Supermarket
Progetto di Programmazione
Febbraio 2010

1 Supermercato

Dovete realizzare un insieme di classi e interfacce che riguardano la gestione di un supermercato. nella Sezione 2 verranno descritte alcune classi per la manipolazione di date che risulteranno utili per lo svolgimento del progetto, mentre nella Sezione 3 si darà una descrizione della struttura delle classi e dei metodi da realizzare.

Notiamo che:

- non è necessario implementare tutte le classi e i metodi elencati; in particolare:
  - i metodi indicati con un † sono opzionali, e possono essere sviluppati in un secondo momento;
  - i metodi indicati con $\star$ sono obbligatori per chi segue il corso avanzato (da 7 cfu), facoltativi per gli altri;
- è possibile sviluppare altre classi o metodi, arricchendo il progetto proposto: è particolarmente importante, in tal caso, fornire una documentazione chiara delle nuove funzionalità;
- la documentazione può essere fornita in qualsiasi forma, ma si suggerisce in particolare di commentare il codice usando commenti javadoc.

2 Classi per le date

La classe di base per la gestione delle date in Java è GregorianCalendar (in java.util)

Che cosa è. Un'istanza di GregorianCalendar rappresenta un *istante di tempo*, cioè una data e un'ora (per esempio, 29 novembre 1968, ore 9:16): in realtà la precisione arriva fino al millisecondo, ma noi nel seguito ci limiteremo sempre e solo alla granularità del minuto. Per altro, d'ora in avanti, per "istante di tempo" intenderemo proprio una data e un'ora, come sopra.

Costruttori. La classe ha vari costruttori; i due che probabilmente vi occorrerà usare sono i seguenti:

- uno, senza argomenti, costruisce l'istante di tempo attuale (cioè, la data/ora di adesso);
- uno prende (in quest'ordine) anno, mese (0=gennaio, 1=febbraio ecc.), giorno (1,2,...), ore, minuti.

Esempi (vedi anche sopra):

```
// Crea e stampa la data 29 nov 1968, ore 9:16
data1 = new GregorianCalendar( 1968, 10, 29, 9, 16 );
// Crea la data/ora attuale
data2 = new GregorianCalendar();
```

Confronto. I metodi after e before permettono di stabilire se una data viene dopo o prima di un'altra. È anche disponibile un metodo x.compareTo(y) che confronta due date e restituisce un valore positivo, nullo o negativo a seconda che $x > y$, $x = y$ o $x < y$.

Stampa. Il metodo toString di GregorianCalendar non è adatto per stampare le date. Questo perché la stessa data si può stampare in modi diversi a seconda della nazione in cui ci si trova e di altri parametri. Per convertire una data in una stringa si usa un DateFormat (classe di java.text), come segue:

```
DateFormat formattatoreDiDate = DateFormat.getDateTimeInstance(
 DateFormat.LONG, DateFormat.SHORT, Locale.ITALY );
String outData;
data1 = new GregorianCalendar( 1968, 10, 29, 9, 16 );
outData = formattatoreDiDate.format( data1.getTime() );
out.println( outData );
```

3 Le classi

Le classi e le interfacce da realizzare sono elencate nel seguito.

3.1 Classe CategoriaMerceologica

Il supermercato vende prodotti collocati in diverse categorie merceologiche. Le categorie merceologiche costituiscono una specie di gerarchia: ogni categoria può avere una super-categoria di cui fa parte (ad esempio, la categoria *Frutta e verdura* fa parte delle super-categoria *Generi alimentari*).

La classe ha i seguenti costruttori e metodi:

- CategoriaMerceologica(String nome, String descrizione): crea una categoria con dato nome e descrizione; in questo caso, si intende che non c'è alcuna super-categoria[1];
- CategoriaMerceologica(String nome, String descrizione, CategoriaMerceologica): crea una categoria con dato nome e descrizione, e una certa supercategoria;
- CategoriaMerceologica **super**(): restituisce la super-categoria di quella su cui è invocato (**null** se la categoria non ha supercategorie);
- String toString(): stampa il nome della categoria seguito, fra parentesi, dalla sua descrizione;
- †String gerarchia(): stampa l'elenco delle super-categorie nella gerarchia fino a quella su cui è invocato. Ad esempio, se x è la categoria *Frutta fresca*, che ha *Frutta e verdura* come super-categoria, che ha *Generi alimentari* come super-categoria, invocando x.gerarchia() deve essere restituita la stringa:

  Generi alimentari / Frutta e verdura /Frutta fresca.

