**INTRODUZIONE**

“Vacciniamo” è un progetto sviluppato per Laboratorio B per il corso di laurea in Informatica dell’università degli studi dell’Insubria.

Il progetto è sviluppato in Java 16, usa un’interfaccia grafica costruita con Swing ed è stato sviluppato e testato su:

* Window 10-11
* Mac Os Ventura 13.0 -13.0.1
* Linux popOS

**STRUTTURA GENERALE DELLE CLASSI**

Il progetto è strutturato in due suddivisioni: **“ClientLab”**, che si occupa della gestione grafica e di elaborazione/visualizzazione dei dati lato client e **“ServerLab”** che svolge il medesimo compito lato server.

* **CLIENT LAB:**
  + Elaborazione e visualizzazione Dati:
    - ClientMain (main)
    - View <Interfaccia>
    - ViewImpl
    - Model <Interfaccia>
    - ModelImpl
    - ControllerImpl
    - Proxy
    - ServerInterface <Interfaccia>
  + Gestione Grafica:
    - SceltaOperatoreGUI
    - GUICambiaIP
    - OperatoreGUI
    - RegistraCentroGUI
    - RegistraVaccinatoGUI
    - CittadinoGUI
    - RegisterGUI
    - CercaCentroGUI
    - ElencoCentriGUI
    - LoginGUI
    - EventiAvversiGUI
    - VisualizzaEventiAvversiGUI
    - Colori
* **SERVER LAB**:
  + Elaborazione e visualizzazione dati:
    - ServerMain (main)
    - ServerWorker
    - ViewInterface <interfaccia>
    - ModelInterface <interfaccia>
    - ModelImpl
    - Controller
    - SkeletonInterface <interfaccia>
    - DataBaseConnessione
    - EsecutoreQuery
  + Gestione Grafica:
    - ServerGUI

**CLIENT LAB**

Di seguito verranno elencate le classi lato client, con una breve descrizione di carattere architetturale e implementativo.

**CLIENTMAIN**

La classe principale che contiene il metodo **public static void main (String[] args) {}**, necessario per generare i componenti utili al funzionamento del modulo ClientLab.

**VIEW**

Interfaccia che fornisce l’overload dei metodi alle classi che la implementano.

**VIEWIMPL**

Questa classe estende JFrame ed implementa l’interfaccia View.

Si occupa di gestire tutte le componenti grafiche dell’applicazione, aggiornando anche i singoli elementi, in base ai dati ricevuti nel metodo **updateView ()**.

**MODEL**

Interfaccia che fornisce l’overload dei metodi alle classi che la implementano.

**MODELIMPL**

Questa classe implementa l’interfaccia Model.

Il costruttore salva un riferimento di tipo View passato come parametro e istanzia un riferimento di tipo ServerInterface.

È dotata di un metodo denominato **updateModel ()** che riceve come parametri in input:

* Source 🡪 Il nome dell’oggetto che ha generato l’evento.
* Dati 🡪 inseriti dall’utente tramite interfaccia.

Dopo aver verificato l’oggetto che generato l’evento chiama i propri metodi secondo le condizioni elencate.

Con i dati ricevuti dalla classe Controller, dopo averli elaborati rendendoli compatibili con il formato richiesto, richiama i metodi di ServerInterface che si occuperanno di completare le operazioni richieste.

Infine, grazie al riferimento di tipo View, viene richiamato il metodo **updateView ()** utile per aggiornare, in base alle informazioni passate come parametro, le componenti grafiche.

**CONTROLLERIMPL**

Implementa l’interfaccia ActionListener.

Il costruttore salva i riferimenti di Model e View passati come parametri, e si occupa di catturare e aggiungere ai componenti grafici la possibilità di generare eventi.

All’interno di essa è presente il metodo **actionPerformed ()** che prende in input:

* e 🡪 di tipo ActionEvent, ossia l’evento generato.

Successivamente estrapola il nome dell’oggetto verificando l’istanza di appartenenza e seguendo un serie di condizioni, ricava le informazioni inserite nella View.

Fatto ciò, richiama il metodo del model, che come descritto in precedenza elaborerà i dati sul bottone d’origine.

**SERVERINTERFACE**

Interfaccia contenente la definizione delle operazioni offerte dal server.

**PROXY**

Implementa l’interfaccia ServerInterface.

La comunicazione tra client e server è gestita da questa classe, la quale fa da intermediario tra le esigenze del client e la ricerca di risorse sul server, fornendo dei metodi che per ogni specifica funzionalità, inviano, sottoforma di stringhe le informazioni richieste, le quali viaggiano su dei buffer di tipo **ObjectInput** e **ObjectOutput**, in particolare:

* writeObject () 🡪 metodo che scrive dati sul buffer per essere letti dal server.
* readObject () 🡪 metodo che legge i dati inviati dal server sul buffer.

