“Ingegneria del Software”
2022-2023

Docente: Professore Angelo Furfaro

<Mini-CAD>

Data	<gg/mm/aaaa>
Documento	Documento Finale – D3

Team Members		
Nome e Cognome	**Matricola**	**E-mail address**
Jhenalyn Subol	**213485**	**SBLJNL96D70Z216P@studenti.unical.it**

Mini-CAD

Sommario

Nella presente relazione viene documentata la progettazione e lo sviluppo del sistema MiniCAD, un mini-interprete di comandi destinato a essere integrato in un'applicazione esistente per la manipolazione di oggetti grafici bidimensionali.

Nell'applicazione esistente sono presenti tre tipi di oggetti: rettangoli, cerchi e immagini. In aggiunta, essa supporta già alcuni comandi base per la creazione e la manipolazione di questi oggetti. L'obiettivo è pertanto integrare un mini-interprete di comandi che consenta agli utenti di eseguire operazioni più avanzate.

Il mini-interprete di comandi di MiniCAD offre diverse funzionalità tra cui la creazione, rimozione, spostamento, ridimensionamento, visualizzazione delle proprietà, creazione e rimozione di gruppi, e il calcolo dell'area e del perimetro.

L'architettura della relazione è organizzata come segue:

A. (Stato dell'Arte) Viene descritto il sistema CAD, l'utilità di tale sistema, la sua evoluzione nel tempo e alcuni esempi tradizionali oggi utilizzati.
B. (Raffinamento dei Requisiti) Vengono elencati i requisiti del progetto, sia quelli funzionali sia quelli non funzionali oltre ai servizi esclusi da tale progetto e alle relative assunzioni. Inoltre, vengono descritti vari scenari d'uso del sistema corredati da un diagramma *use-case*.
C. (Architettura Software) Viene descritta l'architettura del sistema sia attraverso una vista statica che illustra l'organizzazione dei componenti, sia una vista dinamica che descrive il proprio comportamento durante l'esecuzione,
D. (Dati e loro Modellazione) In dettaglio, vengono descritte le informazioni (dati) strutturate e rappresentate all'interno del sistema.
E. (Scelte Progettuali) Vengono elencate le scelte progettuali che guidano lo sviluppo del sistema MiniCAD semplice, efficiente e user-friendly.
F. (Progettazione di Basso Livello) Vengono definiti, in maniera dettagliata, i componenti del sistema, le loro interazioni e l'implementazione dei servizi del sistema stesso. Vengono inoltre specificati i moduli, le classi, i metodi e gli algoritmi utilizzati al fine di realizzare le funzionalità previste dal sistema.
G. (Analisi del Soddisfacimento dei Requisiti Funzionali e Non Funzionali)Viene spiegato come lo sviluppo di MiniCAD soddisfa i requisiti funzionali e non funzionali.
H. (Prototipo) Viene mostrato un prototipo del progetto.

List of Challenging/Risky Requirements or Tasks

Challenging Task	Date the task is identified	Date the challenge is resolved	Explanation on how the challenge has been managed

A. Stato dell'Arte

Evoluzione e Importanza dei Sistemi CAD
Il CAD (*Computer-Aided Design*) è un sistema che utilizza software e hardware del computer per assistere durante la creazione, la modifica, le analisi o l'ottimizzazione di un progetto (Sarcar et al. 3). Questo strumento è fondamentale in ingegneria, in architettura e nella produzione, in quanto aumenta la produttività del progettista, migliora la qualità del design e fornisce una base documentale per il progetto stesso.

Il CAD è emerso negli anni '60 come risposta ai problemi e ai frequenti errori legati alla progettazione manuale.

In particolare, la necessità di disegnare a mano progetti complessi e la crescente richiesta di automazione nei processi produttivi hanno spinto l'industria e il mondo accademico a sviluppare i sistemi rappresentandoli matematicamente su carta. (Ball 4)

I moderni sistemi CAD si basano sulla grafica computerizzata interattiva (o *interactive computer graphics*, ICG) che consente agli utenti di interagire direttamente con il sistema. L'uso di software specializzati come l' ICG consente agli utenti del CAD di creare modelli digitali tridimensionali e complessi, così facilitando il processo di progettazione, analisi e produzione.

Alcuni dei software CAD più noti e più utilizzati al giorno d'oggi includono:

- AutoCAD - uno dei software CAD più popolari, sviluppati da Autodesk. E' ampiamente utilizzato per il disegno 2D e 3D e supporta un'ampia gamma di settori.
- SolidWorks - è un software di modellazione 3D utilizzato prevalentemente nel design di prodotti e nell'ingegneria meccanica.
- SketchUp - conosciuto per la sua facilità d'uso, è spesso utilizzato in architettura e design di interni.

La Sfida della Complessità nei Sistemi CAD Moderni
Negli ultimi anni, i sistemi CAD si sono evoluti notevolmente, diventando strumenti potenti e sofisticati. Questa evoluzione ha anche incrementato la complessità delle interfacce utenti, con comandi avanzati essenziali per la creazione di modelli complessi.

Il Progetto MiniCAD: Un CAD semplice e immediata
Il progetto mira a creare un sistema CAD traendone ispirazione dai sistemi moderni e semplificando notevolmente l'uso e il suo sviluppo. Il sistema si basa su un interprete di comandi, semplice e integrabile in un'applicazione esistente idonea a manipolare gli oggetti grafici bidimensionali.

L'impiego di design pattern specifici consente al sistema MiniCAD di eseguire operazioni specifiche in modo rapido ed efficiente, garantendo al contempo che l'architettura del sistema rimanga modulare, robusta e facilmente manutenibile.

Grammatica EBNF: La base del comando in MiniCAD

Una delle caratteristiche distintive di MiniCAD è l'adozione di una grammatica EBNF(*Extended Backus-Naur Form*) per definire formalmente i comandi del sistema.
La grammatica EBNF è una notazione estesa per la descrizione delle grammatiche dei linguaggi di programmazione. Questo approccio consente un programmatore o un compilatore di determinare se un programma è sintatticamente corretto, cioè se aderisce alle regole grammaticali e di punteggiature del linguaggio di programmazione.

Nell'applicazione MiniCAD, la grammatica dei comandi viene definita nel seguente modo:

<cmd>::=<create>|<remove>|<move>|<scale>|<list>|<group>|<ungroup>|<area>|<perimeter>
<create>::= **new** <typeconstr> <pos>
<remove>::= **del** <objID>
<move>::= **mv** <objID> <pos> | **mvoff** <objID> <pos>
<scale>::= **scale** <objID> <posfloat>
<list>::= **ls** <objID>| **ls** <type> | **ls all** | **ls groups**
<group>::= **grp** <listID>
<ungroup>::= **ungrp** <objID>
<area>::= **area** <objID>| **area** <type> | **area all**
<perimeter>::= **perimeter** <objID>| **perimeter** <type> | **perimeter all**
<pos>::=**(** <posfloat> **,** <posfloat> **)**
<typeconstr>::= **circle (**<posfloat>**)** | **rectangle** <pos> | **img (**<path>**)**
<type>::= **circle** | **rectangle** | **img**
<listID>::= <objID> { **,** <objID> }

Le seguenti sono esempi di comandi seguendo la grammatica EBNF:

1. Creazione di oggetti:
```
create circle (5.0) (3.1,4.5)
create img ("./pippo.png") (6.1,4.6)
```
2. Rimozione di oggetti:
```
del id1
```
3. Spostamento di oggetti:
```
mv id1 (5.9,8.2)
mvoff id1 (5.9,8.2)
```
4. Ridimensionamento di oggetti:

```
scale id1 2.0
```

5. Visualizzazione delle proprietà:
```
ls id1
ls circle
ls all
ls groups
```
6. Creazione e rimozione di gruppi:
```
grp id1, id2, id3
ungrp gid
```
7. Calcolo dell'area e del perimetro:
```
area id1
perimeter rectangle
area all
```

Conclusione

In sintesi, MiniCAD si propone come uno strumento utile per utenti che necessitano di una soluzione CAD leggera, semplice e immediata. Grazie alla semplicità e all'efficienza del sistema, MiniCAD offre un'architettura modulare, facilitandone l'uso e la manutenzione. Queste caratteristiche sono gli obiettivi principali su cui mi sono concentrata durante lo sviluppo del progetto.

B. Raffinamento dei Requisiti

B.1 Servizi (con prioritizzazione)

ID	Requisito	Descrizione	Priorità	
			Importanza	Complessità
S1	**Creazione di Oggetti Grafici**	• Consente di creare diversi tipi di oggetti grafici bidimensionali quali cerchi, rettangoli e immagini. • Ad ogni oggetto creato viene assegnato un ID univoco per effettuare la manipolazione successiva.	Alta	Media
S2	**Rimozione di Oggetti**	• Consente di rimuovere oggetti o gruppi di oggetti tramite il loro ID.	Alta	Bassa
S3	**Spostamenti di Oggetti**	• Consente di spostare gli oggetti. • Sono disponibili due tipi di spostamento: 1. *Spostamento Assoluto* – l'oggetto viene spostato in base alla nuova coppia di coordinate fornita dall'utente 2. *Spostamento Relativo* – l'oggetto viene spostato rispetto alla posizione attuale facendo la sua somma con la posizione fornita dall'utente	Media	Media
S4	**Ridimensionamento di Oggetti**	• Consente di ridimensionare gli oggetti grafici applicando un fattore di scala.	Media	Media
S5	**Visualizzazione delle Proprietà**	• Fornisce informazioni dettagliate sulla proprietà degli oggetti, dei gruppi di oggetti o di tutti gli oggetti presenti nel disegno. • Offre diversi tipi di funzionalità: 1. *Visualizzazione delle proprietà di un oggetto* - Tramite questo servizio, è possibile visualizzare tutte le proprietà rilevanti all'oggetto scelto, ad esempio il	Alta	Alta

ID	Requisito	Descrizione	Priorità	
			Importanza	Complessità
		tipo, la dimensione, la posizione, e così via.. 2. *Visualizzazione dell'elenco di oggetti di un certo tipo* – Funge come un filtro di oggetti, in cui, dato il tipo di oggetto che l'utente vuole visualizzare, il sistema effettua il filtraggio facendo vedere tutti gli oggetti di quel tipo. 3. *Visualizzazione di tutti gli oggetti* – Permette la visualizzazione completa di tutti gli oggetti presenti nel pannello grafico. 4. *Visualizzazione dei gruppi di oggetti* – Permette la visualizzazione di tutti i gruppi di oggetti e i loro membri.		
S6	**Gestione dei Gruppi di Oggetti**	● Permette di creare e gestire gruppi di oggetti. ● Sono disponibili due tipi di operazioni: 1. *Creazione di un gruppo* – è possibile scegliere degli oggetti esistenti nell'area di lavoro e raggrupparli per poterli manipolare contemporaneamente. 2. *Rimozione di un Gruppo* – il raggruppamento degli oggetti viene sciolto ma gli oggetti persistono nell'area di lavoro.	Media	Alta
S7	**Calcolo delle Area e dei Perimetri**	● Consente di calcolare le aree e i perimetri di : 1. un singolo oggetto 2. gruppi di oggetti 3. oggetti di un determinato tipo 4. tutti gli oggetti	Media	Alta

B.2 Requisiti non Funzionali

Elencare i requisiti non funzionali più' importanti per il vostro Sistema

1. **Affidabilità**
   - Il sistema deve essere in grado di gestire errori in modo robusto e deve informare l'utente di eventuali problemi senza compromettere la stabilità dell'applicazione.
   - L'operazione di *undo* deve funzionare in modo accurato per garantire che gli utenti possano annullare le modifiche senza errori.
2. **Usabilità**
   - L'interfaccia utente deve essere intuitiva e facile da usare.
3. **Manutenibilità**
   - Il codice deve essere modulare facilitando future estensioni e manutenzioni.
4. **Testabilità**
   - Il sistema deve includere un framework di test per testare le funzionalità principali e garantire che il sistema funzioni come previsto.
   - Deve essere possibile testare singoli moduli in isolamento per garantire la correttezza delle implementazioni

B.3 Scenari d'uso dettagliati

Descrivere gli scenari più comuni, più interessanti, o più complicati d'uso dei vostri servizi.

