Progetto ACMC

Architettura e Dependency Injection

Il progetto ACMC si basa su un'architettura che centralizza la gestione delle dipendenze attraverso una classe chiamata Context. Questa classe funziona come un contenitore che si occupa di creare e fornire tutti i componenti necessari all'applicazione. Quando l'applicazione si avvia, il blocco statico del Context configura la connessione al database e registra tutte le dipendenze principali. Il vantaggio di questo approccio è che i vari componenti dell'applicazione non devono preoccuparsi di come vengono create le loro dipendenze: chiedono semplicemente al Context di fornirgliele. Questo meccanismo si chiama autowiring e funziona cercando le dipendenze in base al loro tipo.

Utility per la Validazione degli Input

La classe InputValidator raccoglie tutta la logica necessaria per validare gli input che arrivano dalla console. L'idea è evitare di duplicare questo codice in ogni controller. Il pattern utilizzato è quello del ciclo infinito con ritorno anticipato: si crea un while(true) che continua a chiedere l'input finché non ne arriva uno valido, momento in cui il metodo restituisce immediatamente il valore. Questo approccio è più pulito rispetto all'uso di flag booleani per controllare l'uscita dal ciclo.

Per quanto riguarda i numeri decimali, c'è una questione importante legata al formato. In Italia scriviamo i numeri con il punto come separatore delle migliaia e la virgola per i decimali, mentre BigDecimal accetta solo il formato americano con il punto per i decimali. La classe BigDecimalUtil si occupa proprio di questa conversione, rimuovendo i punti delle migliaia e sostituendo la virgola con il punto.

Quando si confrontano valori BigDecimal, non bisogna mai usare equals() ma sempre compareTo(). Questo metodo restituisce -1 se il primo valore è minore, 0 se sono uguali e 1 se è maggiore. La ragione è che equals() considera anche la scala del numero, quindi 2.0 e 2.00 risulterebbero diversi anche se matematicamente sono uguali.

Per le date, il pattern yyyy-MM-dd segue lo standard ISO 8601. È importante ricordare che questo pattern è case-sensitive: la y minuscola indica l'anno mentre

la M maiuscola indica il mese.

Gli enum in Java hanno un metodo valueOf() che converte una stringa nella corrispondente costante dell'enum. Questo metodo è case-sensitive e lancia un'eccezione IllegalArgumentException se la stringa non corrisponde a nessuna costante, quindi va sempre usato dentro un blocco try-catch.

I Controller

Il Main dell'applicazione funziona come un semplice orchestratore che non contiene logica. Il suo compito è mostrare il menu principale e instradare le scelte dell'utente verso i controller specializzati. I menu sono caricati da file esterni, il che permette di modificarli senza dover ricompilare il codice e facilita eventuali traduzioni in altre lingue.

Ogni controller è specializzato in un'area specifica: MemberController gestisce i soci, DonationController le donazioni ed ExpenseController le spese. Le dipendenze vengono iniettate tramite il Context, così i controller non sono accoppiati alle implementazioni specifiche dei repository.

Le entità stesse contengono la logica di validazione dei propri dati. Un oggetto Donation sa quali sono le regole che lo rendono valido e può verificarle autonomamente. Questo pattern si chiama Validation in Entity.

Le Entità e gli Enum

L'enum MembershipLevel definisce i livelli di appartenenza all'associazione in ordine gerarchico: BRONZE, SILVER, GOLD, GRAY e infine BANNED. L'ordine in cui le costanti sono dichiarate determina la loro posizione nella gerarchia. L'enum contiene anche metodi utili come isActive() per verificare se un livello permette l'accesso al sistema e getNextLevel() per ottenere il livello successivo nella progressione.

Una distinzione importante riguarda i metodi name() e toString() degli enum. Il metodo name() restituisce il nome esatto della costante così come è dichiarata nel codice ed è quello da usare quando si salva il valore nel database. Il metodo toString() può essere sovrascritto per restituire una rappresentazione più leggibile, adatta all'interfaccia utente.

