Relazione progetto metodologie di programmazione

Ian Tirso Cini 1933403

Settembre 2021

Contents

1 Introduzione

Il progetto sviluppato tra quelli proposti è quello del webscraper di Wikipedia sugli imperatori romani. Nonostante sia un progetto da quattro persone, ho deciso di procedere con la realizzazione di questa proposta che mi è sembrata sin da subito molto interessante.

Dovendo lavorare da solo la parte più importante è stata sin da subito quella della progettazione di packages e classi; per rispettare gli insegnamenti ricevuti durante il corso, oltre che le richieste dei progetti, ho individuato la presenza di tre aree di competenza in cui suddividere il lavoro:

1. Scraping e recupero dati utili da Wikipedia.
2. Modellazione dei dati raccolti per in un albero genealogico.
3. Rappresentazione nella gui dell'albero genealogico già costruito.

Nella fase di studio preliminare alla fase di scrittura del codice ho fatto degli approfondimenti sui design patterns per poterli integrare all'interno del codice. Da questo approfondimento decido di utilizzare il pattern Model-View-Controller che sembra ben calzare per le richieste di modularità e riutilizzabilità del codice.

A conclusione del lavoro i packges del progetto sono iRomaniModel dove viene elaborata la parte dei dati, iRomaniView dove trovano posto le classi che riguardano l'interfaccia e iRomaniController con la classe Controller per unire model a view e la classe iRomaniMain per eseguire il programma.

Il compimento di questi packages è stato successivo a quelli di wikipediaWebScraperLib e alberoGenealogicoLib. Questi due packages sono nati a seguito della decisione di sviluppare il progetto partendo dalla soluzione più generale al problema. I packages iRomani che compongono la realizzazione del progetto sono stati in buona parte sviluppati partendo dalle funzionalità sviluppate in questi altri due.

Nel testo sono presenti le descrizioni di tutte le classi e la descrizione sulle funzionalità sviluppate nel codice spiegando i passi necessari per l'ottenimento dei dati.

2 Descrizione delle classi

2.1 wikipediaScraperLib

Il package contiene le classi per effettuare lo scraping su di una qualsiasi pagina Wikipedia. WikipediaNavigator si occupa di prelevare il sorgente da una qualsiasi wiki fornita e ci sono classi di utilità effettuano l'analisi del codice per recuperarne le informazioni utili e salvare in rispettive classi.

Nel package si rispetta il principio di Singola responsabilità per tute le classi, assegnando ad ognuna un preciso compito. Ci sono classi astratte che vengono estese da classi concrete per rispettare i principi di Open/Close e Liskov Substitution (TabellaWikipedia, ParserTabellaWikipedia, WikipediaWebPage).

2.1.1 WikipediaNavigator

class WikipediaNavigator
-webDriver : WebDriver
+WikipediaNavigator()
-buildChrome() +getHtmlPagina(String) throws WikipediaUrlErratoException : String -urlValido(String url) : boolean +closeBrowser() : void

La classe incapsula alcune funzionalità fornite dalla libreria Selenium, in particolare delle classi WebDriver e ChromeDriver.

Il costruttore chiama il metodo privato buildChrome() per inizializzare il browser, incapsulandone così l'implementazione.

Il metodo *getHtmlPagina()* utilizza il metodo privato *urlValido(..)* che effettua un controlla sulla stringa passata come parametro sia un url valido di wikipedia, cioè che cominci con https://it.wikipedia.org/wiki/ altrimenti viene lanciato l'errore WikipediaUrlErratoException. Se l'url è corretto il metodo ritorna il sorgente html della pagina richiesta.

closeBrowser() chiude Chrome.

2.1.2 WikipediaUrlErratoException

class WikipediaUrlErratoException extends Exception
+WikipediaUrlErratoException() +WikipediaUrlErratoException(String)

La classe estende Exception ed è l'errore che viene lanciato da WIkipediaNavigator quando viene inserito un url errato.

2.1.3 WikipediaWebPage

abstract class WikipediaWebPage
+abstract getTitle() : String
+abstract getUrl() : String

La classe è stata modellata per essere un'astrazione di una pagina Wikipedia. I metodi *getTitle()* e *getUrl()* sono astratti e sono le due informazioni che ogni pagina web non può non avere.

2.1.4 PaginaWikipedia

class PaginaWikipedia extends WikipediaWebPage
-url : String
-titoloPagina : String
-urlImmagine : String
-sinottico : Sinottico
-sinotticoHtml : String
+PaginaWikipedia(String, String, String , Sinottico, String)
+getTitle() : String
+getUrl() : String
+getUrlImmagine() : String
+getSinottico() : Sinottico
+toString() : String
+getSinotticoHtml() : String

La classe è un'implementazione di WikipediaWebPage. Gli attributi della classe sono tutte le informazioni che prelevo da una pagina di Wikipedia: url, titolo della pagina, implementato con una classe specifica il Sinottico e l'html del sinottico stesso. Non salvo il resto del sorgente in quanto non l'ho utilizzato ai fini del progetto.

Il costruttore prende tutti questi campi nella costruzione e ci sono i getter per ognuno. Non ci sono setter, voglio evitare che lo stato della classe cambi durante la sua esistenza perchè il contenuto di una pagina Wikipedia non muta durante l'esecuzione del programma.

Eventuali controlli di validità dei dati devono essere fatti esternamente, la classe si occupa solamente di contenere le informazioni e di fornirle all'esterno.

2.1.5 PaginaWikipediaBuilder

class PaginaWikipediaBuilder <Builder>
-url : String -titoloPagina : String -urlImmagine : String -sinottico : Sinottico -sinotticoHtml : String
+url(String urlPagina) : PaginaWikipediaBuilder +titoloPagina(String titoloPagina) : PaginaWikipediaBuilder +urlImmagine(String urlImmagine) : PaginaWikipediaBuilder +sinottico(Sinottico sinottico) : PaginaWikipediaBuilder +sinotticoHtml(String sinotticoHtml) : PaginaWikipediaBuilder +build() : PaginaWikipedia +reset() : void

La classe è realizzata con il design pattern builder. Ha i metodi per ricevere i parametri necessari alla costruzione di un oggetto PaginaWikipedia che restituisce con il metodo *build()*. Avendo lavorato con l'intenzione di progettare una libreria ho deciso di introdurre questa classe per un possibile utilizzo da parte del programmatore.

2.1.6 TabellaWikipedia

abstract class TabellaWikipedia <T extends RigaTabella>
+abstract getRiga(String identificativoRiga) : T +abstract addRiga(T riga) : void +abstract getRighe() : Collection <T>

La classe astratta rappresenta una tabella su Wikipedia. Essendoci diversi tipi di tabelle ho solo imposto dei metodi necessari. Per questa classe ho usato un generico che implementi l'interfaccia riga tabella, questo per evitare che la tabella possa essere riempita di oggetti che siano diverse dalle righe ed anche

per obbligare a modellare una classe che rappresenti la riga di una tabella a seconda delle esigenze.

I metodi della classe sono per aggiungere o recuperare una riga secondo un indentificativo di tipo stringa, e un metodo per ottenere tutte le righe in una Collection di preferenza.

2.1.7 RigaTabella

interface RigaTabella

E' un'interfaccia che mi permette di poter utilizzare una classe che estenda *TabellaWikipedia* rispettando le richieste del tipo generico inserito. L'interfaccia è priva di metodi in quanto un riga può essere fatta in modo totalmente diversa rispetto un'altra, lasciando quindi al programmatore l'implementazione.

2.1.8 Sinottico

class Sinottico extends TabellaWikipedia implements Iterable
-righe : List<RigaSinottico>
+addRiga(RigaTabella riga) : void
+creaRiga(String categoria) : void
+addInfoToRiga(String categoria, String informazione)
+addInfoEUrlToRiga(String categoria, String informazione, String url) : void
+addUrlToInfoToRiga(String categoria, String informazione, String url) : void
+getRiga(String categoria) : RigaSinottico
+getRighe() : List<RigaSinottico>
+iterator() : Iterator<RigaSinottico>

La classe rappresenta un sinottico, ovvero il box a destra che contiene le informazioni principali del contenuto di una pagina Wikipedia. Ho modellato il sinottico in questo modo: il sinottico essendo una tabella html estende *TabellaWikipedia*; la tabella è composta da righe e quindi ho modellato una classe **RigaSinottico** e quindi una List che le raccolga.

Per la classe ho anche implementato l'interfaccia Iterable, fornendo al chiamente del metodo da implementare l'iterator della lista contenente le righe. Implementando questa interfaccia rendo anche più semplice la consultazione delle righe con il for-each di Java.

I metodi della classe: *addRiga(..)* permette di aggiungere al sinottico una riga già istanziata.

creaRiga(..) crea una nuova riga tramite passando come argomento la categoria della informazioni contenute nella riga, ovvero la cella sinistra della riga del sinottico.

addInfoToRiga(..) fornendo il della categoria della cella sinistra, cerca la riga corrispondente ed aggiunge l'informazione ovvero il contenuto della cella destra.

addInfoEUrlToRiga(..) fornendo il nome della categoria, aggiunge alla riga corrispondente l'informazione contenuta nella cella destra, che possiede anche un collegamento verso un'altra pagina Wikipedia.

addUrlToInfoToRiga(..) nome della categoria (cella sinistra) e l'informazione (cella destra), aggiunge l'url a quest'ultima.

getRiga(..) passando come argomento il contenuto della cella sinistra cerca la riga corrispondente e la ritorna.

getRiga() ritorna una lista contenente tutte le righe del sinottico.

2.1.9 RigaSinottico

class RigaSinottico implements RigaTabella, Iterable
-categoria : String -cellaDestra : List<Informazione>
+RigaSinottico(String categoria)
+getCategoria() : String +getInformazioni() : List<Informazione> +iterator() : Iterator<Informazione> +addInformazione(String nomeInfo) : void +addInformazione(String nomeInfo, String url) : void +addUrl(String nomeInfo, String url) : void +toString() : String

Figli	Giulia (da Cornelia); Cesarione (da Cleopatra); Ottaviano (pronipote adottato)
Gens	Iulia
Padre	Gaio Giulio Cesare

Figure 1: Porzione di sinottico

La classe modella una riga appartenente ad un sinottico di Wikipedia; ha una classe interna Informazione che rappresente il contenuto della cella destra. Il

campo categoria è il contenuto di una cella sinistra, che è il tipo di informazione presente nella riga.

cellaDestra è una lista di Informazione considerando che possono essercene più di una.

L'oggetto viene costruito passando quindi il nome della categoria che viene salvato nell'omonimo attributo.

getInformazioni() ritorna la lista di Informazione raccolte dalla classe.

