

# Data Mining



Studente: Giuseppe Accardo

Matricola: 0124000879

Anno Accademico: 2016/2017

Docente: Prof. A. Ciaramella



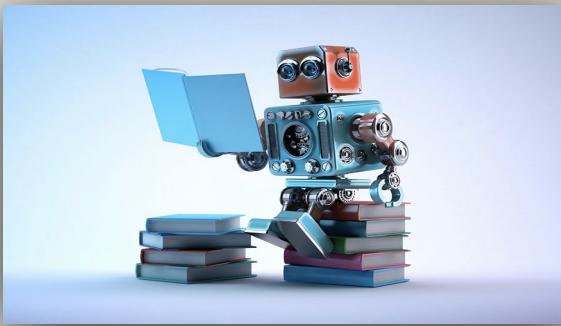
# Traccia e Descrizione del problema

# Traccia - Data Mining

- Si vuole sviluppare un sistema di **Data Mining** per l'elaborazione di dati. Il Data Mining comprende un insieme di tecniche e metodologie che hanno per oggetto l'estrazione e visualizzazione di informazioni da grandi quantità di dati (vedi sotto).
- Le fasi principali del sistema che si vuole sviluppare sono: *selezione delle caratteristiche, clustering e visualizzazione*.
- Si suppone di avere un data set contenuto in un file .data (vedi sotto). Il data set è relativo alla classificazione di 3 tipi di rose (setosa, versicolour, virginica) mediante sue quattro caratteristiche (feature): lunghezza e larghezza del sepalo, lunghezza e larghezza del petalo.
- Nella fase di selezione delle caratteristiche un utente può scegliere *il numero di caratteristiche da selezionare per l'analisi* (le colonne del data set). Nella fase di clustering viene usato un algoritmo per “agglomerare” dati simili. Nel caso specifico viene usato l'algoritmo **K-Means** (vedi sotto). L'utente può scegliere il numero di “cluster” da usare.
- Nella fase di visualizzazione i dati “agglomerati” sono visualizzati in 2 e 3 dimensioni. Nel caso in cui il numero delle feature è più grande di 3 viene applicato un algoritmo di *Analisi delle Componenti Principali* (vedi sotto) per la visualizzazione.

# Cos'è il Datamining?

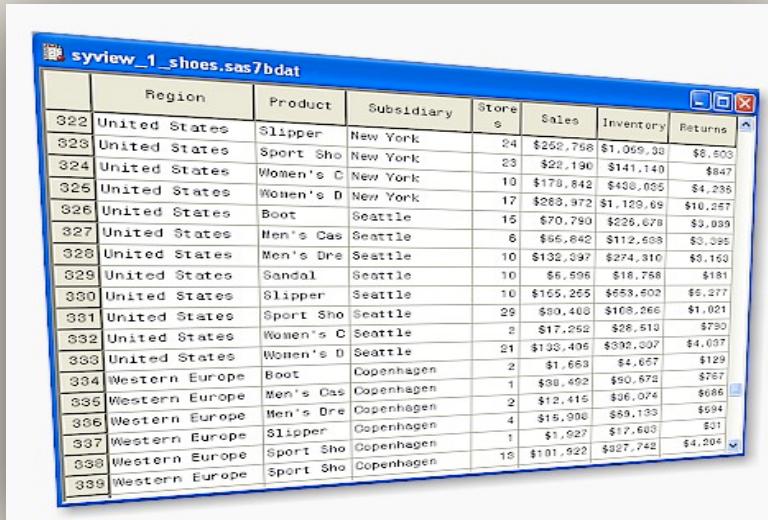
- linsieme di **tecniche e metodologie che hanno per oggetto l'estrazione di un sapere o di una conoscenza a partire da grandi quantità di dati** (attraverso metodi automatici) e l'utilizzo scientifico, industriale o operativo di questo sapere.
- I pattern identificati possono essere il punto di partenza per ipotizzare e quindi verificare nuove relazioni di tipo causale fra fenomeni; in generale, **possono servire in senso statistico per formulare previsioni su nuovi insiemi di dati.**



- Un concetto correlato al data mining è quello di apprendimento automatico: **Machine learning**, ossia l'identificazione di pattern può paragonarsi all'apprendimento di una relazione causale precedentemente ignota, cosa che trova applicazione in ambiti come quello degli algoritmi euristici e dell'intelligenza artificiale. Il processo di data mining è sempre sottoposto al rischio di rivelare relazioni causali che poi si rivelano inesistenti.

# Cos'è un Dataset?

- Un **dataset** (o data set) rappresenta una collezione di dati.
- Più comunemente un dataset costituisce un insieme di dati strutturati in forma relazionale (Matrice)
- La dimensione del dataset è data dal numero dei membri presenti (**osservazioni**), che formano le righe, e dal numero delle variabili di cui si compone (**features**), che formano le colonne.



The screenshot shows a SAS dataset viewer window titled "syview\_1\_shoes.sas7bdat". The table has the following structure:

|     | Region         | Product   | Subsidiary | Stores | Sales     | Inventory  | Returns  |
|-----|----------------|-----------|------------|--------|-----------|------------|----------|
| 322 | United States  | Slipper   | New York   | 24     | \$262,758 | \$1,069,38 | \$6,803  |
| 323 | United States  | Sport Sho | New York   | 23     | \$22,190  | \$141,140  | \$847    |
| 324 | United States  | Women's C | New York   | 10     | \$178,842 | \$438,055  | \$4,235  |
| 325 | United States  | Women's D | New York   | 17     | \$260,972 | \$1,129,69 | \$10,267 |
| 326 | United States  | Boot      | Seattle    | 15     | \$70,790  | \$226,678  | \$2,089  |
| 327 | United States  | Men's Cas | Seattle    | 6      | \$66,842  | \$112,688  | \$2,395  |
| 328 | United States  | Men's Dre | Seattle    | 10     | \$132,397 | \$274,310  | \$0,160  |
| 329 | United States  | Sandal    | Seattle    | 10     | \$6,595   | \$18,758   | \$181    |
| 330 | United States  | Slipper   | Seattle    | 10     | \$165,265 | \$553,502  | \$6,277  |
| 331 | United States  | Sport Sho | Seattle    | 29     | \$80,403  | \$165,266  | \$1,021  |
| 332 | United States  | Women's C | Seattle    | 2      | \$17,252  | \$28,513   | \$780    |
| 333 | United States  | Women's D | Seattle    | 21     | \$103,405 | \$302,307  | \$4,037  |
| 334 | Western Europe | Boot      | Copenhagen | 2      | \$1,663   | \$4,657    | \$129    |
| 335 | Western Europe | Men's Cas | Copenhagen | 1      | \$88,402  | \$80,872   | \$787    |
| 336 | Western Europe | Men's Dre | Copenhagen | 2      | \$12,415  | \$26,024   | \$686    |
| 337 | Western Europe | Slipper   | Copenhagen | 4      | \$15,900  | \$89,103   | \$31     |
| 338 | Western Europe | Sport Sho | Copenhagen | 1      | \$1,927   | \$17,603   | \$694    |
| 339 | Western Europe | Sport Sho | Copenhagen | 15     | \$101,922 | \$227,742  | \$4,204  |

# ...e Dataset Iris?

- Dataset multivariato introdotto da Ronald Fisher nel 1936.
- Consiste in 150 istanze di Iris (tipo di rose) misurate da Edgar Anderson e classificate secondo tre specie: *Iris setosa*, *Iris virginica* e *Iris versicolor*.
- Le quattro **features** (o caratteristiche) considerate sono *la lunghezza e la larghezza del sepalo e del petalo*.

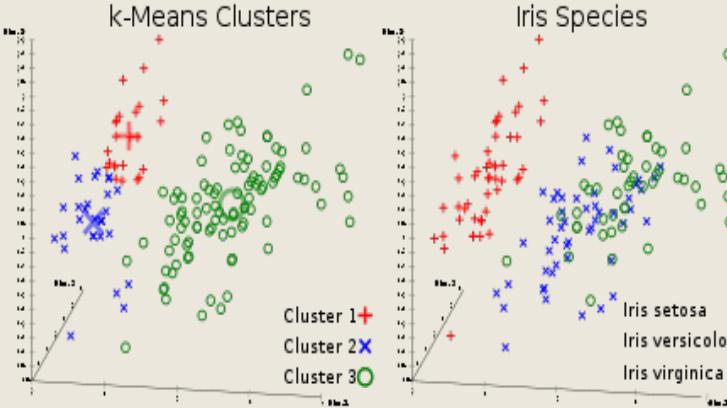
| Attributes |              |             |              |             |                |
|------------|--------------|-------------|--------------|-------------|----------------|
|            | sepal_length | sepal_width | petal_length | petal_width | Iris_class     |
|            | 5            | 2           | 3.5          |             | 1 versicolor   |
|            | 6            | 2.2         | 4            |             | 1 versicolor   |
|            | 6.2          | 2.2         | 4.5          |             | 1.5 versicolor |
|            | 6            | 2.2         | 5            |             | 1.5 virginica  |
|            | 4.5          | 2.3         | 1.3          |             | 0.3 setosa     |
|            | 5.5          | 2.3         | 4            |             | 1.3 versicolor |
|            | 6.3          | 2.3         | 4.4          |             | 1.3 versicolor |
|            | 5            | 2.3         | 3.3          |             | 1 versicolor   |
|            | 4.9          | 2.4         | 3.3          |             | 1 versicolor   |
|            | 5.5          | 2.4         | 3.8          |             | 1.1 versicolor |
|            | 5.5          | 2.4         | 3.7          |             | 1 versicolor   |
|            | 5.6          | 2.5         | 3.9          |             | 1.1 versicolor |
|            | 6.3          | 2.5         | 4.9          |             | 1.5 versicolor |
|            | 5.5          | 2.5         | 4            |             | 1.3 versicolor |
|            | 5.1          | 2.5         | 3            |             | 1.1 versicolor |
|            | 4.9          | 2.5         | 4.5          |             | 1.7 virginica  |
|            | 6.7          | 2.5         | 5.8          |             | 1.8 virginica  |
|            | 5.7          | 2.5         | 5            |             | 2 virginica    |
|            | 6.3          | 2.5         | 5            |             | 1.9 virginica  |
|            | 5.7          | 2.6         | 3.5          |             | 1 versicolor   |
|            | 5.5          | 2.6         | 4.4          |             | 1.2 versicolor |
|            | 5.8          | 2.6         | 4            |             | 1.2 versicolor |

Annotations:

- A yellow arrow points from the text "Attributes" to the first row of the table.
- A green arrow points from the text "Data point /example" to the 12th row of the table.
- A purple arrow points from the text "Numerical value" to the numerical values in the 12th row.
- A red arrow points from the text "Categorical value" to the "Iris\_class" column of the 12th row.

# K-MEANS: Cos'è e come funziona - 1

- Algoritmo che **raggruppa un insieme di oggetti in K gruppi (cluster) sulla base dei loro attributi**, ossia raggruppare per oggetti "simili".



- Ogni cluster viene identificato da un **centroide**, inizialmente in modo random o seguendo euristiche.
- Procedura iterativa: inizialmente crea K partizioni (K è l' input) e assegna ad ogni partizione i punti d'ingresso utilizzando la minima distanza euclidea (**MDM**).