Caso d'uso	**Crea nuovo oggetto grafico**
Descrizione	Creare un nuovo oggetto (cerchio,rettangolo, immagine, gruppo di oggetti) in una posizione specifica dell'area di lavoro.
Tipologia	Primario
Svolgimento	1. L'utente invia il comando `'create <typeConstructor> <position>'` al sistema. 2. Il Lexer del sistema prende l'input e lo divide in una sequenza di token.

	3. Il Parser prende i token generati e li analizza. Esso costruisce un oggetto di comando che rappresenta il comando in una forma strutturata. Una volta che il comando è stato creato, il parser invia l'oggetto di comando al sistema per l'esecuzione. 4. Il sistema esegue il comando, creando l'oggetto specificato e lo aggiunge al pannello grafico. 5. Il sistema restituisce l'ID univoco dell'oggetto creato.
Postcondizioni	● Un nuovo oggetto viene creato e aggiunto al sistema. ● Viene assegnato un ID univoco e le sue caratteristiche sono memorizzate nel sistema.

Caso d'uso	**Modifica Oggetto Grafico**
Descrizione	Modificare l'oggetto grafico creato eseguendo operazioni di rimozione, spostamento o ridimensionamento.
Tipologia	Primario
Svolgimento	1. L'utente invia il comando `:` a. `del <id>` per la rimozione b. `mv <id> <position>` per lo spostamento assoluto c. `mvoff <id> <position>` per lo spostamento relativo d. `scale <id> <scaleFactor>` per il ridimensionamento 2. Il Lexer del sistema prende l'input e lo divide in una sequenza di token. 3. Il Parser prende i token generati e li analizza per formare una struttura dati interpretabile dal sistema. Viene identificato il tipo di comando e i relativi parametri. 4. Il sistema identifica l'oggetto tramite l'ID fornito nel comando. Utilizza il contesto (*Context*) per trovare l'oggetto grafico corrispondente. 5. Il sistema esegue l'operazione specificata (rimozione, spostamento, ridimensionamento) dal comando.

	6. Il pannello viene aggiornato per riflettere le modifiche. L'oggetto viene ridisegnato o rimosso dalla vista, a seconda del comando eseguito.
Estensioni	5a. L'utente vuole rimuovere un oggetto che fa parte di un gruppo precedentemente creato. .1 Il sistema elimina l'oggetto dal gruppo creato .2 Il sistema elimina l'oggetto dall'elenco degli oggetti creati.

Caso d'uso	**Visualizza proprietà**
Descrizione	Visualizzare le proprietà dell'oggetto, l'elenco degli oggetti dello stesso tipo, l'elenco di tutti i gruppi, e l'elenco di tutti gli oggetti presenti nell'area di lavoro
Tipologia	Primario
Svolgimento	1. L'utente invia il comando: a. `ls <ID>` per visualizzare le proprietà dell'oggetto o del gruppo di oggetti b. `ls <type>` per visualizzare l'elenco degli oggetti del tipo indicato c. `ls all` per visualizzare l'elenco di tutti gli oggetti presenti nell'area di lavoro d. `ls groups` per visualizzare l'elenco degli identificatori di tutti i gruppi 2. Il Lexer del sistema prende l'input e lo divide in una sequenza di token. 3. Il Parser prende i token generati e li analizza per formare una struttura dati interpretabili dal sistema. 4. Il sistema utilizza il *Context* per recuperare le informazioni relative all'oggetto specificato dall'ID, al tipo di oggetto o all'elenco degli oggetti. 5. Il sistema visualizza le proprietà degli oggetti.

Caso d'uso	**Crea gruppi di oggetti**

Descrizione	Creare un gruppo di oggetti grafici identificati da una lista di ID
Tipologia	Primario
Svolgimento	1. L'utente invia il comando `'grp <listId>'` per creare un gruppo di oggetti. 2. Il Lexer del sistema prende il comando e lo divide in una sequenza di token. 3. Il Parser prende i token generati e li analizza, verificando la loro correttezza e successivamente vengono determinate le azioni da intraprendere. 4. Il sistema crea un nuovo gruppo contenente gli oggetti identificati dagli ID forniti. 5. Il sistema restituisce l'ID del gruppo creato confermando la riuscita dell'operazione.
Precondizioni	Gli oggetti che l'utente desidera raggruppare esistono già nel sistema e sono identificabili tramite ID univoci.
Postcondizioni	● Un nuovo gruppo di oggetti viene creato nel sistema. ● L'ID del nuovo gruppo è restituito.
Estensioni	1a. Se uno degli ID non è valido 1. Il sistema restituisce un messaggio di errore.
Requisiti Speciali	● Gli oggetti da raggruppare possono essere di tipi diversi. ● L'ID del gruppo deve essere univoco.

Caso d'uso	**Rimuove gruppi di oggetti**
Descrizione	Rimuovere un gruppo di oggetti grafici senza eliminare i singoli componenti.
Tipologia	Primario
Svolgimento	1. L'utente invia il comando `'ungrp <group id>'` per rimuovere il gruppo. 2. Il Lexer del sistema esamina il comando e lo suddivide in una sequenza di token.

	3. Il Parser analizza i token per verificare la loro correttezza e determinare le azioni da intraprendere. Il parser conferma che il comando è valido e che il gruppo e gli oggetti esistono nel sistema. 4. Il sistema identifica il gruppo tramite l'ID fornito. 5. Il sistema rimuove il gruppo, mantenendo intatti i singoli oggetti. 6. Il sistema restituisce un messaggio di conferma all'utente, indicando che gli oggetti sono stati rimossi dal gruppo con successo.
Precondizioni	● Il gruppo di oggetti da cui rimuovere gli oggetti esiste nel sistema e può essere identificato tramite un ID ● Gli oggetti da rimuovere sono identificabili tramite ID e fanno parte del gruppo
Postcondizioni	● Il gruppo viene rimosso dal sistema. ● Gli oggetti rimangono nel sistema.
Estensioni	1a. Se l'ID non è valido 1. il sistema restituisce un messaggio di errore.

Caso d'uso	**Calcolo dell'area**
Descrizione	Calcolare l'area di un oggetto grafico, di un gruppo di oggetti o di tutti gli oggetti.
Tipologia	Primario
Precondizioni	L'oggetto di cui si desidera calcolare l'area esiste nel sistema e può essere identificato tramite un ID
Postcondizioni	Il sistema calcola e restituisce l'area dell'oggetto specificato.
Scenario principale di successo (main scenario)	1. L'utente invia il comando a. `area <objID>` per calcolare l'area dell'oggetto b. `area <type>` per calcolare l'area di tutti gli oggetti del tipo indicato

	c. `area all` per calcolare l'area di tutti gli oggetti 2. Il Lexer esamina il comando e lo suddivide in una sequenza di token 3. Il Parser analizza i token per verificarne la correttezza e determinare le azioni da intraprendere. Il parser conferma che il comando è valido e che l'ID dell'oggetto esiste nel sistema. 4. Il sistema recupera le caratteristiche dell'oggetto identificato dall'ID fornito. 5. Il sistema calcola l'area dell'oggetto in base al suo tipo e alle sue dimensioni. Ad esempio: a. Per un cerchio: Area = π * r^2 b. Per un rettangolo: Area = larghezza * altezza c. Per un'immagine: Area = larghezza * altezza 6. Il sistema restituisce il risultato del calcolo.

Caso d'uso	**Calcolo del perimetro**
Descrizione	Calcolare il perimetro di un oggetto grafico, di un gruppo di oggetti dello stesso tipo o di tutti gli oggetti.
Tipologia	Primario
Svolgimento	1. L'utente invia il comando a. `perimetro <objID>` b. `perimetro <type>` c. `perimetro all.` 2. Il Lexer esamina il comando e lo suddivide in una sequenza di token 3. Il Parser analizza i token per verificarne la correttezza e determinare le azioni da intraprendere. Il parser conferma che il comando è valido e che l'ID dell'oggetto esiste nel sistema. 4. Il sistema recupera le caratteristiche dell'oggetto identificato dall'ID fornito. 5. Il sistema calcola l'area dell'oggetto in base al suo tipo e alle sue dimensioni. Ad esempio: a. Per un cerchio: Perimetro= 2*π * r

	b. Per un rettangolo: Perimetro= 2*larghezza + 2* altezza c. Per un'immagine: Perimetro= 2* larghezza + 2* altezza 6. Il sistema restituisce il risultato del calcolo.

B.4 Excluded Requirements

Le seguenti funzionalità sono state escluse per rendere più semplice il sistema. Identificare i requisiti esclusi mi ha aiutato a mantenere il focus del progetto sui bisogni principali del sistema.

1. **Supporto per Oggetto Grafici Avanzati** - Il progetto richiede di concentrarsi su oggetti grafici semplici come cerchi, rettangoli, immagini e gruppi di oggetti. L'aggiunta di forme avanzate richiederebbe una complessità maggiore nella progettazione.
2. **Editor Grafico Interattivo** - L'interfaccia attuale è basata su comandi testuali e su un campo di testo per l'inserimento dei parametri. Un'interfaccia grafica per la manipolazione visiva e interattiva degli oggetti (*drag-and-drop*, ridimensionamento grafico) potrebbe richiedere un'interfaccia complessa e ulteriori risorse di sviluppo.

B.5 Assunzioni

1. **Semplificazione dei Tipi di Oggetti**
   Saranno supportati solo tre tipi di oggetti semplici bidimensionali (cerchi, rettangoli e immagini). Non verranno aggiunti oggetti tridimensionali in futuro.

2. **File di input**
   Si assume che i percorsi dei file forniti per le immagini siano validi e che l'immagine sia disponibile in un formato compatibile. Il sistema non gestirà errori dovuti a file inesistenti o formati non supportati.

