



SAPIENZA
UNIVERSITÀ DI ROMA

Department of Informatics

Wikipedia Web Scraper: Roman Emperors

Programming Methodologies

Authors

Chiarello Enrico

Ciafardoni Luca

Crea Michelangelo

Di Nella Daniel

Academic Year 2021/2022

Contents

1	<u>Introduzione al progetto</u>	3
2	<u>Descrizione delle classi</u>	4
3	<u>Descrizione delle funzionalità</u>	17
4	<u>Manuale GUI</u>	19
5	<u>Referenti di sviluppo</u>	26

Introduzione al progetto

Per la realizzazione del progetto “ Web Scraper Wikipedia sugli Imperatori Romani ” a cura di Chiarello Enrico, Ciafaldoni Luca, Crea Michelangelo e Di Nella Daniel; il lavoro è stato suddiviso in tre blocchi rispettando in particolare l'utilizzo della modularità e quindi una esplicita separazione tra la parte logica e la parte grafica.

Il browser di riferimento è “Google Chrome” pertanto l'applicazione funziona correttamente SOLO se la macchina su cui si avvia il programma ne ha effettuato l'installazione.

Parte logica:

Un ottimo lavoro è stato svolto da Crea Michelangelo e Di Nella Daniel che si sono occupati della creazione un codice il quale estrae le informazioni dall'enciclopedia online più comune al mondo: Wikipedia. In particolare l'algoritmo svolto si occupa di analizzare i codici HTML della pagina web cercata, selezionare solo le informazioni inerenti al progetto, estrapolarle e processarle.

Parte intermedia:

Determinante è stato il lavoro di Ciafaldoni Luca il quale, partendo proprio dalle informazioni ricavate nella parte logica e processandole a suo modo, ha scritto un codice ricorsivo in grado di generare, facendo cura anche dei minimi dettagli, un'immagine completa dell'albero genealogico dotata di informazioni riguardanti il loro ruolo nella dinastia e la data di appartenenza.

Parte grafica:

Se l'utente è in grado di usare l'applicazione in modo facile e intuitivo questo è dovuto al lavoro meticoloso di Chiarello Enrico. La parte grafica infatti si occupa di soddisfare le richieste dell'utente che può eseguire la ricerca di una intera dinastia ma anche di sottoalberi di un qualsiasi membro ordinario. Le immagini che si possono creare grazie al lavoro svolto in precedenza, vengono mostrate a schermo.

Descrizione delle classi

Classe Person

La classe Person viene usata per la creazione degli oggetti di tipo persona.

Tali oggetti presentano i seguenti attributi:

- `ChromeOptions options`: tale attributo viene utilizzato per impostare delle informazioni alla finestra che viene aperta, nel nostro caso rendiamo la finestra invisibile quando viene aperta.
- `WebDriver driver`: tale attributo viene utilizzato per aprire la pagina wikipedia della persona che è stata creata. Una volta che vengono estratti tutti i dati dalla pagina essa viene chiusa.
- `String nome`: tale attributo rappresenta il nome dell'oggetto persona.
- `String link`: tale attributo rappresenta il link alla pagina wikipedia dell'oggetto persona.
- `String dinastia`: tale attributo rappresenta il nome della dinastia di appartenenza dell'oggetto persona.
- `String padre`: tale attributo rappresenta il nome del padre dell'oggetto persona.
- `String madre`: tale attributo rappresenta il nome della madre dell'oggetto persona.
- `ArrayList figli`: tale attributo rappresenta una lista con al suo interno altri oggetti di tipo persona che rappresentano i figli dell'oggetto persona.
- `ArrayList coniuge`: tale attributo rappresenta una lista di nomi dei coniugi o consorti dell'oggetto di tipo persona.
- `List<WebElement> tab`: tale attributo rappresenta una lista di `WebElement`, in particolare dentro questa lista troviamo tutti gli elementi presenti nella tabella della pagina wikipedia della persona che viene cercata. Questa lista viene usata per la ricerca delle informazioni della persona.
- `boolean checkImperatore`: questo attributo di tipo booleano viene usato per verificare se la persona è oppure no un imperatore.

La Classe Person inoltre ha vari metodi per impostare o modificare i vari attributi precedentemente elencati:

Metodo del Costruttore:

- `public Person(String nome, String link)`: Il metodo del costruttore riceve come parametri il nome ed il link per creare l'oggetto di tipo Person. Inoltre inizializza alcuni parametri. Assegna il nome ed il link ai valori corrispondenti dell'oggetto, effettua i settaggi per le impostazioni della pagina ed apre il sito wikipedia della persona cercata. Infine inizializza l'attributo `tab` mettendo in esso la lista di tutti i campi presenti nella tabella che possiede le informazioni della persona cercata.

Altri metodi:

- `public ArrayList getFigli()`: tale metodo è utilizzato per restituire la lista dei figli dell'oggetto Person.

- `public String getDinastia():` tale metodo è utilizzato per restituire il nome della dinastia appartenente alla persona.
- `public String getNome():` tale metodo è utilizzato per restituire il nome dell'oggetto persona.
- `public String getPadre():` tale metodo è utilizzato per restituire il nome del padre dell'oggetto persona.
- `public String getMadre():` tale metodo viene utilizzato per restituire il nome della madre dell'oggetto di tipo persona.
- `public ArrayList getConiuge():` tale metodo viene utilizzato per restituire la lista dei nomi dei coniugi dell'oggetto di tipo persona
- `public boolean checkImp():` tale metodo viene utilizzato per restituire se una persona è un imperatore oppure no.
- `public List<WebElement> getTab():` Tale metodo viene utilizzato per restituire la lista dei campi nelle tabelle presenti nella pagina wikipedia della persona.
- `public WebDriver getDriver():` tale metodo viene usato per restituire il WebDriver della pagina wikipedia dell'oggetto persona.
- `public void setDinastia():` tale metodo si occupa dell'assegnazione della dinastia all'oggetto di tipo persona, nel caso in cui questo campo non viene trovato l'attributo dinastia verrà assegnato ad un valore vuoto ("").
- `public void setFigli():` questo metodo si occupa di riempire ricorsivamente la lista dei figli. All'interno di questa lista ci sono degli oggetti di tipo persona con a loro volta tutti gli attributi che ha un oggetto di tipo persona. Una volta creato un nuovo oggetto di tipo persona viene fatta la verifica se tale persona appartiene alla dinastia che viene cercata. Se appartiene alla stessa dinastia viene aggiunto alla lista dei figli, in caso contrario l'oggetto viene scartato. In alcuni casi la pagina wikipedia non presenta il campo inerente ai figli, in tal caso il metodo cesserà subito la sua esecuzione.

- `public void setPadre()`: tale metodo si occupa dell'assegnazione dell'attributo padre il suo nome. Se nella pagina wikipedia tale campo non è presente all'attributo padre verrà assegnata la stringa vuota (`""`).
- `public void setMadre()`: tale metodo viene usato per l'assegnazione all'attributo madre il nome di tale persona. Nel caso in cui nella pagina wikipedia questo campo non sia presente tale attributo verrà assegnato ad una stringa vuota (`""`).
- `public void setConiuge()`: tale metodo si occupa del riempimento della lista dei coniugi o consorti dell'oggetto di tipo persona. Nel caso in cui questo campo non è presente nella pagina wikipedia questa lista verrà assegnata ad una lista vuota.
- `public void setCheckImperatore()`: tale metodo viene utilizzato per verificare se l'oggetto di tipo persona è un imperatore.
- `public void closeDriver(WebDriver driver)`: tale metodo serve per chiudere e terminare la ricerca nella pagina wikipedia della persona cercata.

Classe Imperatore

La classe Imperatore viene usata nel caso in cui la persona processata è stata un imperatore. La classe Imperatore estende la classe persona, perciò eredita tutti i metodi che ha la classe persona precedentemente descritti ed ha in più l'attributo mandato con i relativi metodi per impostare e restituire tale attributo.

Attributi della classe Imperatore:

- `String mandato`: tale attributo viene utilizzato per mostrare gli anni in cui tale imperatore è stato al potere.

I metodi della classe imperatore sono:

Metodo del costruttore:

- `public Imperatore(String nome, String link)`: come nella classe persona gli vengono passati come parametri il nome ed il link. Nel metodo viene richiamata la classe Persona per la creazione dell'oggetto.

