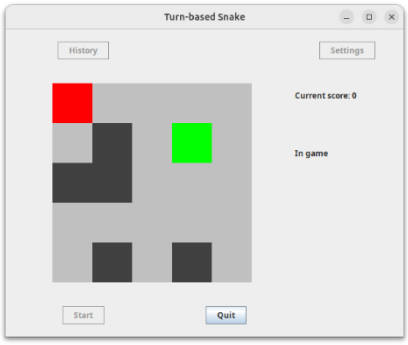
Turn-Based Snake

Autore: Davide Zhang Matricola: 7176748 Data: 13-09-2025

1 Introduzione

“Turn-based Snake” è un semplice minigioco in cui controlli un serpente attraverso i tasti freccia per muoverlo su una mappa rettangolare. La mappa può ospitare quattro tipi di celle: la cella è verde se è occupata dal serpente, rossa se è occupata da una mela, grigio scura se è occupata da un ostacolo e grigio chiara se invece è libera. L’obiettivo del gioco è quello di accumulare più punti possibili raggiungendo celle rosse mentre si evitano quelle grigio scure.



2 Contents

3 Caratteristiche del software

L'UI dell'applicazione è suddiviso in quattro schede:

- **Scheda benvenuti**: contiene il nome del software in primo piano e tre bottoni che servono per cambiare scheda visualizzata.

- **Scheda storia**: contiene un elenco che mostra la storia delle partite. Ci sono due bottoni che consentono di operare sulla storia. Una consente di eliminare tutto mentre l'altra consente di eliminare il record selezionato.

- **Scheda configurazioni**: è la scheda che permette di personalizzare le partite. Abbiamo in primo piano la lista delle configurazioni disponibili, e sotto, un'interfaccia che consente di eliminare, rinominare, usare e creare nuove configurazioni. Qui abbiamo un bottone per eliminare la configurazione selezionata, una per applicarla al gioco, e un'altra per modificarne il nome attraverso l'uso combinato con un campo testo. Questo campo testo si riempie col nome originale dell'impostazione al momento della selezione. Infine, nella parte inferiore abbiamo una serie di campi testo che permettono di configurare nuove impostazioni di gioco. Ognuna di queste caselle è accompagnata da una etichetta che ne spiega lo scopo. L'applicazione effettua un controllo dell'input in due step. Prima di tutto non fa inserire caratteri illegali (ad esempio permettendo solo cifre), e quindi controlla se valori inseriti sono legali o meno, lanciando un popup che avverte l'utente dell'errore e ripristinando valori precedenti nel caso.

- **Scheda partita**: il cuore dell'applicazione. A destra abbiamo due etichette che permettono di seguire lo stato del gioco, cioè il punteggio raggiunto e se è in corso una partita. Nella parte inferiore della scheda ci sono due bottoni per entrare e uscire da una partita. Il bottone "start" è disattivato al lancio del software, e viene abilitato non appena una impostazione verrà selezionata nella scheda configurazioni. Durante una partita i bottoni "start", "history" e "settings" sono disabilitate per evitare interferenze indesiderate. Questi bottoni saranno riattivati di nuovo a conclusione partita per click del bottone "quit" o per fine partita. La sottofinestra "canvas" mostra lo stato della mappa al procedere della partita.

