Alma Mater Studiorum · Università di
Bologna

facoltà di scienze matematiche, fisiche e naturali
Corso di Laurea Triennale in Informatica

JavaFx e le applicazioni Web

Tesi di Laurea in Programmazione di Internet

Relatore:
Chiar.mo Prof.
Antonio Messina

Presentata da:
Stefania Piras

Sessione I
Anno Accademico 2010/2011

Ai miei genitori
che lo hanno reso possibile...

Introduzione

Questa tesi ha lo scopo principale di fornire una visione generale del linguaggio JavaFX, delle sue funzioni, degli elementi che lo caratterizzano e che lo rendono cosi promettente per la programmazione e la creazione di RIA (Rich Internet Applications). La Sun descrive JavaFx come una nuova piattaforma per scrivere RIA's, di un Runtime Environment per permettere al programmatore di essere indipendente dal dispositivo, quindi avremo un unico linguaggio per scrivere applicazioni per web, desktop o mobile.
Inoltre il lavoro di tesi comprende un progetto nel quale verrà sviluppata un'applicazione per il web in JavaFX, nello specifico un sito web per un'agenzia di viaggi. In questo modo ci sarà la possibilità di descrivere alcune delle funzionalità del linguaggio e di verificare le prestazioni della nuova piattaforma. Per l'esecuzione del progetto, ho usato il plug-in disponibile JavaFx 1.3.1 per NetBeans IDE 6.9.1.
Nel primo capitolo viene descritto il linguaggio JavaFx, si parla delle RIA e vengono analizzate le novità più importanti del linguaggio. Inoltre viene dedicata una parte del capitolo anche ad una breve storia del linguaggio e alla descrizione dei tools di sviluppo.
Nel secondo capitolo si ha una panoramica sul linguaggio JavaFx Script, dove vengono descritti i principali mezzi per costruire un applicazione in JavaFx e un'accattivante User Interface. Nella seconda parte del capitolo troviamo una dettagliata descrizione di come creare animazioni e importare un Database nell'applicazione.
Il terzo capitolo descrive il progetto svolto, analizzando tutte le principali fasi di sviluppo dell'applicazione, quindi nel dettaglio, di come viene svolta a livello pratico la programmazione con NetBeans, e di come vengono affrontate le scelte di implementazione durante tutta la durata dello svolgimento del progetto. In particolare viene descritta la realizzazione del JavaDataBase fatto con MySql, la realizzazione delle animazioni e l'integrazione dell'applicazione in un sito Web.

Nel quarto capitolo vengono analizzate le prestazioni dell'applicazione e presentate le conclusioni.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Il linguaggio JavaFx

1.1 Cos'è JavaFX

JavaFx[1] è un software applicativo basato sulla piattaforma Java, adatto sia ad applicazioni stand-alone, sia alle Rich Internet Applications. Questo linguaggio, infatti, si integra al browser via Applet e permette di creare applicazioni web con caratteristiche e funzionalità delle Desktop Applications. JavaFX include un linguaggio di programmazione - indipendente da Java - chiamato JavaFx Script, oltre ad una crescente libreria di funzionalità grafiche. JavaFX Script è un linguaggio di scripting fortemente orientato alla programmazione grafica rendendo la programmazione di Rich Internet Applications (e di applicazioni grafiche in genere), particolarmente agevolata.
Si tratta di un linguaggio dichiarativo che permette di definire le componenti della User Interface, catturarne gli eventi e gestirli attraverso una semplice sintassi. Inoltre permette di essere indipendente dal dispositivo [2], quindi avremo un unico linguaggio per scrivere applicazioni per web, desktop o dispositivi mobili.
Avendo una sintassi dichiarativa, supporti per l'animazione e built-in di effetti visivi, JavaFX consente di generare meno codice per ottenere determinati risultati, con cicli di sviluppo più brevi e perciò maggiore produttività. JavaFX si distacca dalla classica sintassi Java e permette lo sviluppo di applicazioni con un linguaggio dichiarativo e staticamente tipizzato. Con linguaggio dichiarativo si indica un linguaggio con il quale si istruisce il computer su "cosa" deve fare e non su "come" farla. Con linguaggio staticamente tipizzato si indica che il linguaggio esegue il controllo del tipo principalmente durante la fase di compilazione.

Con JavaFx il tipo di dato può anche non essere definito perché viene riconosciuto in automatico (fase di compilazione), caratteristica non verificabile nella programmazione con Java.
In più, supportando i protocolli JNPL e MIDP, è utilizzabile anche per le soluzioni Web Start e per i dispositivi mobili. Il protocollo JNLP (Java Network Launching protocol) permette la comunicazione tra le applicazioni di rete e la Java Virtual Machine e il MIDP (Mobile Information Device Profile) permette di scrivere applicazioni scaricabili da Internet e servizi per dispositivi mobili collegabili in rete.
La piattaforma JavaFX consiste in un compilatore, un insieme di librerie Runtime e dei tool di sviluppo, che comprendono i plug-in per l'ambiente di sviluppo integrato (IDE) NetBeans, permettendo cosìlo sviluppo di applicazioni in modo agevolato.
Un aspetto importante di JavaFX è proprio che viene eseguito sotto il controllo del Java Runtime, questo comporta che un'applicazione scritta con JavaFx può utilizzare tutte le funzionalità di Java ed ha anche accesso a tutte le interface di programmazione di applicazioni Java oltre che a quelle fornite da JavaFx.
Al momento, ci sono 3 versioni di Runtime: una per Ambient desktop eseguita su Java SE, un'altra per i dispositivi mobili eseguita su Java ME e una terza versione che viene eseguita su JavaTv .
Le prime versioni erano esclusivamente rivolte alle piattaforme Microsoft Windows e Apple Mac OS X , ma ora ci sono versioni che supportano anche Linux e OpenSolaris.
JavaFx Script ha delle "nuove funzionalità per creare facilmente applicazioni immersive che integrano in maniera trasparente contenuti, media e dati sulle più diverse piattaforme" [3] , come spiega Jeet Kaul, Senior Vice President del Client Software Group di Sun.
La vera innovazione di questo linguaggio è il "binding" e il sistema di "Applet Extension" che permette il drag'n'drop dell'applicazione dal browser al desktop:

- Il binding consente di collegare un valore "A" proveniente da una determinata parte di codice ad un altro valore "B", e se valore "A" cambia, allora automaticamente cambia anche il valore "B".

- La Applet Extension dà la possibilità di trascinare, con una semplice azione di trascinamento, l'applet, visualizzato dal sito, all'interno del desktop cambiando il contesto operativo dell'applicazione.

  Per implementare l'Applet Extension si deve utilizzare una classe del pacchetto stage: javafx.stage.AppletStageExtension.

La nuova piattaforma dovrebbe contribuire a dare il via allo sviluppo del Web3.0 [4] (chiamato così perché aggiunge funzionalità alle applicazioni web, ad esempio la "tridimensionalità ").

Java, grazie al supporto di numerosi operatori di sistemi mobili e al sostegno di milioni di sviluppatori software, si è affermata come la piattaforma mobile più solida dell'intero settore. JavaFx Script è un innovativo Domain-Specific Language (DSL)[1] per creare elaborate Interfacce Utente.

JavaFX permette anche di interagire senza alcuno sforzo con classi Java preesistenti. È inoltre possibile fare l'opposto: si può includere una applicazione JavaFX all'interno di un normale programma scritto in Java e utilizzare le classi Swing.

1.2 Le Rich Internet Applications

Fino a non molto tempo fa, i contenuti presenti all'interno della rete Internet venivano definiti in maniera generica, ma con il moltiplicarsi dei servizi sulla rete abbiamo una varietà di contenuti che vengono definiti più specificatamente; ad esempio, "blog", "web-services", "podcasting" e "RIA". Una Rich Internet Application (RIA) è un applicazione web che ha molte della caratteristiche delle applicazioni desktop ed è in grado di reagire dinamicamente agli input dell'utente; possiamo considerarle applicazioni desktop con il vantaggio di essere raggiungibili da Internet. RIA è quindi l'acronimo di Rich Internet Application, una definizione che include una serie di tecnologie e piattaforme, pensate per il Web, che consentono un'interazione più ampia, non solo in termini di contenuti (spesso multimediali) ma anche per quanto riguarda l'usabilità e l'immediatezza dell'interfaccia. Per eseguire le RIA, gli utenti hanno bisogno di installare un Framework usando un sistema operativo che prima di lanciare un applicazione fa gli aggiornamenti ed esegue le RIA. Adobe flash, Java e Microsoft SilverLight sono attualmente le tre piattaforme più comuni e sebbene siano emersi nuovi standard Web, si usano ancora i principi che stanno dietro le RIA.[2]

Proprio la Adobe scrive: "Le RIA offrono un'esperienza avanzata per una soddi-

[1] Un DSL è un linguaggio di programmazione progettato da zero per soddisfare un particolare insieme di sfide e per risolvere specifici problemi

[2] Il termine RIA sembra sia stato usato per la prima volta, in un rapporto di Macromedia (l'azienda che ha progettato e realizzato Flash) nel 2002; Flash è la prima RIA sul mercato e attualmente la più diffusa. RIA meno diffuse sono Java e Silverlight, e sebbene quest'ultima, forte dell'enorme diffusione del browser di casa Microsoft Internet Explorer, è la RIA con il più alto tasso di crescita, grazie all'integrazione completa con le ultime versioni di Internet Explorer.

sfazione maggiore da parte degli utenti e un incremento della loro produttività. Tramite la vasta portata di Internet è possibile implementare le RIA su vari browser, desktop e dispositivi" [5] .
Lo scopo delle RIA è quello di "simulare" l'utilizzo e l'aspetto delle applicazioni desktop classiche, potenziando in questo modo le pagine web che altrimenti, con l'ausilio del solo codice HTML, consentirebbero una interazione limitata. In alcuni casi, come quello di Flash o Silverlight, è necessario installare un plug-in che viene integrato nel browser e consente di visualizzare correttamente l'applicazione; una volta terminata l'installazione, il plug-in gestirà in autonomia il proprio aggiornamento.

In altri casi, invece, la RIA sfrutta le funzionalità dei web browser e, in questi casi, non necessita di alcuna installazione (anche se potrebbe non funzionare se utilizzata con determinati browser).

Nello schema riportato nella figura sottostante vengono riassunte le caratteristiche delle RIA's:

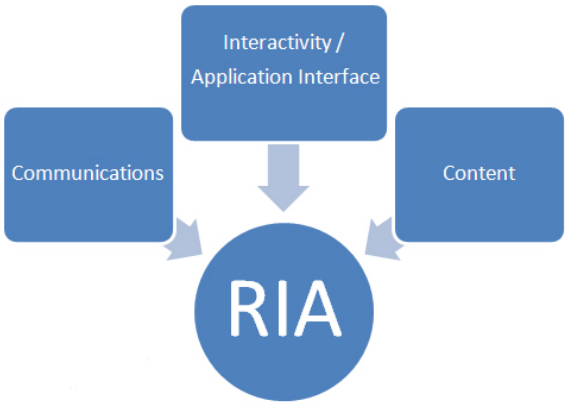

Figura 1.1: Componenti RIA (da [6])

I vantaggi delle RIA sono numerosi, sia per l'efficacia che offrono alle organizzazioni nel distribuire applicazioni al passo coi tempi, sia in termini di business:

1. Incremento della fidelizzazione dei clienti con conseguenti maggiori profitti;
2. Possibilità di sfruttare processi e infrastrutture già esistenti in precendenza.
3. Possibili infinite varietà di effetti, combinazioni e degli stili;

4. Possibilità di personalizzare l'informazione e di indirizzarla all'utente secondo le sue esigenze;

5. Possibilità del raggiungimento della quasi totalità dei desktop abilitati a Internet;

6. Possibilità di sviluppare un'unica interfaccia utente con dicersi supporti multimediali;

Nel 2007 una società di ricerca americana , la Forrester Research (che si occupa di analizzare i cambiamenti delle tecnologie e i conseguenti impatti sui diversi business), ha pubblicato uno studio basato su interviste a fornitori e programmatori RIA, e ai clienti della stessa compagnia. Questo studio chiamato "The Business Case for Rich Internet Applications", dimostra che "RIA ben progettate possono produrre risultati straordinari che aiutano a dimostrare il valore degli attuali investimenti e preparano la strada per progetti RIA futuri" [7], come riferisce la Adobe.
Oggi le RIA vengono utilizzate in ogni campo di sviluppo per il web, dai giochi on-line ai servizi di e-banking fino alle caselle e-mail e alla visualizzazione di video; permettono di costruire applicazioni web-based semplici e intuitive, che riducono i tempi di elaborazione e le curve di apprendimento degli utenti.
Per questo si ritiene che le RIA, utilizzate a livello aziendale, aumentino la produttività, oltre a offrire contenuti e interfacce grafiche migliori.