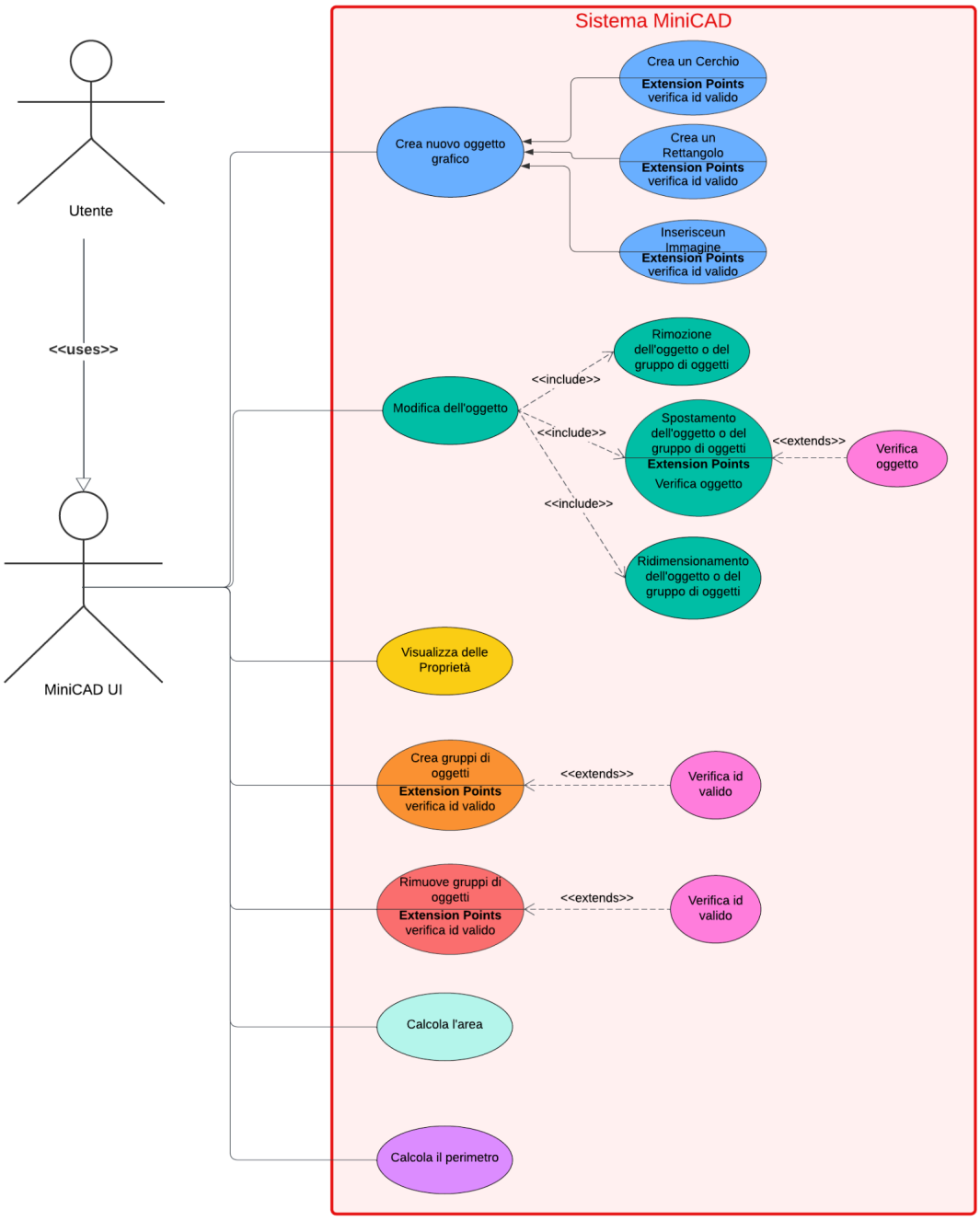
3. **Creazione di Oggetti Grafici**
   All'inizio, la dimensione e la posizione degli oggetti sono già impostate e predefinite. L'utente può modificare queste proprietà utilizzando i pulsanti disponibili nel pannello laterale.

4. **Cancellazione dell'oggetto che fa parte di un gruppo esistente**
   Quando si rimuove un oggetto che fa parte di un gruppo precedentemente creato, l'oggetto viene rimosso definitivamente sia dall'elenco degli oggetti sia dal gruppo. Tuttavia, il gruppo continua a persistere anche dopo la rimozione di un suo membro.

5. **Cancellazione di un gruppo di oggetti**
   Quando viene passato come parametro del comando del l'ID di un gruppo di oggetti, il gruppo viene cancellato e anche gli oggetti che costituiscono vengono definitivamente cancellati.

6. **Prestazioni**
   Si assume che il sistema non gestirà un numero elevato di oggetti (es. migliaia), quindi l'efficienza non sarà una priorità critica in questa fase.

B.6 Use Case Diagrams

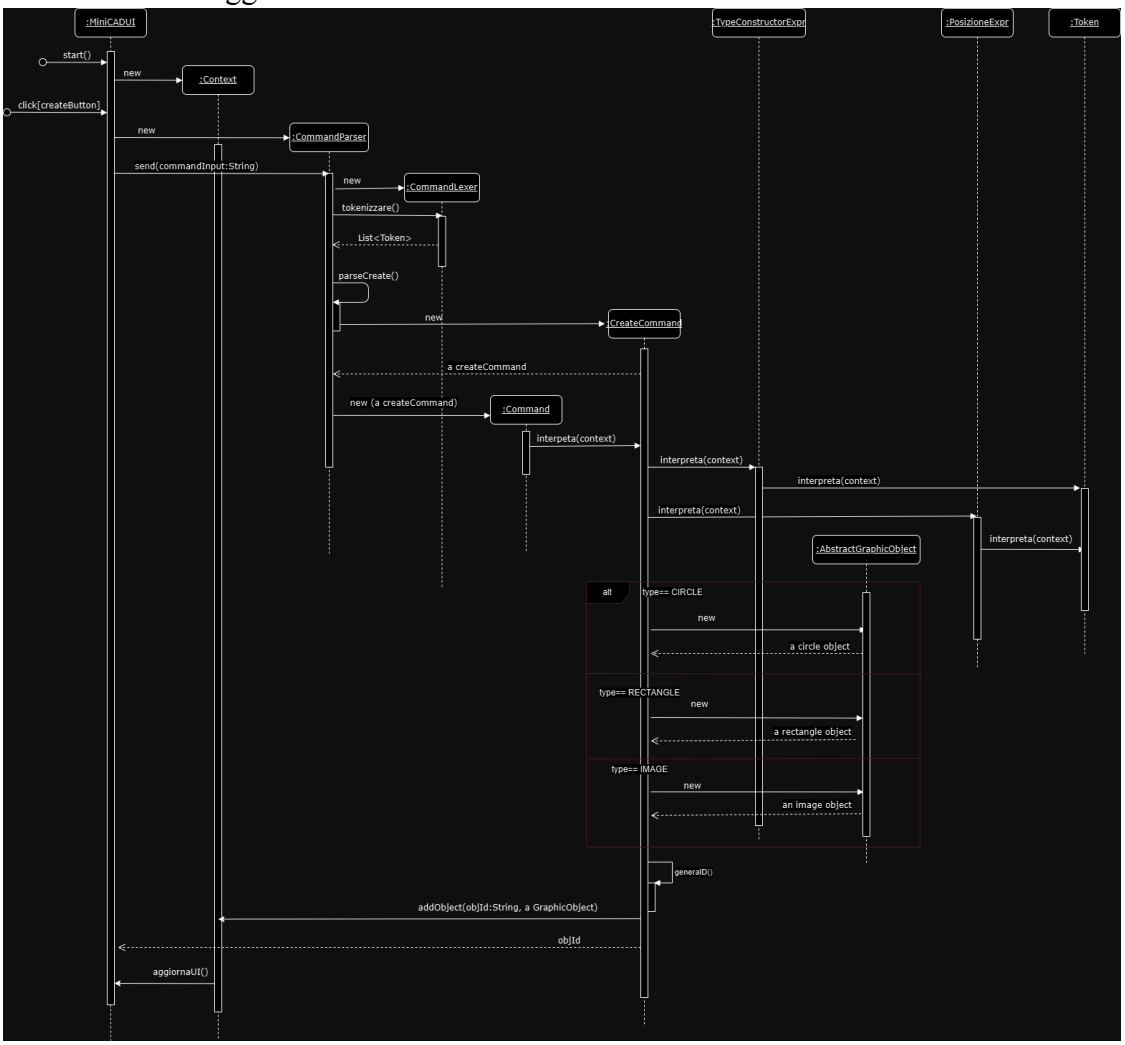


C. Architettura Software

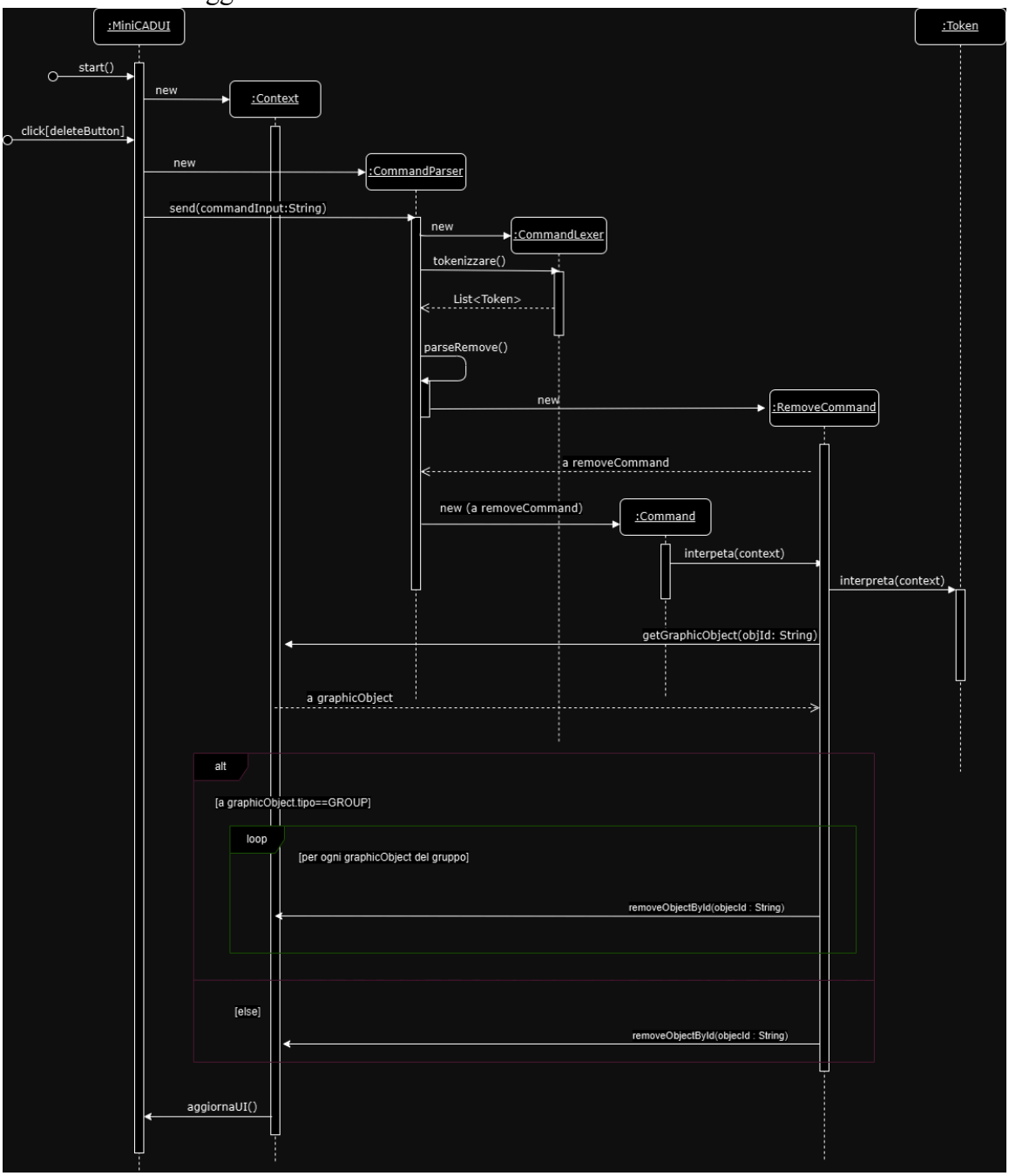
C.1 The static view of the system: Component Diagram