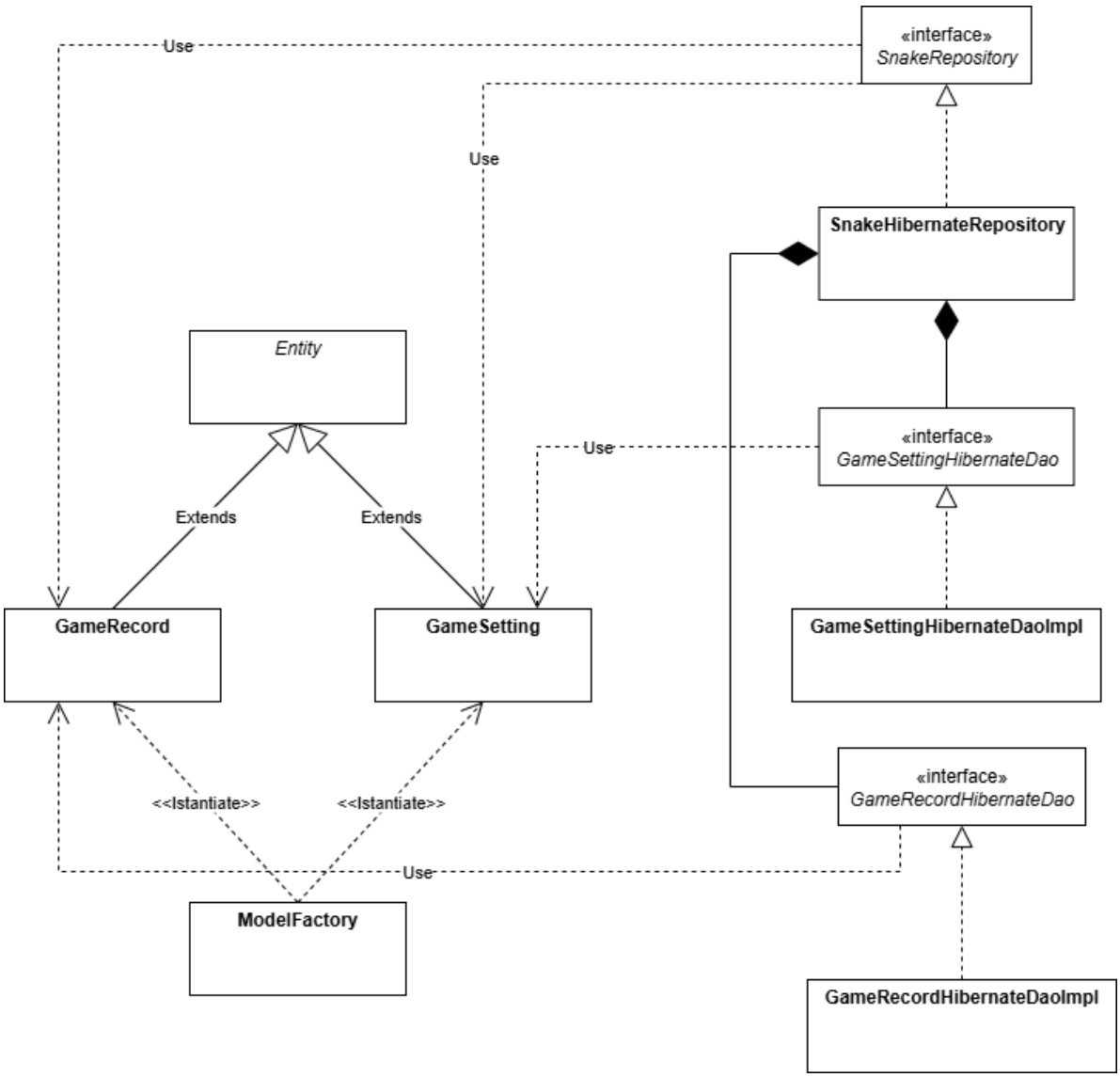
Figura 1: Schede applicazione

4 Struttura del software

L'applicazione è suddivisa in tre pacchetti secondo il pattern architetturale model-view-presenter:

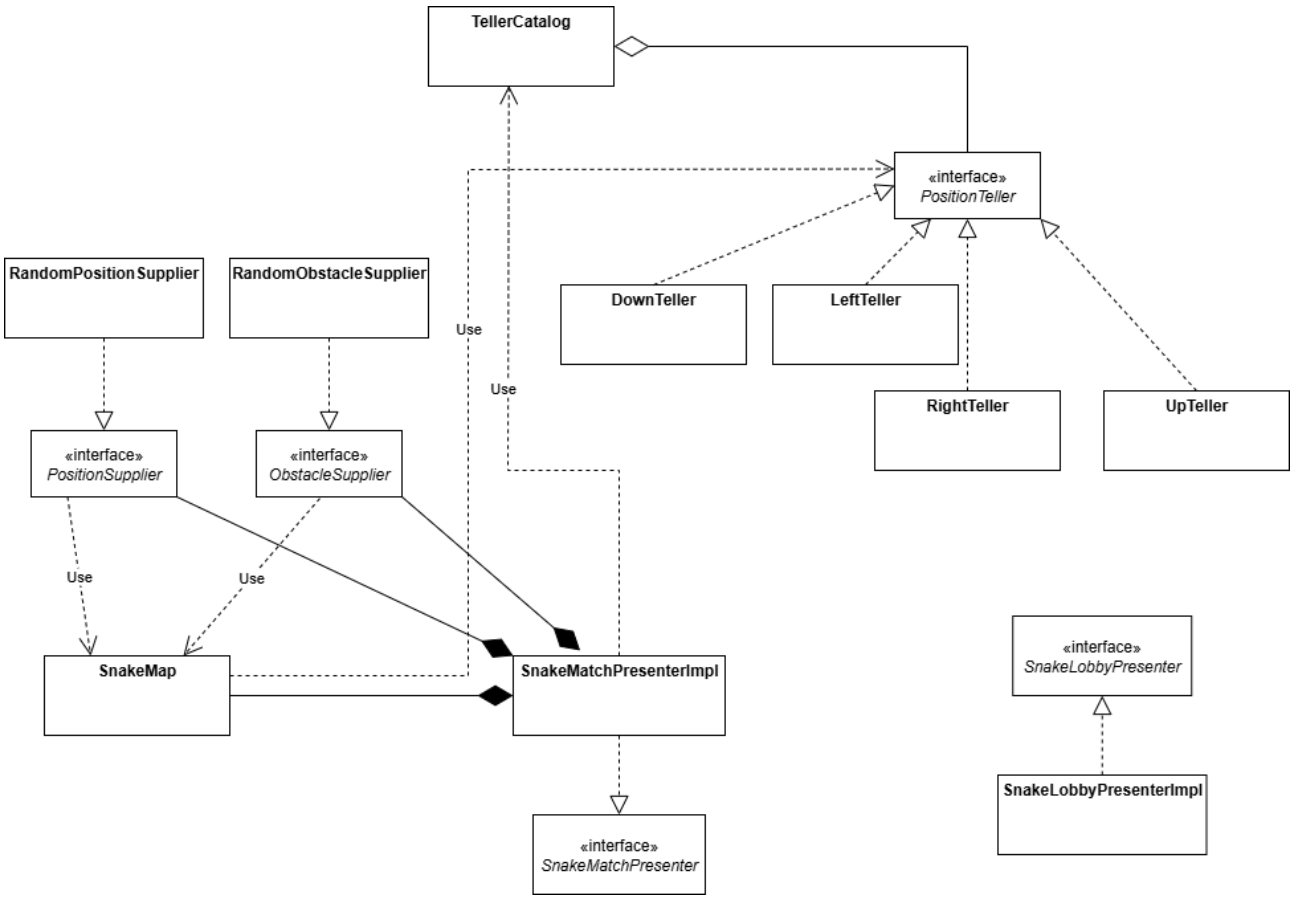
4.1 Model

Il pacchetto **model** gestisce le entità e le comunicazioni col database. Usa Hibernate come ORM per effettuare operazioni sul database in maniera semplice e pulita. Il pattern DAO (Data Access Objects) viene usato in modo che le operazioni di accesso a mysql avvengano in un'unica classe per entità. Il pacchetto ha anche una classe dedicata per la creazione di oggetti entità, ModelFactory. ModelFactory ha due static factory method che permettono di generare automaticamente UUID casuali per oggetto entità creato. Infine, l'altro pattern importante nel pacchetto è Repository che aiuta a realizzare la Dependency Inversion Principle e favorire riusabilità ed estendibilità del software.



