Università degli Studi di Padova
Dipartimento di Matematica
Corso di Laurea in Informatica
A.A. 2017-2018

MatrixKalQ

Relazione progetto di Programmazione ad Oggetti

Studente:
Antonio Moz (#1097503)

Compagno:
Leo Moz (#1029352)

Docente:
Francesco Ranzato

0 Compilazione, Suddivisione del Lavoro, e Tempo

Sistemi operativi di sviluppo: *"Ubuntu 16.04 LTS"* e *"Windows 10"*

Versione di *Qt* utilizzata: *"Qt 5.5.1"*

Versione del compilatore: *"gcc 5.4.3"* e "*mingw 4.9.2*"

Versione di *Java*: "1.8.0"

0.1 Istruzioni per la compilazione

0.1.1 C++

A causa dei cambiamenti introdotti da *Qt5* "*qmake -project"* non è più sufficiente se si fa uso di *Qwidget*, pertanto per una corretta compilazione si usi il file "*progetto.pro"* consegnato o si diano i seguenti comandi dalla cartella *"cplusplus"*: qmake -project "QT +=widgets" => qmake => make

0.1.2 Java

Per *"compilare"* i file *.java*, dalla cartella *"java"*, dare (notare il punto alla fine!):
javac *java -d .
Per eseguire il *"main()"* dimostrativo, sempre dalla cartella *"java"*, dare:
java Use

0.2 Suddivisione del Lavoro Progettuale

Nel complesso il lavoro è stato svolto da entrambi gli studenti, senza alcuna netta suddivisione del lavoro.
Scelte progettuali, sviluppo dell'intera gerarchia, e scelte per l'interfaccia grafica sono state svolta in condivisione continua fra i due studenti. Per quanto concerne la *GUI* lo scrivente questa relazione ha curato maggiormente l'interfaccia grafica (più precisamente la classe *"LayoutManager"*), mentre l'altro studente si è occupato della classe *"LogicManager"* e *"Cute-MatrixKalk"*.
Inoltre, lo scrivente, ha curato la stesura della classe *"Use"* del codice *Java*, mentre i dettagli della compilazione e del packaging di *Java* sono stati curati maggiormente dall'altro studente.

0.3 Tempo impiegato

Si può stimare un tempo di circa *60-70 ore* per ciascuno dei componenti del gruppo, di cui circa un *50%* per la fase iniziale di progettazione e sviluppo della gerarchia in *C++*, *20%* per la *GUI*, *15%* per *Java*, ed il restante per la stesura della *relazione indivuduale* e *testing* conclusivo.

Il totale delle ore previsto per lo sviluppo del progetto è stato superato a causa delle seguenti motivazioni:

- Scelta iniziale dell'*argomento* da trattare e successiva progettazione della *gerarchia di tipi*: questa fase ci ha impiegato più tempo di quanto avessimo stimato a causa della decisioni e il recupero delle informazioni sul operazioni, algoritmi, e funzionalità da poter esporre all'utente finale;
- Utilizzo del *framework Qt* con relativa *documentazione*: estrememante vasto anche se chiaro ed esplicativo, ma cercare una funzionalità specifica di volta in volta ha impiegato inevitabilmente tempo;
- *Java*: riportare la gerarchia di tipi anche in *Java* ha impiegato più tempo di quello stimato, a causa del fatto che il linguaggio è vasto ed espone moltissime funzionalità e classi di libreria che potevano essere utili, inoltre è inevitabile porre attenzione a sintassi e differenze con *C++*, e per poterlo fare con attenzione impiega un maggior quantitativo di tempo.

Indice

1 Abstract

MatrixKalQ è una calcolatrice per svolgere operazioni sulle matrici con elementi appartenenti all'insieme *R* dei numeri reali.

Sviluppato come progetto per il corso di Programmazione ad Oggetti cercando il più possibile di scrivere un programma che rispetti le regole della *OOP* apprese durante il corso.

Il progetto è stato sviluppato principalmente in *C++* con un'interfaccia grafica resa grazie all'utilizzo del *framework Qt*. Inoltre la parte *core* del progetto è stata sviluppata anche il linguaggio *Java*, e reso disponibile un esempio delle funzionalità esposte grazie ad una classe *Use* che mostra degli esempi delle relative operazioni.

Il progetto mantiene una netta separazione fra la parte riguardante la gerarchia di tipi delle matrici dalla parte concernente la *GUI* sviluppata grazie al meccanismo della *programmazione ad eventi* di *Qt*.