C.2 The dynamic view of the software architecture: Sequence Diagram

Creazione di un oggetto



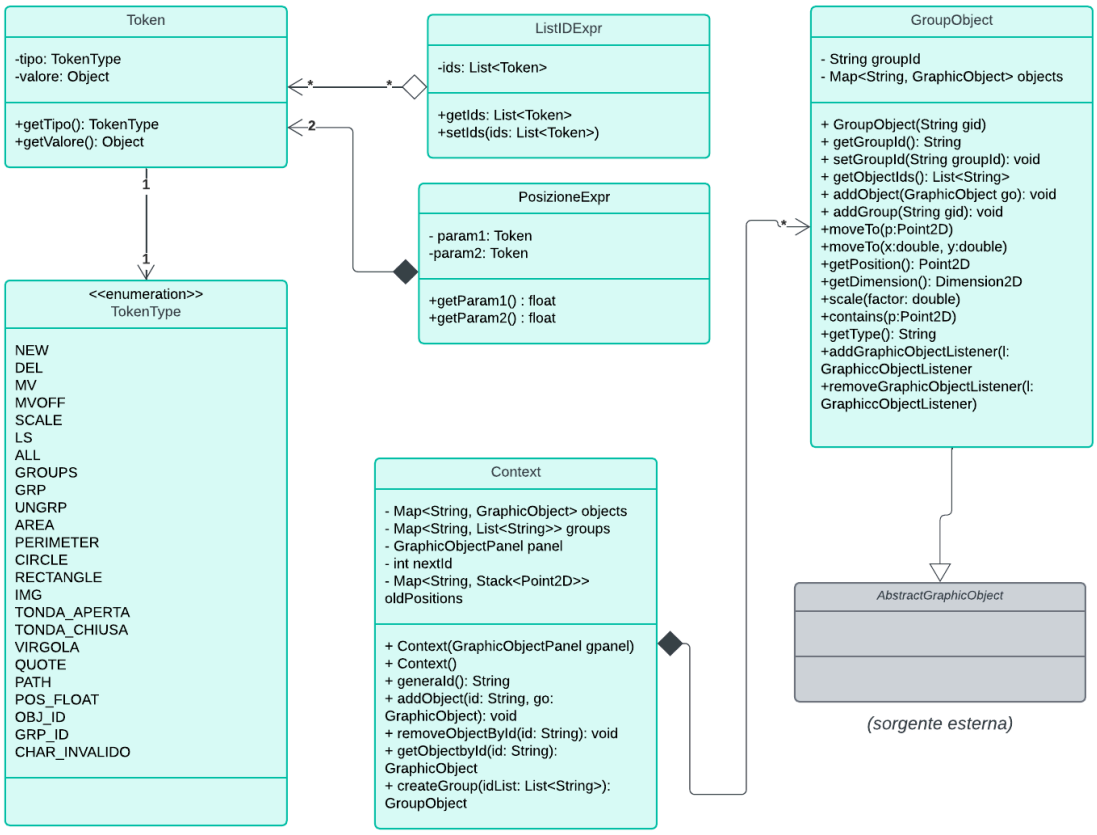
Rimozione di un oggetto



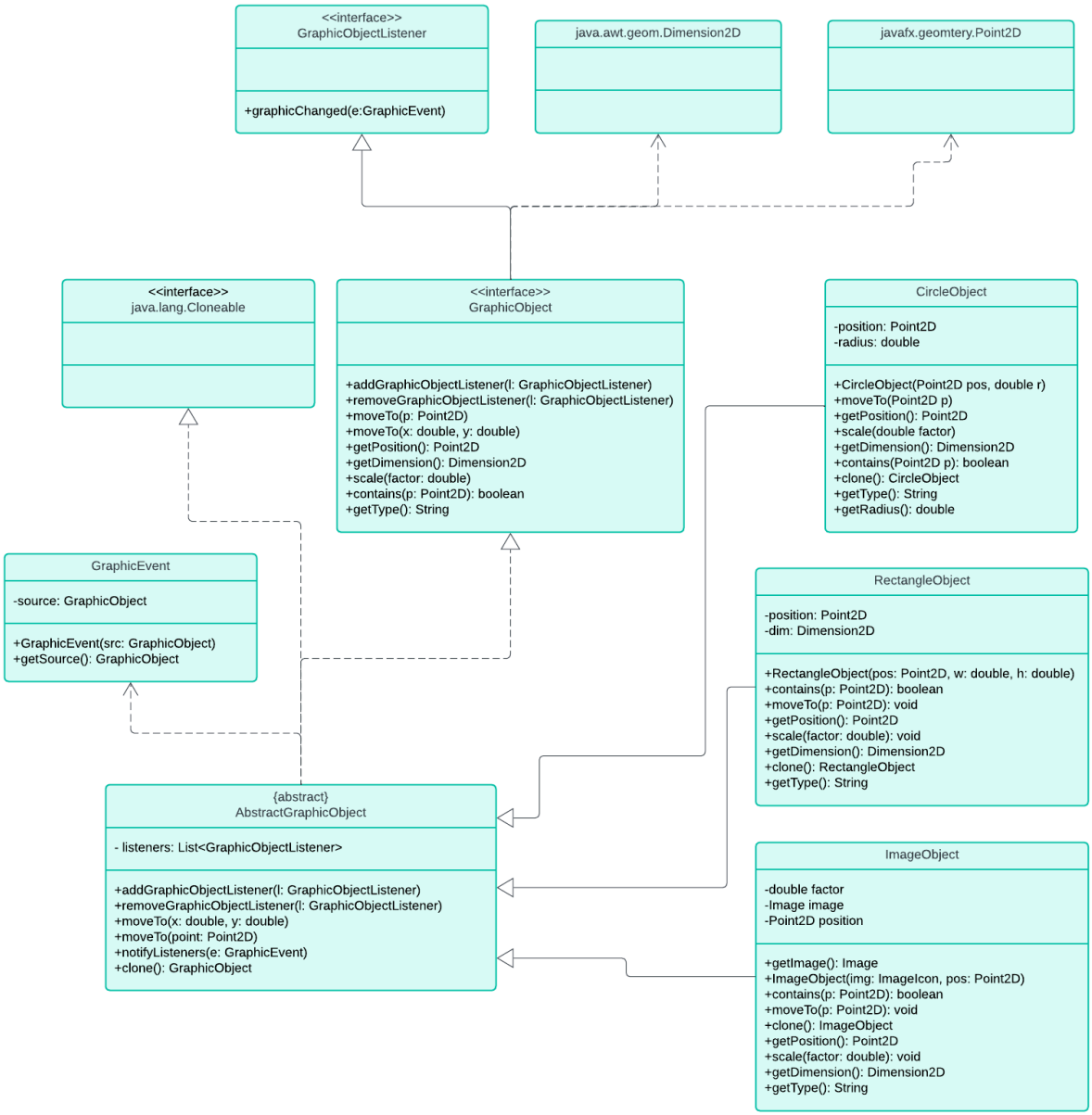
D. Dati e loro modellazione (se il sistema si interfaccia con un DBMS)

Il sistema MiniCAD è progettato per integrarsi ad un'applicazione esistente che manipola degli oggetti grafici bidimensionali. L'applicazione esterna fornisce i dati e consente a MiniCAD di accedervi tramite l'interfaccia *GraphicObject* e le classi concrete. L'interfaccia e le classi consentono a MiniCAD di manipolare e di recuperare le informazioni necessarie agli oggetti grafici.
Oltre ai dati provenienti dalla sorgente esterna, MiniCAD gestisce i dati relativi ai gruppi di oggetti nonché i comandi eseguiti sugli oggetti grafici. I dati in argomento vengono gestiti tramite la classe *Context*, nella quale sono implementati i metodi per la creazione, per la modifica e per la rimozione di gruppi.

1. Sorgente Interna (Sistema MiniCAD)



2. Sorgente Esterna (Applicazione Esterna)



E. Scelte Progettuali (Design Decisions)

1. Architettura del Sistema

Il Progetto MiniCAD è stato sviluppato come un'estensione di un'applicazione esistente, che già utilizza l'architettura MVC. La coerenza architetturale è stata mantenuta per facilitare l'integrazione del sistema e sfruttare al meglio le componenti esistenti.

Architettura MVC

Model-View-Controller (MVC) è un pattern utilizzato in programmazione per dividere il codice in blocchi dalle funzionalità ben distinte:

- *Model* : contiene i metodi di accesso ai dati
- *View* : si occupa di visualizzare i dati all'utente e gestisce l'interazione fra quest'ultimo e l'infrastruttura sottostante.
- *Controller* : riceve i comandi dell'utente attraverso il View e reagisce eseguendo delle operazioni che possono interessare il Model e che portano generalmente ad un cambiamento di stato del View. [1]

Integrazione del MiniCAD nell'Architettura Esistente

- **Model**
  - Vengono utilizzate le stesse classi e interfacce nel package *model* dell'applicazione esistente.
  - E' stata implementata la classe *GroupObject* che estende la classe astratta *AbstractGraphicObject* per la gestione dei gruppi.
- **View**
  - Vengono utilizzate le componenti del pacchetto *view* dell'applicazione esistente.
  - E' stata implementata solo la classe *CreateObjectActionMiniCad* che estende la classe *AbstractAction* di *Swing* per gestire le azioni relative alla creazione degli oggetti grafici.
- **Controller**
  - E' stata implementata una nuova classe controller per riconoscere i nuovi comandi del MiniCAD.
- **SpecificCmds**

[1] Camarlinghi, Francesco. "Pattern MCV: cos'è, vantaggi e come utilizzarlo." *HTML.it*, 24 June 2009, https://www.html.it/pag/18299/il-pattern-mvc/.

    - Questo package contiene l'implementazione di tutti i comandi specifici per il MiniCAD. A differenza dell'applicazione esistente, ho utilizzato il design pattern Interpreter per implementare i comandi specifici del sistema.
- **UndoManager**
    - E' stata implementata una nuova classe *MiniCADHistoryCommandHandler* anziché utilizzare la classe *HistoryCommandHandler* dell'applicazione esterna. Questa scelta mi ha consentito di integrare la funzionalità di Undo ai comandi implementati seguendo il pattern Interpreter.

