SAPIENZA
UNIVERSITÀ DI ROMA

DIPARTIMENTO DI INFORMATICA

Web Scraper Wikipedia Imperatori Romani

CORSO DI METODOLOGIE DI PROGRAMMAZIONE

Professori:
Walter Quattrociocchi
Gabriele Etta

Studenti:
Diego Guzman
Victor Lopata
Maxim Cuzuc
Kevin Huang

Anno accademico 2021/2022

Contents

Introduzione

1.1 Ripartizione dei lavori

Il Web Scraper Wikipedia sugli Imperatori Romani è stato realizzato dai quattro studenti sopracitati, il lavoro per il progetto è stato internamente suddiviso, per cui: Maxim Cuzuc e Kevin Huang si sono predisposti alla progettazione dello scraper e del grafo, mentre la cura del design e la creazione dell'interfaccia grafica sono stati possibili grazie al contributo di Victor Lopata e di Diego Guzman Aguirre.

1.2 Strumenti ausiliari

Il progetto è stato portato a termine tramite Java, grazie all'utilizzo:

- L'ambiente di sviluppo IntelliJ IDEA integrato con il Selenium WebDriver, strumento con il quale ci è stato possibile controllare il nostro headless web browser HtmlUnit, headless in quanto senza interfaccia grafica;
- Il servizio hosting GitHub, ove venivano caricati di volta in volta le novità, modifiche del nostro gruppo di lavoro.

1.3 Problematiche incontrate

Ad un primo impatto il tema del WebScraper è risultato molto astratto da comprendere, ma a seguirsi di giornate di studi, ricerche per intuire ciò che era necessario progettare, quali erano i limiti che Selenium ci poteva offrire, siamo riusciti a perfezionare lo Scraper.
Ci siamo posti molte domande, tra cui: quali erano le informazioni che ci servivano? Come ottenerle? visto che Wikipedia non ci offriva una locazione fissa; quale sarebbe stato il design che avremmo preferito? E soprattutto come implementarlo?...
Attraverso lunghe discussioni di gruppo e indagini online siamo riusciti ogni volta a raggiungere una soluzione e lo Scraper che abbiamo realizzato ne è la dimostrazione.

NOTA: Ogniqualvolta che è presente un * è possibile cliccare il label antecedente per maggiori informazioni.

WebScraper

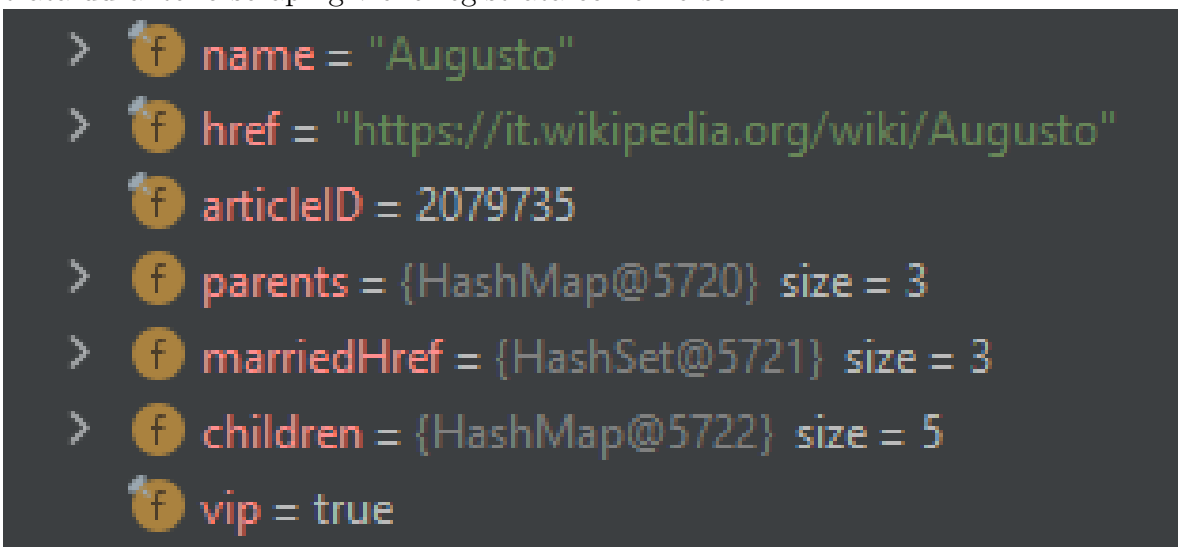
Il WebScraper è costituita da tre classi principali:

- **Person**, ovvero la classe con cui viene registrato ogni individuo, da cui possiamo inserire/prelevare tutte le informazioni che ci è necessario alla costituzione del grafo;
- **Scraper**, dove vengono impostate le informazioni necessarie per iniziare veramente lo scraping;
- **Utils**, che in realtà è la classe che compie effettivamente il "lavoro sporco", vedremo il tutto a tempo debito.

Cominciamo dall'analisi della classe Person...

2.1 Person

Una volta selezionata la profondità di ricerca e dopo aver cliccato l'apposito pulsante presente sull'interfaccia grafica "*LOAD DATA*" (Cap. 4.3.1 - Descrizione Finestra MainWindow*) vengono avviate una serie di operazioni per cui ogni persona incontrata durante lo scraping viene registrata come Person.



(Person di Augusto - depth:1)

L'immagine precedente mostra i parametri di un entità facente parte della classe Person, in questo caso possiamo osservare:

- **Parametri identificativi**, quali il nome Augusto con la sua pagina Wikipedia* associata e il suo articleID unico per ognuna persona, trovato nella sezione "wgArticleId" presente in uno script posseduto da tutte le entità;
- **Due HashSet**, di cui una contiene gli href dei parenti di Augusto, mentre l'altra ospita gli href delle sue mogli;
- **Un HashMap**, che dispone come chiavi gli href dei suoi figli e come valori un parametro booleano per identificare se sia adottato o meno;
- **Vip**, un parametro booleano per suddividere le entità che possiedono una tabella sinottica da quelle che non ne sono provviste.

2.2 Scraper

Per iniziare portiamo il nostro HtmlUnit driver nella pagina che ci serve, ovvero quello contenente tutti gli **Imperatori Romani*** e da questa pagina salveremo *tables*, le tabelle contenenti tutti gli Imperatori suddivise in dinastie con l'ausilio dell'XPath cercando i tables appartenente alla classe wikitable.
Per ogni tabella in tables creiamo un nuovo HashSet e per ogni Imperatore nella tabella navigheremo con il nostro driver verso il suo href per poi richiamare nella sua pagina il metodo ricorsivo *fetchData* che osserveremo a breve.
Ogniqualvolta che viene analizzata un Person quest'ultimo verrà mostrato sul progressBar per poi una volta terminata l'operazione avvertire l'utente che lo scraping è terminato.

Dinastia giulio-claudia (27 a.C.-68 d.C.) [modifica | modifica wikitesto]

Immagine	Nome	Nascita	Regno		Morte	Note
			Inizio	Fine		
	**Augusto** Gaio Ottavio Turino Gaio Giulio Cesare Ottaviano Imperatore Cesare Augusto	23 settembre 63 a.C., Roma, Italia	16 gennaio 27 a.C.	19 agosto 14	vecchiaia[3]	Figlio adottivo di Cesare, governò come triumviro dal 42 a.C., assieme a Marco Antonio e Marco Emilio Lepido.
	**Tiberio** Tiberio Claudio Nerone Tiberio Giulio Cesare	16 novembre 42 a.C., Roma, Italia	19 agosto 14	16 marzo 37	malattia non specifica[4]	Figlio adottivo di Augusto
	**Caligola** Gaio Giulio Cesare Germanico	31 agosto 12, Anzio, Italia	18 marzo 37	24 gennaio 41	assassinato dai pretoriani	Nipote paterno di Druso maggiore, fratello di Tiberio; bisnipote materno di Augusto
	**Claudio** Tiberio Claudio Druso Nerone Germanico Tiberio Claudio Cesare Augusto Germanico	1° agosto 10 a.C., Lione, Gallia	24 gennaio 41	13 ottobre 54	avvelenato probabilmente da Agrippina Minore	Figlio di Druso maggiore; Lucio Arrunzio Camillo Scriboniano usurpatore in Dalmazia
	**Nerone** Lucio Domizio Enobarbo Tiberio Claudio Druso Nerone Cesare Germanico Domiziano	15 dicembre 37, Anzio, Italia	13 ottobre 54	9 giugno 68	suicidio assistito	Nipote materno di Caligola. Salì sul trono grazie agli intrighi di sua madre Agrippina minore, che fece poi uccidere; all'inizio governò sotto la tutela di Agrippina stessa e del tutore Seneca. Nel 68 fu dichiarato dal Senato "nemico pubblico"