1.3 L'evoluzione di JavaFx

La Sun Microsystems presentò per la prima volta JavaFx alla conferenza di JavaOne nel maggio 2007. L'annuncio metteva ben in evidenza lo scopo di JavaFx: consentire lo sviluppo e la diffusione di Rich Internet Applications su dispositivi di largo impiego, come telefoni cellulari e browser.
L'intento della famiglia di prodotti JavaFX è fornire la capacità di creare contenuti interattivi, applicazioni e servizi, con caratteristiche di desktop applications, in altri dispositivi.
Il linguaggio JavaFX Script era stato progettato da Chris Oliver, e la Sun Microsystem lo presentò per la prima volta sempre alla conferenza JavaOne 2007. L'anno successivo la Sun annunciò l'intenzione di distribuire JavaFx per desktop entro l'anno e la versione per dispositivi mobili nell'anno seguente. Inoltre la Sun prese accordi con On2 Technologies per poter riprodurre, nelle applicazioni sviluppate in JavaFX, determinati formati video e poter quindi affrontare la concorrenza che utilizzava il rivale Adobe Flash.

I passi compiuti dopo questo primo annuncio si sono susseguiti ad un ritmo accelerato:

- 4 Dicembre 2008: Rilascio della prima versione ufficiale, JavaFX 1.0 , disponibile per il download;
- Febbraio 2009: Viene annunciato che la versione JavaFX 1.1 è la prima versione del linguaggio ad includere le componenti per lo sviluppo su piattaforme mobili;
- Giugno 2009: Viene reso pubblico JavaFX 1.2 che introduce un notevole aumento delle prestazioni e molte novità (supporto per Linux e Solaris, controlli e Layout Grafico integrati, integrazione linguaggio CSS, possibilità di creare svariati tipi di grafici);
- Aprile 2010: Rilascio di JavaFX 1.3 che introduce diverse novità tra cui miglioramento delle prestazioni grazie all'introduzione di un motore grafico chiamato Prism e un editor per NetBeans 6.9 chiamato composer. Inoltre vengono potenziati i supporti per le applicazioni Tv.
- Maggio 2011: Rilascio Versione JavaFx 2.0 Beta. Molte delle nuove funzionalità sono incompatibili con la versione 1.3. Questa versione comprende il Software Develpment Kit (SDK) che consente agli utenti di sviluppare applicazioni e utilities JavaFX e un motore multimediale che supporta la riproduzione di contenuti da internet. Inoltre introduce un componete Web che consente di inglobare del codice HTML all'interno di un'applicazione JavaFX.

È interessante notare che lo sviluppo di JavaFx non è stato rallentato dall'aquisizione da parte di Oracle, della Sun Microsystem. L'aquisizione è avvenuta nel 2009 per 7,4 miliardi di dollari (equivalente a circa 5,2 miliardi di euro) e ha consentito alla Oracle di acquisire Java, Solaris e MySql. A proposito di MySql, l'azienda afferma che andrà ad inserirsi nella gamma di database già distribuita da Oracle. In seguito all'acquisizione della Sun Microsystems[8], Oracle ha assunto automaticamente responsabilità e controllo anche per il linguaggio di programmazione Java.
Da allora, nelle discussioni della comunità degli sviluppatori di software, periodicamente si manifesta la preoccupazione di una possibile inversione di rotta sul carattere open source del linguaggio, sostenuto sin dall'inizio dalla Sun. Per quanto riguarda la posizione di Oracle su questo argomento, si sa solo che il gruppo ha affermato che il Java Development Kit continuerà ad essere open-source con licenza GPL (General Public License, licenza per software libero).

1.4 Architettura di piattaforma

Nella figura 1.2 si può osservare l'architetturadella piattaforma JavaFX. Questa comprende:

- Application Framework, cioè una raccolta di Librerie e Tools che aiutano lo sviluppatore fornendogli elementi che può usare nello sviluppo della sua applicazione;

- Il linguaggio JavaFX Script che permette allo sviluppatore di poter produrre applicazioni Desktop, Tv, e per Dispositivi mobili utilizzando sia elementi comuni a tutte e 3 le versioni di runtime, sia elementi specifici per ogni categoria;

- Tools di supporto sia per il programmatore che per il grafic designer.

Il linguaggio di scripting, JavaFx Script è eseguito su JVM e il meccanismo di profilazione permette la generazione di codice verso diverse piattaforme e la creazione di applicazioni, contenuti e servizi.

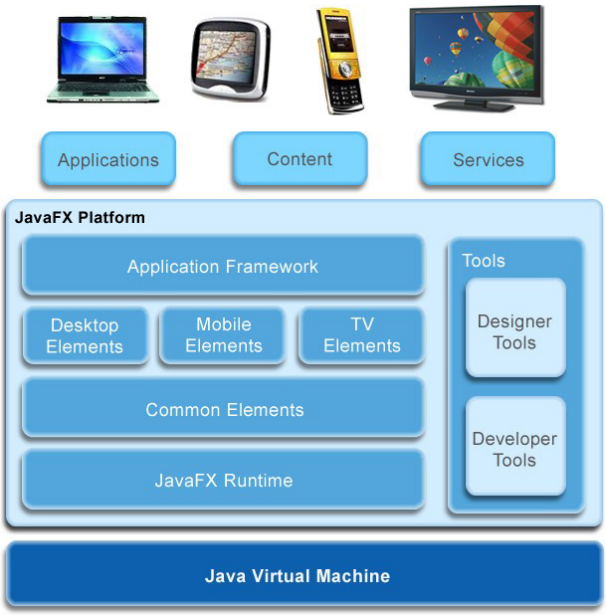

Figura 1.2: Architettura della piattaforma JavaFX [9]

1.5 Tools per lo sviluppo

JavaFX rende più facile il lavoro dei Grafic designer professionali a creare una Interfaccia utente sia per la sua facilità di utilizzo, sia grazie ad alcuni plug-in che permettono di creare originali elaborati con Adobe Photoshop e Adobe Illustrator per animare immagini in risposta ad azioni utente o allo scorrere del tempo. Per i programmatori è disponibile Production-Suite, un download separato che contiene plug-in che esportano creazioni grafiche dai programmi sviluppati con gli strumenti di sviluppo della Adobe in un formato che può essere letto dalla Java Runtime.
Con i programmi della Adobe, dopo aver installato i plug-in, si possono creare componenti grafici e salvarli nel formato .fx.
Grazie a questa operazione si possono quindi importare i propri lavori grafici nell'applicazione JavaFX.
Per sviluppare l'applicazione vera e propria è necessario installare sul proprio calcolatore una macchina virtuale Java (JVM) e la piattaforma JavaFX.
Ci sono 2 modi per installare Java runtime nel proprio pc:

1. Si può prendere Java SDK che permette di sviluppare applicazioni sul pc usando un editor a piacimento e un set di command tools e poi installare la piattaforma JavaFX (il download è reperibile sul sito della Oracle);

2. Si può installare il JavaFX plug-in per NetBeans[3] o Eclipse[4] , che hanno incorporato lo SDK di JavaFx.

Per lo sviluppo del progetto presentato in questa tesi, si è usato l'ambiente di sviluppo integrato NetBeans 6.9.1, e installato il plug-in per JavaFx che mette a diposizione una serie di tool molto comodi per lo sviluppatore.
In particolare il JavaFx Composer che integra nell'IDE una "Palette" che permette di trascinare nell'applicazione funzioni preimpostate che producono codice autogenerato. È sufficiente cliccare su una categoria per usufruire delle funzioni suddivise per categoria, per esempio di seguito possiamo vedere i Controlli disponibili

[3] Ambiente di sviluppo multi-linguaggio sviluppato dalla Sun Microsystems e ora della Oracle Corporation

[4] Ambiente di sviluppo multi linguaggio sviluppato dalla Eclipse Foundation, un consorzio di grandi società

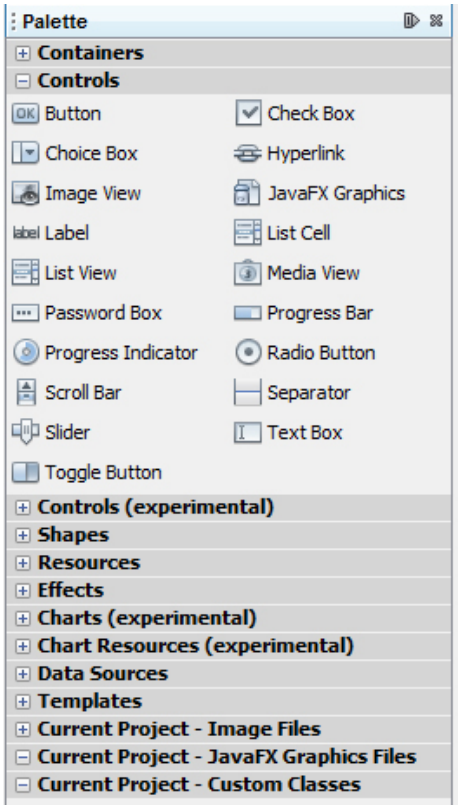

Figura 1.3: Alcune funzionalitá del plug-in JavaFx Composer per NetBeans

Capitolo 2

Panoramica sul linguaggio

2.1 Variabili e tipi di dati

Le applicazioni JavaFX manipolano dati nelle variabili degli script o nelle istanze di variabili delle classi JavaFX. Ognuno di questi dati ha un tipo specifico, come "String" o "Integer" che determina i valori accettati nelle operazioni che sono eseguite con essi.
Come Java, JavaFX ha un set di "tipi" di dato base , delle librerie di runtime che non fanno parte del linguaggio, e che si possono usare per costruire interfacce utente ecc ecc.
Una variabile JavaFX può essere dichiarata in 2 modi:

- Utilizzando "var" (cioè un valore riassegnabile) o
- Utilizzando "def" (cioè una variabile che non si può modificare, in poche parole una costante).