I due metodi addInformazione(..) istanziano un'informazione richiamando i costruttori della classe Informazione (descritta subito dopo).

addUrl(..) ricerca nella lista delle informazioni quella di nome corrispondente al parametro nomeInfo passato al metodo ed aggiunger l'url.

Anche questa classe implementa il metodo iterable richiamando iterator dell'attributo cellaDestra per facilitarne la consultazione.

2.1.10 Informazione

class Informazione
-nomeInfo : String -url : String
+Informazione(String nomeInfo) +Informazione(String nomeInfo, String url)
+setUrl(String url) : void +getNomeInfo() : String +getUrl() : String +toString() String

Questa classe è interna a RigaSinottico rappresenta una singola informazione che si trova nella cella destra della riga di un sinottico di una pagina di Wikipedia. Salva al suo interno il nome dell'informazione, ovvero quello che può leggere in modo naturale dalla pagina Wikipedia, e se presente un url eventualmente collegato. L'url deve cominciare con https://it.wikipedia.org/wiki per essere aggiunto (i link in Wikipedia sono sempre verso l'interno); la classe controlla se l'indirizzo sia valido altrimenti lo imposta a null. La correttezza dell'inserimento dell'url deve essere fatta esternamente.

Quest'oggetto si può costruire o solo con l'informazione o con informazione ed url.

Si può aggiungere l'url anche successivamente con *setUrl(..)*, e sono forniti i getter per i due attributi.

2.1.11 RigaNonPresenteException

class RigaNonPresenteException extends Exception
+RigaNonPresenteException()

La classe è l'errore che viene lanciato se la riga cercata nel Sinottico non è presente.

2.1.12 ParserTabellaWikipedia

interface ParserTabellaWikipedia
+analizzaTabella(String sorgente) : TabellaWikipedia

Questa interfaccia è stata pensata per integrare diverse classi che si occupano di fare il parsing di un sorgente di Wikipedia. Una possibile estensione del codice potrebbe essere quella di utilizzare una libreria esterna per crearne uno nuova classe che parsa il sorgente da integrare poi nella libreria.

Il metodo da implementare è quello per far partire l'analisi del sorgente che viene passato come argomento.

2.1.13 ParserSinotticoWikipedia

class ParserSinotticoWikipedia implements ParserTabellaWikipedia
+analizzaTabella(String sinottico) : Sinottico
-static createSinottico(String sinottico) : Sinottico
-static analizzaRiga(String riga) : RigaSinottico
+static testoEsternoTag(String riga) : String
-static estraiTestoEsternoTag(String riga) : String
+static estrapolaLink(String riga) : String
-static estraiLink(String riga) : String
-static analisiCella(String riga, RigaSinottico rigaSinottico) : void
-static testoTraParentesi(String riga, int start) : int
-static internoTag(String, int, int, RigaSinottico) : int
-static esternoTag(String riga, int start, int end, RigaSinottico rigaSinot.) : int

La classe si occupa di analizzare un sorgente html di Wikipedia e di creare l'oggetto Sinottico. Implementa l'interfaccia ParserTabellaWikipedia e quindi il tipo di ritorno Sinottico è conforme in quanto è un'estensione di TabellaWikipedia.

I metodi sono statici perchè la classe è composta solamente da algoritmi che svolgono la funzione di analisi per diversi casi e fasi e non c'è nessuno stato interno da mantenere. La maggior parte dei metodi sono privati per nascondere l'implementazione interna della classe, ci sono solamente due metodi pubblici, oltre ad analizzaTabella ereditato dall'interfaccia, che spiegherò in seguito. Per poter controllare meglio il flusso di esecuzione, ho diviso ogni fase che ho individuato in un metodo seguendo il principio di singola responsabilità; la classe è articolata ma in questo modo sono riuscito a seguire il flusso dell'esecuzione individuando con precisione i metodi da sistemare.

La classe si occupa solo di analizzare il sinottico, deve essere quindi passato quella parte di sorgente.

createSinottico(..) è il metodo che inizia il processo; istanzia il nuovo sinottico e vi aggiunge le righe (come oggetti di tipo RigaSinottico) che gli vengono passate dagli altri metodi. In questo metodo il sorgente viene sezionato nelle righe che verranno analizzate dagli altri metodi.

analizzaRiga(..) effettua un controllo sulla riga, ovvero che questa sia formato da due colonne, altrimenti non è una riga utile.

estraiTestoEsternoTag(..) della riga del sinottico preleva il testo che si trova al di fuori di qualsiasi tag html lavorando in maniera ricorsiva.

estraiLink(..) riceve una porzione della riga che contiene il tag href, e preleva la parte che riguarda solamente l'url; viene aggiunto il resto dell'indirizzo di wikipedia che non è presente nel tag.

analisiCella(..) il metodo si occupa di leggere carattere per carattere la stringa con il sorgente html della riga del sinottico; i casi principalmente distinti sono quelli di individuazione di un tag html, di testo tra parentesi tonde, di testo fuori da un tag. A secondo della casistica invia la porzione di riga verso gli altri metodi specializzati. Il funzionamento è tramite due indici, start ed end, che vengono passati agli altri metodi e che vengono ogni volta aggiornati dopo aver concluso la parte di analisi.

testTraParentesi(..) se è stata trovata una parentesi tonda scarta tutto il testo contenuto all'interno, solitamente sono rimandi verso pagine che non sono utili ai fini del progetto o note varie; ho creato questo metodo per non dover cablare nel codice troppi casi particolari nella fase di costruzione degli alberi genealogici.

internoTag(..) il metodo riceve la riga in analisi nel momento in cui viene trovato un tag; se il tag è quello di un link viene inviata la stringa al metodo che recupera il testo esterno, e poi viene recuperato il link con il metodo *estraiLink(..)*, vengono poi aggiunte le informazioni ricavate alla rigaSinottico che era stata passata tra gli argomenti.

esternoTag(..) manda prima la porzione di riga al metodo per estrarre il testo, e poi controlla che non si tratti di qualche caso che potrebbe sporcare il risultato, in caso contrario aggiunge alla RigaSinottico.

I due metodi pubblici *testoEsternoTag(..)* e *estrapolaLink(..)* richiamano i due metodi dai nomi simili per poter essere acceduti anche all'esterno.

2.1.14 AnalisiInfoWikipedia

class AnalisiInfoWikipedia
+urlImmagine(String sorgente) : String
+urlPagina(String sorgente) : String
+titoloPagina(String sorgente) : String
+sinottico(String sorgente) : Sinottico
+sinottico(String sorgente, Supplier<ParserTabellaWikipedia>parser) : Sinottico
+estraiSinottico(String sorgente) : String

La classe è costituita da metodi di utilità per ricavare informazioni da un sorgente di una pagina di Wikipedia. E' composta da metodi statici non avendo nessun tipo di stato interno; riceve il sorgente e restituisce il risultato dal metodo chiamato.

urlImmagine(..) recupera l'url dell'immagine principale del sorgente inviato.

urlPagina(..) dal sorgente ricava l'url della pagina corrispondente, presente all'interno del codice.

titoloPagina(..) dal sorgente recupera il titolo della pagina.

sinottico(String sorgente) si occupa di inviare alla classe ParserSinotticoWikipedia il codice con il sinottico da analizzare. Utilizza il metodo *estraiSinottico* per sezionare il codice in maniera corretta.

estraiSinottico(String, Supplier<ParserTabellaWikipedia>) è un metodo di utilità pensato per una possibile estensione del codice che sfruttanda la programmazione funzionale; la classe che effettua il parsing del codice viene passato tramite un Supplier potendo quindi utilizzare un altra classe parser che implementi l'interfaccia ParserTabellaWikipedia.

estraiSinottico(..) si occupa di prelevare dal sorgente di Wikipedia solamente la parte contenente il sinottico.

2.2 alberoGenealogicoLib

Il package contiene delle classi per costruire un albero genealogico. Ho utilizzato JGraphT all'interno delle classi ed i metodi più adatti allo scopo aggiungendone altri per una comoda consultazione.

Nelle classi si rispetta sempre il principio di Singola Responsabilità, si rispetta il principio di Open/Close con l'interfaccia Persona, e la progettazione di AlberoGenealogico che prende oggetti che implementano Persona tende per l'utilizzo del principio di Liskov substitution ed al polimorfismo.

2.2.1 AlberoGenealogico

class AlberoGenealogico
-albero : Graph<Persona, Archi> -nomeFamiglia : String -capostipite : Persona
+AlberoGenealogico(String nomeFamiglia)
+addPersona(Persona persona) : void +setCapostipite(Persona capostipite) : void +getCapostipite() : Persona +addFiglio(Persona genitore, Persona figlio) : void +deletePadreFiglio(Persona persona1, Persona persona2) : void +getFigli(Persona genitore) : List<Persona> +getGenitori(Persona figlio) : List<Persona> +getPersone() : Set<Persona> +getNomeFamiglia() : String +toString() : String +getAlbero() : Graph<Persona, Archi>

La classe sfrutta la libreria JGraphT incapsulando il suo utilizzo all'interno. Si possono inserire le relazioni tra genitore e figlio; e ricevere le informazioni riguardo l'albero genealogico costruito.

I campi della classe sono: *albero* per salvare il grafo di JGraphT, istanziato come grafo semplice orientato, utilizzando come vertici la classe *Persona* e come archi la classe *Archi*, estensione di DeafultEdge di JGraphT; *nomeFamiglia* il nome della famiglia ed il *capoStipite* ovvero la radice della famiglia o possibile punto di accesso al grafo.

addPersona(..), *addFiglio(..)*, *deletePadreFiglio*, *getFigli(..)*, *getGenitori*, *getPersone* utilizzano i metodi di JGrapT per ritornare i risultati richiesti.

setCapostipite(..) imposta come capostipite la Persona passata come argomento.

getAlbero ritorna il grafo che si è costruito.

2.2.2 Archi

class Archi extends DefaultEdge
toString() : String

La classe estende DefaultEdge di JGraphT. L'estensione effettua l'override del metodo toString per sfruttare il polimorfismo in fase di costruzione dell'albero visuale in quanto viene chiamato il toString per scegliere la colorazione del vertice.

2.2.3 Persona

interface Persona
+getNome() : String

L'interfaccia necessaria per popolare la classe AlberoGenealogico.

2.3 iRomaniModel

Il package è l'implementazione della parte del codice che costruisce la parte dei dati riguardanti le genealogie degli imperatori romani, seguendo il pattern MVC da questo package si può fare l'accesso ai dati che saranno forniti alla parte view.