4.2 Presenter

Il pacchetto **presenter** è responsabile per la gestione della logica del gioco, per l'inoltrazione delle operazioni dal view al model e aggiornamento del view in seguito alle operazioni. Al cuore del modulo ci sono le classi di SnakeMatchPresenterImpl e SnakeLobbyPresenterImpl, una responsabile per la logica del gioco, l'altra per la logica di tutte le altre schede. Responsabile del calcolo della posizione della testa del serpente sono le classi teller, introdotte per evitare istruzioni switch su enumerazioni che non possono essere coperte al cento per cento. Per rendere i teller più accessibili ho definito una classe TellerCatalog per raccoglierne una istanza per tipo, simulando un po' una enumerazione. ObstacleSupplier e PositionSupplier sono delle interfacce per la generazione di oggetti sulla mappa. Sono presenti due implementazioni concrete nel pacchetto che generano oggetti in posizioni random. Lo SnakeMap è una classe usata da SnakeMatchPresenter per memorizzare tutte le informazioni riguardanti una partita in corso. Due interfacce, SnakeLobbyPresenter e MatchPresenter consentiranno a classi della view di accedere alle funzionalità dello strato in maniera sicura e centralizzata.



4.3 View

Il pacchetto **view** infine implementa l'interfaccia utente dell'applicazione tramite java swing. La classe più importante qui è SnakeWindowView, implementante SnakeView, l'interfaccia usata dai presenter per effettuare aggiornamenti. SnakeView richiede due metodi updateMatch e updateLobby chiamati dai presenter nel momento giusto per aggiornare schede lobby o mappa ed elementi collegati. SnakeWindowView estende JFrame e usa un layout di tipo CardLayout, che può essere pensato come una pila di carte, ciascuna dei quali corrisponde ad una scheda. La scheda più in su è quella dei benvenuti seguita dalle altre secondo l'ordine di aggiunta. Possiamo passare da una carta all'altra usando il metodo show col nome della carta. Ogni carta qua è implementata come classe separata per evitare che SnakeWindowView diventi gigantesca. Le classi delle schede estendono JPanel e sono incaricate di creare i componenti visuali della scheda, posizionarli e inizializzare opportuni listener. I metodi per l'inizializzazione dei componenti sono stati refattorizzati nella classe ComponentInitializer. Le classi panel mettono inoltre a disposizione il metodo refresh per consentire l'aggiornamento della scheda in corrispondenza al cambiamento del model sottostante. Tra le classi Panel di particolare interesse è lo SnakeMatchPanel che è la scheda contenente la finestra del gioco. È responsabile di reagire ai comandi del giocatore e aggiornare in maniera reattiva la mappa. Questo viene effettuato attraverso diversi listener attaccati ai bottoni. In particolare, al premere del bottone "start" viene aggiunto un KeyListerner alla finestra del gioco che permette di registrare eventi di pressione dei tasti freccia e delegare al presenter per effettuare l'azione corrispondente. Infine, SnakeCanvas si occupa unicamente di disegnare la mappa coi dati forniti dal presenter e aggiornarla su richiesta.

5 Progetto Maven e GitHub Workflow

Il pom.xml è suddiviso principalmente in cinque parti:

- In dependencyManagement vengono risolti le versioni delle dipendenze e se necessario importato la Bill Of Material per dipendenze che richiedono le stesse versioni.
- In dependencies si importano le dipendenze definite in dependencyManagement, assegnando a ciascuna uno scope appropriato.
- In pluginManagement ho definito le versioni e configurazioni dove necessario.
- In plugins ho effettivamente abilitato i plugin definite in pluginManagement e fornito istruzioni più dettagliate riguardo alle loro configurazioni.
- Infine, in profile ho definito i profili per mutation testing e code coverage.