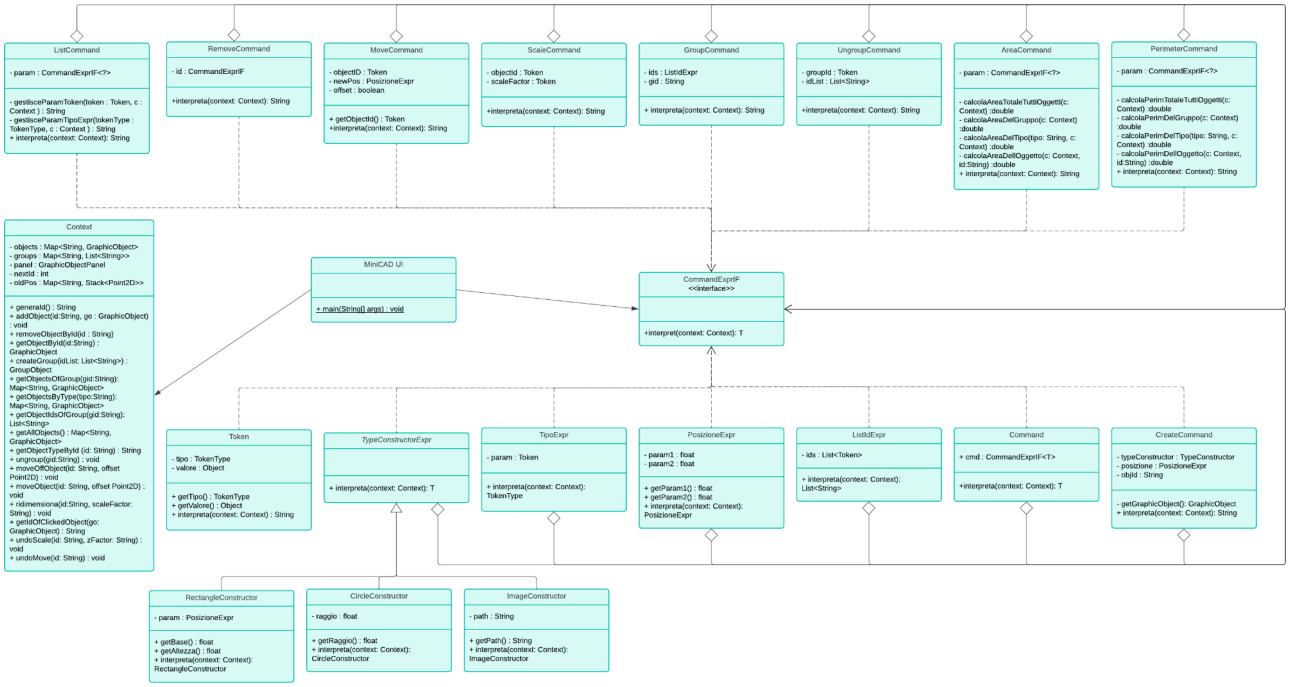
2. Utilizzo dell'Interpreter Pattern per l'implementazione dei comandi specifici

L'utilizzo del *Pattern Interpreter* è motivato dalla necessità di interpretare ed eseguire i comandi testuali dei vari servizi offerti dal sistema. Inoltre, la traccia del progetto ha fornito la grammatica EBNF (*Extended Backus-Naur Form*) che definisce chiaramente la sintassi dei comandi, rendendo quindi il pattern *Interpreter* una scelta naturale per l'implementazione . Grazie alla grammatica EBNF, la struttura dei comandi viene descritta in modo precisa. Essa facilita anche la creazione di un parser che possa interpretare e trasformare i comandi testuali in azioni eseguibili.

Partecipanti

- **Abstract Expression (CommandExpression):** l'interfaccia che dichiara il metodo `interpreta(...)` e prende un contesto come argomento. Tutte le espressioni concrete devono implementare questa interfaccia
- **Terminal Expression (Token) :** espressione che rappresenta il nodo figlio dell'albero sintattico.
- **Nonterminal Expression (Command, CreateCommand, RemoveCommand, ScaleCommand, ListCommand, MoveCommand, GroupCommand, UngroupCommand, AreaCommand, PerimeterCommand, TipoExpr, ListIdExpr, PosizioneExpr, TypeConstructorExpr) :** espressioni che rappresentano le regole grammaticali non terminali. Combinano altre espressioni (sia terminali che non terminali) e implementano il metodo `interpreta(...)` per interpretare la combinazione.
- **Context (Context) :** contiene informazioni globali necessarie durante l'interpretazione delle espressioni.
- **Client (MiniCADGUI) :** implementa l'interfaccia utente dell'applicativo.

Struttura



3. Utilizzo del Pattern Composite per l'implementazione degli oggetti gruppi

Il Composite Pattern è un pattern strutturale che semplifica la gestione di oggetti singoli e gruppi di oggetti in modo uniforme. E' particolarmente utile quando abbiamo una struttura gerarchica complessa, come alberi, in cui possiamo trattare sia elementi singoli (Cerchio, Rettangolo, Immagine) sia gruppi di elementi (GroupObject) nello stesso modo.

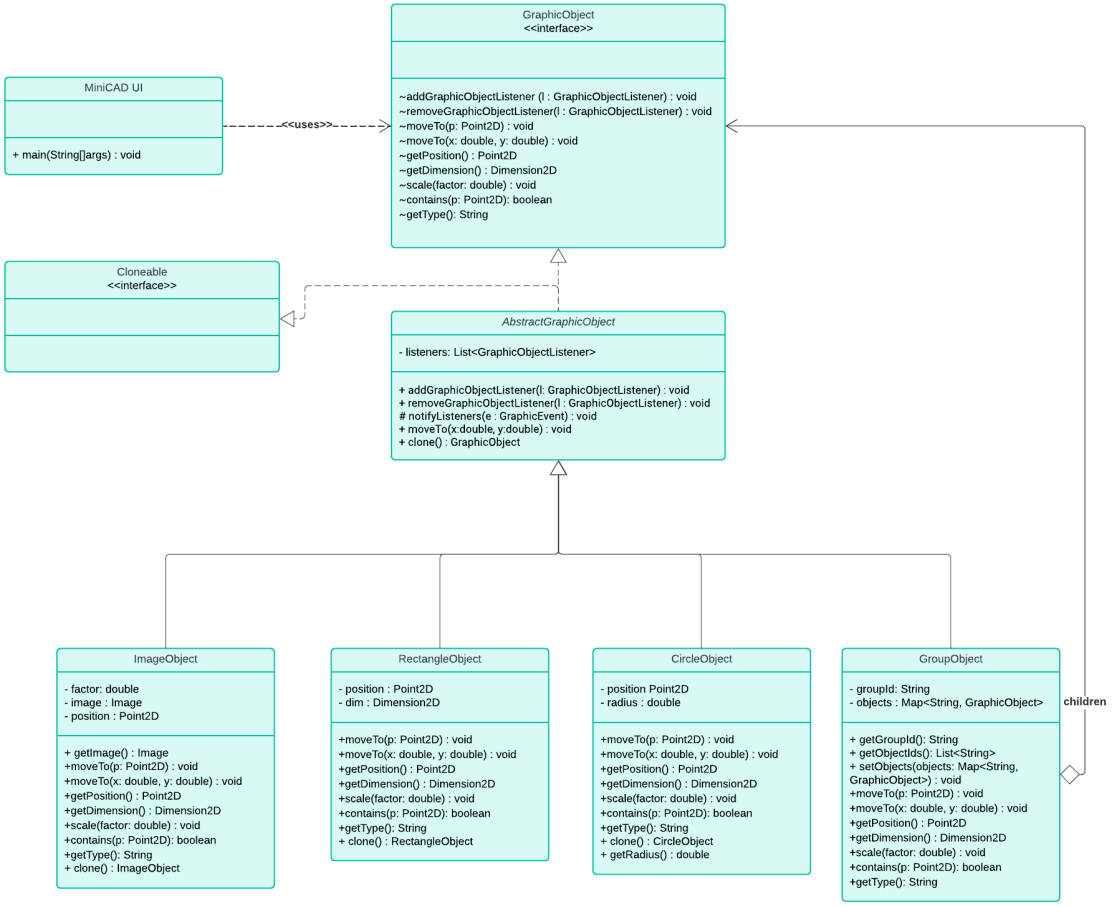
Questo pattern consente di applicare le stesse operazioni a singoli oggetti e a gruppi di oggetti, rendendo la gestione più semplice e coerente.

Partecipanti

- **Component (GraphicObject) :** l'interfaccia comune che definisce le operazioni comuni per gli oggetti.

- **Leaf (RectangleObject, CircleObject, ImageObject) :** oggetti che implementano l'interfaccia *GraphicObject* e rappresentano gli oggetti grafici singoli.
- **Composite (GroupObject) :** è il compositore che può contenere una collezione di *GraphicObject* e altri *GroupObject*, permettendo di trattare singoli oggetti e gruppi in modo uniforme.

Struttura



4. Implementazione della funzionalità di "Undo"

L'operazione *Undo* è un'operazione essenziale per migliorare in maniera significativa l'usabilità e affidabilità del sistema.
La funzionalità consente l'annullamento degli effetti di una o più operazioni precedentemente eseguite , riportando il disegno a uno stato precedente. Occorre

quindi definire chiaramente le operazioni specifiche annullabili per poter garantire che questa funzionalità sia effettivamente efficace e utile per gli utenti.

Operazioni specifiche per le quali si può effettuare l'undo :

Operazioni scelte	**Motivazioni**
Creazione di Oggetti Grafici	Gli utenti potrebbero commettere errori durante la creazione di oggetti, ad esempio, aggiungere per errore oggetti indesiderati.
Rimozione di Oggetti	La rimozione accidentale di oggetti può comportare la perdita di dati importanti. La funzionalità di undo permette di ripristinare facilmente gli oggetti rimossi senza doverli ricreare da zero.
Spostamento di Oggetti	Gli oggetti potrebbero essere spostati in posizioni indesiderate o errate. L'*undo* riduce il tempo necessario per riposizionare manualmente gli oggetti spostati accidentalmente.
Ridimensionamento di Oggetti	Gli utenti possono riparare le modifiche apportate alla dimensione dell'oggetto tornando allo stato precedente tramite l' undo.
Creazione e Modifica del gruppo di oggetti	L'utente può aggiungere per errore oggetti non desiderati ad un gruppo.. L' *undo* permette di meglio gestire l'errore senza dover cancellare tutti gli oggetti di quel gruppo.