In tutte le classi viene rispettato il principio di Singola responsabilità; nella classe CostruisciAlberoGenealogico viene sfruttato il principio di Liskov substitution potendo inserire oggetti di tipo AnticoRomano e Imperatore all'interno dell'albero genealogico in quanto estensioni di Persona.

2.3.1 AnticoRomano

class AnticoRomano implements Persona, Comparable
-url : String -paginaWikipedia : PaginaWikipedia
+AnticoRomano(String nome, String url)
+getUrl() : String +setPaginaWikipedia(PaginaWikipedia paginaWikipedia) : void +getPaginaWikipedia() : PaginaWikipedia +equals(Object o) : boolean +compareTo(AnticoRomano romano) : int +hashCode() : int +toString() : String +thisPersonIs() : Boolean

La classe implementa Persona del package *alberoGenealogicoLib*. É modellata per essere la rappresentazione di un antico romano costruito sulla base delle informazioni ottenute dallo scraping su Wikipedia.

La classe implementa l'interfaccia Comparable perchè JGraphT utilizza i Set per salvare le informazioni e per essere sicuro della corretta integrazione ho deciso di implementare i metodi necessari: *equals*, *compareTo* e *hashCode*. Per equals ho implementato il confronto tra gli url, dato che per certo sono un elemento unico, per compareTo invece ho utilizzato la comparazione tra i nomi.

url è l'url corrispondente alla pagina Wikipedia.

paginaWikipedia è l'oggetto PaginaWikipedia del package wikipediaWebScraperLib, *setPaginaWikipedia(..)* e *getPaginaWikipedia()* impostano e ritornano la wiki costruita con l'oggetto PaginaWikipedia.

2.3.2 Imperatore

class Imperatore extends AnticoRomano
+Imperatore(String nome, String url)
+toString() : String

La classe estende AnticoRomano ed individua un imperatore romano.

La classe è stata creata per utilizzare il polimorfismo nella fase di creazione visuale dell'albero che chiama il metodo toString per creare le etichette da apporre sopra ai vertici.

2.3.3 WikiImperatoriRomaniPagina

class WikiImperatoriRomaniPagina <Singleton> extends WikipediaWebPage
-static WikiImperatoriRomaniPagina instance - List<TabellaDinastie>dinastie - String TITOLO_PAGINA - String URL_PAGINA
-WikiImperatoriRomaniPagina()
+ getInstance() : WikiImperatoriRomaniPagina + getTitle() : String + getUrl() : String + getDinastie() : List<TabellaDinastie> - analisiSorgentePagina() : void + getElencoDinastie() : String[] + getDinastia(String nomeDinastia) : TabellaDinastie

La classe estende WikipediaWebPage del package wikipediaWebScraperLib e rappresenta la pagina Wikipedia dove sono elencate tutte le dinastie con gli imperatori romani.

In questa classe ho optato per l'utilizzo del desing patter singleton: la classe rappresenta una precisa pagina di Wikipedia e quindi è unica; per essere costruita deve essere eseguito uno scraping sulla pagina corrispondente e per ottimizzarne l'utilizzo ho preferito evitare ogni volta di ripetere questa operazione.

Il campo *istance* seguendo la struttura del design pattern, ritorna l'istanza dell'oggetto

Il campo *dinastie* contiene oggetti di tipo *TabellaDinastie* dove vengono raccolti i risultati dello scraping sulle informazioni degli imperatori romani.

TITOLO_PAGINA e *URL_PAGINA* sono statici dato che la classe si riferisce ad un oggetto che modella una precisa entità.

Il costruttore è privato e viene chiamato dal metodo *getInstance()* che ritorna l'istanza dell'oggetto una volta pronto. Il costruttore chiama il metodo privato *analisiSorgentePagina()* in cui viene creata un'istanza di WikipediaNavigator che preleva il sorgente dall'url in *URL_PAGINA*, poi utilizza le classi *ParserTabellaDinastie* e *TabellaDinastie* (descritte a seguire) per recuperare le informazioni dalla pagina.

I metodi *getTitle()* e *getUrl()* ritornano titolo e url della pagina Wikipedia.

getDinastie() ritorna la lista con tutte le dinastie.

getElencoDinastie() cicla su tutte le TabelleDinastie nella lista *dinastie* e ritorna un array di stringhe con i nomi di tutte le dinastie.

getDinastia(..) ritorna la TabellaDinastia che ha lo stesso nome di quello passato come argomento. Se non viene trovata nessuna dinastia corrispondente viene lanciato l'errore DinastiaNonTrovataException.

2.3.4 DinastiaNonTrovataException

class DinastiaNonTrovataException
+DinastiaNonTrovataException() : void

Classe di errore che viene lanciata da WikiImperatoriRomaniPagina quando viene cercata una dinastia tramite nome e questa non è presente.

2.3.5 TabellaDinastie

La classe rappresenta le tabelle presenti all'interno della pagina wiki con gli imperatori romani. Per ogni dinastia deve essere creato un oggetto di tipo TabellaDinastie. La classe estende TabellaWikipedia e prende come tipo generico per le righe quello dalla sua classe interna Riga, quindi *TabellaDinastie.Riga*. Implementa l'interfaccia Iterable iterando sulla lista che contiene le righe.

Il costruttore della classe prende come parametro il nome della dinastia che viene salvato in *nomeDinastia*.

Sono disponibili il set ed il get per *urlDinastia* ovvero l'indirizzo alla pagina Wikipedia corrispondente.

nuovaRiga(..) istanzia una nuova riga dai parametri passati al metodo e la aggiunge alla lista *righe*. Dalla riga di ogni imperatore salvo i dati riguardanti il nome e l'url della pagina.

getRiga(String nomeImperatore) cerca la riga di uno specifico imperatore cercandolo con il nome passato per argomento.

getRiga(int numeroRiga) ritorna la riga della tabella desiderata. Il metodo è stato pensato per lavorare sulle numerazioni "reali" e non quelle "informatiche" cioè la prima riga è la numero 1 e non 0.

addRiga(..) aggiunge una riga creata esternamente.

class TabellaDinastie extends TabellaWikipedia <TabellaDinastie.Riga> implements Iterable
- nomeDinastia : String - urlDinastia : String - righe : List<Riga>
+ TabellaDinastie(String nomeDinastia)
+ setUrlDinastia(String urlDinastia) : void + getUrlDinastia() : String + getNomeDinastia() : String + nuovaRiga(String nomeImperatore, String urlPagina) : void + getRiga(String nomeImperatore) : Riga + getRiga(int numeroRiga) : Riga + addRiga(RigaTabella riga) : void + getRighe() : List<Riga> + iterator() : Iterator<Riga> + toString() : String + geUrlImperatori() : List¡String¿

getRighe() ritorna la lista con tutte le righe della tabella.
getUrlImperatori ritorna la lista con tutti gli indirizzi alle pagine Wikipedia degli imperatori della dinastia.

2.3.6 RigaTabella

class Riga implements RigaTabella
- String nomeImperatore - String urlPagina
+ Riga(String nomeImperatore, String urlPagina)
+ getNomeImperatore() : String + getUrlPagina() : String + toString() : String

La classe rappresenta una riga della tabella presente all'interno della pagina degli imperatori romani. Le informazioni salvate sono quelle contenute nella seconda colonna, ovvero il nome dell'imperatore romano e l'url della sua pagina Wikipedia.

I campi ed i metodi fanno riferimento solamente a queste due informazioni.

2.3.7 ParserTabellaDinastie

class ParserTabellaDinastie
- int LUNGHEZZA_TAG - string GUERRA_CIVILE_1
+ analisiSorgente(String sorgente) : List<TabellaDinastie> - analisiHtmlTabella(String sorgente) : TabellaDinastie -dataDinastia(String riga) : String - cercaLink(String sorgente) : String - cercaImperatori(String htmlTabella, TabellaDinastie tabellaDinastia) : void - analisiRiga(String) : Record

La classe è un parser scritto per lavorare sulla wiki degli imperatori romani. *LUNGHEZZA_TAG* è la lunghezza del tag di fine riga *GUERRA_CIVILE_1* è l'url agli imperatori di quel periodo che viene utilizzato in seguito come segnalino per non perdere la dinastia dei Flavi che ha una formattazione diversa rispetto alle altre. C'è una piccola classe interna *Record* che è utilizzata per il salvataggio momentaneo dei dati prelevati da una riga per avere maggior ordine durante l'esecuzione degli algoritmi.

analisiSorgente(..) è il metodo che si occupa di ricevere il sorgente della wiki. Al suo interno viene fatta la divisione tra le varie tabelle della pagina ed inviata la porzione di sorgente ai metodi specializzati. Nel metodo vengono salvate le varie tabelle in una lista che viene ritornata dal metodo

analisiHtmlTabella(..) si occupa di smistare il codice agli altri metodi e di istanziare la TabellaDinastie da riempire con le informazioni.

dataDinastia(..) recupera dalla riga sopra la tabella il periodo storico chiuso tra le parentesi. Nel metodo viene chiamato il metodo testoEsternoTag(..) della classe ParserSinotticoWikipedia del package wikipediaWebScraperLib per recuperare alcune informazioni.

cercaLink(..) cerca il tag html per i link, taglia quella parte e la invia al metodo statico di ParserSinotticoWikipedia per il recupero del dato.

cercaImperatori(..) è il metodo che si occupa di sezionare riga per riga la tabella con le informazioni inviandole ad *analisiRiga(..)* per l'analisi. Costruisce le rige che popoleranno la tabella finale.

analisiRiga(..) è il metodo che nello specifico elabora la singola riga estraendone il nome dell'imperatore e l'url corrispondente ritornando a *cercaImperatori(..)* i dati raccolti.

2.3.8 Record

class Record
- String nomeImperatore - String urlImperatore
+ Record(String, String) + getNome() : String + getUrl() : String

Record è una piccola classe interna a ParserTabellaDinastie creata con l'unico scopo per avere un codice più chiaro nella fase di scraping. Viene utilizzata per salvare il nome dell'imperatore e il suo url.

2.3.9 Model

class Model
paginaImperatori : WikiImperatoriRomaniPagina
startModel() : void getListaDinastie() : String[] alberoGenealogicoDinastia(String nomeDinastia) : List<AlberoGenealogico>

La classe Model è il punto di accesso ai dati.

Nell'attributo *paginaImperatori* viene salvato il riferimento all'istanza di WikiImperatoriRomaniPagina.

startModel() istanzia WikiImperatoriRomaniPagina.

getListaDinastie() ritorna la lista con i nomi delle dinastie degli imperatori.

alberoGenealogicoDinastia(..) ritorna la lista di alberi genealogici richiesta tramite il nome passato da parametro.