5.1 Dipendenze

Nel progetto ho importato diverse librerie sia per uso nei test che nel codice sorgente. Commons-cli di Apache viene usato nella classe main per effettuare il parsing di eventuali argomenti da riga di comando. L'uso di questa libreria è abbastanza semplice e consiste in tre fasi: la definizione del CLI attraverso addOption, il parsing dello stesso e poi il suo uso attraverso i metodi hasOption e getOptionValue. Le dipendenze log4j sono state importate per permettere a Hibernate di mostrare i log di accesso a mySQL. Consistono in log4j-core, log4j-api e log4j-slf4j2-impl il quale è la dipendenza che fa da binding per permettere a Hibernate di usare la funzionalità di log4j2. Log4j è stato configurato attraverso due file properties, una per main una per test. Hibernate logga i messaggi per differenti categorie e livelli che include le categorie citate nei file properties, quali org.hibernate.SQL, org.hibernate.orm.jdbc.bind, org.hibernate.orm.jdbc.extract e org.hibernate.orm.jdbc.batch. Queste categorie vengono usate per registrare quattro tipi di messaggi: istruzioni SQL eseguite con JDBC, i parametri passati alle chiamate per JDBC, i valori estratti da JDBC e relativi all'esecuzione in batch. La libreria dei testcontainers per è invece usata solo nei unit test per testare le classi DAO. Mysql-connector-j serve come driver JDBC per mysql e verrà usato da hibernate-core per comunicare col database. Infine, per quanto riguarda altre dipendenze per il testing ho incluso junit, assertj-swing-junit per ui testing e asserzioni più fluenti e mockito-core per la creazione dei mock mirati a isolare i SUT.

5.2 Plugin e Maven build

Oltre ai plugin già inclusi, ho aggiunto diversi plugin per poter effettuare varie operazioni nella build. La build userà java 11. Ecco la lista dei plugin in più aggiunti e per cosa vengono usati:

- build-helper-maven-plugin: questo plugin viene usato per aggiungere cartelle sorgenti per Integration testing ed end-to-end testing attraverso il goal add-test-source, legato alla fase generate test-sources nel default lifecycle. I test contenuti saranno poi eseguiti nella fase integration-test separatamente grazie al plugin maven-failsafe-plugin.
- maven-assembly-plugin: usato per generare il file jar eseguibile come stand-alone. È legato alla fase package.
- exec-maven-plugin: questo plugin viene usato per testare che il jar generato dal plugin precedente possa essere effettivamente eseguito. È legato alla fase integration-test per sfruttare il docker mysql presente in questa fase.
- jacoco-maven-plugin: viene configurato per escludere la classe main, le classi dei entities o altre classi che non contengono molta logica. Il plugin viene attivato col corrispondente profilo.
- coveralls-maven-plugin: usato per inviare a coveralls dati relativi alla copertura del codice calcolati da jacoco.
- pitest-maven: usato per effettuare mutation testing. Vengono testate principalmente le classi del livello presenter, che implementano la logica business del programma.

- sonar-maven-plugin: plugin necessario per comunicare con sonar cloud e ottenere risultati relativi alla qualità del codice.
- docker-maven-plugin: usato principalmente per far partire il docker contenente il database mySQL prima dei test di integrazione ed end-to-end e farlo terminare dopo la loro conclusione. Viene configurato con un timer di quindici secondi in modo che la sua inizializzazione possa essere completata prima della partenza dei test.

Il progetto è infine configurato con una serie di proprietà nella sezione properties. Servono a specificare la versione del compilatore usato da Maven, le classi da escludere per il coverage in sonarCloud, e il path per i file conteneti i risultati dei test. La proprietà argLine serve per aggiungere ulteriori argomenti al JVM, nel nostro caso viene usato per aggiungere un argomento che permette a java swing di accedere per reflection alle classi di java.util ed evitare warning nei log.

5.3 GitHub workflow

Il workflow è attivato ogni volta che c'è un pull request o un push sul main branch in GitHub. Il workflow viene su ubuntu-latest, con java 17 distribuzione oracle. Gli step del workflow sono:

1. Clonazione del repository sul server di GitHub.
2. Inizializzazione del contenitore con sistema operativo e distribuzione più versione di java corretta.
3. Recuperazione i pacchetti Maven e SonarQube dalla cache con chiavi appositi.
4. Esecuzione build Maven con profili code-coverage e mutation-testing attivati. Xvfb-run serve per eseguire i UI test nell'ambiente del workflow di GitHub.
5. Caricamento della copertura a coveralls e collegamento con SonarCloud per scansione qualità codice.
6. Archiviazione dei risultati dei test.

