**Caso di studio per il corso di “Metodi Avanzati di Programmazione”**

**Studenti:** Pugliese Francesco, Pino Nicolò, Simone Giuseppe

# **Indice**

1. Applicazione
   1. Idea di partenza
   2. Come si gioca
   3. Gestione server
2. Aspetti implementativi
   1. Architettura del sistema
   2. Diagramma delle classi
   3. Specifica Algebrica
   4. Dettagli di realizzazione

account prova🡪 username: user, password: 1234

1. Applicazione

L’applicazione in questione è Story Maker.

* 1. Idea di partenza

Come si può dedurre dal titolo, l’applicazione è stata realizzata con lo scopo di dare la possibilità ai giocatori di essere anche, in un certo senso, creatori delle loro avventure testuali. Tutto ciò perché il giocatore, oltre a poter giocare a delle avventure testuali già presenti sul server dell’applicazione, può anche registrarsi sulla piattaforma e crearne una a suo piacimento e, successivamente, chiederne il caricamento online. In questo modo, abbiamo reso l’applicazione molto più interessante e dinamica.

* 1. Manuale d’uso

**Creazione storia**

Con lo story maker puoi creare la tua avventura testuale, che si compone di diversi livelli.

Per ogni livello si possono creare:

* un numero variabile di stanze/luoghi in cui si svolge l’avventura
* dei personaggi con cui interagire
* degli oggetti attivi
* un solo oggetto passivo.

L’oggetto attivo è un oggetto prendibile, quindi da inserire nell’inventario, potenzialmente combinabile con altri oggetti attivi.

L’oggetto passivo è un oggetto non prendibile sul quale si applicano gli oggetti attivi appartenenti al medesimo livello.

Ad esempio, in un livello ci può essere l’oggetto attivo “chiave” che va applicato sull’oggetto passivo “porta”, permettendo così di passare al livello successivo.

L’oggetto passivo può essere inteso anche come un personaggio (es. in una stanza vi è un personaggio a cui dare qualcosa).

A seconda dell’entità dell’oggetto passivo, bisogna specificare tutti gli alias del comando per applicare gli oggetti attivi a quello passivo (es. per applicare l’oggetto attivo “telefono” sull’oggetto passivo “destinatario chiamata”, i comandi che si possono utilizzare sono “chiama”, “contatta”, ecc.). Quindi bisogna inserire tutti i comandi possibile nel campo “Object passive command alias”.

Un livello si considera superato quando vengono raccolti tutti gli oggetti attivi e applicati all’oggetto passivo.

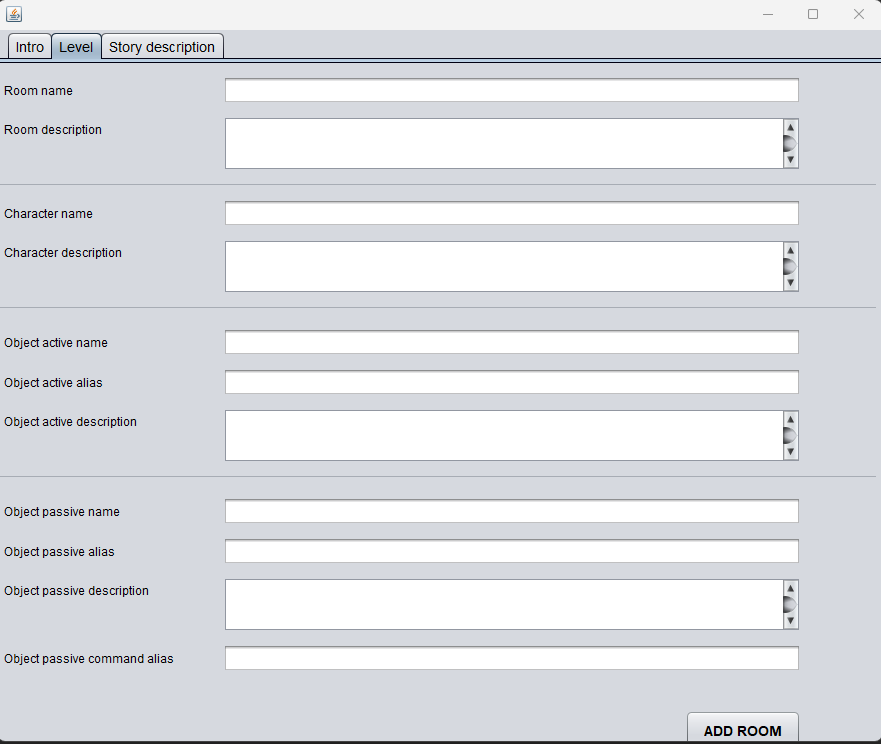
Un’avventura si considera portata a termine quando si sono superati tutti i livelli di cui essa si compone.