2 Gerachia di tipi (C++ e Java)

2.1 Classe base astratta: MatrixBase

All'apice della gerarchia si trova una classe astratta che dichiara solamente che ogni matrice è composta da due campi interi positivi che indicano il numero di righe e di colonne di cui ogni matrice dev'essere dotata.

I metodi di tale classe astratta sono tutti metodi virtuali puri perché in tale classe non è possibile la loro corretta implementazione, in quanto questa potrebbe variare in base alla struttura dati scelta per la rappresentazione del concetto stesso di matrice.

Oltre ai due metodi banali per ottenere il numero di righe e colonne della matrice, e il costruttore per la costruzione del sottooggetto, la classe espone solamente gli operatori di *uguaglianza*, *disuguaglianza*, *input*, e *output*, i quali contengono chiamate a metodi virtuali puri della classe, tali metodi in modo polimorfo scglieranno il giusto *overriding* dei metodi in base al tipo dinamico degli oggetti di invocazione.

2.2 Classe astratta: DenseMatrix

La classe riguardante le matrici dense è l'unica classe derivata direttamente dalla classe base astratta, la scelta di aggiungere un'ulteriore classe astratta è stata presa per permettere, dando uno sguardo all'evolvibilità futura della gerarchia, l'implementazione di classi riguardanti le cosiddette matrici sparse, che spesso vengono implementate con struttutre dati particolari e algoritmi diversi per migliorare l'efficienza computazionale e spaziale.

La classe *DenseMatrix* dona una struttura alle matrici da noi prese in esame intrducendo un campo dati composto da *un vector di vector di double*, ed inizia la concretizzazione dei metodi astratti della classe base, lasciando astratti quelli che come tipo di ritorno prevedono un tipo non primitivo, e che effettuano solamente con cambio di tipo per *covarianza* rispetto a quelli della classe base astratta.

Aggiunge anche dei metodi propri per esporre il concetto di generazione di una matrice con elementi casuali.

2.3 Classi per le Matrici Quadrate

Il ramo sinistro della gerarchia di tipi riguarda le matrici quadrate che sono caratterizzate dall'avere il numero di righe uguale al numero di colonne.

2.3.1 Classe: SquareMatrix

La classe *SquareMatrix* è concreta, e perciò implementa tutti i restanti metodi virtuali puri della sua superclasse diretta.

Sulle matrici quadrate, rispetto alle matrici rettangolari, è possibile effettuare diverse operazioni aggiuntive come ad esmpio il calcolo del determinante o il calcolo dell'inversa di una matrice. Tali metodi sono resi disponibili in questa classe e, ovviamente, nelle sue sottoclassi.

Essendo una classe concreta, si è deciso di implementare anche alcuni operatori matematici che svolgono il medesimo lavoro di alcuni metodi concretizzati già in precedenza dalla superclasse diretta. La scelta di implementare anche gli operatori è stata realizzata al fine di rendere disponibili alcune funzionalità e operazioni che un utente si aspetta di poter fare quando si tratta di matrici come concetto matematico.

In *Java* non è permesso *l'overloading* degli operatori, e per questo motivo vi sono solamente i metodi che in *C++* operano grazie al tipo di ritorno covariante, e gli operatori non sono disponibili.

2.3.2 Classe: Matrix2

Questa classe deriva direttamente dalla classe *SquareMatrix* e identifica specificatamente le matrici *2x2* ovvero con una numero di righe ed un numero di colonne uguali a 2.

In tale classe non viene aggiunta alcuna funzionalità ulteriore, ma viene effettuato *l'overriding* dei metodi allo scopo di migliorne l'efficienza rispetto a quelli ereditati dalla sua superclasse.

2.3.3 Classe: Matrix3

Anche la classe *Matrix3* deriva direttamente dalla classe *SquareMatrix* e identifica le matrici con un numero di righe ed un numero di colonne uguali a 3.

Anche in questo caso, *l'overriding* di alcuni metodi è effettuato per un discorso di efficienza, e non per una diversità nel comportamento di essi.

2.4 Classi per le Matrici Rettangolari

Il ramo destro della gerarchia di tipi riguarda le *matrici rettangolari* ed i *vettori*.

2.4.1 Classe: RectMatrix

La classe *RectMatrix* è concreta, e perciò implementa tutti i restanti metodi virtuali puri della sua superclasse diretta *DenseMatrix*.

