Maintenance

Gabriele Chignoli

Agosto 2025

Contents

1 Introduzione

Nel seguente documento vengono elencate alcune delle maggiori attività di refactoring eseguite sul progetto.

2 Attività

2.1 Ridenominazione Metodi - 01/08/2025

È stata corretta la denominazione dei metodi presenti nella classe *ObjectManager* per i quali non si erano rispettate le convenzioni sulla scrittura del codice presenti nel Project Plan. Essendo un'operazione semplice non c'è stato bisogno di alcuna pianificazione.

2.2 Refactoring - 16/08/2025

Obiettivo Rendere più leggibile e più corretto come la GUI gestisce la rappresentazione della matrice dei (9) prodotti, la sua generazione e come si passa da una pagina all'altra.

branch

```
main/GUI/db-interaction
```

`MainWindow` Il codice per la inizializzazione della matrice è stato messo in un metodo (`matrixInit`) che prende i nomi dei prodotti da mostrare (`p_names`) chiamando la classe `MatrixRenderer`, che è quella che interagisce con le classi che eseguono le operazioni effettive sul database. Così facendo il metodo può essere chiamato dalle operazioni (le azioni dei pulsanti, per esempio) che hanno bisogno di aggiornare la matrice.

```
public static void matrixInit() {
    String[] p_names = MatrixRenderer.getProductsToShow(
        MainWindow.getProductSearchText());
    ...
}
```

`MatrixRenderer` Per cambiare pagina, la classe incrementa/decrementa un valore che poi viene utilizzato come indice di partenza per prelevare i prodotti dalla lista che viene ritornata dalle ricerche nel database. Per cambiare questo valore (`current_page`) è stata introdotta una classe (`brosePage`) che utilizza un tipo enumerativo per determinare se aggiungere o togliere all'indice. Il metodo viene chiamato dalle azioni dei pulsanti, che specificano se richiedono di andare alla pagina precedente o successiva. Si ritiene questa soluzione più pulita e chiara al lettore.

```
public static void browsePage(SwipePage operation) {
    switch(operation) {
        case SwipePage.NEXT:
            current_page ++;
        break;
        case SwipePage.PREV:
            current_page --;
        break;
        default:
```

```
    }
}
```

dove `SwipePage` è un tipo enumerativo per aumentare la leggibilità. Le classi delle azioni che eseguono i componenti dell'interfaccia saranno inoltre spostate nella cartella `controller`, per cercare di rispettare meglio il paradigma MVC.

2.3 Refactoring - 27/08/2025

Obiettivo Rendere più leggibile la classe `MainWindow` che gestisce la finestra principale dell'applicazione, limitando in particolare l'utilizzo di variabili e metodi `static`. Viene inoltre riscritta per rispettare il paradigma *singleton*.

branch

```
main/mainWindow-refactor
```

`MainWindow` Si è deciso di applicare il pattern *singleton* alla classe poiché si ritiene che questa debba essere istanziata una sola volta, e i componenti che ci interagiscano debbano accedere a questa unica istanza.

```
public static MainWindow getInstance() {
    if(instance == null) {
            instance = new MainWindow();
    }
    return instance;
}
```

Il codice qui sopra istanzia una pagina solo se non lo è già stata, e fornisce ai componenti esterni un handle per interagire con la schermata.

Il codice è stato inoltre ordinato e formattato più correttamente, rendendo la lettura più semplice. Purtroppo si ritiene che questa classe "faccia troppe cose" e andrebbe snellita estrapolando metodi delegando, per esempio, l'aggiunta di pulsanti/pannelli ad altri componenti.