(*Table della dinastia Giulio Claudia*)

2.3 Utils

La classe Utils compie effettivamente il lavoro dello scraping, è Utils che prende un entità lo registra come Person e va a ricercare i suoi relativi, vediamo come funziona nel dettaglio.

fetchData, metodo ricorsivo che prende in input il driver; due HashSet rispettivamente *dynasty* per inserire i Person solo della dinastia selezionata mentre in *entityList* vengono inseriti tutti senza differenziare nulla; il *progressBar* per configurarne l'avanzamento a pari passo con lo scraping; la profondità di ricerca selezionata (Cap. 4.3.1 - Slider*) e ignoreDepth se la profondità selezionata è pari a 10.
Il driver, ovvero il navigatore che nel nostro caso si tratta dell'*HtmlUnit*; ogni volta che viene richiamato il metodo il driver arriva all'href corrispondente dell'entità interessata , da cui prima di tutto andremo a cercare il suo univoco *articleID* situato nell'XPath */html/head/script[1]*, con cui invocheremo il metodo *getFrom* (Cap. 2.3.1 - getFrom*) e in caso non tornasse null significherebbe che questa entità è già presente in entityList e quindi possiamo tornare nella istanza ricorsiva precedente, altrimenti dobbiamo creare il Person ed inserirlo nell'entityList.
Verifichiamo se la persona è un Vip allora ricercheremo nella sua tabella sinottica il suo nome e lo mostriamo subito sul progressBar, successivamente cerchiamo anche i parenti, mogli e figli attraverso i metodi *getKins e getRelatives*(Cap. 2.3.1 - getKins*) ora che abbiamo tutto il necessario possiamo creare il Person e in caso non sia ripetuto andremo a href per href tra tutti i parenti, mogli e figli a richiamare su di loro fetchData con profondità di ricerca meno uno. Se non fosse Vip andremo a prendere il nome dall'inizio della pagina e metteremo il resto a null.

Coniuge	Clodia Pulcra[26] (fino al 40 a.C.) Scribonia[26] (40–38 a.C.) Livia Drusilla[26] (38 a.C.-14)
Figli	Giulia maggiore[29] Adottivi: Lucio Cesare[30] Gaio Cesare[30] Marco Vipsanio Agrippa Postumo (poi ripudiato e mandato in esilio)[31] Tiberio[31]
Dinastia	Giulio-claudia
Padre	Gaio Ottavio[27] Gaio Giulio Cesare (adottivo)
Madre	Azia maggiore[28]

Augusto

Imperatore romano

2.3.1 Metodi Ausiliari

- **pageExist**. Wikipedia in alcuni casi in cui la pagina non esiste mostra un link cliccabile rosso che non fornisce nulla di quel che ci serve per cui utilizziamo pageExist per verificare se la pagina esiste, in caso negativo saltare il link.

- **getFrom**. Ci permette di ricercare in un Set di Person una entità attraverso il suo articleID, se venisse trovato allora viene tornato il Person, altrimenti null.

- **getHeaderContent**. Prendendo in input un header e la tabella su cui si vuole ricercato viene tornato tutto il contenuto presente nella cella a destra dell'header, verrà utilizzato per i due metodi seguenti.

- **getKins e getRelatives**. Dato la tabella in cui vogliamo cercare un tipo di parentela: *padre*, *madre* o *figli*, mogli per getRelatives. Utilizzando getHeaderContent otteniamo la cella e dopo una serie di controlli per aggirare le varie sfaccettature di Wikipedia aggiungiamo i parenti e in caso di figlio adottivo impostare la variabile booleano a true.

Figli	Domizia Faustina Aurelia Tito Aurelio Antonino Tito Elio Aurelio Lucilla Annia Aurelia Galeria Faustina Tito Elio Antonino

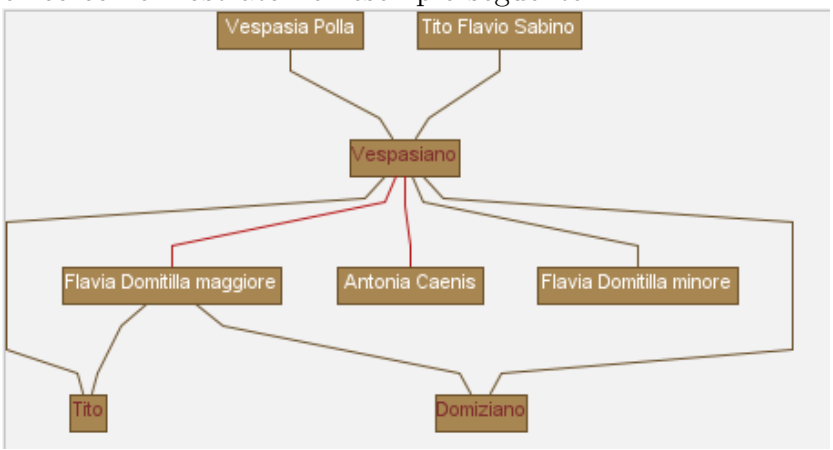
(*Sezione figli della tabella sinottica di Marco Aurelio*)

Grafo

Grazie alla classe SimpleGraph abbiamo realizzato un grafo che non è né direzionato né pesato, le cui relazioni sono definite da RelationshipEdge che estende DefaultEdge, ma a differenza di quest'ultimo RelationshipEdge richiede un parametro "label", ovvero la relazione che collega due vertici.
Aggiungeremo vertici e relazioni grazie alla classe DynastyTree.

3.1 DynastyTree

DynastyTree è il fulcro della creazione del grafo, utilizzando il metodo *addAllVertices* aggiungiamo ogni persona presente in entityList come vertici nel grafo. Per ogni persona presente tra i vertici andiamo a creare due liste ricercando una volta i coniugi e un'altra i figli attraverso un filtro nell'entityList; colleghiamo i vertici con le loro relazioni, controllando in caso se l'entità sia o meno adottato con l'apposito parametro booleano.
L'ausilio di JGraphXAdapter ci permette di visualizzare il grafo che abbiamo appena realizzato e attraverso l'algoritmo dell'HierarchicalLayout otteniamo un grafo gerarchico come mostrato nell'esempio seguente:

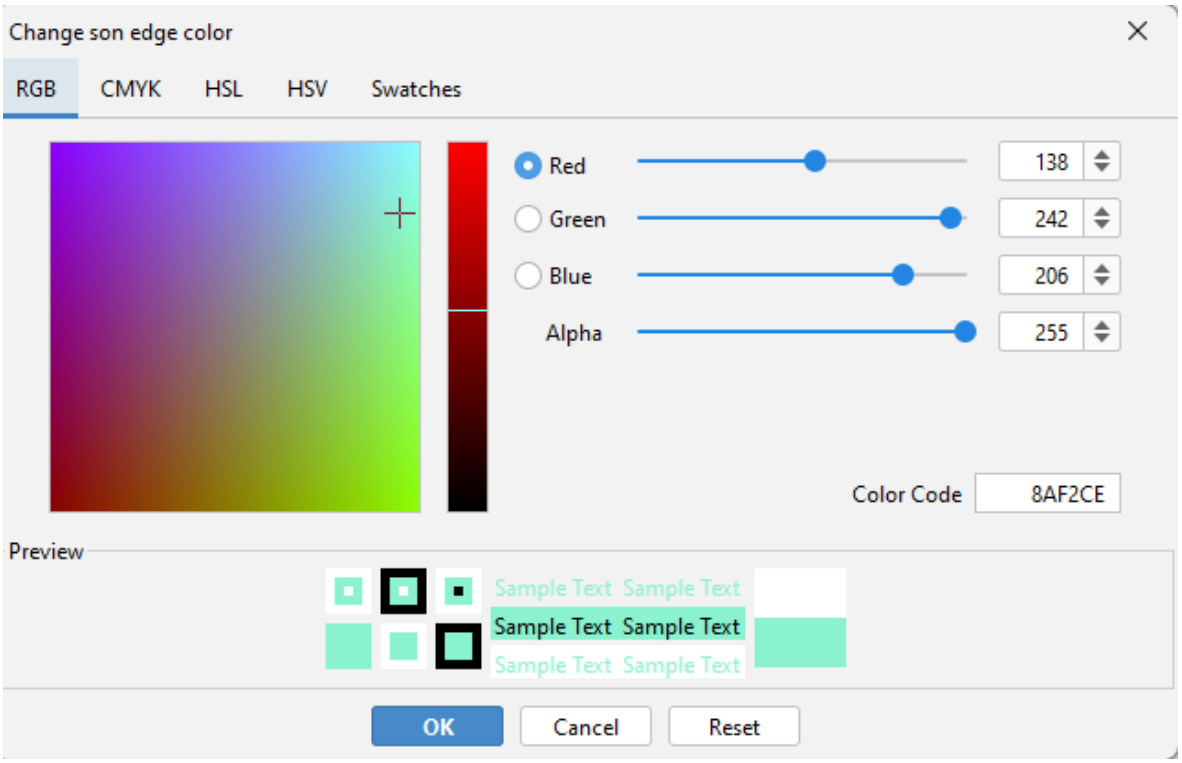


Tramite l'interfaccia mxIEventListener, ogniqualvolta che viene rilevato un evento nel grafo, viene eseguito il metodo *"invoke"*, il quale ne controlla la tipologia e apre la pagina Wikipedia correlata in caso corrisponda a mxEvent.CHANGE (Es. selezione di una cella)

3.2 Personalizzazione delle celle

Convertiamo il graphAdapter, precedentemente creato, in un HashMap di relazioni e celle, e successivamente inizializziamo tre ArrayList: marriedList, adptedList, kinList; nei quali verranno inseriti la giusta relazione scorrendo una alla volta l'HashMap.

