1506
UNIVERSITÀ
DEGLI STUDI
DI URBINO
CARLO BO

Corsi di Laurea in Informatica Applicata a.a 2023/2024

Relazione per il Progetto del Corso di Programmazione e Modellazione a Oggetti

Gianmarco Beligni
Mattia Gjyzeli
Paolo Dettori

Calcolatrice grafica

September 13, 2025

1 Analisi

Il Team si pone come obiettivo lo sviluppo di un programma in grado di tracciare il grafico di più funzioni $f : D \to R$ con $D \in R$ o funzioni $f : R^2 \to R$ con $D \in R^2$ calcolata nella curva di livello z=0.

Grafico a una variabile

$$Gf(f) = \{(x, y) \in D \times R \, tale \, che \, y = f(x)\}$$

Grafico a due variabile

$$Gf(f) = \{(x, y, z) \in D \times R \, tale \, che \, z = f(x, y) \, e \, z = 0\}$$

Le funzioni sono formate da

- Variabili: x,y
- Costanti: 0, 4, 5.35 $\{numero \mid numero \in A \subset Q\}$
- Funzioni unarie: sin, cos, log, tan
- Funzioni binarie(o anche operazioni): $+, -, *, /, \wedge$
- Parentesi: (,)

Il grafico si ottiene calcolando la funzione in ogni punto di un intervallo intersecato con il proprio dominio.

1.1 Requisiti

- Il programma dovrà controllare errori semantici e sintattici nelle funzioni, evidenziarli e comunicarli all'utente
- il programma dato un certo intervallo e funzione sarà in grado di calcolare tutte le immagini di quei punti(dove è possibile)
- il programma dovrà avere una visualizzazione grafica interattiva dei risultati
- il programma dovrà valutare delle funzioni matematiche con il corretto ordine di precedenza delle operazioni
- il programma dovrà essere in grado di traslare, ingrandire e rimpicciolire il piano cartesiano in base all'input dell'utente.
- interfaccia utente intuitiva con controlli per zoom e pan

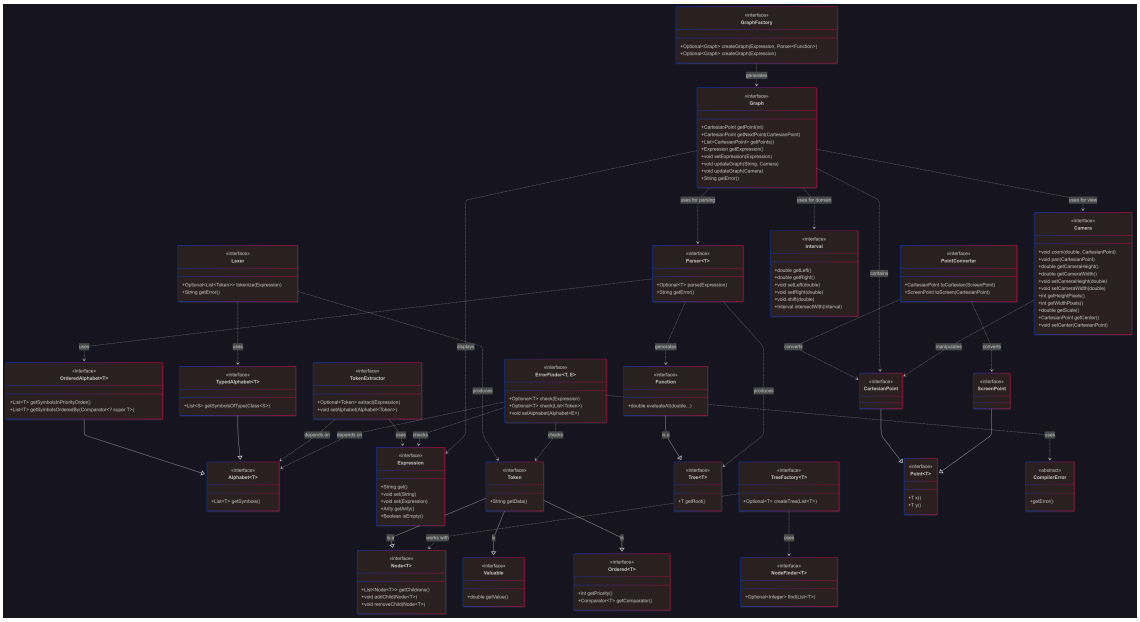


Figure 1: Flow semplificato del programma

1.2 Modello del dominio

Il programma avrà un solo piano cartesiano che può avere al suo interno zero o più grafici in base a quante espressioni ben formate l'utente fornisce, i grafici calcoleranno i propri punti di ascissa dentro all'intervallo mostrandoli su schermo.

