

# Calcolatrice Anna

## Progetto per il corso di programmazione a oggetti

2017-2018

### 1.Introduzione

- Ambiente di sviluppo
- Comandi di compilazione
- Cosa viene consegnato

### 2.Scopo dell'applicazione

### 3.Descrizione parte logica

- Descrizione grafica della gerarchia
- C++
- Java
- Descrizione polimorfismo
- Descrizione incapsulamento
- Funzionalità offerte

### 4.Descrizione parte grafica

### 5.Divisione tra parte grafica e logica

### 6.Realizzazione del lavoro

- Ore impiegate
- Suddivisione lavoro

# 1)Introduzione

## **-Ambiente di sviluppo:**

Il progetto è stato sviluppato con Qt Creator e librerie Qt 5.5.1 in ambiente Windows 10, con compilatore MinGw 4.8.1.

## **-Comandi compilazione**

Per compilare il progetto è necessario aprire la shell nella cartella di consegna, generare il Makefile appropriato attraverso il comando qmake, e infine lanciare l'applicazione attraverso il comando make.

In seguito per eseguire il progetto invocare comando \.Anna.

Per quanto riguarda i file java invece vanno invocati da shell: javac \*.java per compilare ogni file .java esistente nella cartella, e infine java Use per eseguire.

## **-Cosa viene consegnato**

È stata consegnata una cartella contenente a suo interno le seguenti sottocartelle:

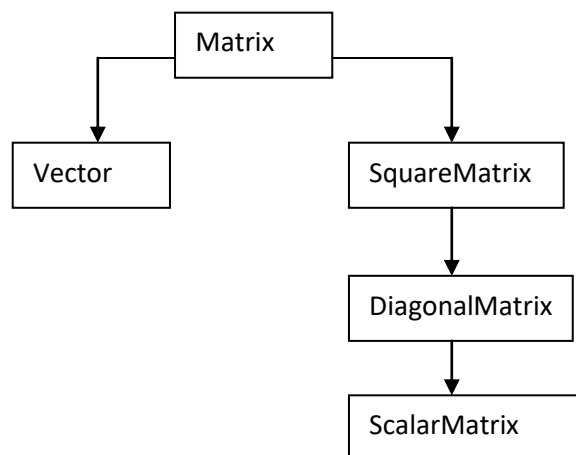
- “Parte Logica”, nella quale è implementata la gerarchia con classe base matrice in c++, all'interno della quale sono inclusi tutti i file .h e .cpp;
- “Java” nella quale è presente la corrispondente traduzione della gerarchia da c++ a Java, all'interno della quale sono inclusi i file .java ;
- “Parte grafica” nella quale è implementata la GUI per l'applicazione.

## **Scopo dell' applicazione**

Lo scopo del progetto è lo sviluppo di un'applicazione dotata di interfaccia grafica, che fornisca una calcolatrice estendibile in grado di gestire diverse tipologie di matrici e inerenti calcoli su di esse.

## **Descrizione parte logica**

### **-Descrizione grafica della gerarchia**



## **-C++**

### **-Matrix**

È la classe base della gerarchia, e ha lo scopo di rappresentare una matrice di tipo numerico generico. Pertanto è caratterizzata da 3 campi dati: due interi positivi "h" e "l" che rappresentano rispettivamente l'altezza e la lunghezza della matrice, e un array monodimensionale di tipo generico "rawMatrix" di dimensione h\*l utilizzato per tenere traccia degli elementi contenuta nella matrice.

### **-SquareMatrix**

Estende la classe base Matrix, e ha lo scopo di rappresentare una matrice quadrata di tipo numerico generico. Come campi dati eredita quelli della classe base, con l'unica fondamentale differenza che essendo una matrice quadrata l'altezza e la larghezza sono uguali: pertanto quando viene costruito un oggetto di questo tipo viene invocato il costruttore della classe base usando un solo parametro intero che sta a rappresentare entrambe le dimensioni.

### **-DiagonalMatrix**

Estende la classe SquareMatrix, e ha lo scopo di rappresentare una matrice diagonale di tipo numerico generico. Con matrice diagonale è definita una matrice quadrata nella quale gli unici elementi diversi da zero possono essere solo quelli appartenenti alla diagonale principale. Pertanto vengono forniti due costruttori: uno nel quale viene passato un parametro intero che sta a indicare le dimensioni della matrice e viene semplicemente richiamato il costruttore della classe base, e un altro nel quale sono passati un intero per le dimensioni e un parametro di tipo generico: all'interno di tale costruttore viene invocato il costruttore della classe base per costruire la matrice, in seguito essa viene riempita di zeri a meno degli elementi che appartengono alla diagonale principale che sono settati uguali al parametro generico passato.

