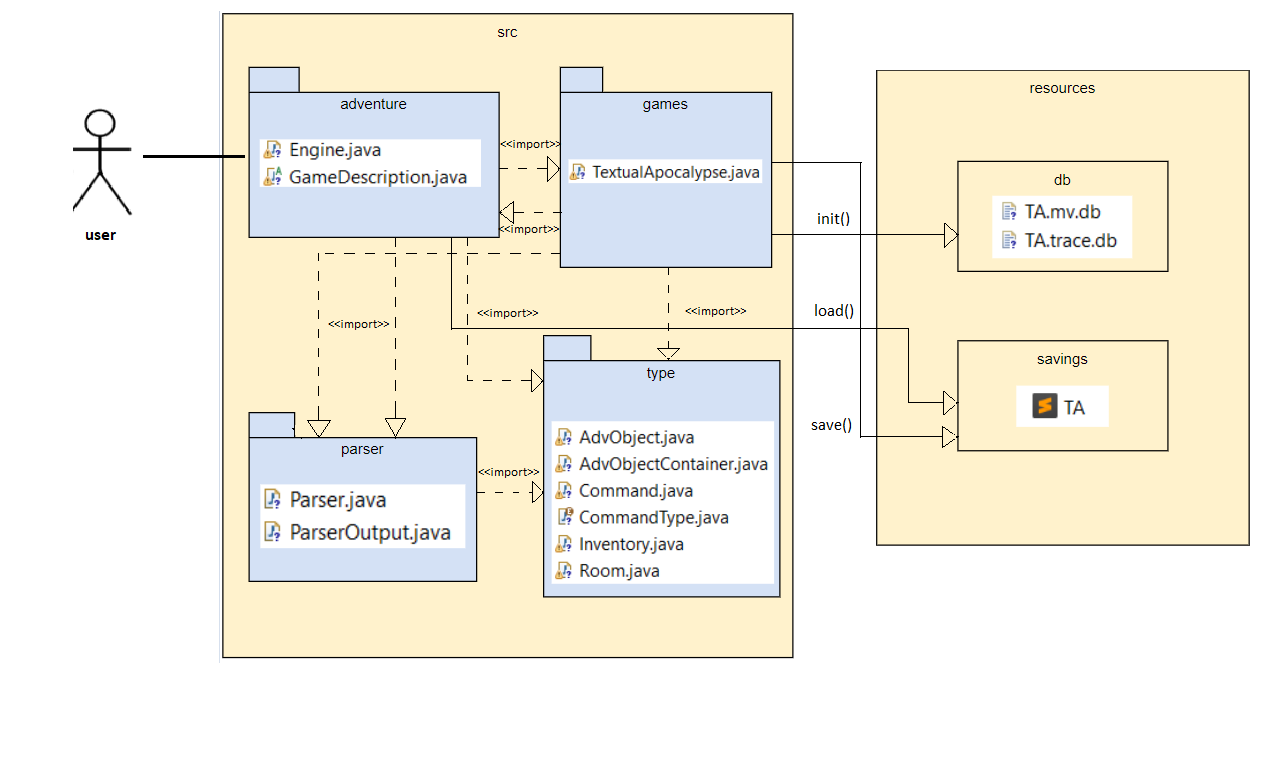
**TEXTUAL APOCALYPSE**

INTRODUZIONE

**Textual Apocalypse** è un’avventura testuale ambientata in uno scenario post apocalittico, creata da Romito Vincenzo e Ranieri Francesco nell’ambito del corso di Metodi Avanzati di Programmazione tenuto dal prof. Luigi Quaranta.

Il protagonista dell’avventura è un ex berretto verde che ora si ritrova ad affrontare un altro tipo di nemico, ben più pericoloso di un esercito: una **pandemia zombie**. Nel tentativo di aiutare un equipe di ricercatori con l’obbiettivo di creare un vaccino per questo virus, il nostro protagonista dovrà superare, tra mille pericoli ed incognite, **zombie** affamati di carne umana e numerosi **test** **automatici** per il riconoscimento di attività celebrare umana.

ARCHITETTURA DEL SISTEMA

****Il package **adventure** contiene il main dell’applicazione e consente l’IO con l’utente tramite la classe Engine.java che potrà avviare qualunque gioco presente in **games**, GameDescription.java definisce un generico gioco astratto.

Il package **games** importa il package adventure in modo da poter ridefinire tutti i giochi che potranno poi essere avviati da Engine. All’interno di questo package potranno risiedere molteplici avventure testuali.