La sintassi di una variabile con la dichiarazione "var" è la seguente:

```
[modificatore] var  nome_variabile [:tipo_dato][= espressione]
```

Il modificatore, che determina la visibilità della variabile per il codice al di fuori dello script, è opzionale, il nome_variabile è il nome assegnato dall'utente alla variabile, il tipo_dato è anche questo un campo opzionale, che se vuoto viene definito dal compilatore.
La visibilità di una variabile di script, per il codice al di fuori dello script, è determinata da modificatori che possono essere dichiarati prima della parola "var" o "def" nella dichiarazione di una variabile.
I modificatori e il tipo di accesso che permettono sono elencati nell'elenco sottostante:

Default : L'accesso all'elemento è pubblico solo al package della relativa classe. Si ha una visibilità di questo tipo quando non viene specificato nessun modificatore di visibilità;

Public : L'elemento risulta visibile da qualsiasi parte del programma senza alcuna restrizione;

Protected : L'elemento è visibile solo dalle classi dello stesso package e dalle sue sottoclassi;

Private : L'elemento è visibile solo all'interno della classe o del metodo in cui è stato dichiarato;

Package : L'elemento è visibile e utilizzabile da tutte le classi presenti in uno stesso package;

public-read : l'elemento può essere letto da qualsiasi parte del codice, ma scritto solo dal codice sullo stesso script. Può essere inizializzato da unObject Literal nello stesso script;

public-read package :L'elemento può essere letto da tutto il codice, ma scritto solo dal codice dello stesso package;

public-read protected : L'elemento può essere letto da tutto il codice, ma scritto solo dal codice dello stesso package o dalle sue sottoclassi;

public-init : L'elemento può essere letto da tutto il codice ma scritto solo dal codice sullo stesso script. Può essere inizializzato da tutto il codice;

public-init package : L'elemento può essere letto da tutto il codice ma può essere scritto solo dal codice dello stesso package;

public-init protected : L'elemento puó essere letto da tutto il codice ma puó essere scritto solo dal codice dello stesso package o dalle sue sottoclassi;

L'espressione, anch'essa opzionale, fornisce il valore iniziale della variabile. Di seguito vediamo degli esempi validi di dichiarazione di variabili:

```
var value:Number = 3;
public var nome:String;
```

La sintassi di una variabile con la dichiarazione "def" è la stessa usata per "var", di seguito degli esempi di dichiarazione di variabile valide:

```
def value:Number = 10;
private def nome:String;
```

JavaFX inoltre dispone di variabili pre-definite (pseudo-variabili):

**PROFILE** : per tutti gli ambienti;

**FILE** : nome del file;

**DIR** : Nome della directory dell'elemento;

Queste variabili, disponibili in sola lettura, sono utilizzabili in tutti i programmi scritti in JavaFX.

Le Classi JavaFX sono contrassegnate dalla parola chiave" class". Tra parentesi graffe è possibile inserire gli attributi, funzioni e attività. Di seguito la sintassi:

```
[modificatore] class Nome\_class {
            Dichiarazione variabili }
```

I tipi di dato disponibili per il linguaggio JavaFX sono:

TIPO	*DESCRIZIONE*
Number	*Valore in virgolamobile*
Integer	*Valore intero*
Byte	*Valore piccolo definito*
Short	*Valore piccolo definito*
Long	*Valore grande*
Float	*Valore in virgola mobile*
Double	*Doppio valore di precisione in virgola mobile*
Character	*Un singolo carattere*
Boolean	*Valore true o false*
String	*Una stringa immutabile*
Duration	*Un periodo di tempo*

Tabella 2.1: Tipi di dato base di JavaFX

Il seguente codice dichiara una classe chiamata Posizione che ha 2 variabili chiamate x e y:

```
public class Posizione{
     public var x:Number;
     public var y:Number;
}
```

La classe può essere istanziata creando un oggetto di tipo "Posizione":

```
Posizione{
    x:''10'';   // inizializza la variabile x con il valore 10
    y:''20'';   // inizializza la variabile y con il valore 20
}
```

Di seguito vediamo la creazione di un'istanza associata a una variabile:

```
public class Posizione{
    public var  x:Number;
    public var  y:Number;
}
public function printPosizione(){
    println("Variabile x: {Posizione.x}");
}
```

È quindi possibile dichiarare una variabile che si riferisce a qualsiasi istanza di un oggetto utilizzando il nome della classe di tale oggetto come tipo.
Le classi possono ereditare da altre classi, estendendole.

Una classe JavaFX può estendere più classi, le classe ereditate vengono chiamate "superclassi", quelle che invece ereditano vengono definite "sottoclassi". Per mettere in atto queste proprietà, si specifica la parola chiave "extends" e un elenco separato da virgole con i nomi di quelle classi. Nel codice sottostante ne abbiamo un esempio:

```
abstract class Animale {
    public var nome:String;
    public function mangia(cibo:String){
        println("mastica {cibo}");
    }
}
abstract public class Gatto extends Animale {
    public function Fusa(volume:Integer){
        println("fusa...livello({volume})");
    }
    public function graffia(){
        println("Il gatto graffia...")
    }
 }
```

In JavaFx gli Object Literals sono utilizzati per creare e inizializzare le istanze di una classe. La sintassi richiede che si specifichi il nome della classe seguito dall'istanza della variabile della classe di cui si vogliono inizializzare i valori.
Un esempio qui sotto:

```
var s1:Stage = Stage {
                    title: ''Stage 1''
                    x:20
                    y:30
                    width: 200
                    height: 200
 }
var s2: Stage = Stage {
                    title: ''Stage 2''
                    x: s1.x +s1.width
 }
```

Una funzione JavaFx è un gruppo di dichiarazioni e espressioni e la sintassi per la sua dichiarazione che consente il riutilizzo del codice. La sintassi per la sua dichiarazione è:

```
[modificatore] function nome\_funz([argomenti])[:tipo_di_ritorno] {
    body}
```

Le funzioni possono ricevere e restituire argomenti. In caso la funzione non restituisca nulla, il tipo di ritorno sarà Void. In JavaFX i parametri della funzione (arguments) sono sempre e solo passati per valore (mai per riferimento), il che significa che non si potranno mai riassegnare i valori dei parametri (arguments) passati alla funzione.
Di seguito esempi validi di dichiarazione di funzione:

```
function stampa Data() : Void {
    println(newDate());
}
function somma (valore1 : Number,  valore2 : Number) : Number {
    return (valore1+valore2)/2 ;
}
```

Nel seguente esempio per invocare una funzione definita con una classe, viene usata un'istanza di classe:

```
  var stage:Stage =Stage {
```

```
            title: ''Questo e' lo Stage''
        };
        stage.close()
```

Esiste una funzione speciale che viene invocata automaticamente dall'engine. Questa è la funzione "Run", simile a un metodo "main" che accetta parametri di avvio.
Una subclasse può talvolta sovrascrivere una funzione nella sua classe base. Ovviamente il modificatore di visibilità dev'essere settato adeguatamente per non rendere l'accesso illegale. Per sovrascrivere una funzione astratta è necessario una sublasse concreta, dichiarata nella sua classe base.
Per esempio, se si ha una classe base "A" che dichiara una funzione chiamata "Fare" e se c'é una subclasse di "B" che implementa "Fare", una subclasse "c" di "B" potrebbe sovrascrivere "Fare" perché un'altra implementazione esiste già in "B". I possibili motivi per sovrascrivere una funzione sono:

1. Per implementare una funzione astratta definita nella classe base;
2. Per rimpiazzare completamente l'implementazione della funzione sovrascritta;
3. Per permettere l'esecuzione di passaggi addizionale prima o dopo della sovrascrizione della funzione.

Di seguito abbiamo un esempio:

```
public class Rettangolo extends Esempio{
    public var x:Number;
    public var:Numer ;
    public var largezza:Number;
    public var altezza:Number;
 }
public override bound function getArea() :Number {
    altezza * larghezza;
 }
```

La parola chiave "override" è usata per indicare che quella è una sovrascrizione di una funzione esistente.

2.2 Binding

Il data binding è una delle caratteristiche più potenti del linguaggio JavaFX. Grazie al data binding possiamo collegare tra loro i valori di diverse variabili.

In pratica, quando la variabile collegata cambia di valore, cambiano anche tutte quelle dipendenti.
È possibile associare un valore a una variabile di script, a un' istanza di variabile di una classe JavaFX, a un'espressione, a un valore restituito da una funzione, a una sequenza. L'uso più comune del binding in JavaFX coinvolge le componenti GUI o le proprietà di un nodo grafico.
Di seguito, un esempio valido di binding:

```
var ciao="Ciao";
var temp="bind a"';
println("ciao:{ciao} temp:{temp}");
```

Il codice sovrastante produce questo output:

```
ciao:Ciao temp:Ciao
```

Il binding può essere unidirezionale o bidirezionale, cioè può essere creato in 2 modi:

1. Quando viene dichiarata e inizializzata una variabile di script o l'istanza di una variabile di classe;

2. Quando fornisce il valore iniziale di un' istanza di variabile in un oggetto che inizializza.

Nell'esempio precedente si osserva un esempio di binding a una variabile di script. Nel seguente esempio invece si può osservare un esempio di binding in un Object Literal:

```
class StampaValore{
    var valore:Integer;
 }
    var targetValore=1;
    var  tot = StampaValore {
                        valore: bind tot;
     }
 println(tot.valore);
```

Viene creata una classe chiamata StampaValore con una singola istanza chiamata valore; poi viene dichiarata una variabile di script chiamata target Valore a cui viene assegnato un valore, e dopo viene creata un'istanza della classe StampaValore. Il risultato di questo codice è: "1" Se invece noi scriviamo:

```
targetValore=2;
```

verrà stampato "2".
Di seguito si può osservare un esempio di Binding a una funzione:

```
function ipotenusa(a,b) {
    return Math.sqrt(a*a + b*b);
 }
var base = 3;
var altezza=4;
def  Ipo = bind ipotenusa(base,altezza);
println(Ipo);
```

Nel caso appena visto, il valore di "Ipo" varia solo se vengono variati i valori delle variabili esterne alla funzione , cioè "base" e "altezza". Per collegare invece una variabile all'interno di una stessa funzione bisogna applicare una funzione Bound, che funziona anche al di fuori del binding[11].
Di seguito un esempio in cui il contenuto di una Text box cambia dopo un evento di I/O:

```
bound function getTotal():String {
    def item1 = if (itemOne.selected) .75 else 0;
    def item2 = if (itemTwo.selected) .50 else 0;
    def item3 = if (itemThree.selected) .25 else 0;
    return "$ {total}";
 }
def finalOrder = Text {
    content: bind getTotal()
    font : Font {
             size: 18
     }
 }
```

JavaFx supporta il binding bidirezionale, specificato mediante le parole chiave "with inverse" alla fine dell'espressione soggetta al binding.

```
Var a=bind b with inverse
```

Questo codice assicura che "a" viene ricalcolata in caso di modifiche a "b" e b è calcolato come modifica di "a" ; si puó dire liberamente che questa espressione è equivalente a:

```
var a= bind b
b= bind a
```

2.3 Triggers

I Trigger sono parte integrante della dichiarazione delle variabile e delle costanti in JavaFx.
Il Trigger è la parte facoltativa della dichiarazione di espressioni e costanti e viene introdotto con la parola chiave REPLACE seguita da una specifica modifica di una variabile e un blocco chiamato Trigger Block.
La variabile o costante a cui il Trigger è collegato si chiama variabile "Osservata". Il Trigger block viene quindi eseguito ad ogni modifica della variabile/espressione a cui è collegato.
Nel prossimo esempio si può osservare un esempio valido di Trigger con aggiornamento di una variabile
Questo viene eseguito 2 volte: una quando la variabile "i" viene inizializzata, e una quando la variabile viene eaggiornata:

```
var i= 1000 on replace {
    println(''Variabile {i}.'');
 }
i=2000
```

L'output di questo codice è:

```
Variabile 1000
Variabile 2000
```

In questo caso la variabile è stata inizializzata a 1000, ma se non viene inizializzata, come nel successivo caso, il risultato è:

```
var valore: Integer on replace{
   println("il valore cambia a {i}");
};
valore=1;
valore=2;
```

In questo caso verranno stampati 3 valori, 2 per l'assegnamento esplicito del valore della variabile e 1 per il valore di default della variabile, che in questo caso è 0:

```
il valore cambia a 0
il valore cambia a 1
il valore cambia a 2
```

Qualche volta è utile conoscere il valore che la variabile ha assunto prima dell'aggiornamento provocato dal trigger. Si può avere accesso al vecchio valore aggiungendo la parola chiave "oldValue" nella dichiarazione del Trigger:

```
var valore:Integer = 20 on replace oldValue {
    println("il valore cambia da {oldValue} a {valore}");
 }
valore=40;
```

