**Università degli Studi di Camerino**
**Informatica per la comunicazione digitale**
**ST1414 Modellazione e Gestione della Conoscenza**

# Relazione Progetto JBudget

**Appello:** 16/02/26
**Studente:** Filippo Verdecchia
**Matricola:** 119474
**Anno Accademico:** 2024/2025

JBudget è un'applicazione Java modulare per la gestione del bilancio familiare. L'obiettivo principale è fornire uno strumento che consenta di registrare entrate e uscite, organizzare i movimenti in categorie gerarchiche, gestire scadenze future e visualizzare statistiche sull'andamento finanziario personale.

L'applicazione è stata sviluppata in Java utilizzando JavaFX per l'interfaccia grafica e il framework Jackson per la persistenza in formato JSON. È pensata per essere facilmente estendibile, seguendo i principi di buona progettazione e separazione delle responsabilità.

## Funzionalità implementate

L'applicazione offre un insieme di funzionalità progettate per coprire le esigenze fondamentali della gestione economica familiare. Tali funzionalità sono state realizzate con attenzione alla semplicità d'uso da parte dell'utente, ma anche all'architettura sottostante, pensata per estensioni future. Le principali funzionalità implementate sono:

- **Inserimento e consultazione dei movimenti finanziari**: l'utente può registrare entrate e uscite, specificando descrizione, importo, data e tag associato.
- **Tag personalizzabili e organizzati in forma gerarchica**: i movimenti possono essere classificati mediante un sistema di tag strutturato ad albero (es. "Casa > Bollette > Gas"), utile per analisi dettagliate.
- **Gestione dei movimenti programmati**: è possibile definire pagamenti futuri ricorrenti (es. rate di un prestito), organizzati tramite un apposito piano (**RatePlan**).
- **Scadenzario**: l'interfaccia fornisce una sezione dedicata alla visualizzazione delle spese future in base alle date programmate.
- **Statistiche e confronto tra periodi**: l'utente può confrontare periodi diversi in termini di spese o entrate, filtrando per categoria o intervallo temporale.

- **Persistenza dei dati:** tutte le informazioni inserite possono essere salvate su file tramite l'apposito comando dell'interfaccia grafica e caricate manualmente all'avvio o durante l'utilizzo dell'applicazione.

L'interfaccia grafica è stata progettata per risultare chiara e funzionale, con schermate separate per ogni sezione dell'applicazione.

## Responsabilità individuate

Durante la progettazione dell'applicazione JBudget, sono state individuate e suddivise con chiarezza le responsabilità tra i vari componenti del sistema, seguendo un'architettura modulare e rispettando il principio di separazione dei compiti.

- **Model**: le contiene entità principali del dominio dell'applicazione, come Movement, ScheduledMovement, RatePlan e Tag. Ogni classe è stata progettata per rappresentare fedelmente un concetto concreto del bilancio familiare.

- **Repository**: si occupa della gestione in memoria dei dati applicativi. L'interfaccia MovementRepository e la sua implementazione MovementRepositoryImpl garantiscono un accesso centralizzato e modulare ai movimenti e ai tag.

- **Persistence**: il modulo JsonPersistence fornisce i meccanismi per il salvataggio e il caricamento automatico dei dati tramite file JSON. Tale componente è stato astratto per poter essere facilmente sostituito o esteso in futuro.

- **Statistics**: il servizio StatisticsService si occupa dell'aggregazione e del confronto dei dati, fornendo all'utente finali strumenti di analisi come il confronto tra periodi o la visualizzazione per tag.

- **View (GUI)**: tramite JavaFX e la classe JBudgetView, viene gestita l'interazione grafica con l'utente. La GUI è stata organizzata in sezioni (movimenti, scadenzario, statistiche) e progettata per essere facilmente estendibile.

Tale suddivisione è stata pensata per massimizzare la **manutenibilità del codice**, favorire il **riutilizzo** dei componenti e rendere più semplice l'aggiunta di nuove funzionalità nel tempo.

## Classi e interfacce sviluppate

L'architettura del progetto JBudget è stata costruita attorno a un insieme ben definito di classi e interfacce, ciascuna con responsabilità specifiche. Di seguito si riportano le principali componenti sviluppate:

- **Movement**: rappresenta un movimento finanziario (entrata o uscita), contenente data, descrizione, importo e riferimento al tag associato.

- **ScheduledMovement**: estende il concetto di movimento per rappresentare una transazione pianificata in una data futura. È utilizzata anche nel contesto dei piani di ammortamento.

- **RatePlan**: incapsula una serie di **ScheduledMovement** e modella, ad esempio, un piano di rimborso rateale o una sequenza di pagamenti programmati.

- **Tag / TagTree**: interfacce per la gestione delle categorie, supportano una struttura gerarchica che consente di organizzare i movimenti in sotto-categorie.

- **TagTreeImpl**: implementazione concreta dell'interfaccia **TagTree**, basata su una struttura ad albero. Supporta operazioni di creazione, recupero e navigazione tra i tag.