Altri metodi nella classe imperatore:

- `public void setMandato()`: tale metodo viene utilizzato per cercare nella tabella presente nella pagina wikipedia il campo inerente al mandato che ha avuto tale imperatore. All'interno di questo



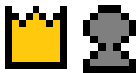
metodo è presente anche una manipolazione di una stringa provvisoria per andare ad eliminare contenuti superflui presenti nella pagina wikipedia. Una volta eliminati gli elementi superflui viene assegnata all'attributo mandato la variabile provvisoria usata per la manipolazione della stringa.

- `public String getMandato()`: tale metodo viene utilizzato per restituire l'attributo mandato, che sarà una stringa con le date di inizio e fine mandato.

Classe Node

La classe Node viene usata per creare contenere le informazioni relative al disegno (è un rettangolo) di quest'ultimi.

Tali oggetti presentano i seguenti attributi:

- `int tile_x`; tale attributo indica la "mattonella" che occupano in ampiezza
- `int tile_y`; tale attributo indica la "mattonella" che occupano in lunghezza
- `int[2] top`; tale attributo contiene le coordinate x e y in pixel della cima del rettangolo, utile per il disegno e il collegamento delle frecce entranti.
- `int[2] bottom`; tale attributo contiene le coordinate x e y in pixel del fondo del rettangolo, utile per il disegno e il collegamento delle frecce uscenti.
- `int n_children`; tale attributo indica i figli di un nodo e di conseguenza, quante frecce uscenti avrà.
- `boolean imp`; tale attributo indica se il nodo appartiene ad un imperatore, in quel caso il suo nome sarà affiancato da un'icona a forma di corona altrimenti sarà un membro ordinario.
- `String text`; tale attributo contiene il testo da scrivere all'interno del nodo, quindi il nome del membro che contiene e i suoi coniugi, consorti e amanti.
- `final int RECT_B`; tale attributo indica la dimensione costante della base.
- `final int RECT_H`; tale attributo indica la dimensione costante dell'altezza.

I metodi della classe Node:

Metodo del costruttore:

- `public Node(String t, int tile_y, int tile_x, ArrayList<Integer> members_per_height, int img_width)`:
A partire dai parametri forniti, è possibile calcolare tutti i valori degli attributi sopra citati

Altri metodi nella classe Node:

- `public int getTileX()`: esegue il return dell'attributo `tile_x`.

- `public int getTileY():` esegue il return dell'attributo `tile_y`.
- `public int getN_children():` esegue il return dell'attributo `n_children`.
- `public boolean hasImp():` esegue di "Vero" se il nodo contiene un imperatore
- `public String getText():` esegue il return dell'attributo `text`
- `public int getRECT_H():` esegue il return dell'attributo `RECT_H`
- `public int getRECT_B():` esegue il return dell'attributo `RECT_B`
- `public int getTopX():` esegue il return dell'attributo `top` all'indice 1
- `public int getTopY():` esegue il return dell'attributo `top` all'indice 0
- `public int getBtmX():` esegue il return dell'attributo `bottom` all'indice 1
- `public int getBtmY():` esegue il return dell'attributo `bottom` all'indice 0

Classe StringProcessor

La classe `StringProcessor` è concepita come una classe "Utility" (dunque static) per processare le informazioni estratte dal Web Scraper ed aiutare le altre classi in operazioni per la processazione ulteriore della String Codifica. Questa classe è stata creata per rispettare il principio SOLID noto come Single Responsibility.

Tale classe presenta il seguente attributi:

- `static String codifica;` è un testo indentato che contiene la struttura dell'albero genealogico finale con relative informazioni di ciascun membro ad ogni riga. E' un attributo concepito come statico per mantenere l'ordine cronologico all'interno del risultato finale.

Rispettivamente, il formato di ciascuna riga e un esempio di codifica:

Emperor Y / N	Name Mandatory	Reign Optional	Spouses Optional	N° of Children Optional
V	V	V	V	V
"*"/""	"Full name"	" (n-n)"	" + [List]"	" N_F:n"

Example:

```

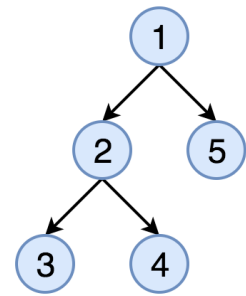
**Vespasiano (69-79) + [Flavia Domitilla, Caenis] N_F:3 /n
  *Tito (79-81) + [Arrecina Tertulla, Marcia Furnilla] N_F:2 /n
    Giulia Flavia (79-81) + [Tito Flavio Sabino] /n
    Flavia /n
  *Domiziano (81-96) + [Domizia Longina] N_F:2 /n
    Flavio Clemente /n
    Flavia Domitilla /n
    Flavia Domitilla minore"

```

Ogni riga presenta uno spazio (" ") alla fine eccetto l'ultima e la riga iniziale sarà "null", queste imprecisioni saranno gestite dopo la creazione di codifica.

Metodi della classe StringProcessor:

- `public static void processString(Imperatore root, String tabs)`: tale metodo assembla la linea relativa alla radice fornita come parametro secondo i criteri stabiliti nel formato della linea della codifica.
Se la radice ha dei figli la funzione viene chiamata ricorsivamente usandolo come parametro e ciascun figlio viene tipizzato come istanza di `Person` o `Imperatore` a seconda del risultato di `getImp()`, mentre il numero di `tab (/t)` per l'indentazione viene incrementato di 1 rispetto alla radice utilizzata. La visita dell'albero segue la logica Pre-Order.
- `public static void processString(Person root, String tabs)`: tale metodo esegue le medesime istruzioni della sua versione quando `root` è di tipo `Imperatore`, la differenza sostanziale è data dal fatto che non viene aggiunto l'asterisco per indicare la presenza di un membro ordinario nella radice.
- `public static ArrayList<String> removeDuplicates(ArrayList<String> lines)`: tale metodo rimuove le linee di testo duplicate e ciò avviene quando due membri della stessa dinastia decidono di sposarsi e ciò verifica la creazione di 2 sottoalberi identici e ridondanti. Come sottolineato precedentemente, ogni riga presenta uno spazio alla fine eccetto l'ultima, per evitare un duplicato all'ultima riga viene effettuato un controllo aggiuntivo ed eventualmente quest'ultima viene rimossa. Infine l'`ArrayList` senza duplicati viene restituita.
- `public static int countMatches(String str, String target)`: tale metodo conta il numero di occorrenze di una substring (`target`) in una data String (`str`), per poi eseguirne il return.
- `public static ArrayList<String> separate(String toSeparate, String el)`: tale metodo separa una String (`toSeparate`) utilizzando un parametro come criterio di suddivisione (`el`). Infine l'`ArrayList` di quanto suddiviso viene restituito.
- `public static String getCodifica()`: restituisce codifica senza la prima riga (null) ed spazio iniziale.
- `public static void resetCodifica()`: imposta l'attributo `codifica` come una String vuota ("").



Classe TreelImage

La classe `TreelImage` è concepita come la classe responsabile della creazione dei nodi e in seguito dell'immagine finale contenente l'albero genealogico, è una classe di static poiché è superfluo crearne istanze dal momento che il risultato finale viene salvato su un supporto esterno.

Tale classe presenta i seguenti attributi:

- `static String codifica`; tale attributo viene fornito da `StringProcessor` e viene passato per essere suddiviso riga per riga.
- `static BufferedImage img`; è l'immagine sulla quale verrà disegnato l'albero genealogico ed in seguito salvata.

- static Graphics2D g2d; è l'oggetto actor responsabile delle operazioni di disegno sull'attributo img.
- static int tree_height; tale attributo indica da quanti livelli è composto l'albero.
- static int tree_width; tale attributo indica quanto è ampio l'albero.
- static int img_height; tale attributo indica la lunghezza dell'immagine in pixel.
- static int img_width; tale attributo indica la larghezza dell'immagine in pixel.
- static ArrayList<Integer> members_per_height; tale ArrayList indica quanti membri ci sono per ogni livello, l'indice di ogni elemento indica un livello di profondità.
- static ArrayList<ArrayList<Integer>> tree_tiles_map; tale ArrayList indica i Tile X e Y di ciascun nodo all'interno di nodes_list mantenendo lo stesso ordine.
- Static ArrayList<Node> nodes_list; tale ArrayList contiene tutti gli oggetti di classe Node che verranno disegnati utilizzando le informazioni in essi contenuti.