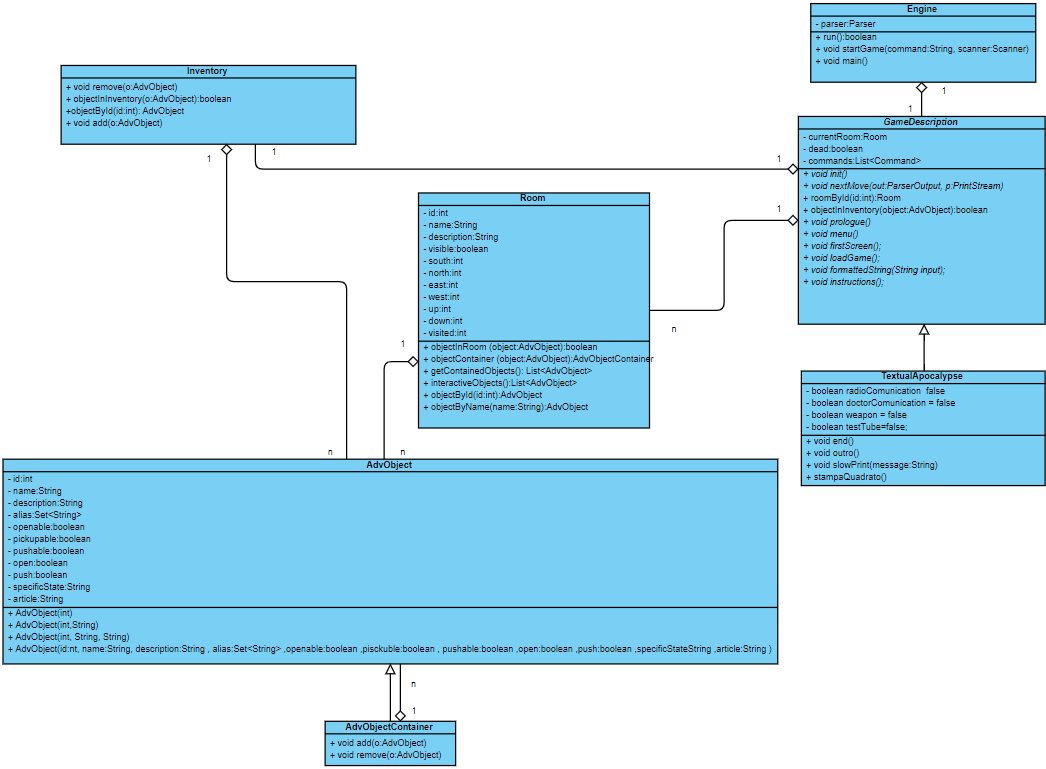
La classe Parser.java viene utilizzata dall’engine per trasformare l’input dell’utente in un comando comprensibile al gioco sotto forma di istanza di tipo ParserOutput.

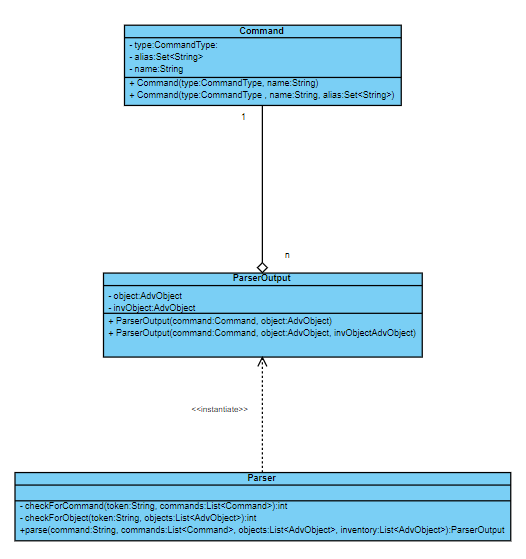
Il package **type** contiene le nuove strutture dati che verranno utilizzate all’interno del programma: si riferiscono all’inventario di gioco, ad oggetti e stanze, ma anche tipologie di comandi che devono essere riconosciuti dal Parser e gestiti nel gioco.

I dati relativi ad ogni avventura testuale sono salvati in un database embedded H2 all’interno della cartella resources/**db**, accedibile dal gioco ogni qual volta verrà avviata una nuova partita.

In qualunque momento della partita il giocatore potrà decidere di salvare e/o di caricare i propri progressi di gioco presenti in resources/**savings**.

