Laboratorio di Programmazione
Edizione Diurna - Turni A, B, C

Appello del 22 settembre 2014

Avvertenza: Nello svolgimento dell'elaborato è possibile usare qualunque classe delle librerie standard di Java. Non è invece ammesso l'uso delle classi del package `prog` allegato al libro di testo del Prof. Pighizzini e impiegato nella prima parte del corso.

Tema d'esame

Lo scopo è realizzare un'applicazione che permette di gestire un carrello in cui possono essere inseriti dei prodotti da acquistare. Per ogni prodotto viene specificato il nome, il prezzo unitario e la quantità acquistata. Su alcuni prodotti è applicato uno sconto. Le classi da realizzare sono le seguenti (dettagli nelle sezioni successive):

- `Prodotto`: classe che descrive un prodotto.
- `ProdottoScontatoPerc`: sottoclasse di `Prodotto` che descrive uno prodotto a cui è applicato lo sconto definito da una percentuale.
- `ProdottoScontato3x2`: sottoclasse di `Prodotto` che descrive uno prodotto con sconto di tipo 3x2 (sconto *tre per due*, compri tre paghi due).
- `Carrello`: classe che descrive un carrello in cui possono essere posti i prodotti da acquistare.
- `Esercizio1`, `Esercizio2`, `Esercizio3`: classi che definiscono il metodo `main`.

Specifica delle classi

Le classi dovranno esporre almeno i metodi specificati sotto, più eventuali altri metodi; in alcuni casi le definizioni dei metodi sono incomplete (vanno aggiunti i tipi mancanti). Gli attributi (campi) delle classi devono essere *privati*; per leggere e modificarne i valori, creare opportunamente, e solo dove necessario, i metodi di accesso (`set` e `get`). Si suggerisce di utilizzare, dove possibile, le classi parametriche opportunamente istanziate (es. *ArrayList*<*E*> invece di *ArrayList*). Ogni classe (eccetto quelle con metodo `main`) deve avere il metodo `toString` che rappresenti lo stato delle istanze.

class Prodotto

Classe che rappresenta un prodotto, caratterizzato da nome, prezzo unitario del prodotto e quantità acquistata. La classe deve disporre dei seguenti costruttori e metodi pubblici:

- `Prodotto(String nome, double prezzo, int q)`

  Costruisce un prodotto con nome, prezzo unitario e quantità specificati dai parametri. Solleva una eccezione se il nome è `null` oppure se prezzo o quantità sono negativi.

- `Prodotto(String nome, double prezzo)`

  Costruisce un prodotto con nome e prezzo unitario specificati dai parametri e quantità uguale a uno. Solleva una eccezione se il nome è `null` oppure se il prezzo è negativo. Notare che è sufficiente chiamare il costruttore precedente.

- `costo()`

  Calcola il costo complessivo del prodotto.

class ProdottoScontatoPerc (extends Prodotto)

Sottoclasse di `Prodotto` che rappresenta un prodotto a cui si applica lo sconto definito da una percentuale. La classe deve definire il seguente costruttore pubblico:

- `ProdottoScontatoPerc(String nome, double prezzo, int q, int sconto)`

  Costruisce un prodotto con nome, prezzo unitario, quantità e sconto specificati dai parametri. Lo sconto deve essere un intero compreso fra 1 e 100; ad esempio, se sconto vale 20, lo sconto applicato è del 20%. Solleva una eccezione se il nome è `null` oppure se prezzo o quantità sono negativi oppure se lo sconto non è compreso fra 1 e 100.

La classe deve riscrivere il metodo `costo()` della superclasse; il costo va calcolato tenendo conto dello sconto applicato.

class ProdottoScontato3x2 (extends Prodotto)

Sottoclasse di `Prodotto` che rappresenta un prodotto a cui è applicato lo sconto 3x2 (tre per due): ogni 3 pezzi di prodotto acquistato, se ne pagano 2. Ad esempio, se si acquistano 4 pezzi, se ne pagano 3; se si aquistano 10 pezzi, se ne pagano 7. La classe deve disporre del seguente costruttore pubblico:

- `Prodotto(String nome, double prezzo, int q)`

  Costruisce un prodotto con sconto 3x2 avente nome, prezzo unitario e quantità specificati dai parametri. Solleva una eccezione se il nome è `null` oppure se prezzo o quantità sono negativi.

Anche in questo caso il metodo `costo()` della superclasse va riscritto considerando lo sconto applicato.

class Carrello

Classe che rappresenta un carrello in grado di contenere dei prodotti; non vanno posti limiti al numero di prodotti che il carrello può contenere. La classe definisce i seguenti costruttori e metodi pubblici:

- `Carrello()`

  Costruisce un carrello vuoto.

- `aggiungiProdotto(Prodotto prod)`

  Aggiunge al carrello il prodotto specificato.

- `totale()`

  Calcola il totale speso per i prodotti nel carrello.