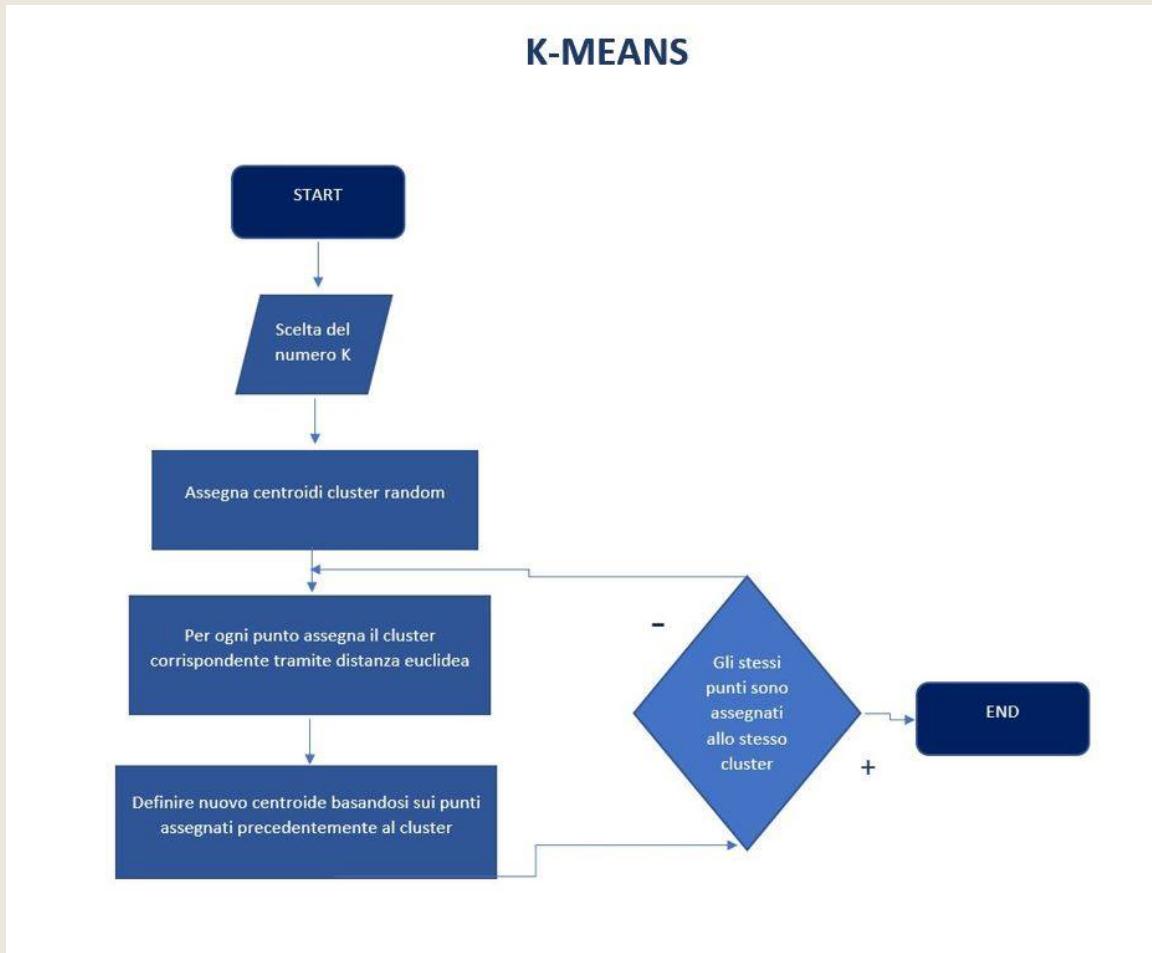
$$\text{objective function} \leftarrow J = \sum_{j=1}^k \sum_{i=1}^n \|x_i^{(j)} - c_j\|^2$$

Annotations pointing to parts of the equation:

- number of clusters  $\rightarrow k$
- number of cases  $\rightarrow n$
- case  $i$   $\rightarrow x_i^{(j)}$
- centroid for cluster  $j$   $\rightarrow c_j$
- Distance function  $\rightarrow \| \cdot \|^2$

# K-MEANS: Cos'è e come funziona - 2

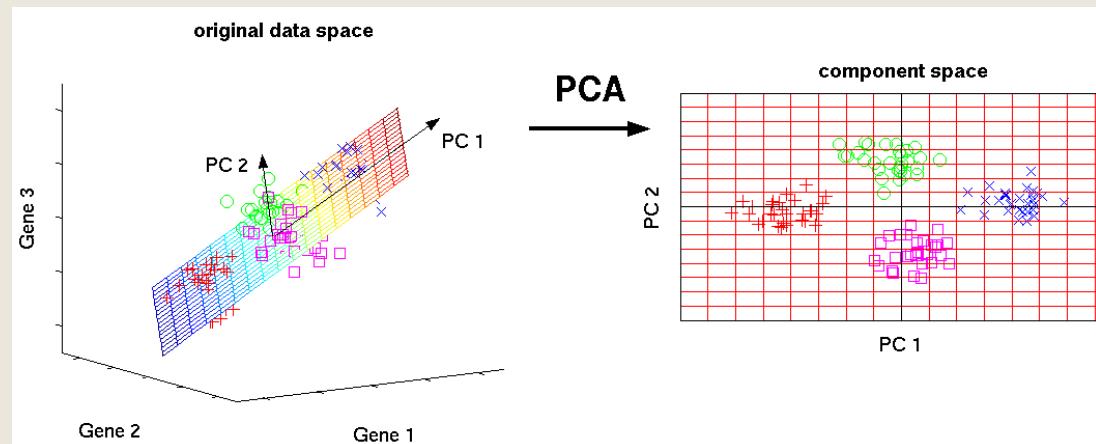
- Assegnati i punti, vengono ricalcolati i nuovi centroidi per i “nuovi” cluster come **la media dei punti calcolati per ogni cluster** e così via, finché l’algoritmo non converge.



- **Convergenza non sempre garantita:** ottenere numero di cluster inferiori da quelli richiesti in input K perché vengono scelti centroidi troppo vicini.

# Analisi delle Componenti Principali (PCA)

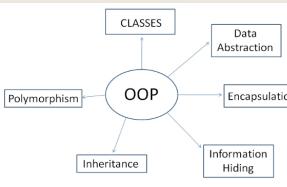
- Tecnica per la semplificazione dei dati utilizzata nell'ambito della statistica multivariata.
- Avviene tramite una trasformazione lineare delle variabili che proietta quelle originarie in un nuovo sistema cartesiano.
- Scopo primario di questa tecnica è la riduzione di un numero più o meno elevato di variabili (rappresentanti altrettante caratteristiche del fenomeno analizzato) in alcune variabili latenti (feature reduction). Infatti tale è **necessaria nel caso si volesse visualizzare un dataset con un numero di Features maggiori di 3**, poichè non sarebbe possibile effettuare la visualizzazione, senza perderne il contenuto informativo.



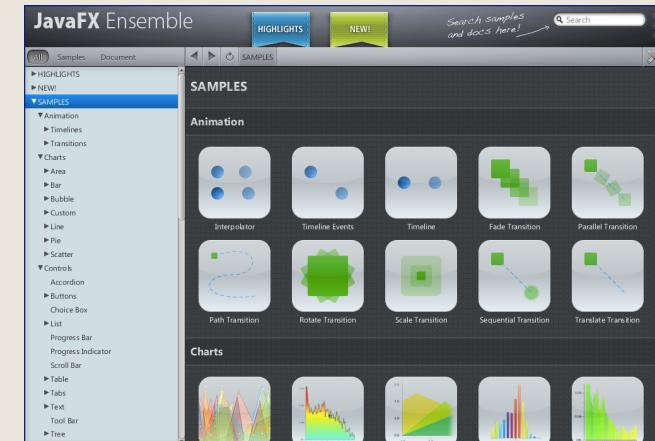
# Progettazione

# Analisi: prefazione sulla progettazione

- Il problema è analizzato in 2 macro-parti: **Back End e Front end**.
- Analisi e modellizzazione è effettuata utilizzando il *paradigma della programmazione orientata agli oggetti (OOP)*.



- Il linguaggio utilizzato per la progettazione è **Java 8 SE** (utilizzando anche nuovi costrutti spiegati più avanti), mentre è stato utilizzato unicamente **JavaFX** (oramai standard di java) per l'interfaccia grafica.



# Analisi: Back-End

*Forinire le strutture dati e le funzionalità necessarie alla creazione del Dataset e del Datamining.*

Il **Dataset** è caratterizzato dalle seguenti peculiarità:

- Informazioni fondamentali sul Dataset, come il nome e il nome delle features;
- Caricamento del dataset;
- Memorizzazione dai dati, cioè come verranno memorizzati i record del dataset.

Il **Data Mining** rappresenta un insieme di metodi e tecniche sintetizzati:

- Decisione del Dataset su cui operare;
- Selezione delle features, cioè decidere quali caratteristiche scegliere per la clusterizzazione;
- Clusterizzazione mediante l'algoritmo K-Means, indicando il numero K di cluster da creare.

NB la clusterizzazione non fornisce ancora la visualizzazione, perché dipenderà dal gestore del front-end plottare i vari grafici.

# Analisi: Front-End

*Fornire un'adeguata **GUI** in JavaFX per una maggior esaltazione della user experience.*

- Creazione di un menù per garantire l'interazione con l'utente;
- Applicazione di una grafica adeguata, cioè mediante una serie di applicazioni di fogli di stile e vari accorgimenti grafici;
- Visualizzazione del risultato del clustering, cioè la visualizzazione dei grafici 2D/3D utilizzati per l'output del DataMining.

# Novità java 8: Espressioni Lambda

- Rappresentano una **sintassi più semplice per definire-creare un'istanza di una classe anonima che implementa un'interfaccia con un solo metodo astratto** (interfaccia funzionale).

lambda-parametri -> lambda-corpo

Si noti lo speciale segno "->". Questa è praticamente equivalente a:

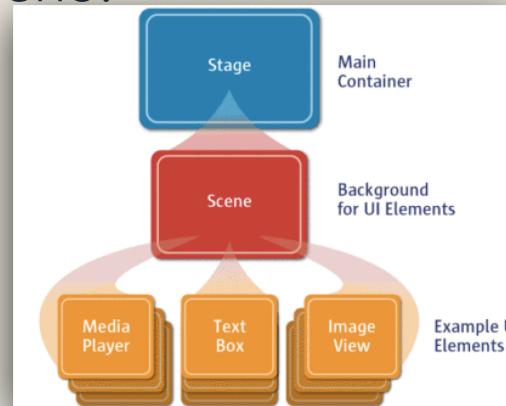
```
new NomeI() {  
    metodo(lambda-parametri) {  
        lambda-corpo  
    }  
}
```

In generale può essere utilizzata anche per riferimento a metodo o a costruttori mediante il simbolo '::::'che non sono anonime, ma che si riferiscono ad un specifico metodo di una determinata classe (o di una istanza). Di seguito un esempio completo:

```
// Senza Lambda  
List list1 = Arrays.asList(1,2,3,5);  
for(Integer n: list1) {  
    System.out.println(n);  
}  
/*foreach è metodo che accetta consumer, un'interfaccia funzionale che corrisponde all'azione da  
eguire per ogni elemento della lista*/  
//Lambda Anonime  
List list2 = Arrays.asList(1,2,3,5);  
list2.forEach(n -> System.out.println(n));  
//Lambda con Riferimenti anonimi,  
list2.forEach(System.out::println);
```

# Novità java 8: JavaFX

- **Framework**, divenuto std di Java 8, per lo sviluppo grafico.
- Spazio grafico JavaFX è un oggetto **Stage** (finestra), che può contenere uno o più oggetti **Scene**, che a loro volta confengono generici componenti grafici (**Node**) che presentano delle proprie proprietà (Property). Il concetto simile a quello del DOM html.
- Nodi organizzati in un linguaggi di Markup non compilato **FXML** che fornisce la struttura di un'interfaccia grafica separandola dal codice consentendo di assemblare interfacce grafiche.