Nel vedere l'dentificativo "oldValue", il compilatore crea una variabile dello stesso nome e lo stesso tipo di variabile del trigger. Il codice precedente, che inizializza la variabile "valore" a 20 e poi la cambia a 40, stampa :

```
il valore cambia da 0 a 20
il valore cambia da 20 a 40
```

È possibile collegare un Trigger a una variabile dichiarata Bound nello stesso modo con cui si farebbe per qualsiasi altra variabile. Questo può esere applicato sia alle variabili dichiarate "var" o "def". il seguente codice fa un bind della variabile boundValore al valore di un'altra variabile chiamata targetValore e riporta ogni volta che quest'ultima cambia, cioé in seguito ad un nuovo assegnamento al valore:

```
var targetValore = 1;
def boundVariabile = bind targetValore on replace {
    println("boundValore cambia in {boundValore}");
 }
targetValore =2;
```

L'output generato da questo codice è:

```
boundValore cambia in 1
boundValore cambia in 2
```

La sintassi dei Trigger visti fino ad ora permette di collegare un Trigger a una variabile nel momento della sua dichiarazione. A volte però, può essere utile collegare un trigger a una variabile dichiarata in un'altra parte del codice.
È possibile collegare un Tigger a un'istanza di variabile di un oggetto, il cui codice crea un'istanza aggiungendo una dichiarazione di "Override' nell'oggetto inizializzato (il che significa che la variabile deve essere stata precedentemente dichiarata con il modificatore public).
Nell'esempio successivo, ci si riferisce ad una variabile chiamata "visibile" , a cui non si può attaccare un Trigger nella dichiarazione della variabile, ma si può fare questo:

```
        Stage{
            title: "Test"
            width: 100
            height: 100
            override var visibile = true  on replace{
                println("Visibile cambia in {visibile}");
             }
          }
```

La parola chiave OVERRIDE indica che ci si riferisce a una variabile esistente chiamata "visibile", e se questa viene omessa, sarà simile a un tentativo di dichiarare una variabile locale di quel Object Literal.
Non si può quindi aggiungere un Trigger in una variabile interna ud un'oggetto che è stato già istanziato.

2.4 Animazioni

L'animazione è una delle caratteristiche più importanti di JavaFX e la struttura di animazione di base è parte della piattaforma JavaFX , non delle librerie GUI. Il tutto si riduce alla capacità di modificare il valore di una o più variabili in un periodo di tempo definito.
Le animazioni sono controllate da una "timeline" cioé un'arco di tempo. Le timeline usano i keyframes per definire le variabili i cui valori devono essere fissati a funti specifici dell'animazione.
Di seguito un esempio:

```
var nodo: Node;
var scene.Scene;
nodo =Circle {
           centerX: 30, centerY: 100
           radius: 40
           fill: Color.MAGENTA
 };
Stage{
    title: "Animazione"
    scene: Scene {
    width: 200
    height: 100
```

```
    content[nodo]
     }
}

var timeline = Timeline {
    repeatCount: Timeline.INDEFINITE
    autoReverse: true
    keyframes: [ at (5s) {nodo.translateX => scene.width - nodo.layoutBounds.width;
                          nodo.rotate => 360; } ]
    }
timeline.play();
```

Il keyframe specifica cosa dovrà accadere trascorsi 3 secondi dall'avvio dell'animazione, richiamata da timeline.play(), la funzione repeatCount invece indica il numero di volte che deve essere eseguita l'animazione, e la funzione autoreverse (variabile booleana) indica se il nodo dovrà eseguire l'animazione a ritroso alla fine del ciclo.
È possibile animare qualsiasi variabile di un nodo per produrre effetti di traslazione, dissolvenza, rotazione, e così via. Per farlo, è necessario creare una Timeline, associarla al nodo, e poi avviarla. Le variabili controllate dalla classe Timeline sono descritte nella tabella 2.2 :

VARIABILE	*TIPO*
keyFrames	*KeyFrame*[]
repeatcount	*Number*
autoReverse	*Boolean*
time	*Duration*
rate	*Number*
framerate	*Number*
interpolate	*Boolean*
currenRate	*Number*
cycleDuration	*Duration*
totalduration	*Duration*
paused	*Boolean*
running	*Boolean*

Tabella 2.2: [Variabili della classe Timeline]

Il runtime JavaFX comprende un piccolo numero di classi che fanno la maggior

parte di questo lavoro. Queste classi sono chiamate Transitions, e sono tutte derivate dalla classe astratta Transitions che può essere trovata nel pacchetto javafx.animation.transition.
Una Transition è una timeline gestita dalla Runtime di JavaFX e deriva dalla class Timeline. Il linguaggio JavaFX mette a disposizione 6 transitions predefinite e due classi che consentono di eseguire più transizioni una dopo l'altra o in parallelo.
Le Tansition disponibili sono:

- TranslateTransition
- FadeTransition
- RotateTransition
- PathTransition
- ScaleTransition
- PauseTransition

Ogni Transitions ha le sue variabili aggiuntive che specificano gli effetti che questo fornisce, ma tutte gestiscono determinate variabili che sono elencate nella tabella 2.3 :

VARIABILE	*TIPO*
node	*Node*
interpolator	*Interpolator*
action	*function* () : *Void*

Tabella 2.3: [Variabili gestite da tutte le classi Transition]

I valor intermedi che vengono assegnati ad una variabile che compare nel keyframe vengono calcolati da un "interpolator" che tiene conto dei valori di partenza, dei valori finali e della posizione corrente senza la parte di animazione che interessa la variabile. L'Interpolator viene specificato usando la parola chiave tween, seguita da un'istanza dell'oggetto che estende la classe javafx.animation.Interpolator. Ci sono sei standard interpolators forniti dalla classe Interpolators nel package javafx.animation che sono elencati di seguito:

Interpolator.LINEAR : È l'interpolator di default. Il cambiamento nel valore della variabile associata è proporzionale allo scorrere del tempo;

interpolator.DISCRETE : Questo Interpolator permette alla variabile si conservare il suo valore iniziale fino alla fine del keyframe, e a quel punto viene assegnato il valore;

Interpolator.EASIN : Ha le stesse proprietà dell' Interpolator.LINEAR ma il valore cambia + lentamente all'avvio del keyframe;

Interpolator.EASEOUT : Ha le stesse proprietà dell' Interpolator.LINEAR ma il valore cambia + lentamente alla fine del keyframe;

Interpolaor.EASEBOTH : Ha le stesse proprietà dell' Interpolator.LINEAR ma il valore cambia + lentamente all'inizio e alla fine del keyframe;

Interpolator.SPLINE : Il modo in cui il valore della variabile cambia dipende da una curva spline associata con l'Interpolator

Di seguito possiamo osservare un esempio valido in cui viene usato l'Interpolator:

```
Timeline{
    keyFrames: [
        at (1s) {
            valore1 => 15.0 tween Interpolator.EASEBOTH;   }
     ]
};
```

La classe TranslateTransition muove un nodo dalla sua posizione corrente ad un'altra specifica posizione. Le variabili controllate dalla classe TranslateTransitions sono elencate nella tabella 2.4. Di seguito un esempio funzionante di TranslateTransition:

```
var cerchio: Node;
var scene.Scene;
cerchio =Circle {
           centerX: 30, centerY: 100
           radius: 40
           fill: Color.MAGENTA
 };
Stage{
     title: "Animazione"
     scene: Scene {
     width: 200
     height: 100
     content[ cerchio ]
```

VARIABILE	*TIPO*
duration	*Duration*
fromX	*Number*
fromY	*Number*
fromZ	*Number*
toX	*Number*
toY	*Number*
toZ	*Number*
byX	*Number*
byY	*Number*
byZ	*Number*

Tabella 2.4: Variabili gestite dalla classe TranslateTransition

```
      }

var tran = TranslateTransition {
      duration:10s
      node: cerchio
      fromX:30
      fromY:100
      toX:400
      toY:100
      autoReverse:false   }
   }
}
tran.play();
```

La classe ScaleTransition permette l'incremento o il decremento della grandezza del nodo in un certo periodo di tempo. Le variabili controllate dalla classe ScaleTransition sono le stesse usate dalla classe TranslateTransition, ma invece di controllare le coordinate di layout coordinano il valore della scala.
Di seguito troviamo un esempio valido di ScaleTransition, per comodità ci rifeririamo al codice sovrastante dalla riga 1 alla 13:

```
ScaleTransition {
       duration: 2s
       node: cerchio
       byX: 1.5 byY: 1.5
```

```
        repeatCount:4
        autoReverse: true
 }
```

La classe FadeTransition anima la variabile opacità del nodo per creare effetti di dissolvenza. Di seguito troviamo un'esempio valido di FadeTransition, per comodità ci riferiamo al codice dell'esempio precedente dalla riga 1 alla riga 13:

```
var transition = FadeTransition {
    duration:3s
    node: cerchio
    repeatCount :Timeline.INDEFINITE
    autoReverse:true
    fromValue: -1.0
    byValue: 1.0
}
transition.play();
```

I parametri utilizzati sono elencati nella tabella :

duration	*Duration*	*Il tempo in cui l'animazione viene eseguita*
fromValue	*Number*	*Il valore iniziale di opacita*
toValue	*Number*	*Il valore finale di opacita*
byValue	*number*	*Il valore da cui l'opacita dev'essere cambiata*

Tabella 2.5: [Parametri della classe FadeTransition]

La classe PathTransition permette di animare un oggetto che si muove entro una percorso arbitrario (geometrico) in un perido di tempo. La path nella quale il nodo si muove è identificata da una variabile di tipo AnimationPath che si trova nel package javafx.scene.transition. Di seguito un esempio valido di PathTransition:

```
var path = Path { elements: [
                    MoveTo {   x: 10, y: 100 }
                    CubicCurveTo {
                        controlX1: 50, controlY1: 10
                        controlX2: 150, controlY2: 200
                        x: 190, y: 10
```

```
                    }
                    VLineTo {   y: 190   }
                    HLineTo {   x: 10     }
                    ClosePath{} ]
 };
var scene : Scene;
var node: Node;
var cerchio = Circle {
    centerX: 100, centerY: 100
    radius: 40
    fill: Color.MAGENTA
};
Stage {
    title: "MyApp"
    scene: Scene {
    width: 600
    height: 450
    content: [cerchio,path ]
    }
 }
var transition=PathTransition {
    duration: 10s
    node: cerchio
    repeatCount: Timeline.INDEFINITE
    autoReverse: false
    orientation: OrientationType.ORTHOGONAL\_TO\_TANGENT
    path: AnimationPath.createFromPath(path)
  }
transition2.play();
```

La classe PauseTransition aggiunge soltanto una variabile chiamata duration di tipo duration nella sua classe base. Quando è richiamata, causa un ritardo (espresso in millisecondi) che può essere ripetuto con la variabile repeatCount. Un esempio di PauseTransition è:

```
 PauseTransition {
       duration: 2s
       repeatcount : 4
  }
```

La classe RotateTransition ruota in riferimento al suo punto centrale da un certo angolo ad un'altro o in riferimento ad uno specifico valore.
Le variabili che controlla la classe RotateTransition sono elencate nella tabella 2.6:

VARIABILE	*TIPO*
duration	*Duration*
axis	*point3D*
fromAngle	*Number*
toAngle	*Number*
byAngle	*Number*

Tabella 2.6: [Parametri della classe RotateTransition]

Il linguaggio JavaFx, oltre a mettere a disposizione queste funzionalità, permette di eseguire un set di effetti di transition uno dopo l'altro o nello stesso momento. questo grazie alle classi SequentialTransition e ParallelTransition.
La variabile che gestisce questa classe è solo una la variabile content di tipo Timeline[ ].
Di seguito è riportato un esempio di SequentialTransition, che è un po un collage degli esempi precedenti nella quale vengono eseguiti un FadeTransition per 10 secondi e un TranslateTransition per altri 10 secondi:

```
var scene : Scene;
var node: Node;
var cerchio = Circle {
    centerX: 100, centerY: 100
    radius: 40
    fill: Color.MAGENTA
};
Stage {
     title: "MyApp"
     scene: Scene {
     width: 600
     height: 450
     content: [cerchio,path ]
     }
}
```

```
var seq=SequentialTransition {content: [
       FadeTransition {
                 duration: 10s
                 node: cerchio
                 fromValue: -1.0 toValue: 1.0
                 autoReverse: false
       },
      TranslateTransition {
                 duration:10s
                 node: cerchio
                 fromX:30
                 fromY:100
                 toX:400
                 toY:100
                 autoReverse:false }
 ] }
seq.play();
```

Per quanto riguarda il ParallelTransition si può affermare che viene utilizzata la stessa procedura adottata per il SequentialTransition.