Ora è il momento in cui entra in gioco il metodo *setCellsStyle* che dato una determinata relazione e un colore preimpostato in esadecimale è possibile cambiare il colore di una relazione. Invece se l'utente volesse modificare a proprio piacere lo stile di una relazione può utilizzare gli appositi comandi presenti nel TreeWindow.
(Cap. 4.4.3 - Cambiare Colore alle Relazioni*)



(Personalizzazione della relazione con i figli)

Interfaccia grafica

4.1 Introduzione

La realizzazione dell'interfaccia grafica è stata resa possibile tramite l'utilizzo di due librerie principali: AWT e Swing.

AWT(Abstract Window Toolkit) è una libreria per la creazione e raffigurazione di componenti grafici che dipendono dal sistema operativo. Ciò significa che lo stile dei componenti grafici sarà impostato dal sistema operativo.

Swing è la libreria grafica di seconda generazione di Java. Essa non si pone come sostituto di AWT, infatti possiamo definirla come un'estensione di AWT. A differenza della libreria AWT, i componenti grafici di Swing vengono chiamati *lightweight* (leggeri) perché sono scritti interamente in Java, mentre quelli di AWT vengono detti *heavyweight* (pesanti) perché utilizzano risorse di sistema scritte con codice nativo.

Durante la realizzazione della GUI sono state maggiormente preferite l'utilizzo delle classi Swing poiché sono ritenute più evolute di quelle AWT e consentono di creare interfacce grafiche senza nessun limite di fantasia.

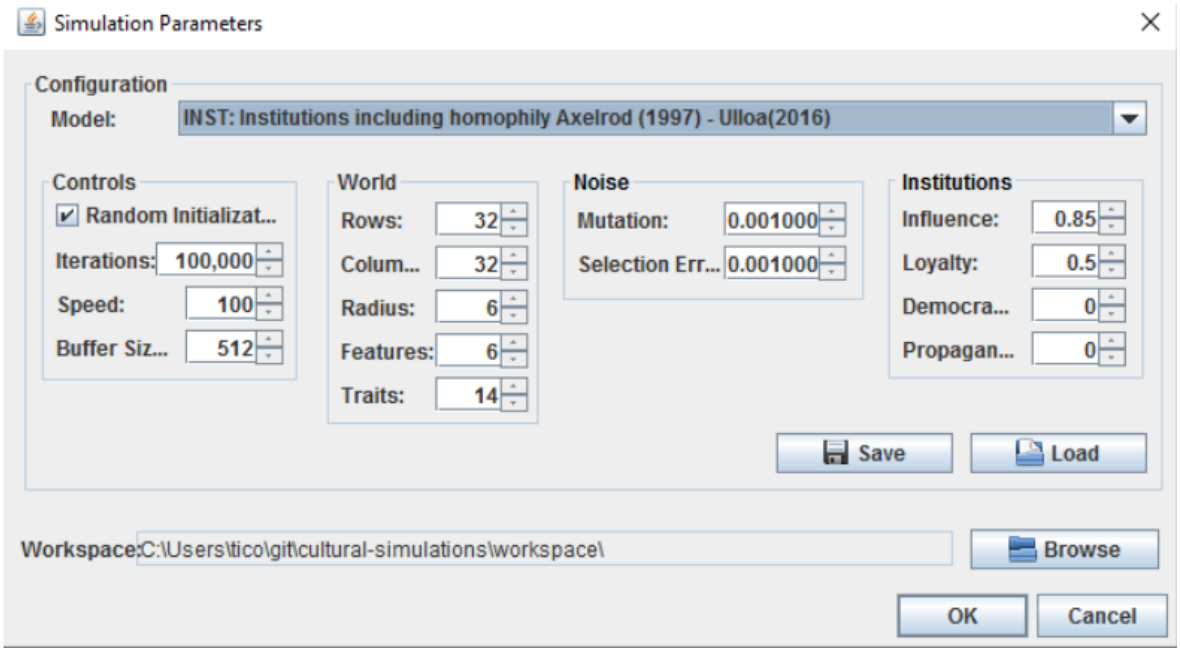
Alla fine della lettura di questo capitolo il lettore avrà le idee chiare sulle scelte compiute dagli sviluppatori riguardo tutte quelle dinamiche che caratterizzano la realizzazione di un'interfaccia grafica.

Le principali tematiche affrontate sono:

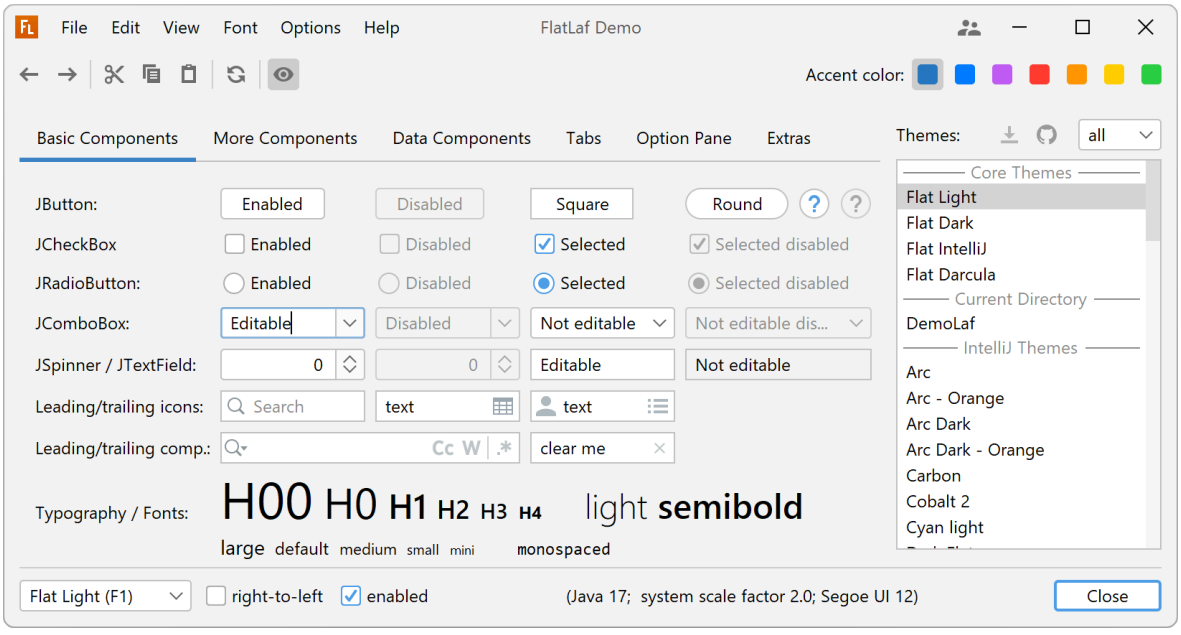
- Il setting del *Look and Feel* FlatLaf;
- Descrizione della classe Window e tutte le sue dinamiche;
- Descrizione della classe TreeWindow e tutte le sue dinamiche;
- Descrizione della classe SaveWindow e tutte le sue dinamiche;
- Descrizione di altre classi.

4.2 Look and Feel

Come introdotto, l'architettura Swing è costruita in modo tale da poter cambiare il *Look and Feel* del programma, cioè tutte le apparenze e i comportamenti dei vari componenti della GUI: infatti *Look* si riferisce alle apparenze e all'aspetto esteriore, mentre *feel* si riferisce al comportamento dei widget. Fondamentalmente determina come l'utente percepisce l'applicazione e interagisce con essa. Il vantaggio che comporta a cambiare il *Look and Feel* della GUI è sicuramente avere la possibilità di modernizzare la propria interfaccia rendendola meno obsoleta e più evoluta.



GUI con Look and Feel di default.



GUI con Look and Feel utilizzato nel progetto.