Metodi della classe TreelImage:

- public static void createlImage(int cont): è il metodo principale di TreelImage ed esegue il calcolo dei valori degli attributi sopra citati mediante la chiamata di altri metodi, infine crea l'immagine e la salva utilizzando il parametro cont come parte del nome ("tree<cont>.png").
- public static int calcHeight(ArrayList<String> lines): questo metodo viene richiamato da createlImage e serve a calcolare l'altezza dell'albero, per farlo è necessario trovare la riga col maggior numero di tab utilizzati per l'indentazione. Una volta trovato questo numero viene restituito.
- public static ArrayList<Integer> calcWidthPerHeight(ArrayList<String> lines): questo metodo viene richiamato da createlImage ed inizializza un ArrayList contenente soli zeri quanti sono i livelli dell'albero, in seguito viene analizzata ogni riga della codifica e calcolato il numero di tab presenti in quest'ultima e il valore all'indice uguale al numero appena trovato viene incrementato di uno. Infine questa ArrayList viene restituita.
- public static ArrayList<ArrayList<Integer>> calcNodesTiles(ArrayList<String> lines): questo metodo viene richiamato da createlImage ed inizializza un ArrayList contenente coppie di valori relative al Tile Y e il Tile X per ciascuna riga della codifica. Infine questa ArrayList viene restituita.
- public static void createNodes(ArrayList<String> lines): questo metodo viene richiamato da createlImage ed inizialmente svuota l'ArrayList contenente i nodi generati da un eventuale ricerca avvenuta precedentemente ed in seguito la riempie di oggetto di classe Node utilizzando quanto calcolato in precedenza. Infine questa ArrayList viene restituita.

- `public static void drawNodes(ArrayList<Node> nodes_list, Graphics2D g2d)`: questo metodo viene richiamato da `createImage` ed è responsabile del disegno da parte di `g2d` dello sfondo, dei rettangoli che indicano i nodi, scrittura del testo interno, disegno icone e le frecce che li collegano.
- `public static void setCodifica(String s)`: serve all'inizializzazione del valore contenuto in `codifica`, viene chiamato all'inizio di ogni ricerca

Classe Home

Eseguendo la classe `Home` apparirà sullo schermo un frame.

`Home`, perciò, è la finestra da cui l'utente dovrà partire prima di visualizzare qualsiasi immagine e quindi prima di passare alla classe `Secondary`. È una classe di introduzione dove è possibile capire lo scopo e le funzionalità del progetto.

Tale classe presenta i seguenti attributi:

- `private javax.swing.JTextField txtName`: tale attributo si presenta graficamente come una casella di testo su cui è possibile scrivere da tastiera.

- `private javax.swing.JButton btnSearchName`: tale attributo nella grafica rappresenta il bottone per effettuare ricerche scrivendo da tastiera.

- `private javax.swing.JComboBox<String> jComboBox`: tale attributo permette di selezionare graficamente una dinastia e si presenta come un menu a tendina.

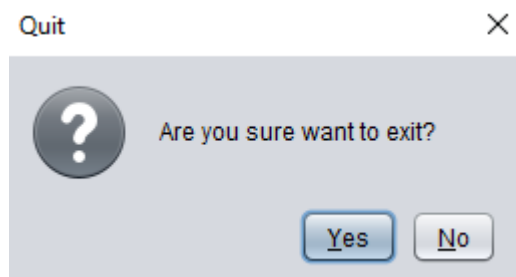
- `private javax.swing.JButton btnSearchComb`: tale attributo riguarda il bottone in basso al menu a tendina e fa partire la ricerca.

- `private javax.swing.JLabel lbTitle`: tale attributo si riferisce alla scrittura fissata in alto al frame.
- `private javax.swing.JMenuBar menu`: tale attributo si riferisce alla barra di menu in alto al frame.
- `private javax.swing.JMenu voice1`: tale attributo rappresenta l'unica voce del menu che in grafica si presenta con la scritta "info" dalla quale si aprirà un pannello per alcune chiarificazioni.

- `private javax.swing.JPanel jPanel1`: tale attributo racchiude gli attributi utili alla ricerca per dinastia ed ha il compito ordinare la grafica.
- `private javax.swing.JPanel jPanel2`: tale attributo racchiude gli attributi utili alla ricerca per nome ed ha il compito ordinare la grafica.

Metodi della classe Home:

- `public Home()`: è il costruttore che chiama la funzione `initComponents()` e definisce colore e posizione della finestra sullo schermo.
- `private void initComponents()`: Questo metodo viene chiamato dall'interno del costruttore per inizializzare il form. In particolare si occupa del posizionamento, dimensione, sfondo, colore e layout in generale di ogni attributo grafico. Dichiara inoltre gli ascoltatori i quali hanno il compito di scatenare eventi e quindi richiamare metodi.
- `private void txtNameMouseClicked(java.awt.event.MouseEvent evt)`: Quando l'utente fa click sull'area di testo della ricerca per nome, la scritta in grigio "Write a name..." viene rimossa e il colore viene impostato sul nero.
- `private void voice1MouseClicked(java.awt.event.MouseEvent evt)`: Quando l'utente fa click sull'icona "info" si apre una finestra con una breve descrizione.
- `private void btnSearchNameActionPerformed(java.awt.event.ActionEvent evt)`: In questo metodo è possibile effettuare la prima ricerca (scrivendo il nome) che segue la creazione di un'immagine. Per visualizzare correttamente lo storico questo metodo cancella la cronologia delle ricerche precedenti. Se l'input è corretto, da qui si passa al secondo frame.
- `private void btnSearchCombActionPerformed(java.awt.event.ActionEvent evt)`: In questo metodo è possibile effettuare la prima ricerca (selezionando la dinastia dal menu a tendina) che segue la creazione di un'immagine. Per visualizzare correttamente lo storico questo metodo cancella la cronologia delle ricerche precedenti. Se l'input è corretto, da qui si passa al secondo frame.
- `private void formWindowClosing(java.awt.event.WindowEvent evt)`: Viene chiamato quando si tenta di chiudere la cornice. Infatti, quando si fa clic sulla "x" viene generato un messaggio di conferma.

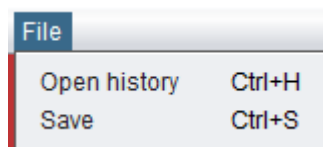


Classe Secondary

Dalla classe Home, effettuando la prima ricerca, si passerà automaticamente alla classe Secondary, la quale ha la funzione di mostrare a schermo l'immagine dell'albero genealogico dell'imperatore appena cercato e di proseguire con la ricerca aggiornando man mano l'immagine al centro della finestra. Inoltre è possibile effettuare salvataggi delle immagini e visualizzare la cronologia.

Tale classe presenta il seguente attributi:

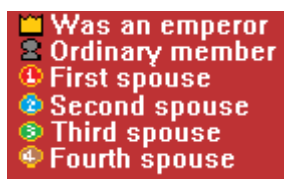
- `private javax.swing.JComboBox<String> jComboBox1`: tale attributo permette di selezionare graficamente una dinastia e si presenta come un menu a tendina.
- `private javax.swing.JButton btnSearchComb`: tale attributo riguarda il bottone in basso al menu a tendina e fa partire la ricerca.
- `private javax.swing.JTextField txtName`: tale attributo si presenta graficamente come una casella di testo su cui è possibile scrivere da tastiera.
- `private javax.swing.JButton btnSerchName`: tale attributo nella grafica rappresenta il bottone per effettuare ricerche scrivendo da tastiera.
- `private javax.swing.JMenuBar jMenuBar`: tale attributo si riferisce alla barra in alto alla finestra contenente il menu.
- `private javax.swing.JMenu jMenu`: tale attributo si riferisce all'unico bottone della barra del menu e contiene a suo volta due altri bottoni.



- `private javax.swing.JMenuItem open`: bottone di `jMenu` che si occupa dell'apertura della cronologia.
- `private javax.swing.JMenuItem save`: bottone di `jMenu` che si occupa del salvataggio dell'immagine appena creata.
- `private javax.swing.JFileChooser jFileChooser`: viene utilizzato per selezionare il percorso della cartella su cui salvare l'immagine.
- `private javax.swing.JLabel lbTitle1`: tale attributo si riferisce alla scritta fissata in alto al frame.
- `private javax.swing.JLabel lbTitle2`: tale attributo si riferisce alla scritta fissata in basso a sinistra.

CHOOSE THE DYNASTY:

- `private javax.swing.JLabel lbTitle3`: tale attributo si riferisce alla scritta fissata in basso a destra.
- `private javax.swing.JLabel lbImage`: tale attributo permette di visualizzare sul frame l'immagine ottenuta riguardante l'albero genealogico.
- `private javax.swing.JScrollPane jScrollPane`: l'immagine si poggia su tale attributo il quale vedendo le sue dimensioni stabilisce se c'è bisogno o meno di uno scroll.
- `Private javax.swing.JLabel lbLegend`: tale attributo si riferisce all'immagine della legenda fissata in alto a destra.