2.5 Creazione Database

In javaFx ci sono vari modi per effettuare l'accesso a un DB (DataBase). In qualsiasi modo si voglia realizzare, c'è bisogno di un Driver per creare la connessione con il DB.
Esistono diversi driver che si possono scaricare,per esempio MySQL Connector/J Drive, JavaDB (Network), PostgreSQL o JDBC-OJBC Bridge (un implementazione di driver per DB che usano ODBC; converte le chiamate JDBC in funzioni ODBC).
Dopo aver scelto con che Driver connettersi al database, per accedervi, bisogna creare una classe che può essere configurata con le informazioni necessarie per individuare e connettersi a un DB Server che leggerà il contenuto delle tabelle. Il codice seguente mostra come fare:

```
var nomeDriver = 'Driver';
var url= 'UrlDriver';
Var user ='nome_user';
Var password ='inserire_password';
```

```
Var connessione : Connection = null;
Var statm : Statement = null;

Class.forName (nomeDriver);
Connessione =DriverManager.getConnection(url,user,password);
Statm = connessione.createStatement();
Statm.executeQuery(inserire_query);
Statm.close();
Connessione.close();   }
```

Bisogna perciò definire una variabile di connessione e una variabile di Statement per eseguire la query scritta il SQL.
Ovviamente nelle variabili 'Driver', 'UrlDriver' e in tutti gli altri campi bisogna inserire i dati relativi al Driver che si è scelto di usare.

Capitolo 3

Il progetto

Il progetto realizzato per questa tesi è un'applicazione web per un'agenzia di viaggi. Generalmente questi tipi di progetti vengono realizzati con linguaggi di scripting come JavaScript o PHP. Quest'ultimo è stato principalmente utilizzato per la realizzazione di pagine web dinamiche.
In questo caso, la realizzazione dell'applicazione è realizzata completamente con JavaFx Script.
L'applicazione realizzata,consente all'utente di :

1. Consultare i viaggi proposti dell'agenzia sottoforma di pacchetti all- inclusive per svariate destinazioni;

2. Fare una ricerca personalizzata tra i viaggi disponibili, dove l'utente sceglie sia la città di partenza sia la destinazione;

3. Visualizzare i dettagli di ogni viaggio.

I servizi che l'applicazione offre, invece, agli amministatori del sito web sono:

1. Inserimento di un nuovo viaggio;

2. Inserimento di una nuova destinazione.

Queste funzionalità permettono così il proprietario dell'agenzia di inserire nuovi viaggi-pacchetto o nuove destinazioni, che poi verranno aggiunte alle proposte già presenti e di essere indipendente dal programmatore che gli ha venduto il progetto.
Per capire meglio le funzionalità dell'applicazione, si guardi il diagramma Use-Case realizzato in UML (Unified Model Language).

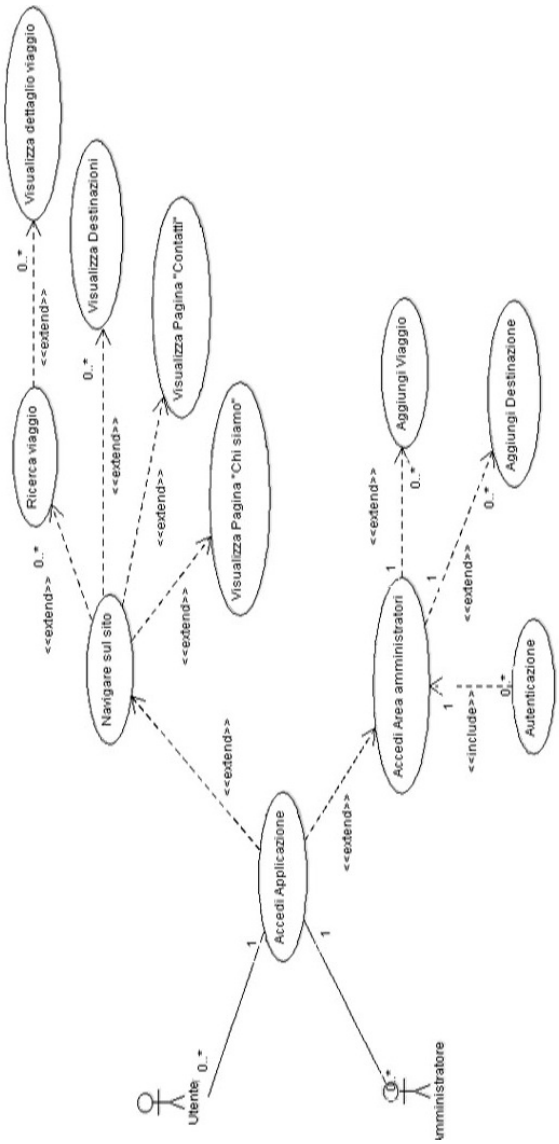

Figura 3.1: Diagramma Use Case

Lo schema Use-Case è un diagramma che permette di descrivere le funzioni di un sistema grazie ai suoi componenti di modello e alle associazioni che intercorrono tra queste.
Quando il proprietario dell'agenzia vorrà aggiungere dati nel suo DB è sufficente che inserisca i dati necessari all'autenticazione (username e password) nel form dedicato nell'Area Amministratori presente nella HomePage.
L'applicazione inoltre possiede 2 animazioni, una che viene eseguita all'avvio

dell'applicazione, e la seconda che viene eseguita in seguito al click dell'aereo (node) presente nella parte alta della pagina principale.

3.1 Creare una User Interface con NetBeans

Creare una User Interface con NetBeans, è abbastanza semplice grazie a tutti i controlli grafici messi a disposizione dai plug-in scaricabili, in particolare dal JavaFx Composer.
Grazie a questo plug-in, viene messa a disposizione del programmatore una mascherina contenente una serie di funzionalità grafiche, che l'utente può trascinare nel programma.
In questo modo viene autogenerato del codice non modificabile.
Creare un'interfaccia utente accattivante è molto importante perciò bisogna scegliere con cura gli elementi messi a disposizione dal linguaggio per avere un risultato quanto migliore possibile.
L'interfaccia utente non si basa su uno specifico contenitore, ma è come costruire una scena in un contesto, entrambi indipendenti dalla piattaforma.
Lo stage rappresentato dalla classe javafx.stage.Stage agisce come sostituto di livello superiore, mentre scene, un'istanza della classe javafx.scene.Scene è il container effettivo dove collocare gli elementi dell'interfaccia grafica.
Durante la fase di esecuzione, lo stage è legato ad un'appropriato contenitore di livello superiore che dipende dall'ambiente in cui l'applicazione è in esecuzione.
Nel caso sia una Desktop Application è un frame, in un browser è un applet, su un dispositivo mobile è una midlet e su un dispositivo TV è una xlet.
Questi 4 contenitori hanno diverse API's, quindi se si sta scrivendo una GUI in JavaFX il codice dipende, almeno in parte, dal contenitore utilizzato. La selezione e la gestione di tale contenitore compete al runtime JavaFX.
Molte applicazioni richiedono un singolo stage, ma nelle applicazioni Desktop e possibile anche l'utilizzo di più stage.
La classe stage ha istanze di variabili che è danno la possibilità di personalizzare l'aspetto e il comportamento del contenitore di livello superiore, come ad esempio il titolo.
Queste sono riassunte nella tabella 3.1.

$VARIABILE$	$TIPO$
$contains - focus$	$Boolean$
$icons$	$Image[]$
$title$	$String$
$visible$	$Boolean$

Tabella 3.1: [Variabili gestite dalla classe Stage]

Inoltre la classe Stage ha un set di variabili che permettono di modificare la sua posizione e la grandezza che sono elencate nella tabella 3.2.

$VARIABILE$	$TIPO$
x	$Number$
y	$Number$
$width$	$Number$
$height$	$Number$
$resizable$	$Boolean$
$iconified$	$Boolean$
$fullScreen$	$Boolean$

Tabella 3.2: [Variabili gestite dalla classe Stage che determinano la posizione e la grandezza]

La classe scene, è il punto di partenza per costruire un'applicazione in JavaFx ed è formata da una struttura gerarchica di nodi che rappresentano tutti gli elementi visivi/grafici dell'interfaccia. La scena grafica è radicata nella variabile "content" della scene, che è una sequenza di nodi.
Le variabili definite dalla classe Scene sono elencate nella tabella 3.3 .
Disposti all'interno della scene, implementata dalla classe javafx.scene.Scene si trovano i Node. I node (nodi) sono strutture di base istanziati della classe javafx.scene.Node, e costituiscono l'interfaccia grafica dell'applicazione JavaFx. Il Runtime fornisce una serie di tipi di nodi che è possibile usare per costruire una scena.
Questi nodi, che vanno dalle forme più semplici (forme basilari come rettangoli), a controlli più complessi che includono componenti della classe Swing di Java e un visualizzatore multimediale che permette la riproduzione di musica e video, sono organizzati nella Scene come una struttura ad albero aciclica, cioé che non

VARIABILE	*TIPO*
stages	*Stage*
x	*Number*
y	*Number*
widht	*Number*
height	*Number*
fill	*Paint*
cursor	*Cursor*
stylesheets	*String[]*
camera	*Camera*
content	*Node[]*

Tabella 3.3: [Variabili definite dalla classe Scene]

punta a nessun'altro albero. La struttura della Scena grafica é riassunta nella figura 3.2:

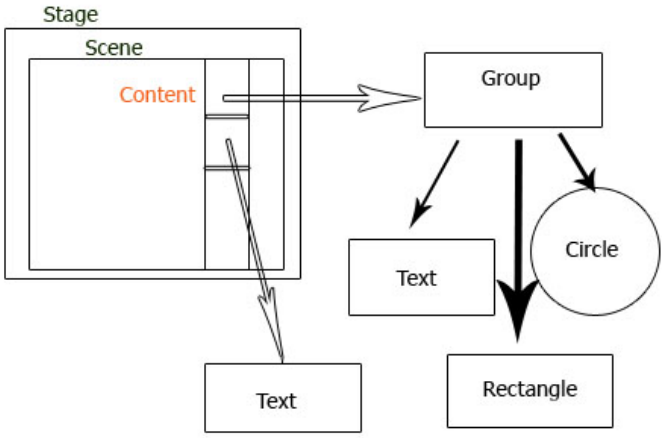

Figura 3.2: Organizzazione Scena Grafica

Per iniziare, ho avviato NetBeans Ide 6.9.1 per creare un nuovo progetto e ho selezionato il linguaggio di interesse, in questo caso JavaFx. Quando viene richiesto di scegliere il tipo del file di progetto, ho selezionato la voce Desktop Application.
In seguito a queste operazioni, l'ambiente di sviluppo mette a disposizione una finestra composta da diverse parti:

- La sezione "Design" mostra le componenti grafiche che formano il progetto;
- La sezione "Source" mostra il codice sorgente sul quale il programmatore può scrivere (tranne nella sezione di codice autogenerato);
- Una sezione "Resource", che nel caso di visualizzazione del Design mostra un albero con gli identificativi dei componenti grafici, nel casi di visualizzazione del Codice mostra un albero con tutte le funzionalità e le risorse utilizzate;
- Una finestra di Output" che visualizza l'output generato dall'applicazione e gli eventuali errori presenti;
- La palette del JavaFX Composer;

La finestra con tutte le componenti è riportata nella figura 3.3:

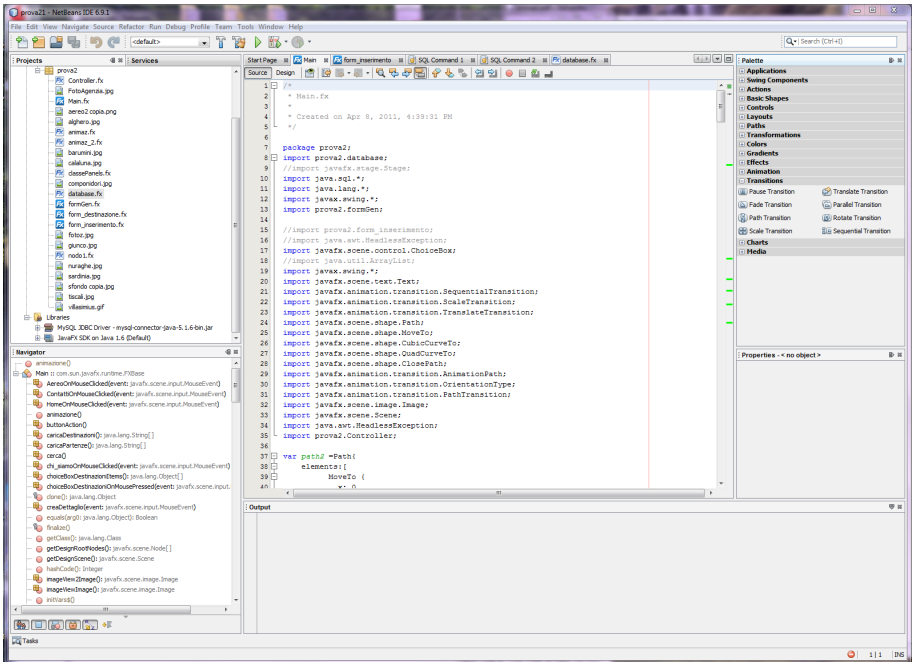

Figura 3.3: Finestra principale di NetBeans

Nel realizzare questo progetto mi sono servita di numerose funzioni grafiche, che compaiono anche sotto forma di controlli nella palette:

- Dei ChoiceBox che permettono all'utente di selezionare delle opzioni per la ricerca di viaggi. I choiceBox sono gestiti dalla classe javafx.scene.control.

VARIABILE	*TIPO*
item	*Object*[]
selectedIndex	*Integer*
selectedItem	*Object*
showing	*Boolean*
unitincrement	*Number*
blockincrement	*Number*

Tabella 3.4: [Variabili gestite dalla classe ChoiceBox]

La lista delle variabili della classe ChoiceBox è riportata nella tabella 3.4 :

- Un MenuBar che gestisce la visualizzazione delle pagine principali del sito web realizzato. Il menu Bar si può definire un contenitore nella quale vengono introddotti vari Menu e viene dichiarato richiamando la classe javafx.preview.control.MenuBar.
  Tra le variabili che gestisce questa classe c'è la variabile "menus[ ] nella quale vengono richiamati i vari Menu contenuti nel MenuBar. I Menu sono gestiti dalla classe javafx.preview.control.Menu.

- Dei Button (pulsanti) che gestiscono l'accesso all'area privata del sito e la ricerca di specifici viaggi. Le variabile che gestisce la classe Button sono elencate nella tabella:

VARIABILE	*TIPO*
action	$function() : Void$
strong	*Boolean*
armed	*Boolean*

Tabella 3.5: Variabili gestite dalla classe Button

Il Button, quando cliccato esegue un'azione specificata dalla funzione richiamata dalla variabile action.

- Delle forme geometrice, più specificatamente dei rettangoli, per evidenziare dei piccoli form. Per inserire una forma geometrica nell'interfaccia grafica è sufficente trascinare dalla palette di JavaFx Composer la forma desiderata, e successivamente ridimensionarla e posizionarla a proprio piacimento. Le figure geometriche sono gestite dalla classe javafx.scene.shape

e le variabili a disposizione per customizzare l'aspetto e il comportamento della figura sono innumerevoli.

- Diversi visualizzatori di immagine chiamati "Image View" che permettono di introdurre delle foto nelle risorse dell'applicazione e di visualizzarle nell'interfaccia. Per visualizzare un'immagine in JavaFx, bisogna seguire 2 procedure:

  1. Il primo passo è caricare i dati "grezzi" pre l'immagine, un compito eseguito dalla classe javafx.scene.image.Image.
  2. Una volta creato l'oggetto Image, si carica in un'istanza dell'ImageView, che è un nodo e può essere aggiunto nella scena grafica.

- Una ListView che visualizza i risultati della ricerca personalizzata. Cliccando su un risultato della lista verrà poi aperta un'altra pagina con i dettagli del viaggio selezionato. Le variabili della classe ListView sono elencate nella tabella 3.6:

$VARIABILE$	$TIPO$
$items$	$Object[]$
$vertical$	$Boolean$
$pannable$	$Boolean$
$echoChar$	$String$
$hideDelayx$	$Integer$
$selectedIndex$	$Integer$
$selectedItem$	$Object$
$focusedIndex$	$Integer$
$focusedItem$	$Objectr$
$cellFactory$	$function() : ListCell$

Tabella 3.6: [Variabili gestite dalla classe ListView]

- Un Exclusively Visible Template. Questo particolare Template, che si trova anche nella palette del JavaFX Composer, permette di creare le pagine che presentino un particolare stato, in modo che il pannello sia visibile solo in quel determinato stato. Per impostarlo è necessario compilare la finestra illustrata nella figura 3.4:

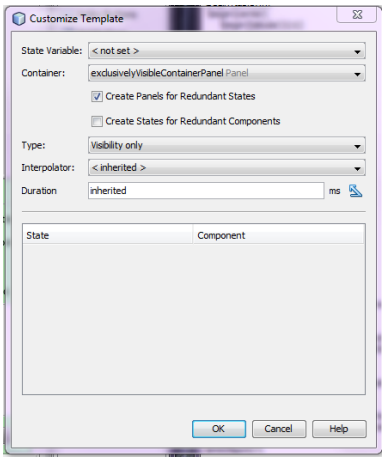

Figura 3.4: Finestra del Wizard del ExclusiveVisibility Template

La figura 3.5 mostra la pagina principale dell'applicazione, con tutte le componenti grafiche:

Figura 3.5: Pagina principale dell'applicazione

3.2 Creare la connessione al Database con NetBeans

Il Database utilizzato nell'applicazione, è stato realizzato in MySql ed è formato da 3 tabelle:

Viaggio in cui sono contenuti tutti i dati dei viaggi proposti dall'agenzia;

Users in cui sono contenuti i dati necessari all'autenticazione dell'Area Amministratori;

Destinazioni in cui sono contenuti i dati relativi ad ogni destinazione proposta dall'agenzia .

Per importare il Database e renderlo visibile all'applicazione, ho selezionato la sezione Services presente in NetBeans e cliccando col tasto destro del mouse nell'opzione Database, ho creato una nuova connessione al database. La figura mostra 3.6 la finestra che viene visualizzata in seguito alla creazione di una nuova connessione:

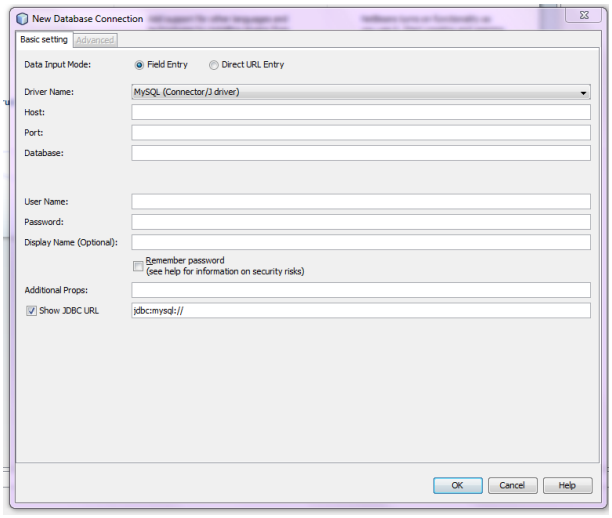

Figura 3.6: Nuova connessione a un Database in NetBeans

Come primo passo, bisogna scegliere il Driver Name in base al tipo di connessione che si vuole creare,quindi scegliere tra le opzioni esistenti quella di interesse, in questo caso MySQL(Connector/JDrive).
Come secondo passo bisogna inserire Host, in questo caso localhost, e Port, cioé il numero di porta utilizzato per la connessione al database, nel mio caso 3360.

Infine bisogna inserire i dati per l'autenticazione.
Per rendere effettivo l'accesso al database, bisogna importare una Libreria, e per questa connessione ho importato la libreria MySQL JDBC Driver, nello specifico mysql connector java 5.1.6 bin.jar.
Dopo aver completato questo passaggio, il database è importato e risulta visibile all'applicazione percui comparirà nel menu a tendina disponibile cliccando nella sezione Services e poi su Databases:

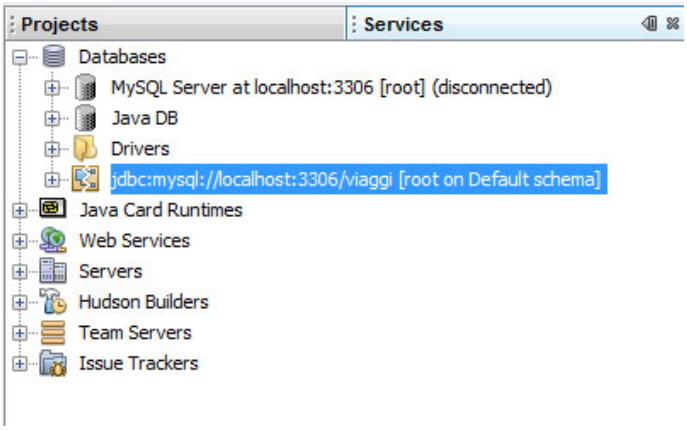

Figura 3.7: Sezione Services in NetBeans

Per connettersi al Database, bisogna selezionarlo e poi cliccare Connect, in questo modo sono visibili tutti i dati:

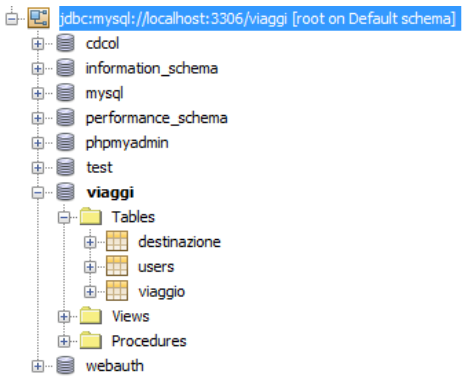

Figura 3.8: Particolari del Database

Se si vogliono fare delle interrogazioni al database per testarne le funzionalità o solo per visualizzarne i dati , è sufficente cliccare col tasto destro del mouse su una tabella, selezionare l'azione che si vuole compiere e verrà visualizzata una

schermata dove si scrivono i comandi in linguaggio SQL e si possono visualizzare i risultati, come nella figura 3.9:

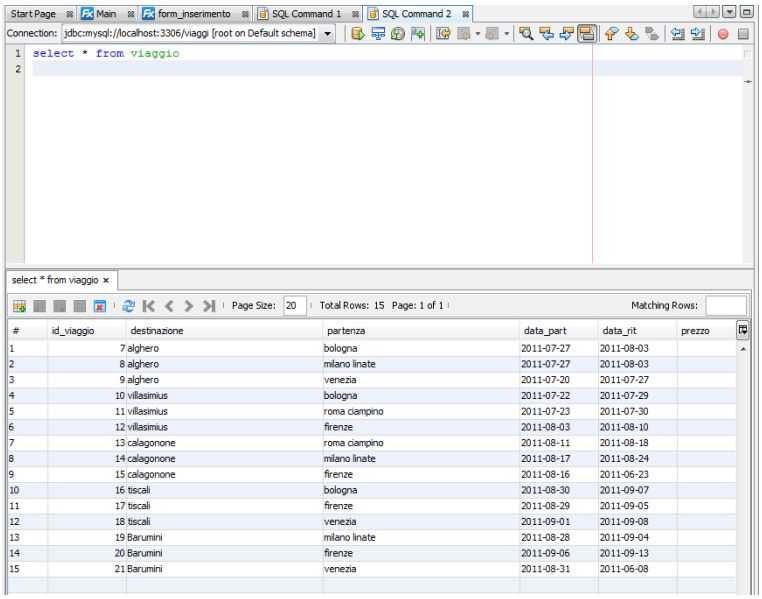

Figura 3.9: Funzioni SQL

Per interagire con i dati presenti nelle varie tabelle, ho creato la classe Database dove ho settato le variabili driverName, url, user, e password.
Nella classe sono presenti 2 funzioni, la funzione connetti(), e la funzione disconnetti() che possono essere richiamate in qualsiasi parte del programma per connettersi al database e quindi eseguire le query.