La differenza tra le due immagini è netta. Nella prima immagine viene mostrata un'interfaccia grafica avente il Look and Feel di base, nella seconda immagine viene raffigurata un'interfaccia con Look and Feel usato nel progetto che trasmette all'utente un senso di modernità e minimalismo.

E' stato utilizzato FlatLaf come *Look and Feel* seguendo le istruzioni prese direttamente dal sito `https://www.formdev.com/flatlaf/`*, quindi:

- Aggiungere alle dipendenze (librerie esterne) le coordinate date dal sito;
- Aggiungere un pezzo di codice al metodo principale;

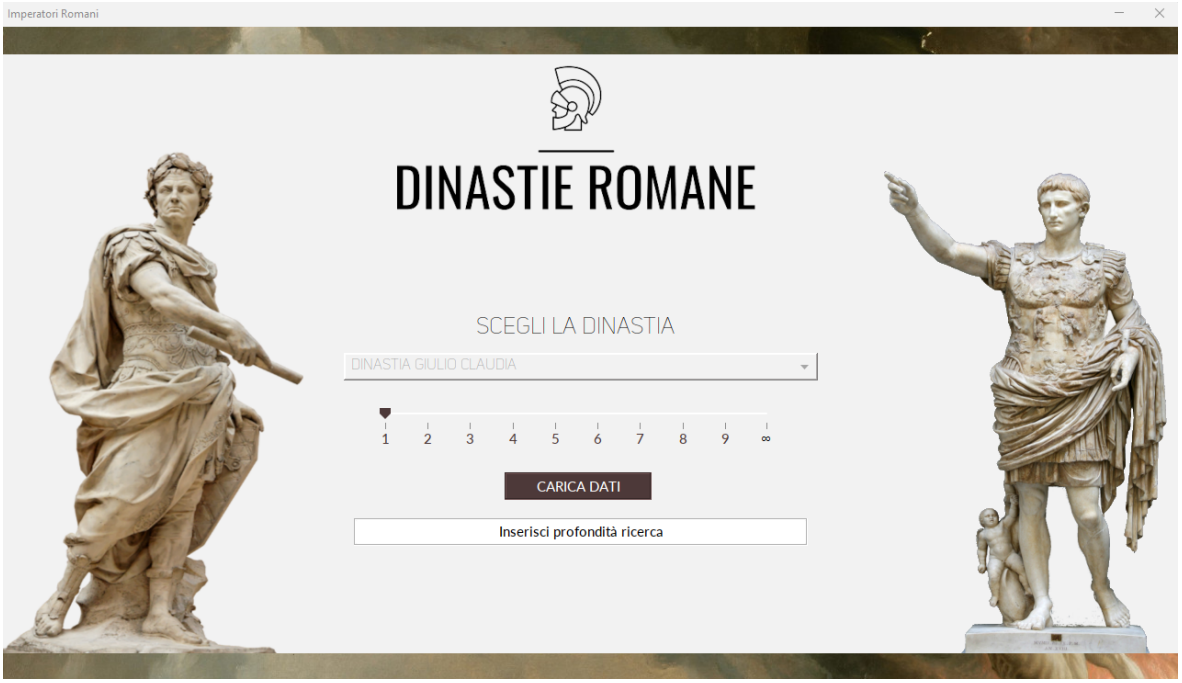
Con questi due passaggi è stato possibile modificare il *Look and Feel* di base con quello di FlatLaf.

Codice sorgente di FlatLaf: `https://github.com/JFormDesigner/FlatLaf`*

4.3 MainWindow

In questo sottocapitolo esamineremo la struttura della classe Window e analizzeremo tutte le dinamiche che hanno portato gli sviluppatori a preferire una scelta rispetto ad altre.
Inizialmente verranno spiegate tutte le funzionalità che la finestra offre, in seguito verrà raffigurato il Layout Composition che la compone.

4.3.1 Descrizione Finestra MainWindow

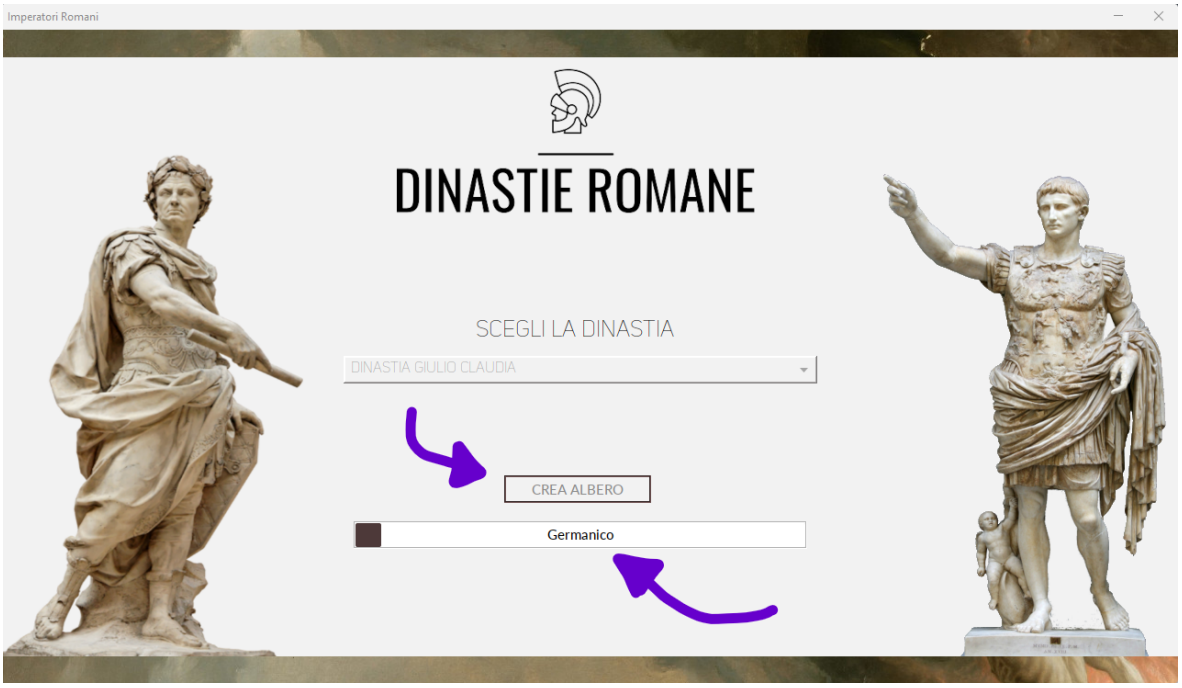


Finestra MainWindow.

La classe MainWindow implementa la finestra principale che, una volta eseguito il programma, ci compare sullo schermo. Il suo compito è far scegliere all'utente quale dinastia di Imperatori Romani voler visualizzare tramite un menù a tendina. Tramite uno "Slider", invece, è possibile scegliere quanto lo Scraper debba andare in profondità nella ricerca. Questa implementazione dà la possibilità all'utente di ridurre il tempo di ricerca nel caso si dovesse scegliere una profondità bassa.

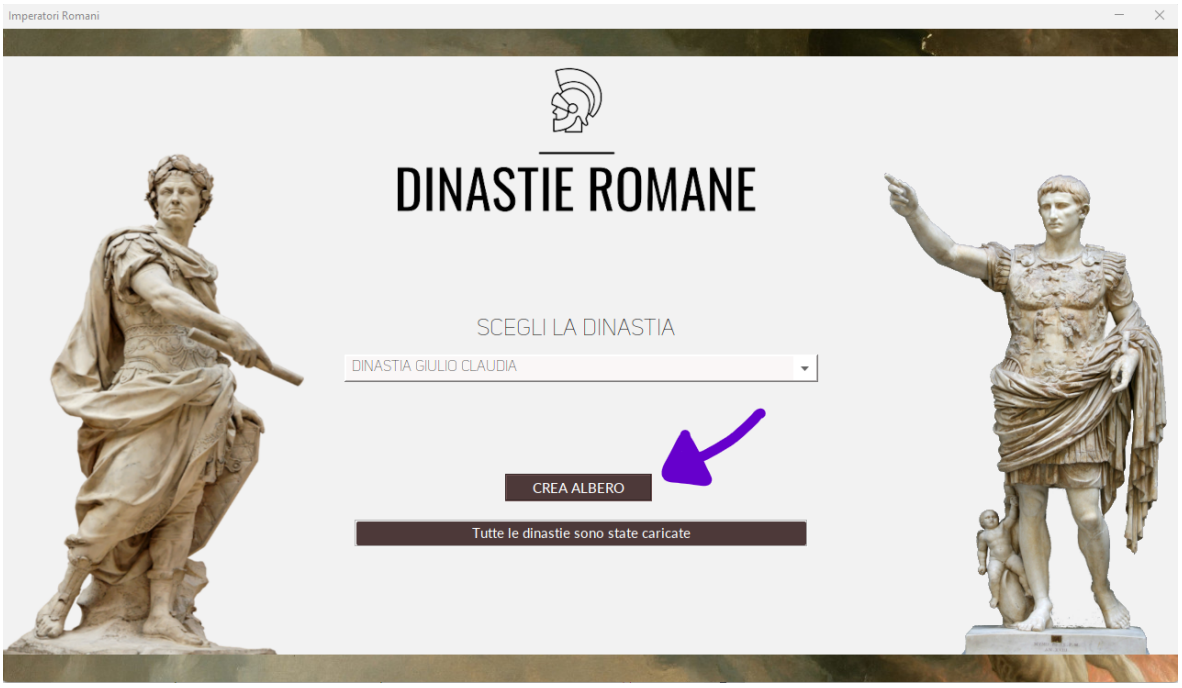
Slider

Inoltre, dopo aver scelto il numero di dinastie da caricare e premuto il tasto "CARICA DATI", la finestra offre una visualizzazione del caricamento dell'albero genealogico mediante una barra di caricamento, dove è possibile leggere i nomi che lo Scraper sta raccogliendo (vedi capitolo 2).