5. Utilizzo del Command Pattern per la funzionalità di Undo

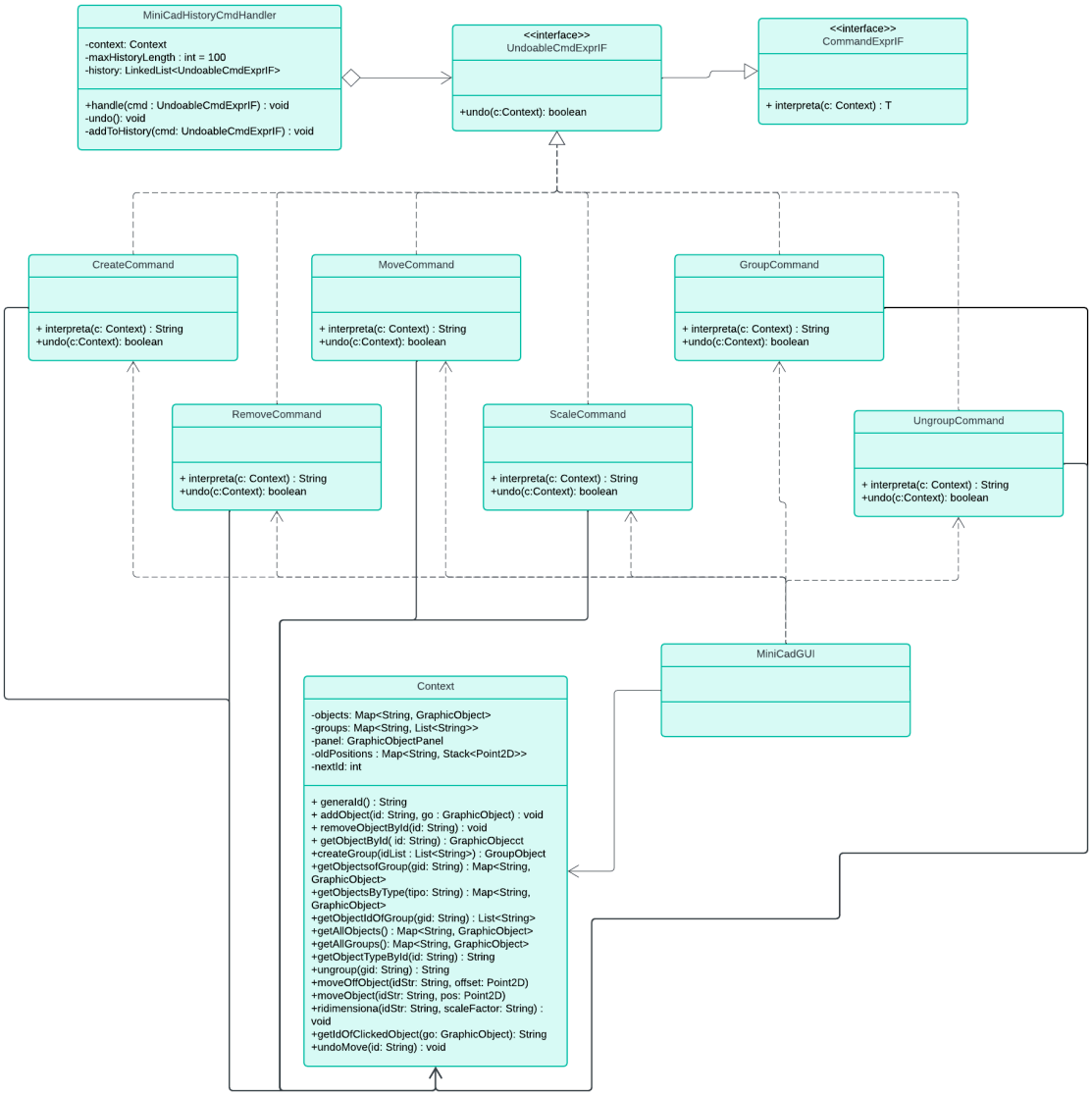
Il *Pattern Command* è un design pattern che separa l'invocazione di un'operazione dalla sua esecuzione, incapsulando ogni operazione come un oggetto
Nel sistema MiniCAD, ogni operazione annullabile è rappresentata come un comando che implementa l'interfaccia "*UndoableCmdExprIF*"). Questi comandi possono sia eseguire l'operazione (con il metodo interpreta()) sia annullarla (con il metodo "undo").

Quando l'utente esegue un'azione, viene creato un oggetto comando per quell'azione e viene passato al gestore di comandi "*MiniCADHistoryCmdHandler*".
Il gestore di comandi mantiene una lista di comandi eseguiti (*history*). Ogni volta che un comando viene eseguito, viene aggiunto a questa lista di cronologia.
Quando l'utente richiede un'operazione di *undo*, il gestore recupera l'ultimo comando dalla lista di cronologia e ne invoca il metodo *undo*.
Grazie al Command Pattern, l'invocazione delle operazioni è separata dalla loro effettiva esecuzione, semplificando quindi la gestione di operazioni complesse.

Partecipanti

- **Command (UndoableCmdExprIF) :** definisce l'interfaccia comune per tutti i comandi che devono implementare il metodo `undo(...)` per annullare l'operazione.
- **ConcreteCommand (CreateCommand, RemoveCommand, MoveCommand, ScaleCommand, GroupCommand, UngroupCommand,) :** implementano l'interfaccia "UndoableCmdExprIF" e implementano il metodo `undo(...)`.
- **Client (MiniCadGUI) :** crea i comandi e li passa all'*invoker* per l'esecuzione
- **Invoker (MiniCadHistoryCmdHandler) :** il gestore dei comandi che riceve i comandi concreti, li esegue e gestisce la cronologia per l'operazione di *undo*. Questa classe memorizza i comandi eseguiti nella lista *history* e gestisce le richieste di *undo*.
- **Receiver (Context) :** il contesto in cui vengono eseguiti i comandi. La classe *Context* è quella che rappresenta lo stato del sistema MiniCAD. Viene passato ai comandi e viene utilizzato per eseguire e annullare le operazioni.

Struttura



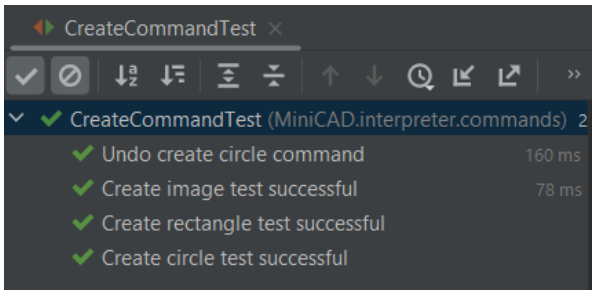
6. Adozione dello Unit Testing

L'integrazione dello unit testing nel processo di sviluppo del sistema MiniCAD mi ha aiutato per garantire la qualità del codice e la correttezza delle funzionalità implementate. Questo approccio prevede la scrittura di test automatici per verificare il comportamento delle singole unità di codice in modo isolato.

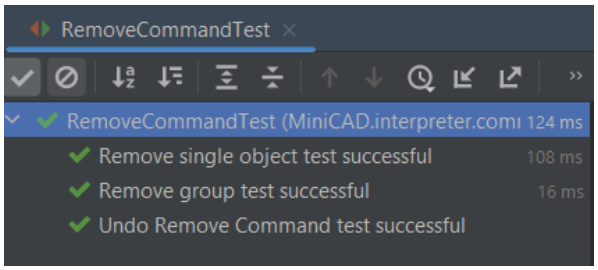
Per implementare lo unit testing, ho utilizzato il framework Junit, che supporta la creazione e l'esecuzione di test automatici in modo semplice ed efficiente.

Di seguito è riportato il risultato di alcuni test dei comandi:

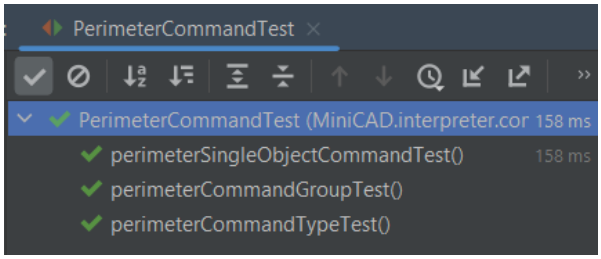
CreateCommand Test:



RemoveCommand Test:



PerimeterCommand Test:



F. Progettazione di Basso Livello

Panoramica dei Moduli/ Componenti

I principali moduli del sistema MiniCAD sono:

1. **miniinterpreter package -** contiene :
    a. **controller package**
        i. **MiniCADController** - è responsabile della gestione e dell'esecuzione dei comandi dell'utente. I comandi sono inviati tramite i vari pulsanti dell'interfaccia utente. Il controller si occupa di interpretare questi comandi e di applicarli sugli oggetti grafici. Dopo l'esecuzione di un comando, l'interfaccia utente viene aggiornata per riflettere le modifiche oppure mostrare risultati specifici.
        ii. **undoMngr package** - contiene classi per gestire la funzionalità *Undo* del sistema
            1. **MiniCadCommandHandler**- l'interfaccia che definisce un contratto per la gestione di comandi annullabili nel sistema.
            2. **MiniCadHistoryCmdHandler** - gestisce la cronologia dei comandi, permettendo di annullare le operazioni specifiche. Il metodo `handle()` è responsabile della gestione dei comandi ricevuti. Se il comando è annullabile, viene chiamato il metodo `addToHistory(cmd)` per memorizzare i comandi nella cronologia. Quando viene selezionato il pulsante UNDO, invoca il metodo `undo(...)` del comando specifico per annullare l'ultima azione eseguita.
    b. **model package** - un'estensione del pacchetto *model* dell'applicazione esterna integrata al sistema MiniCAD, che include oggetti grafici non disponibili nell'applicazione esterna.
        i. **GroupObject**- crea i gruppi di oggetti grafici e li gestisce come un'unica entità.
    c. **specificCmds package** - contiene le implementazioni dei principali classi del mini-interprete.
        i. **commandsExpr package** - pacchetto dei comandi all'interno del sistema
            1. **CommandExprIF** - interfaccia per la definizione di comandi. Questa interfaccia rappresenta un'espressione che può essere interpretata dal sistema.

2. **UndoableCmdExprIF** - esterne l'interfaccia *CommandExprIF* e aggiunge la capacità di annullare i comandi
3. **Command** - questa classe concretizza le azioni da eseguire e le operazioni da svolgere quando il comando viene eseguito.
4. **CreateCommand**
5. **RemoveCommand**
6. **MoveCommand**
7. **ScaleCommand**
8. **ListCommand**
9. **GroupCommand**
10. **UngroupCommand**
11. **AreaCommand**
12. **PerimeterCommand**

ii. **lexerparser package** - contiene le classi responsabili della tokenizzazione (*lexing*) e del *parsing* dei comandi.
1. **CommandLexer** - tokenizza l'input dei comandi. La classe analizza la stringa di input e identifica i token rilevanti e infine, fornisce un flusso di token al *parser* per l'elaborazione successiva.
2. **CommandParser** - analizza e interpreta i token generati dal "CommandLexer" e costruisce una rappresentazione strutturata del comando che possa essere eseguito dal sistema.

iii. **utilExpr package** - contiene i costrutti utili per la manipolazione dei comandi.
1. **TokenType** - definisce i diversi tipi di *token* che possono essere riconosciuti e utilizzati nel sistema
2. **Token**- rappresenta l'istanza concreta di un token
3. **TypeConstructorExpr** - utilizzati per interpretare o costruire oggetti grafici specifici nel sistema.
4. **PosizioneExpr** - costituisce l'espressione che gestisce la posizione di un oggetto grafico.
5. **ListIdExpr** - gestisce un elenco di identificatori che possono essere utilizzati nei vari comandi.