- il grafico sarà rappresentato dalla entità Graph
- Interval da un limite superiore e inferiore per il calcolo dei punti
- Camera contiene i dati dello spazio che l'utente potrà vedere.
- Token è l'unita fondamentale del linguaggio matematico
- i punti sono modellati da CartesianPoint quelli che sono calcolati da Function e CameraPoint-Converter li può trasformare in ScreenPoint che potranno essere mostrati su un generale canvas nello schermo
- Expression rappresentà l'input dell'utente
- Function rappresenta la funzione con la capacità di valutare quella espressione per qualsiasi valore di x o y se presente.
- Alphabet rappresenta l'alfabeto su cui si lavora nelle espressioni

Il problema più grande sarà la creazione dell'oggetto Graph con tutti i punti al suo interno.

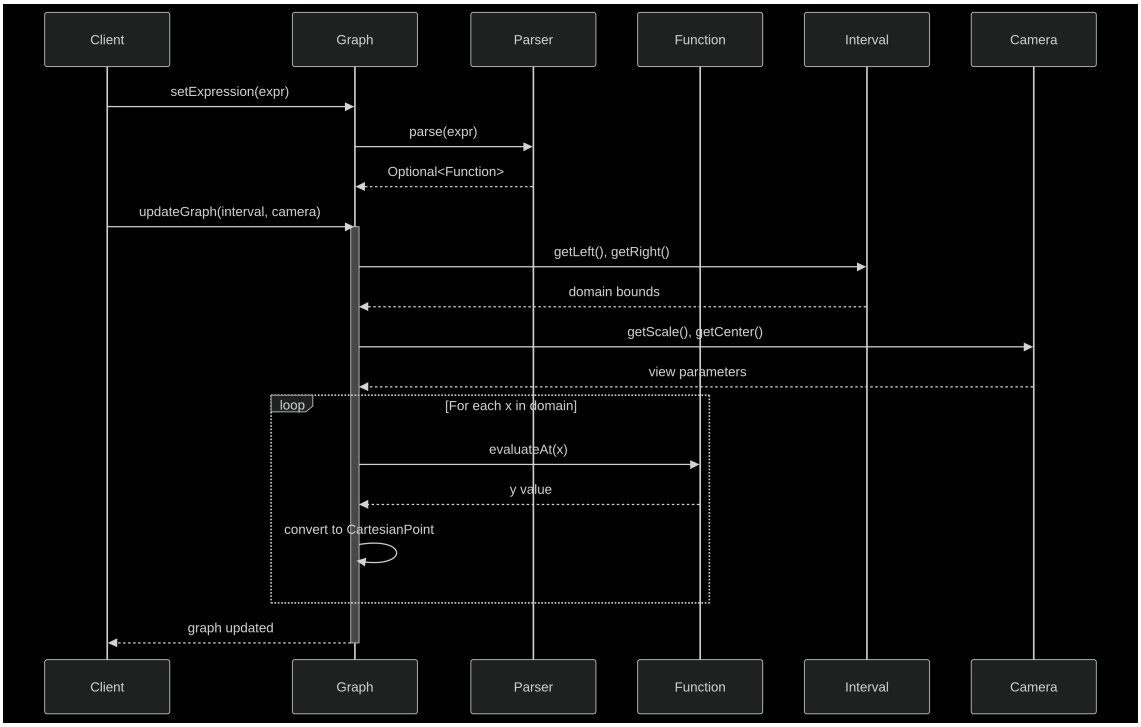


Figure 2: Flow semplificato del programma

2 Design

2.1 Architettura

Per lo sviluppo dell'architettura di questo programma si è deciso di seguire il pattern architetturale MVC (Model - View - Controller).

