Dispensa

Progetto di Programmazione del 29 Gennaio 2018

Introduzione

In questo esame vi chiederemo di scrivere un insieme di classi per la gestione della dispensa familiare.

Il progetto verrà valutato prima di tutto in base al suo corretto funzionamento rispetto ai requisiti qui descritti; suggeriamo di commentare con cura il codice, in particolare anche scrivendo i commenti `javadoc`.

Realizzazione

Classe Prodotto

La classe `Prodotto` rappresenta un prodotto che può far parte della dispensa (per esempio, "latte", "pan baulino", "tarallucci", "fronzies").

Un prodotto è caratterizzato solo da un nome (una stringa), e due prodotti sono uguali se hanno lo stesso nome (a meno di maiuscole e minuscole: cioè "latte" e "Latte" sono considerati lo stesso prodotto).

La classe ha solo un costruttore

- `public Prodotto(String nome)`

e sovrascrive i metodi `toString`, `equals` e `hashCode` di `Object`.

Classe Dispensa

Gli oggetti di questa classe rappresentano delle dispense familiari. Una dispensa è caratterizzata da:

- un elenco (*senza ripetizioni*) di prodotti
- per ogni prodotto, una quantità attualmente presente in dispensa
- per ogni prodotto, una quantità minima di scorta che desidereremmo avere sempre in dispensa.

Per realizzare la classe esistono vari modi. Vi suggeriamo tre soluzioni in ordine crescente di difficoltà (naturalmente, nel valutare il vostro compito terremo conto di quale soluzione avrete adottato):

- *Soluzione 1*: usate tre array paralleli (un array di `Prodotto` e due array di interi); potete assumere

per questa soluzione che al mondo esistano solo 1000 prodotti possibili.

- *Soluzione 2*: usate tre `ArrayList` parallele (una di `Prodotto` e due di `Integer`).
- *Soluzione 3*: usate due `HashMap<Prodotto,Integer>` (la descrizione della classe `HashMap` è data in fondo).

La dispensa ha i seguenti costruttori e metodi:

- `public Dispensa()` : crea la dispensa vuota.
- `public void scortaMinima(Prodotto p, int q)` : stabilisce che il prodotto `p` abbia quantità minima `q` (se era stata precedentemente fissata un'altra quantità minima per quel prodotto, ora essa verrà modificata).
- `public int qta(Prodotto p)` : restituisce la quantità presente in dispensa del prodotto `p`.
- `public void consuma(Prodotto p, int q)` : assume che una quantità `q` del prodotto `p` sia stata consumata; se la dispensa contiene meno di `q` unità di quel prodotto, questo metodo deve sollevare l'eccezione (non controllata) `NoSuchElementException` (del pacchetto `java.util`), senza fare null'altro.
- `public ListaDellaSpesa preparaLista()` : restituisce una lista della spesa, contenente tutti i prodotti la cui quantità in dispensa è inferiore alla quantità minima prevista. La quantità da acquistare è data dalla differenza fra la quantità minima prevista e quella attualmente presente in dispensa.
- `public void riponi(ListaDellaSpesa s)` : la lista è stata acquistata, e i prodotti vengono tolti dai sacchetti e messi nella dispensa. [*Metodo facoltativo*]

Riscrivete opportunamente anche il metodo `toString`.

Classe ListaDellaSpesa

Le istanze di questa classe sono liste della spesa (elenchi di prodotti con associata la quantità da acquistare). Decidete voi come realizzare gli attributi di questa classe (avete essenzialmente le stesse soluzioni descritte per `Dispensa`).

Ha un costruttore:

- `public ListaDellaSpesa()` : crea una lista della spesa vuota.

Ha inoltre i metodi

- `public void aggiungi(Prodotto p, int q)` : aggiunge la quantità `q` del prodotto `p` alla lista.
- `public int qta(Prodotto p)` : restituisce la quantità del prodotto `p` da acquistare secondo la lista.
- `public Prodotto[] prodotti()` : restituisce un array con tutti i prodotti che compaiono nella lista. (Se vi risulta più comodo, potete far restituire a questo metodo un' `ArrayList<Prodotto>` invece di un array). [*Metodo facoltativo*].

e un'opportuno `toString`.

Introduzione: mappe in Java

Nel pacchetto `java.util` della libreria standard, ci sono delle interfacce e classi per realizzare *mappe*. Una mappa è una specie di funzione che contiene un insieme di coppie (chiave, valore).In ogni istante, a ciascuna chiave può essere associato un solo valore.

Più precisamente `HashMap<K,V>` è una classe che corrisponde a mappe con chiavi di tipo `K` e valori di tipo `V`. Ad esempio, una `HashMap<String,Integer>` è una mappa da stringhe a interi. All'inizio (quando la costruite) la mappa è vuota.

- Il metodo `put(K k, V v)` aggiunge una nuova coppia `(k,v)` alla mappa; se la chiave `k` era precedentemente già associata a un valore, diciamo `v'`, viene prima rimossa la coppia `(k,v')` dalla mappa e poi viene inserita la nuova coppia `(k,v)`.
- Il metodo `containsKey(K k)` che restituisce `true` se e solo se alla chiave `k` è associato un valore.
- Il metodo `get(K k)` restituisce il valore associato alla chiave `k` (oppure `null`, se in questo momento la mappa non contiene alcun valore associato a quella chiave).
- Potete iterare su tutte le chiavi con un foreach; se `mappa` è la mappa:

  ```
  for (K k: mappa.keySet()) {
      ...
  }
  ```

- Potete ottenere un array di tutte le chiavi con l'istruzione paradigmatica:

  ```
  mappa.keySet().toArray(new K[] {})
  ```

che restituisce un array di `K`.

> Notate che per garantire il corretto funzionamento di una mappa `HashMap<K,V>` occorre essere certi che la classe `K` sovrascriva correttamente sia il metodo `equals` che il metodo `hashCode`.