Queste sono le principali peculiarità che rendono molto interessante JavaFX:

- Immediato uso del pattern Architetturale **MVC**;
- Supporta il formato e i file **CSS**;
- Supporto a **motore grafico 2D/3D**;
- Portabilità e multipiattaforma;

# Organizzazione e struttura progetto

La progettazione delle classi e la costruzione dell'applicazione è avvenuta utilizzando l'idee "Eclipse Java EE IDE Versione: Mars.1 (4.5.1)". Le principali librerie da linkare utilizzate per la progettazione sono:

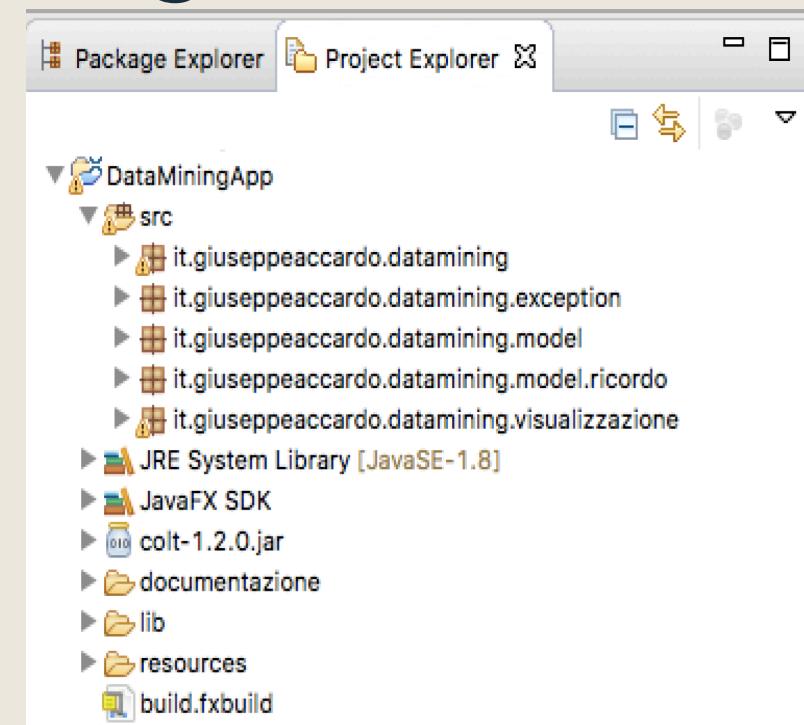
- **JavaFX SDK** per utilizzare javafx;
- **colt-1.2.0.jar**, sviluppata dal CERN (Organizzazione europea per la ricerca nucleare) necessaria per la costruzione della classe 'PCAcolt', indispensabile per applicare la PCA.

Nel Package Explorer di Eclipse sono presenti **5 package, resources** contenente dati e immagini e la cartella **Documentazione** contenente la documentazione generata da Javadoc con un totale di **20 file.java**.

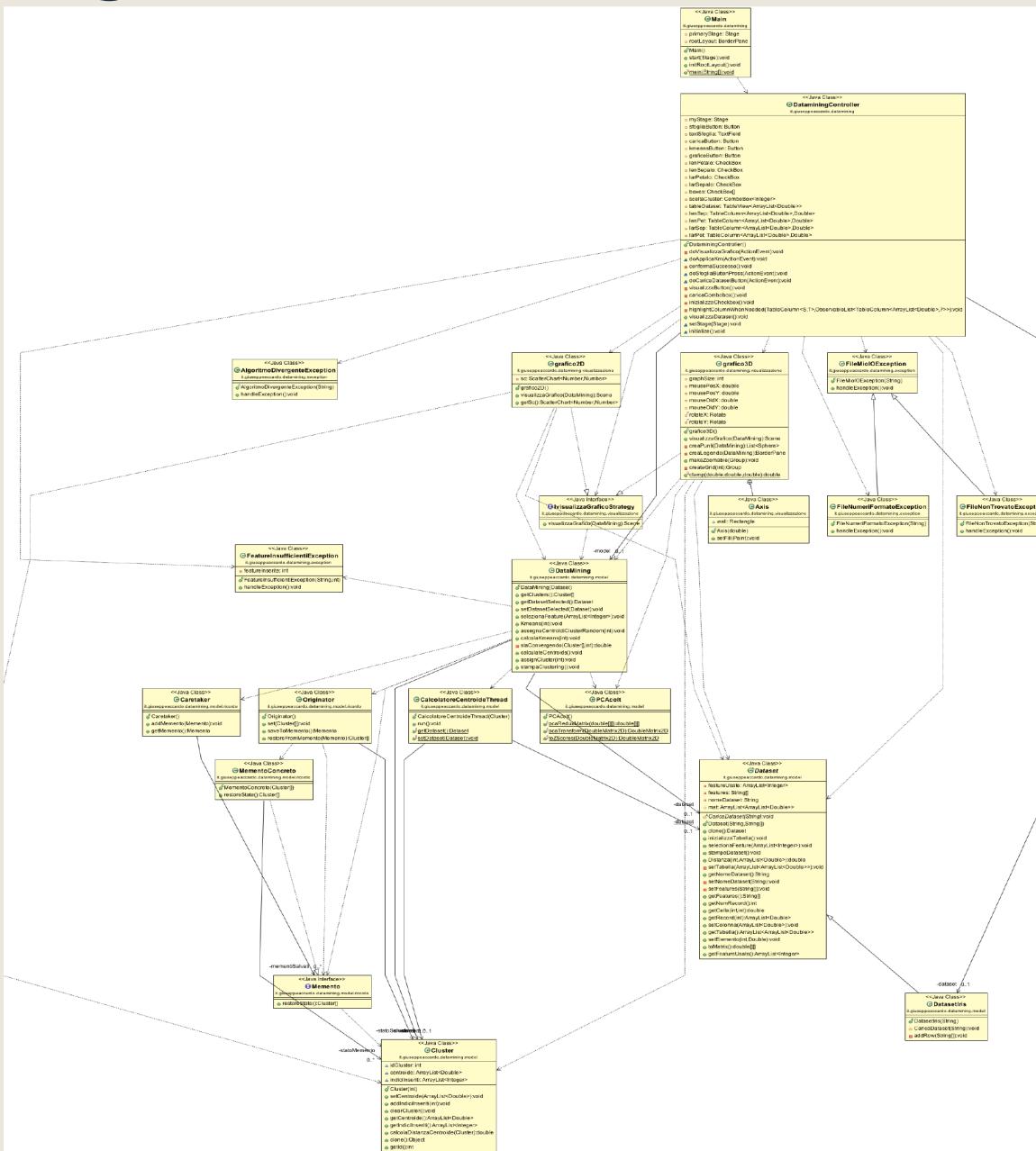
- Il primo package contiene file principali dell'applicazione appartenenti alla 'View' e 'Controller', insieme all'ultimo package per effettuare la visualizzazione grafica. Il secondo package è utilizzato per contenere particolari eccezioni utilizzate dal controller per la visualizzazione di alert o conferme.

I package che terminano con 'model' e 'model.ricordo' sono invece i 'Model' utilizzati dal controller per il funzionamento dell'intera applicazione.

Il progetto presenta applicazioni di 'Ingegneria del Software' mediante l'utilizzo dei principi **SOLID** e dei **Design Pattern**.



# Class Diagram UML completo



# Design Pattern utilizzati

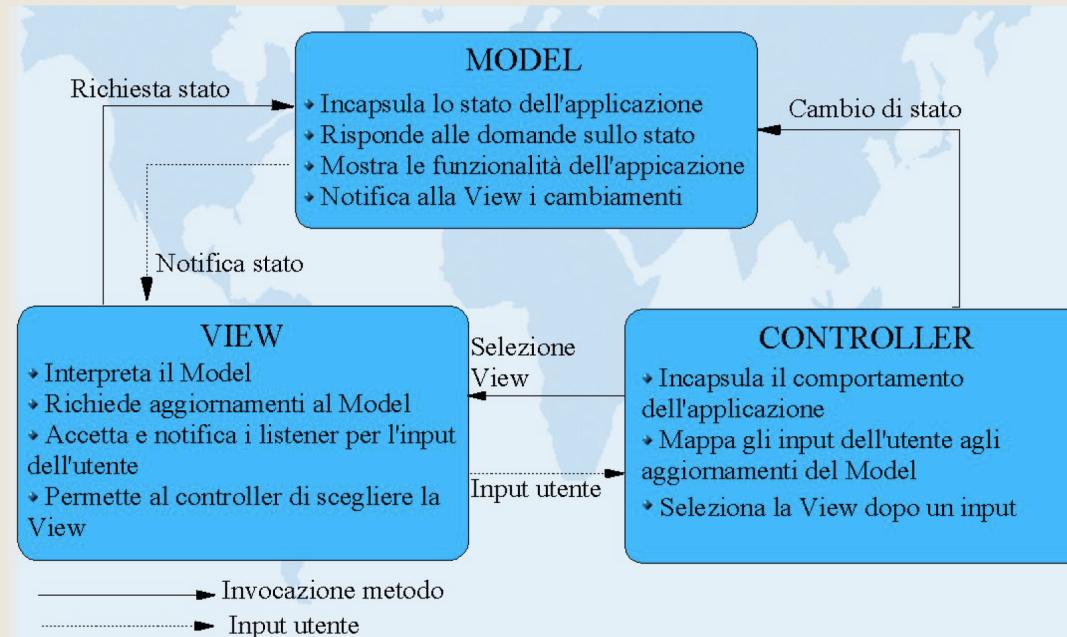
In totale sono stati utilizzati 6 Design Pattern (1 architettonale e 5 della GOF):

- **MVC**
- **Template Method**
- **Prototype**
- **Strategy**
- **Memento**
- **Observer**

# MVC

In totale sono stati utilizzati 6 Design Pattern (1 architetturale e 5 della GOF):

- Model-View-Controller (**MVC**) è un **Pattern Architetturale** (una soluzione progettuale) molto diffuso nello sviluppo di software (J2EE, .NET), in grado di separare la logica di presentazione dei dati dalla logica di business.
- Tale pattern è composto da 3 componenti che comunicano tra loro e sono: **Model, Controller e View**.