2.3.10 CostruisciAlberoGenealogico

class CostruisciAlberoGenealogico
-nomeDinastia : String -webSurfer : WikipediaNavigator -dinastia : TabellaDinastie -urlImperatori : LinkedHashMap<String, Integer> -alberiGenealogici : List<AlberoGenealogico> -paroleVietate : List<String> -pagineWikiErrate : List<String> -GIULIO_CESARE : String -GIULIO_CLAUDIA : String -CLAUDIO : String -IMPERATORI_ADOTTIVI : String -IMPERATORE_PIO : String -FAUSTINA_MINORE : String
+CostruisciAlberoGenealogico(String nomeDinastia)
+init() : void +getAlberiGenealogici() : List<AlberoGenealogico> -urlImperatori(List<String>listaUrl) : void -costruisciAlberi() : void -costruzioneAlberoRicorsiva(String urlWikipedia, AlberoGenealogico albero) -costruisciPaginaWikipedia(String sorgente) : PaginaWikipedia -costruisciRomano(PaginaWikipedia wiki) : AnticoRomano -costruisciImperatore(PaginaWikipedia wiki) : Imperatore -cercaFigli(PaginaWikipedia wiki) : List<Informazione>

La si occupa di costruire gli alberi genealogici di ogni dinastia/periodo storico di tutti gli imperatori romani. Sono inseriti i casi particolari che possono alterare il risultato finale rendendolo non perfettamente congruente o poco chiaro.

Per la costruzione dell'albero con la classe AlberoGenealogico viene utilizzato il polimorfismo aggiungendo oggetti di tipo AnticoRomano estensione di Persona, il tipo accettato da AlberoGenealogico.

nomeDinastia salva il nome della dinastia a cui corrispondono gli alberi genealogici.

webSurfer per la classe di WikipediaNavigator.

dinastia per la TabellaDinastie dove sono elencati tutti gli imperatori.

urlImerpatori una mappa dove sono contenuti tutti gli imperatori ed un Integer per segnalare se l'imperatore è stato visitato o meno.

alberiGenealogici la lista che contiene gli alberi genealogici generati.

paroleVietate la lista con le parole dei casi particolari che non devono apparire negli alberi genealogici.

pagineWikiErrate le pagine Wikipedia errate che appaiono nei risultati

Il resto dei campi sono i casi particolari che vengono controllati durante l'esecuzione.

Il costruttore della classe riceve una stringa con il nome della dinastia che viene ricercato in WikiImperatoriRomaniPagina. Se la dinastia non viene trovata c'è il lancio l'errore DinastiaNonTrovataException. In caso di ricerca andata a buon fine, viene chiamato il metodo *urlImperatori(..)* che crea la mappa *urlImperatori*.

init() inizializza webSurfer con WikipediaNavigator e chiama il metodo *costruisciAlberi()* per iniziare il processo di scraping e costruzione dell'albero genealogico.

costruisciAlberi() cicla la mappa con gli imperatori e per ognuno inizializza l'albero genealogico. Il metodo si occupa di aggiungere l'albero una volta completato alla lista che li raccoglie.

costruzioneAlberoRicorsiva(..) questo metodo si occupa di aprire le wiki di ogni figlio in maniera ricorsiva, popolando l'AlberoGenealogico di oggetti AnticoRomano costruiti di volta in volta. Il processo termina quando non ci sono più pagine da visitare.

costruisciPaginaWikipedia(..) riceve il sorgente della pagina Wikipedia dal metodo *costruzioneAlberoRicorsiva* e ritorna l'istanza di PaginaWikipedia ricavata. Vengono utilizzati il builder ed i metodi di AnalisiInfoWikipedia del package wikipediaWebScraperLib.

costruisciRomano(..) e *costruisciImperatore* dalla PaginaWikipedia passata la corrispondente classe da inserire nell'albero genealogico.

cercaFigli(..) dalla PaginaWikipedia consulta il sinottico e ritorna la lista di Informazione corrispondenti ai figli.

2.4 iRomaniView

Come negli altri package viene sempre rispettato il principio solid di singola responsabilità; nella classe **componenteGrafoIllustrato** viene messo in pratica il principio di Liskov substitution avendo utilizzato l'interfaccia Persona per le operazioni e quindi anche il polimorfismo.

La gui è divisa in 3 sezioni: quella dove vengono mostrati gli alberi (PanelPrincipale con PanelAlberiGenealogici), il menù dove si possono scegliere le dinastie (PanelMenuDinastie) e quello dove è possibile visualizzare le informazioni di riepilogo sui personaggi storici (PanelFinestraInfoBox). Queste classi panel sono estensioni di JPanel per permettere un'utilizzo diretto con JFrame. In ogni panel sono state inserite delle costanti per altezza, larghezza, posizione verticale ed orizzontale per organizzare in maniera più ordinata il posizionamento all'interno della gui.

2.4.1 ComponenteGrafoIllustrato

class ComponenteGrafoIllustrato
-component : mxGraphComponent -STYLE_IMPERATORE : String -STYLE_ROMANO : String -STYLE_ARCHI : String -STYLE_ARCHI_IMP : String
+ComponenteGrafoIllustrato(AlberoGenealogico albero)
+init(JGraphXAdapter <Persona, Archi>wrapGrafo, Graph<Persona, Archi>grafo) : void +getComponente() : mxGraphComponent -coloraVerticiGrafo(Map<Persona, mxICell>vertici, JGraphXAdapter <Persona, Archi>wrapGrafo) : void -coloraArchiGrafo(Map<Persona, mxICell>archi, JGraphXAdapter <Persona, Archi>wrapGrafo Graph<Persona, Archi>grafo) : void -settaComponente(JGraphXAdapter <Persona, Archi>wrapGrafo) : void

La classe è quella che si occupa di costruire l'oggetto in cui verranno visualizzate le dinastie degli imperatori romani. Per questa classe ho utilizzato la libreria JGraphT che fornisce la classe *JGraphXAdapter* che estende *mxgraph* che permette la possibilità di avere delle versioni visuali dei grafi creati.

Per la composizione grafica viene fatto uso del polimorfismo per i metodi toString sovrascritti in AnticoRomano, Imperatore ed Archi che viene chiamato per costruire le etichette che verranno apposte sui vertici e sugli archi.

component è il componente grafico di mxgraph che permette la visualizzazione dei grafi da inserire in un JPanel o JFrame della libreria swing.

Gli altri campi sono le scelte di colori e forma delle etichette per differenziare imperatori e altri personaggi storici.

Il costrutture riceve come parametro un AlberoGenealogico del package alberoGenealogicoLib e inizializza il wrapper per il grafo con JGraphXAdapter che permette al grafo creato di comunicare con la parte per la grafica.

init() chiamato dal costruttore, si occupa di smistare vertici e archi verso i metodi di colorazione.

coloraVerticiGrafo(..) e *coloraArchiGrafo(..)* si occupano di effettuare le colorazioni controllando se la Persona in ogni vertisce *isinstaceof* Imperatore.

settaComponente(..) riceve il wrapper con cui istanziare il mxGraphComponent, ovvero l'oggetto che potrà essere usato con la libreria swing per avere la visualizzazione degli alberi. In questo metodo vengono anche scelte le impostazioni grafiche del componente.

getComponent() ritorna il componente con gli alberi disegnati.

2.4.2 PanelAlberiGenealogici

class PanelAlberiGenealogici extends JTabbedPane
-alberi : List<AlberoGenealogico> -LARGHEZZA : int -ALTEZZA : int -X : int -Y : int
+PanelAlberiGenealogici(List<AlberoGenealogico>alberi)
+costruisciPannello() : void -istanziaAlberiVisuali() : List<mxGraphComponent> -buildPannello(List<mxGraphComponent>grafi) : void

La classe raccoglie gli oggetti di tipo ComponenteGrafoIllustrato, essendo un'estensione di JTabbedPane è stata usata per creare un tab per ogni albero genealogico presente in una dinastia ed avere un risultato completo.

alberi e la lista degli alberi genealogici ricevuti nel costruttore

costruisciPannello chiama il metodo *istanziaAlberiVisuali* e *buildPannello* che sono quelli deputati alle operazioni di realizzazione della classe.

istanziaAlberiVisuali si occupa di creare la lista con gli mxGraphComponent di ogni albero istanziando oggietti di tipo ComponenteGrafoIllustrato.

buildPannello ricevuta la lista di componenti dal metodo precedente, per ognuno di questi crea un tab.

2.4.3 PanelPrincipale

class PanelPrincipale extends JPanel
-ALTEZZA : int -LARGHEZZA : int -X : int -Y : int
+PanelPrincipale()
+disegnaAlbero(PanelAlberiGenealogici alberiGenealogici) : void

La classe estende JPanel ed è il pannello della finestra mostrerà il *PanelAlberiGenealogici* dove sono rappresentati gli alberi. Per un corretto funzionamento ogni volta che vengono richiesti gli alberi genealogici di una dinastia viene creato un nuovo PanelAlberiGenealogici che viene sostituito con quello precedente, in questo modo non è necessario dover modificare tutto il JFrame ma solo questo componente.

disegnaAlbero(..) riceve il pannello con gli alberi e lo mostra.

2.4.4 PanelMenuDinastie

class PanelMenuDinastie extends JPanel
-boxListaDinastie : JComboBox<String> -labelScegli : JLabel -buttonSeleziona : JButton -ALTEZZA : int -LARGHEZZA : int -X : int -Y : int
+PanelMenuDinastie()
-boxListaDinastie() : void -buttonSeleziona() : void -labelScegli() : void +setDinastie(String[]) : void +getButtonSeleziona() : JButton +getBoxListaDinastie() : JComboBox<String>

La classe è il pannello in alto a destra della gui in cui è possibile selezionare la dinastia di cui si vuole avere la rappresentazione dell'albero genealogico.

boxListaDinastie è il menù a tendina dove è possibile scegliere la dinastia; *labelScegli* l'etichetta subito sopra e *buttonSeleziona* il pulsante al di sotto.

Il costruttore si occupa di istanziare tutti i componenti e di posizionarli con il *GridBagLayout*.

I metodi privati *boxListaDinastie*, *buttonSeleziona* e *labelScegli* sono i metodi che istanziano i rispettivi omonimi negli attributi della classe.

setDinastie riceve l'array di stringhe contenente le dinastie degli imperatori per popolare il boxListaDinastie.

I get ritornano i rispettivi bottone e menù.