**CLASSI GRAFICHE**

Classi che gestiscono solo i componenti grafici delle singole schermate.

Dotate di oggetti di tipo JFrame, JPanel, JLabel, JButton ecc.

**SERVER LAB**

Di seguito verranno elencate le classi lato server, con una breve descrizione di carattere architetturale e implementativo.

**SERVERMAIN**

La classe principale che contiene il metodo **public static void main (String[] args) {}**, necessario per generare i componenti utili al funzionamento del modulo ServerLab.

**VIEWINTERFACE**

Interfaccia che fornisce l’overload dei metodi alle classi che la implementano.

**SERVERGUI**

Implementa l’interfaccia ViewInterface.

Questa classe si occupa di gestire i componenti grafici della schermata di avvio del server.

Contiene il metodo **updateView** (), necessario ad aggiornare lo stato della schermata in base al valore del campo dati passato come parametro.

**MODELINTERFACE**

Interfaccia che fornisce l’overload dei metodi alle classi che la implementano.

**MODELIMPL**

Implementa l’interfaccia model.

Il costruttore salva un riferimento di tipo View passato come parametro e istanzia un riferimento di tipo SkeletonInterface, che si occuperà della comunicazione con il Server, gestendo le richieste.

La classe è dotata del metodo **avvioServer ()**, il quale attende la connessione del client e lancia un thread di gestione dei task ogni qual volta qualcuno si connette.

È presente, inoltre, il metodo **updateModel ()**, che prende in input i dati elaborati dal controller.

Questi, dopo essere stati modellati secondo le compatibilità richieste (generalmente salvati su liste), vengono passate come parametri ai metodi di ServerInterface, in grado di completare le operazioni.

Infine, viene richiamato il metodo **updateView ()** con lo scopo di aggiornare le componenti grafiche.

**CONTROLLER**

Questa classe implementa l’interfaccia actionListener.

Il costruttore salva i riferimenti di Model e View passati come parametri, e si occupa di catturare e aggiungere ai componenti grafici la possibilità di generare eventi.

All’interno di essa è presente il metodo **actionPerformed ()**, ereditato da ActionListener che prende in input:

* e 🡪 di tipo ActionEvent, ossia l’evento generato.

Successivamente estrapola il nome dell’oggetto (in questo caso si occupa di intercettare solo il bottone di avvio del server), ricavando le informazioni inserite nella schermata principale ServerGUI.

Fatto ciò, richiama il metodo del model, che come descritto in precedenza elaborerà i dati sul bottone d’origine.

**SKELETONINTERFACE**

Questa interfaccia contiene la definizione dei metodi necessari al funzionamento del sistema.

**SERVERWORKER**

Estende la classe Thread.

Al costruttore vengono passati come parametri:

* socket 🡪 connessione con il client.
* swi 🡪 riferimento di tipo ServerInterface, condiviso con le altre istanze di ServerWorker.

I thread vengono lanciati, con il metodo **start ()**, ereditato dal superclasse, quando un nuovo client si connette, in modo tale da gestire in parallelo operazioni da parte di più client.

Viene inoltre ereditato il metodo **run ()** il quale, si occupa attraverso il metodo **readObject ()** richiamato su un oggetto di tipo ObjectInput di leggere i dati inviati dal client, elaborare le richieste sulla risorsa condivisa di tipo ServerInterface, ed infine, inviare l’esito attraverso il metodo

**writeObject ()**, sempre di proprietà di ObjectOutput.

**DATABASECONNESSIONE**

Classe che fornisce riferimenti di tipo:

* connection
* statement

Attraverso il metodo **getConnection ()** si occupa della connessione al database di PostegreSQL dove sarà gestita la parte back-end dell’applicazione.

La connessione avverrà mediante username e password del DB, specificando la porta dove questo girerà.

Il metodo **getStatement ()** svolge la funzione di restituire l’oggetto statement appartenente all’oggetto connection.

Lo statement eseguirà l’interrogazione al DB sottoforma di query.

**ESECUTOREQUERY**

Questa classe implementa l’interfaccia SkeletonInterface.

Il costruttore salva riferimenti di tipo Connection e Statement relativi alla connessione con il Database.

ModelImpl crea un’istanza di questa classe e fornisce il riferimento dell’oggetto appena creato a tutte le istanze della classe ServerWorker.

Il compito principale è quello di fornire tutti i metodi necessari al funzionamento del sistema, ogni metodo, in base alle esigenze da parte del client, richiede come parametri i dati necessari all’esecuzione delle query, che completerà grazie al metodo **executeQuery ()** dell’oggetto di tipo Statement, ritornando un valore in base allo stato finale.