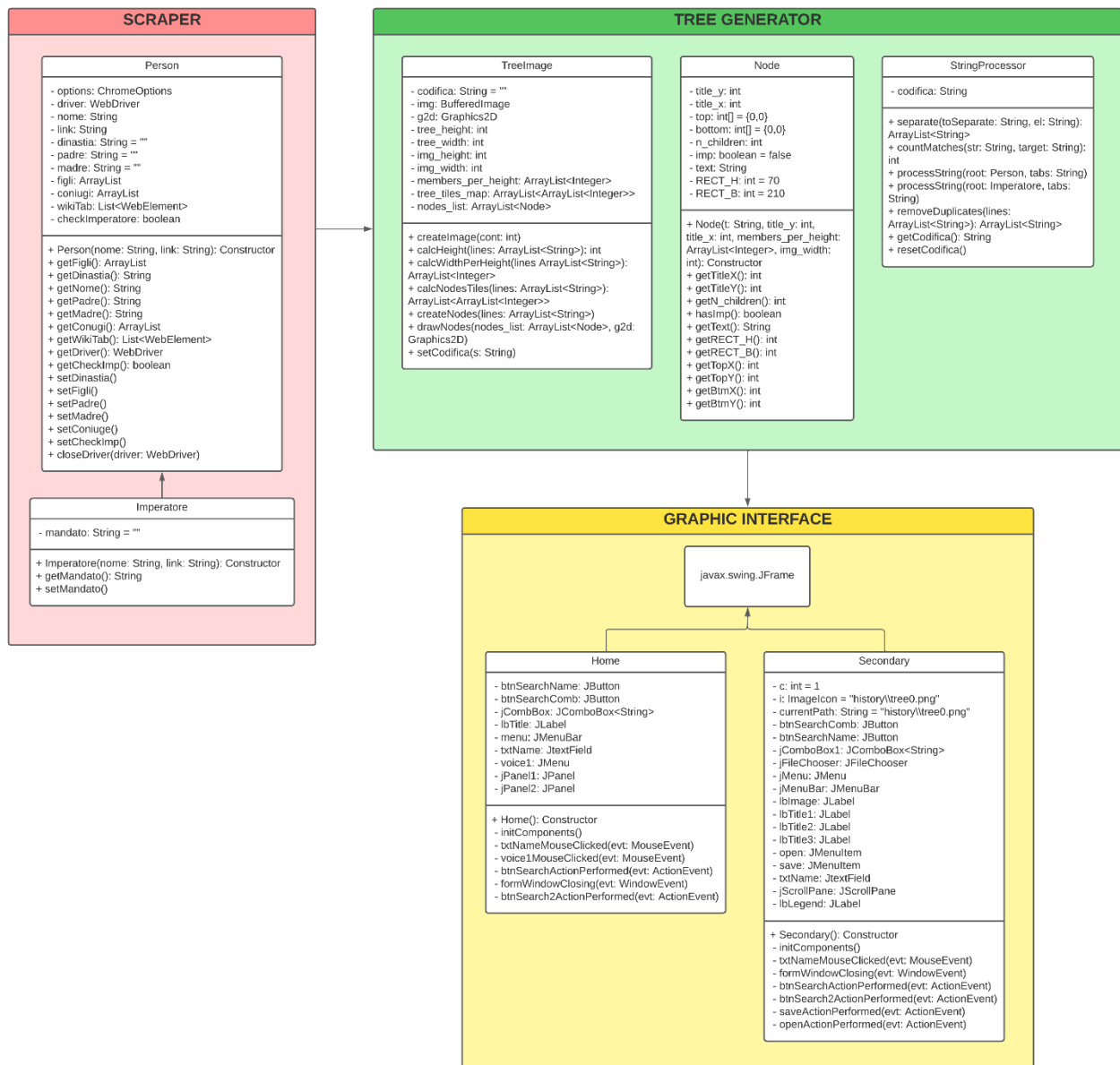
- `private int c = 1`: contatore che permette di creare e salvare immagini con path tutti diversi.
- `private ImageIcon i = new ImageIcon("history\\tree0.png")`: l'icona dell'immagine di start.
- `private String currentPath = "history\\tree0.png"`: tale attributo si riferisce al percorso dell'immagine iniziale del frame e sarà sempre l'immagine zero.

Metodi della classe Secondary:

- `public Secondary()`: è il costruttore che chiama la funzione `initComponents()` e definisce colore e posizione della finestra sullo schermo.
- `private void initComponents()`: Questo metodo viene chiamato dall'interno del costruttore per inizializzare il form. In particolare si occupa del posizionamento, dimensione, sfondo, colore e layout in generale di ogni attributo grafico. Dichiara inoltre gli ascoltatori i quali hanno il compito di scatenare eventi e quindi richiamare metodi.
- `private void txtNameMouseClicked(java.awt.event.MouseEvent evt)`: Quando l'utente fa click sull'area di testo della ricerca per nome, la scritta in grigio "Write a name..." viene rimossa e il colore viene impostato sul nero.
- `private void formWindowClosing(java.awt.event.WindowEvent evt)`: Viene chiamato quando si tenta di chiudere la cornice. Infatti, quando si fa clic sulla "x" viene generato un messaggio di conferma.
- `private void btnSerchNameActionPerformed(java.awt.event.ActionEvent evt)`: In questo metodo, si esegue la ricerca per nome. Se l'input è corretto, la nuova immagine creata viene visualizzata sullo schermo. Le immagini che vengono create di volta in volta sono conservate nello storico della cronologia e sono numerate in base al contatore.

- `private void btnSearchCombActionPerformed(java.awt.event.ActionEvent evt)`: In questo metodo, si esegue la ricerca per dinastia dal menù a tendina. La nuova immagine creata viene visualizzata sullo schermo. Le immagini che vengono create di volta in volta sono conservate nello storico della cronologia e sono numerate in base al contatore.
- `private void saveActionPerformed(java.awt.event.ActionEvent evt)`: Consente di selezionare il percorso di una cartella in cui verrà quindi salvata l'immagine corrente.
- `private void openActionPerformed(java.awt.event.ActionEvent evt)`: Apre la cartella dello storico per visualizzare tutta la cronologia dall'avvio del programma.

UML



Il progetto si impegna a rispettare i principi SOLID della programmazione ad oggetti, infatti:

Single Responsibility:

Invita a sviluppare più classi che hanno una unica funzionalità ben specifica anziché una singola classe che fa molteplici cose.

Open-Closed principle:

Il nome deriva dalla frase "Open for extension, closed for modification". In sostanza, questo principio ci invita a estendere la funzionalità di una classe/funzione invece di modificarla direttamente.

Liskov Substitution:

Questo principio afferma che in un buon codice OO si possono sostituire variabili con i loro sotto-tipi senza causare errori.

Interface segregation:

Il principio afferma che una classe non dovrebbe essere mai forzata ad implementare un metodo che non utilizza.

Dependency Inversion:

Principio che invita a creare classi che non dipendono da implementazioni concrete ma da interfacce o astrazioni.

Descrizione delle funzionalità

Prima di creare l'albero andiamo a estrarre tutte le informazioni che sono presenti su wikipedia che poi verranno utilizzate per la creazione dell'albero e la descrizione delle particolarità dei nodi che sono presenti sull'albero, come ad esempio la lista dei coniugi o consorti della persona o la rappresentazione di una corona nel caso in cui nel nodo è presente un imperatore.

Per la creazione iniziale viene inizializzato un oggetto che sarà di tipo Person o di tipo Imperatore in base al tipo di persona che si incontra, successivamente vengono impostati tutti i suoi parametri (come nome, padre, madre, ecc.) e grazie al metodo figli verrà creata tutta la sua dinastia.

```
String nome = r.getText();
String link = r.getAttribute("href");
Person per = new Person(nome, link);
per.setCheckImperatore();
if(per.checkImp()) {
    per.closeDriver(per.getDriver());
    Imperatore imp = new Imperatore(nome, link, true);
    imp.setDinastia();
    imp.setPadre();
    imp.setMadre();
    imp.setConiuge();
    imp.setCheckImperatore();
    imp.setMandato();

    if (imp.getDinastia() != "") {
        imp.setFigli();
    }

    // ADD SON OBJECT TO SONS' ARRAYLIST OF THE DAD
    this.figli.add(imp);

    imp.closeDriver(imp.getDriver());
}
else {
    per.setDinastia();
    per.setPadre();
    per.setMadre();
    per.setConiuge();

    if (this.dinastia != "") {
        per.setFigli();
    }

    // ADD SON OBJECT TO SONS' ARRAYLIST OF THE DAD
    this.figli.add(per);

    per.closeDriver(per.getDriver());
}
```

Nel caso in cui il figlio non appartiene più alla dinastia il figlio non verrà messo nella lista e la ricerca terminerà.