# MVC

Nell'applicazione creata, vengono assunti i seguenti ruoli ed implementazioni:

- **Model** è realizzato mediante le classi 'Dataset' e 'Datamining' poichè forniscono l'entry per accedere ai dati e di memorizzare uno stato applicativo. Quindi il funzionamento del DataMining e del K-Means è completamente trasparente all'utente, cioè a livello Back-End.
- **Controller** riceve i comandi dall'utente (attraverso view) e di mappa tutti gli aggiornamenti e cambiamenti di stato verso il model. Effettuate tale modifica, il controller selezionerà la view su cui voler mostrare i gli eventuali risultati del model o effettuare una determinata azione (layout dinamico). Il controller è realizzato mediante la classe DataminingController. L'MVC effettua una divisione comportamentale e logica delle componenti, quindi ad esempio parte di View sarà contenuta anche nel controller.
- La **View** si occupa di interpretare il model a livello di Front-end. Da essa possono partire delle richieste verso il controller. La View riceverà risposte (selezione) dal controller, ma può richiedere anche semplicemente lo stato direttamente al model o ricevere da esso una notifica di cambiamento di stato. La view è rappresentata dal layout sviluppato in FXML, comprendente di fogli di stile CSS e dal Main.
- Legame tra controller-view o view-model (**Binding**) è effettuato mediante il design pattern "**Observer**" per definire un forte dipendenza tra view e model in modo tale che se una cambia, l'altra è aggiornata automaticamente.  
In JavaFX è implementato direttamente mediante la collection interface '**ObservableList<E>**' che appunto tiene traccia di tutti i dati e dei suoi aggiornamenti.

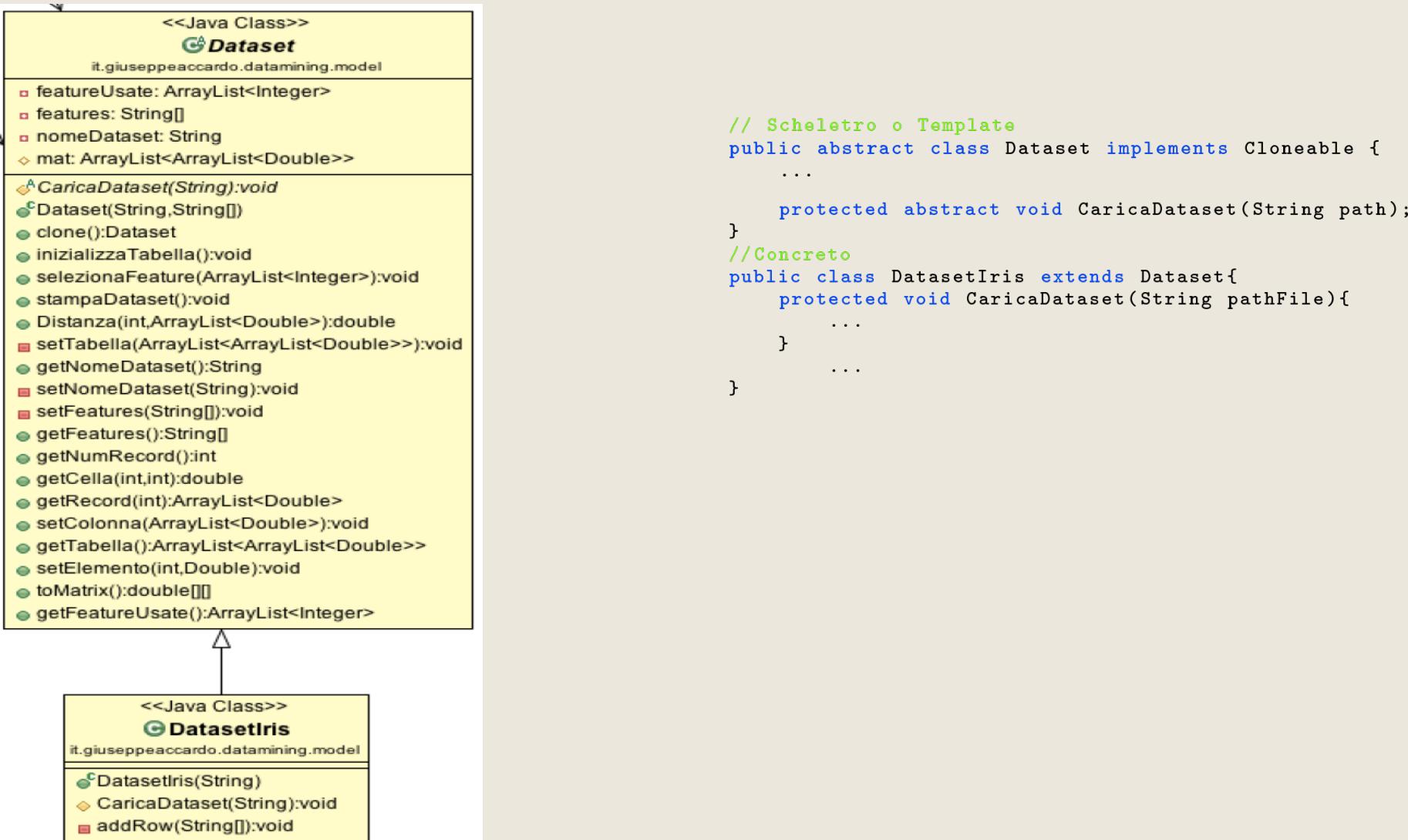
In conclusione, la scelta di costruire l'applicazione solo con JavaFX è perchè presenta una serie di vantaggi, tra cui la predisposizione alla costruzione di applicazioni seguendo l'MVC. Inoltre l'MVC garantisce una facile portabilità di sistema (Servlet ed ecc).

# Template Method

- Definire lo scheletro di un algoritmo in un'operazione, rinviano alcuni passi alle sottoclassi client. Consente alle sottoclassi di ridefinire alcuni passi senza cambiare la struttura dell'algoritmo.
- Quando si vuole implementare la parte invariante di un algoritmo una volta sola e lasciare che le sottoclassi implementino il comportamento che puo variare. Quando il comportamento comune di piu classi puo essere fattorizzato in una classe a parte per evitare di scrivere piu volte lo stesso codice.
- La necessità di applicare tale pattern nasce dal fatto che si vuole **creare uno "scheletro" o uno schema di una classe (framework)** che è il 'Dataset' e il dataset concreto, che in questo caso è 'DatasetIris' rappresenterà la sua implementazione concreta (ogni dataset concreto avrà il proprio caricamento).
- Dataset è posta in tal modo poichè si vuole rinviare la responsabilità di "**come caricare un dataset**" alla classe che estende dataset, utilizzando la stessa struttura di dataset. **Creare la possibilità di inserire futuri dataset concreti in modo semplice**. Altri design pattern che potevano essere applicati erano il "Factory Method" o "Decorator"

# Template Method

Il vantaggio principale di questo pattern è che garantisce una facile estensione e riuso del codice per realizzare altri eventuali dataset concreti.



# Prototype

- Specifica il tipo di oggetti da creare usando un'istanza prototipale e creando nuovi oggetti copiando questi oggetti
- Sistema indipendente da come i suoi prodotti sono creati, composti e rappresentati. Le classi da istanziare sono specificate a run-time per evitare di scrivere una gerarchia di classi. E' più conveniente copiare un'istanza esistente che crearne una nuova.
- Il "Prototype" è stato utilizzato per diversi compiti .Ad esempio è presente in Dataset poichè viene utilizzato (invocato) dal Datamining quando si **selezionano le features di un dataset poichè si ha la necessità di creare nuovi oggetti copiando il dataset non selezionato di partenza in tal modo da non perdere lo stato iniziale**. E' stato utilizzato anche per immagazzinare lo stato di un cluster.

# Prototype

- Il "Prototype" è stato utilizzato mediante l'implementazione dell'interfaccia di '**Cloneable**'.

```
// Implementazione di Cloneable
public abstract class Dataset implements Cloneable {
    ...
    @Override
    public Dataset clone(){
        try{
            /* Shallow copy --> Copia tutto ciò che resterà immutato ->reference
               Deep copy --> Crea le istanze che cambieranno->Valore */
            /* Crea un clone del dataset. Da notare che si necessita di effettuare anche un
               clone di mat che è una collection e
               * ciò lo renderà "Deep", necessario per la seleziona delle features senza
               modificare il dataset iniziale */
            Dataset datasetClone = ((Dataset) super.clone());
            datasetClone.setTabella( ((ArrayList<ArrayList<Double>>) datasetClone.getTabella()
                .clone()) );
            return datasetClone;
        }
        catch(CloneNotSupportedException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
            return null;
        }
        ...
    }
    ...
}

// Applicazione di datamining
/* Clona il DATASET di PARTENZA che dovrà essere 'immune' alla selezione */
Dataset datasetSel = (Dataset) dataset.clone();
/* Setta nuovo dataset clonato e di questo effettua la selezione delle feature richieste */
this.setSelectedDataset(datasetSel);
```

- La clonazione definita è di tipo "**Deep Copy**" perchè clone di default effettua una "**Shallow Copy**", ossia una copia del reference di oggetti o collection presenti.

# Strategy

- Definire una famiglia di algoritmi, incapsularli e renderli intercambiabili. L'algoritmo cambia indipendentemente dai client che lo usano.
- Classi correlate differiscono solo per il loro comportamento. Abbiamo bisogno di varianti di un algoritmo. Un algoritmo usa dati che i client non dovrebbero conoscere. Una classe definisce diversi comportamenti
- Lo "Strategy" è stato utilizzato da DataminingController per la visualizzazione di un corrispondente grafico 2D o 3D in base a delle determinate circostanze: il numero delle features selezionate.
- In sostanza, si è voluta creare **creare un'intercambiabilità di grafici 2D e 3D, ossia la necessità di modificare dinamicamente gli algoritmi.**