### **-ScalarMatrix**

Estende la classe DiagonalMatrix, e ha lo scopo di rappresentare una matrice scalare di tipo generico. Con matrice scalare è definita una matrice diagonale nella quale gli elementi appartenenti alla diagonale principale sono tutti uguali. Pertanto viene fornito un unico costruttore, nel quale sono passati un intero per le dimensioni e un parametro di tipo generico: il comportamento è analogo al costruttore con uguale lista dei parametri della classe base diagonal matrix.

### **-Vector**

Estende la classe Matrix, e ha lo scopo di rappresentare un vettore(riga o colonna) di tipo generico. Pertanto si tratta di un particolare tipo di matrice nella quale o l'altezza o la larghezza sono settati a 1. Il costruttore è analogo a quello della classe base.

## **-Java**

La traduzione in java implementa le stesse funzionalità offerte dalla parte c++. Una fondamentale differenza, però, sta nel come viene rappresentato il tipo generico T dei dati contenuti all'interno delle matrici.

Mentre in c++ si è deciso di rappresentare attraverso un template di classe il tipo generico dei dati T contenuti nella matrice, nella parte Java si è deciso di utilizzare un approccio differente. Si è deciso di tradurre questa caratteristica dichiarando rawMatrix come un array di generici oggetti Object, sebbene la soluzione più vicina ai template di classe di c++ in java fossero i generics. Si è deciso di scartare questa opzione per diverse motivazioni:

-Non è consentita l'istanziamento di array di tipo generico T. Pertanto all'interno dei costruttori sarebbe stato necessario istanziare dapprima un array di Object[] e in seguito fare un cast da Object[]→T[], soluzione inadeguata perché potrebbe trattarsi potenzialmente di una conversione con perdita d'informazione;

-Non sono definiti operatori algebrici per i tipi di dato generici T: anche in questo caso per svolgere le operazioni all'interno delle funzioni per il calcolo sulle matrici si sarebbe dovuto fare un cast non sicuro Object→T per gli elementi in questione. Va precisato che in realtà neanche per gli oggetti di tipo Object non sono definiti operatori, e si è deciso di arginare l'inconveniente facendo una conversione Object→double dei dati per eseguire le operazioni (tramite l'apposita funzione ObjToDouble) e poi ritornare il risultato sottoforma di Object. Nello specifico si è deciso di lavorare con i double per svolgere le operazioni perché sono il tipo di dato che meglio generalizza un tipo numerico, e in caso di conversioni verso altri tipi numerici esse sono sicure.

### **-Descrizione polimorfismo**

Si è cercato di massimizzare, all'interno della gerarchia logica, il concetto di polimorfismo. Questo perché le matrici sono un tipo di calcolo che si presta molto alla ridefinizione di metodi a seconda del particolare tipo di matrice. Questo per due motivazioni:

1)Determinate operazioni sono molto semplificate a seconda del tipo di matrice con cui si tratta:

-Nelle matrici diagonali il calcolo del determinante e delle operazioni aritmetiche è semplificato: basta trattare solo gli elementi sulla diagonale principale (ancora più semplice per le matrici scalari, dove gli elementi sono tutti uguali);

-Nelle matrici diagonali e in particolare in quelle scalari operazioni inerenti all'algebra sulle matrici (come eliminazione di Gauss, GaussJordan) sono molto semplificate: basta operare sugli elementi della diagonale principale.

2)Determinate operazioni sono definite solo per specifici tipi di matrice :

-Il calcolo del determinante e dell'inversa riguarda solo le matrici quadrate;

-Il calcolo della norma e la normalizzazione riguarda solo i vettori.

### **-Descrizione incapsulamento**

Si è deciso di massimizzare questo aspetto rispettando la filosofia dell'information hiding, favorendo l'interazione utente-applicazione solo attraverso l'interfaccia pubblica. Per fare un esempio le funzioni getH() e getL(), all'interno della classe base Matrix permettono l'uso di elementi inerenti all'implementazione della classe (pertanto non di interesse per l'utente) attraverso metodi pubblici.

### **-Funzionalità offerte**

-Matrix

- Operazioni algebriche sui dati contenuti nelle matrici:+,-,\*,/ (in c++ usando usuali simboli, in java definendo nomi add, subtract..);
- Overload operatore uguaglianza(== in c++, equalsTo in java);
- Overload operatore di scoping([] in c++, getEl in java);
- Operazioni inerenti all'aritmetica delle matrici: Trasposta, eliminazione di Gauss, eliminazione di GaussJordan, prodotto tra matrici (per mezzo della definizione di prodotto vettoriale).

-SquareMatrix

- Metodi ereditati da base;

- Operazioni inerenti all'aritmetica tra matrici quadrate: Matrice inversa e calcolo determinante.