2.4.5 PanelFinestraInfoBox

class PanelFinestraInfoBox extends JPanel
-LARGHEZZA : int -ALTEZZA : int -X : int -Y : int -box : JComboBox<PaginaWikipedia> -seleziona : JButton -contenitoreTopPagina : JPanel -scrollPaneSinottico : JScrollPane -testoSinottico : JTextPane
+PanelFinestraInfoBox()
-box() : void -private void seleziona() : void -contenitoreTopPagina() : void -testoSinottico() : void -scrollPaneSinottico() : void +costruisciMenuTendina(List<AlberoGenealogico>alberi) : void -istanziaPannelloSinottico(String htmlSinotticox) : void +costruisciPannelloInfo(PaginaWikipedia pagina) : void +getMenuTendina() : JComboBox +getBottoneSeleziona() : JButton

La classe è il pannello basso a destra della gui dove è possibile avere il

riassunto del personaggio storico selezionato. L'idea nasce dalla conseguenza di avere molti dati presi da Wikipedia e poterli riutilizzare. Infatti il testo contenuto è il sorgente del sinottico salvato in PaginaWikipedia.

Il menù a tendina è un JComboBox popolato con oggetti di tipo PaginaWikipedia, il nome viene di ogni persona visualizzabile è preso dal metodo toString in override per sfruttare il polimorfismo. Nel menù vengono inserite solo le pagine in cui il campo Sinottico non sia null, altrimenti non ci sarebbero informazioni da mostrare.

box il menù a tendina dove vengono mostrati i nomi.

seleziona il bottone seleziona sotto il menù a tendina

contenitoreTopPagina il JPanel che contiene menù a tendina e bottone.

scrollPaneSinottico un pannello con la barra di scroll dove viene inserito il *testoSinottico* ovvero il JTextPane che conterrà il testo html del sinottico.

Il costruttore chiama i metodi con i nomi omonimi ai campi per inizializzarli.

costruisciMenuTendia(..) riceve la lista con gli alberi genealogici di una dinastia e popola il menu a tendina *box* con le PaginaWikipedia di ognuno, sinottico permettendo.

costruisciPannelloInfo(..) dalla pagina wikipedia ricevuta chiama il metodo *istanziaPannelloSinottico* passandogli il sorgente del sinottico.

istanziaPannelloSinottico(..) è il metodo che si occupa di mostrare nel panel *testoSinottico* il sinottico. Chiama i metodi statici di ModificaHtmlSinottico per pulire il sorgente ed avere nella visualizzazione un'impaginazione ottimale.

getMenuTendina() e *getBottoneSeleziona()* forniscono i rispettivi componenti.

2.4.6 ModificaHtmlSinottico

class ModificaHtmlSinottico
+puliziaSinottico(String htmlSinottico) : String
+rimuoviLink(String htmlSinottico) : String
+riposizionaSinottico(String htmlSinottico) : String
+rimuoviAnnotazioni(String htmlSinottico) : String

La classe nasce dalla necessità di dover ripulire il sorgente del sinottico da link ed annotazioni, per avere un effetto migliore nella visualizzazione del testo; il testo che viene rappresentato è il codice html del sinottico che spesso contiene link a pagine interne a Wikipedia. La classe è composta di soli metodi statici.

puliziaSinottico(..) è un metodo che si occupa di ricevere il sorgente e poi di smistarlo agli altri metodi.

rimuoviLink(..) si occupa di rimuovere tutti i tag href dal testo.

riposizionaSinottico riposiziona l'inizio del sorgente se la prima riga del sinottico non è nome.

rimuoviAnnotazioni cancella le parentesi quadre che nel sinottico contengono i riferimenti alle citazioni.

2.4.7 View

class View
-frame : JFrame -panelMenuDinastie : PanelMenuDinastie -panelPrincipale : PanelPrincipale -panelInfoBox : PanelFinestraInfoBox -selezioneIniziale : int
View()
-messaggioAlert() : void +getSelezioneIniziale() : int +tabPanelAlberi(List<AlberoGenealogico>) : void +getJFrame() : JFrame +getPanelMenuDinastie() : PanelMenuDinastie +getPanelInfoBox() : PanelFinestraInfoBox

La classe View è quella che permette la costruzione e l'accesso ai vari componenti della gui. I campi della classe sono il JFrame ed i tre panel che compongono l'interfaccia.

selezioneIniziale è il valore di ritorno del messaggio di avviso quando si apre il programma.

Il costruttore della classe istanzia il JFrame e tutti i settaggi grafici e di misure, inoltre costruisce tutti i panel che vengono aggiunti al JFrame; infine chiama il metodo *messaggioAlert()* che mostrerà il messaggio di avviso con JOptionPane.showConfirmDialog(...). Il valore del bottone premuto sarà salvato in *selezioneIniziale* ed è possibile chiederne il valore con il metodo *getSelezioneIniziale()*

Il metodo *tabPanelAlberi(..)* si occupa di creare il PanelAlberiGenealogici (il JPanel con i tab dove vengono mostrati gli alberi genealogici) passando la lista degli alberi ricevuti e lo passa al corretto componente che li deve contenere ovvero *panelPrincipale*.

Sono poi disponibili i get per i vari componenti della gui.

2.5 iRomaniController

In questo package c'è la classe controller che permette all'utente di utilizzare il programma; presente anche la classe main per lanciare il progetto svolto.

2.5.1 Controller

class Controller
-model : Model -view : View -infoBoxAttivo : boolean
+Controller(Model model, View view)
+init() : void -initController() : void -dinastiaSelezionata() : void -costruisciInfoBox(List<AlberoGenealogico>alberi) : void -aggiornaInfoBox() : void -attivaPulsanteInfoBox() : void

Classe controller che unisce la parte dei dati all'interfaccia quindi iRomaniModel con iRomaniView.

model è il campo in cui viene salvata la classe Model.

view è il campo in cui viene salvata la classe View.

Il costruttore riceve come parametri model e view.

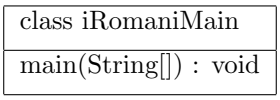
init è il metodo che inizializza il programma. Per primo viene visualizzato il messaggio dal metodo *messaggioAlert* della clase View, se l'utente premerà il pulsante "si" verrà chiamato il metodo privato *initController* che si occuperà di chiamare da Model il metodo *getListaDinastie* che verrà poi passato al *PanelMenuDinastie* tramite la classe View per riempiera il menù a tendina dove appariranno i nomi delle dinastie. Viene anche aggiunto l'action listener che chiama il metodo *dinastiaSelezionata* al bottone subito sotto il menù a tendina.

Il metodo *dinastiaSelezionata* viene chiamato dall'action listener del pulsante, controlla la dinastia selezionata e ne passa il nome al metodo del Model *alberoGenealogicoDinastia* ricevendo gli alberi genealogici. Gli alberi vengono poi passati alla classe *tabPanAlberi* che si occupa della visualizzazione grafica.

costruisciInfoBox ricevendo gli alberi genealogici dal metodo precedente popola il menù a tendina sottostante a quello delle dinastie con gli oggetti PaginaWikipedia degli AnticoRomano. Controlla se il pulsante del pannello è attivo altrimenti chiama il metodo *attivaPulsanteInfoBox* che aggiunge l'action listener sul metodo *aggiornaInfoBox*.

aggiornaInfoBox aggiorna il pannello testuale con le informazione contenute nel Sinottico della PaginaWikipedia selezionata.

2.5.2 iRomaniMain



La classe costruisce il Controller e lo attiva facendo partire l'interfaccia del programma.

3 Descrizione delle funzionalità

Il modus operandi è stato sempre quello prima di progettare le classi in maniera "analogica", iniziando sempre su carta per capire quante e di quali classi avrei avuto bisogno per poi scrivere l'interfaccia pubblica ed in seguito riportare tutto su Eclipse. Anche i vari algoritmi scritti sono sempre stati progettati prima su carta e poi riportati; l'ideazione e scrittura su carta hanno richiesto qualche giorno, soprattutto per la parte di analisi di Wikipedia, ma in compenso i codici hanno funzionato quasi subito, gli errori sono stati minimi e facili da individuare, tendenzialmente mancava solo un ++ per risolvere la situazione.

3.1 Costruzione PaginaWikipedia (wikipediaWebScraperLib package)

3.1.1 Salvataggio dei dati

Il punto di partenza è stato quella di creare una cassetta degli attrezzi che avrei utilizzato in seguito per puntarla sulle pagine Wikipedia degli imperatori romani. I sorgenti delle pagine Wikipedia sono simili ma non sempre ugali negli elementi presenti, l'unica costante è il box laterale della pagina (il sinottico) abbastanza simile in tutte le pagine. Lavorando da solo per un progetto così grande ho deciso subito di incentrare il lavoro sull'analisi di questo elemento.

Ho ideato quindi una classe PaginaWikipedia che potesse modellare le pagine wiki che possiedono un sinottico al loro interno. La classe è un'estensione di WikipediaWebPage, astratta, da cui partire per creare nuovi tipi di pagine.

Modellare il sinottico è stato un lavoro articolato, erano da evitare possibili utilizzi impropri della classe ed ho quindi considerato il sinottico come una classe contenitore, con metodi di accesso e inserimento dei dati. Ho poi creato una classe RigaSinottico che avrebbe rappresentato la singola riga e per non perderne l'ordinamento ho utilizzato una lista per raccoglierle. Per identificare una riga dalle altre, dispongono dell'attributo *categoria* di tipo stringa, da passare nel loro costruttore e che corrisponde alla cella sinistra del sinottico. L'idea era quella di riuscire a mantenere il più possibile la fedeltà con l'oggetto reale e quindi di avere anche un ordinamento interno il più fedele possibile.

Per la scelta della rappresentazione delle informazioni contenute nella cella destra della riga l'intenzione era sempre quella di mantenerne l'ordinamento originale; il tipo di dato presente può essere misto: solo testo o testo con url ed inizialmente l'idea era stata di utilizzare una mappa in combinazione con un altra Collection, però ho poi deciso creare una classe interna apposita per questo compito, per poter avere dei metodi più chiari da utilizzare ed evitare una costruzione poco chiara del comparto. La classe creata per lo scopo è Informazione ed ha due campi, *nomeInfo* ovvero quello che una persona può leggere dal sinottico e *url* per un evenutale link associato. La creazione di una nuova informazione può essere fatta solo da RigaSinottico tramite il metodo *addInformazione*, in modo da evitare la possibilità di inserimenti in righe sbagliate.