DIAGRAMMA DELLE CLASSI





Come scelta progettuale abbiamo deciso di non includere nel diagramma UML:

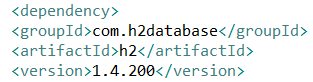
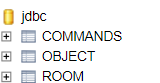
* Metodi Get() e Set()
* Costruttori, laddove fossero risultati superflui o ce ne fosse stato solo uno
* Attributi e metodi ereditati e/o ridefiniti

Tutto ciò in modo da non appesantire più del dovuto lo schema.

La classe **Engine** si occupa, tramite il metodo run(), dell’avvio dell’applicazione e quindi dell’interazione con l’utente tramite il menu di gioco. La fase di processazione della mossa viene delegata alla **GameDescription**, nello specifico, l’input dell’utente viene trasformato dal **Parser** in un input comprensibile dal gioco sotto forma di istanza di **ParserOutput**. **TextualApocalypse**, tramite la logica presente nel metodo NextMove(), e alla decisione del giocatore, provvederà a cambiare lo stato della partita: modificando la **stanza** corrente, rimuovendo o aggiungendo **oggetti** dall’ambiente circostante, aggiornando l’**inventario** e comunicando all’utente gli effetti delle sue scelte.

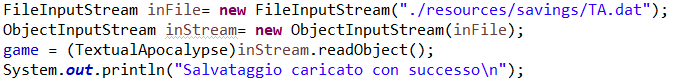
SOLUZIONI E FUNZIONALITA’ DI RILIEVO

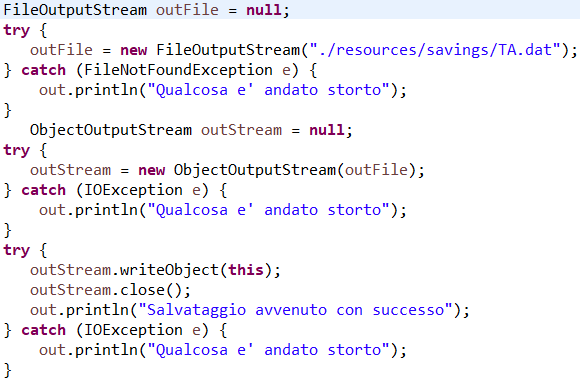
* **DATABASE EMBEDDED H2**: Tramite le dipendenze Maven è stato importato un database H2 embedded che contente la totalità delle stanze, oggetti e comandi che il gioco TextualApocalipse prevede.





* **SALVATAGGIO e CARICAMENTO PROGRESSI DA FILE:** Per una migliore esperienza di gioco è prevista la possibilità di salvare la partita in qualunque momento. Nel menu iniziale si può decidere se iniziare una nuova partita o riprendere una situazione da disco. Per fare ciò è ogni classe interessata implementerà la classe Serializable. **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

****

* **AGGIUNTA DI COMANDI:** Aggiunta la possibilità di salire di stanza tramite comando **UP**, scendere di stanza tramite comando **DOWN**, versare liquidi con comando **POUR**, sparare con un’arma tramite **SHOOT** e come già anticipato di salvare e caricare un salvataggio con **SAVE** e **LOAD**
* **ARTICOLI AD HOC:** Previsto campo articolo per ogni oggetto per fornire al gioco un italiano più fluido e piacevole
* **OGGETTI CONTENUTI:** Quando si apre un oggetto contenitore, il suo contenuto non verrà passato alla stanza ma sarà visibile al giocatore fino a quando il container rimarrà aperto
* **COMUNICAZIONE INTELLIGENTE:** La prima volta che il giocatore entrerà nella stanza gli verrà fornita una descrizione completa, le volte successive gli verrà solo ricordato in che stanza si trova e quali sono gli oggetti interagibili e quali potrebbero mutare il loro stato nel tempo
* **ENIGMI RICERCATI:** Utilizzo di caratteri UTF-8 che consentono di proporre al giocatore enigmi visivi. Sono presenti enigmi di logica, memoria e di decriptazione di messaggi.
* **MIGLIORIE NEL PARSER:** Ideato un sistema per riconoscere comandi a vuoto, su oggetti che non esistono, o su oggetti con cui non si può interagire nel modo indicato
* **STORY LINE:** Si è cercato di rendere l’esperienza quanto più immersiva possibile inserendo descrizioni complete ed esaustive per ogni stanza ed oggetto. La storia si basa su due ambienti principali: uno di introduzione che serve al giocatore per familiarizzare con comandi e con la storia, il secondo prevede un gameplay più incalzante con enigmi a frequenza e difficoltà maggiore, il tutto contornato da transizioni ben curate.