“Ingegneria del Software”
2023-2024

Docente: Prof. Angelo Furfaro

MINI-CAD

Data	08/05/2024
Documento	Documento Finale – D3

Team Members		
Nome e Cognome	**Matricola**	**E-mail address**
LORENZO CICCONE	**219916**	**CCCLNZ01S29C710W@STUDENTI.UNICAL.IT**

Sommario

Non è stata trovata alcuna voce d'indice.

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

Il programma consente di interpretare righe di testo scritte all'interno di una textbox posta in fondo all'unica pagina che si presenta dopo l'avvio dello stesso. La lista dei comandi utilizzabili con le sintassi corrette è presente in una finestra pop up che compare una volta cliccato il pulsante 'help'. In base al comando inserito, avvengono le azioni svolte nel pannello bianco sopra. Ci sono tutti i comandi specificati dalla richiesta, quindi di creazione, raggruppamento, spostamento e modifica delle figure. È possibile azionare nuovamente il comando inserito in precedenza tramite il bottone 'redo', sempre in alto a sinistra, insieme al pulsante 'undo' che permette di tornare indietro, qualunque sia stato il comando o i comandi utilizzati, in quanto è in grado di salvare una cronologia di istruzioni per poi ripeterle. Permette tutti i comandi citati nel comando 'help' con 3 tipi di oggetti: circle, rectangle e img (qualsiasi immagine tramite percorso).

B. Raffinamento dei Requisiti

Realizzare un mini interprete di comandi da integrare in una applicazione per la manipolazione di oggetti grafici 2d. Esistono solo 3 tipi di oggetti grafici: rettangoli, cerchi ed immagini.

L'interprete, la cui grammatica EBNF è riportata di seguito, supporta i seguenti comandi, alcuni dei quali sono reversibili (se ne può richiedere l'*undo*).

- Creazione di un nuovo oggetto grafico
- Rimozione di un oggetto o un gruppo di oggetti
- Spostamento di un oggetto o un gruppo di oggetti
- Ridimensionamento di un oggetto o un gruppo di oggetti
- Visualizzazione delle proprietà di un oggetto, un gruppo di oggetti o di un tipo di oggetti
- Creazione di un gruppo di oggetti
- Rimozione di un gruppo di oggetti
- Calcolo area (perimetro)

B.1 Servizi (con prioritizzazione)

- 1) **Creazione di un nuovo oggetto grafico** (importanza: Alta, Complessità: Media): permette la creazione degli oggetti con le specifiche inserite;
- 2) **Rimozione di un oggetto o gruppo di oggetti** (importanza: Alta, Complessità: Facile): permette la rimozione degli oggetti specificati;
- 3) **Creazione di un gruppo di oggetti** (importanza: Alta, Complessità: difficile): permette la creazione di un gruppo astratto che contiene al suo interno gli oggetti che abbiamo selezionato;
- 4) **Spostamento di un oggetto o gruppo di oggetti** (importanza: Alta, Complessità: media): permette lo spostamento in determinate coordinate inserite di oggetti singoli o gruppi di oggetti;
- 5) **Ridimensionamento oggetti o gruppi di oggetti** (importanza: Media, Complessità: bassa): permette di modificare i parametri che riguardano le dimensioni di una figura o di un gruppo di figure.
- 6) **Rimozione gruppo di oggetti** (importanza: Media, Complessità: media): permette di eliminare uno dei gruppi creati in precedenza.
- 7) **Visualizzazione delle proprietà di un oggetto, un gruppo di oggetti o di un tipo di oggetti** (importanza: Bassa, Complessità: media): permette di visualizzare tramite finestra popup le informazioni di oggetti, intere categorie di oggetti o gruppi esistenti
- 8) **Calcolo area (perimetro)** (importanza: Bassa, Complessità: bassa): permette di visualizzare sul terminale l'area della figura scelta, la somma delle aree del tipo selezionato oppure la somma delle aree di tutto. Funziona anche col perimetro.

B .2 Requisiti non Funzionali

1. Usabilita':
   a. Interfaccia Utente: Il design dell'interfaccia è minimalistico, creato così intenzionalmente, per agevolare l'utente finale, pratica e diretta;
   b. Documentazione: nell'output di sistema, ci sono vari messaggi che aiutano a capire cosa si sta facendo e cosa ha dato errore;

2. Performance:
    a. **Tempo di Risposta**: Le operazioni come l'aggiunta, la rimozione e il movimento degli oggetti sono implementate in modo efficiente, suggerendo tempi di risposta brevi. La struttura a mappa per la gestione degli oggetti offre un accesso rapido agli stessi.
    b. **Scalabilità**: L'uso di HashMap per la gestione degli oggetti consente al sistema di scalare efficacemente con un numero elevato di oggetti.

3. Affidabilità:
    a. **Disponibilità**: Non sono stati segnalati crash o problemi di stabilità durante lo sviluppo. L'implementazione robusta delle operazioni suggerisce una buona disponibilità del sistema.
    b. **Gestione degli Errori**: Il codice gestisce situazioni di errore comuni, come il tentativo di rimuovere un oggetto non esistente, con messaggi di errore chiari.