3.3 Schema delle classi

La creazione di un Custom Control, ovvero un controllo personalizzato, richiede più lavoro di quello richiesto per creare controlli sui nodi per renderli personalizzati, perché c'è bisogno di dividerei i controlli in 3 classi separate: il controllo stesso, l'aspetto e il comportamento.
Un controllo è logicamente composto da 3 parti: una parte che implementa le funzionalità del controllo, una parte che fornisca la rappresentazione visiva del controllo e una parte che gestisce l'interazione dell'utente con il controllo, cioé il comportamento.
Un pattern è uno schema di progettazione che permette la soluzione a problemi ricorrenti, si può definire anche come una specie di modello a cui ispirarsi per risolvere determinate difficoltà di progettazione del software. Il modello di pattern su cui mi sono basata per la modellazione del software è il pattern MVC Model View Control (ovvero Modello - Vista - Controllore).
Il pattern MVC, è un pattern architetturale largamente utilizzato per lo sviluppo di interfacce grafiche utente.
I pattern architetturali sono basati su schemi che aiutano il programmatore all'organizzazione strutturale del software. Questo pattern separa il programma in 3 sezioni distinte:

- Model: le variabili contenute in questa classe consentono di specificare quali controlli devo essere fatti. Lo stesso controllo fornisce i "ganci" a cui il codice puó collegare le funzione di callback (ovvero le funzioni di ritorno) chiamate quando il controllo cambia stato.
  Il control, é anche un nodo, e le varibili pubbliche del suddetto nodo sono rese disponibili al programmatore.

- View: le variabili contenute in questa classe fornisconoi le informazioni che determinano l'aspetto visivo del controllo.

- Controlle: le funzioni di questa classe catturano gli eventi determinati dall'utente attraverso mouse o tastiera e traduce queste azioni in cambiamenti di stato nella sezione View

Nella figura 3.9 si puó osservare lo schema del Pattern MVC e le interazioni tra le classi che caratterizzano questo pattern.

Nel realizzare il progetto ho realizzato varie classi, ma per comodità e qualche problema ad interagire con il codice autogenerato ho unito il Model e il View nella stessa classe. Le classi realizzate sono:

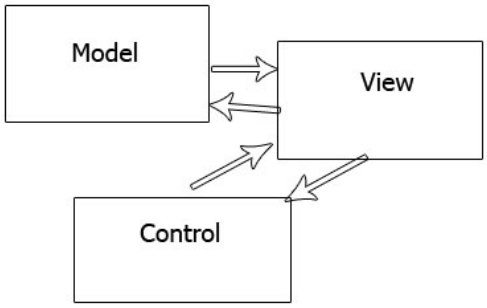

Figura 3.10: Schema Model-View-Control

- Main.fx : Questa classe contiene la quasi totalità del codice e ospita tutta l'interfaccia grafica e parte dei controlli.
- Database.fx : fornisce i metodi per connettersi al database di interesse, ed è visibile in qualsiasi parte del codice. Quando dev'essere stabilita una connessione con il database dalle function presenti nella classe Main, queste dichiarano:

```
var db : Database = new Database();
var stmt :Statement = db.connetti();
```

- Controller.fx : Questa classe contiene una funzione che si occupa di visualizzare il giusto Panel al verificarsi di un evento. Si può dire che assume il ruolo della classe View del Mode-View-Control. La funzione contenuta in questa classe permette di settare la visibilità dei Panel, che sono:

  PanelHome Dove è contenuta l'interfaccia grafica della pagina principale dell'applicazione;

  PanelDinamico Dove viene visualizzato il dettaglio del viaggio selezionato dai risultati della ricerca personalizzata (ListView);

  PanelDestinazione In questo Panel viene visualizzato il dettaglio della destinazione selezionata tramite la ChoiceBox contenuta nel Menu "Destinazione" dellal MenuBar;

  PanelContatti In questo Panel viene visualizzata l'interfaccia grafica relativa alla pagina "Chi Siamo", che viene selezionato dal MenuBar;

  PanelChiSiamo Stesso meccanismo del PanelContatti, ma con i contentui relativi alle informazioni che l'agenzia mette a disposizione per contattarla.

All'apertura dell'applicazione, è stata impostato visibile il PanelHome, e ogni volta che c'è la necessità di rendere non visibile questo Panel e di renderne visibile un'altro viene interpellata la classe Controller. quando la calsse Controller viene richiamata dalle funzioni del Main, queste dichiarano:

```
var c :Controller = new Controller();
```

e per richiamare le funzioni

```
c.nome_funzione
```

Nell'immagine 3.9, si nota delimitata dalla linea azzurra tratteggiata, l'area dedicata ai Panel la cui visibilità viene gestita dalla funzione della classe Controller.fx:

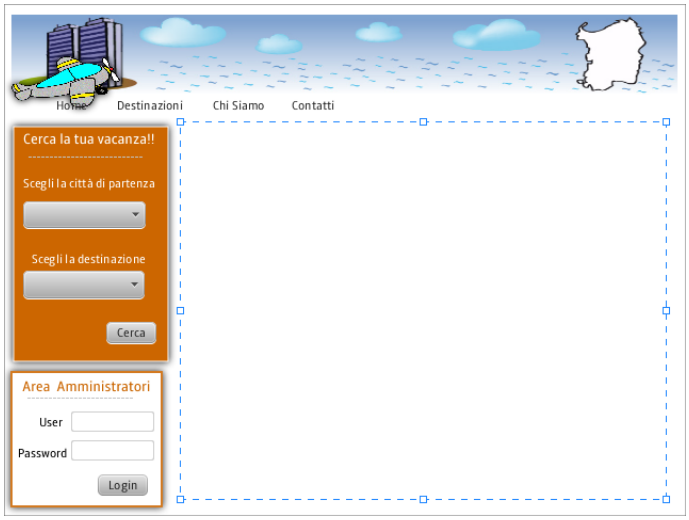

Figura 3.11: Area occupata dai Panel

- formGen.fx : Questa classe è stata creata per rappresentare la finestra che viene mostrata soltanto se viene effettuato l'accesso all'Area Amministratori. come spiegato in precedenza, gli Amministratori, hanno dei privilegi particolari, cioé possono aggiungere dati relativi a viaggi o destinazioni nel proprio database.
  Questi dati quindi potranno essere visualizzati nell'applicazione. Questa classe permette all'Amministratore di scegliere se inserire un nuovo viaggio

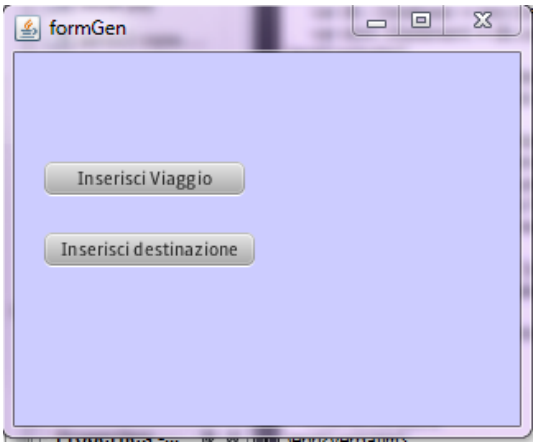

Figura 3.12: Interfaccia grafica della finestra riservata all'Area Amministrazione

o una nuova destinazione, e l'interfaccia grafica è visualizzata nell'immagine 3.12.
Nell'interfaccia grafica della finestra, sono presenti due Button su cui sono stati applicati dei controlli, percui cliccando su "Inserisci Viaggio" verrá poi interpellata la classe "form_inserimento". Se clicchiamo "Inserisci Destinazione" verrá richiamata invece la classe "form_destinazione".

- form_inserimento.fx: Questa classe contiene elementi che permetto all'amministratore di inserire un nuovo viaggio compilando i campi del form:

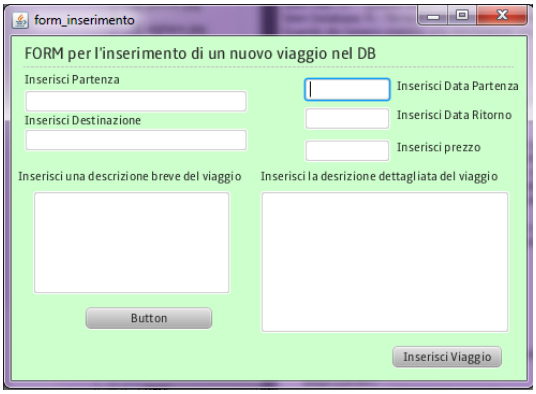

Figura 3.13: Interfaccia grafica della finestra che permette di inserire nuovi Viaggi al Database

- form_destinazione.fx : Questa classe contiene elementi che permetto all'amministratore di inserire una nuova destinazione compilando i campi del form:

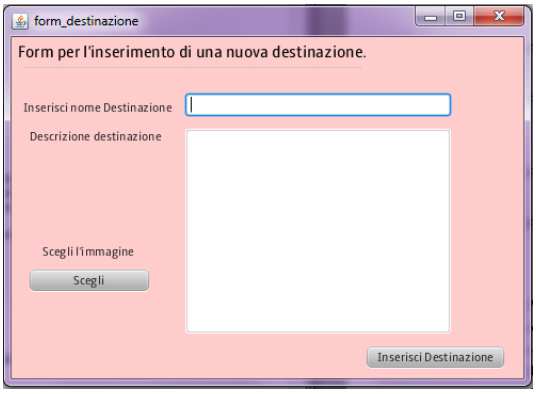

Figura 3.14: Interfaccia grafica della finestra che permette di inserire nuove Destinazione al Database

Per capire meglio come sono organizzate le classi, nella figura 3.15 viene rappresentato il diagramma delle classi dell'applicazione da me creata utilizzando il linguaggio UML.
Questo diagramma rappresenta gli elementi che compongono l'applicazione attraverso un grafo nella quale vengono illustrate le classi e le relazioni che intercorrono tra queste Una classe viene rappresentato con un rettangolo suddiviso in 3 sezioni:

Nome della classe in cui viene inserito appunto il nome che è stato assegnato alla classe;

Attributi Una sezione opzionale nella quale vengono elencati gli attributi piú importanti della classe;

Operazioni Questa sezione contiene le funzioni piú significative presenti nell'applicazione;

Le relazioni che intercorrono tra le classi del mio progetto sono 2:

- Associazione: relazioni struttrali il cui nome indica il tipo di relazione che intercorre tra le 2 classi. Nel diagramma delle classi relativo al progetto, questo tipo di relazione esiste tra la classe Main e la classe formGen, e tra la classe formGen e le classi form_destinazione e form_inserimento.