- †CategoriaMerceologica radice(): restituisce la categoria merceologica radice, che si ottiene risalendo l'albero delle super-categorie fino a raggiungere la categoria che non ha super-categorie; questo metodo può essere scritto in modo ricorsivo, osservando che se x non ha super-categorie, è radice di sé stessa, altrimenti se y è la super-categoria di x allora la radice di x è la radice di y.

Esempio di uso:

```
CategoriaMerceologica generiAlimentari, fruttaVerdura, fruttaFresca;
generiAlimentari=new CategoriaMerceologica("Generi alimentari",
 "Tutti i generi alimentari");
fruttaVerdura=new CategoriaMerceologica("Frutta e verdura",
 "Tutti i tipi di frutta e verdura", generiAlimentari);
fruttaFresca=new CategoriaMerceologica("Frutta fresca",
 "Solo frutta fresca (banco frigo)", fruttaVerdura);
System.out.println(fruttaFresca.radice()); // Stampa generiAlimentari
```

[1] Consigliamo di mettere il campo corrispondente a **null**.

3.2 Classe Prodotto

Un *prodotto* rappresenta un oggetto in vendita presso il supermercato. Esso è caratterizzato dai seguenti dati: un nome, il nome del produttore, la categoria merceologica, il codice del prodotto e il prezzo unitario.

- Prodotto(String nome, String produttore, CategoriaMerceologica cat, String codice, **double** prezzo): crea un prodotto con i dati indicati;
- String toString(): decidete voi, in modo opportuno, che cosa questo metodo debba restituire;
- **boolean** stessaCategoriaDi(Prodotto x): restituisce true se e solo se il prodotto è della stessa categoria di x;
- †**boolean** stessaFamigliaDi(Prodotto x): restituisce true se e solo se le radici delle due categorie merceologiche sono uguali.

Esempio di uso:

```
Prodotto fagioliBonduelle, zucchini;
fagioliBonduelle=new Prodotto("Fagioli messicani Bonduelle (conf. 500g)",
 "Bonduelle", verduraInScatola,
 "A38132A88F", 1.45);
zucchini=new Prodotto("Zucchine (al g.)",
 "-", verduraFresca,
 "A88213B11Z", 0.93);
zucchini.stessaCategoriaDi(fagioliBonduelle); // false
zucchini.stessaFamigliaDi(fagioliBonduelle); // true
```

3.3 Interfaccia Promozione

Una promozione è un sistema di sconto che, data la quantità di prodotto acquistato e dato il suo prezzo unitario, stabilisce il prezzo scontato. Poiché esistono molti modi di ottenere tale sconto, l'interfaccia si limita a specificare un metodo:

- **double** prezzoScontato(**int** quantita,**double** prezzo)

che restituisce il prezzo totale per l'acquisto di una certa quantità con un certo prezzo nominale. Il valore restituito da questo metodo sarà sempre minore di o uguale a quantita∗prezzo.

Ecco alcune implementazioni di questa interfaccia:

- ScontoSemplice: è una promozione molto semplice, che consiste nell'applicare una certa percentuale di sconto, indipendentemente dalla quantità acquistata; la percentuale di sconto è fornita nel costruttore di questa classe;
- TrePerDue: è una promozione che consiste nel vendere tre prodotti al prezzo di due (notate che se i prodotti acquistati non sono multipli di tre, al resto viene applicato il prezzo normale);

- †KPerH: una generalizzazione del tre per due, il k per h, dal significato ovvio; se implementate questa classe, rendete TrePerDue una sua sottoclasse;
- TreFasce: sono fissate due quantità $q_1 \leq q_2$ e due percentuali di sconto $p_1 \leq p_2$; se la quantità acquistata q è $q < q_1$, non viene applicato alcuno sconto; se la quantità q è tale che $q_1 \leq q < q_2$, viene applicata la percentuale di sconto p_1; se $q_2 \leq q$, viene applicata la percentuale q_2.

3.4 Classe CampagnaPromozionale

Una campagna promozionale consiste in un insieme di prodotti a ciascuno dei quali è associata una certa promozione. Più precisamente, una campagna promozionale è caratterizzata da:

- una data di inizio;
- una data di fine;
- un booleano che dice se la campagna è riservata oppure no; una campagna riservata è una campagna che è applicata solo ai clienti che hanno la carta fedeltà;
- un elenco di prodotti, ciascuno con associata una promozione:
  - gli studenti del corso ordinario possono realizzare questo elenco con un array (sovradimensionato) di prodotti e un array parallelo di promozioni;
  - $\star$ per gli studenti del corso avanzato, il modo naturale per realizzare questo elenco consiste in una Map<Prodotto,Promozione>.

