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>

### Gestione della data

Per gestire le date in modo pulito e stabile si userà le classi di Java “Date”, “DateFormat” e “Calendar”.

La data inserita dall’utente sarà prima parsata tramite DateFormat in Date e poi trasformata in Calendar in modo da poter essere poi gestita e editata. Una volta finita la modifica verrà trasformata in String tramite il DateFormat.

## 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.

## Classe di inizializzazione dati

La classe si chiamerà InizializzaDati, esporrà un metodo principale richiamabile solo una volta per l’inserimento dei dati del paziente in sequenza.

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 | Long | 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 dove chiede se continuare e iniziare l’inserimento oppure di annullare la funzione tramite 0.

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.

## 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 | Long | FK like, per associare il dato all’esame inserito |
| idPrestazione | Long | 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 |
| idMalattia | Long | FK like, id della malattia per cui è stato richiesto l’esame medico |
| esito | Stringa | Descrizione o valore dell’esito dell’esame |

Tabella 2

|  |  |  |
| --- | --- | --- |
| Nome campo | Tipologia campo | Descrizione |
| idMalattia | Stringa | ID univoco per la malattia  Generato tramite System.currentTimeMillis al momento della creazione dell’esame |
| nome | Stringa | Nome della malattia |
| dataInizio | Stringa | Data in formato descritto nel punto 1.3 |
| dataFine | Stringa | Data in formato descritto nel punto 1.3 |
| sintomi | Stringa | Descrizione dei sintomi della malattia |
| diagnosi | Stringa | Descrizione della diagnosi medica |
| elencoEsami | Lista di Stringhe | Lista di esami prescritti per questa malattia. Sarà gestito tramite una semplice lista di stringhe inserite dall’utente |
| terapia | Stringa | Descrizione della terapia medica prescritta |

## Classe di persistenza

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

Verrà creata una classe di persistenza con nome Malattia, conterrà i dati della tabella 2 ed esporrà le variabili tramite i 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 3 metodi uno di ricerca tipologia, uno di ricerca prestazione e uno di ricerca malattia. I metodi saranno pubblici e non statici.

Ricerca tipologia ritornerà un numero che indicherà l’ID univoco dell’esame.

Ricerca prestazione ritornerà un numero che indicherà l’ID univoco della prestazione.

Ricerca malattia ritornerà un numero che indicherà l’ID univoco della malattia.

### 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.

### Ricerca della malattia

Verrà richiesto all’utente il nome della malattia e verrà ricercato quel nome ciclando le malattie precedentemente inserite in sessione, quelle che contengono completamento o in parte il nome inserito saranno messi in una lista.

Alla fine, verrà stampato nome e data di inizio della malattia per ogni oggetto in lista con a fianco un numero per indicare il selettore e verrà richiesto all’utente di selezionare il numero della malattia.

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

## Inserimento prestazione medica

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.

L’inserimento della malattia è gestito con una ricerca delle malattie precedentemente inserite.

### Data e ora

Per creare la data e l’ora della prestazione si chiederà in modo separato la data1 e l’ora specificando il formato di inserimento. Si dovrà effettuare controlli sul formato “yyyy/MM/dd” e “hh:mm”.

## Inserimento della malattia

Verrà creata una nuova classe con nome InserimentoMalattia, con un metodo principale void, pubblico, non statico e senza argomenti di input.

Aggiornare la classe di menù inserendo una nuova voce di funzionalità che indica l’inserimento della malattia inserita e abbinare il numero corrispondente alla chiamata della nuova classe di inserimento.

La nuova classe dovrà chiedere in successione i dati della malattia all’utente.

### Id Malattia

L’ID sarà generato tramite semplice System.currentTimeMillis() in modo da avere un ID incrementale univoco.

### Date

Per la data di inizio e di fine verrà richiesto solo il giorno in formato yyyy/MM/dd e settata l’ora delle 12:00:00.

La data di fine è opzionale, si chiederà quindi di inserire o la data oppure il numero 0 o stringa vuota per indicare di saltare l’inserimento di questa.

### Elenco esami

In un loop, si chiederà l’inserimento testuale del nome degli esami associati alla malattia indicando che con il numero 0 è possibile terminare l’inserimento e continuare al prossimo campo.