Per mantenere segreti i token usati per accedere a coveralls e sonarCloud viene fatto uso di due repository secrets, SONAR_TOKEN e COVERALLS_REPO_TOKEN. I repository secrets possono essere settati su GitHub nella pagina apposita e usati nel file yaml del workflow attraverso la sintassi ${{secrets.NAME_OF_SECRET}}.

6 Test

I test, come il codice sorgente sono strutturati in tre pachetti: model, presenter e view. Ogni test verifica una classe residente nello stesso pacchetto.

6.1 Unit test

Gli unit test sono stati costruiti seguendo la logica del TDD. Quando è opportuno è stato usato il mocking per garantire massima indipendenza tra i unit tests. Un'eccezione a questa regola sono i test dei DAO concreti che vengono abbinati al dataset reale (nel docker container). Questo deriva dal fatto che è impratico effettuare il mocking di dipendenze

esterne, che spesso hanno un comportamento troppo complesso per essere compreso e ricostruito completamente. Dall'altra parte i database in-memory, più semplici, non permetteno di testare il comportamento del database reale. La soluzione a questo problema è semplicemente usare il database vero attraverso i testcontainer. Per quanto riguarda gli UI test in view, SnakeWindowViewTest raccoglie test riguardanti l'aggiornamento dell'UI, inizializzazione dei componenti visuali e il passaggio tra le schede. In particolare, per testare la logica di visualizzazione della mappa nel gioco ho usato la classe ScreenshotTaker che permette di acquisire l'immagine mostrata dalla finestra in seguito ad un'azione dell'utente, ed effettuare le verifiche necessarie. Le asserzioni sull'immagine catturata controllano pixel per pixel il colore con un ragionamento basato su celle. Per i test che riguardano l'aggiornamento delle schede mi sono concentrato a confermare che hanno gli effetti desiderati sui componenti come scritte giuste per le etichette e contenuti corretti per le liste. I test case per le singole schede controllano il corretto funzionamento di elementi interattivi come la corretta delega alle classi presenter o la modifica dello stato degli altri elementi. Infine, abbiamo un test case parametrizzato per la scheda delle configurazioni. Questo permette di ridurre la duplicazione di codice nel test case che si ha per i controlli in input delle caselle testo. Siccome ho usato junit 4 ho dovuto estrarre i test interessati in una classe separata.

6.2 Integration ed end-to-end Test

Nei integration test ho principalmente testato la corretta interazione tra i diversi livelli dell'applicazione. Il pacchetto presenter contiene test case dedicati a testare le interazioni tra il livello model e il livello presenter. Mentre il pacchetto view contiene test dedicati a verificare l'integrazione tra il livello view e quello presenter. Quando c'è bisogno si creano mock per classi di livelli non coinvolti nei test ma comunque necessari come dipendenza. Siccome gli integration e i end-to-end test sono eseguiti in un momento della build in cui è presente un mysql container non ci sarà bisogno di usare la libreria dei testcontainer. Per quanto riguarda i test end-to-end ho testato in maniera esaustiva le funzionalità degli elementi principali nelle schede incluso la finestra del gioco dal punto di vista dell'utente. In particolare, per testare le funzionalità della mappa ho scritto alcuni script di controllo del serpente che permette di simulare tutti i tipi di eventi che il giocatore può incontrare durante una partita e confermare gli effetti desiderati. La necessità di questi script deriva dal fatto che gli oggetti appaiono in maniera randomica sulla mappa, e i test devono comunque simulare gli eventi in maniera deterministica. Lo stesso problema esiste nei test di integrazione tra il view e il matchPresenter. in quel caso la soluzione è stata l'uso del setter per cambiare la strategia di generazione degli ostacoli in una strategia deterministica. Negli unit test questo stesso problema non c'è perché possiamo semplicemente fare uso dei mock.