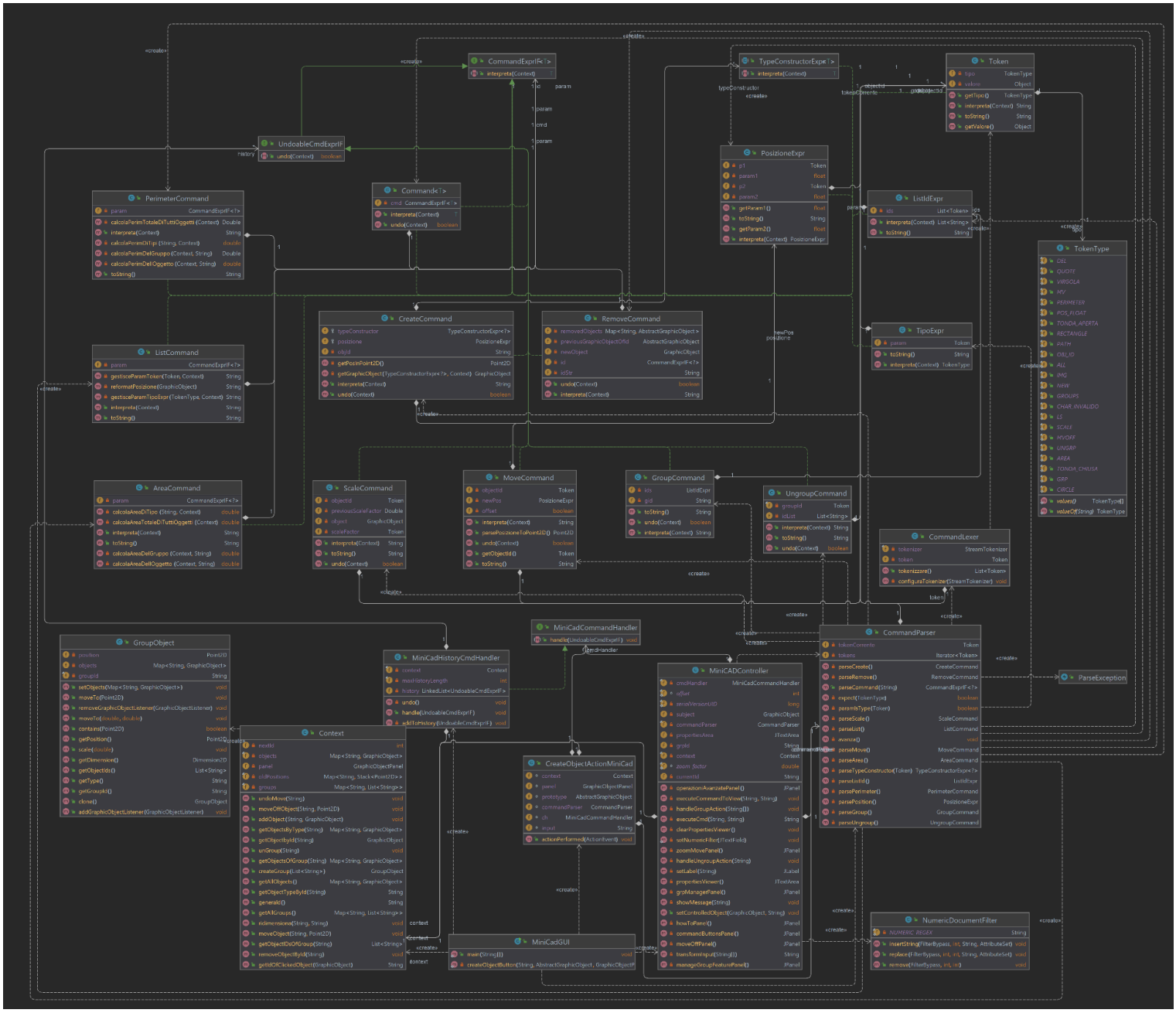
iv. **Context Class** - serve come contesto globale per il sistema. la classe gestisce la creazione, la manipolazione, il raggruppamento e la rimozione di oggetti grafici.

d. view package - estensione del pacchetto *view* dell'applicazione esterna. Il pacchetto contiene la classe che gestisce la creazione di nuovi oggetti grafici nel sistema tramite l'interazione con l'interfaccia utente

   i. **CreateObjectActionMiniCad** - progettata per interpretare un comando testuale fornito dall'utente, per creare un nuovo oggetto basato su questo comando e per visualizzarlo nel pannello grafico dell'applicazione MiniCAD.

2. **ui package**
   a. **MiniCadGUI** - responsabile della gestione dell'interfaccia grafica principale dell'applicazione MiniCAD. Utilizza il componente *MiniCadController* per gestire l'interazione diretta tra l'utente e gli oggetti grafici presenti nel pannello.
3. **utilpackage**
   a. **NumericDocumentFilter** -viene utilizzato all'interno della classe *MiniCadController* per assicurare che solo valori numerici siano accettati come input dei textbox specifici
   b. **Util**- contiene classi ausiliarie utilizzate da diversi componenti del sistema.

Struttura dei Componenti



minicad class diagram2.png

G. Spiegare come il progetto soddisfa i requisiti funzionali (FRs) e quelli non funzionali (NFRs)

<Report in this section how the architectural and low level design you produced satisfies the FRs and the NFRs>

Requisiti Funzionali (FRs)

ID	Requisito	Soluzioni
S1	**Creazione di Oggetti Grafici**	Il sistema MiniCAD deve consentire all'utente di creare vari tipi di oggetti grafici (cerchi, rettangoli, immagini, gruppi di oggetti grafici) e aggiungerli al contesto grafico corrente. L'implementazione della classe *CreateCommand* soddisfa questo requisito utilizzando le classi *TypeConstructorExpr* per rappresentare l'astrazione del tipo di oggetto grafico ,e *PosizioneExpr* per gestire la posizione dell'oggetto. Dopo la creazione, l'oggetto viene clonato e aggiunto all'oggetto Context, tramite il metodo `context.addObject(objectId, graphicObject)`. Infine, la classe implementa l'interfaccia *UndoableCmdExprIF* e descrive nel metodo `undo(...)` l'annullamento della creazione dell'oggetto creato dal contesto.
S2	**Rimozione di Oggetti**	L'implementazione della classe *RemoveCommand* soddisfa il requisito funzionale relativo alla rimozione di oggetti grafici attraverso i seguenti meccanismi: - L'ID dell'oggetto da rimuovere viene interpretato attraverso l'oggetto Token, che estrae l'ID dal contesto. - Siccome la classe permette di rimuovere sia oggetti grafici singoli che gruppi di oggetti, viene controllato il tipo di oggetto prima della rimozione. Se l'oggetto da

ID	Requisito	Soluzioni
		rimuovere è di tipo *Group*, tutti gli oggetti associati al gruppo vengono anche rimossi. - Per poter effettuare l'annullamento del comando, viene utilizzata una mappa *removedObjects*. Questa struttura dati contiene l'ID e una coppia dell'oggetto. Questo consente di tracciare quali oggetti sono stati rimossi e eventualmente recuperarli senza alterare l'originale. - Se l'oggetto rimosso era un gruppo di oggetti, il comando di *Undo* ripristina prima gli oggetti e li raggruppa di nuovo. Tuttavia, il gruppo ricreato, non conserva l'ID originale. Invece, viene assegnato un nuovo ID poiché la generazione dell'ID del gruppo avviene nel metodo `createGroup(...)` gestito dal *Context* e non nel *RemoveCommand*.
S3	**Spostamenti di Oggetti**	La classe *MoveCommand* gestisce lo spostamento di un oggetto grafico in due modalità: spostando in una nuova posizione assoluta o utilizzando un offset da sommare alla posizione corrente. Invece di implementare due classi separate per queste operazioni, ho utilizzato l'attributo booleano *offset*. Quando *offset* è impostato su *true*, il comando applica un *offset* alle coordinate correnti dell'oggetto, spostandolo di un valore specificato. Questo è gestito chiamando il metodo `context.moveOffObject(...)`. Se invece *offset* è *false*, l'oggetto viene spostato in una posizione assoluta, come specificato nel parametro del comando. Questo è gestito chiamando il metodo `context.moveObject(...)`. Il comando identifica l'oggetto da spostare utilizzando un Token che rappresenta l'ID dell'oggetto. Questo ID viene interpretato nel contesto per ottenere l'istanza dell'oggetto grafico da spostare. Le nuove coordinate di destinazione sono gestite dalla classe *PosizioneExpr*, che viene poi convertita in un oggetto *Point2D*.

ID	Requisito	Soluzioni
		Per implementare la funzionalità di undo, il metodo `context.undoMove(id)` ripristina la posizione originale dell'oggetto.
S4	**Ridimensionamento di Oggetti**	La classe *ScaleCommand* gestisce il ridimensionamento degli oggetti grafici attraverso l'uso di un ID e un fattore di scala. Il comando utilizza i Token *scaleFactor* per interpretare il fattore di scala nel contesto e *objectId* per l'ID dell'oggetto. Il comando chiama il metodo `context.ridimensiona(idStr, sfStr)` che riceve l'id dell'oggetto e il fattore di scala. Per gestire l'*undo*, il comando ripristina la dimensione originale scalando nuovamente l'oggetto con l'inverso del fattore di scala precedente. Questo è realizzato tramite il metodo `object.scale(1.0/previousScaleFactor)` dove *object* rappresenta l'oggetto grafico che chiama il suo metodo scale.
S5	**Visualizzazione delle Proprietà**	L'implementazione della classe *ListCommand* soddisfa il requisito funzionale relativo all'elenco e alla visualizzazione delle proprietà. degli oggetti grafici e dei gruppi di oggetti. La classe *ListCommand* accetta un parametro di tipo *CommandEprIF<?>* , il che consente di gestire vari tipi di parametri attraverso il costruttore. Questo permette di specificare il tipo di elenco desiderato: ● oggetti di un tipo specifico (TipoExpr) ● oggetto con un ID specifico (Token di tipo OBJ_ID) ● tutti gli oggetti (Token ALL) ● tutti i gruppi (Token GROUPS) Per migliorare la leggibilità e mantenere il codice ordinato, sono stati creati due metodo privati: ● `gestisceParamTipoExpr(...)`: gestisce l'elenco quando il parametro è un espressione di tipo TipoExpr ● `gestisceParamToken(...)`: gestisce l'elenco quando il parametro è un Token

ID	Requisito	Soluzioni
S6	**Gestione dei Gruppi di Oggetti**	Il sistema soddisfa il requisito relativo al raggruppamento degli oggetti grafici attraverso l'implementazione delle classi *GroupCommand* per il raggruppamento e lo scioglimento del gruppo di oggetti tramite la classe *UngroupCommand.* GroupCommand La classe accetta un parametro di tipo *ListIdExpr*, che rappresenta una lista di ID degli oggetti da raggruppare. Il metodo `interpreta(...)` della classe utilizza questa lista per creare un nuovo gruppo invocando `context.createGroup(idList).` Questo ultimo restituisce un oggetto *GroupObject* identificato da un ID. La classe implementa il metodo `undo(...)` che utilizza l'ID del gruppo per sciogliere il raggruppamento chiamando il metodo `context.unGroup(gid)`, e ripristina gli oggetti al loro stato individuale. UngroupCommand La classe gestisce lo scioglimento di un gruppo controllando prima se l'ID fornito corrisponde a un gruppo. All'interno della classe *context*, utilizzo la struttura dati *groups* per tenere traccia di tutti i gruppi creati e gli oggetti singoli che ne fanno parte. Quando viene chiamato il servizio, il sistema cancella l'ID da *groups* rimuovendo l'associazione degli oggetti come gruppo senza cancellare gli oggetti stessi.
S7	**Calcolo delle Area e dei Perimetri**	Il requisito prevede la possibilità di calcolare l'area e il perimetro totale in base ai seguenti tipi di parametri: ● ID dell'oggetto o gruppo di oggetti Quando il parametro è un ID, il metodo `interpreta(...)` utilizza `calcolaAreaDellOggetto(...)` per calcolare l'area e `calcolaPerimDellOggetto(...)` per calcolare il perimetro dell'oggetto specifico.

ID	Requisito	Soluzioni
		• tipo di oggetti Se il parametro *param* è un'istanza di *TipoExpr*. il metodo esegue il calcolo dell'area o del perimetro per tutti gli oggetti del tipo specificato. La struttura del codice include un blocco *switch* per gestire i vari tipi di oggetti. Nuovi tipi di oggetti possono essere aggiunti facilmente aggiornando il blocco *switch* e il metodo `calcolaAreaDiTipo(...)` e `calcolaPerimDiTipo(...).` • "all" Se il parametro è un Token di tipo ALL, il metodo `interpreta(...)` utilizza `calcolaAreaTotaleDiTuttiOggetti()` o `calcolaPerimTotaleDiTuttiOggetti()` per calcolare l'area o il perimetro totale di tutti gli oggetti nel pannello grafico.