Finestra MainWindow durante il caricamento.

Elementi meno importanti sono le immagini che caratterizzano questa finestra, infatti è possibile osservare le statue di Cesare ed Augusto ai lati. Al fine di creare un ambiente che rappresentasse al meglio il tema degli anni romani, sono state inserite due frammenti del dipinto "Distruzione" del pittore Thomas Cole al bordo superiore ed inferiore della finestra.
Una volta che la barra avrà completato il suo caricamento, sarà possibile scegliere la dinastia e visualizzare il suo albero genealogico cliccando sul pulsante "CREA ALBERO" che fino a quel momento era disattivato.



Finestra MainWindow dopo il caricamento.

Cliccando il pulsante "CREA ALBERO" si aprirà una nuova finestra esterna, dando la possibilità all'utente di poter ritornare alla finestra MainWindow, scegliere una nuova dinastia e visualizzarla in contemporanea. Infatti le dinastie vengono caricate tutte insieme nello stesso momento al fine di poter visualizzare tutti gli alberi genealogici senza dover ogni volta scegliere la dinastia e ri-premere il pulsante "CREA ALBERO".

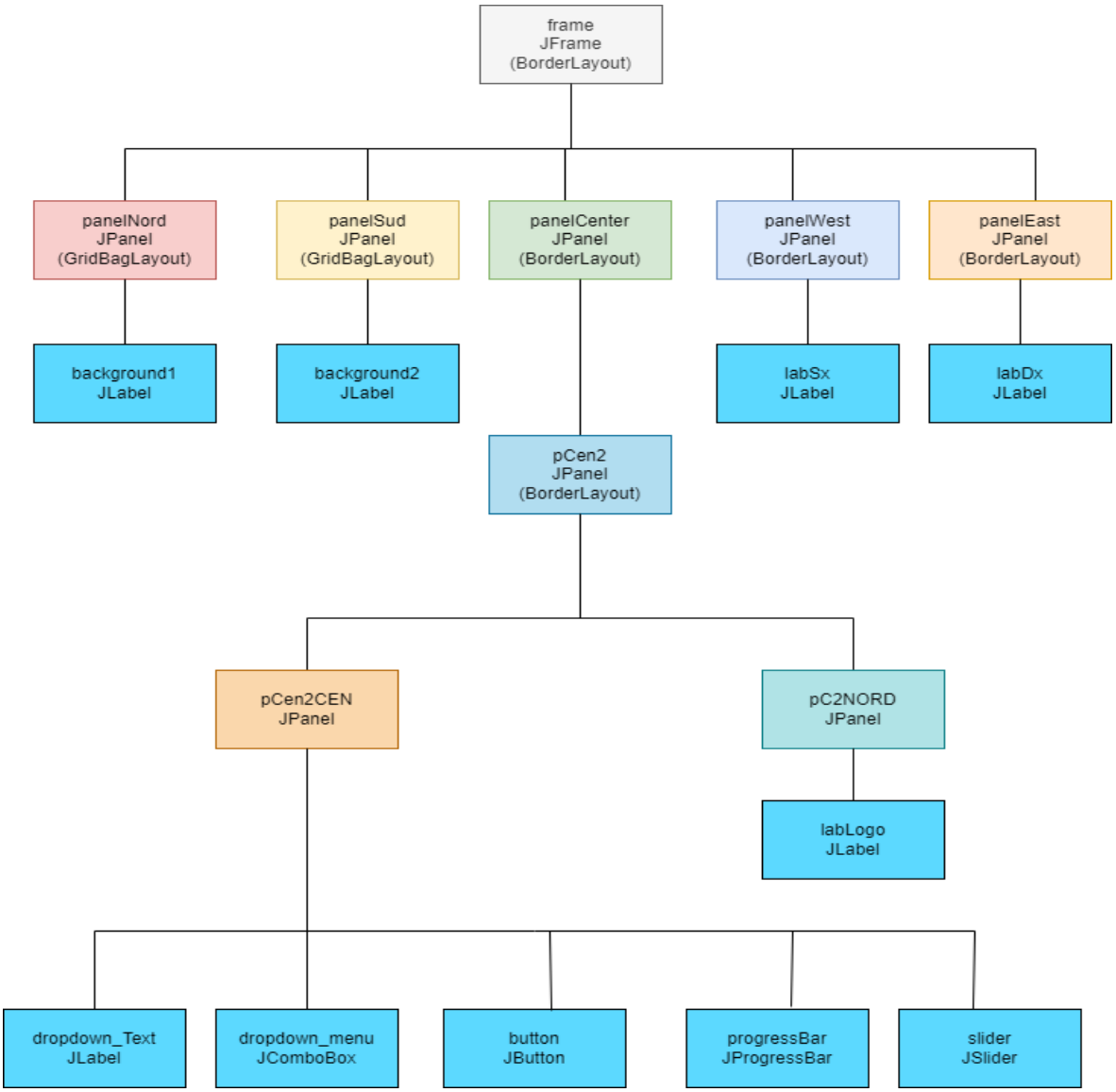
Il font presente è stato applicato attraverso un metodo statico della classe MyFont, che verrà approfondita nei capitoli successivi.
Il sito al quale è stato fatto riferimento al download dei vari font con estensione .ttf è: `https://www.1001fonts.com/minimalistic-fonts.html?page=1`*

In particolare sono stati utilizzati i seguenti font:

- Uni Sans Thin Italic;
- Uni Sans Thin;
- Uni Sans Heavy;
- Lato Thin;
- Lato Regular;

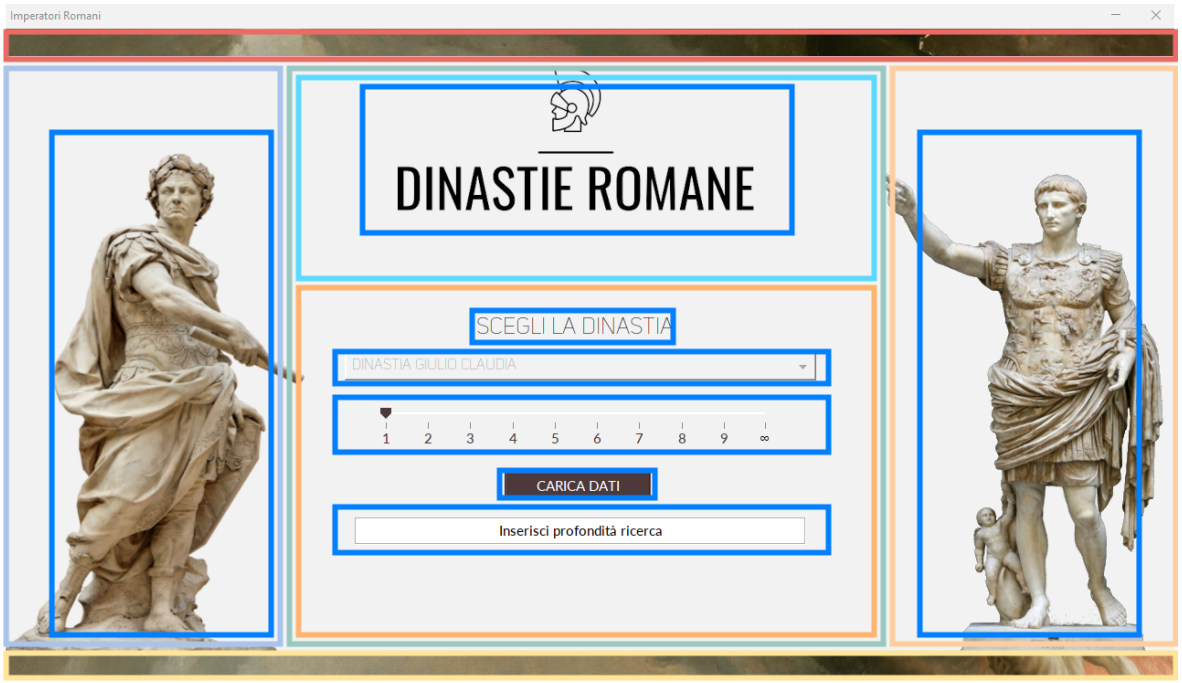
4.3.2 Layout Composition

Il Layout Composition ci aiuta a comprendere come un determinato frame è composto. Grazie ad esso sarà possibile analizzare le caratteristiche specifiche dei vari Layout Manager.