## Modifica prestazione medica

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

Aggiornare la classe di menù inserendo una nuova voce di funzionalità che indica l’aggiornamento della prestazione medica inserita e abbinare il numero corrispondente alla chiamata della nuova classe di modifica.

La nuova classe dovrà per prima cosa fare una ricerca della prestazione medica tramite chiamata alla classe CercaEsame e al metodo di ricerca prestazione medica.

Selezionata la prestazione si dovrà mostrare i vari dati della prestazione selezionata e, tramite un menù numerico, dare la possibilità di selezionare un campo e modificarlo.

### Modifica della data

La data dovrà essere visualizzata per intero, ma nel menù di modifica dovrà essere divisa la possibilità di modifica o del giorno, mese e anno e la modifica dell’ora e dei minuti. In questo modo l’utente sarà più agevole nell’inserimento dei dati.

### Modifica della malattia

La modifica della malattia deve essere fatta con un’iterazione dove prima si chiede se si vuole annullare la modifica con 0, eliminare la malattia attualmente assegnata con 1 oppure con un altro carattere si fa partire la ricerca della malattia e si sovrascrive quella precedentemente assegnata.

## Modifica della malattia

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

Aggiornare la classe di menù inserendo una nuova voce di funzionalità che indica l’aggiornamento della malattia inserita e abbinare il numero corrispondente alla chiamata della nuova classe di modifica.

La nuova classe dovrà per prima cosa fare una ricerca della malattia tramite chiamata alla classe CercaEsame e al metodo di ricerca malattia.

Una volta selezionata la malattia, si dovrà stampare la lista dei campi assegnati con a fianco il numero di selettore e chiedere di selezionare uno dei campi corrispondenti per iniziare la modifica, inoltre dovrà essere presente il campo [0] per indicare la fine della modifica.

### Modifica alle date

La modifica delle date deve essere effettuata chiedendo solo il giorno nel formato “yyyy/MM/dd” settando fisso le ore 12:00:00.

### Modifica degli esami

Selezionando la modifica degli esami, il programma deve iniziare una richiesta di ricerca degli esami associati mostrando la lista degli esami con il numero di selettore associato, inoltre deve essere presente il selettore 0 che indica il termine di questa modifica.

Una volta selezionato l’esame, bisognerà chiedere se annullare la modifica con 0, rimuovere l’esame con 1 o modificarlo con un qualsiasi altro tasto.

Questa funzionalità di modifica verrà ripetuta fino all’inserimento di 0 da parte dell’utente.

# Requisito D: Visualizzazioni dei dati memorizzati

## Visualizzazione sintetica dei dati del paziente

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

Aggiornare la classe di menù inserendo una nuova voce di funzionalità che indica la visualizzazione dei dati del paziente e abbinare il numero corrispondente alla chiamata della nuova classe di visualizzazione.

La nuova classe dovrà caricare da sessione i dati Paziente, PrestazioniEsame, Malattia.

Tramite StringBuffer, si caricherà in modo sequenziale i dati come segue:

### Dati del paziente

Dalla struttura Paziente in sessione si caricheranno i dati “nome” e “cognome”.

### Prestazioni mediche

Ciclando la lista delle prestazioni mediche si dovranno stampare: “nome”, “data” e, nel caso di prestazioni legate ad esami periodici, anche “esito”.

Il caricamento dell’esame associato dovrà essere fatto tramite idEsame caricando l’esame da mappa in sessione.

### Malattie

Ciclando la lista delle malattie, si dovrà visualizzare “nome” e “data” troncando la data solo al giorno con formato “yyyy/MM/dd” (con un substring all’undicesimo carattere).

## Menù di approfondimento

Dopo aver mostrato i dati del punto 6.1, verrà chiesta un’iterazione all’utente proponendo un menù con 3 opzioni di visualizzazione e un opzione legata al numero 0 per il termine della funzionalità e il ritorno al menù principale.

Per gestire queste visualizzazioni in modo ordinato bisognerà creare una nuova classe di utils chiamata “VisualizzazioneUtils” che esporrà diversi metodi statici pubblici.