7 Esclusioni SonarQube

L'annotazione @SuppressWarnings("ruleID") permette di escludere l'applicazione della regola citata all'elemento annotato. Tuttavia, Eclipse non riconosce i token forniti con questo metodo e quindi va a segnalarli come non supportati. Ho escluso le seguenti regole:

- java:S2699 – ho escluso questa regola nei test che usano assertJSwing. Questa regola si attiva perché trova test che non contengono asserzioni. Tuttavia, questo si tratta di un falso positivo in quanto le asserzioni in realtà ci sono ma sono in una forma che sonarCloud non riesce a vedere, tramite metodi in assertJSwing.
- java:S3577 – questa regola impone una convenzione nella denominazione dei test case. Questa regola non è applicabile ai end-to-end test, che invece di finire con Test o IT, finiscono con E2E.
- java:S2160 – sottoclassi che aggiungono nuovi campi a superclassi che ridefiniscono il metodo equals devono anche loro ridefinire equals. Questa regola è stata esclusa perché l'uguaglianza tra oggetti di entità nel software si basa sul UUID invece che sull'identità dell'oggetto, il quale è già stato implementato nella superclasse.

8 Hibernate

Hibernate è un ORM implementante JPA, che fornisce un modo semplice di persistere e richiamare oggetti da un database mySQL. Il primo step nell'uso di Hibernate sta nell'annotare le classi delle entità in maniera opportuna. L'annotazione @Entity serve per dire a Hibernate se una classe deve essere mappata o no a una tabella, con @Id specifichiamo poi la chiave primaria dell'entità. L'annotazione @Column serve per specificare proprietà fisiche della colonna corrispondente come se, ad esempio, può o no contenere valori nulli o il nome. Per rappresentare l'ereditarietà nei database relazionali, Hibernate mette a disposizione diverse annotazioni, questa applicazione usa @MappedSuperclass che crea una tavola separate per sottoclasse, ognuna delle quali eredita le proprietà della classe annotata. Per quanto riguarda le associazioni tra le classi delle entità, ogni istanza di GameRecord ha un riferimento ad un GameSetting. Questo risulta essere un'associazione molti a uno da parte di GameRecord. Per esprimere questa relazione ho usato l'annotazione @ManyToOne, specificando che la colonna join sarà creata nella tavola di GameRecord. JPA richiede dei costruttori vuoti nelle classi delle entità. Il ciclo di vita delle entità è gestito dai EntityManager che a loro volta possono essere container-managed (i quali si basano su dependency injection e context management) o application-managed se il ciclo di vita è completamente sotto controllo del programmatore. Il secondo è il modo in cui le classi del software gestiscono l'EntityManager. Infine, anche la configurazione del entity manager può avvenire in due modi: attraverso un file META-INF/persistence.xml o programmaticamente attraverso metodi opportuni. in questo progetto è stato usato l'approccio programmatico. L'approccio programmatico si basa sull'uso della classe PersistenceConfiguration che richiede di specificare proprietà (il driver usato, url per

raggiungere il database, ecc...) e classi gestite (classi delle entità). Una volta creato l'Entitymanager dovremo controllare se gli schemi delle tabelle sono presenti o meno e creali nel caso.

Fonti

Hibernate:

- Slides del corso di Software Architecture and Methodologies su JPA, università degli studi di Firenze 2024-2025
- Jakarta Persistence 3.2 Javadoc: https://jakarta.ee/specifications/persistence/3.2/apidocs/jakarta.persistence/module-summary.html

Log4j2:

- slf4j2: https://logging.apache.org/log4j/2.12.x/log4j-to-slf4j/
- Categorie logging Hibernate: https://docs.jboss.org/hibernate/orm/7.0/logging/logging.html
- File configurazione: https://logging.apache.org/log4j/2.x/manual/configuration.html

Commons-cli:

- Documentazione ufficiale: https://commons.apache.org/proper/commons-cli/

mysql-connector-java:

- JDBC references: https://dev.mysql.com/doc/connector-j/en/connector-j-reference.html

Assert-swing:

- Come eseguire GUI con assertj-swing test senza estendere AssertJSwingJUnitTestCase, da documentazione ufficiale: https://assertj-swing.readthedocs.io/en/latest/assertj-swing-getting-started/#write-your-first-gui-test