Cartella sanitaria

Progetto universitario d’informatica – linguaggio Java

Specifiche Architetturali tecniche

# Definizione architetturale

## Gestione input/output

Per il progetto è richiesto una semplice interazione tramite console di comando, per questo motivo il software sarà un unico eseguibile in linguaggio Java.

L’eseguibile verrà incapsulato in un executable jar standard eseguibile da una JRE installata sul sistema operativo, questo consentirà di avere un eseguibile cross-platform compatibile con diversi SO.

### Referenze

Definizione e spiegazione dei JAR files.

<https://docs.oracle.com/en/java/javase/21/docs/specs/jar/jar.html>

## Gestione dei dati

Il progetto richiede un salvataggio e un caricamento di dati da un sistema di persistenza e specifica che deve essere un file su file system. Per questo motivo non verranno usati DB, ma semplici scritture e letture su unico file. Il sistema leggerà i dati all’avvio e li manterrà in sessione, finita una transazione funzionale il file verrà aggiornato rimpiazzando il precedente.

I dati verranno scritti serializzandoli in un formato JSON standard.

I dati non saranno criptati in quanto il progetto non richiede questo tipo di sicurezza.

### Referenze

Definizione e spiegazione dei JSON

<https://datatracker.ietf.org/doc/html/rfc8259>

## Date

Le date usate nel software avranno il seguente formato: *yyyy-MM-dd'T'HH:mm:ss.SSSZ*

Esempio: *2001-07-04T12:08:56.235-0700*

Si utilizzerà questo per rispettare lo standard ISO 8601 con adattamento Java secondo RFC 822.

### Referenze

<https://en.wikipedia.org/wiki/ISO_8601>

<https://www.ietf.org/rfc/rfc0822.txt>

## Librerie

Come linguaggio principale verrà usato Java versione 17.

Verranno utilizzate librerie open source o con licenze free.

Come spiegato in questo articolo, l’utilizzo delle librerie non è solo fortemente consigliato, ma anche una buona regola di programmazione al fine di ottimizzare, migliorare, rendere più sicuro e diminuire i costi del software sviluppato.

<https://www.cleveroad.com/blog/check-out-the-benefits-of-open-source-library-and-open-source-software/>

### Gestore librerie

Come gestione di librerie verrà usato lo strumento Maven. Questo, oltre a offrire una gestione centralizzata e automatizzata delle librerie, offre molte funzionalità come compilare e generare l’eseguibile ed eseguire gli unit tests.

<https://maven.apache.org/>

## Repository del codice

Il codice è depositato come open source su GitHub®.

<https://github.com/Cartella-Sanitaria-Project>

## Licenza

Il progetto viene esposto con licenza GNU GPL 3.0

<https://www.gnu.org/licenses/gpl-3.0.html>

## Logging e Metriche

Non verranno inseriti metriche di utilizzo, logging e altri software di troubleshooting in quanto non richiesto dal progetto e dato che il software è indicato all’utilizzo di un singolo utente tramite console di comando. Eventuali errori o segnalazioni saranno riportate direttamente in console.

# Impostazione preliminare del progetto

## Creazione classe di main

La classe di avvio si chiamerà Main, avrà un metodo main statico e sarà l’avvio dell’applicazione.

Questa classe avrà il compito di inizializzare la sessione, caricare dati da file locale e avviare la prima richiesta all’utente, ovvero un menù contestuale per richiedere la funzionalità da eseguire.

## Creazione della classe di sessione

La classe di sessione si chiamerà ApplicationSession, sarà una classe con costruttore privato e quindi accessibile alla classe stessa.

La classe conterrà un'unica istanza statica di sé stessa creata una sola volta e ogni interazione dalle classi esterne avverrà caricando la classe tramite metodo statico esposto.

Questo tipo di soluzione è definita Singleton:

<https://www.oracle.com/technical-resources/articles/java/singleton.html>

La sessione conterrà le informazioni dell’utente che serviranno anche per aggiornare il file.

## Creazione della classe di menù

La classe di menù si chiamerà Menu, sarà una classe con un metodo pubblico non statico senza argomenti e void. Questo stamperà in console le varie funzioni disponibili con un numero associato e richiederà in input all’utente il numero della funzione.

La richiesta di input deve essere contenuta in un ciclo while che finirà quando l’utente inserisce un numero corretto della lista.

Una volta ricevuto il numero, la classe di menù chiamerà una classe esterna in base alla funzionalità scelta.

In più nella lista deve essere aggiunto un campo “0 – Termina applicazione” per consentire di terminare il software. La chiusura dell’applicazione avverrà tramite semplice “System.exit(0)”.

