

Relazione Progetto Java: Gestore Liste della Spesa

1. Introduzione

Il progetto, svolto da Timothy Giolito 20054431 e Franzon Luca 20054744, consiste nello sviluppo di un'applicazione Java per la gestione di liste della spesa. L'obiettivo principale è fornire uno strumento che permetta di creare multiple liste, gestire un catalogo globale di prodotti, organizzare gli articoli per categorie e reparti/corsie e monitorare il budget tramite il calcolo dei totali. L'applicazione è stata progettata seguendo i principi della programmazione orientata agli oggetti.

2. Architettura e Descrizione delle Classi

Il progetto è strutturato in pacchetti logici per separare le responsabilità, distinguendo il modello dei dati dall'interfaccia utente.

2.1 Package **modello**

Questo package contiene il cuore della logica di business.

- **Articolo**: Rappresenta l'unità fondamentale del sistema. Ogni oggetto **Articolo** incapsula dati quali nome, categoria, prezzo, note aggiuntive e il reparto di appartenenza. La classe implementa controlli di validazione nel costruttore per garantire che i dati siano coerenti (es. prevenire nomi vuoti o prezzi negativi) e gestisce valori di default per i campi opzionali.

Gestione Dinamica delle Categorie (Sostituzione di Enum) Per soddisfare il requisito funzionale che prevede *"categorie definite dall'utente"*, è stata scartata l'ipotesi iniziale di utilizzare una **Enum** statica (troppo rigida e non modificabile a runtime). È stata invece implementata una gestione basata su una lista dinamica di stringhe (**List<String> categorie**) all'interno della classe static **GestioneListe**.

Questa scelta progettuale garantisce:

1. **Estensibilità a Runtime**: L'utente può creare e rimuovere nuove categorie sia da riga di comando (CLI) che da interfaccia grafica (GUI) senza necessità di ricompilare il codice.
2. **Validazione dei Dati**: Il sistema implementa controlli preventivi per evitare l'inserimento di categorie duplicate (controllo *case-insensitive*) o vuote.
3. **Integrità Referenziale**: È stato introdotto un vincolo che impedisce la cancellazione della categoria di default ("Non categorizzato"), garantendo che ogni articolo abbia sempre una classificazione valida.

- **ListaDiArticoli**: Gestisce una singola lista della spesa. Una caratteristica chiave è la gestione del ciclo di vita dell'articolo:
 - **Lista Attiva**: Contiene la lista di prodotti da acquistare.
 - **Cestino**: Quando un articolo viene rimosso, non viene cancellato definitivamente ma spostato nel "cestino", cioè una lista di prodotti cancellati dalla lista attiva (**cancellati**), permettendo all'utente di ripristinarlo in caso di errore.
 - Implementa l'interfaccia **Iterable<Articolo>** per permettere l'iterazione sequenziale su tutti gli elementi (sia attivi che nel cestino).
 - Offre funzionalità di "Smart Search" (**TrovaArticoloPerPrefisso**) per cercare articoli sia nella lista attiva che nel cestino tramite prefisso.
- **GestioneListe**: È una classe statica che funge da "Database in memoria" e controller centrale. Gestisce:
 - La mappa globale delle liste della spesa (**Map<String, ListaDiArticoli>**).
 - Il catalogo globale degli articoli (**List<Articolo>**) per gli inserimenti veloci.
 - L'elenco delle categorie personalizzate.
 - Garantisce l'unicità dei nomi delle liste e degli articoli nel catalogo, lanciando eccezioni in caso di duplicati.

2.2 Package **modello.eccezioni**

Sono state definite eccezioni personalizzate per gestire errori specifici e migliorare la robustezza del codice:

- **ArticoloException**: Sollevata per errori nella creazione o modifica di un articolo (es. prezzo negativo).
- **ListaDiArticoliException**: Sollevata per errori relativi alla singola lista.
- **GestioneListeException**: Sollevata per errori a livello globale, come il tentativo di creare una lista con un nome già esistente.

2.3 Package **interfaccia**

Gestisce l'interazione con l'utente attraverso due modalità distinte.