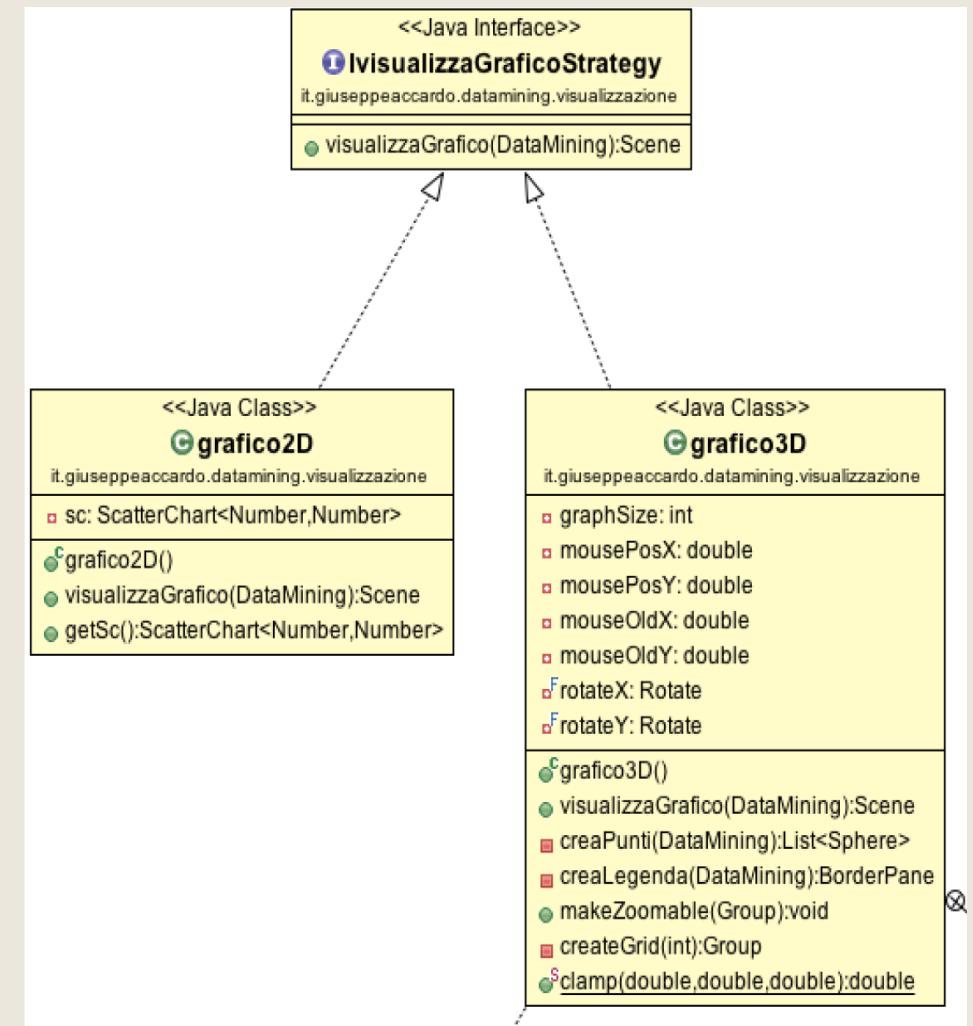
# Strategy

- In MVC è molto frequente l'utilizzo dello Strategy, per gestire la mappatura tra algoritmi che regolano il processo tra Controller e View.
- L'implementazione completa si trova nel Package “Visualizzazione”

```
// interfaccia per strategy
@FunctionalInterface
public interface IvisualizzaGraficoStrategy {
    public Scene visualizzaGrafico(DataMining model);
}

...
// Grafici che implementano strategy
public class grafico2D implements IvisualizzaGraficoStrategy {
    ...
    public Scene visualizzaGrafico(DataMining model){
        ...
    }
}
...
public class grafico2D implements IvisualizzaGraficoStrategy {
    ...
    public Scene visualizzaGrafico(DataMining model){
        ...
    }
}
...
//Applicazione di DataminingController
IvisualizzaGraficoStrategy grafico;
/* La scena conterrà il grafico corrispondente */
Scene sceneGrafico;
...
/* Istanziamo il grafico 2D o 3D, in base alle features utilizzate */
if(model.getDatasetSelected().getFeatureUsate().size()<3)
    grafico = new grafico2D();
else
    grafico = new grafico3D();

/* Con questo metodo otteniamo direttamente la scena corrispondente senza che sia il programmatore
   a doverlo riprogrammare */
sceneGrafico = grafico.visualizzaGrafico(model);
```

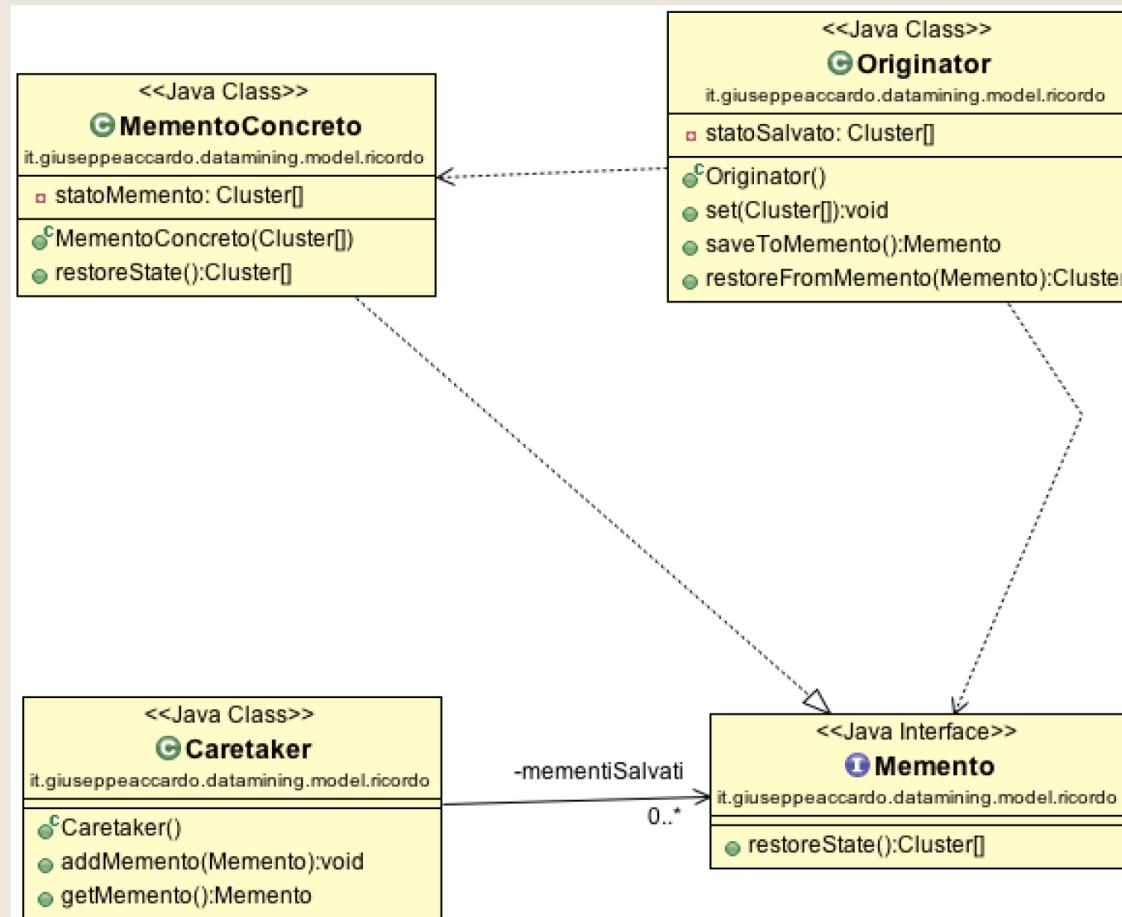


# Memento

- L'intento di questo modello è di catturare lo stato interno di un oggetto senza violare encapsulamento e fornendo così un mezzo per ripristinare l'oggetto allo stato iniziale quando necessario.
- viene utilizzato quando uno stato di un oggetto deve essere catturato in modo che possa essere ripristinato a quello stato più tardi. In situazioni in cui il passaggio in modo esplicito dello stato dell'oggetto violerebbe encapsulamento.
- Il design pattern "Memento" è stato utilizzato in Datamining durante la fase di clustering in cui viene usato un algoritmo per agglomerare dati simili, ossia il K-Means che prenderà in input il numero di cluster K su cui operare. In questo caso si ha la **necessità di dover salvare lo stato precedente dei cluster per poterli confrontare successivamente con lo stato successivo** e verificarne l'eventuale convergenza.

# Memento

- **Memento** rappresenta l'interfaccia che il implementa **ConcreteMemento** per immagazzinare lo stato generato da **Originator**. Quest'ultimo rappresenta l'istanza grazie alla quale viene generato uno stato fisico da salvare in Memento. Le istanze di ConcreteMemento saranno conservate in **Caretaker**.



# Memento

■ **Memento** è situato nel package “*it.giuseppeaccardo.model.ricordo*”.

```
// interfaccia per Memento
public interface Memento{
    public Cluster[] restoreState();
}

// Memento concreto
public class MementoConcreto implements Memento {
    /** Stato del memento */
    private Cluster[] statoMemento;
    /**
     * Registra uno stato di originator
     * @param statoDaSalvare stato che originator vuole salvare im memento
     */
    public MementoConcreto(Cluster[] statoDaSalvare) {
        this.statoMemento = statoDaSalvare;
    }
    /** Ripristina uno stato di originator
     * @return statoMemento ritorna lo stato di memento*/
    public Cluster[] restoreState() {
        return statoMemento;
    }
}
// Originator
public class Originator {
    /** Stato transitorio salvato */
    private Cluster[] statoSalvato;
    /*Setta lo stato da salvare*/
    public void set(Cluster[] statoDaSalvare) {
        statoSalvato = new Cluster[statoDaSalvare.length];
        for(int i = 0;i<statoDaSalvare.length; i++)
            statoSalvato[i] = (Cluster) statoDaSalvare[i].clone();
    }
    /**crea e restituisce il memento con lo stato salvato
     * @return MementoConcreto restituisce il memento concreto contenendo lo stato salvato
     *         in originator*/
    public Memento saveToMemento() {
        return new MementoConcreto(this.statoSalvato);
    }
    /**Ripristina uno stato da memento
     * @param m memento da cui ripristinare lo stato
     * @return statoSalvato ritorna lo stato ripristinato da memento*/
    public Cluster[] restoreFromMemento(Memento m) {
        statoSalvato = m.restoreState();
        return statoSalvato;
    }
}

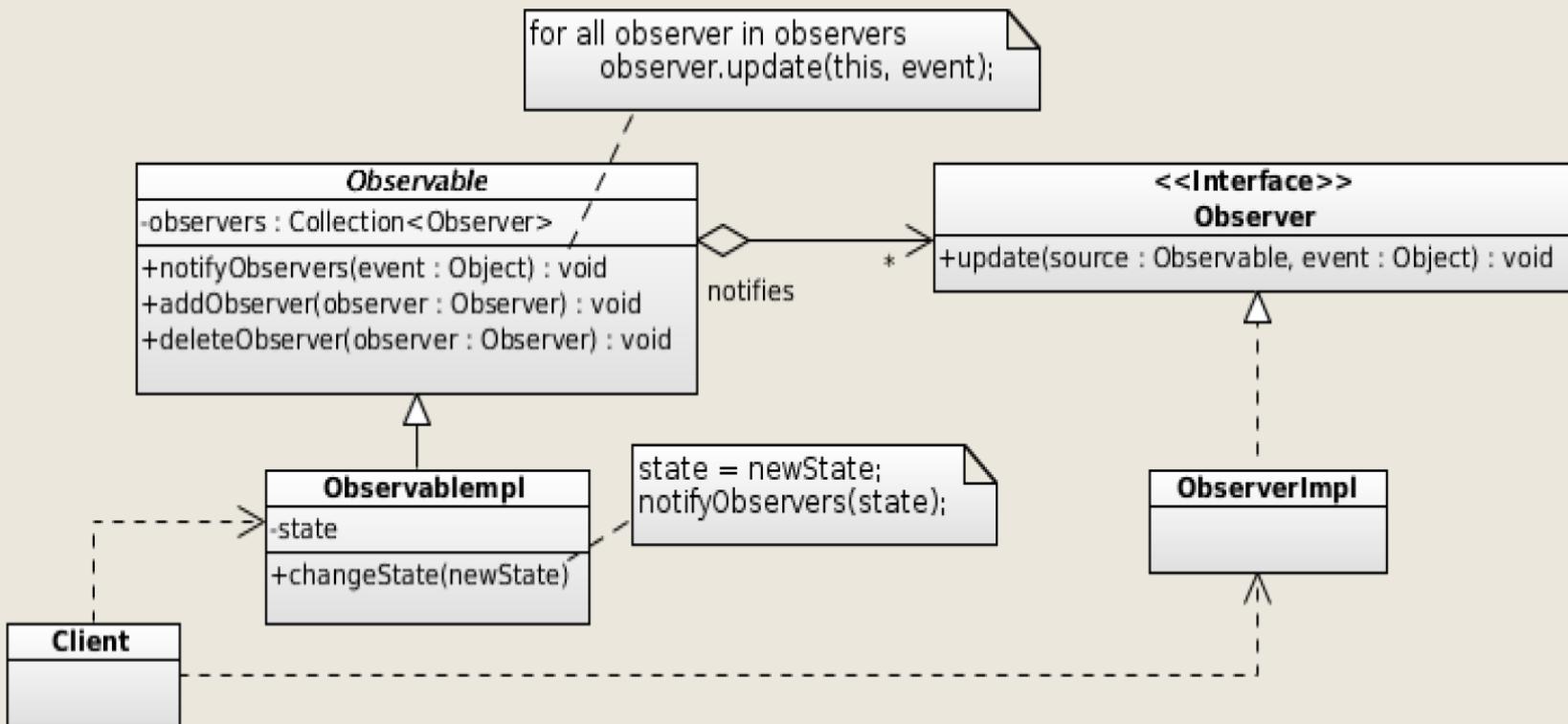
private Queue<Memento> mementiSalvati = new LinkedList<Memento>(); //queue
interfaccia
/** Aggiungi un memento nella Coda
 * @param m memento da salvare*/
public void addMemento(Memento m) {
    this.mementiSalvati.add(m);
}
/** Estrai memento dalla coda
 * @return memento*/
public Memento getMemento() {
    return this.mementiSalvati.poll();
}
}
...
...
// Salvataggio di un memento
/* Origina stato e custodiscilo nel memento */
originator.set(clusters);
caretaker.addMemento(originator.saveToMemento());
// Ripristino di un memento
/*Aggiorna lo stato originator e restituisce lo stato precedente*/
Cluster[] oldClusters = originator.restoreFromMemento( caretaker.getMemento());
```