La struttura contiene le informazioni è dunque così composta:

1. PaginaWikipedia, il livello superiore con i metodi di accesso.
2. Sinottico, che rappresenta la tabella laterale con i propri metodi di accesso.
3. RigaSinottico, identificabile per il nome categoria preso dalla cella sinistra della riga del sinottico
4. Informazione, conservate mediante una lista in RigaSinottico, utilizzata per i dati nella cella destra della riga; ad una singola informazione corrisponderà un oggetto Informazione.

3.1.2 Prelievo, recupero e salvataggio dei dati

Selenium è stato utilizzato all'interno della classe WikipediaNavigator incapsulandone le funzionalità necessarie; principalmente è stata usata per ricavare il sorgente dalla pagina Wikipedia che gli viene passata dal metodo *getHtmlPagina*. Le funzionalità della classe sono minime, la scelta è dovuta dal fatto di lavorare sul sorgente della pagina e non con gli XPath ed esclusivamente su Wikipedia, per avere un set di classi con funzioni generali ma con un obiettivo preciso. La scelta del sorgente piuttosto che degli XPath è stata una diretta conseguenza del fatto di voler affrontare il problema partendo dal punto più lontano e generico e creare delle classi che non fossero troppo specializzate; passare un link di Wikipedia ed andare in ricerca di punti precisi nella pagine avrebbe reso il prodotto finale meno modulare ed estensibile rispetto alle potenzialità di lavorare direttamente sul sorgente.

Decidendo di approfondire i design pattern per integrarli all'interno del progetto, mi è sembrato interessante sviluppare il builder per costruire le pagine di Wikipedia, per avere un tool extra da poter utilizzare. La classe è *PaginaWikipediaBuilder*. Il suo utilizzo può essere integrato con la classe di utilità *AnalisiInfoWikipedia* che contiene dei piccoli algoritmi per ricavare le informazioni dal sorgente scaricato con WikipediaNavigator. Questa classe offre anche un punto di accesso verso *ParserSinotticoWikipedia*; in questo modo ho riunito in una sola classe tutti i tool necessari all'analisi dei sorgenti.

Per la progettazione di ParserSinottico ho scelto di renderla specializzata solo per il sinottico e non per tutto il codice per rispettare il principio SOLID di singola responsabilità, deve quindi ricevere solo quella parte di codice. Questo "sezionamento" può essere fatto dal metodo *estraiSinottico(..)* di AnalisiInfoWikipedia.

ParserSinottico è suddivisa in metodi ognuno deputato ad uno specifico compito. Questa suddivisione è stata utile nella fase di debugging degli algoritmi scritti all'interno. La progettazione della classe ha richiesto diverso tempo ed è proceduta per step; una prima fase, dopo aver approfondito l'html, è stata quella di cercare dei pattern dentro al codice del sinottico. Il sinottico è una tabella e quindi è divisa in rige con i tag <tr>, </tr>. Dopo questa prima fase sono andato a cercare le righe che contenevano le informazioni che volevo prelevare e queste erano delimitate dai tag <th> e <td>con le rispettive chiusure, uno per la cella sinistra e l'altro per la destra della riga.

Passo successivo è stato quello di capire nel codice delle celle cosa accadeva. I tag erano molteplici però quello di cui avevo bisogno era quel testo che si può leggere dalla pagina Wikipedia e se a quel testo era associato un link, di prendere anche questo. Quindi il primo passo è stato di cominciare con prendere il testo dalla cella sinistra; creo quindi un metodo che mi taglia il sinottico in righe (*createSinottico*) e poi un altro metodo per prendere il testo fuori da tutti i tag html che sono principalmente di formattazione. Scrivo quindi un metodo *estraiTestoEsternoTag(..)* che riceve il codice di una riga del sinottico, ed in maniera ricorsiva scarta tutti i tag html fino a quando non trova il testo al di fuori di questi.

Dopo passo all'analisi della cella. Procedo in maniera simile a quanto fatto in precedenza usando degli indici per scorrere i caratteri della stringa per distinguere tra tag e testo in chiaro. Per i tag ho bisogno di quelli che contengano un link, scrivo quindi un metodo *internoTag(..)* che parsa la riga fino a trovare la fine del tag e che controlli se è quello di un link o no, ed un metodo *estraiLink(..)* che si occupi di prelevarlo. Per il testo fuori da un link il metodo *esternoTag(..)*, che richiama *estraiTestoEsternoTag(..)* creato in precedenza, che controlli che il testo ricavato sia valido o non sia una sporcatura per il risultato finale.

Tutti questi metodi vanno a costruire la RigaSinottico che viene ritornata al metodo createSinottico per aggiungerla al Sinottico ed il giro si ripete per ogni riga della tabella presente nel sorgente.

I passaggi che vengono compiuti per la cattura delle informazioni dalla pagina Wikipedia sono dunque:

1. WikipediaNavigator, per l'apertura del browser e recupero delle informazioni.

2. PaginaWikipediaBuilder, per processare le operazioni (ma non necessario) e creare PaginaWikipedia.

3. AnalizzaInfoSinottico, che ricevendo il sorgente recupera le informazioni.

4. ParserSinotticoWikipedia parsa il sorgente e crea un oggetto Sinottico con le RigheSinottico che contengono le Informazioni, dove vengono salvate le informazioni recuperate.

5. PaginaWikipedia, che raccoglierà al suo interno le informazioni eleaborate da AnalizzaInfoSinottico ed il Sinottico creato con il parser.

Il risultato ottenuto sarà questo (screen del risultato di TesterPaginaWikipediaBuilderEInfoWikipedia).

```
Titolo pagina: Gaio Giulio Cesare
Url pagina: https://it.wikipedia.org/wiki/Gaio_Giulio_Cesare
Url immagine: https://upload.wikimedia.org/wikipedia/commons/thumb/8/8f/Gaius_Iulius

Stampa del sinottico di Giulio Cesare
[ Categoria informazione: Nome originale ]
Info: Gaius Iulius Caesar | Url: null

[ Categoria informazione: Titoli ]
Info: Pater Patriae | Url: https://it.wikipedia.org/wiki/Pater_Patriae
Info: Divus Iulius | Url: null

[ Categoria informazione: Nascita ]
Info: 13 luglio | Url: null
Info: 101 a.C. | Url: https://it.wikipedia.org/wiki/101_a.C.
Info: o 12 luglio | Url: null
Info: 100 a.C. | Url: https://it.wikipedia.org/wiki/100_a.C.
Info: Roma | Url: https://it.wikipedia.org/wiki/Roma_(citt%C3%A0_antica)

[ Categoria informazione: Morte ]
Info: 15 marzo | Url: null
Info: 44 a.C. | Url: https://it.wikipedia.org/wiki/44_a.C.
Info: Roma | Url: https://it.wikipedia.org/wiki/Roma_(citt%C3%A0_antica)

[ Categoria informazione: Coniuge ]
Info: Cossuzia | Url: https://it.wikipedia.org/wiki/Cossuzia
Info: Cornelia | Url: https://it.wikipedia.org/wiki/Cornelia_(moglie_di_Cesare)
Info: Pompea Silla | Url: https://it.wikipedia.org/wiki/Pompea_Silla
Info: Calpurnia | Url: https://it.wikipedia.org/wiki/Calpurnia

[ Categoria informazione: Figli ]
Info: Giulia | Url: https://it.wikipedia.org/wiki/Giulia_(figlia_di_Cesare)
Info: Cesarione | Url: https://it.wikipedia.org/wiki/Tolomeo_XV
Info: Ottaviano | Url: https://it.wikipedia.org/wiki/Augusto

[ Categoria informazione: Gens ]
Info: Iulia | Url: https://it.wikipedia.org/wiki/Gens_Iulia

[ Categoria informazione: Padre ]
Info: Gaio Giulio Cesare | Url: https://it.wikipedia.org/wiki/Gaio_Giulio_Cesare_(padre_di

[ Categoria informazione: Madre ]
Info: Aurelia Cotta | Url: https://it.wikipedia.org/wiki/Aurelia_Cotta

[ Categoria informazione: Questura ]
Info: 69 a.C. | Url: https://it.wikipedia.org/wiki/69_a.C.

[ Categoria informazione: Edilità ]
```

(a) Parziale del risultato dello scraping

Nome originale	*Gaius Iulius Caesar*
Titoli	*Pater Patriae*, *Divus Iulius* (dopo la morte)
Nascita	13 luglio 101 a.C.[1] o 12 luglio 100 a.C.[2] Roma
Morte	15 marzo 44 a.C. Roma
Coniuge	Cossuzia (86-84 a.C.)?[N 1] Cornelia (83-68 a.C.) Pompea Silla (68-62 a.C.) Calpurnia (59-44 a.C.)
Figli	Giulia (da Cornelia); Cesarione (da Cleopatra); Ottaviano (pronipote adottato)
Gens	Iulia
Padre	Gaio Giulio Cesare
Madre	Aurelia Cotta
Questura	69 a.C.
Edilità	65 a.C.
Pretura	62 a.C.
Propretura	61 a.C. nella Spagna ulteriore
Consolato	59 a.C.; 48 a.C.; 46 a.C.; 45 a.C.; 44 a.C.
Proconsolato	58-50 a.C. nelle Gallie
Dittatura	49-44 a.C.
Pontificato max	63-44 a.C.

(b) Sinottico da Wikipedia

3.2 Alberi genealogici (alberoGenealogicoLib)

Successivamente ho puntato a creare la struttura per ottenere gli alberi genealogici. Come fatto per la parte dello scraping, anche questa parte doveva essere indipendente dal resto del progetto per essere modulare ed estensibile; ho quindi rimosso dalla mente quanto già fatto e creato un nuovo package. Il primo passo è stato quello di creare l'interfaccia *Persona* da utilizzare con la struttura dati da creare ed utilizzare quindi il polimorfismo per le operazioni necessarie.

Secondo step la struttura dati. Per modellare la classe ho usato la libreria JGraphT che permette di creare grafi in java. Ho quindi incapsulato i metodi necessari all'interno della classe *AlberoGenealogico* per poterlo popolare. Inizialmente la classe era pensata per avere oltre le relazioni tra padre e figlio anche quella tra coniugi, il problema si è posto quando arrivato al punto di progettare

la parte per visualizzare gli alberi: non avendo ancora studiato i grafi durante questo primo anno ho approfondito in autonomia l'argomento, trovando alcune difficoltà, ho deciso allora di sfruttare maggiormente la libreria JGraphT, che porta con se JgraphX ed un wrapper per poter far lavorare queste due insieme e permettere la visualizzazione dei grafi. Jgraphx (com.mxgraph all'interno del codice) è un progetto che è stato dismesso, il forum ufficiale dove venivano poste le domande è stato chiuso, la documentazione in molti punti è un pò criptica ed anche non aggiornata in altri ed in rete non ci sono molte risorse utili per rispondere ai mille quesiti. Nonostante venga fornito tutto con JGraphT, questi forniscono solo uno snippet di codice per mostrare l'integrazione.