Le opzioni saranno:

1. visualizzazione completa dei dati anagrafici;
2. visualizzazione completa di un singolo esame, selezionato tra quelli presenti;
3. visualizzazione completa di una singola malattia, selezionata tra quelle presenti.

### Dati anagrafici

Nella classe di utils, creare un nuovo metodo pubblico, statico, con ritorno void e senza argomenti. Questo caricherà la sessione e prenderà l’oggetto Paziente. Con questo oggetto verrà stampato un messaggio contenente tutte le informazioni anagrafiche del paziente.

Questo metodo di utils verrà invocato quando l’utente seleziona la visualizzazione dei dati anagrafici.

### Visualizzazione del singolo esame

Nella classe di utils, creare un nuovo metodo pubblico, statico, con ritorno void e con un argomento numero che indica l’id esame selezionato.

Questo metodo deve caricare la sessione e da questa prendere la mappa degli esami e fare una *.get()* passando l’id di input. Caricato l’oggetto esame, il metodo stamperà tutti i dati di questo.

Nella classe di menù: VisualizzazioneSinteticaUtente, quando l’utente selezione la visualizzazione del singolo esame, il flusso chiamerà il metodo di ricerca tipologia nella classe di CercaEsame. Una voltà individuato l’esame e caricato l’opportuno id, bisognerà invocare il nuovo metodo di utils appena creato inserendo come input l’id selezionato.

### Visualizzazione singola malattia

Nella classe di utils, creare un nuovo metodo pubblico, statico, con ritorno void e con un argomento numero che indica l’id della malattia selezionata.

Questo metodo deve caricare la sessione e da questa prendere la mappa delle malattie e fare una *.get()* passando l’id di input. Caricato l’oggetto malattia, il metodo stamperà tutti i dati di questo.

Nella classe di menù: VisualizzazioneSinteticaUtente, quando l’utente selezione la visualizzazione della singola malattia, il flusso chiamerà il metodo di ricerca malattia nella classe di CercaEsame. Una volta selezionata la malattia e ritornato l’ID corrispondente, bisognerà invocare il nuovo metodo di utils appena creato inserendo come input l’ID della malattia selezionato dalla ricerca.

## Visualizzazione degli esiti di un esame periodico con esito misurabile

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

Aggiornare la classe di menù inserendo una nuova voce di funzionalità che indica la visualizzazione degli esiti misurabili degli esami periodici e abbinare il numero corrispondente alla chiamata della nuova classe di visualizzazione.

La classe dovrà caricare la sessione dall’ApplicationSession e dovrà iniziare una nuova ricerca dell’esame richiamando il metodo di ricerca tipologia nella classe CercaEsame; una volta selezionato l’esame, il flusso prenderà da sessione l’oggetto corrispondente caricandolo dalla mappa degli esami tramite il *.get(idEsame)*.

### Selezione esame periodico

Bisognerà controllare se l’esame selezionato è di tipo periodico, in caso contrario si dovrà visualizzare un messaggio che indica di selezionare un esame di tipo periodico e ricominciare la ricerca.

Quando viene selezionato un esame periodico, si chiede all’utente se vuole filtrare per date i vari risultati o se vuole prenderli in considerazione tutti con l’inserimento di 0.

### Filtro per date

In caso inserisca un valore diverso da 0, il programma dovrà chiedere 2 date in formato yyyy/MM/dd.

Le date saranno rispettivamente quella di inizio e quella di fine del range di considerazione.

Questo compito verrà affidato a un metodo privato all’interno della classe di logica che ritornerà un array di 2 elementi data oppure null se è stato selezionato di non filtrare.

### Caricamento delle prestazioni

Una volta ottenuto l’ID dell’esame periodico e le date di filtro, si caricherà da sessione la lista delle prestazioni mediche con id Esame pari a quello selezionato e si scarteranno quelle con data precedente all’inizio o successiva alla fine delle date di filtro.

Per compiere questo mestiere, creare un metodo privato che ritorni una lista di prestazioni in cui si passerà l’ID dell’esame, l’array di 2 elementi delle date e la mappa delle prestazioni mediche presa da sessione.

### Visualizzazione delle informazioni