# Observer

- Definisce una dipendenza una a molti tra oggetti, tale che se un oggetto cambia stato, tutte le sue dipendenze sono notificate e aggiornate automaticamente
- Quando un'astrazione ha due aspetti che dipendono l'uno dall'altro. L'incapsulamento di questi aspetti in oggetti separati permette il loro riuso in modo indipendente. Quando il cambiamento di un oggetto richiede il cambiamento di altri e non si conoscono quanti oggetti hanno bisogno di essere cambiati. Si puo usare nel- l'ambito comunicazione numeri
- In certe situazioni, **JavaFX obbliga l'utilizzo**, ad esempio in questi casi:
  - *La tabella che si relazione con il model;*
  - *cliccando su un checkbox, si aggiorna automatica mente lo stato delle colonne delle tabelle per essere così evidenziate (vedi ad esempio nella classe DataminingController).*

# Observer

- In JavaFX tale design pattern è implementato direttamente mediante la collection interface '**ObservableList<E>**' che appunto tiene traccia di tutti i dati e dei suoi aggiornamenti.
- Dalla documentazione di JavaFX: “*A list that allows listeners to track changes when they occur.*”



# Principi SOLID

- Consentono di avere un software estendibile e manutenibile, evitando di avere software rigido, difficile da cambiare, con cut and paste inutili e complesso (code smells).

Sono riassumibili in:

## 1. Single Responsibility Principle (SRP)

- Una classe dovrebbe avere un solo motivo per cambiare

## 2. Open Closed Principle (OCP)

- Le entità software come classi, moduli e funzioni dovrebbero essere aperte per l'estensione, ma chiuse per le modifiche

## 3. Liskov's Substitution Principle (LSP)

- I tipi derivati devono essere completamente sostituibili ai loro tipi base

## 4. Interface Segregation Principle (ISP)

- Un client non dovrebbe dipendere da metodi che non usa.

## 5. Dependency Inversion Principle (DIP)

- I moduli di alto livello non dovrebbero dipendere dai moduli di basso livello.

# Principi SOLID

I principi rispettati sono:

1. **Single Responsibility Principle (SRP)**: Ogni singola classe è costruita in modo tale da avere classi con singole responsabilità.

Ad esempio **visualizzazione grafica è una classe separata rispetto dalla classe Datamining**. Infatti in base al contenuto di un Datamining, viene generato in un apposita classe una nuova classe Visualizzazione che presenta delle responsabilità distinte.

2. **Open Closed Principle (OCP)**

Le classi sono molto più predisposte all'estensione, che alla modifiche delle stesse. Come detto in precedenza, utilizzando classi astratte o determinati pattern, come il Template o lo Strategy, riusciamo a garantire facilmente questo principio. Un ottimo esempio è il **Template Method utilizzato per fornire un Framework di Dataset in cui Dataset Iris non necessita di effettuare cambiamenti alla classe stessa, poichè è aperta all'estensione**.

Stessa considerazione vale per Visualizzazione poichè utilizza lo Strategy Pattern.

# Principi SOLID

## 3. Liskov's Substitution Principle (LSP)

Il principio di Liskov è ampiamente garantito.

Basta osservare che **le implementazioni concrete di Visualizzazione o Dataset, potrebbero essere completamente sostituite dal altri tipi derivati, senza perderne la logica**. Questo è garantito, oltre dai pattern applicati, dall'uso di interfacce e classi astratte. Rigidamente, anche `ArrayList<E>` possiede il reference di un `List<E>` poichè è la collection da cui eredita.

## 4. Interface Segregation Principle (ISP)

Tale principio è soddisfatto poichè **vengono utilizzate un numero consistente di interfacce che presentano pochi metodi**.

In particolare non sono presenti casi in cui è stato dovuto applicare (ad esempio un'interfaccia con tanti metodi scissa in più interfacce).

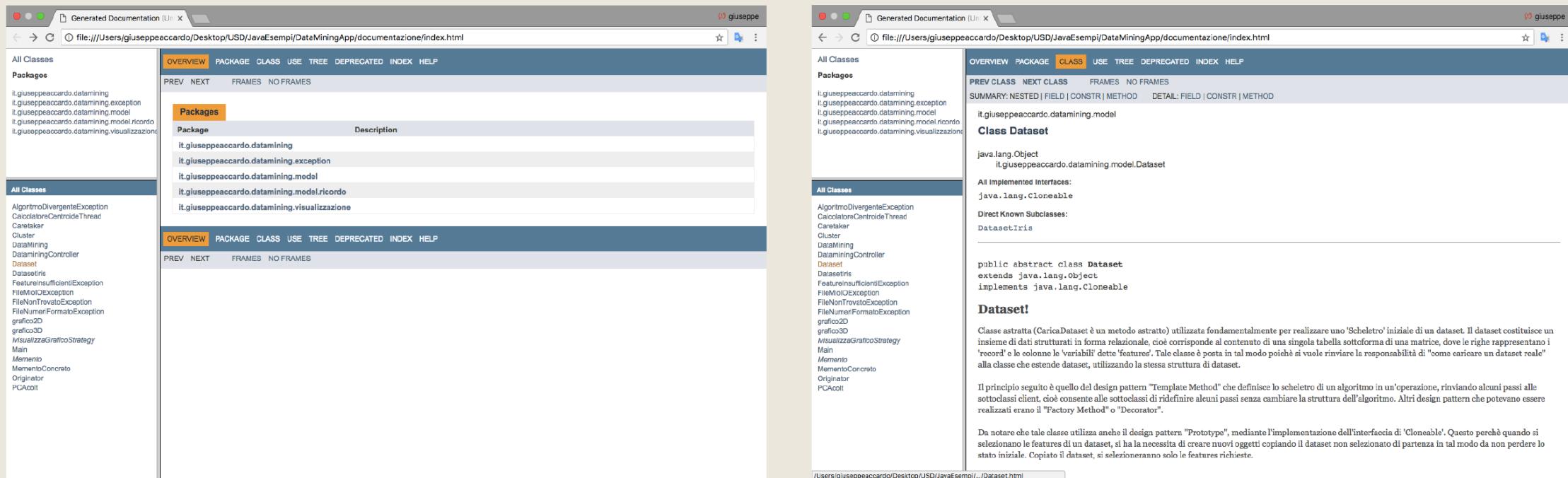
## 5. Dependency Inversion Principle (DIP)

I moduli di alto livello non dipendono da quelli di basso livello, mentre i dettagli si.

**Il caricamento del dataset viene effettuato da una sua implementazione concreta (Iris) che ne implementa un comportamento a basso livello, mentre dataset separa la logica di alto livello.** Un ottimo pattern da utilizzare sarebbe stato il DAO, non approfondito nel corso.

# Documentazione

- La documentazione comprende i Class Diagram, annotazioni e i commenti inseriti per package, classi, metodi e altro, questi vengono trasformati in documentazione da **javadoc** che 'e uno strumento che genera un insieme di pagine **HTML** tra loro collegate che rispecchiano la struttura del progetto. Si possono raccogliere informazioni navigando nella pagine della documentazione.



# Funzionamento della GUI

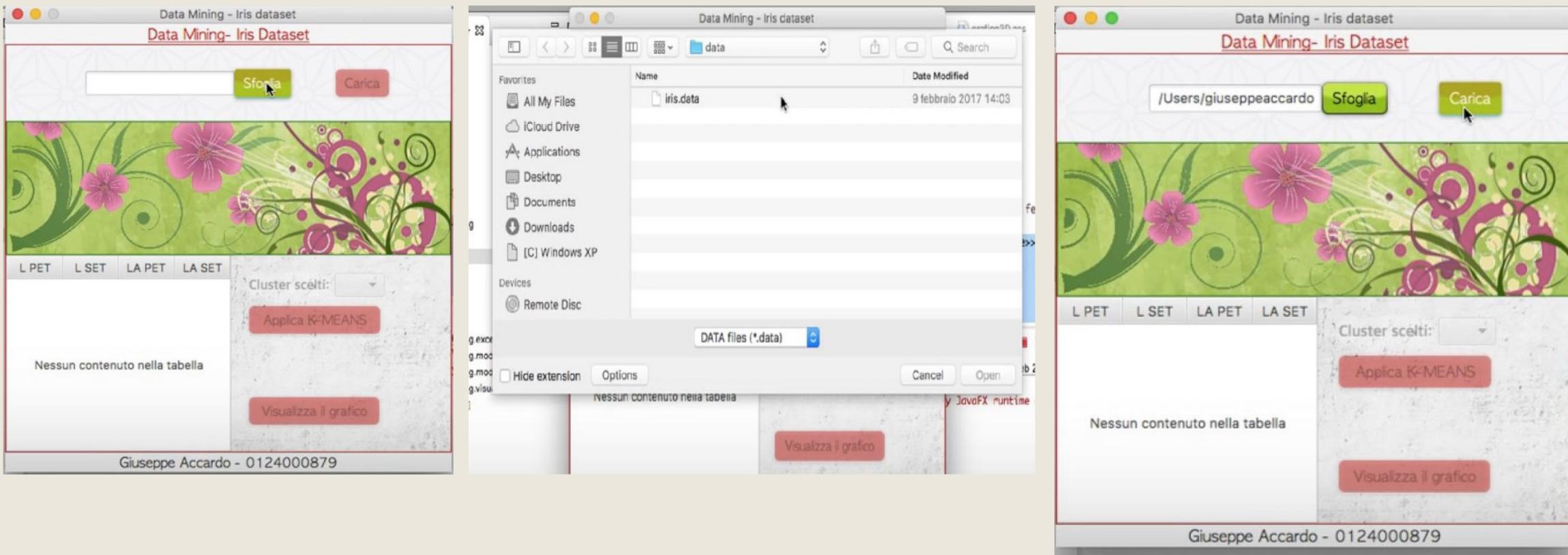
# User-Experience

- Al lancio dell'applicazione, viene visualizzato un menù realizzato interamente in **FXML** e **CSS**. La potenzialità di javaFX è di offrire un confortante e immediata interfaccia utente.