- **MovementRepository / MovementRepositoryImpl**: forniscono un'interfaccia e una relativa implementazione per la gestione e il recupero dei movimenti e dei tag in memoria.

- **JsonPersistence**: gestisce il salvataggio e il caricamento persistente dei dati attraverso file JSON, utilizzando la libreria Jackson.

- **StatisticsService**: fornisce funzionalità di analisi dei dati, come il confronto tra periodi temporali, la somma degli importi per tag e la generazione di statistiche aggregate.

- **JBudgetView**: rappresenta la vista principale dell'interfaccia grafica (GUI), costruita con JavaFX. Gestisce l'interazione dell'utente con l'applicazione e visualizza i dati ottenuti dagli altri moduli.

## Organizzazione dei dati e persistenza

Nel progetto JBudget, la gestione dei dati e della loro persistenza è stata affrontata con particolare attenzione alla modularità e alla riutilizzabilità. I dati vengono mantenuti in memoria durante l'esecuzione dell'applicazione tramite strutture dedicate (**MovementRepositoryImpl**, **TagTreeImpl**), e possono essere salvati su file tramite l'interfaccia grafica per essere recuperati in futuro.

Per realizzare questo comportamento, è stato utilizzato il processo di serializzazione: ogni oggetto (come un movimento o un tag) viene convertito in una rappresentazione testuale JSON, leggibile e facilmente trasferibile. L'operazione inversa, detta deserializzazione, consente di ricostruire automaticamente gli oggetti originali a partire dal contenuto del file JSON.

Tali operazioni vengono gestite dalla classe **JsonPersistence**, che sfrutta il framework **Jackson**, una delle librerie più utilizzate in Java per la gestione della serializzazione in formato JSON.

Le informazioni persistite includono:

- l'elenco completo dei movimenti (**Movement**), ciascuno con data, descrizione, importo e tag associato;
- l'elenco dei movimenti futuri o programmati (**ScheduledMovement**);
- la struttura ad albero dei tag, gestita tramite **TagTreeImpl**, che consente la classificazione gerarchica delle spese.

Il salvataggio dei dati avviene in un file denominato **dati.json**, generato dall'applicazione durante l'esecuzione e localizzato nella directory del progetto. Questo file viene aggiornato quando l'utente utilizza la funzione di salvataggio dall'interfaccia grafica. All'avvio successivo dell'applicazione, i dati possono essere ripristinati tramite il comando di caricamento, garantendo la continuità dello stato dell'applicazione tra sessioni.

La persistenza è stata progettata in modo astratto: l'uso di interfacce permette, in futuro, di sostituire il salvataggio su file con altre tecnologie (es. database, archiviazione cloud) senza modificare il resto dell'applicazione. Quindi i moduli sono pensati per essere indipendenti e intercambiabili.

La scelta di utilizzare una persistenza basata su file JSON è stata motivata dalla semplicità, dalla leggibilità del formato e dalla possibilità di migrare in futuro verso soluzioni più avanzate, come database relazionali o servizi cloud, senza impattare sulla logica applicativa.

## Scenari di utilizzo

L'applicazione JBudget è stata progettata per adattarsi a molteplici scenari d'uso all'interno del contesto domestico, ma è strutturata in modo tale da poter essere adottata anche in ambiti più ampi. Di seguito si riportano alcuni esempi di utilizzo realistico:

**• Gestione del bilancio personale o familiare**

L'utente può utilizzare JBudget per registrare le proprie spese quotidiane (es. spesa alimentare, bollette, carburante) e monitorare le entrate (stipendi, rimborsi). I tag gerarchici consentono una classificazione dettagliata, e le statistiche aiutano a evidenziare aree di maggiore spesa.

**• Pianificazione di spese future**

Grazie ai movimenti programmati e al sistema di scadenze, l'utente può pianificare pagamenti futuri come affitti, rate di prestiti, abbonamenti mensili o trimestrali. Lo scadenzario fornisce una visione organizzata delle prossime uscite.

**• Analisi dell'andamento economico**

Con la sezione "Statistiche" l'utente può confrontare periodi diversi (es. un mese rispetto al precedente, o lo stesso periodo dell'anno prima) per valutare miglioramenti o peggioramenti della propria gestione finanziaria.

**• Supporto per il risparmio e il controllo delle spese**

Attraverso il monitoraggio continuo delle uscite, è possibile individuare aree dove ridurre i costi. Ad esempio, l'utente può osservare quanto ha speso in un anno per la categoria "ristoranti" o "tempo libero" e decidere se intervenire.

**• Utilizzo collaborativo (estensione futura)**

In un contesto familiare, l'applicazione può essere condivisa tra più membri (ad es. genitori e figli), ognuno dei quali contribuisce al bilancio familiare. In questo caso, l'aggiunta di un sistema multi-utente rappresenta un'estensione naturale e coerente.

## Estendibilità del progetto

Uno degli obiettivi principali durante la progettazione di JBudget è stato quello di garantire l'estendibilità dell'applicazione, in modo da renderla pronta per implementazioni future senza la necessità di riscrivere la logica già esistente.