L'unico modo che ho avuto per far fare alla libreria quello che volevo è stato girare tra i sorgenti per capire come funzionavano i vari componenti che avrei voluto utilizzare. Sono partito quindi disegnando prima gli imperatori ed i loro figli e così via, e per ottenere un risultato che poteva essere accettabile ho impiegato diversi giorni cercando di capire il funzionamento del tutto. Per aggiungere i coniugi mi sarei dovuto inoltrare nelle profondità dei sorgenti contenenti gli algoritmi (che sono molti e molto vasti), ma non conoscendo ancora come fare le visite sui grafi avrei dovuto affrontare anzitutto questa questione, lavorando però al progetto da solo ho deciso di procedere solo con le relazioni padre-figlio e concentrare gli sforzi sul migliorare la visualizzazione del grafo il più possibile.

Le operazioni ed i metodi di questa parte di codice sono abbastanza lineari:

1. Estendere la classe Persona.
2. Aggiungere con i metodi forniti da AlberoGenealogico una persona e suo figlio e ripetere fino a necessità.

Per la consultazione dell'albero sono forniti i metodi dalla classe.

3.3 Popolamento dell'albero genealogico

3.3.1 Pagina wiki imperatori romani (iRomaniModel package)

La pagina wiki con le dinastie degli imperatori romani oltre ad essere stata richiesta per le specifiche del progetto è necessaria per recuperare le informazioni sulle dinastie perchè contenendo la lista di tutti gli imperatori offre il punto di accesso per l'inizio di ogni albero. La pagina però è diversa rispetto a quella che avevo modellato in precedenza, ha uno stile diverso con molte tabelle e titoli da recuperare e nessun sinottico. Modello quindi una nuova pagina Wikipedia (*WikiImperatoriRomaniPagina*) estendendo WikipediaWebPage in cui il titolo e l'url sono statici, e per le tabella creo una nuova classe TabellaDinastie in cui verranno salvati gli imperatori romani. Per questa classe estendo la TabellaWikipedia e procedo con una modellazione simile a quella fatta per Sinottico. Creo una riga più semplice rispetto a quella del sinottico, decidendo di prendere le informazioni della secondo colonna, contenenti il nome ed il link alla pagine dell'imperatore, in particolare prendo il primo nome che solitamente è quello che corrisponde alla pagina. Essendo la classe dotata di poche funzionalità e legata con TabellaDinastie, la creo interna a questa e scrivo il metodo

nuovaRiga per permettere l'aggiunta di nuove righe esclusivamente passando dalla classe che le dovrà contenere.

Per popolare le tabelle creo una nuova classe *ParserTabellaDinastie* che dal sorgente dovrà recuperare le informazioni necessarie.

Come fatto in precedenza cerco dei pattern dentro la pagina e trovo che sopra ad ogni tabella c'è il nome della dinastia (o fase storica) tra il tag di titolo <h4>. Il metodo *analisiSorgente* è quello di accesso alla classe che si occupa di dividere il codice nelle tabelle e di raccogliere le TabelleDinastie create durante il processo di analisi.

Come per l'altro parser, la classe è scritta in modo da avere metodi specifici per ogni fase. Il metodo che si occupa dello smistamento del codice con la singola tabella da studiare è *analisiHtmlTabella* che prende prima il nome della dinastia nel tag del titolo, la data del periodo storico ed inseguito invia il sorgente a *cercaImperatori*. *cercaImperatori* si occupa di sezionare la tabella nelle righe; ogni riga viene inviata a *analisiRiga* che con i metodi di ParserSinotticoWikipedia della libreria di scraping recupera dalla secondo colonna della riga il nome dell'imperatore ed il link alla sua pagina. Ho creato una piccola classe Record che salva il nome e l'url per ritornarli al metodo chiamante, in modo da avere un codice più chiaro.

Con questa classe quindi creo una lista di TabelleDinastie, una per ogni dinastia/tabella presente nella pagina. Per la scrittura del codice ho seguito il modus operandi del lavoro fatto in precedenza con il sinottico, l'unico problema avuto è stato con la dinastia dei Severi che ha il nome della dinastia racchiuso con il tag <h3> , la soluzione è stata quella di inserire il caso all'interno del codice: appena si conclude l'analisi della dinastia precedente, ovvero la Guerra civile romana (193-197), invece di tagliare come al solito taglio su h3 e in analisHtmlTabellax quando per tagliare il titolo non trova il tag e quindi l'indice da come risultato -1, in quel caso taglia su h3.

A questo punto devo collegare le tabelle delle dinastie con WikiImperatoriRomaniPagina, decido allora di utilizzare il design pattern del singleton: questa classe rappresenta esattamente una pagina di Wikipedia, istanziarne più di una non sarebbe coerente, inoltre per essere completa bisogna eseguire lo scraping delle tabelle, cosa che richiede l'apertura del browser per recuperare il sorgente e poi la sua analisi. Se fosse necessario avere questa classe in più punti del codice o tra classi diverse ogni volta potrebbero essere fatte tutte queste operazioni, con il singleton invece ho una sola volta la costruzione dell'oggetto evitando tutto il processo successivamente.

Decido però che la classe effettui in autonomia tutte le operazioni necessarie in modo da evitare la possibilità di avere dati incompleti; il metodo *analisiSorgentePagina* tramite un oggetto WikipediaNavigator recupera il sorgente dalla pagina, e poi lo manda a ParserTabellaDinastie che ritornerà la lista con le tabelle delle dinastie.

Le operazioni sono quindi così eseguite:

1. Viene chiamata l'istanza di WikiImperatoriRomaniPagina col metodo statico getInstance().

2. La classe con un oggetto WikipediaNavigator recupera il sorgente dalla corrispondente pagina.

3. Il sorgente viene poi inviato a ParserTabellaDinastie che lo analizza e crea le TabellaDinastia con le informazioni scelte in fase di progettazione.

WikiImperatoriRomaniPagina fornisce i metodi di accesso alle informazioni contenute.

Un estratto delle informazioni contenute nella classe a costruzione completa:

```
Titolo della pagina: Dinastie imperatori romani
Url della pagina: https://it.wikipedia.org/wiki/Imperatori_romani

Elenco delle dinastie e imperatori:

Dinastia: Gens Iulia (49_a.C.-44_a.C.) | Url: https://it.wikipedia.org/wiki/Gens_Iulia
Elenco degli imperatori:
    Cesare | https://it.wikipedia.org/wiki/Gaio_Giulio_Cesare

Dinastia: Dinastia giulio-claudia (27_a.C.-68_d.C.) | Url: https://it.wikipedia.org/wiki/Dinastia_giulio-claudia
Elenco degli imperatori:
    Augusto | https://it.wikipedia.org/wiki/Augusto
    Tiberio | https://it.wikipedia.org/wiki/Tiberio
    Caligola | https://it.wikipedia.org/wiki/Caligola
    Claudio | https://it.wikipedia.org/wiki/Claudio
    Nerone | https://it.wikipedia.org/wiki/Nerone

Dinastia: Guerra civile romana (68-69) | Url: https://it.wikipedia.org/wiki/Guerra_civile_romana_(68-69)
Elenco degli imperatori:
    Galba | https://it.wikipedia.org/wiki/Galba
    Otone | https://it.wikipedia.org/wiki/Otone
    Vitellio | https://it.wikipedia.org/wiki/Vitellio

Dinastia: Dinastia dei Flavi (69-96) | Url: https://it.wikipedia.org/wiki/Dinastia_flavia
Elenco degli imperatori:
    Vespasiano | https://it.wikipedia.org/wiki/Vespasiano
    Tito | https://it.wikipedia.org/wiki/Tito_(imperatore_romano)
    Domiziano | https://it.wikipedia.org/wiki/Domiziano

Dinastia: Imperatori adottivi (96-192) | Url: https://it.wikipedia.org/wiki/Imperatori_adottivi
Elenco degli imperatori:
    Nerva | https://it.wikipedia.org/wiki/Nerva
    Traiano | https://it.wikipedia.org/wiki/Traiano
    Adriano | https://it.wikipedia.org/wiki/Adriano
    Antonino Pio | https://it.wikipedia.org/wiki/Antonino_Pio
    Marco Aurelio | https://it.wikipedia.org/wiki/Marco_Aurelio
    Lucio Vero | https://it.wikipedia.org/wiki/Lucio_Vero
    Commodo | https://it.wikipedia.org/wiki/Commodo

Dinastia: Guerra civile romana (193-197) | Url: https://it.wikipedia.org/wiki/Guerra_civile_romana_(193-197)
Elenco degli imperatori:
<
```

3.3.2 Generazione degli alberi genealogici

Arrivato a questo punto mancavano solo le classi per concretizzare le informazioni ricavata da wikipedia, di trasformare PaginaWikipedia nel personaggio che avrebbero contenuto.

Procedo con la scrittura della classe AnticoRomano, che sarebbe stata la concretizzazione delle informazioni delle wiki e che implementa l'interfaccia Persona

per poter essere inserita nella classe AlberoGenealogico; la estendo poi con Imperatore in cui c'è l'override del metodo toString per avere l'etichetta del grafo disegnata in un modo diverso rispetto ai personaggi storici che non lo sono stati. AnticoRomano implementa tutti i metodi per poter interagire con AlberoGenealogico: JGraphT, utilizzato per AlberoGenealogico, utilizza i set per salvare i dati e quindi per avere un corretto funzionamento ho implementato l'interfaccia Comparable con il metodo compareTo, equals e hashCode.

Per la popolare l'albero infine creo una classe *CostruisciAlberoGenealogico*.

Una problematica che ho incontrato durante il processo è stato quello di avere imperatori in una dinastia che non hanno legami di sangue con gli altri e volendo lavorare cercando di avere dei dati puliti, navigando solo i figli, per non dover in seguito inserire correzioni artificiose per pulire i dati raccolti comunque contenerle, la soluzione è stata quella di creare una mappa in cui le chiavi sono gli url degli imperatori della dinastia ed il valore associato un intero a 0. Passando tra le pagine Wikipedia controllo sempre se mi trovo su di un imperatore cercandone l'url nella mappa, e se c'è la corrispondenza metto l'intero ad 1. In questo modo se alla fine di una ricerca ci sono imperatori a 0 e che quindi non sono stati visitati, possono iniziarne una nuova su loro e costruire così tutti gli alberi genealogici ed avere un risultato completo. Con questo controllo posso anche individuare chi sono gli imperatori ed istanziare Imperatore invece di AnticoRomano.