Per ogni rettangolo presente nello schema soprastante, ci sono informazioni sul componente: nella prima riga il nome del componente, nella seconda il tipo e infine nell'ultima riga il Layout Manager presente. I rettangoli che hanno come sfondo il colore blu sono componenti diversi da JPanel.

Possiamo visualizzare il seguente schema direttamente applicato sulla nostra finestra MainWindow:

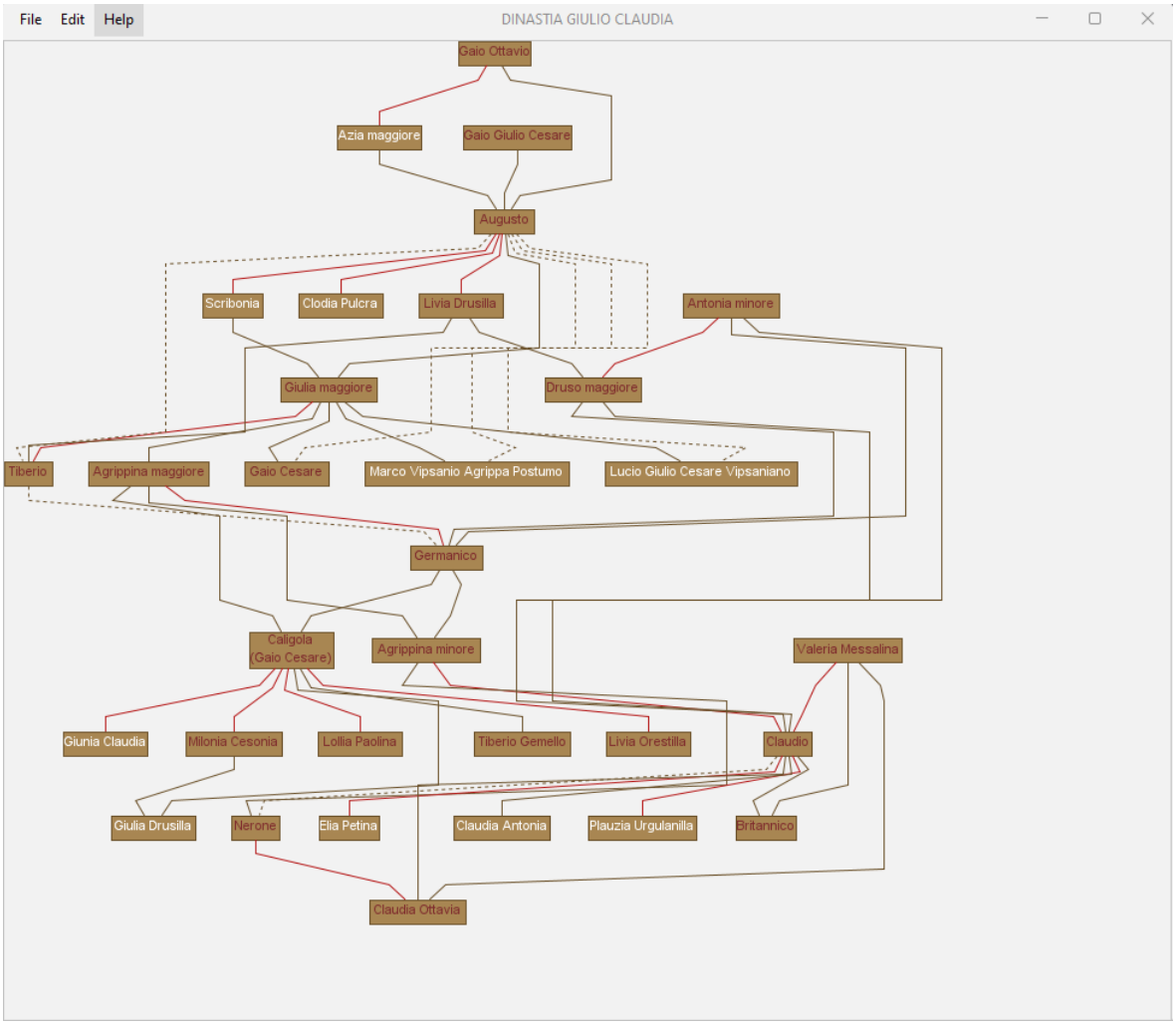


Finestra MainWindow con Layout Composition.

4.4 TreeWindow

TreeWindow è la finestra che si apre una volta cliccato il tasto "CREA ALBERO" nel frame MainWindow. Essa raffigura l'albero genealogico della dinastia Romana scelta precedentemente.
La realizzazione dell'albero è stata discussa nei capitoli precedenti (vedi capitolo 3).
A livello strutturale, oltre all'albero, la finestra può sembrare priva di funzionalità; vedremo che invece presenta alcune implementazioni tra le quali lo zoom, il drag, poter salvare il grafico (in formato jpeg), disegnarci sopra e cambiare colore alle varie relazioni.

La finestra TreeWindow si presenta così:



Finestra TreeWindow.

Possiamo notare di come la finestra è composta solo dall'albero e da una barra menù posizionata in alto.
I colori di base delle varie relazioni sono:

- Linea continua rossa raffigura la relazione con la moglie;
- Linea continua oro rappresenta la relazione con il figlio;
- Linea tratteggiata oro raffigura la relazione con il figlio adottivo;

Cliccando col tasto sinistro su uno dei rettangoli del grafico si viene reindirizzati alla pagina Wikipedia di tale personaggio storico.
I personaggi con il nome di colore rosso sono considerati VIP (vedi capitolo 2 e 3)

Vediamo ora le varie funzionalità che questa finestra offre...

4.4.1 Zoom

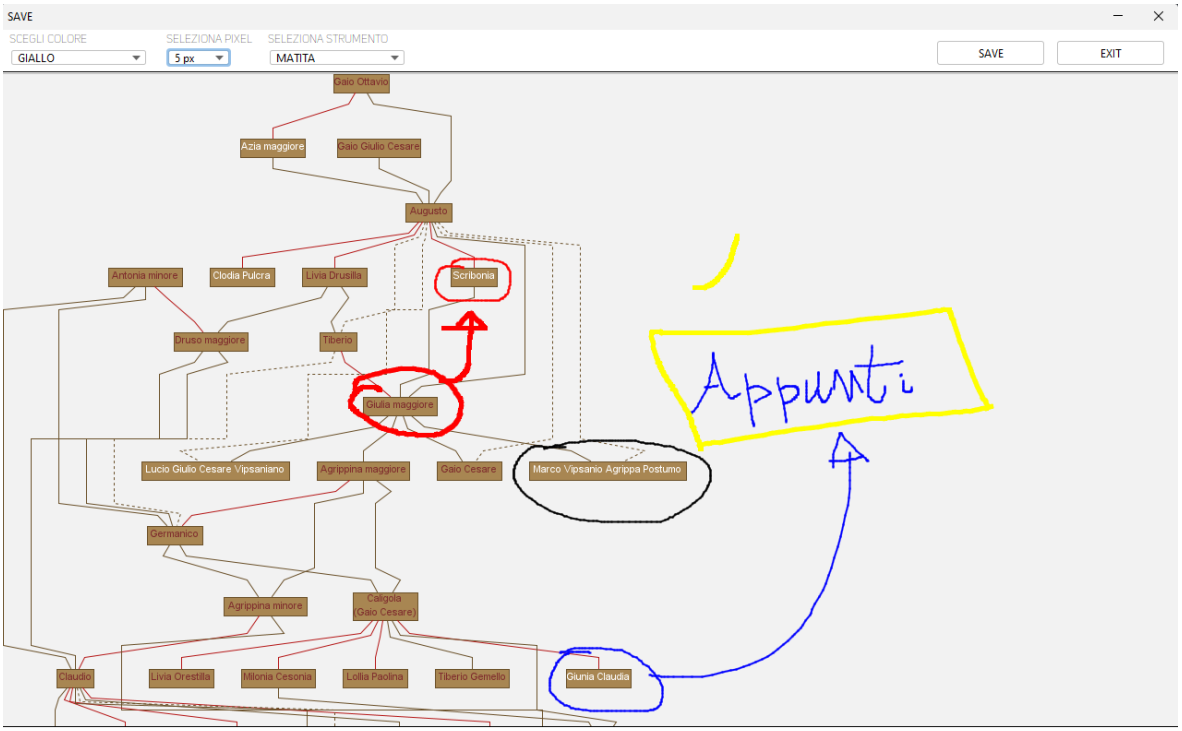
Lo zoom è una funzione del grafo ed è implementato grazie al *mouseWheelListerner*, per cui se il tasto control è premuto e nel mentre viene ruotando la rotella in avanti otteniamo un ingrandimento dello schermo, invece se viene ruotata all'indietro avviene il rimpicciolimento. Grazie al counter initialZoom che viene ogniqualvolta incrementata e diminuita, se e solo se initialZoom è maggiore o uguale a zero avviene lo zoom out per evitare che il grafo diventi più piccolo della dimensione originale.