**Vincoli di creazione storia**

Per ogni livello è necessario creare almeno una stanza e un oggetto passivo (inserito nell’ultima stanza del livello) che rappresenta l’ostacolo da superare per passare al livello successivo o per completare l’intera avventura testuale.

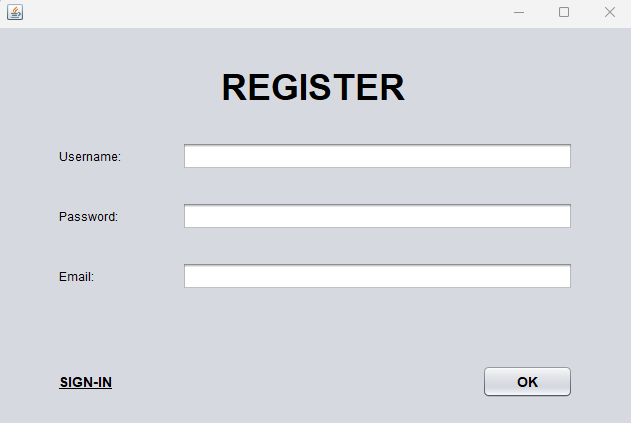
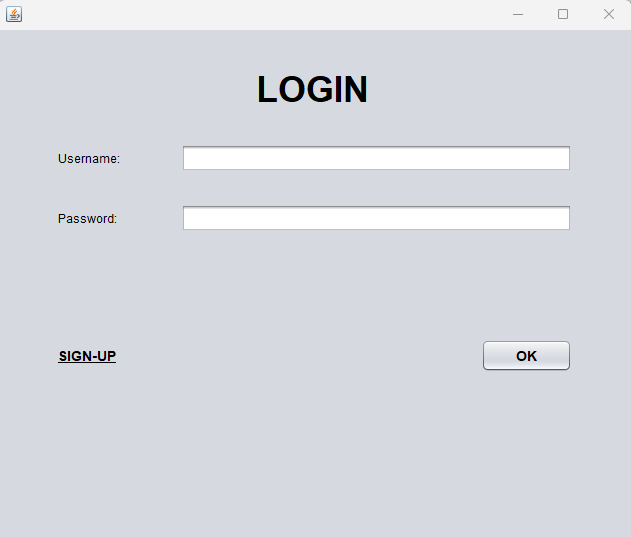
Per ogni stanza che si va a creare bisogna necessariamente compilare i campi “RoomName” e “RoomDescription”.

Per ogni stanza si possono creare esclusivamente un oggetto attivo, un personaggio e un oggetto passivo.



**Gestione account**

È possibile creare il proprio account inserendo username, e-mail e password, oppure effettuare l’accesso ad un account già esistente tramite username e password. Una volta effettuato l’accesso, viene visualizzato in alto a destra lo username dell’utente, cliccando il quale è possibile effettuare il logout. Senza effettuare il login, è possibile solamente creare una storia e giocarla in locale, senza poter chiedere di caricarla online.



**Caricamento di una storia dal server**

Una volta effettuato l’accesso, l’utente può giocare ad una storia presente online. Alla fine della sessione di gioco, l’utente può dare una valutazione che andrà a influire sullo score della storia appena giocata.

Immagine che contiene testo

Descrizione generata automaticamente

**Altro**

All’avvio dell’applicazione si avvia anche la riproduzione musicale. Quest’ultima può essere abilitata o disabilitata dal menù impostazioni.

Ogni qualvolta che il giocatore supera un livello o termina la storia viene mostrata una gif.