Nei campi della classe sono inseriti i casi particolari che sporcherebbero il risultato finale.

Nel costruttore riceve il nome di una dinastia e chiede la corrispettiva TabellaDinastia a WikiImperatoriRomaniPagina, lanciando un errore DinastiaNonTrovataException in caso di fallimento nella ricerca. Se viene richiesta la dinastia di Giulio Cesare (gens Iulia) viene aggiunta anche quella Giulio-Claudia: da Giulio Cesare si può scendere fino a Nerone seguendo i figli, però la tabella della dinastia è composta solo da Giulio Cesare e non è possibile individuare gli imperatori per avere una visualizzazione dell'albero completa.

Il metodo *costruisciAlberi()* si occupa di effettuare un ciclo tra gli imperatori che nella mappa hanno l'intero a 0 ed in quel caso chiamare il metodo *costruzioneAlberoRicorsiva(..)*. Al ritorno dal metodo ricorsivo l'imperatore viene inserito nell'albero, altrimenti nei casi in cui non abbia avuto figli non verrebbe mai inserito avendo come risultato nella gui una finestra vuota. Per la mappa ho scelto una LinkedHasMap per mantenere l'ordinamento delle righe della tabella che corrispondono anche ad un ordinamento temporale degli imperatori stessi ed evitare imprevisti nella realizzazione degli alberi genealogici.

In *costruzioneAlberoRicorsiva* viene ricavato il sorgente della pagina Wikipedia dall'url della persona in esame; viene effettuato il controllo se si tratta di un imperatore o meno per istanziare la classe corretta. Viene costruita la PaginaWikipedia di ogni persona effettuando lo scraping sul codice per ricavare le informazioni necessarie e viene richiesta la lista con le informazioni sui figli dal Sinottico. Vengono poi tutti visitati ricorsivamente anche i figli dei figli fino a quando non ci sono più pagine da visitare. Al ritorno della chiamata ricorsiva il figlio viene aggiunto all'albero genealogico insieme al padre, mentre se non era

disponibile il link viene aggiunto immediatamente.

Un altro caso particolare scritto all'interno di questo metodo è quello di Faustina Minore, moglie di Marco Aurelio. Entrambi sono presenti nello stesso albero, in quanto il marito è stato adottato dall'imperatore Antonino Pio, padre di Faustina. Hanno avuto molti figli insieme e per avere un risultato grafico migliore sposto la donna nell'ultimo posto della lista che dove sono raccolti i figli di Antonino Pio per avvicinarla al marito (la dinastia è quella degli imperatori adottivi).

Le operazioni che devono essere svolte per ottenere gli alberi genealogici di una dinastia sono:

1. Passare il nome di una dinastia alla classe CostruisciAlberoGenealogico.
2. La classe chiede a WikiImperatoriPagine di fornire la TabellaDinastia corrispondente.
3. Viene istanziato l'AlberoGenealogico e con WikipediaNavigator vengono prelevati i sorgenti ed analizzati per costruire le PaginaWikipedia corrispondenti.
4. Dalla PaginaWikipedia si costruisce l'AnticoRomano o l'Imperatore.
5. Dal Sinottico di PaginaWikipedia si recupera l'Informazione sui figli.
6. Per tutti quelli che hanno link viene richiamato il processo in maniera ricorsiva, per gli altri viene creata una pagina Wikipedia solo con l'informazione del nome ed aggiunti direttamente all'albero genealogico con il padre.
7. All'uscita delle chiamate ricorsive vengono aggiunti i figli con i padri all'albero.
8. Se sono stati visitati tutti gli imperatori viene ritornata la lista con gli alberi genealogici.

3.4 Visualizzazione degli alberi

Tutte le classi della gui sono nel package iRomaniView. La parte dell'interfaccia viene collegata alla parte dei dati con la classe Controller del package iRomaniController. La prima parte di tutto il lavoro è stato quello di costruire la classe in cui potevano essere visualizzati gli alberi e successivamente ho creato il resto della gui. I componenti principali necessari alle richieste del progetto sono due: la parte in alto a destra dove è possibile scegliere la dinastia, e quella centrale dove vengono mostrati gli alberi genealogici. Una volta creati questi due blocchi sono passato al box della classe PanelInfoBox dove è possibile vedere il sinottico delle persone presenti negli alberi genealogici, una parte extra ideata per sfruttare i dati che riesco a prelevare da Wikipedia.

3.4.1 Disegnare il grafo

La gestione della visualizzazione degli alberi è su due livelli: sopra c'è la classe *PanelAlberiGenealogici* che è un'estensione di JTabbedPane, sotto *ComponenteGrafoIllustrato* che si occupa di creare gli oggetti in cui è possibile avere visualizzazione di un albero partendo da un oggetto AlberoGenealogico che vengono inseriti nei tab del livello superiore.

La scelta di JTabbedPane è dovuta dal fatto che potendoci essere più alberi genealogici per una singola dinastia, con i tab è possibile visionarli tutti.

In ogni tab viene aggiunto un oggetto di tipo *mxGraphComponent* della classe mxGraph, menzionata in precedenza per le difficoltà nell'utilizzo, creati nella classe PanelAlberiGenealogici.

In PanelAlberiGenealogici il primo passaggio è quello della richiesta dell'oggetto Graph con il metodo *getAlbero* dall'albero genealogico passato; in seguito si costruisce un oggetto JGraphXAdapter che è il wrapper fornito da JGraphT per l'integrazione con le classi di mxGraph.

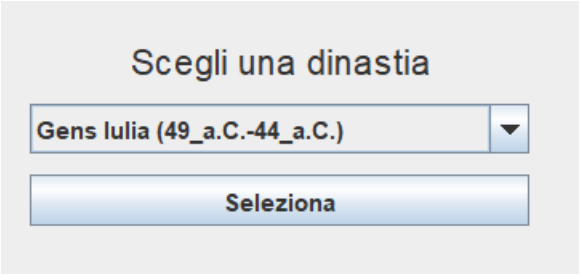
Nel metodo *init()* vengono create due Map: una per i vertici ed i corrispettivi di mxGraph, che sono oggetti di tipo mxICell, e lo stesso per gli archi. Le due mappe vengono poi inviate a due metodi che si occuperanno di impostare lo stile di colore controllando se l'oggetto è un'istanza di Imperatore o no.

Esaurita la parte di colorazione viene creato il componente utilizzabile con la libreria swing nel metodo *settaComponente(..)*, in cui viene scelto il layout grafico e creato l'oggetto con gli alberi visualizzabili. Avendo reimplementato il metodo toString, con il polimorfismo oltre ai colori vengono create delle etichette diverse per imperatori e non.

Il metodo *getComponente()* ritorna alla classe *PanelAlberiGenealogici* il componente creato, che viene salvato in una lista successivamente consultata nel metodo *buildPanel* in cui verranno creati i tab per ogni albero visuale costruito.

3.4.2 La gui

Il primo passo è quello del riempimento del menù a tendina con la lista delle dinastie degli imperatori romani.



Viene quindi creata un'istanza di WikiRomaniImperatoriPagina dal model per avere questi dati. Il controller chiama il metodo *model.getDinastie()* per ricevere la lista degli imperatori da passare alla al pannello che gestisce il menù a ten-

dina attraverso il metodo *view.getPanelMenuDinastie().setDinastie(..)*; viene poi attivato il bottone selezione con l'action listener verso il metodo *dinastiaSelezionata()* del controller.

Una volta selezionata la dinastia, il metodo appena menzionato recupera dal menù a tendina la scelta e passerà il nome della dinastia a *model.alberoGenealogicoDinastia(..)*, che chiamando la classe CostruisciAlberoGenealogico ritornerà la lista con gli alberi genealogici della dinastia scelta.

Avendo chiesto i nomi per il menù a tendina da WikiImperatoriRomaniPagina non ci sono errori nel recuperare succesivamente le dinastie essendo sempre questa, la stessa istanza della classe chiamata da CostruisciAlberoGenealogico per recuperare i dati necessari.

Recuperati gli alberi genealogici, vengono passati al metodo *tabPanelAlberi* della View che si occuperà di creare l'oggetto in cui è possibile visualizzare gli alberi genealogici che verrà aggiunto a *PanelPrincipale*.

PanelPrincipale, classe estensione di JPanel, è stata ideata per riuscire a posizionare in maniera corretta tutta la parte degli alberi all'interno di JFrame. In questo modo tutte le modifiche che sono state necessarie durante la costruzione della gui, tipo posizionamento o repaint dei componenti, è stato più immediato e mirato, evitando di delegare al JFrame a queste operazioni.

Una volta terminata la costruzione della gui ho inserito il messaggio di avviso all'avvio del programma. Questo passaggio è stato ideato successivamente perchè al lancio, la prima cosa che viene effettuata è lo scraping su Wikipedia per costruire la pagina degli imperatori romani, e questo potrebbe essere visto come un comportamento strano per l'utente che non ne è a conoscenza. Introducendo il popup iniziale con il messaggio che comunica questa cosa riesco a far controllare questo comportamento. Quest'azione si trova nella classe View ed è nel metodo *messaggioAlert* che viene lanciato subito dopo la costruzione e visualizzazione del JFrame.

Le fasi dell'esecuzione nell'interfaccia, dall'apertura del programma fino alla visualizzazione di un albero sono:

1. Il Popup iniziale, a cui se si preme "Si" viene chiamato il metodo *initController()* che inizializza l'interfaccia.

2. Viene creata un'istanza di WikiImperatoriRomaniPagina da cui il controller chiede la lista degli imperatori romani.

3. la lista viene inviata al pannello della gui che si occuperà di popolare il menù a tendina.

4. Scelta la dinastie e premuto il bottone seleziona viene chiamata la costruzione degli alberi genealogici al Model.

5. gli alberi vengono passati alla View che istanzia il pannello con gli alberi visuali e li aggiunge al PanelPrincipale.

3.4.3 PanelInfoBox

Il funzionamento di questo componente extra aggiunto al progetto è simile a quello per creare l'oggetto dove vengono visualizzati i dati: tramite la classe View vengono inviati gli alberi genealogici della dinastia trovata ed invece di utilizzare delle stringe come per le dinastie, per riempire il JComboBox del menù a tendina vengono utilizzate direttamente gli oggetti PaginaWikipedia. In questo modo nel momento della selezione viene ricevuto direttamente l'oggetto dove è salvato in un apposito campo il sorgente del sinottico.

Con la classe di utilità *ModificaHtmlSinottico* vengono rimossi link e notazioni per avere un effetto visivo migliore.