2.1.1 Model

Il model gestiste l'accesso e la modifica dei dati dell'applicazione, determinando come i dati input dell'utente influiscano sull'applicazione. Il model incapsula al suo interno la logica del programma e il suo obbiettivo è di notificare al Controller l'istanza attuale dell'applicazione. Il model non dovrà dipendere da View e Controller.

2.1.2 View

La View si occupa della parte grafica dell'applicazione che viene presentata all'utente. La View è responsabile della visualizzazione dei dati in modo che sia leggibile per l'utente, i dati sono forniti dal Model passando per il Controller. La View è priva di logica ed ha solo fine visivo. La View non dovrà dipendere dall'implementazione del linguaggio. Come team abbiamo deciso di utilizzare Java swing e abbiamo fatto del nostro meglio per non far dipendere la view dall'implementazione.

2.1.3 Controller

Il controller è il cosiddetto "middle man" dell'architettura MVC, osserva cambiamenti della View causati dall'utente per cambiare l'istanza dell'applicazione che a sua volta provocano cambiamente nella View. Abbiamo deciso di soddividere il lavoro del Controller creando più listener uno per ogni elemento della view che mandano dei paccheti di dati chiamati event al controller che verranno usati per riaggornare il model.

2.2 Design Dettagliato

2.2.1 Gianmarco Beligni

2.2.2 Graph

Mi sono posto come obiettivo la creazione dell'oggetto Graph, che contiene all'interno tutti i punti del grafico. Graph al proprio interno contiene una funzione e un intervallo.

Ho avuto la necessità di utilizzare il tipo Graph per creare due classi che sono GraphSingleVar e GraphTwoVar poiché il metodo per trovare i punti della funzione e quello di trovare l'intervallo su cui lavora l'oggetto Graph sono differenti. Per questo ho utilizzato il **Pattern Template Method** con i metodi astratti `createInterval` e `iterateAndCheckIfPointInGraph`.

Per la creazione del Grafico mi sono affidato al **Pattern Factory Method**.