**Lista dei comandi**

Di seguito la lista dei comandi a disposizione del giocatore durante l’avventura testuale:

* Parla 🡪 comando per parlare con un personaggio (viene stampata la descrizione del personaggio).
* Menu 🡪 comando per stampare a video l’elenco dei comandi ammissibili.
* Next 🡪 comando per muoversi verso la stanza successiva
* Previous 🡪 comando per muoversi verso la stanza precedente
* End 🡪 comando per uscire dal gioco
* Inventario 🡪 comando per visualizzare gli oggetti presenti nell’inventario
* Osserva 🡪 comando per osservare l’ambiente circostante (verranno stampate le descrizioni degli oggetti attivi e passivi presenti nella stanza corrente).
* Raccogli 🡪 comando per raccogliere un oggetto
* Usa 🡪 comando per usare/applicare un oggetto attivo dall’inventario su un oggetto passivo

Per ogni comando è stata stilata una lista dei sinonimi da poter utilizzare in alternativa a quelli predefiniti.

Parla 🡪 ascolta, discuti, dialoga

Menu 🡪 menù, comandi, mosse

Next 🡪 avanti, su, prossima

Previous 🡪 indietro, giù, precedente

End 🡪 fine, esci, muori, ucciditi, suicidati, exit

Inventario 🡪 inv

Osserva 🡪 guarda, vedi, trova, cerca, descrivi

Raccogli 🡪 prendi

Premi 🡪 spingi, usa, attiva, dai

* 1. Gestione server

Il server si occupa della gestione delle avventure testuali mandate dagli utenti, in particolare ogni avventura testuale inviata al server viene inserita in una directory StoryReceived. Il server a questo punto manda una mail di conferma della ricezione della storia. Ogni storia, prima di poter essere giocata online lato client, deve essere aggiunta nella cartella StoryConfermed. Quindi, lato server, è possibile visualizzare il contenuto di ogni storia ricevuta e decidere se è opportuno caricarla online.

**Comandi server**

* **story 🡪** permette di visualizzare i titoli delle storie e le dimensioni in byte, situate nella cartella StoryReceived.
* **story “*nomestoria”*🡪**permette di visualizzare tutte le informazioni della storia, situata nella cartella StoryReceived.
* **add *“nomestoria”* 🡪** permette di aggiungere la storiaalla lista delle storie caricabili online, quindi nella cartella StoryConfermed.



* **delete *“nomestoria”* 🡪** permette di rimuovere la storia dal server.