Una volta terminata l'estrazione di tutte le informazioni relative all'albero genealogico quest'ultimo sarà radicato per intero a partire nella variabile "imp" o "per", dipendentemente dal fatto che quest'ultimo inizi da un membro che è stato imperatore/imperatrice oppure un membro ordinario della famiglia.

A questo punto è necessario codificare e processare la struttura dell'albero in una variabile di tipo String con indentazioni per rendere le informazioni estratte più maneggevoli per i passi successivi. Un esempio di codifica è il seguente:

```
"*Vespasiano (69-79) + [Flavia Domitilla, Caenis] N_F:3      /n
  *Tito (79-81) + [Arrecina Tertulla, Marcia Furnilla] N_F:2  /n
    Giulia Flavia (79-81) + [Tito Flavio Sabino]              /n
    Flavia                                                      /n
  *Domiziano (81-96) + [Domizia Longina] N_F:2               /n
    Flavio Clemente                                           /n
    Flavia Domitilla                                           /n
    Flavia Domitilla minore"
```

In cui ogni riga presenta la seguente struttura:

Emperor	Name	Reign	Spouses	N° of Children
Y / N	Mandatory	Optional	Optional	Optional
V	V	V	V	V
"*" / ""	"Full name"	" (n-n) "	" + [List] "	" N_F:n "

Responsabile di questa procedura è la classe StringProcessor mediante il metodo processString, che dato in ingresso il membro in cui è radicato l'intero albero, eseguirà una visita Pre-Order di quest'ultimo e assemblerà una riga della String finale per ogni chiamata ricorsiva della funzione. Ogni chiamata ricorsiva si occuperà di un membro specifico che può essere di classe Person o Imperatore e di conseguenza la realizzazione della riga verrà gestita diversamente (informazioni riportate nella descrizione della classe).

In seguito a tale operazione verranno rimosse le righe duplicate per evitare la ridondanza di sottoalberi nel risultato finale, ovvero l'immagine che si andrà a realizzare.

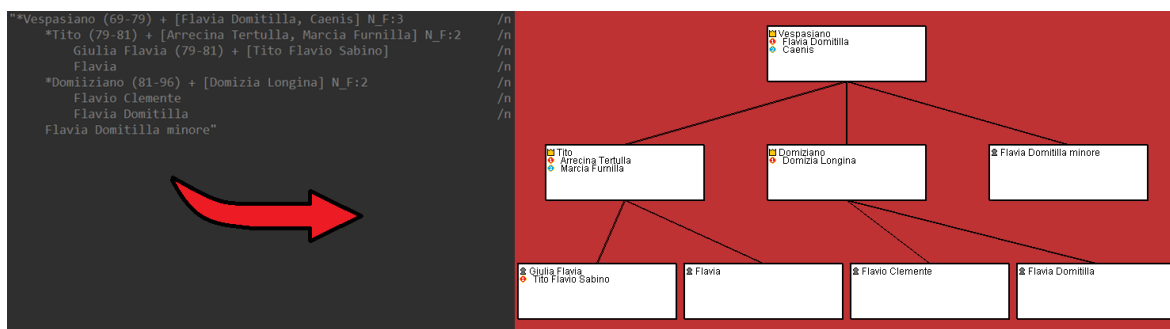
```
public static void processString(Imperatore root, String tabs) {
    //Blank space and start a new line (\n)
    codifica += "\n" + tabs;
    //*** character means there is an emperor in the current line
    codifica += " *";
    //Control if the root hasn't got any children ...
    if(root.getFigli().isEmpty())
        //... if that's the case only its name and spouses needs to be added to the line
        //If the root hasn't got any spouses ...
        if(root.getConiugi().isEmpty())
            //... only its name and years of reign get added
            codifica += (root.getNome() + " (" + root.getMandato() + ")");
        else
            //... otherwise its name, years of reign and spouses get added
            codifica += (root.getNome() + " (" + root.getMandato() + ") " + " + " + root.getConiugi());
        //otherwise...
    else {
        //... if that's the case only its name and spouses needs to be added to the line
        //If the root hasn't got any spouses ...
        if(root.getConiugi().isEmpty())
            //... only its name and years of reign get added
            codifica += (root.getNome() + " (" + root.getMandato() + ")");
        else
            //... otherwise its name, years of reign and spouses get added
            codifica += (root.getNome() + " (" + root.getMandato() + ") " + " + " + root.getConiugi());
        //Since root has got children they will be put in an ArrayList
        ArrayList figli = root.getFigli();
        //If figli ArrayList isn't empty ...
        if(!figli.isEmpty())
            //... the number of children will be added next
            codifica += (" N_F :"+figli.size());
        //Tabs number gets increased by 1
        tabs = tabs + "\t";
        //For each children in the ArrayList ...
        for(int i = 0; i < figli.size(); i++) {
            //... the current element will be put in a temporary Persona type variable
            Person fi = (Person) figli.get(i);
            //If the child is an emperor ...
            if(fi.getCheckImp()) {
                // ... it will be casted as an Imperatore type object, its mandato attribute will be conserved
                Imperatore imp = (Imperatore) figli.get(i);
                //Then the method using an Imperatore as root will be recursively called
                processString(imp, tabs);
            }
            else {
                // ... it will be casted as a Persona type object
                Person per = (Person) figli.get(i);
                //Then the method using a Imperatore as root will be recursively called
                processString(per, tabs);
            }
        }
    }
}
```

Adesso che la codifica è pronta, viene passata alla classe TreelImage che la suddividerà riga per riga ed estrarrà le informazioni utili da ciascuna ed utilizzerà quest'ultime per la creazione di oggetti di classe Node, che verranno posti in un'apposita ArrayList.

```
public static void createNodes(ArrayList<String> lines) {
    //The nodes list gets cleared for every search call to avoid overlapping trees
    nodes_list.clear();
    //For each element in tree_tiles_map ...
    for(int i = 0; i < tree_tiles_map.size(); i++) {
        int y = tree_tiles_map.get(i).get(0);
        int x = tree_tiles_map.get(i).get(1);
        // ... a new Node object gets created and put in the nodes_list
        nodes_list.add(new Node(lines.get(i), y, x, members_per_height, img_width));
    }
}
```

Una volta realizzati i Nodes rimarrà soltanto da disegnare l'immagine, tale compito è affidato all'oggetto Actor Graphics2D g2d, in seguito l'immagine verrà salvata nell'apposita cartella "History", concepita come un meccanismo per simulare una cronologia delle ricerche effettuate.

Di seguito è riportato un esempio della codifica realizzata inizialmente e del risultato finale:

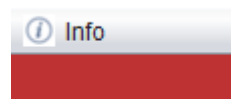


Manuale GUI

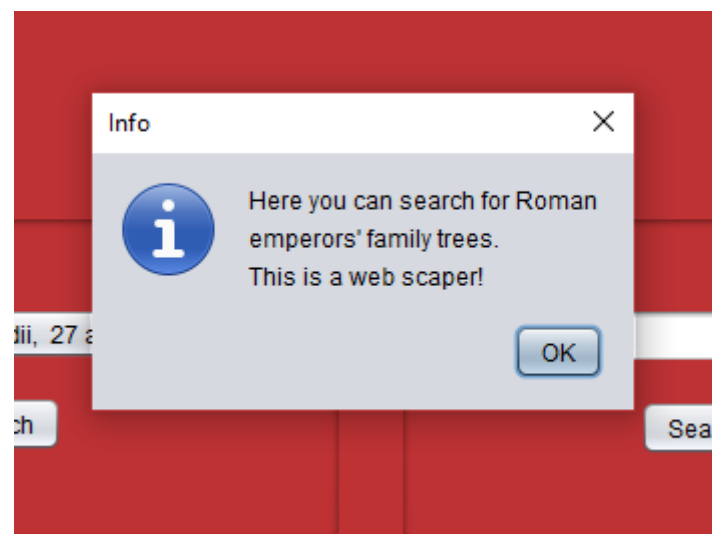
L'esecuzione del progetto parte dalla classe "Home" e sullo schermo apparirà questa facciata:



L'utente può cliccare sul pulsante "info" in alto a sinistra

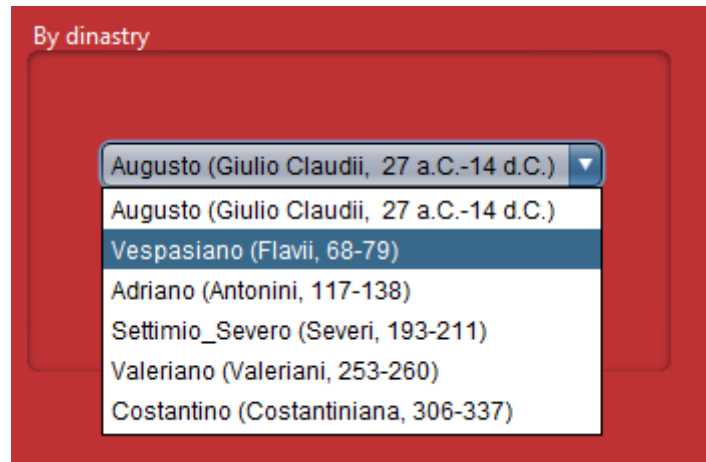


dove verrà brevemente spiegato lo scopo dell'applicazione:

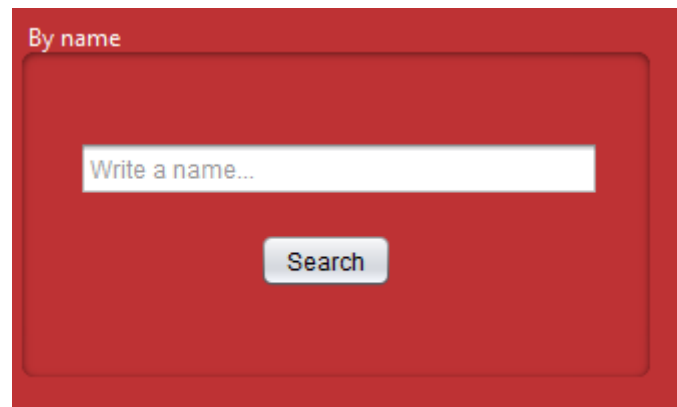


È possibile effettuare la ricerca in due modi differenti:

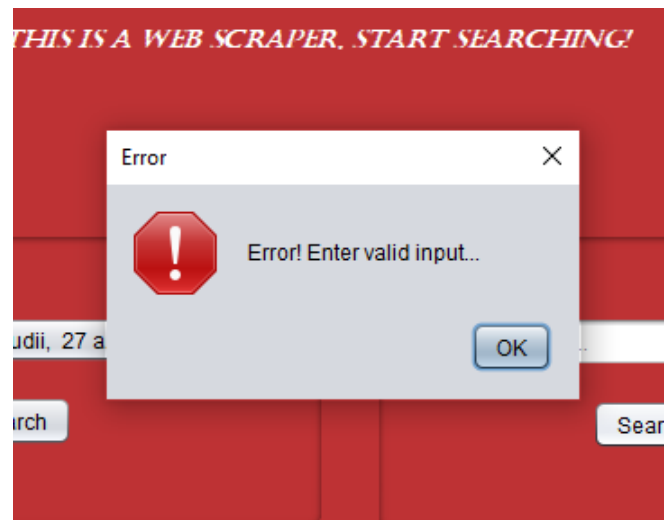
- nel primo caso basta selezionare una delle dinastie nel menu a tendina e premere il pulsante "Search" subito sotto.



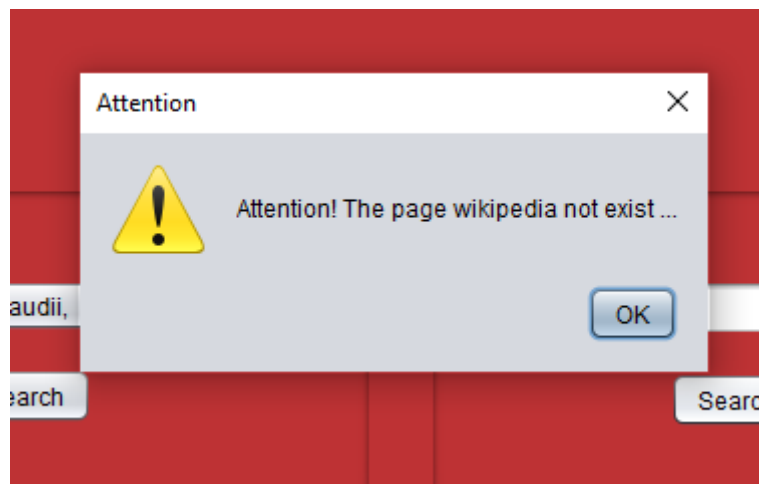
- oppure è possibile digitare il nome direttamente da tastiera e avviare la ricerca con il pulsante "Search" subito sotto.



In particolare, nella ricerca per nome, possono essere generate due situazioni di errore: una se l'utente non digita alcun nome;



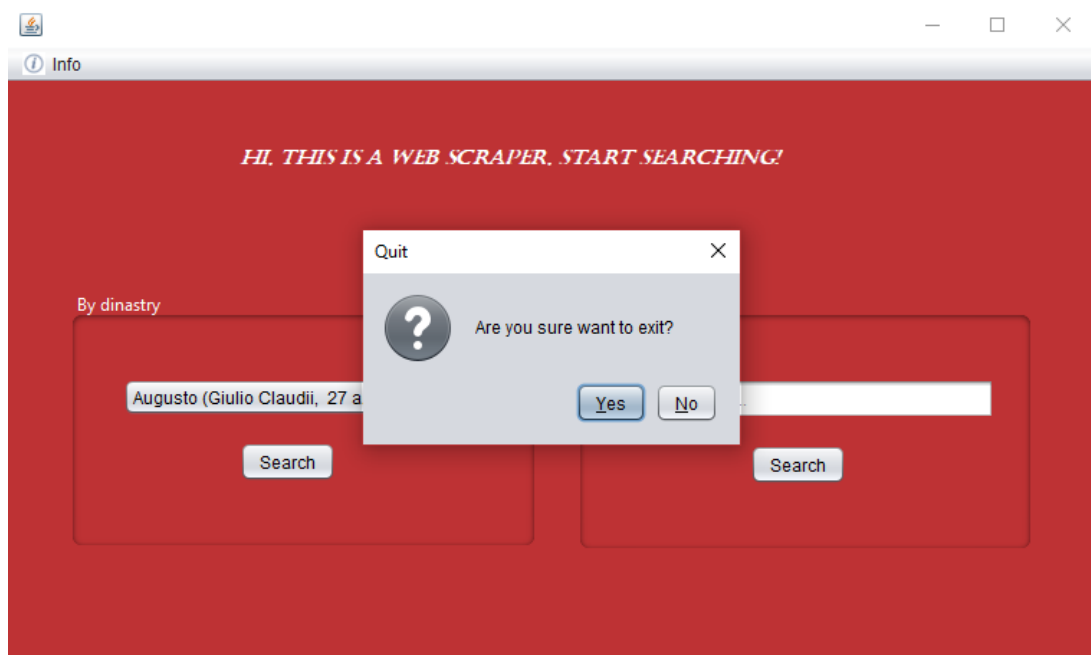
un secondo errore se l'utente digita un nome sconosciuto per Wikipedia.



Un altro controllo riguarda la chiusura del programma. Infatti, se si clicca accidentalmente sulla crocetta in alto a destra