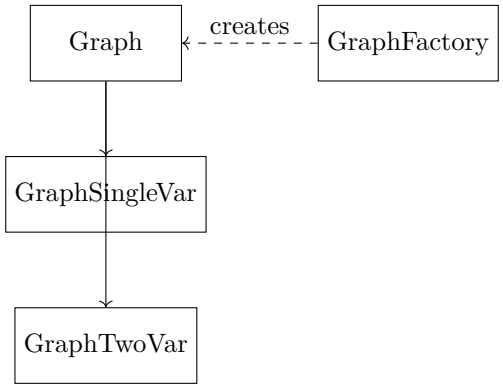


Figure 3: UML Graph - Pattern Template e Factory

2.2.3 Function

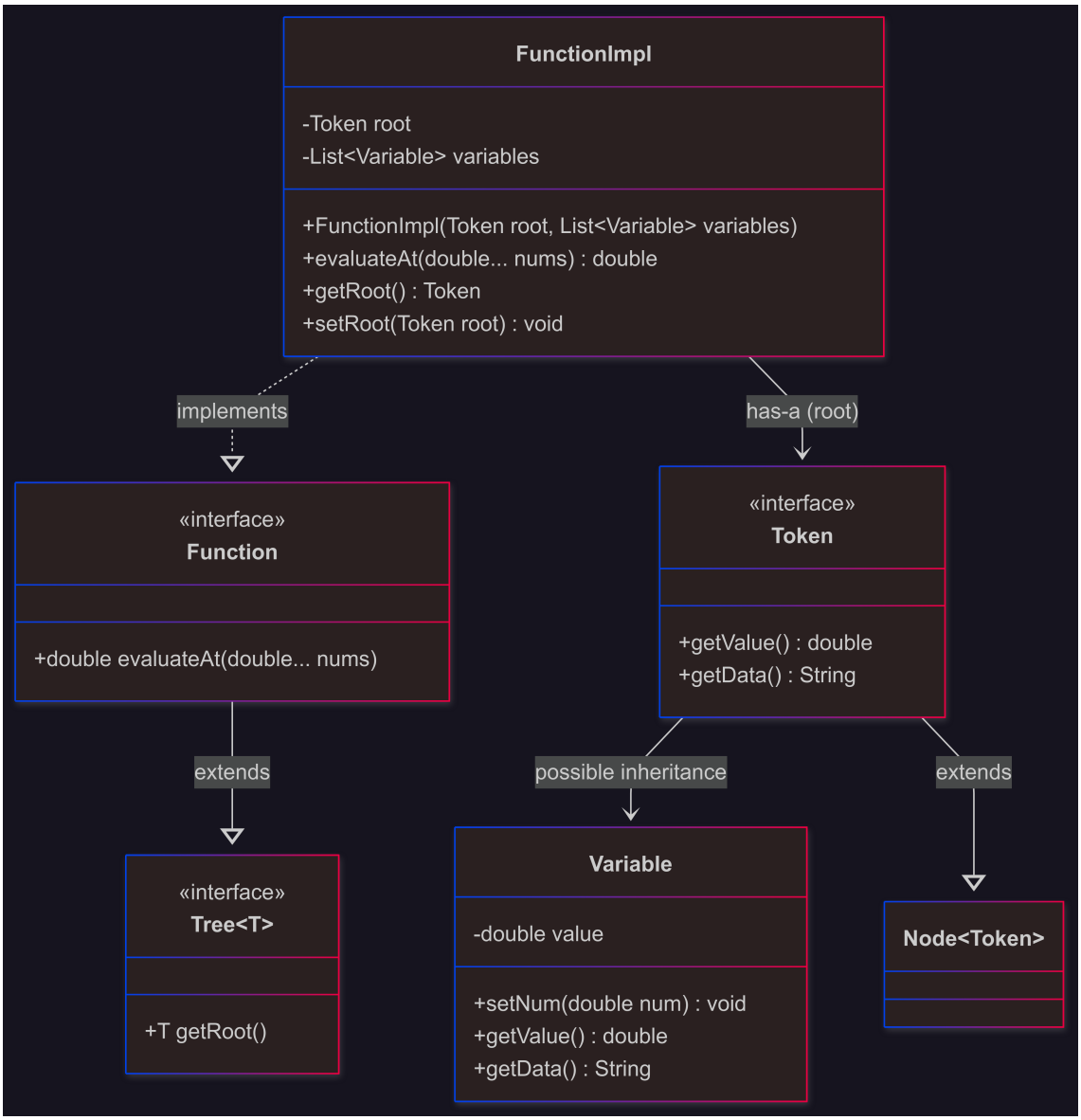
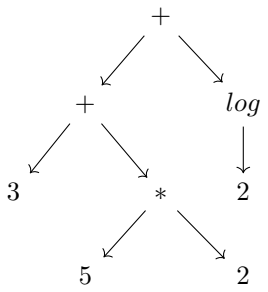


Figure 4: UML function

La classe Function racchiude l'essenza della funzione. Il metodo più elegante per rispettare l'ordine di operazione è con l'utilizzo di un albero attraverso **il Pattern Composite**

Esempio: espressione $= 3 + 5 * 2 + log(2)$



La funzione sarà rappresentata da un albero di Token. Si risale al valore dell'espressione con il

Pattern Chain Of Responsibility chiamando ricorsivamente il metodo `getValue` sulla radice che invocherà ricorsivamente `getValue` sui figli. Ogni tipo di Token dovrà implementare questo metodo.