4.4.2 Disegno e Salvataggio: Introduzione

Accedere al disegno e al salvataggio del file è possibile mediante due metodi:

- Accedere dal menu File $\longrightarrow$ Save;
- Premere ctrl + s;

Una volta avviata la procedura di salvataggio, verrà catturato il pannello sul quale risiede l'albero e si aprirà una finestra esterna SaveWindow (approfondita nel capitolo successivo) dove poter disegnare ed appuntare i propri schemi/appunti.

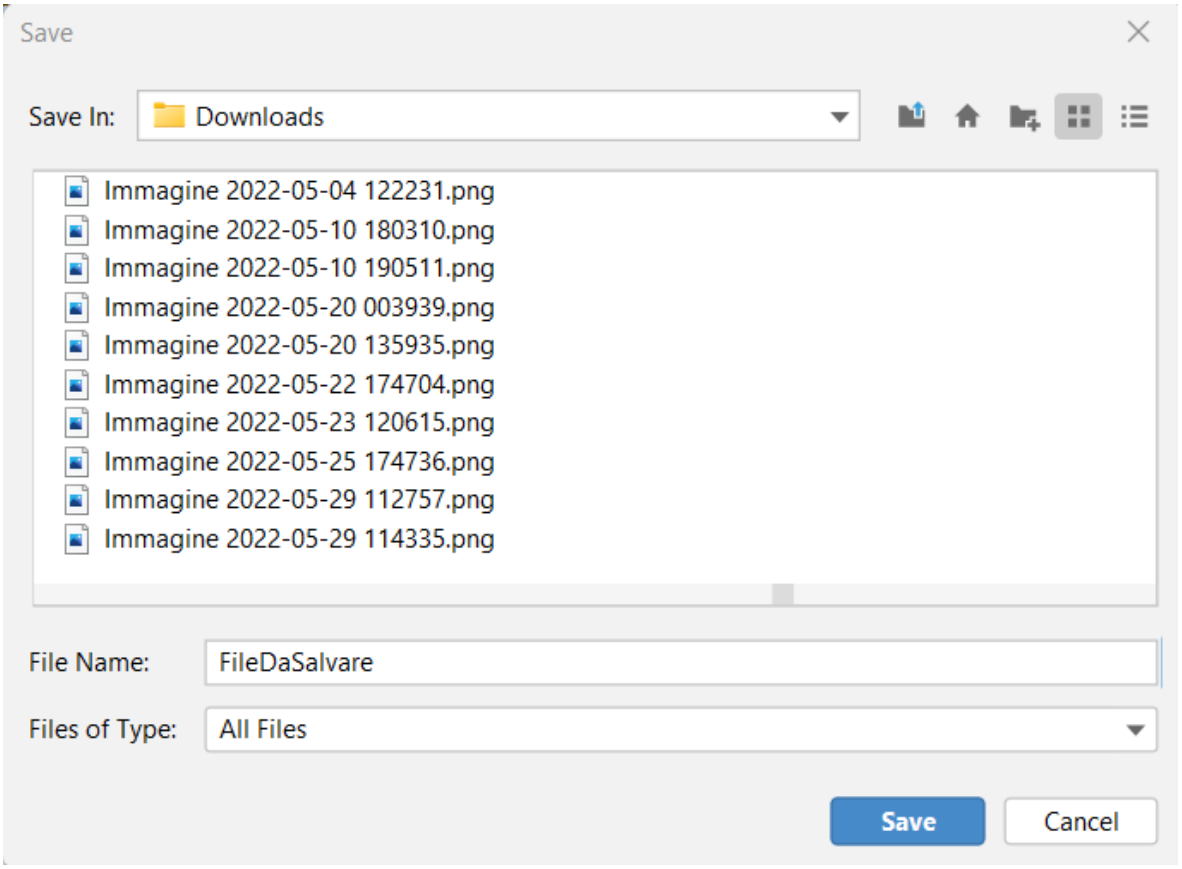


Possiamo notare come questa finestra presenti diversi elementi utili in fase di disegno raggruppati nella barra in alto.

Barra Strumenti Disegno.

Gli strumenti selezionabili sono due: matita e gomma. Ognuno di questi due strumenti è regolabile in dimensione selezionando i pixel nel menù a tendina dedicato. Utilizzando la matita è possibile scegliere uno dei 4 colori (nero, giallo, rosso, blu) selezionabili nel menu a tendina "Scegli Colore".

Una volta finito il disegno (o anche senza) è possibile salvare il file in formato jpeg. Questo processo è reso possibile tramite il tasto "SAVE" nel menu della barra in alto. Cliccandolo infatti si aprirà una finestra di dialogo di file dove l'utente potrà scegliere il percorso di destinazione dove salvare l'immagine. La cartella di partenza è quella di Download.



Finestra Save.

Questa finestra è stata implementata grazie all'utilizzo della classe JFileChooser. Essa fornisce un semplice meccanismo accompagnato da una GUI per consentire all'utente di navigare nel file system del nostro dispositivo come mostrato nell'immagine sopra.

La visualizzazione del file chooser è avvenuta utilizzando l'API di JFileChooser per mostrare una finestra di dialogo (Dialog Window) contenente il selettore di file.

```
//Create a file chooser
JFileChooser jF = new JFileChooser();
//Set the current directory
jF.setCurrentDirectory(new File( pathname: System.getProperty("user.home") + System.getProperty("file.separator")+ "Downloads"));
//In response to a button click
int res = jF.showSaveDialog( parent: this);
if(res == JFileChooser.APPROVE_OPTION){
    File file = jF.getSelectedFile();
    try
    {
        ImageIO.write(image, formatName: "jpeg", new File( pathname: file.getPath() + ".jpeg"));
    }
    catch(Exception exception)
    {
        JOptionPane.showMessageDialog( parentComponent: null, exception.getMessage());
    }
}
```

Parte di codice.

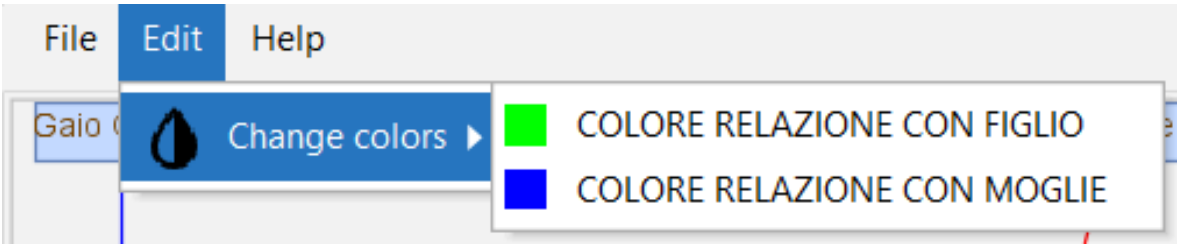
Il salvataggio effettivo dell'immagine è avvenuto utilizzando la classe ImageIO che

fornisce un modo semplice di salvarle in vari formati.

