Relazione – Assignment 3

Clean Code

Componenti del gruppo

Nome	Matricola
Bergamin Samuele	844861
Andrea Ciacci	899883
Baggio Tommaso	912356

Contents

Assignment scelto

L'assignment scelto è **Clean Code**. L'attività prevede la selezione e l'analisi di **5 classi eterogenee** appartenenti a un progetto software.

Repository di lavoro del gruppo

Link Repository

1 Repository analizzata

https://github.com/RishabhS66/Expense-Management-Software-Backend

2 Descrizione del progetto

Il progetto scelto è il backend di un software per l'automazione dei processi di compilazione e inoltro delle richieste di rimborso spese da parte dei dipendenti di un'azienda.

Il progetto è implementato interamente in **Java**, utilizzando un'architettura **multistrato** basata su un modello **MVC esteso**. Sono presenti le tipologie di classi tipiche di tale architettura:

- **Controller**: REST controller utilizzati per il routing delle richieste ai livelli sottostanti.
- **Service**: classi dedicate all'implementazione della logica di business.
- **DAO/Repository**: classi per separare dati e logica applicativa e per l'accesso al database.
- **Entity**: modellazione dei dati come oggetti persistenti, derivati dalla modellazione delle tabelle del DB (ORM).
- **Util**: insieme eterogeneo di classi di supporto (es. conversione JSON $\leftrightarrow$ oggetti Java).

3 Classi selezionate

Per soddisfare i requisiti dell'assignment sono state scelte 5 classi eterogenee, sia per scopo sia per livello dell'MVC esteso di appartenenza:

- **ExpenseController.java** (Link)
  Classe RestController: gestisce il mapping degli URL e le richieste HTTP relative alle spese.
  197 righe di codice.
- **Address.java** (Link)
  Classe `@Entity`: rappresenta in Java la tabella di database `address` (mappatura ORM).
  208 righe di codice.
- **ExpenseService.java** (Link)
  Classe `@Service`: contiene la logica applicativa e invoca i metodi del DAO/repository `expenses`, che esegue le query sul database per recuperare i dati dalla tabella `expenses`.
  143 righe di codice.

- **GenericJsonAttributeConverter.java** (Link)
  Classe utility (converter JPA): serializza un oggetto in una stringa JSON per salvarlo nel database e deserializza una stringa JSON per ricostruire l'oggetto.
  199 righe di codice.

- **GlobalExceptionHandler.java** (Link)
  Classe di gestione eccezioni (handler globale): estende `ResponseEntityExceptionHandler` e definisce metodi `@ExceptionHandler` per gestire eccezioni dell'applicazione (es. accesso negato, errori PostgreSQL), restituendo risposte HTTP appropriate.
  87 righe di codice.

4 Analisi di conformità a Clean Code (per classe)

4.1 ExpenseController.java

- L'indentazione non rispetta pienamente i principi del Clean Code: sono presenti parentesi annidate difficili da leggere e righe troppo lunghe in orizzontale (oltre 90 colonne).
- Vengono ripetuti frequentemente i seguenti controlli, che potrebbero essere estratti in metodi dedicati:
  - `emp.getId() == expense.getEmployee().getId()`
  - `expense.getProject().getProjectManager().getId() == emp.getId()`
- Alcuni nomi di variabili non sono abbastanza descrittivi; in alcuni casi sono singole lettere (es. variabili di tipo `Expense` chiamate semplicemente `e`).
- Alla riga 158 è presente *commented-out code*.

Funzioni (pinch principles)

- **Lunghezza delle funzioni**: rispettata.
- **Nameable**: le funzioni sono descrittive nel nome.
- **Insulated**: le funzioni hanno meno di 4 parametri.
- **Homogeneous**: le funzioni contengono istruzioni omogenee e allo stesso livello di astrazione.
- **Contextual**: il contesto del controller è rispettato.
- **Pure**: le funzioni dipendono dai loro argomenti.
- **Command–Query Separation**: rispettata.

4.2 ExpenseService.java

- Indentazione non ottimale e lunghezza orizzontale eccessiva (oltre 90 colonne).
- Presente *commented-out code* alle righe 139 e 140.
- Argomenti di funzione denominati con singola lettera, riducendo la leggibilità.
- La funzione `getExpenseTotal` non restituisce solo il totale, ma un report completo: il nome non è rappresentativo della logica.

Funzioni (pinch principles)

- **Lunghezza delle funzioni**: rispettata.
- **Nameable**: le funzioni sono descrittive nel nome.
- **Insulated**: le funzioni hanno meno di 4 parametri.
- **Homogeneous**: le funzioni contengono istruzioni omogenee e allo stesso livello di astrazione.
- **Contextual**: il contesto del service è rispettato.
- **Pure**: le funzioni dipendono dai loro argomenti.
- **Command–Query Separation**: rispettata.

4.3 Address.java

- Sono presenti molti commenti: sia Javadoc sia *noise comments*. Secondo le indicazioni del Clean Code, non essendo un'API pubblica, i commenti Javadoc risultano superflui e candidati alla rimozione.
- L'indentazione del metodo `toString()` produce righe troppo lunghe in orizzontale (oltre 90 colonne).