1. Aspetti implementativi
   1. Architettura del sistema

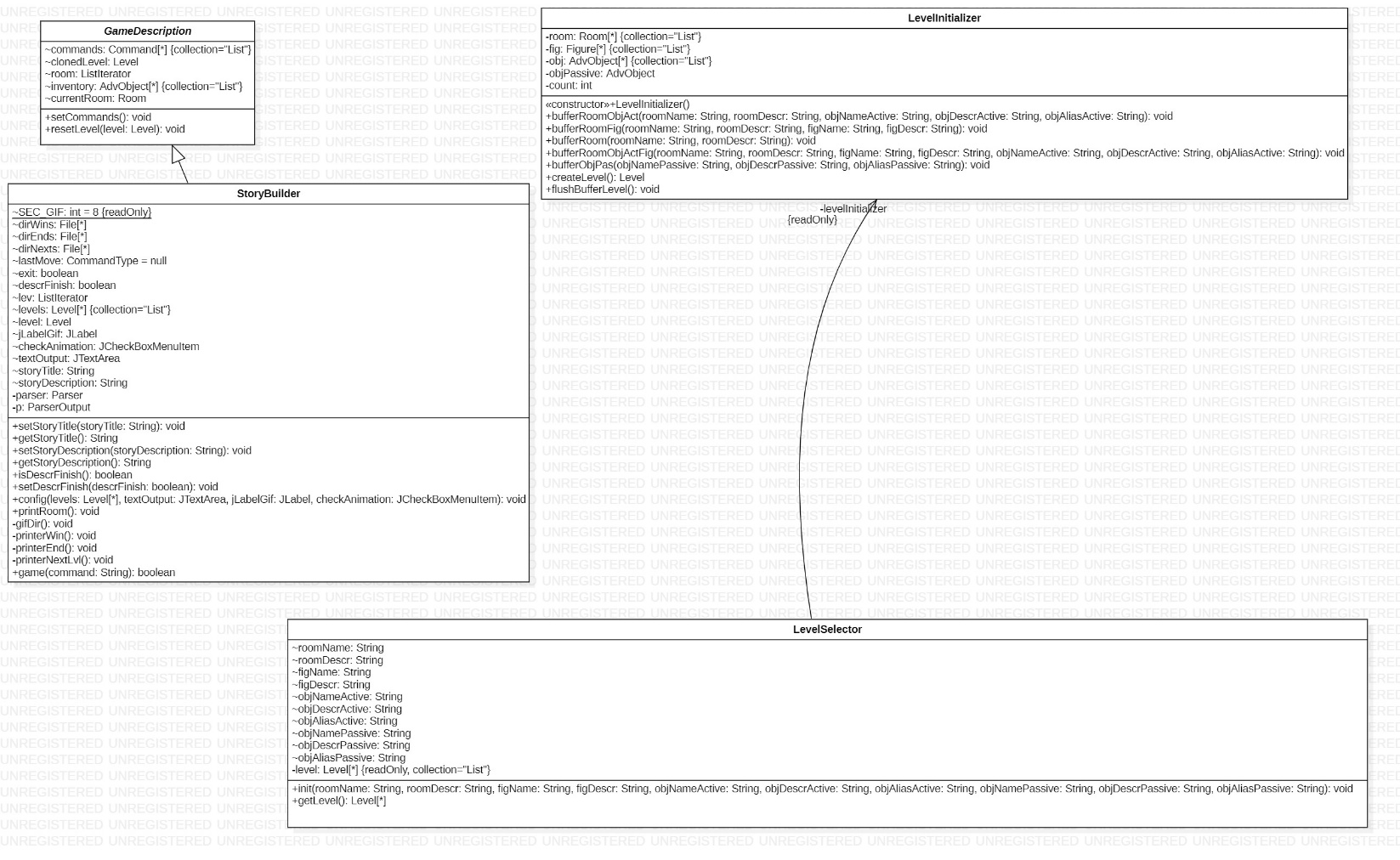
Sia il software lato server sia il software lato client possono essere interpretati come MVC (Model – View – Controller). La View visualizza i dati contenuti nel Model. Il Model fornisce i metodi per accedere ai dati utili per l’applicazione. Il Controller riceve i comandi dall’utente, in genere attraverso la View, e li attua modificando lo stato degli altri due componenti.

Questo schema implica anche la separazione tra la logica applicativa, a carico del Controller e del Model, e l’interfaccia utente a carico della View.