4.4.3 Cambiare Colore alle Relazioni

Un'altra funzionalità che offre la finestra TreeWindow è il poter cambiare i colori delle varie relazioni; ci si accede dal menù Edit ⟶ Change Colors ⟶ si sceglie la relazione. Le relazioni che possono cambiare colore sono:

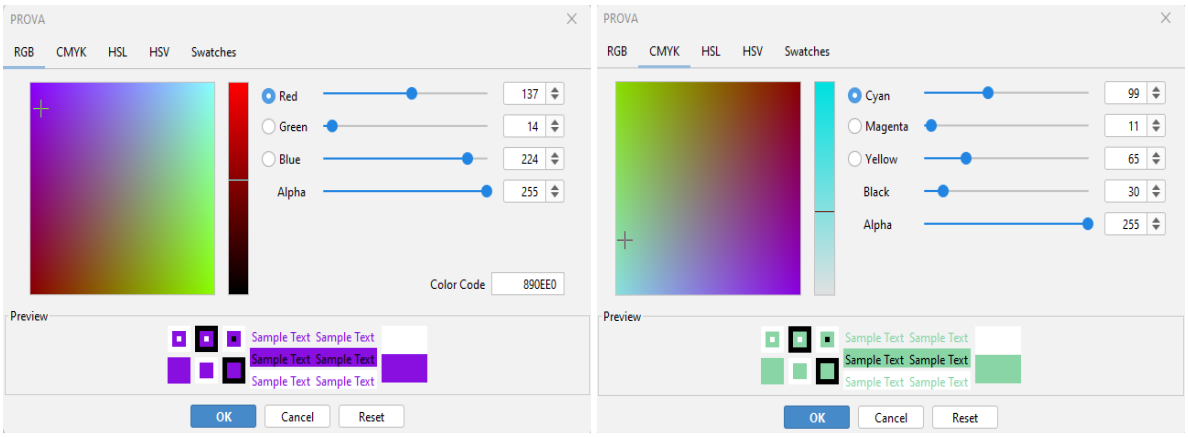
- Relazione con moglie;
- Relazione con figlio;

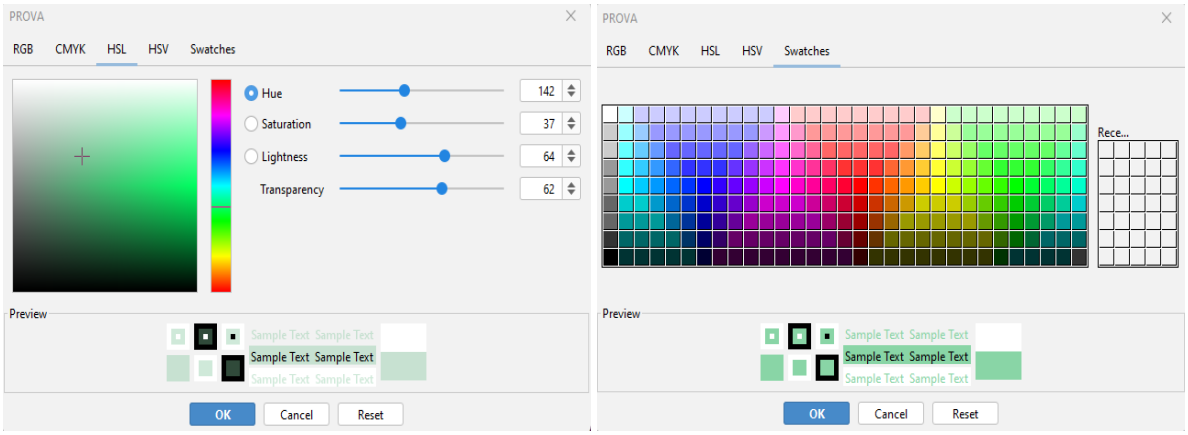


Menu Cambia Colore.

Notiamo che alla sinistra del testo "COLORE RELAZIONE CON ..." è rappresentato il colore della relazione attraverso un quadratino. La creazione di tali quadratini ha portato modifiche estensive alla classe JMenuItem; infatti, l'ultimo livello del menù sono oggetti di tipo BackgroundMenuItem che estende JMenuItem. Lo scopo della BackgroundMenuItem è quello di disegnare i quadratini e rendere possibile il set di colori di quest'ultimi.

Una volta scelta la relazione, si aprirà una finestra dove sarà possibile scegliere tra un vastissimo numero di colori.





Questi pannelli sono stati implementati grazie all'utilizzo della classe JColorChooser, che fornisce all'utente la possibilità di scegliere un colore attraverso la sua API.

4.5 SaveWindow

In questo sottocapitolo verrà approfondita la classe SaveWindow, in particolar modo l'implementazione del disegno.

L'elemento caratterizzante di questa funzionalità è sicuramente l'implementazione di MouseMotionListener, interfaccia interessata alla ricezione di eventi legati al movimento del mouse: quando viene rilevato un movimento del mouse, viene richiamato il metodo pertinente nell'oggetto listener e gli viene passato MouseEvent.
Il metodo dell'interfaccia principalmente utilizzato è chiamato mouseDragged, invocato quando un pulsante del mouse viene premuto su un componente e poi trascinato. Gli eventi continueranno ad esser inviati fino al rilascio del pulsante del mouse. Ci servirà anche l'utilizzo del metodo mousePressed dell'interfaccia MouseListener, richiamato quando il pulsante del mouse è stato cliccato su un componente.

Un altro elemento vitale alla realizzazione del Disegno è l'utilizzo della classe Graphics2D che estende Graphics e che fornisce un controllo più sofisticato sulla geometria, sulle trasformazioni delle coordinate, sulla gestione del colore e sul layout del testo.

4.5.1 Funzionamento

Il meccanismo che sta alla base è il disegno di linee da un punto (x1,y1) ad un punto (x2,y2). Infatti quando disegniamo con la matita sul grafico non facciamo altro che disegnare tante piccole linee continue tutte unite tra di loro.
Appena clicchiamo con il mouse sul componente da disegnare, viene richiamato il metodo mousePressed che ci salverà in due variabili la posizione x,y (oldX, oldY).

```java
@Override
public void mousePressed(MouseEvent e) {
    oldX = e.getX();
    oldY = e.getY();
}
```

Appena iniziamo a spostare il mouse (quindi a trascinarlo) viene invocato il metodo mouseDragged che come prima cosa salverà la posizione x,y in altre due variabili (currentX, currentY) e successivamente disegnerà la linea dalla posizione (oldX, oldY) alla posizione (currentX, currentY). Infine assegnerà ad oldX e oldY i valori currentX e currentY.

```java
//salvo la posizione corrente del mouse
currentX = e.getX();
currentY = e.getY() ;
//imposto la grandezza della matita
g.setStroke(new BasicStroke(pixel));
//disegno
g.drawLine(oldX , oldY , currentX , currentY);
//salvo la posizione corrente nella posizione precedente
oldX = currentX;
oldY = currentY;
```

Questo meccanismo smetterà di funzionare appena rilasciamo il click del mouse.
Il cambio di colore della matita avviene attraverso il richiamo del metodo setColor della classe Graphics2D.

4.6 Altre Classi

Oltre alle classi già descritte in precedenza, il progetto dispone di altre classi secondarie meno corpose.

4.6.1 Classe MyFont

MyFont (citata già in precedenza) è la classe che si occupa di modificare il font di un testo di un certo componente. Presenta un solo metodo statico che verrà chiamato dai vari componenti quando viene chiamato il metodo setFont.
Il metodo statico, chiamato creaFont, prende in input due valori:

- Percorso di un file avente estensione .TTF (il font scaricato);
- un numero a virgola mobile rappresentante la dimensione del font;

Alla fine ritorna un oggetto di tipo Font.

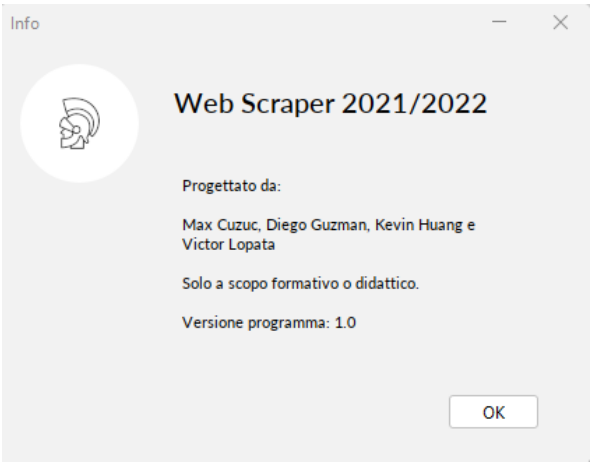
Il sito al quale è stato fatto riferimento al download dei vari font con estensione .ttf è: `https://www.1001fonts.com/minimalistic-fonts.html?page=1`*

In particolare sono stati utilizzati i seguenti font:

- Uni Sans Thin Italic;
- Uni Sans Thin;
- Uni Sans Heavy;
- Lato Thin;
- Lato Regular;

4.6.2 Classe InfoWindow

InfoWindow è la classe che implementa la finestra che si accede dal menù tramite Help $\longrightarrow$ Info oppure con ctrl + i.



Finestra InfoWindow.

Fine

E con questo abbiamo analizzato tutti gli aspetti del Web Scraper Wikipedia sugli Imperatori Romani. È stato molto frustrante gestire tante, ma tante eccezioni che Wikipedia ha, e capire come sfruttare al meglio Swing, nonostante ciò, al contempo abbiamo imparato molte nozioni nuove e a lavorare in gruppo. Abbiamo avuto una vera sensazione di aver compiuto un progetto serio e di questo ne siamo grati.