Programmazione 2
Appello d'Esame – 7/12/2007

Nome..... Cognome.....

Matricola..... Laurea in.....

Non si possono consultare appunti e libri. Dove previsto scrivere CHIARAMENTE la risposta nell'apposito spazio.

Esercizio 1

Si consideri la seguente realtà concernente i biglietti del treno. Come ben noto, un biglietto per un viaggio in treno può essere di prima o seconda classe.

1. Definire una classe `Biglietto` i cui oggetti rappresentano un biglietto per un viaggio in treno. Ogni `Biglietto` è caratterizzato dalla distanza chilometrica del viaggio. La classe `Biglietto` dichiara un metodo virtuale puro `double prezzo()` che prevede il seguente contratto: una invocazione `b.prezzo()` ritorna il prezzo del biglietto `b`. Per tutti i biglietti, il prezzo base al km è fissato in 0.1 €.

2. Definire una classe `BigliettoPrimaClasse` derivata da `Biglietto` i cui oggetti rappresentano un biglietto per un viaggio di prima classe. Il prezzo di un biglietto di prima classe con distanza inferiore a 100 km è dato dal prezzo base (prezzo base al km moltiplicato per la distanza chilometrica) aumentato del 30%, altrimenti l'aumento del prezzo base è del 20%. `BigliettoPrimaClasse` implementa quindi `prezzo()` ritornando il prezzo di un dato biglietto di prima classe.

3. Definire una classe `BigliettoSecondaClasse` derivata da `Biglietto` i cui oggetti rappresentano un biglietto per un viaggio di seconda classe. Un biglietto di seconda classe può essere con prenotazione oppure senza (la prenotazione garantisce il posto a sedere). Per tutti i biglietti di seconda classe, il costo della prenotazione è fissato in 5 €. Il prezzo di un biglietto di seconda classe è dato dal prezzo base (prezzo base al km moltiplicato per la distanza chilometrica) più l'eventuale costo della prenotazione.

4. Definire una classe `BigliettoSmart` i cui oggetti rappresentano dei puntatori smart a `Biglietto`. La classe `BigliettoSmart` dovrà essere dotata dell'interfaccia pubblica necessaria per lo sviluppo della successiva classe `Treno`.

5. Definire una classe `TrenoPieno` i cui oggetti rappresentano delle eccezioni che segnalano che non vi sono posti disponibili in un dato treno. Una eccezione di `TrenoPieno` è caratterizzata dalla classe (1ˆ o 2ˆ) in cui non vi sono più posti disponibili.

6. Definire una classe `Treno` i cui oggetti rappresentano un certo viaggio in treno (la semplificazione prevede che non vi siano fermate intermedie). Ogni oggetto `Treno` è quindi caratterizzato dall'insieme dei biglietti venduti per quel viaggio in treno, e tale insieme deve essere rappresentato mediante un vector `venduti` di puntatori smart `BigliettoSmart`. Un oggetto `Treno` è inoltre caratterizzato dal numero massimo di posti disponibili per biglietti di prima classe e dal numero massimo di posti disponibili per biglietti di seconda classe con prenotazione.

   Devono essere disponibili nella classe `Treno` le seguenti funzionalità:

   - Un metodo `int* bigliettiVenduti()` con il seguente comportamento: una invocazione `t.bigliettiVenduti()` ritorna un array `ar` di 3 interi tale che:
     - `ar[0]` memorizza il numero di biglietti venduti di prima classe per il treno `t`;
     - `ar[1]` memorizza il numero di biglietti venduti di seconda classe con prenotazione per il treno `t`;
     - `ar[2]` memorizza il numero di biglietti venduti di seconda classe senza prenotazione per il treno `t`.
   - Un metodo `void vendiBiglietto(const Biglietto&)` con il seguente comportamento: una chiamata `t.vendiBiglietto(b)` aggiunge `b` tra i biglietti venduti per il treno `t` quando possibile, altrimenti solleva una opportuna eccezione di `TrenoPieno`. Più dettagliatamente:
     - Se `b` è un biglietto di prima classe e vi sono ancora posti di prima classe disponibili in `t` allora viene aggiunto al vector `venduti` un puntatore smart a `b`; se invece non vi sono posti di prima classe disponibili viene sollevata una eccezione `TrenoPieno` in prima classe.
     - Se `b` è un biglietto di seconda classe con prenotazione e vi sono ancora posti di seconda classe con prenotazione disponibili in `t` allora viene aggiunto al vector `venduti` un puntatore smart a `b`; se invece non vi sono posti di seconda classe con prenotazione disponibili viene sollevata una eccezione `TrenoPieno` in seconda classe.
     - Se `b` è un biglietto di seconda classe senza prenotazione allora viene sempre aggiunto al vector `venduti` un puntatore smart a `b`.
   - Un metodo `double incasso()` con il seguente comportamento: una chiamata `t.incasso()` ritorna l'incasso totale per tutti i biglietti sinora venduti per il treno `t`.

Esercizio 2

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

class B: public A {
private:
  virtual void h() {cout <<" B::h ";}
public:
  virtual void g() {cout <<" B::g ";}
  void m() {cout <<" B::m "; g(); h();}
  void k() {cout <<" B::k "; g(); h(); m();}
  B* n() {cout <<" B::n "; return this;}
};

B* b = new B(); A* a = new B();
```

La compilazione delle precedenti definizioni non provoca errori (con gli opportuni `include` e `using`). Si supponga che ognuno dei seguenti frammenti sia il codice di un `main()` che può accedere alle precedenti definizioni. Si scriva nell'apposito spazio contiguo:

- **NON COMPILA** quando tale `main()` non compila;
- **ERRORE RUN-TIME** quando tale `main()` compila ma l'esecuzione provoca un errore run-time;
- la stampa che produce in output su `cout` nel caso in cui tale `main()` compili ed esegua senza errori; se non provoca alcuna stampa si scriva **NESSUNA STAMPA**.

`b->f();`	
`b->m();`	
`b->k();`	
`a->f();`	
`a->m();`	
`a->k();`	
`(b->n())->g();`	
`(b->n())->n()->g();`	
`(a->n())->g();`	
`(a->n())->m();`	