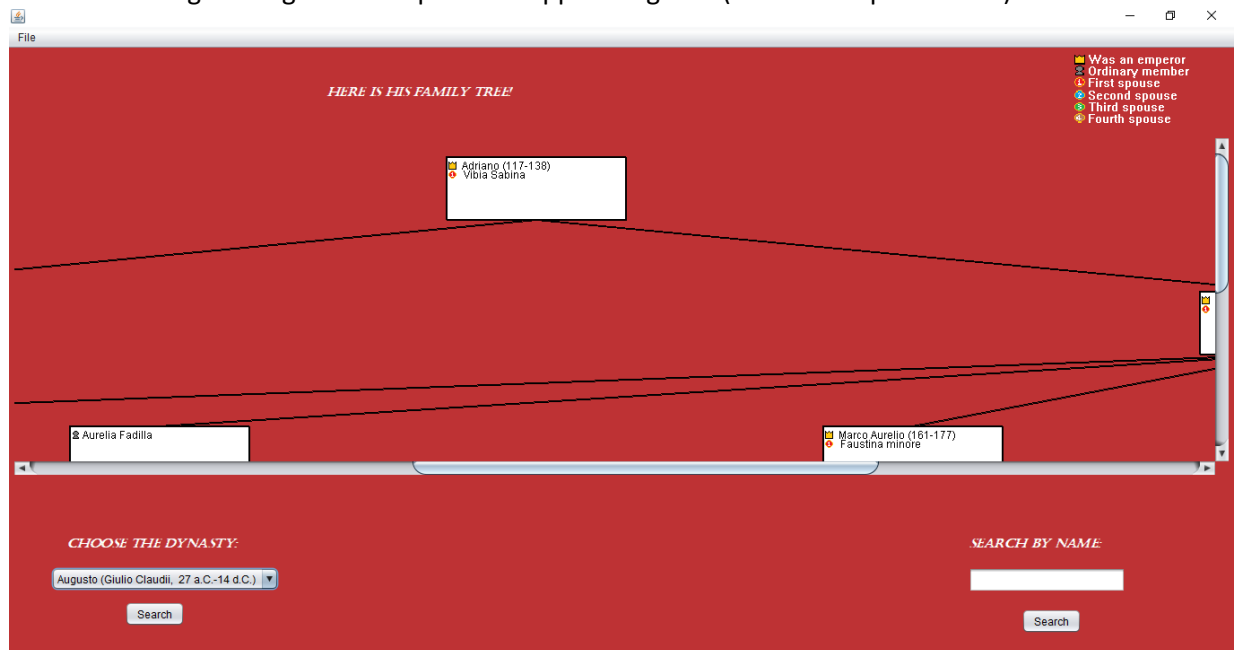


non c'è motivo di preoccuparsi perché il programma lancia un pannello di conferma di chiusura.



Cliccando su "Yes" il programma termina altrimenti con "No" è possibile proseguire con la ricerca.

Se la ricerca va a buon fine, si passa alla seconda schermata dove è possibile visualizzare l'albero genealogico dell'imperatore appena digitato (Adriano in questo caso):

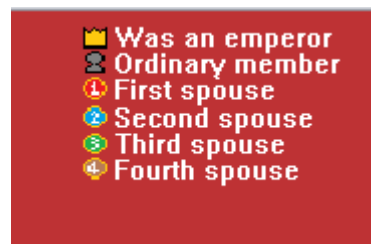


Se l'immagine è troppo grande può essere visualizzata senza alcun problema grazie al pannello di scorrimento su cui si poggia l'immagine:

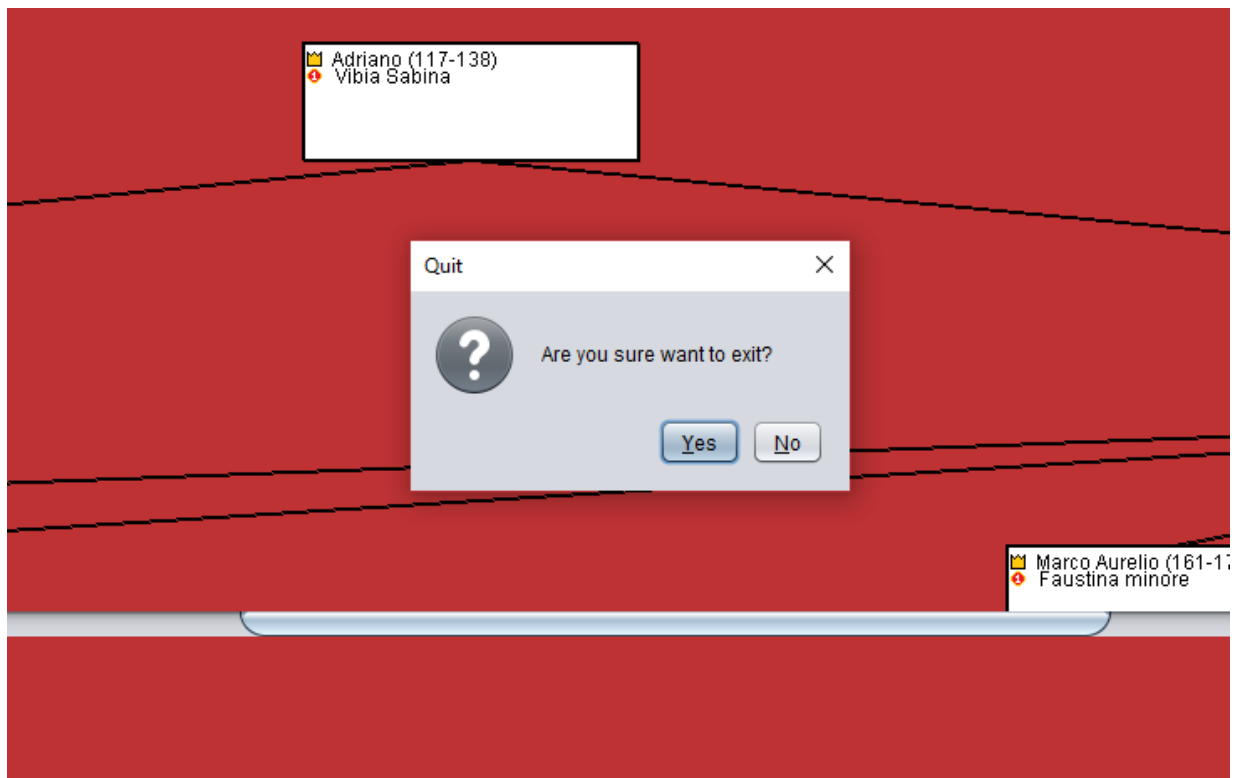


Nell'area in cui è possibile inserire il nome dell'imperatore, non appena ci si clicca sopra, il testo in grigio scompare e si può inserire il nome. (Questo vale anche per il frame precedente).

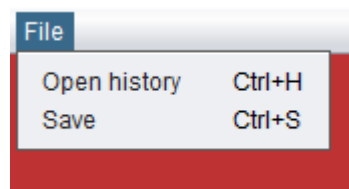
L'immagine della legenda è inserita in alto a destra.



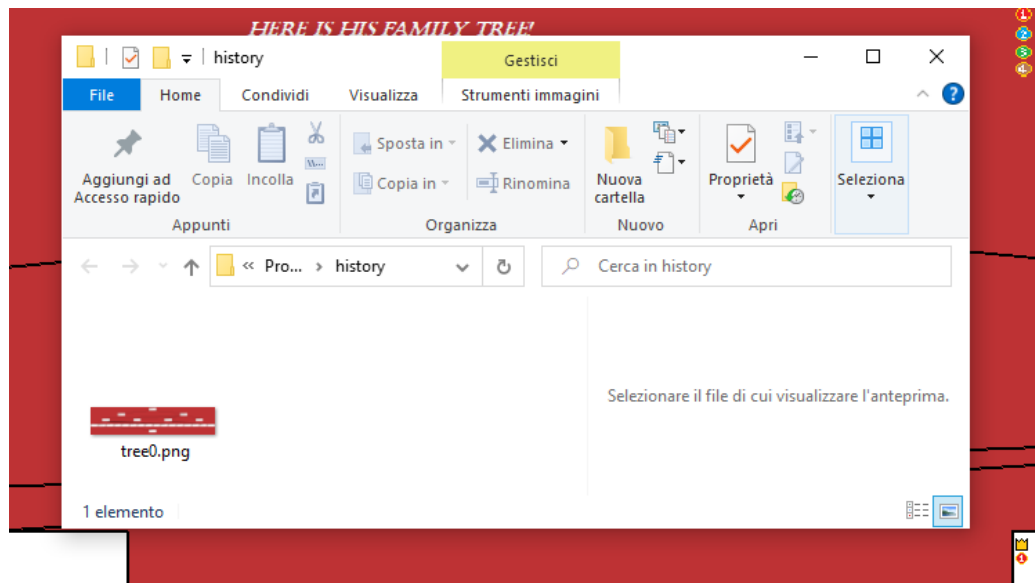
Anche in questa finestra è stato inserito il controllo sulla chiusura del programma.