# Requisito A - Memorizzazione e gestione delle informazioni personali dell’utente

## Dati di persistenza

Tabella 1:

|  |  |  |
| --- | --- | --- |
| Nome campo | Tipologia campo | Descrizione |
| nome | Stringa | Nome del paziente |
| cognome | Stringa | Cognome del paziente |
| indirizzo | Stringa | Indirizzo del paziente |
| email | Stringa | Email del paziente |
| telefoni | Array di Stringhe | Uno o più numeri di telefono |
| dataNascita | Stringa | Data in formato descritto nel punto 1.3 |
| luogoNascita | Stringa | Luogo di nascita del paziente |
| genere | Stringa | Maschio o Femmina |
| codiceFiscale | Stringa | Composta: 6 caratteri [A-Z], 2 cifre [0-9], 1 carattere [A-Z], 2 cifre [0-9], 1 carattere [A-Z], 3 cifre [0-9], 1 carattere [A-Z] |
| codiceSanitario | Stringa | Codice univoco, dettagli al punto 3.3 |
| gruppoSanguigno | Object | Ref. GruppoSanguigno |

Tabella 2:

|  |  |  |
| --- | --- | --- |
| Nome campo | Tipologia campo | Descrizione |
| fattoreRh | Stringa | Positivo o Negativo |
| gruppo | Stringa | 0, A, B, AB |

## Classi di persistenza

È necessario creare le seguenti classi di persistenza per gestire i dati, le classi implementeranno l’interfaccia Serializable per essere serializzate come JSON.

La classe Paziente deve essere aggiunta nella classe di sessione come variabile privata non statica e singola (non array), l’accesso a questa verrà fatto prendendo l’istanza della sessione e accedendo alla variabile tramite metodo di getter, l’aggiornamento avverrà nello stesso modo ma usando invece il setter.

### Classe del paziente

La classe si chiamerà Paziente e conterrà le informazioni della tabella 1, le variabili saranno private ed esposte tramite getter e setter.

### Classe del gruppo sanguigno

La classe si chiamerà GruppoSanguigno e conterrà le informazioni della tabella 1, le variabili saranno private ed esposte tramite getter e setter.

## Generazione codice sanitario

La creazione avverrà tramite UUID senza dash (-), questo genererà un codice univoco e casuale.

Questo compito sarà affidato ad una classe di utils che userà il metodo *UUID.randomUUID()* e farà il replace dei trattini tramite *.replace("-", "")*.

<https://it.wikipedia.org/wiki/Universally_unique_identifier>

## Aggiornamento della classe di Menù

La classe Menu deve essere aggiornata aggiungendo nel messaggio della lista delle funzionalità la funzione di aggiornamento dei dati.

Inoltre, bisogna implementare la chiamata ad una nuova classe di aggiornamento dei dati in caso l’utente scelga quella voce del menù.

## Classe di aggiornamento dati

La classe si chiamerà AggiornaDati, esporrà un metodo principale che stamperà la lista dei dati con a fianco il relativo numero e chiederà all’utente di inserire il numero del campo da aggiornare, in più nella lista deve essere aggiunto un campo “0 – Uscita, torna al menù principale” per consentire di tornare al menù principale.

### Aggiornamento del dato

Il metodo principale, una volta capito il campo da aggiornare, chiamerà uno dei vari e specifici metodi della stessa classe per l’aggiornamento del campo, questo chiederà un input all’utente e controllerà la bontà del dato in base al campo scelto. In caso il valore inserito non sia corretto chiederà di nuovo l’input all’utente.

Prima della richiesta del campo deve essere stampato anche un messaggio che informa dicendo: “Per tornare al menù dei campi inserisci: 0-exit” e in caso l’input sia quello il software deve tornare al menù di aggiornamento dati.

AGGIORNARE IL FILE DI PERSISTENZA

# Requisito B - Definizione e memorizzazione dei possibili esami inerenti allo stato di salute

## Dati di persistenza

Tabella 1:

|  |  |  |
| --- | --- | --- |
| Nome campo | Tipologia campo | Descrizione |
| idEsame | Stringa | Generato tramite System.currentTimeMillis al momento della creazione dell’esame |
| nome | Stringa | Nome dell’esame |
| tipologia | Stringa | Periodico o Diagnostico |
| preparazione | Stringa | Lunga descrizione della preparazione all’esame medico |
| valoreNormalitaMin | Numero | Descrizione dei valori di normalità minimo atteso |
| valoreNormalitaMax | Numero | Descrizione del valore di normalità massimo atteso |
| sogliaErroreInserimento | Numero | Numero di soglia da controllare dettagli al punto 4.2 |
| trattamentiPostEsame | Stringa | Descrizione dei trattamenti da effettuare post esame |

## Gestione delle soglie e valori di normalità

Queste informazioni sono richieste solo in caso di esame di tipo Periodico.

La soglia è un valore numerico che verrà applicata come controllo al momento dell’inserimento dell’esito. Il campo sarà facoltativo e, nel caso non specificato o con input 0, assumerà il valore del campo valoreNormalitaMax x5 (in valore assoluto).