#### -DiagonalMatrix

- Operazioni algebriche ridefinite;
- Metodi ereditati da base;
- Operazioni inerenti all'aritmetica tra matrici ridefinite.

#### -ScalarMatrix

- Metodi ereditati da base;
- Operazioni inerenti all'aritmetica tra matrici ridefinite (nello specifico calcolo del determinante).

#### -Vector

- Metodi ereditati da base;
- Operazioni inerenti aritmetica tra vettori: calcolo norma e normalizzazione del vettore.

## Descrizione parte grafica

### 1)Area di settaggio delle matrici

Area nella quale vengono settate il numero di righe e di colonne tramite un QPushButton che permette di incrementare o decrementare le dimensioni.

A seconda delle dimensioni settate viene indicato il tipo di matrice con cui si sta lavorando.

Si noti che inizialmente la matrice contiene come elementi tutti zero, pertanto il tipo di matrice indicato inizialmente, quando l'utente non ha ancora impostato i valori a suo interno potrà essere del tipo: matrice ordinaria, matrice scalare o vettore. Solo in seguito all'aggiunta di valori adeguati la matrice verrà segnalata come quadrata (se esistono elementi all'infuori della diagonale diversi da zero) o diagonale (se gli elementi sulla matrice principale non sono tra loro tutti uguali).

### 2)Spazio di input

	1	2	3	4
1	2	0	0	0
2	0	2	0	0
3	0	0	2	0
4	0	0	0	2

Spazio nel quale l'utente setta i valori all'interno della matrice. Vengono impediti inserimenti di dati non numerici.

Nella sezione in basso se si tratta di una matrice quadrata viene calcolato automaticamente il determinante, in caso contrario non viene calcolato e viene dichiarato come non disponibile.

<- Copia <-

-> Copia ->

Somma

Moltiplicazione

Differenza

### 3)Operatori tra matrici

Spazio nel quale sono indicati gli operatori tra matrici: somma, sottrazione e moltiplicazione riga per colonna e infine funzioni di copia (il cui verso è indicato da una freccia) che permettono di copiare i valori di una matrice nell'altra.

Si notino due dettagli:

-In caso di dimensioni non adatte alle operazioni richieste (nello specifico somma e sottrazione devono avvenire tra due matrici di uguale dimensione e la moltiplicazione tra due matrici tali che la lunghezza della prima sia uguale all'altezza della seconda) viene segnalato tramite un apposito segnale di errore;

-Quando viene fatta la copia tra due matrici di dimensione diversa la matrici di destinazione assume le dimensioni (e ovviamente i valori) della matrice dalla quale si copia.

#### 4) Operazioni sui singoli elementi della matrice



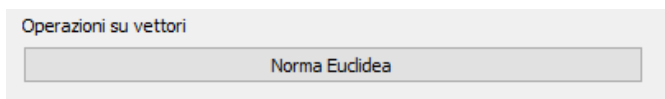
Qui vengono dichiarate le operazioni da svolgere singolarmente su ogni elemento della matrice: radice e potenza di x. Per settare il valore di x vi è una apposita sezione sottostante alle operazioni.

#### 5) Operazioni definite sulla matrice



In quest'area sono definite le operazioni da eseguire sull'intera matrice: trasposta, che setta l'area 2 come trasposta della matrice in input, eliminazione di Gauss e GaussJordan anch'esse eseguite sull'area 2 e infine l'inversa. Si noti che quest'ultima è definita solo su matrici quadrate, pertanto il tentativo di applicarla su altre tipologie di matrice fallirà, segnalando errore.

#### 6) Operazioni sui vettori



Infine viene dedicato un apposito spazio per l'operazione di normalizzazione definita per i vettori.

### Divisione tra parte grafica e parte logica

Si è cercato di massimizzare questo aspetto: infatti la parte logica consiste nell'insieme di classi che implementano la gerarchia Matrix, e non ha alcun legame con la parte inerente alla GUI. Infatti il modello logico offerto può essere utilizzato anche da una calcolatrice a riga di comando non supportata da una GUI.

### Realizzazione

#### **-Ore di lavoro**

Le ore complessive di lavoro impiegate dalla sottoscritta per adempiere ai propri compiti sono state all'incirca in linea con le 50 ore indicate dal docente.

-Modellamento e progettazione: circa 13 ore;

-Svolgimento parte logica in c++ con relativi test e debugging: circa 12 ore;

-Apprendimento e svolgimento parte logica in java: 20 ore.

### ***-Suddivisione lavoro progettuale***

Le fasi di progettazione e modellamento sono state fatte congiuntamente al compagno di progetto Caccaro Sebastiano.

La sottoscritta ha svolto congiuntamente al compagno di progetto Caccaro Sebastiano la parte logica in c++ e ha svolto individualmente l'intera parte logica di java.