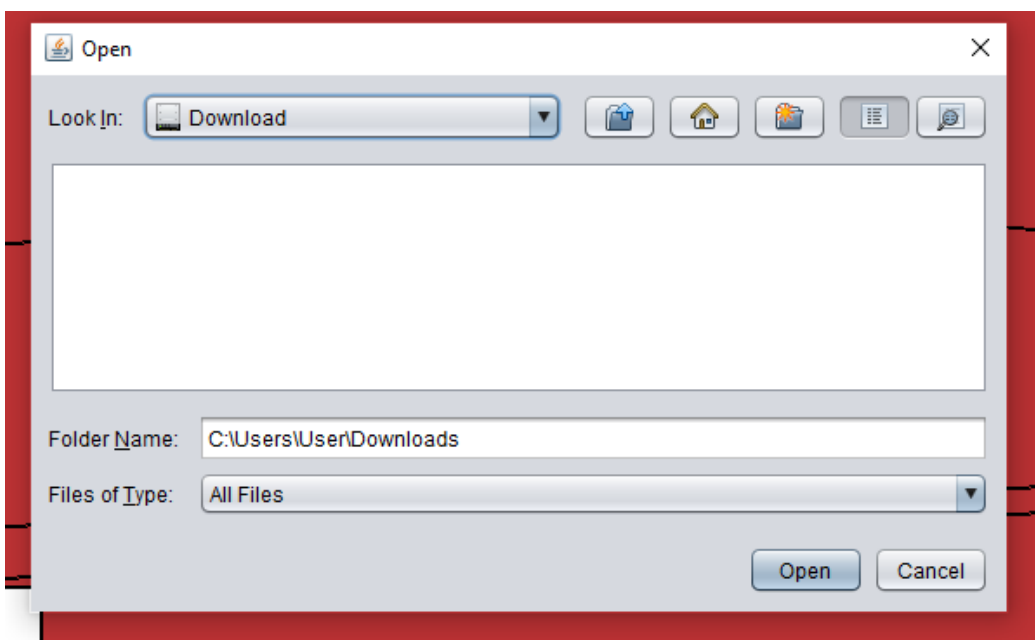
Con il menù in alto a sinistra è possibile:



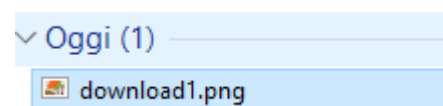
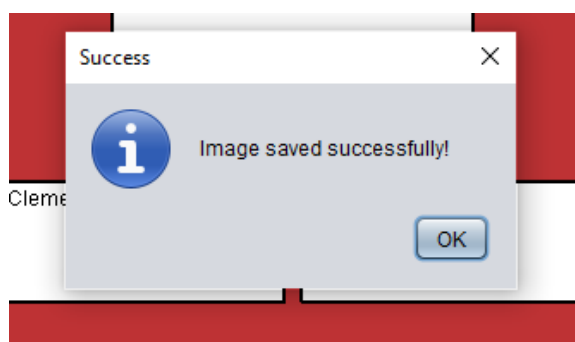
aprire la cronologia delle ricerche effettuate fino a quel momento cliccando sul pulsante "Open history" (o Ctrl + H)



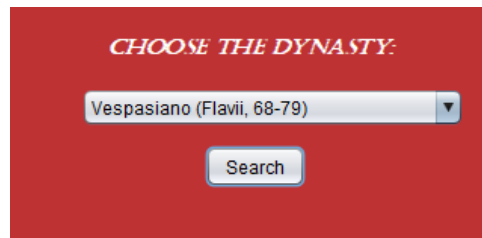
oppure se si vuole salvare l'albero genealogico dell'imperatore appena cercato, basta cliccare sul pulsante "Salva" nel menu (o Ctrl + S) e selezionare la cartella in cui salvarlo.



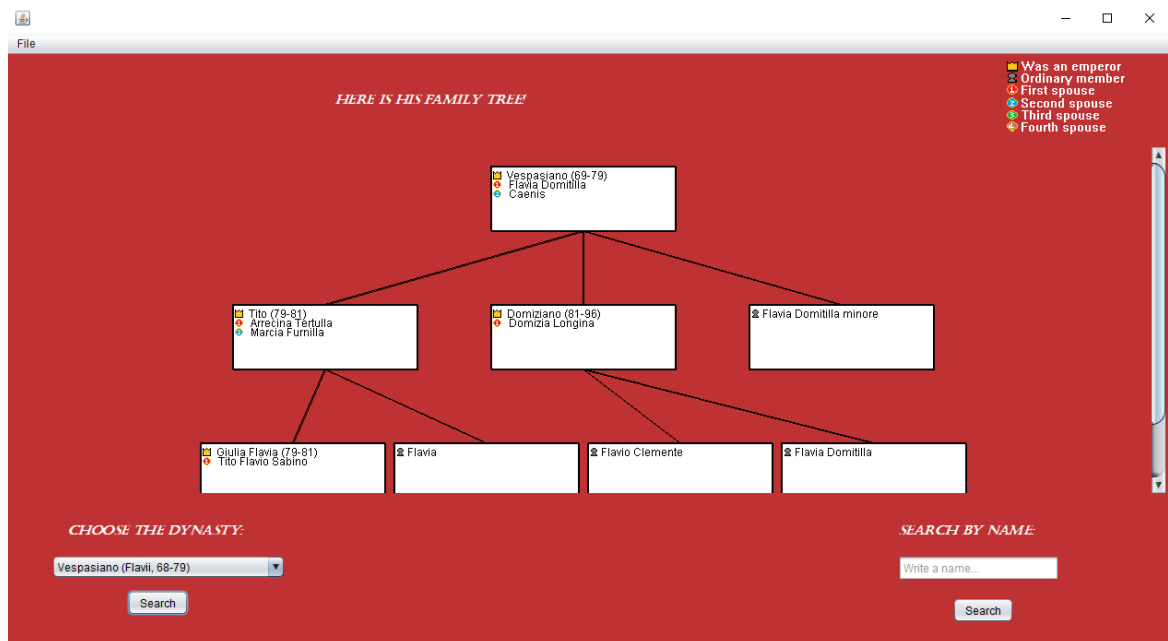
Al termine dell'operazione, il programma invia un messaggio di operazione riuscita



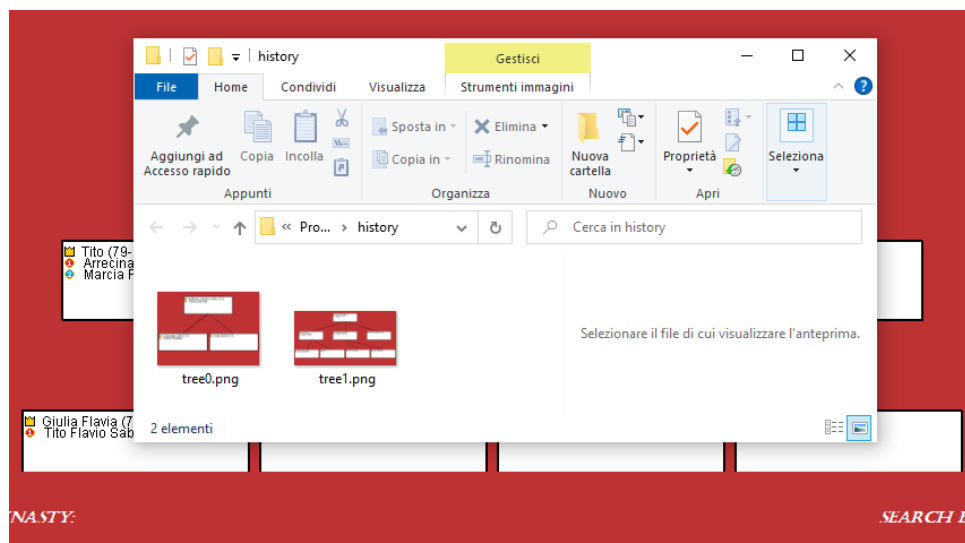
A questo punto è possibile proseguire con la ricerca utilizzando uno dei due metodi (menu a tendina nell'esempio):



Non si apriranno nuovi frame, ma verrà aggiornata solo l'immagine sullo schermo.



Ovviamente la cronologia delle ricerche si aggiornerà automaticamente.



Referenti sviluppo

La suddivisione del lavoro del nostro gruppo è stata organizzata nel seguente modo:

- Michelangelo Crea: si è occupato della parte back-end, creazione delle classi Person e Imperatore, ricerca con selenium delle pagine web ed estrapolazione dei dati dalle pagine wikipedia. Creazione di tutte le caratteristiche della dinastia. Amministrazione di GitHub.
- Daniel Di Nella: si è occupato della parte back-end, creazione delle classi Person e Imperatore, ricerca con selenium delle pagine web ed estrapolazione dei dati dalle pagine wikipedia. Creazione di tutte le caratteristiche della dinastia. Amministrazione di GitHub.
- Luca Ciafardoni: si è occupato della parte front-end per quanto riguarda la processazione ed elaborazione delle informazioni ottenute dall'operazione di Web Scraping e la realizzazione dell'immagine dell'albero genealogico completo.
- Enrico Chiarello: si è occupato della parte front-end e nello specifico, della creazione di un'applicazione con controlli, funzionalità e aggiornamenti in grado di integrarsi al meglio con la tecnica dello Web Scraping.