Esercizio 1

Definire la classe `Esercizio1` (classe con metodo `main`) che costruisce un carrello con i seguenti prodotti:

```
pasta   -  PREZZO UNITARIO: 2.4  - QUANTITA: 4
olio    -  PREZZO UNITARIO: 3.5  - QUANTITA: 10  - SCONTO: 20%
tonno   -  PREZZO UNITARIO: 3.5  - QUANTITA: 9   - SCONTO: 3x2
farina  -  PREZZO UNITARIO: 2.0  - QUANTITA: 5   - SCONTO: 15%
riso    -  PREZZO UNITARIO: 2.5  - QUANTITA: 8
burro   -  PREZZO UNITARIO: 2.5  - QUANTITA: 11  - SCONTO: 3x2
aceto   -  PREZZO UNITARIO: 1.2  - QUANTITA: 10  - SCONTO: 3x2
```

Il carrello va costruito direttamente nel metodo `main` (non vanno effettuate operazioni di input!). Quando il carrello è stato riempito, va stampato il contenuto e la spesa totale. Il formato della stampa è a piacere, purché siano chiaramente leggibili i dati rilevanti; per un esempio si veda la Figura 1.

```
pasta - PREZZO UNITARIO: 2.4 - QUANTITA: 4 - COSTO: 9.6
olio - PREZZO UNITARIO: 3.5 - QUANTITA: 10 - COSTO: 28.0 *** SCONTO 20% ***
tonno - PREZZO UNITARIO: 3.5 - QUANTITA: 9 - COSTO: 21.0 *** SCONTO 3x2 ***
farina - PREZZO UNITARIO: 2.0 - QUANTITA: 5 - COSTO: 8.5 *** SCONTO 15% ***
riso - PREZZO UNITARIO: 2.5 - QUANTITA: 8 - COSTO: 20.0
burro - PREZZO UNITARIO: 2.5 - QUANTITA: 11 - COSTO: 20.0 *** SCONTO 3x2 ***
aceto - PREZZO UNITARIO: 1.2 - QUANTITA: 10 - COSTO: 8.4 *** SCONTO 3x2 ***

==== TOTALE: 115.5
```

Figura 1: Esempio di output per l'Esercizio 1

Esercizio 2

Aggiungere alla classe `Carrello` i seguenti metodi pubblici:

- `bonus()`

  Calcola il bonus guadagnato con i prodotti nel carrello nel modo seguente:
  - per i prodotti senza sconto si ha un punto di bonus ogni 10 euro di spesa;
  - per i prodotti con sconto si ha un punto di bonus ogni 20 euro di spesa.

  Ad esempio, con il carrello dell'Esercizio 1 si spendono 29.6 euro per i prodotti senza sconto, che danno diritto a 2 punti di bonus, e 85.9 euro per i prodotti con sconto, che assegnano un bonus di 4 punti. Quindi il bonus guadagnato è di 6 punti.

- `aggiungiAProdotto(String nome, int n)`

  Se il carrello contiene il prodotto con il nome specificato, aggiunge n pezzi al prodotto, altrimenti non compie alcuna operazione. Si assume $n > 0$.

- `ordina()` **(FACOLTATIVO)**

  Ordina i prodotti nel carrello in ordine crescente rispetto al prezzo unitario; a parità di prezzo, i prodotti vanno ordinati in ordine alfabetico rispetto al nome. Non va scritto l'algoritmo di ordinamento, ma va usato uno degli algoritmi di ordinamento disponibili nelle API.

Definire la classe `Esercizio2` (classe con metodo `main`) che costruisce un carrello come specificato nell'Esercizio 1 e successivamente compie le seguenti operazioni:

1. Stampa il bonus guadagnato.
2. Esegue le seguenti istruzioni (`mioCarrello` va sostituito con la variabile usata per il proprio carrello):

   ```
   mioCarrello.aggiungiAProdotto("tonno", 11);
   mioCarrello.aggiungiAProdotto("zucchero", 10);
   mioCarrello.aggiungiAProdotto("aceto", 5);
   ```

3. Stampa il contenuto del carrello, il totale speso e il bonus guadagnato (devono risultare rispettivamente 147.1 euro e 7 punti). Se si è implementato il metodo `ordina`, ordinare il carrello prima di stamparlo.

Esercizio 3

Definire la classe `Esercizio3` (classe con metodo `main`) che permette di gestire un carrello interagendo con un utente. All'avvio viene stampato il menu

```
1. Inserisci prodotto senza sconto
2. Inserisci prodotto con sconto in percentuale
3. Inserisci prodotto con sconto 3x2
4. Stampa carrello (contenuto, totale, bonus)
5. Esci
```

A seconda dell'operazione selezionata dall'utente, viene eseguita l'operazione corrispondente; se il programma non è terminato, viene di nuovo stampato il menu. Le operazioni 1, 2, e 3 dovranno richiedere l'inserimento di ulteriori dati (nome prodotto, prezzo unitario, ...). È opportuno che le eccezioni sollevate siano catturate in modo da non causare la terminazione dell'applicazione. Si consiglia di partire da una applicazione minimale (operazioni 1,4,5) e di aggiungere il resto in seguito. È possibile aggiungere ulteriori operazioni (esempio, aggiungi a prodotto).

Consegna

Si ricorda che le classi devono essere tutte *public* e che vanno consegnati tutti i file *.java* prodotti. NON vanno consegnati i *.class* Per la consegna, eseguite l'upload dei SINGOLI file sorgente (NON un file archivio!) dalla pagina web: http://upload.di.unimi.it

ATTENZIONE!!! NON VERRANNO VALUTATI GLI ELABORATI CON ERRORI DI COMPILAZIONE. UN SINGOLO ERRORE DI COMPILAZIONE INVALIDA **TUTTO** L'ELABORATO.

- **Per ritirarsi**:
  Fare l'upload di un file vuoto di nome `ritirato.txt`.