# Caricamento Dataset

- Si osservano la serie di passaggi da effettuare per caricare il dataset da File



# Visualizzazione Dataset

- Il caricamento del file, se è tutto ok, produrrà il dataset nel formato della Table-View e verranno visualizzate le possibili checkbox da selezionare.
- In questo caso ObservableList, tiene traccia o “osservazione” dei dati contenuti nel model (dataset).



# Selezione delle Features

- A questo punto vengono **selezionate le Features su cui vogliamo applicare il datamining mediante K-Means**, il numero di cluster K e si cliccherà su "Applica K-Means".
- I Per ogni checkbox selezionato, sarà evidenziata la corrispondente colonna (grazie all'ObservableList a cui sarà notificato l'evento).



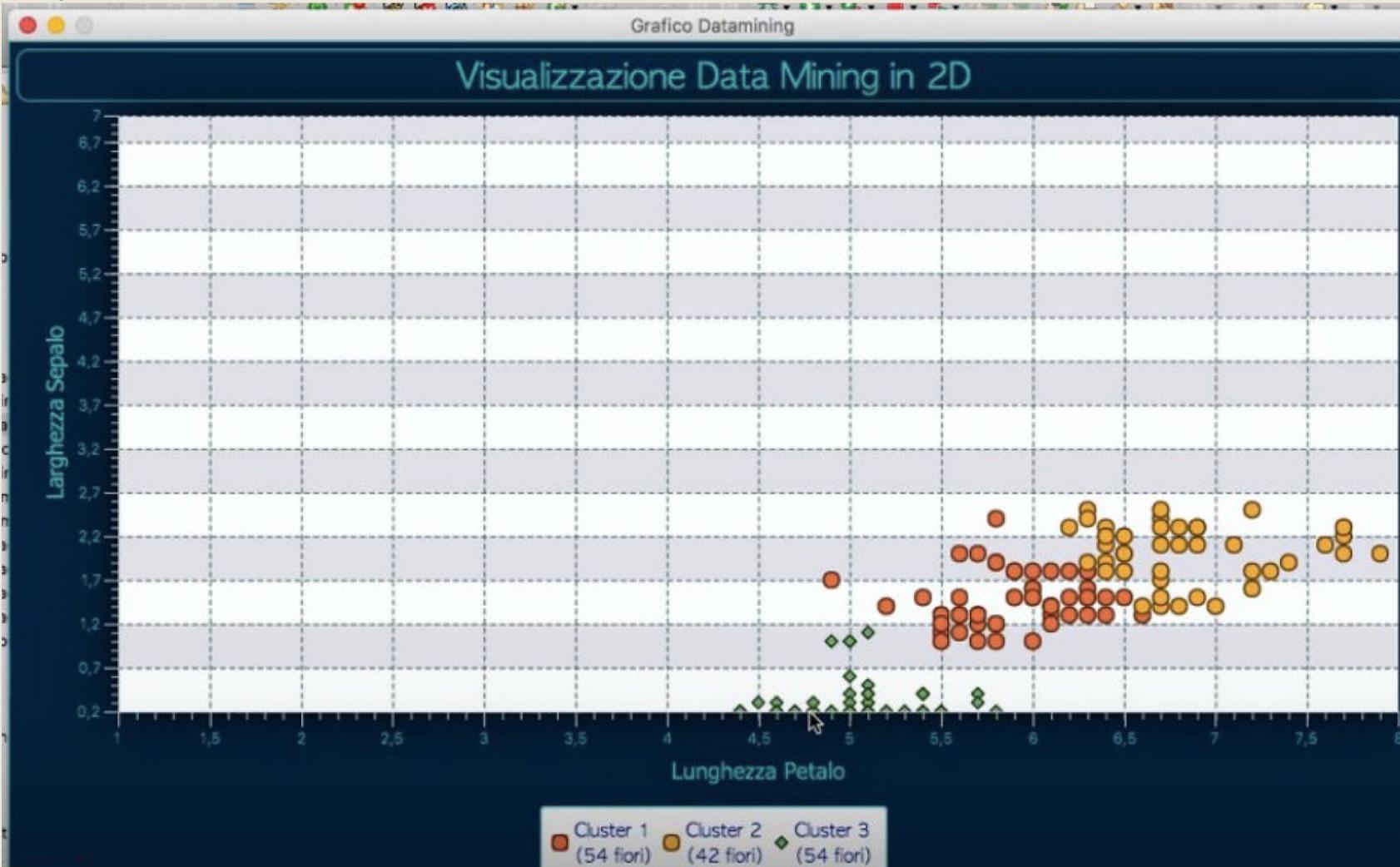
# Calcolo del K-Means

- Se il datamining andrà a buon fine, verrà visualizzato un **alert di successo**. A questo punto può essere visualizzato il risultato andando a cliccare su "**Visualizza Grafico**". Il grafico che ci si aspetta sarà in 2D poichè si sono scelte solo 2 Features. Si osservi come lo **strategy** renda trasparente l'intercambiabilità tra il grafico 2D e 3D



# Visualizzazione: Grafico2D

- I grafici 2D sono stati realizzati con la classe “**Chart**” di JavaFX
- Si osservi come i punti siano stati raggruppati, in base al colore dei loro corrispondenti cluster. Questo visualizzato è un ottimo risultato.



# Visualizzazione: Grafico3D

- Ora si effettua una prova selezionando 3 features per avere il risultato finale in un grafico 3D (ci si aspetta il solito lavoro dallo strategy)

Data Mining - Iris dataset  
Data Mining- Iris Dataset

/Users/giuseppeaccardo Sfoglia Carica

Selezione le features

Lunghezza Petalo  
Lunghezza Sepalo  
Larghezza Petalo  
Larghezza Sepalo

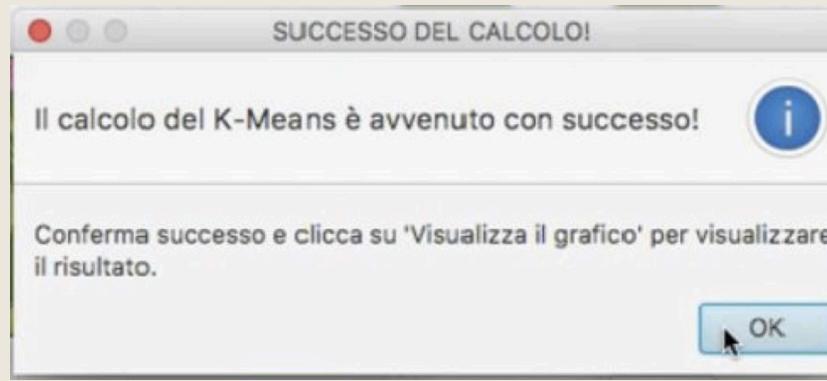
| L PET | L SET | LA PET | LA SET |
|-------|-------|--------|--------|
| 5.1   | 3.5   | 1.4    | 0.2    |
| 4.9   | 3.0   | 1.4    | 0.2    |
| 4.7   | 3.2   | 1.3    | 0.2    |
| 4.6   | 3.1   | 1.5    | 0.2    |
| 5.0   | 3.6   | 1.4    | 0.2    |
| 5.4   | 3.9   | 1.7    | 0.4    |
| 4.6   | 3.4   | 1.4    | 0.3    |
| 5.0   | 3.4   | 1.5    | 0.2    |

Cluster scelti: 3

Applica K-MEANS

Visualizza il grafico

Giuseppe Accardo - 0124000879



Data Mining - Iris dataset  
Data Mining- Iris Dataset

/Users/giuseppeaccardo Sfoglia Carica

Selezione le features

Lunghezza Petalo  
Lunghezza Sepalo  
Larghezza Petalo  
Larghezza Sepalo

| L PET | L SET | LA PET | LA SET |
|-------|-------|--------|--------|
| 5.1   | 3.5   | 1.4    | 0.2    |
| 4.9   | 3.0   | 1.4    | 0.2    |
| 4.7   | 3.2   | 1.3    | 0.2    |
| 4.6   | 3.1   | 1.5    | 0.2    |
| 5.0   | 3.6   | 1.4    | 0.2    |
| 5.4   | 3.9   | 1.7    | 0.4    |
| 4.6   | 3.4   | 1.4    | 0.3    |
| 5.0   | 3.4   | 1.5    | 0.2    |