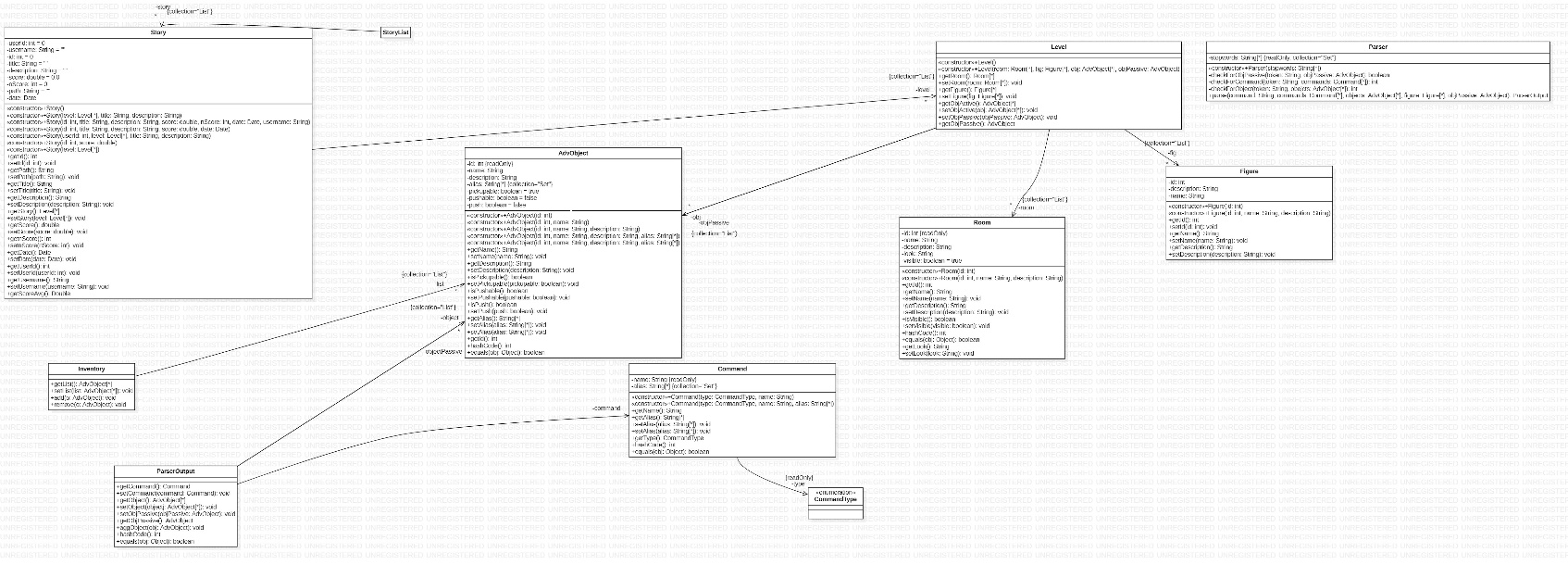
**Package**

* **Controller** si occupa della creazione di una storia, del controllo dell’input dall’utente ed è strettamente connessa con il package Model in quantoistanzia gli elementi della storia come il Level o la Room.
* **View (lato client)** è responsabile degli input inseriti dall’utente e interagisce con il Controller ogni qualvolta avviene la creazione di una storia o si sta giocando.
* **Model** è responsabile di istanziare gli elementi della storia.
  1. Diagramma delle classi

