuto dal centro `p2` mediante questa contabilizzazione (cioè la differenza del totale dei crediti residui di tutte le schede prima e dopo questa contabilizzazione).

Esercizio 2

```
class A {
protected:
    virtual void j() { cout<<" A::j "; }
public:
    virtual void g() const { cout <<" A::g "; }
    virtual void f() { cout <<" A::f "; g(); j(); }
    void m() { cout <<" A::m "; g(); j(); }
    virtual void k() { cout <<" A::k "; j(); m(); }
    virtual A* n() { cout <<" A::n "; return this; }
};
```

```
class C: public A {
private:
    void j() { cout <<" C::j "; }
public:
    virtual void g() { cout <<" C::g "; }
    void m() { cout <<" C::m "; g(); j(); }
    void k() const { cout <<" C::k "; k(); }
};
```

```
class B: public A {
public:
    virtual void g() const override { cout <<" B::g "; }
    virtual void m() { cout <<" B::m "; g(); j(); }
    void k() { cout <<" B::k "; A::n(); }
    A* n() override { cout <<" B::n "; return this; }
};
```

```
class D: public B {
protected:
    void j() { cout <<" D::j "; }
public:
    B* n() final { cout <<" D::n "; return this; }
    void m() { cout <<" D::m "; g(); j(); }
};
```

```
A* p1 = new D(); A* p2 = new B(); A* p3 = new C(); B* p4 = new D(); const A* p5 = new C();
```

Le precedenti definizioni compilano correttamente. Per ognuna delle seguenti istruzioni scrivere nell'apposito spazio:

- NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- ERRORE RUN-TIME** se l'istruzione compila correttamente ma la sua esecuzione provoca un errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a run-time allora si scriva la stampa che l'esecuzione produce in output su `cout`; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p1->g(); .....
p1->k(); .....
p2->f(); .....
p2->m(); .....
p3->k(); .....
p3->f(); .....
p4->m(); .....
p4->k(); .....
p5->g(); .....
(p3->n())->m(); .....
(p3->n())->n()->g(); .....
(p4->n())->m(); .....
(p5->n())->g(); .....
(dynamic_cast<B*>(p1))->m(); .....
(static_cast<C*>(p2))->k(); .....
(static_cast<B*>(p3->n()))->g(); .....
```