Cluster scelti: 3

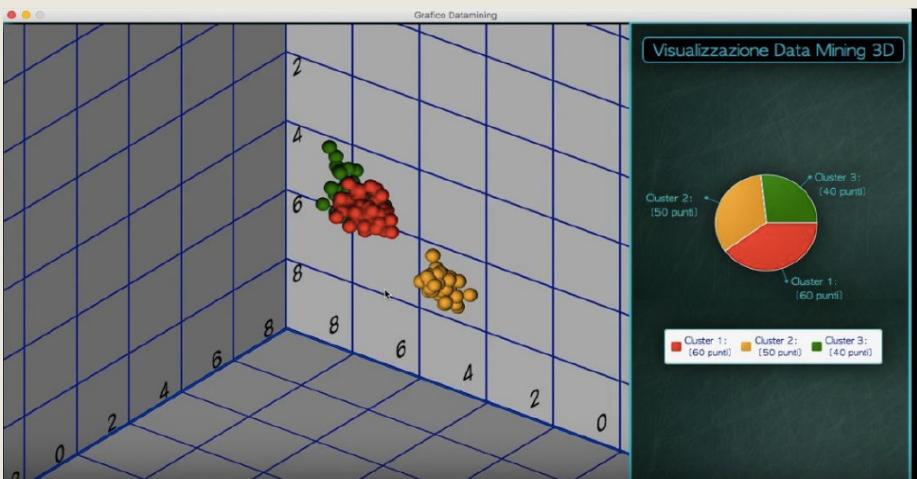
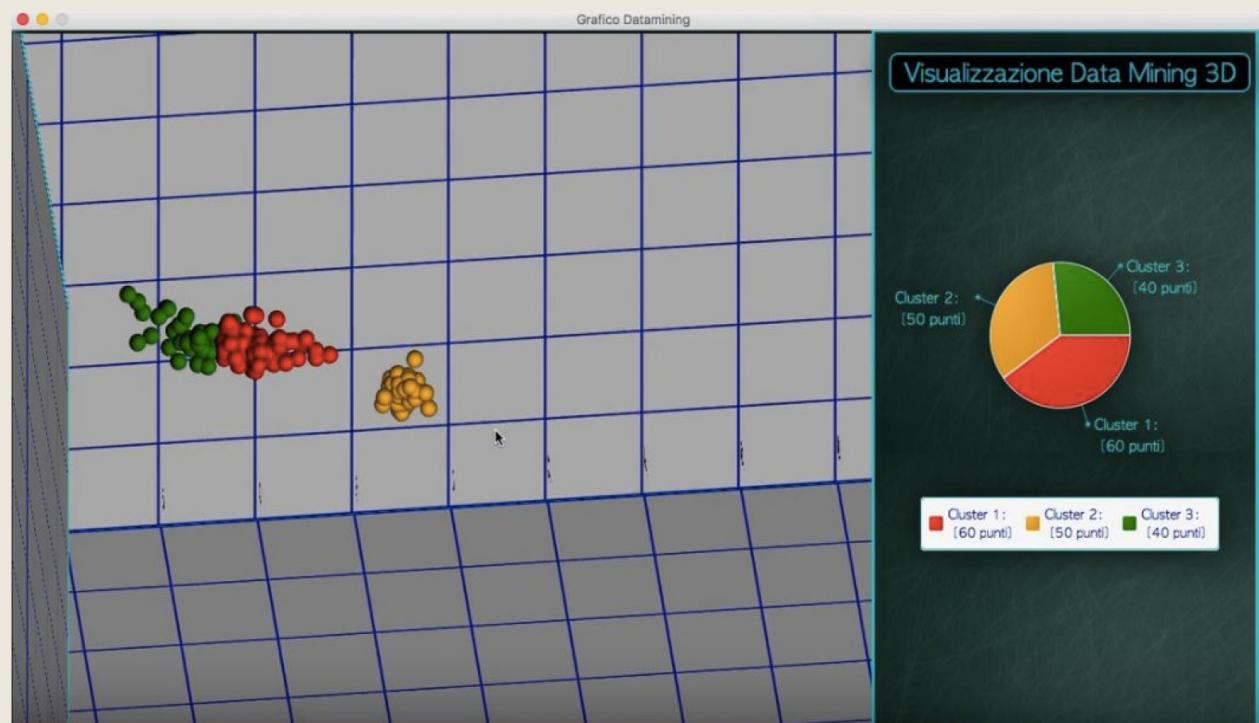
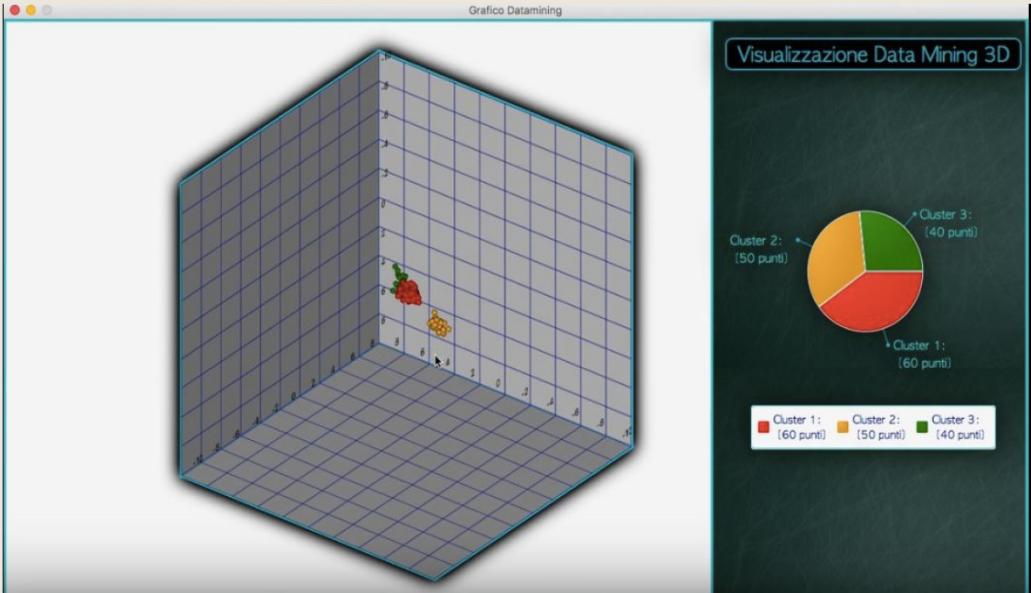
Applica K-MEANS

Visualizza il grafico

Giuseppe Accardo - 0124000879

# Visualizzazione: Grafico3D

- Per il 3D è stato usato il motore grafico di javaFX mediante le classi **Shape3D**



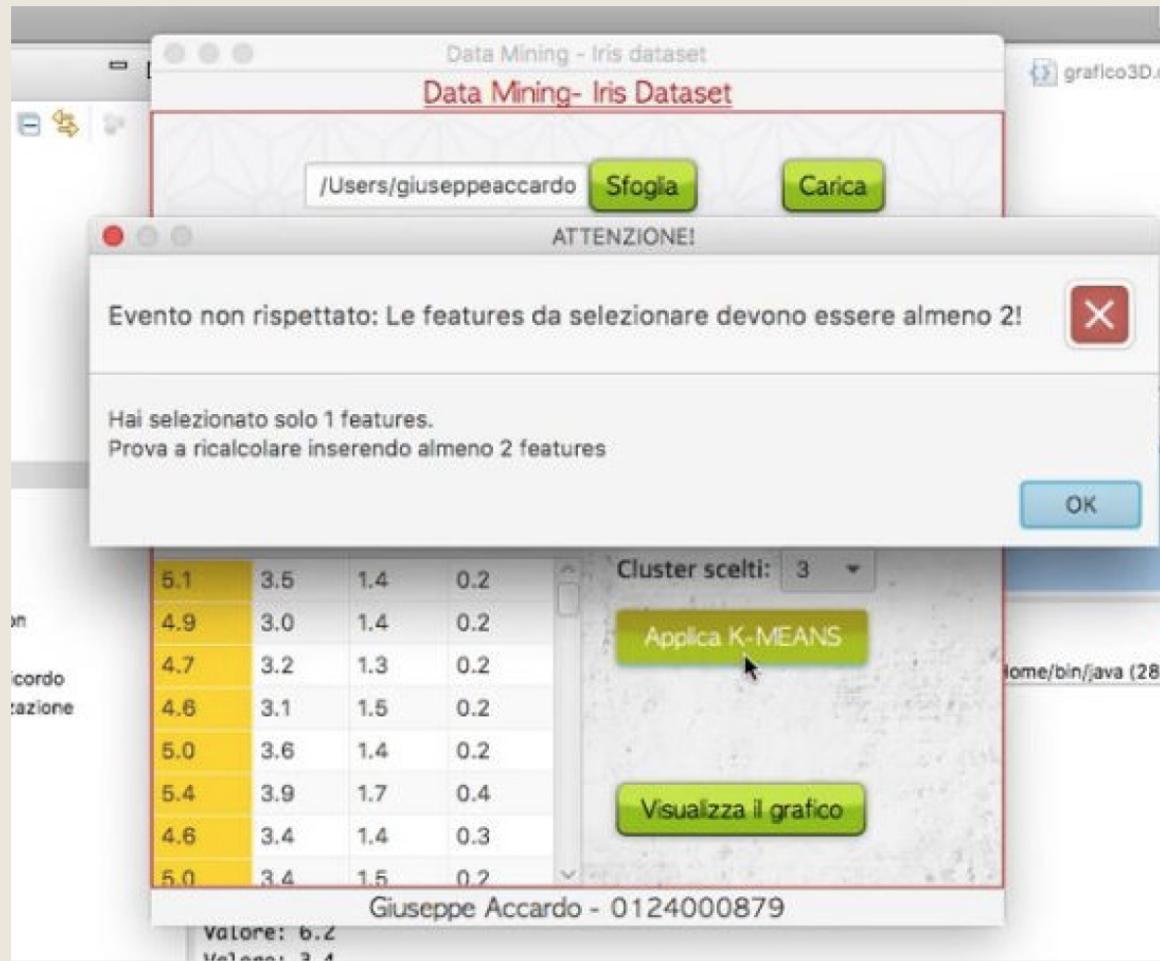
# Eccezioni: gestione anomalie

- kmeans potrebbe generare un numero di cluster inferiore a quelli richiesti poichè potrebbero essere scelti dei centroidi "troppo vicini" che non riescono a creare il gruppo di cluster.
- L'eccezione lanciata è "**AlgoritmoDivergenteException**".



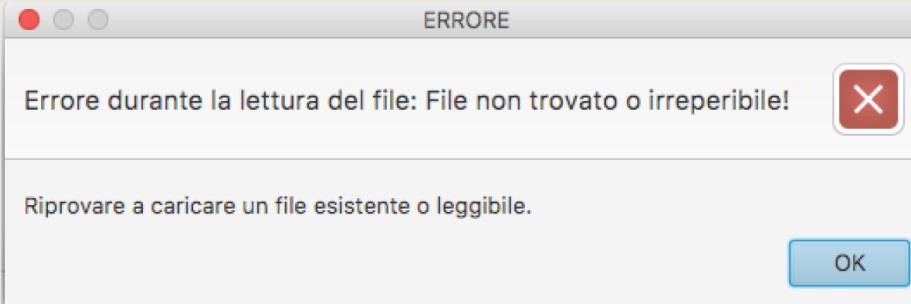
# Eccezioni: gestione anomalie

- Il numero delle features devono essere almeno due, altrimenti il datamining non avrebbe senso.
- L'eccezione lanciata è "**FeatureInsufficientException**"

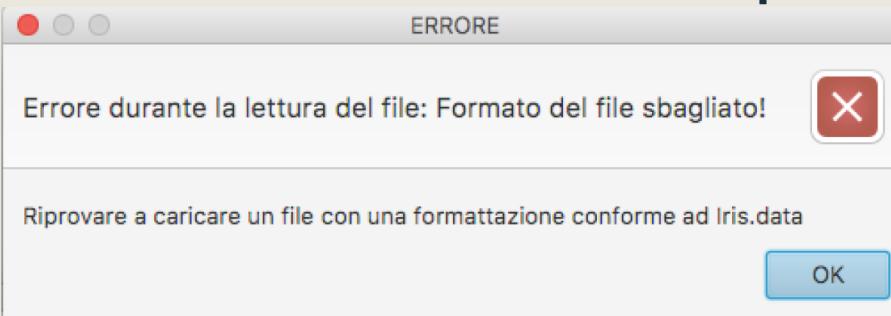


# Eccezioni: gestione anomalie

- Caso in cui si hanno errori sulla reperibilità del file.
- L'eccezione lanciata è "**FileNotFoundException**"



- Il file non è formato da valori adeguati (non numeri).
- L'eccezione lanciata è "**FileNumeriFormatoException**"



- Entrambe l'eccezioni estendono "**FileMioIOException**"

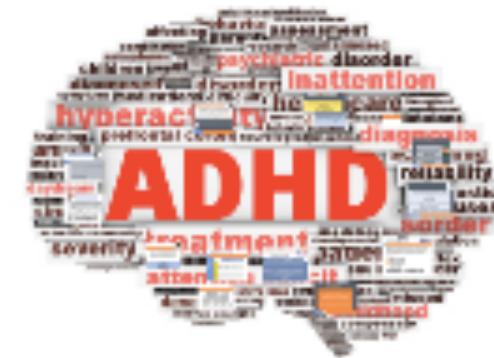
# Conclusione

La realizzazione di questo progetto è stata molto interessante perchè si sono assimilati e maneggiati molti argomenti.

Le maggior competenze acquisite sono:

- Java 8 SE
- JavaFX
- Principi di Ingegneria Software
- Design Pattern e SOLID
- Principi di Datamining e Machine Learning

Grazie per l'attenzione



Link YouTube dove osservare un test pratico:

- <https://www.youtube.com/watch?v=P7iS2-eLsm4>