Tale classe identifica le matrici rettangolari, ovvero quelle matrici con un numero di righe diverso dal numero di colonne. Le operazioni effettuabili su una matrice rettangolare sono molto inferiori a quelle esposte invece da una quadrata, e per questo motivo la classe delle matrici rettangolari non espone metodi aggiuntivi.

Nel codice *C++* della gerarchia, come per la classe concreta *SquareMatrix*, vi è un'implementazione dei metodi sia come operatori, sia come funzioni che possono essere chiamate in modo *polimorfo*. In *Java* per le stesse motivazioni precedentemente riportate vi è solamente la possibilità di utilizzare i metodi che possono essere subire *late binding*, e gli operatori non sono disponibili.

2.5 Classi per i Vettori

Il concetto di vettore può essere visto come una specializzazione delle matrici rettangolari, ovvero una matrice formata da una sola riga o colonna di elementi.

2.5.1 Classe astratta: Vect

Vect è una classe astratta che deriva dalla classe *RectMatrix* è generalizza il concetto di vettore, non indicando in alcun modo la sua "*direzione*". Questa classe rende nuovamente puri alcuni metodi, e ne aggiunge altri, e si pone come classe base per le due sottocolassi *VectHor* e *VectVert*.

2.5.1.1 VectHor

La classe concreta *VectHor* indica un vettore orizzontale (o vettore riga).

Questa classe si occupa di implementare i metodi astratti della classe *Vect*, ed espone un nuovo metodo *"rev()"* disponibile solamente per i vettori e non per gli altri tipi di matrice, tale metodo inverte l'ordine degli elementi del vettore d'invocazione. Inoltre riespone i consueti operatori per la versione in *C++*, cosa che non accade per la versione scritta in *Java*.

Una particolarità dei vettori è che l'operzione di trasposizione su un vettore riga ritorna un vettore colonna.

2.5.1.2 VectVert

La classe concreta *VectVert* indica un vettore verticale (o vettore colonna).

Dualmente alla classe *VectHor* concretizza i metodi della superclasse astratta *Vect*, espone il metodo *"rev()"* e gli operatori come la somma e la differenza, e implementa un metodo di trasposizione duale alla classe dei vettori orizzonatali, e quindi la trasposta di un vettore colonna ritorna un vettore riga.

3 Interfaccia grafica (C++ e Qt)

La *GUI* (*Graphical User Interface*) è stata sviluppata grazie al supporto del f*ramwork Qt* e si base pesantemente sul sistema di *signal* e *slot* che permette una programmazione ad eventi.

La classe che gestisce l'inera interfaccia grafica è la classe “*CuteMatrixKalk*” che genera due oggetti “*LayoutManager*” e “*LogicManeger*” e li mette in comunicazione tra loro.

3.1 Classe: LayoutManager

Questa classe deriva direttamente dalla classe *Qwidget* ed è stata pensata per la realizzazione dell'interfaccia grafica effettivamente esposta all'utente. Si basa su una shermata destinata all'*input* delle matrici divisa a sua volta in una parte sinistra composta dalla scelta della matrice da poter inserire ed una parte destra che espone un'area di testo editabile per l'inserimento dell'input.

Un'altra schermata è destinata per la scelta dell'*operazione* sulla parte sinistra e sulla parte destra si trova un'area di testo non editabile che riporta il risultato dell'operazione richiesta dall'utente in forma matriciale o numerica.

Tale classe si basa sul concetto *OOP* dell'”*has-a”* per la costruzione corretto di un oggetto di tipo *LayoutManager*.

LayoutManager, inoltre, si occupa della comunicazione tramite signal e slot con la classe *“LogicManager”* per comunicare l'input utente, e per riportare l'output delle operazioni richieste.

3.2 Classe: LogicManager

Questa classe deriva direttamente dalla classe *QObject* ed è stata pensata per l'effettiva interazione con la gerarchia di tipi in base alle richieste dell'utente.

Riceve i segnali dalla classe *“LayoutManager”* ed interroga in modo appropriato la gerarchia di tipi per la creazione degli oggetti matrice richiesti e le operazioni su di esse.