- Dipendenza: relazione tra 2 classi che specifica il fatto che per una classe è necessario utilizzare la classe da cui dipende. Nel mio diagramma delle classi questa relazione intercorre tra la classe Main e le classi database e Controller.

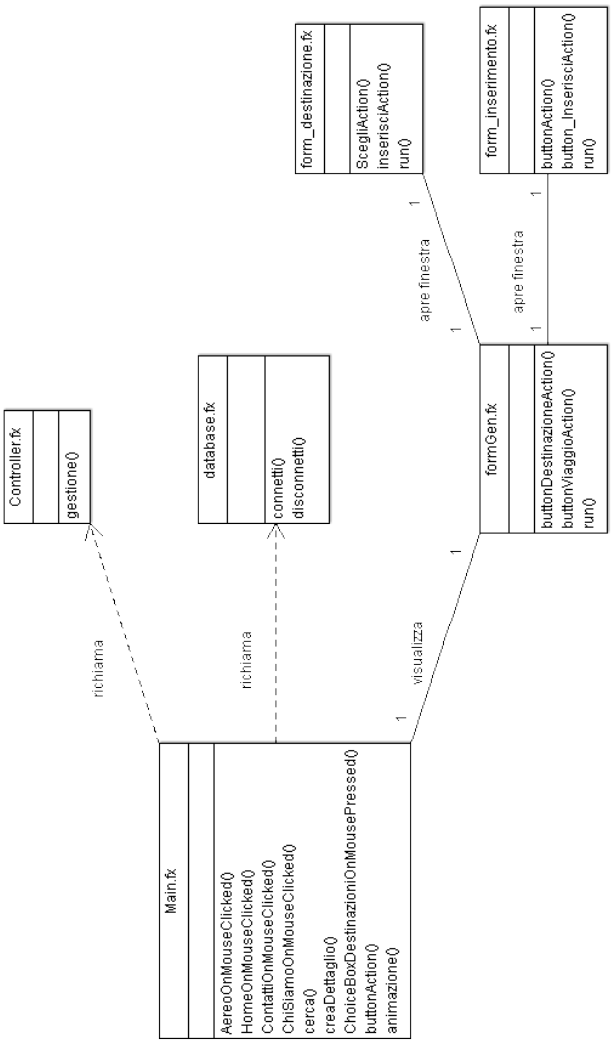

Figura 3.15: Diagramma delle Classi

3.4 Creare un'animazione

Creare un'animazione con JavaFX è stata la parte più divertente della realizzazione del progetto. JavaFX Script mette a disposizione del programmatore innumerevoli funzionii e variabili per gestire le animazioni. Le animazioni presenti nel progetto sono 2:

1. La prima viene avviata in automatico all'apertura dell'applicazione, ed é composta da 3 nodi principali rappresentati da delle fotografie.

2. La seconda viene avviata soltanto se si clicca sul nodo, in questo caso un aereoplano, che si muove seguendo un percorso che va da una parte all'altra della scena.

La prima animazione consiste in una sequenza di transizioni, all'apertura dell'applicazione viene visualizzata la prima foto, che poi si muove verso sinistra e quando raggiunge la sua posizione finale esegue un ingrandimento per poi tornare della sua dimensione originale. Cosi anche per la foto successiva, ma non per l'ultima che si trova già nella posizione finale e esegue solo l'ingrandimento.
Per realizzare l'animazione ho utilizzato la funzione" SequentialTransition" gestita dalla classe javafx.animation.transition.SequentialTransition che utilizza la variabile "content" di tipo Timeline[ ], nella quale si inseriscono le transizioni che si vogliono eseguire in sequenza, cioé una dopo l'altra.
Nell'applicazione realizzata ho utilizzato 2 tipi di transizione, la "TranslateTransition" che sposta un nodo da un punto della scena ad un'altro, e la ScaleTransition che aumenta o diminuisce le dimensioni del nodo stesso.
Per ingrandire e poi riportare alla grandezza originale il nodo, ho dovuto utilizzare 2 ScaleTransition, la prima per aumentare le dimensioni del nodo e la seconda per diminuirle e riportare l'immagine alla dimensione originale.
La sequenza intera di transizioni utilizzate é:

TranslateTransition porta la prima immagine da un punto ad un'altro punto specifico della scena ;

ScaleTransition ingrandisce la prima immagine;

ScaleTransition rimpicciolisce la prima immagine;

TranslateTransition porta la seconda immagine da un punto ad un'altro punto specifico della scena, affiancandosi alla prima immagine;

ScaleTransition ingrandisce la seconda immagine;

ScaleTransition rimpicciolisce la seconda immagine;

ScaleTransition ingrandisce la terza immagine;

ScaleTransition rimpicciolisce la terza immagine;

Nella figura la prima animazione in esecuzione

Figura 3.16: Prima Animazione in esecuzione

La seconda animazione consiste nel movimento di un aereo, da una parte della scena ad un'altra seguendo una traiettoria. Questa animazione viene avviata solo se si verifica un evento, il click del mouse sull'aereo.
Per creare questa animazione ho utilizzato la "PathTransition" gestita da javafx.animation.transition.PathTransition. Nel creare la path, ho specificato sia il punto di partenza che il punto d'arrivo della traiettoria grazie alle variabili "MoveTo " e "QuadCurveTo " con la quale ho potuto realizzare il percorso desiderato per l'animazione del nodo.
In seguito ho dichiarato una variabile chiamata "transition" di tipo PathTransition nella quale specifico i parametri che deve assumere l'animazione come l'orientamento del nodo, il nodo a cui viene applicata la Path e la durata dell'animazione stessa.
Nella parte alta della pagina principale dell'applicazione ho inserito, con un ImageViewer, un'immagine in cui viene raffigurata una cittá separata dal mare dalla Sardegna, meta principale di viaggi messi a disposizione dall'agenzia. L'aereo (nodo) seguirá una traiettoria tale da simulare il viaggio dalla cittá alla Sardegna e si fermerá proprio su quest'ultima.
Nella figura 3.17 la seconda animazione in esecuzione:

Figura 3.17: Seconda Animazione in esecuzione

3.5 Inserire l'applicazione JavaFX in una pagina Web

Prima di distribuire l'applicazione, é necessario creare un package. Si puó creare un package sia con l'ambiente di sviluppo NetBeans IDE o con javaFX Utility, incluso nel JavaFX SDK, infatti entrambi i metodi generano i file appropriati per il tipo di applicazione che si sceglie.
É possibile incorporare l'applicazione in una pagina web copiando gli script che vengono generati quando si comprime l'applicazione (package) , e verrá eseguita in qualsiasi browser che sia abilitato alla piattaforma Java.
Quando l'utente accede una pagina web che contiene un'applicazione JavaFX, il browser lancia il runtime Java, che scarica e memorizza l'applicazione nella memoria cache e la visualizza come parte della pagina web. Sia le applicazioni java che quelle javaFx, prima di essere implementate devono essere compresse e quindi trasformate in package. Un package contiene:

1. Un file JAR, cioé un file compresso che contiene il codice dell'applicazione;
2. Un file JNLP;
3. Un file HTML che contiene uno script che puó essere copiato in qualsiasi pagina web per richiamare l'applicazione.

NetBeans fornisce un'interfaccia utente che permette di specificare le modalitá della creazione del package. Quando l'applicazione dev'essere scaricata sul sitema dell'utente, si puó cercare di diminuire il tempo di avvio comprimendo il file JAR dell'applicazione stessa.
L'applicazione poi vengono memorizzate nella memoria cache degli utenti, percui il download è necessario solo la prima volta che si esegue l'applicazione. Se durante il processo di packaging si impostano le variaibli di configurazione correttamente e si mettono i file JAR e JNLP nella stessa directory, non c'è bisogno di apportare modifiche alla configurazione.
Per incorporare l'applicazione nella pagina web è sufficente copiare i 2 script (file Jar e JNLP) e incollarli nella pagina dove si vuole visualizzare l'applicazione.

Conclusioni

Realizzare un sito web in javaFx, è stata un esperienza nuova ed interessante poiché ho avuto l'occasione di studiare un nuovo linguaggio e di testare le sue prestazioni.
Il lavoro di questa tesi consiste nel dare all'utente un'idea generale di cos'è JavaFX e delle particolari funzionalità che lo distinguono dagli altri linguaggi. Per poter dare un'idea di quali sono gli obiettivi di questo linguaggio, ho presentato l'ambiente in cui il linguaggio compete per lo sviluppo delle RIA e quali sono i vantaggi nell'utilizzarlo.
Oltre a capire come sia nato questo linguaggio e in che modo può essere usato (Desktop Application, Mobile, Tv), il lavoro di tesi presenta anche una panoramica sul linguaggio JavaFx Script.
In questa sezione si è cercato di dare all'utente un'idea delle funzionalità di JavaFX e degli elementi che lo caratterizzano, dedicando una parte anche alla teoria basilare e alla sintassi, resa più comprensibile dai numerosi esempi proposti.
Per quanto riguarda il progetto realizzato posso affermare che prestazioni dell'applicazione sono buone in quanto è possibile realizzare animazioni con pochi sforzi e integrare l'applicazione in una pagina web con estrema facilità.
La parte più ardua del progetto è stato far interagire i singoli componenti grafici con il Database, in particolare visualizzare i risultati della ricerca personalizzata. Nonostante inizialmente abbia fatto numerosi tentativi per importare e utilizzare la JTable della classe Swing di Java, ho riscontrato numerosi problemi dal fatto di aver utilizzato il JavaFx Composer poiché il codice autogenerato non è modificabile. Infatti realizzando una JTable e volendola inserire in un Panel, il fatto che la JTable non fosse visibile nell'interfaccia grafica realizzata con l'ausilio di JavaFX Composer, non permetteva al Panel di visualizzare il contenuto, in questo caso la JTable.
Dopo aver perso un po di tempo a trovare una soluzione a questo problema ho scelto di utilizzare la ListView, e con questa è stato abbastanza semplice interagire con il database.

La nuova versione di JavaFX, JavaFx 2.0 include nelle sue nuove funzionalità la possibilità di costruire tabelle, invece non supportata dalla versione JavaFx 1.3 da me utilizzata. Dopo aver realizzato questo progetto, posso dire in conclusione di essere rimasta soddisfatta delle possibilità di JavaFx e che in futuro c'è l'intenzione di esplorare ulteriormente il linguaggio, ma questa volta non per le applicazioni web, ma per i dispositivi mobili.

Bibliografia

[1] http://it.paperblog.com/javafx-265808/

[2] http://javafxitalia.pbworks.com/w/page/7171620/Introduzione-a-JavaFX

[3] http://www.datamanager.it/news/open-source/sun-annuncia-java-fx-mobile

[4] http://www.ilbloggatore.com/a1/2008/08/30/novita-javafx/

[5] http://www.adobe.com/it/resources/business/rich_internet_apps/

[6] http://www.cleartag.com/blog/web-design/rich-internet-applications-overview/

[7] http://www.adobe.com/it/resources/business/rich_internet_apps/benefits/

[8] http://www.pianetatech.it/open-source/rumors/oracle-open-source-java.html

[9] http://www.javapassion.com/portal/images/pdf_files/javafx/javafx_overview.pdf

[10] http://www.pianetatech.it/open-source/rumors/oracle-open-source-java.html

[11] http://knol.google.com/k/gian-angelo-geminiani/javafx-il-linguaggio/3cwrqwtlbbi7b/4#