**Package control StoryMakerClient**

****

**Package model StoryMakerClient**

****

**Package util StoryMakerClient**

**Immagine che contiene tavolo

Descrizione generata automaticamente**

**Package view StoryMakerClient**

**Immagine che contiene tavolo

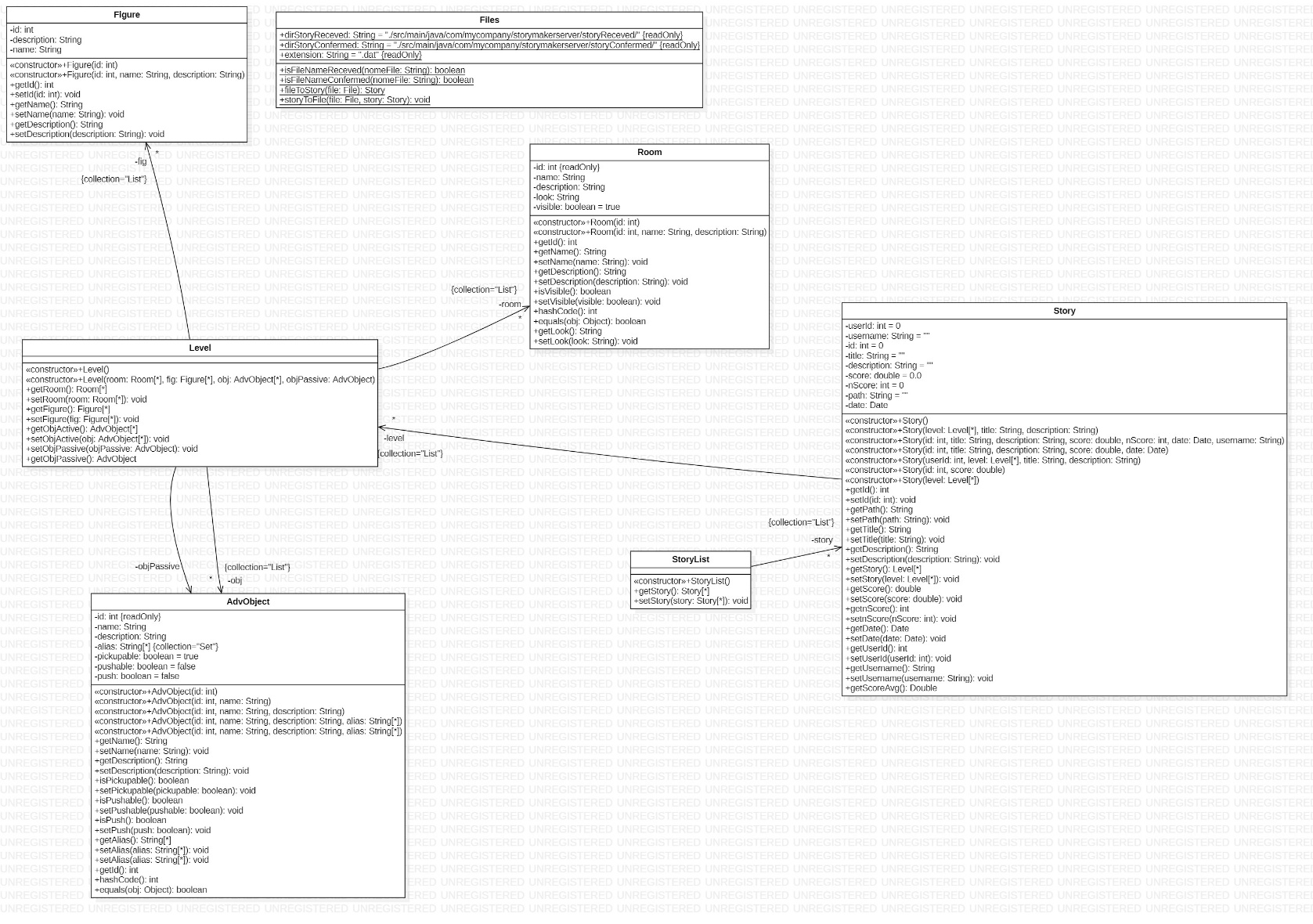
Descrizione generata automaticamente**

**Package control StoryMakerServer**

**Immagine che contiene tavolo

Descrizione generata automaticamente**

**Package model StoryMakerServer**

****

1. Specifica algebrica

Abbiamo optato per la specifica algebrica della struttura dati lista. Definiamo innanzitutto la specifica sintattica di quelli che sono gli operatori da applicare alla lista.

newList() 🡪 lista

insLista(tipoelem, posizione, lista) 🡪 lista

listaVuota(lista) 🡪 boolean

primoLista(lista) 🡪 posizione

ultimoLista(lista) 🡪 posizione

succLista(posizione, lista) 🡪 posizione

predLista(posizione, lista) 🡪 posizione

fineLista(posizione, lista) 🡪 boolean

leggiLista(posizione, lista) 🡪 tipoelem

cancLista(posizione, lista) 🡪 lista

A questo punto, dividiamo gli operatori in costruttori e osservatori.

I costruttori sono:

newList() 🡪 lista

insLista(tipoelem, posizione, lista) 🡪 lista

in quanto istanziano e/o popolano la struttura dati lista.

Gli osservatori sono invece:

listaVuota(lista) 🡪 boolean

primoLista(lista) 🡪 posizione

ultimoLista(lista) 🡪 posizione

succLista(posizione, lista) 🡪 posizione

predLista(posizione, lista) 🡪 posizione

fineLista(posizione, lista) 🡪 boolean

leggiLista(posizione, lista) 🡪 tipoelem

cancLista(posizione, lista) 🡪 lista

in quanto estraggono informazioni circa la struttura dati.

Costruiamo la tabella Osservatori/Costruttori e applichiamo gli osservatori ai costruttori.

|  |  |  |
| --- | --- | --- |
| Osservatori | Costruttori di lista’ |  |
|  | newList() | insList(tipoelem, pos, lista) |
| listaVuota(lista’) | true | false |
| primoLista(lista’) | error | **if** pos!= primoLista(lista) **then** primoLista(lista)  **else** pos |
| ultimoLista(lista’) | error | **if** pos!= ultimoLista(lista)  **then** ultimoLista(lista)  **else** pos |
| succLista(pos’, lista’) | error | **If** pos != pos’  **then** succLista(pos’, lista)  **Else** null |
| predLista(pos’, lista’) | error | **If** pos != pos’  **Then** predLista(pos’, lista)  **Else** null |
| fineLista(pos’, lista’) | error | **If** listaVuota(lista) AND pos ==pos’  **Then** true  **Else** **if** pos == ultimoLista(lista’) AND pos==pos’  **Then** true  **Else** false |
| leggiLista(pos’, lista’) | error | **If** pos!=pos’  **Then** leggiLista(pos’, lista)  **Else** tipoelem |
| cancLista(pos’, lista’) | error | **If** listaVuota(lista) **AND** pos==pos’  **Then** listaVuota  **Else** **if** pos==pos’  **Then** lista  **Else** cancLista(pos’, lista) |