I Repository

Le interfacce dei repository definiscono un contratto chiaro per le operazioni sui dati. Ogni metodo specifica cosa restituisce e quali eccezioni può lanciare. Una convenzione importante è che i metodi che restituiscono liste non restituiscono mai null: se non ci sono risultati, restituiscono una lista vuota.

Le implementazioni SQL dei repository usano PreparedStatement per eseguire le query. Questo approccio previene gli attacchi di SQL injection e offre prestazioni migliori rispetto alla concatenazione di stringhe. I parametri vengono indicati con il punto interrogativo nel SQL e poi associati ai valori tramite i metodi setXXX nell'ordine corretto.

Il metodo rowToX presente in ogni repository SQL è il ponte tra il mondo relazionale del database e quello ad oggetti di Java. Legge i valori dalle colonne del ResultSet e li usa per costruire l'oggetto corrispondente.

Per quanto riguarda le date, LocalDate di Java e Date di JDBC non sono direttamente compatibili. Per salvare si usa Date.valueOf() che converte LocalDate in Date, mentre per leggere si chiama toLocalDate() sull'oggetto Date ottenuto dal database. SQLite in particolare salva le date come stringhe nel formato YYYY-MM-DD.

Il Layer delle View

Il progetto usa due approcci diversi per gestire le view. Il primo è basato sulle lambda expression nel ViewController, il secondo sulla Reflection API nella classe ReflectionView.

La Reflection è la capacità di Java di ispezionare e manipolare le classi a runtime. Invece di chiamare direttamente i metodi di un oggetto, è possibile scoprire dinamicamente quali metodi esistono, invocarli e ottenere i risultati. Nel contesto delle view, questo permette di mappare automaticamente i getter di un'entità ai placeholder di un template: se un'entità ha un metodo getFirstName(), la Reflection lo trova, lo invoca e sostituisce il placeholder [firstname] nel template con il valore ottenuto.

La classe ReflectionView carica il template una sola volta nel costruttore e lo riutilizza per ogni rendering. Questo caching evita di leggere il file dal disco ogni

volta e migliora le prestazioni quando si devono renderizzare molte entità.

ViewFactory applica il Factory Pattern per centralizzare la creazione delle view. Tutte le istanze di ReflectionView vengono create come campi statici quando la classe viene caricata, quindi sono pronte all'uso senza dover ricrearle ogni volta. Il metodo make() decide quale view restituire in base ai parametri ricevuti.

Per le liste che richiedono un template composto, come l'elenco dei membri Gray, si usa un approccio che combina due template: uno per le singole righe e uno per il wrapper che contiene header e footer. Le righe vengono generate con ReflectionView e poi inserite nel wrapper.

Le lambda expression tornano utili quando servono view con parametri dinamici che non sono getter dell'entità. Per esempio, la carta di promozione richiede il livello precedente e quello nuovo, informazioni che non appartengono direttamente all'oggetto Member. In questi casi la lambda "cattura" questi parametri esterni e li usa durante il rendering.

Concetti Trasversali

BigDecimal è una classe immutabile, il che significa che i metodi come add() e subtract() non modificano l'oggetto originale ma restituiscono un nuovo oggetto con il risultato. Bisogna sempre riassegnare il valore: total = total.add(amount).

Il method reference indicato con i due punti doppi è un'alternativa più concisa alle lambda quando esiste già un metodo che fa quello che serve. Scrivere donations.forEach(this::printDonation) è equivalente a donations.forEach(d → this.printDonation(d)) ma risulta più leggibile e viene compilato in modo leggermente più efficiente.

La gestione delle eccezioni deve garantire che l'applicazione non si blocchi mai in modo imprevisto. Gli errori vengono catturati, viene mostrato un messaggio appropriato all'utente e l'applicazione continua a funzionare.