2.2.4 Parser

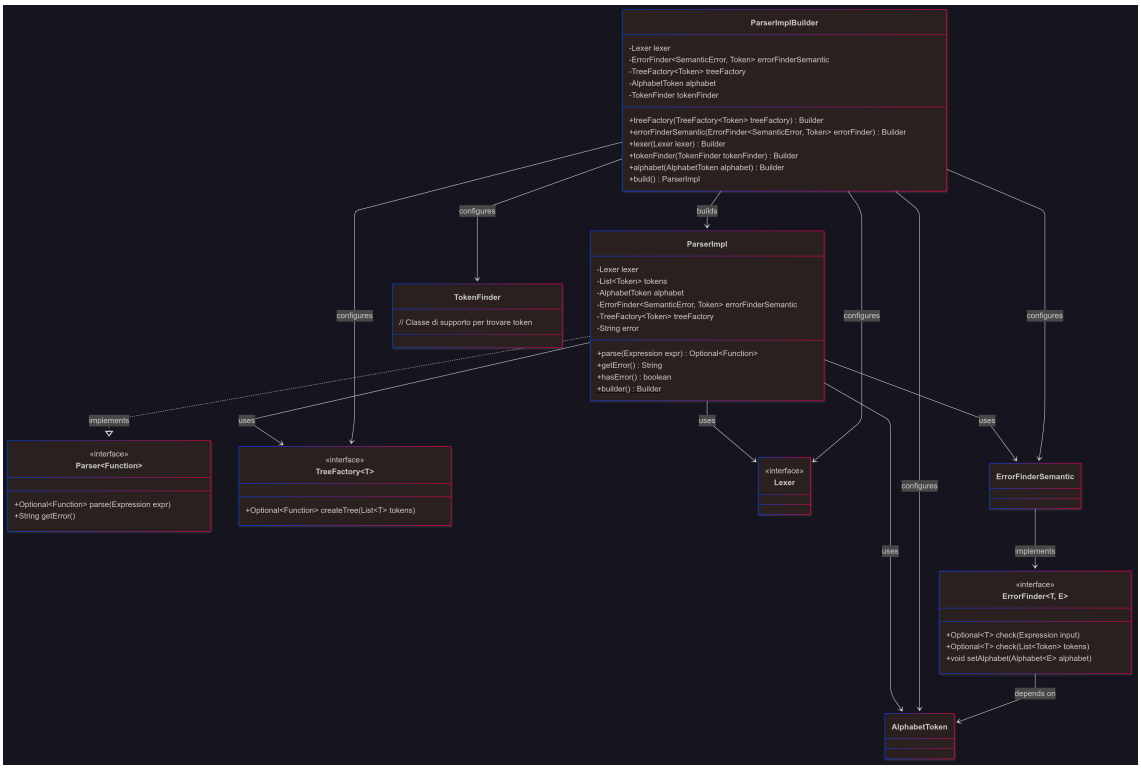


Figure 5: UML Parser

L'albero viene creato da un Top Down Recursive Parser con l'aiuto di altri oggetti per far sì che ogni oggetto abbia un solo compito. Per aiutare il parser ho creato:

- **TreeFactory**: factory per creare l'albero
- **TreeNodeFactory**: creazione ricorsiva dei nodi
- **Lexer**: per avere la lista di token
- **ErrorFinderSemantic** per controllare errori di semantica