- **cli.RigaDiComando**: Implementa l'interfaccia testuale. Utilizza la classe `Scanner` per leggere l'input e presenta un menu nidificato per navigare tra gestione liste, catalogo e categorie.
- **mvc (Model-View-Controller)**: Questo sotto-pacchetto implementa l'interfaccia grafica separando la logica di presentazione dalla gestione degli eventi.
 - **VistaGUI**: Si occupa esclusivamente della visualizzazione. Estende `JFrame` e costruisce il layout grafico (usando `BorderLayout`, `JTable` per gli articoli e pannelli laterali per le liste). È una classe "passiva": non contiene logica di business, ma espone metodi pubblici (es. `aggiornaTabella`, `mostraMessaggio`) per permettere al Controller di modificare ciò che l'utente vede.
 - **ControllerGUI**: Gestisce la logica di interazione. Implementa le interfacce `ActionListener` e `ListSelectionListener` per catturare gli input dell'utente (click sui bottoni, selezione righe). Riceve le notifiche dalla Vista, invoca i metodi del Modello (`GestioneListe`) e aggiorna la Vista di conseguenza.

2.4 Package **main**

- **Main**: È il punto di ingresso (`entry point`) del programma. Chiede all'utente quale interfaccia avviare (1 per CLI, 2 per GUI, 3 per uscire) e istanzia la classe corrispondente.

3. Scelte Progettuali

Durante lo sviluppo sono state adottate le seguenti scelte per garantire manutenibilità ed efficienza:

1. **Adozione del Pattern MVC (Model-View-Controller):** Per l'interfaccia grafica, si è scelto di separare nettamente la **VistaGUI** (che definisce solo il layout e i componenti Swing) dal **ControllerGUI** (che gestisce gli eventi e la comunicazione con il modello). Questo permette di modificare l'aspetto grafico senza rischiare di rompere la logica di gestione degli eventi e mantiene il codice più pulito e modulare rispetto a una soluzione monolitica. La logica dei dati (Modello) rimane così completamente indipendente da entrambe le interfacce (CLI e GUI).
2. **Gestione Statica dei Dati (GestioneListe):** Si è scelto di utilizzare metodi e campi statici nella classe **GestioneListe** per simulare un repository centrale accessibile da qualsiasi punto dell'applicazione. Questo evita la complessità di dover passare un'istanza di "Database" tra le varie finestre o menu.
3. **Soft Delete (Cestino):** Invece di eliminare direttamente gli oggetti, è stato implementato un meccanismo di "cestino" all'interno di **ListaDiArticoli**, che non elimina in modo definitivo gli articoli, bensì li sposta in una lista "cestino", in modo che nel caso di eliminazione di un prodotto in modo accidentale da parte dell'utente, i dati possano essere recuperabili.
4. **Uso delle Eccezioni:** L'uso di eccezioni custom (**Checked Exceptions**) costringe chi usa le classi del modello a gestire esplicitamente i casi di errore, rendendo l'applicazione più robusta.

4. Manuale Utente: Come avviare il programma

Prerequisiti

- Java Development Kit (JDK) 21 o superiore installato.
- Il codice sorgente compilato nella cartella `bin` o importato in un IDE (Eclipse/IntelliJ).

Avvio dell'Applicazione

Il punto di ingresso è la classe `main.Main`.

Da Terminale:

Posizionarsi nella cartella radice del progetto (dove si trova la cartella `bin`).

Eeguire il comando:

```
java -cp bin main.Main
```

Il programma mostrerà il seguente menu:

Seleziona la modalità di avvio:

1. Interfaccia a Riga di Comando (CLI)
2. Interfaccia Grafica (GUI)
3. Esci

Guida alle Interfacce

- **Opzione 1 (CLI):** Si verrà guidati dal menù testuali. Digitare il numero corrispondente all'azione desiderata (es. "1" per creare una lista) e premere Invio. Le azioni possibili sono 8.
- **Opzione 2 (GUI):** Si aprirà una finestra.
 - A sinistra si trova l'elenco delle liste (con la possibilità di crearne di nuove con "Nuova Lista Spesa").
 - Selezionando una lista, al centro apparirà la tabella degli articoli.
 - Usare i pulsanti in basso ("Aggiungi", "Rimuovi", "Copia da Catalogo") per gestire la spesa.
 - Il pulsante "Vedi Cestino" in alto a destra permette di visualizzare e ripristinare gli articoli rimossi.

5. Testing

Il progetto include una suite di test unitari (**JUnit 5**) nel package `modello.test`.

- `testArticolo`: Verifica costruttori e validazione dati.
- `testListaDiArticoli`: Verifica aggiunta, rimozione, calcolo totali e iterazione.
- `testGestioneListe`: Verifica l'unicità delle liste e la gestione del catalogo globale.

Per eseguire i test, è necessario includere la libreria JUnit 5 nel classpath ed eseguire le classi di test tramite l'IDE o runner JUnit.