Funzioni (pinch principles)

- **Lunghezza delle funzioni**: rispettata.
- **Nameable**: i nomi rispettano la struttura standard per una classe Java.
- **Insulated**: le funzioni accettano al massimo 1 parametro; tuttavia, il costruttore ne accetta 9.
- **Homogeneous**: le funzioni contengono istruzioni omogenee e allo stesso livello di astrazione.
- **Contextual**: il contesto è rispettato.
- **Pure**: le funzioni dipendono dai loro argomenti.
- **Command–Query Separation**: rispettata.

4.4 GenericJsonAttributeConverter.java

- Nome della classe poco efficace: `Generic` non aggiunge informazione utile.
- Righe troppo lunghe in orizzontale (oltre 90 colonne).
- Indentazione delle parentesi non ottimale.
- Presenti diversi *noise comments* e *redundant comments*.

Funzioni (pinch principles)

- **Lunghezze**: funzioni generalmente brevi.
- **Nameable**: nomi di variabili e argomenti poco espressivi, che rendono più faticoso capire lo scopo delle funzioni. Esempio:
  - `public X convertToEntityAttribute(String dbData)`

- **Nameable (coerenza)**: il nome `convertToEntityAttribute` non è pienamente coerente con il reale comportamento della funzione.
- **Insulated**: rispettato.
- **Homogeneous**: istruzioni omogenee e allo stesso livello di astrazione.
- **Contextual**: il contesto è rispettato.
- **Pure**: le funzioni dipendono dai loro argomenti.

4.5 GlobalExceptionHandler.java

- Indentazione non conforme ai principi del Clean Code: parentesi annidate difficili da leggere e righe troppo lunghe (oltre 90 colonne).
- La costruzione di `ResponseEntity<>` è ripetuta in quasi tutti i metodi; sarebbe preferibile estrarla in un metodo di supporto.
- Presente un *redundant comment* in cima alla classe che ripete solo il nome della classe.

Funzioni (pinch principles)

- **Lunghezza delle funzioni**: rispettata.
- **Nameable**: la funzione `handleExpiredJwtException` non tratta solo JWT scaduti, ma JWT in generale; il nome risulta quindi fuorviante.
- **Insulated**: le funzioni accettano al massimo 2 parametri; tuttavia, viene citato un costruttore con 9 parametri.
- **Homogeneous**: istruzioni omogenee e allo stesso livello di astrazione.
- **Contextual**: il contesto è rispettato.
- **Pure**: le funzioni dipendono dai loro argomenti.
- **Command–Query Separation**: rispettata.

5 Modifiche effettuate per migliorare la conformità a Clean Code

5.1 ExpenseController.java

ExpenseController.java (Refactoring)

- Migliorata l'indentazione delle funzioni per ridurre la *horizontal length* e aumentare la leggibilità.
- Indentate le parentesi in modo più efficace.
- Nella classe `ExpenseService` sono stati aggiunti i metodi `isOwner(emp, expense)` e `isManager(emp, expense)` per evitare la ripetizione eccessiva dei controlli:
  - `emp.getId() == expense.getEmployee().getId()`
  - `expense.getProject().getProjectManager().getId() == emp.getId()`
- Rimosso il codice commentato alla riga 158.

5.2 ExpenseService.java

ExpenseService.java (Refactoring)

- Migliorata l'indentazione.
- Aggiunte le funzioni di supporto `isManager` e `isOwner` per evitare ripetizioni nel controller e mantenere `ExpenseController` il più leggero possibile, in linea con l'architettura MVC estesa.
- Rinominata la funzione `getExpenseTotal` in `getExpenseReports`.

5.3 Address.java

Address.java (Refactoring)

- Migliorata l'indentazione.
- Rinominate le variabili `addressLine1` e `addressLine2` in `firstAddress` e `secondAddress`, e aggiornati i relativi getter e setter.
- Il costruttore ora accetta una struttura dinamica `AddressData`, contenente i dati necessari per inizializzare correttamente l'oggetto `Address`.
- Rimossi i *noise comments* intrusivi e javadocs comments.

5.4 GlobalExceptionHandler.java

GlobalExceptionHandler.java (Refactoring)

- Migliorata l'indentazione.
- Creata la funzione `buildResponse` per centralizzare la costruzione delle `ResponseEntity`.
- Rinominata la funzione `handleExpiredJwtException` in `handleJwtException`.
- Rimosso il commento ridondante iniziale.

5.5 GenericJsonAttributeConverter.java

JsonAttributeConverter.java (Refactoring)

- Migliorata l'indentazione.
- Rinominata la classe in `JsonAttributeConverter`.
- Rimossi commenti *noise* e commenti che descrivevano il funzionamento ovvio del codice.
- Modificati i nomi di funzioni e variabili per renderli più espressivi.
- Rinominato `JsonTypeLike` in `JsonTypedWrapper`.
- Rinominato `convertToEntityAttribute` in `converDBtColumnIntoAttributex`