Questi oggeti verranno assemblati grazie al Pattern **Builder**

2.2.5 Lexer

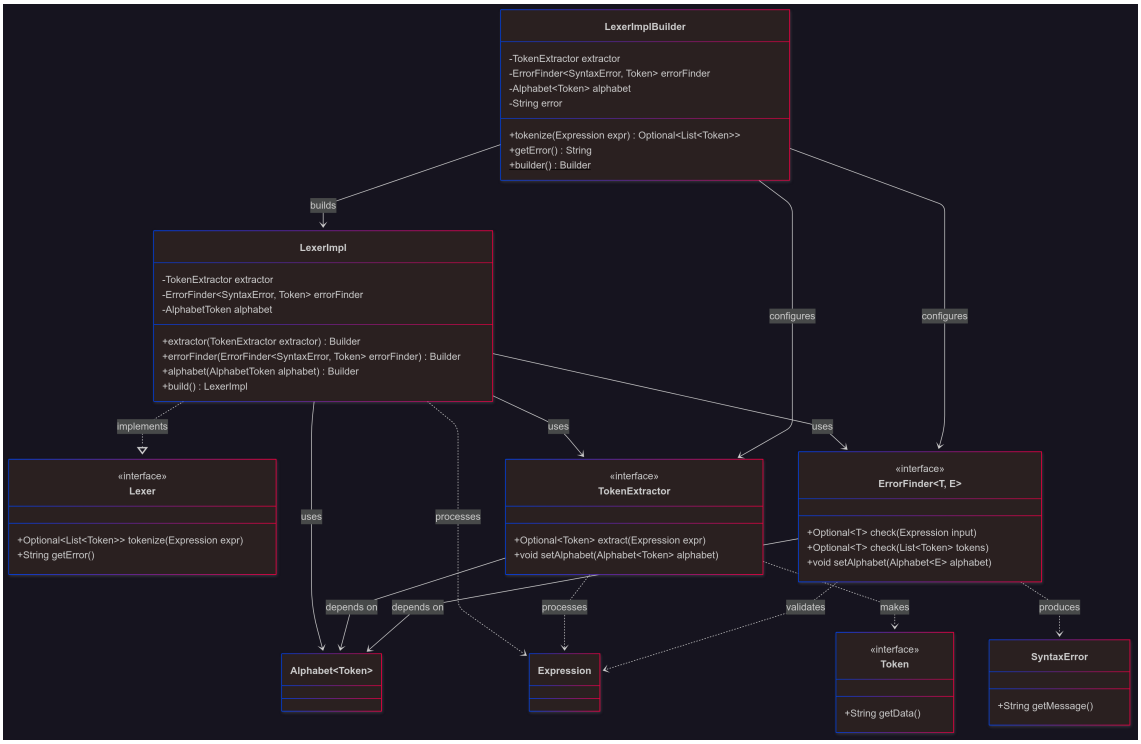


Figure 6: UML lexer

Il compito del Lexer è di trasformare l'espressione in una lista di Token da dare in pasto al parser. Per la creazione dei Token si usa il **Pattern Factory** e il **Builder Pattern** per una configurazione flessibile.

2.2.6 Alphabet

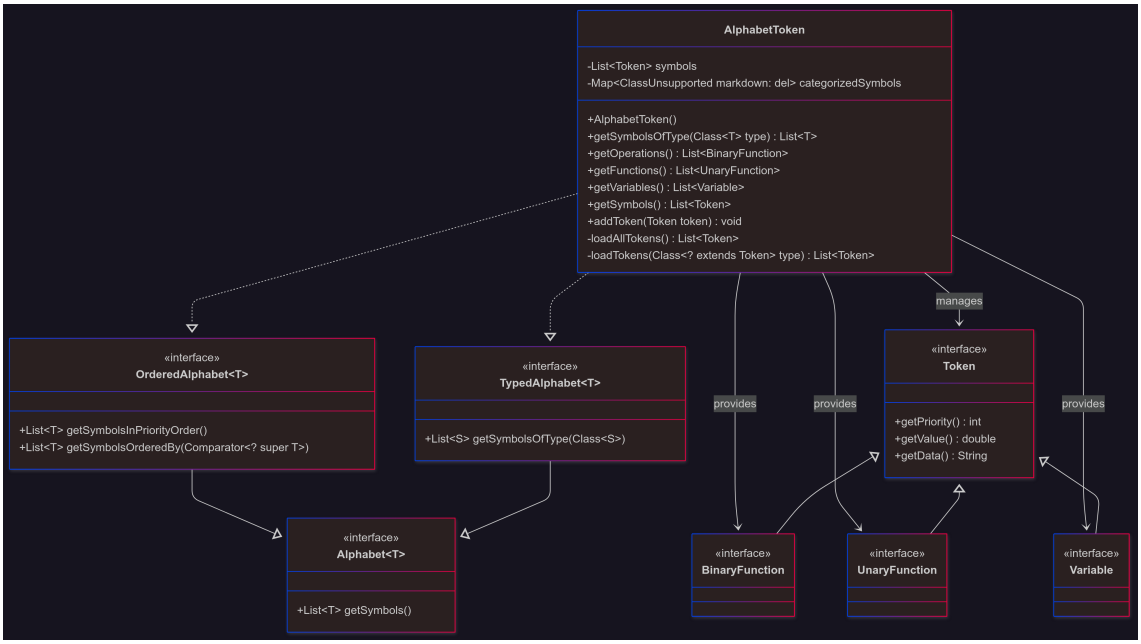


Figure 7: UML alphabet

L'alfabeto viene usato per capire che simboli possono essere trasformati in token. Il lexer usa un alfabeto Tipizzato mentre il parser usa un alfabeto ordinato per riuscire creare l'albero. AlphabetToken implementa sia OrderedAlphabet che TypedAlphabet e verrà usato sia nel lexer che nel parser. AlphabetToken contiene tutti i token della matematica operatori binari "+" "-"..... operatori unari sin... cos... numeri e parentesi, ogni token dovrà avere una priorita e un getValue.

2.2.7 Paolo Dettori

2.2.8 Mattia Gjyzeli

3 Testing Automatizzato

Per il testing abbiamo utilizzato JUnit5, i test sono stati fatti per non implementare feature che rompessero codice precedente.