Se il valore inserito come esito sarà superiore del valoreNormalitaMax + soglia oppure minore del valoreNormalitaMin - soglia, allora il sistema visualizzerà un errore e chiederà conferma all’operatore se il valore inserito è corretto tramite un (Y/N).

## Classe di persistenza

Verrà creata una classe con nome Esame che conterrà le informazioni della tabella 1.

La classe di sessione deve essere aggiornata in modo da avere una variabile privata non statica che conterrà una mappa di Id/Esami, questa sarà accessibile tramite metodi di getter e setter.

La mappa sarà una semplice HashMap con chiave l’id dell’esame e come valore l’esame stesso, questo al fine di velocizzare l’accesso al dato con una lettura diretta tramite indice.

### Referenze

<https://www.baeldung.com/java-arraylist-vs-linkedlist-vs-hashmap>

## Aggiunta di un esame

Dovrà essere creata una nuova classe con nome “AggiungiEsame”, la classe avrà un metodo principale di avvio senza argomenti pubblico e non statico, il ritorno di questo sarà void.

Il metodo stamperà il messaggio informativo iniziale “Inserimento di un nuovo esame medico, inserisci le informazioni oppure digita in qualsiasi momento 0-exit per tornare al menù principale”.

Dopo di che, verranno richiesti i vari dati in successione e controllando l’input, se il dato è corretto si procede con il successivo, se non è corretto si notifica un errore e si chiede il re-inserimento oppure se è “0-exit” si termina l’inserimento senza salvare l’esame.

## Salvataggio della sessione

Quando l’ultimo dato viene inserito, l’oggetto creato deve essere memorizzato nella lista degli esami.

AGGIORNARE IL FILE DI PERSISTENZA

# Requisito C - Memorizzazione e gestione dello stato di salute dell’utente

## Dati di persistenza

Tabella 1

|  |  |  |
| --- | --- | --- |
| Nome campo | Tipologia campo | Descrizione |
| idEsame | Stringa | FK like, per associare il dato all’esame inserito |
| idPrestazione | Stringa | ID univoco per la prestazione. Generato tramite System.currentTimeMillis al momento della creazione dell’esame |
| dataEsame | Stringa | Data in formato descritto nel punto 1.3 |
| luogo | Stringa | Luogo dell’esame |
| malattia | Stringa | Malattia per cui è stato richiesto l’esame medico |
| esito | Stringa | Descrizione o valore dell’esito dell’esame |

## Classe di persistenza

Verrà creata una classe di persistenza con nome PrestazioneEsame, conterrà i dati della tabella 1 e esporrà le variabile con getter e setter.

## Classe di ricerca

Per poter creare una prestazione è necessario conoscere il tipo di esame. È necessario creare una classe di ricerca con nome CercaEsame. La classe avrà 2 metodi uno di ricerca tipologia e uno di ricerca prestazione. Entrambi i metodi saranno pubblici e non statici.

Ricerca tipologia ritornerà una stringa che indicherà l’ID univoco dell’esame.

Ricerca prestazione ritornerà una stringa che indicherà l’ID univoco della prestazione.

### Ricerca della tipologia

Verrà richiesto all’utente il nome dell’esame e verrà ricercato quel nome ciclando gli esami inseriti in sessione, quelli che contengono completamento o in parte il nome inserito saranno messi in una lista.

Alla fine, verrà stampato nome e tipologia esame per ogni oggetto in lista con a fianco un numero per indicare il selettore e verrà richiesto all’utente di selezionare il numero dell’esame.

In aggiunta deve presentarsi anche un’opzione [0] torna al menù principale. In questo caso invece che tornare l’ID dell’esame verrà ritornato null e il chiamante deve capire che la ricerca è stata interrotta e tornare al menù iniziale.

### Ricerca della prestazione

La ricerca della prestazione inizia chiamando la ricerca della tipologia esame, una volta ricevuto l’ID dell’esame si stampano tutte le prestazioni con lo stesso ID mostrando data, ora e luogo.

A fianco di queste si metterà il selettore con la solita opzione di ritorno al menù.

## Inserimento prestazione

Dovrà essere creata una nuova classe con nome InserimentoPrestazione con un metodo principale senza argomenti, con ritorno void, pubblico e non statico.

Aggiornare la classe di menù in modo che ci sia la funzione di inserimento della prestazione medica.

La nuova classe chiederà in successione all’utente i vari dati della prestazione dell’esame.

### Esito

Il campo esito è opzionale, quando viene chiesto all’utente si deve specificare che si può lasciare vuoto.

In caso sia popolato, bisogna effettuare dei controlli nel caso l’esame selezionato sia Periodico: ovvero il campo deve essere un numero con un valore compreso tra le soglie indicate. I dettagli nel punto 4.2

### Malattia

Il campo malattia è opzionale, quando viene chiesto all’utente si deve specificare che si può lasciare vuoto.

### Data e ora

Per creare la data e l’ora della prestazione si chiederà