### Modularità e interfacce

Il progetto è strutturato in moduli distinti (Model, Repository, Persistence, Statistics, View), ciascuno dei quali interagisce con gli altri solo tramite interfacce ben definite. Questo approccio favorisce la sostituzione o l'aggiunta di nuove funzionalità senza compromettere il funzionamento globale.

Ad esempio:

- l'interfaccia **MovementRepository** può essere implementata in futuro per supportare un database SQL o un'API REST;
- la GUI basata su **JBudgetView** può essere affiancata o sostituita da una versione web o mobile, mantenendo intatta la logica sottostante.

### Nuove funzionalità possibili

L'architettura è stata pensata per consentire l'integrazione di:

- **nuovi tipi di movimenti**, come entrate periodiche, abbonamenti o spese fisse;
- **grafici dinamici** o esportazioni dei dati in PDF, CSV o Excel;
- **sistemi di autenticazione multi-utente**, per distinguere i bilanci tra componenti della famiglia;
- **sincronizzazione cloud**, per usare JBudget da più dispositivi con dati condivisi;
- **sistemi di notifica o reminder**, legati alle scadenze dei movimenti programmati.

### Persistenza astratta

La persistenza dei dati, attualmente basata su file JSON locali, può essere facilmente sostituita o affiancata da altri sistemi grazie alla presenza dell'interfaccia **JsonPersistence**. Ciò consente, ad esempio, l'adozione futura di una sincronizzazione online o l'utilizzo di tecnologie di storage più avanzate.

**Scalabilità futura**

Infine, la separazione tra dati, logica e interfaccia, insieme all'uso di standard come JSON e JavaFX, rende l'applicazione adatta ad essere estesa anche in contesti professionali o aziendali, migliorando l'investimento progettuale in termini di riutilizzabilità e sostenibilità del codice.

## Principi di progettazione adottati

Durante la progettazione dell'applicazione JBudget sono stati seguiti alcuni principi di buona progettazione orientata agli oggetti, con particolare attenzione ai principi SOLID, al fine di migliorare la manutenibilità, l'estensibilità e la chiarezza del codice.

In particolare, il Principio di Responsabilità Singola (SRP) è stato applicato separando chiaramente le diverse responsabilità del sistema. Ad esempio, le classi del dominio (come Movement e Tag) si occupano esclusivamente di rappresentare i dati, mentre la gestione dei movimenti è delegata al MovementRepository e la persistenza è affidata al componente JsonPersistence. Questo approccio evita classi troppo complesse e facilita la modifica o l'estensione di singole parti del sistema.

Il Principio Open/Closed (OCP) è rispettato grazie all'uso di interfacce e alla progettazione modulare. Ad esempio, l'interfaccia MovementRepository consente di introdurre nuove modalità di gestione dei dati (come l'uso di un database o di un servizio remoto) senza dover modificare il codice che utilizza il repository. Analogamente, nuove funzionalità possono essere integrate aggiungendo nuovi servizi o componenti senza alterare la logica esistente.

Infine, il Principio di Dependency Inversion (DIP) è applicato facendo dipendere i componenti di alto livello da astrazioni e non da implementazioni concrete. Questo rende il sistema più flessibile e riduce l'accoppiamento tra i moduli, facilitando eventuali evoluzioni future del progetto.

## Limiti dell'attuale implementazione

L'applicazione JBudget rappresenta una prima versione funzionale del sistema e presenta alcuni limiti, individuati consapevolmente durante la progettazione. In particolare, l'applicazione è pensata per un singolo utente e non prevede attualmente meccanismi di autenticazione o gestione multi-utente.

Inoltre, la persistenza dei dati è basata su file locali in formato JSON e non include funzionalità di sincronizzazione tra dispositivi diversi. Anche le funzionalità statistiche, pur consentendo confronti tra periodi e categorie, potrebbero essere estese con grafici più avanzati o analisi predittive. L'architettura adottata consente comunque di integrare queste funzionalità in versioni successive senza modifiche strutturali significative.

## Considerazioni finali

L’applicazione JBudget rappresenta una prima versione funzionale di un sistema per la gestione del bilancio familiare. Tutte le funzionalità principali richieste nella specifica sono state implementate: gestione dei movimenti, uso di tag gerarchici, scadenzario, statistiche e confronto tra periodi. L’applicazione offre un’interfaccia semplice, ma completa, adatta all’utilizzo quotidiano da parte di un utente generico.

Il sistema è stato pensato per essere modulare e facilmente estendibile. In futuro, potrebbe essere adattato per supportare più utenti, permettere la sincronizzazione dei dati su cloud, o introdurre funzionalità avanzate come notifiche di scadenze, esportazione in PDF o visualizzazione tramite grafici.

La struttura dei dati, basata su JSON e supportata da interfacce, rende possibile la migrazione verso altre soluzioni di persistenza come database relazionali o NoSQL. Anche la parte grafica, pur sviluppata in JavaFX, potrebbe essere riorganizzata per supportare l’uso in contesti mobile o web.

Nel complesso, il progetto costituisce una base solida su cui costruire versioni successive dell’applicazione, con funzionalità avanzate e un’interfaccia utente sempre più evoluta.