3.1 Gianmarco Beligni

io mi sono cimentato nel fare i test per la costruzione della funzione che è l'oggetto più complicato dell'intero programma e delle relative sotto parti responsabili della loro creazione come lexer e parser, queste sono il core della applicazione e se non funzionano correttamente tutta la applicazione ne risentirebbe.

- TestParser
- TestLexer
- TestTokenExtractor
- TestExpression

3.1.1 Paolo Dettori

3.1.2 Mattia Gjyzeli

3.2 Metologia di lavoro

Version Control System Come sistema di versionamento distribuito (DVCS) è stato utilizzato Git, con repository hosting su GitHub. Questa scelta ha permesso una collaborazione efficiente e tracciabile tra tutti i membri del team.
Strategia dei Branch Si è adottata una metodologia ispirata a GitFlow, con la seguente struttura dei branch:
main: Branch principale contenente esclusivamente codice stabile e rilasciato
Branch Specifici per Sviluppatori Ogni membro del team ha lavorato su branch feature dedicati: Workflow di Sviluppo Il processo di sviluppo seguiva questo flusso:
Creazione branch: Ogni nuova feature iniziava da develop
Sviluppo isolato: Ogni sviluppatore lavorava sul proprio branch feature senza interferenze

- Test locali: Prima del push, verifica completa del codice
- Code Review: Revisione tra pari prima dell'approvazione del merge
- Merge e cleanup: Integrazione in develop ed eliminazione del branch feature
- fix: per correzioni di bug
- refactor: per refactoring del codice
- test: per test aggiuntivi
- Gestione delle Release
- Quando develop raggiungeva uno stato stabile:

Vantaggi della Strategia

- Isolamento: Sviluppo parallelo senza conflitti
- Tracciabilità: Storia completa di ogni feature
- Quality Control: Review obbligatoria prima del merge
- Rollback semplice: Possibilità di revert specifici
- Release pulite: main sempre stabile e testato

Questa organizzazione ha garantito un workflow ordinato e professionale, permettenci di sviluppare in parallelo le diverse componenti del sistema mantenendo un codice base sempre integro e funzionante.

3.2.1 Gianmarco Beligni

Riassumendo di cosa mi sono occupato: **Realizzazione dell'intero sistema di parsing e valutazione delle espressioni matematiche:**

- Progettazione e implementazione del lexer per la tokenizzazione delle espressioni
- Sviluppo del parser ricorsivo per la costruzione dell'albero sintattico astratto (AST)
- Creazione del sistema di validazione sintattica e semantica con gestione degli errori dettagliata
- Implementazione del meccanismo di valutazione ricorsiva delle funzioni matematiche

Realizzazione del sistema di gestione degli alfabeti di token:

- Creazione di `AlphabetToken` come implementazione centrale per la gestione dei simboli matematici
- Sistema di categorizzazione dei token per operazioni binarie, funzioni unarie e variabili

Sviluppo del motore di visualizzazione grafica:

- Implementazione del sistema di coordinate e trasformazioni tra coordinate cartesiane e schermo
- Creazione del meccanismo di generazione dei punti del grafico basato sull'AST
- Sviluppo degli algoritmi per il calcolo efficiente dei punti con gestione degli intervalli
- Implementazione delle funzionalità di zoom e pan per l'esplorazione del grafico

Progettazione dell'architettura a moduli con pattern creazionali:

- Implementazione del **Builder Pattern** per `LexerImpl` e `ParserImpl`
- Utilizzo del **Factory Pattern** per la creazione di token e generazione di alberi sintatici
- Applicazione del **Composite Pattern** per la struttura ad albero delle funzioni matematiche
- Implementazione del **Template Method Pattern** per strategie di generazione del grafico

Creazione del sistema di gestione errori avanzato:

- Sviluppo di errori specifici per il dominio (`SyntaxError`, `SemanticError`)
- Implementazione di meccanismi di validazione a più livelli (lessicale, sintattico, semantico)
- Sistema di reporting degli errori con messaggi descrittivi e posizionamento preciso

Questa implementazione ha richiesto una profonda comprensione sia dei concetti matematici sottostanti che dei pattern architetturali software, risultando in un sistema robusto, estensibile e ad alte prestazioni per l'analisi e visualizzazione di funzioni matematiche.