4 Funzionalità e Codice Polimorfo

4.1 Operazioni sulle Matrici

Le funzionalità esposte da *MatrixKalQ* sono operazioni attinenti al concetto matematico di matrice. Alcune di queste sono applicabili solamente a determinati tipi di matrice, e si possono suddividere in:

- Operazioni *unarie*: rango, matrice trasposta, determinante, traccia, matrice aggiunta, matrice inversa, inverso di un vettore;
- Operazioni *unarie con parametro*: prodotto per uno scalare, esponenziale;
- Operazioni *binarie*: somma, differenza, e prodotto.

Le suddette funzionalità sono quelle esposte all'utente nella *GUI* nella seconda schermata, dove è presente a sinistra una pulsantiera di scelta dell'operazione e a destra un'area di testo per il risultato. Inoltre è possibile utilizzare la funzione di inserimento di una matrice di numeri interi casuali compresi nell'intervallo *[-100, 100]*, oppure il caricamento di una matrice precedentemente salvata per poter eseguire nuove operazioni.

Si noti che con matrici di dimensioni ridotte (ad esempio fino ad una *5x5* o una *7x7*) il tempo di computazione di qualsiasi operazione su un computer nella norma è irrilevante. Invece crescendo di dimensioni, per il calcolo di operazioni complesse come il determinate, la matrice aggiunta, o la matrice inversa il tempo richiesto potrebbe essere significativo. Il progetto è stato svolto solamente a scopo didattico, e quindi non è previsto di operazioni concorrenti o basato su cache dei risultati per velocizzare il tempo richiesto.

4.2 Codice Polimorfo

In generale l'approccio impiegato nello sviluppo del progetto ha tenuto conto il più possibile dell'utilizzo del codice tramite chiamate polimorfe (ove possibile).

Utilizzi di chiamate polimorfe si possono vedere in chiamate come di metodi come *“load”* e *“print”* direttamente negli operatori di *input* e *output* della classe base astratta *MatrixBase*, la quale marca questi due metodi come astratti, e quindi solamente grazie al late binding è possibile conoscere il tipo dinamico dell'oggetto d'invocazione effettivo.

Inoltre nella classe *“LogicManager”* della *GUI* le chiamate di alcuni metodi come *“trans()”*, *“rank()”* o *“det()”* sono chiamate polimorfe a metodi esposti dalla gerarchia di tipi.

Dualmente per quanto riguarda Java, nella classe *“Use”* che espone il metodo *“main()”*, vengono dichiarati solamente due riferimenti ad oggetti di *“MatrixBase”* e vengono di volta in volta inizializzati con oggetti di qualche sottoclasse concreta. Le chiamate ad i metodi sono tutte chiamate polimorfe, precedute da un *cast* solamente nei casi in cui il metodo non è direttamente disponibile nella classe base astratta.

5 Utilizzo e Manuale Utente

L'applicazione una volta eseguita mostrarà una schermata nella quale si potrà scegliere fra quale tipo di matrice si desidera inserire. Nel caso si scelga una matrice quadrata generica, una matrice rettangolare o un vettore è necessario inserire la dimensione desiderata prima di premere sul bottone relativo alla matrice di interesse, in caso contrario un messaggio avviserà l'errato input inserito.

Una volta scelta la matrice sarà possibile inserirla manualmente, oppure generarne una casuale, o caricarne una da un file precedentemente salvato, se non verrà scelto l'inserimento casuale, al termine dell'inserimento è necessario premere il pulsante “*Prosegui*” per avanzare.

Ora si ha la propria matrice inseita sul lato destro dello schermo, e si potrà decidere che operazione si desidera effettuare. In caso di scelta di un operatore che prevede l'inserimento di un ulteriore valore, si è obbligati ad inseirlo prima di premere sull'operazione richiesta, se non venisse fatto comparirà il consueto pop-up di avviso.

È possibile anche sciegliere un'operazione che prevede l'input di una seconda matrice, in tal caso si verrà riportati alla schermata di input per l'inserimento della seconda matrice, ma questa volta la parte sinistra sarà bloccata con le dimensioni e il tipo della matrice già fissati, questo è dovuto dal fatto che le operazioni binarie sono possibili esclusivamente fra tipi di matrici uguali e delle stesse dimensioni.

Si potrà eseguire un qualsiasi numero di operazioni sulla matrice che viene aggiornata operazione dopo operazione, se si è interessati si può anche salvarla su file per poterla ricaricare in un'altra sessione dell'applicazione.

Inoltre il pulsante “*Reset*”, come indica il nome, riporta la calcolatrice allo stato iniziale dell'esecuzione, pronta ad un nuovo utilizzo.