Dipartimento di Informatica Sistemistica e Comunicazione
Corso di Laurea in Informatica

Programmazione 2

2 Luglio 2015 – Recupero Secondo Compitino

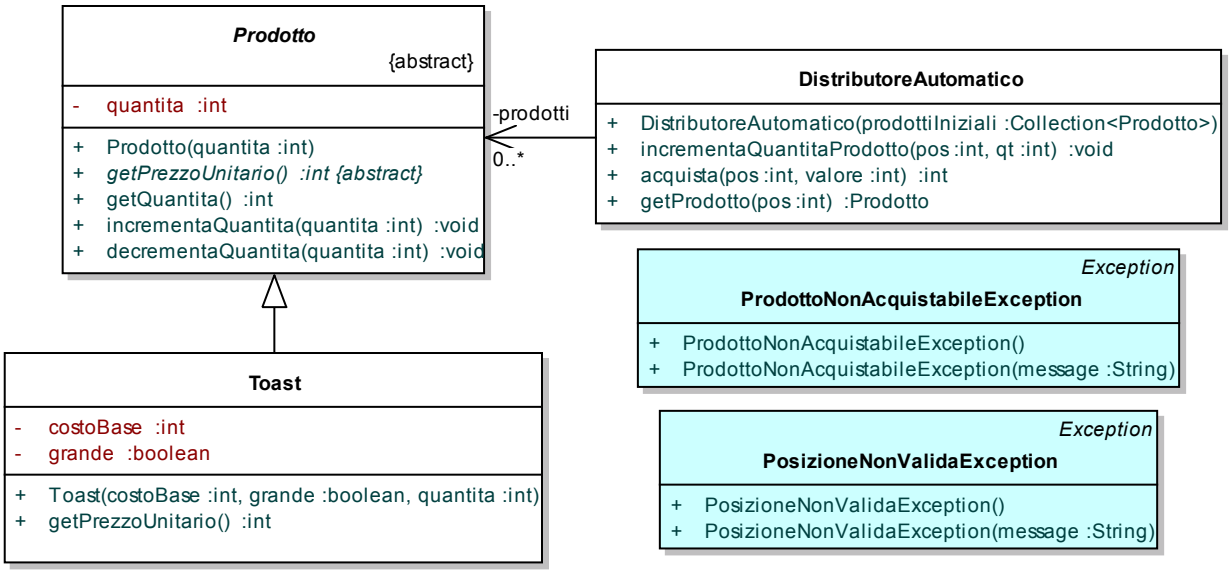
Testo parte di pratica

Si realizzino le classi che modellano un distributore automatico di prodotti (ad esempio, patatine, toast, etc.) seguendo la specifica indicata di seguito. Si provino le classi realizzate con JUnit utilizzando i test forniti nella classe `TestDistributore`.

Nota: Gli elaborati che non superino almeno 4 test fra quelli dati saranno considerati insufficienti.

(Suggerimento: si eviti di eseguire i test solo alla fine del lavoro, quando ormai sarebbe tardi per apportare correzioni; Piuttosto si eseguano i test man mano che si procede con l'implementazione, per verificare incrementalmente il lavoro via via fatto.)

Le classi `PosizioneNonValidaException` e `ProdottoNonAcquistabileException` rappresentano eccezioni che possono essere sollevate dal programma e sono date, mentre le classi `Prodotto`, `Toast` e `DistributoreAutomatico` devono essere implementate in modo coerente al diagramma e alla specifica che seguono:



Classe `Prodotto`:

- È una classe astratta che rappresenta un generico insieme di prodotti collocati su una posizione del distributore (ad esempio alla posizione 2 è presente un prodotto di tipo schiacciatina il cui costo è 20 e ce ne sono 20 pezzi)
- Ha un costruttore con parametro che imposta il valore dell'attributo `quantita`. Se il valore del parametro è negativo, l'attributo deve essere inizializzato a 0.
- Implementa il metodo astratto `int getPrezzoUnitario()`.
- Implementa i metodi `void incrementaQuantita(int quantita)` e `void decrementaQuantita(int quantita)` che rispettivamente incrementano e decrementano il valore dell'attributo `quantita` di un ammontare pari al valore del parametro del metodo.
- Implementa un metodo getter che ritorna il valore dell'attributo `quantita`.

Classe `Toast`:

- È una classe che rappresenta uno specifico tipo di prodotto. La classe implementa l'attributo `costoBase`, che rappresenta il costo di un toast di dimensioni standard, e l'attributo `grande`, che indica se si tratta di un toast grande o standard. Il costruttore inizializza tutti gli attributi. Entrambi gli attributi sono immutabili e non accessibili.
- Implementa il metodo `getPrezzoUnitario()` ritornando il doppio del `costoBase` se il toast è grande, il `costoBase` altrimenti.

Classe `DistributoreAutomatico`:

- È una classe che rappresenta un distributore di prodotti. Ogni prodotto ha una posizione nel distributore. I prodotti sono memorizzati in un attributo `prodotti` di tipo `ArrayList`. La posizione del prodotto coincide con la posizione nell'`ArrayList`. I prodotti possono essere acquistati oppure ricaricati.
- Il costruttore ha un parametro di tipo `Collection<Prodotto>` che rappresenta i prodotti con cui il distributore è caricato inizialmente. Si suppone che all'interno della collection non possano esserci valori `null`. Il costruttore deve inserire in `prodotti` tutti i prodotti nella collection passata come parametro del costruttore.
- Implementa il metodo `getProdotto(int pos)` che ritorna il prodotto in `prodotti` che si trova alla posizione `pos`, oppure l'eccezione `PosizioneNonValidaException` se la posizione non è una posizione valida.
- Implementa il metodo `incrementaQuantita(int pos, int qt)` che incrementa di `qt` la quantità del prodotto in posizione `pos`, oppure l'eccezione `PosizioneNonValidaException` se la posizione non è una posizione valida.
- Implementa il metodo `acquista(int pos, int valore)` che rappresenta il tentativo di acquistare il prodotto alla posizione `pos` inserendo un ammontare pari a `valore` nel distributore. Esso ritorna
  - Il resto, cioè la differenza tra `valore` e il costo del prodotto alla posizione `pos`, se il valore inserito è maggiore oppure uguale al costo del prodotto e la posizione `pos` è una posizione valida
  - l'eccezione `PosizioneNonValidaException` se la posizione `pos` non è una posizione valida.
  - l'eccezione `ProdottoNonAcquistabileException` contenente la stringa "`quantita insufficiente`" come parte del messaggio dell'eccezione se la quantità del prodotto alla posizione `pos` è pari a 0.
  - l'eccezione `ProdottoNonAcquistabileException` contenente la stringa "`valore insufficiente`" come parte del messaggio dell'eccezione se il costo del prodotto alla posizione `pos` è maggiore del valore del parametro `valore`.