3.3 Note di Sviluppo

Paradigmi di Programmazione

- Programmazione Funzionale: Ampio utilizzo durante lo sviluppo, specialmente per la valutazione delle espressioni matematiche e la trasformazione di token. L'approccio funzionale ha permesso di scrivere codice più dichiarativo e meno soggetto a side effect.
- Optional: Massiccio uso di Optional in tutta l'implementazione, particolarmente nei metodi di parsing e valutazione. Questo approccio ha eliminato la necessità di controlli null e ha reso esplicita la possibilità di valori assenti, migliorando la chiarezza del codice.
- Generici: Sfruttati intensivamente per definire interfacce flessibili e riutilizzabili. In particolare:
  Alphabet per un sistema estendibile di token
  Parser per parser generici ma comunque bounded
  ErrorFinder per un sistema di validazione tipizzato
  guardare bene il codice sorgente per i generici questi sono solo esempi.
- Librerie e Framework Java Functional Interfaces: Utilizzo estensivo di Function, Supplier, Consumer e Predicate per creare codice modulare e componibile.
- Java Stream API: Ampio uso per operazioni su collezioni, filtraggio e trasformazione di dati, specialmente nell'implementazione di AlphabetToken e nei processi di validazione.
- Comparator e Comparable: Implementati per l'ordinamento dei token basato sulla priorità, fondamentale per il corretto parsing delle espressioni.

Pattern Architetturali

- Builder Pattern: Implementato per LexerImpl e ParserImpl per una costruzione flessibile e configurabile degli oggetti.
- Factory Pattern: Utilizzato per la creazione di token e la generazione di alberi sintattici attraverso TokenNodeFactory e TreeFactory.
- Composite Pattern: Applicato nella struttura ad albero di FunctionImpl dove ogni token può essere composto da altri token.
- Chain of Responsibility: Implementato nella valutazione ricorsiva dell'AST attraverso il metodo getValue() che propaga la chiamata attraverso tutti i nodi.

Gestione degli Errori

- Eccezzioni per valori: Creazione di SyntaxError e SemanticError per una gestione degli errori granulare e specifica al dominio, non ho usato exceptions per semplificare il flow del codice e semplificare l'uso futuro di questa API.
- Error Handling Functional: Utilizzo di pattern funzionali per la propagazione e gestione degli errori attraverso la catena di parsing.

Testing

- JUnit 5: Framework principale per il testing unitario di tutte le componenti del sistema.

4 Commenti Finali

4.1 Autovalutazione e lavori futuri

4.1.1 Gianmarco Beligni

Questo progetto è stato veramente un impresa, sono contento del risultato finale ma ci sono volute molte iterazioni e refactoring (per il continuo miglioramente delle mie conoscenze). Alla fine abbiamo raggiunto un API modulare ed estendibile con l'uso dei generici. Utilizzerò le conoscenze imparate per i miei futuri progetti come full-stack application attraverso il MVC pattern.

4.2 Difficolta incontrate e commenti per docenti

4.2.1 Gianmarco Beligni

Questo è stato uno dei progetti più difficili dove ho partecipato, l'idea di creare questa applicazione è stata mia poichè sono appassionato della matematica e imparando a programmare mi sono chiesto se potevo riuscire a creare un programma per risolvere gli studi di funzione che mi hanno tormentato per tutta la 4a e 5a superiore. Noi come team ci siamo riusciti attraverso concetti di OOP come modularità, polimorfismo, eredità, classi, oggetti che ci hanno permesso di dividere e conquistare il problema. Infine ringrazio la professoressa Montagna per avermi permesso di fare questo progetto.

5 Guida all'utilizzo

In alto a sinistra ci sono due textbox quella destra è per la funzione quella a sinistra per l'intervallo
Es $[3 * sin(x) + 2][(3;5)]$
Es 2 $[x^2 + y^2 - 36][(0;6)(0;6)]$
con il pulsante più si aggiungono delle textbox e con il pulsante meno si tolgono. Nel canvas si puo traslare tenendo premuto dentro il grafico e spostando il mouse e si può zoommare con la rotellina del mouse. Attenzione a digitare funzioni a più variabili molto complesse perchè sono molto dispendiose a livello di CPU dovendo iterare su tutti i pixel del canvas.