Requisiti Non Funzionali (NFRs)

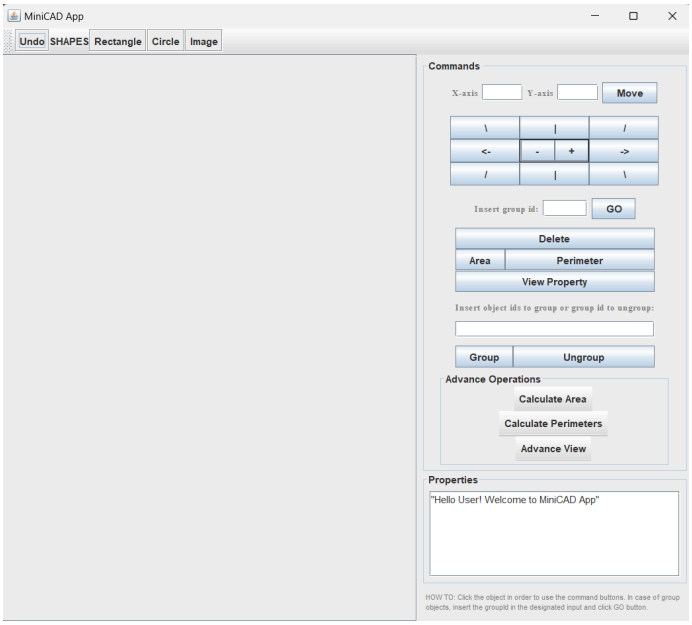
Requisito	Soluzioni
Affidabilità	• Implementazione della funzionalità *undo* utilizzando il pattern *Command* che garantisce una struttura del codice robusta e modulare.
Usabilità	• La GUI è progettata con Swing per essere intuitiva, con controlli e strumenti facilmente accessibili.
Manutenibilità	• L'uso di design patterns come l'*Interpreter Pattern* per l'implementazione dei comandi, il *Command Pattern* per la gestione della funzionalità Undo e il *Composite Pattern* per la creazione di gruppi di oggetti favorisce un codice modulare e facilmente mantenibile.

Requisito	Soluzioni
Testabilità	• Integrazione di test automatici con JUnit.

Appendix. Prototype

<Provide a brief report on your prototype, and especially: information on what you have implemented, how the implementation covers the FR and NFR, how the prototypes demonstrates your project correctness with respect to the FR and NFR. You may add some screenshots to describe what required above. Be ready to show your prototype during the oral examination>

Interfaccia Utente



Copertura dei Requisiti Funzionali (FRs) :

S1: Creazione di Oggetti Grafici

- Nel Toolbar dell'applicazione sono presenti i pulsanti per disegnare un oggetto nel pannello grafico.
- E' presente inoltre il pulsante di *Undo* per annullare le operazioni eseguite.
- Nel *sidebar* dell'applicazione sono elencati i pulsanti per le operazioni che si possono fare all'oggetto.
- A differenza di sistemi CAD avanzati, inizialmente la dimensione e la posizione dell'oggetto saranno già impostate e predefinite.

- Il sistema crea l'oggetto.

- Il sistema utilizza le proprietà predefinite per scrivere la stringa di command input per il pattern Interpreter.

```
AbstractGraphicObject go = new RectangleObject(new Point( x: 10,  y: 10),  w: 20,  h: 50);
```

```
case "Rectangle" -> cmdInput = "create rectangle ("+ go.getDimension().getHeight()+","+go.getDimension().getWidth()+") ("+
        go.getPosition().getX()+","+go.getPosition().getY()+")";
```

- Per poter manipolare l'oggetto tramite i pulsanti nel pannello laterale dell'applicazione, è necessario prima cliccare sull'oggetto nel pannello grafico. Una volta selezionato, il suo ID viene visualizzato nell'area di testo dell'applicazione.

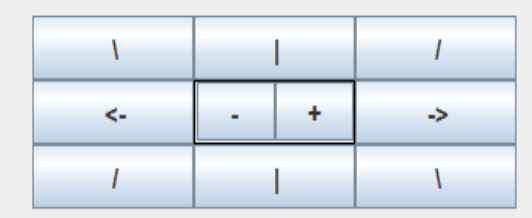
S2: Rimozione di Oggetti

- Il prototipo consente la rimozione dell'oggetto dal pannello grafico tramite il pulsante *Delete*.
- È importante selezionare prima l'oggetto da rimuovere prima di selezionare il pulsante di Delete, in modo che il sistema possa memorizzare l'ID dell'oggetto e utilizzarlo nella stringa di comando di input.

```
String delInput = "del "+ currentId;
```

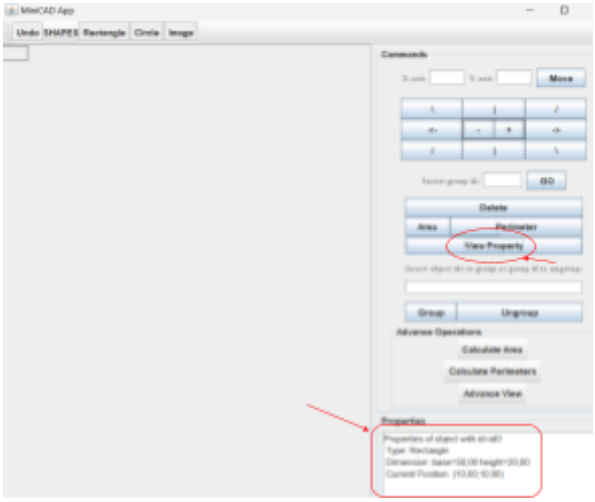
S3 , S4: Spostamento e Ridimensionamento di Oggetti

- Lo spostamento dell'oggetto selezionato avviene tramite i pulsanti che spostano l'oggetto nella direzione indicata dalle frecce.
- Il ridimensionamento avviene tramite il pulsante "+" per ingrandire e il pulsante "-" per ridurre le dimensioni.

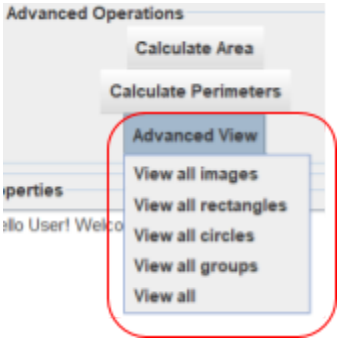


S5: Visualizzazione delle proprietà

- Si possono visualizzare le proprietà dell'oggetto selezionato tramite il pulsante *View Property.*



- Il pulsante *Advanced View* permette altri tipi di visualizzazione:
  - "*View all images*" - per visualizzare l'elenco di tutte le immagini e la loro posizione nel pannello grafico.
  - *View all rectangles* - per visualizzare l'elenco di tutti i rettangoli e la loro posizione nel pannello grafico.
  - *View all circles* - per visualizzare l'elenco di tutti i cerchi e la loro posizione nel pannello grafico.
  - *View all groups* - per visualizzare l'elenco di tutti i gruppi presenti nel pannello e l'ID degli oggetti che ne fanno parte.
  - *View all* - per visualizzare tutti gli oggetti presenti nel pannello grafico.



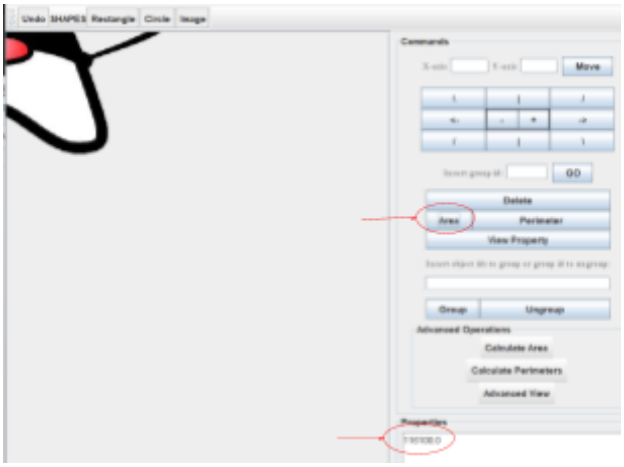
S6: Gestione dei Gruppi di Oggetti

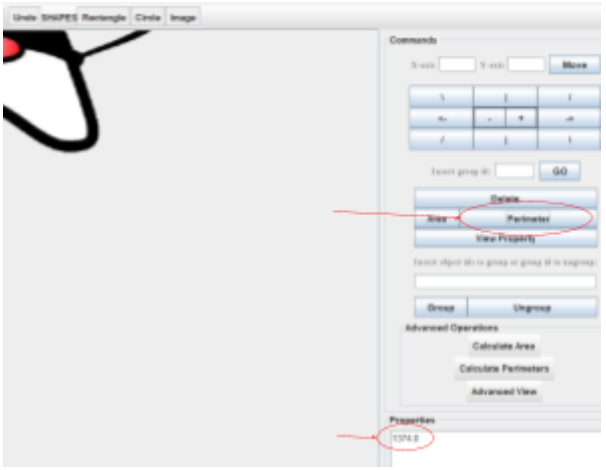
- Per poter raggruppare gli oggetto, occorre inserire nel *Textarea* gli ID degli oggetti e cliccare il pulsante *Group*.
- Per sciogliere il raggruppamento, occorre inserire l'ID del gruppo e selezionare il pulsante *Ungroup*.

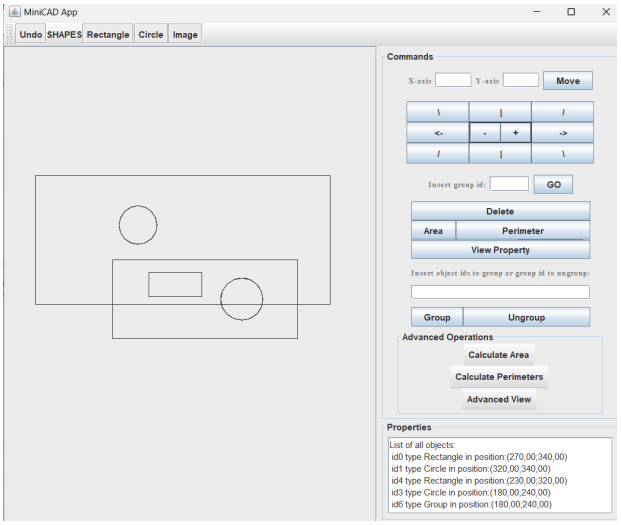
- Per manipolare il gruppo di oggetti creato, è necessario inserire l'ID dell'oggetto nella casella di testo e selezionare il pulsante *Go*. In questo modo, sarà possibile spostare, ridimensionare e rimuovere tutti i membri del gruppo.

S7: Calcolo delle Aree e dei Perimetri

- Il calcolo dell'area e del perimetro avviene tramite i pulsanti *Area* e *Perimeter*, rispettivamente.







Copertura dei Requisiti Non Funzionali (NFRs) :

- **Affidabilità** - Il sistema include il pulsante *Undo* nel *Toolbar* dell'applicazione che consente di annullare le modifiche, migliorando l'affidabilità dell'interazione.

- **Usabilità** - L'interfaccia grafica dell'applicazione è progettata con un layout semplice. La barra laterale offre pulsanti chiari per le azioni principali.

<u>*SOURCES:*</u>

Sarcar, M.M.M, et al. *Computer Aided Design and Manufacturing*. PHI Learning, 2008. Accessed 22 August 2024.

Ball, Alex. Preserving Computer-Aided Design (CAD). Great Britain, Digital Preservation Coalition, 2013. Accessed 23 August 2024.