Tale classe è fornita dei seguenti costruttori e metodi:

- CampagnaPromozionale(GregorianCalendar inizio, GregorianCalendar fine, **boolean** riservata): crea la campagna con i dati indicati;
- **void** add(Prodotto p, Promozione prom) **throws** IllegalArgumentException: aggiunge la coppia indicata alla campagna; se il prodotto è già presente nella campagna, solleva una IllegalArgumentException;
- **double** prezzo(Prodotto p, **int** q, GregorianCalendar d, **boolean** fidelity): questo metodo assume che un certo cliente (che ha o non ha la carta fedeltà a seconda del parametro booleano), abbia acquistato in data d una quantità q del prodotto p, e restituisce il prezzo che dovrebbe pagare: tale prezzo è ottenuto con una semplice moltiplicazione se il prodotto non è in promozione, o se la promozione non è in corso, o se si tratta di una promozione riservata ma il cliente non ha la carta; se invece tutte le condizioni si possono applicare, restituisce il prezzo totale opportunamente scontato;

- †**void** setPuntiFedelta(**double** x, **int** y) **throw** UnsupportedOperationException: questo metodo può essere invocato solo sulle campagne promozionali riservate (sulle altre, il tentativo di invocazione solleva un'eccezione); per le campagne su cui può essere invocato, fissa il rapporto fra sconti e punti ottenuti sulla carta fedeltà. In particolare, ogni x euro di sconto si ottengono y punti fedeltà (di default $x = 1$ e $y = 0$, che significa in pratica che non si ottengono punti fedeltà);

- †**int** punti(Prodotto p, **int** q, GregorianCalendar d, **boolean** fidelity): questo metodo stabilisce quanti (eventuali) punti fedeltà otterrebbe un cliente che nella data specificata acquistasse la quantità indicata del prodotto p; il booleano indica se il cliente possiede o meno la carta fedeltà. Il metodo restituisce 0 nei seguenti casi: se la campagna non è attiva nella data indicata; se la campagna non è riservata; se il cliente non possiede la carta fedeltà; se il prodotto non è in sconto. In tutti gli altri casi, il numero di punti fedeltà restituiti dipende dalla chiamata del metodo setPuntiFedelta.

3.5 Class Carrello

Un carrello è un insieme di prodotti acquistati da un certo cliente. La classe Carrello è fornita dei seguenti costruttori e metodi:

- Carrello(GregorianCalendar d, **boolean** fidelity): crea un carrello vuoto: il booleano dice se il cliente possiede la carta fedeltà;

- **void** add(Prodotto p, **int** q): aggiunge al carrello una quantità q del prodotto p; notate che è possibile che si aggiunga più volte lo stesso prodotto, e vanno sommate le corrispondenti quantità;

- **int** size(): dice quanti prodotti distinti ci sono nel carrello;

- Prodotto get(**int** i): dice qual è l'i-esimo prodotto (i deve essere compreso fra 0 e $n - 1$, dove n è il numero di prodotti attualmente nel carrello);

- **int** quantity(**int** i): dice qual è la quantità dell'i-esimo prodotto (i deve essere compreso fra 0 e $n - 1$, dove n è il numero di prodotti attualmente nel carrello).

3.6 Class Supermercato

Questa classe rappresenta il supermercato. I suoi costruttori e metodi sono

- Supermercato(): crea il supermercato;

- **void** add(CampagnaPromozionale c): aggiunge al supermercato la campagna promozionale c;

- **double** vaiAllaCassa(Carrello x): calcola quanto il cliente deve pagare per il carrello considerato; per tale calcolo, si procede come segue: di ogni prodotto del carrello si determina quali delle campagne promozionali attualmente attive sono applicabili a quel prodotto (nel caso che il cliente non abbia la carta fedeltà, vengono considerate solo le campagne non riservate); fra tutti gli sconti applicabili, viene considerato il più alto (cioè quello con cui il prodotto risulta avere il costo minore);

- ⋆†**void** addCliente(String cognome, String nome, GregorianCalendar d, String cod): aggiunge un cliente con carta fedeltà (con dati cognome, nome, data di nascita e codice cliente); tali dati verranno memorizzati in un oggetto (di nome ClienteFedele) e tenuti in una mappa Map<ClienteFedele,Integer> che, di ogni cliente fedele, memorizza quanti punti fedeltà ha;

- ⋆†**void** aggiornaPunti(Carrello x, String cod): assume che il carrello x sia stato creato da un cliente fedele di codice cod: calcola di quanti punti fedeltà il cliente avrebbe diritto, e aggiorna il suo totale.