Di conseguenza, la specifica algebrica di questa struttura dati si può così definire:

* listaVuota(newList()) 🡪 true
* listaVuota(insList(tipoelem, pos, lista)) 🡪 false
* primoLista(insList(tipoelem, pos, lista)) 🡪 **if** pos!=primoLista(lista)

**then** primoLista(lista)

**else** pos

* ultimoLista(insList(tipoelem, pos, lista)) 🡪 **if** pos!=ultimoLista(lista)

**then** ultimoLista(lista)

**else** pos

* fineLista(pos’, insList(tipoelem, pos, lista))🡪 **If** listaVuota(lista) AND pos==pos’

**then** true

**Else**

**if** pos==ultimoLista(lista’) AND pos==pos’

**Then** true **Else** false

* leggiLista(pos’, insLista(tipoelem, pos, lista)🡪 **If** pos!=pos’

**Then** leggiLista(pos’, lista)

**Else** tipoelem

* cancLista(pos’, insList(tipoelem, pos, lista))🡪 if listaVuota(lista) AND pos==pos’

then listavuota

else if pos==pos’

then lista

else cancLista(pos’, lista)

1. Dettagli implementativi

**Collection**

Abbiamo utilizzato la Collection List per implementare la lista di stanze che compongono un livello, la lista di oggetti che compongono un livello, la lista di figure che compongono un livello, la lista di livelli che compongono una storia.

Abbiamo utilizzato la Collection Set per rappresentare l’insieme delle stopwords.

**Interfacce**

Abbiamo implementato l’interfaccia Serializable con la classe Figure, AdvObject, Level, Room, Story.

**Classi astratte**

Abbiamo utilizzato la classe astratta GameDescription che viene estesa dalla classe StoryBuilder.

**Eccezioni**

Abbiamo creato dei meccanismi per la gestione di Eccezioni di tipo IOExecption, SQLException e ClientException.

**Database**

È stato creato un database per gestire tutti gli utenti che si sono registrati e tutte le storie da loro create.

Attraverso delle query SQL si sono create le tabelle user e story. La tabella user si compone dei seguenti campi:

* + - Id che contraddistingue univocamente l’utente registrato, ovvero la chiave primaria della tabella.
    - Username
    - Password
    - Email

La tabella story ha come attributi:

* + - Id che contraddistingue univocamente la storia caricata
    - Title che rappresenta il titolo della storia
    - Description che rappresenta la descrizione della storia
    - Score che rappresenta la somma delle valutazioni ottenute della storia
    - nScore che rappresenta il numero di valutazioni attribuite alla storia
    - date che rappresenta la data di creazione della storia
    - path che rappresenta il percorso del file contenente la storia
    - idAuthor che rappresenta l’id dell’utente che ha creato la storia, nonché chiave esterna.

Uno strumento importante che abbiamo usato è H2, che ci ha permesso di implementare il protocollo JDBC. Infatti, abbiamo inserito la seguente dipendenza nel pom.xml:

Immagine che contiene testo

Descrizione generata automaticamente

Successivamente, abbiamo aperto la connessione tramite la creazione di un oggetto di tipo Connection, che fornisce tutti i metodi per preparare le query SQL. Per ottenere una connessione è necessario caricare il driver che implementa le API JDBC e ciò si ottiene semplicemente chiamando il metodo getConnection della classe DriverManager passando la stringa di connessione.

Inoltre, sono state previste delle SQLException che si sollevano quando si verifica un’anomalia in sede di interrogazioni sul Database.

**File**

Abbiamo utilizzato i file per salvare le storie in memoria. Tutto questo grazie ai metodi della classe Files storyToFile che permette di convertire un oggetto di classe Story in un File con estensione “.dat”. Abbiamo pensato a due directory di destinazione come StoryReceived e StoryConfirmed per tenere traccia delle storie.