4. Manutenibilità:
    a. **Modularità**: Il codice è strutturato in classi e metodi ben definiti, facilitando la manutenzione e l'espansione futura del sistema.
    b. **Testabilità**: La separazione delle funzionalità in metodi specifici rende il codice facilmente testabile. Ogni operazione può essere testata individualmente.

5. Estensibilità:
    a. **Aggiunta di Nuovi Oggetti**: La struttura del codice, con classi base e gestione degli ID, rende relativamente semplice l'aggiunta di nuovi tipi di oggetti grafici.
    b. **Supporto per Nuovi Comandi**: L'aggiunta di nuovi comandi può essere implementata estendendo le funzionalità esistenti.

B.3 Scenari d'uso dettagliati

Caso D'uso	L'utente vuole creare un nuovo oggetto.
Precondizione	L'applicazione è aperta e l'utente è nella vista principale.
Svolgimento Normale	L'utente scrive nello spazio di testo il comando per la creazione della figura, con i dati della figura richiesti e la posizione nel pannello (x,y). Es: create circle (12.0) (40,50)
Postcondizione	La figura viene creata e mostrata sul pannello bianco.
Descrizione	L'oggetto viene creato, gli viene assegnato un id univoco, inserito in una apposita mappa chiamata objects e poi inviato al GraphicObjectPanel per la visualizzazione.

Caso D'uso	L'utente vuole eliminare un oggetto o un gruppo di oggetti.
Precondizione	L'applicazione è aperta e l'utente è nella vista principale.
Svolgimento Normale	L'utente scrive nello spazio di testo il comando per la eliminazione della figura o gruppo, es: del 1 (1 = id scelto)
Postcondizione	La figura o il gruppo di figure vengono eliminati dal pannello e dalla mappa.
Descrizione	L'oggetto viene cercato tramite id nella mappa, rimosso ed eliminato anche graficamente, dando poi una conferma nel terminale.

Caso D'uso	Creazione Gruppo di oggetti grafici
Precondizione	L'applicazione è aperta e l'utente è nella vista principale. Esistono almeno 2 oggetti.
Svolgimento Normale	L'utente scrive nello spazio di testo il comando per la creazione del gruppo, seguito dagli id degli oggetti da raggruppare
Postcondizione	Il gruppo è stato registrato con successo, ma non è visualizzabile in quanto è visivamente astratto.
Descrizione	Se gli id specificati esistono il sistema crea un nuovo oggetto di tipo GroupObject con id univoco e ci aggiunge gli oggetti con l'id specificato in precedenza nel comando, dopodichè si aggiunge il gruppo alla mappa di oggetti e da conferma nel terminale in caso di successo.

Caso D'uso	Muovere un oggetto grafico
Precondizione	L'applicazione è aperta e l'utente è nella vista principale. Esistono almeno 1 oggetto grafico.
Svolgimento Normale	L'utente scrive nello spazio di testo il comando per lo spostamento degli oggetti grafici, sul pannello gli oggetti si muoveranno nelle posizioni espresse coi comandi.
Postcondizione	Il gruppo di figure o la figura sono in una posizione diversa in quanto è stata aggiornata dal comando.
Descrizione	Se l'id specificato è riconosciuto come id di una figura sola, allora il sistema recupera la posizione dell'oggetto e la aggiorna, stessa cosa avviene se è un gruppo, apre il gruppo, apre gli oggetti singoli e ne aggiorna la posizione in base al comando di movimento selezionato

Caso D'uso	Calcolare L'area o perimetro di un oggetto o di un gruppo di oggetti
Precondizione	L'applicazione è aperta e l'utente è nella vista principale. Esistono almeno 1 oggetto grafico.
Svolgimento Normale	L'utente scrive nello spazio di testo il comando per il calcolo dell'area o del perimetro insieme all'id, avra' il risulatato nel terminale.
Postcondizione	L'area é stata calcolata e mostrata all'utente.
Descrizione	Se l'id specificato è riconosciuto come id di una figura sola, allora il sistema recupera i dati necessari in base al tipo di figura per calcolarne l'area o il perimetro, dopo le manda in output, stessa cosa avviene se è un gruppo, apre il gruppo, apre gli oggetti singoli e ne ricava l'area o il perimetro, sommandoli e poi mandandoli in output.

Caso D'uso	Visualizzazione propriteta' oggetti, gruppi di oggetti o tipo di oggetti.
Precondizione	L'applicazione è aperta e l'utente è nella vista principale. Esistono almeno 1 oggetto grafico.
Svolgimento Normale	L'utente scrive nello spazio di testo il comando per la visualizzazione delle proprieta' (ls) ed in seguito l'id del singolo o del gruppo oppure il tipo.
Postcondizione	Le informazioni richieste vengono esibite in una finestra popup.

Descrizione	Funziona come per l'area ma prende tutti i dati, previo controllo del tipo di dato inserito nel comando, in cui si verifica se è un tipo, gruppo oppure figura singola (funziona anche con ls all in cui mostra tutto)

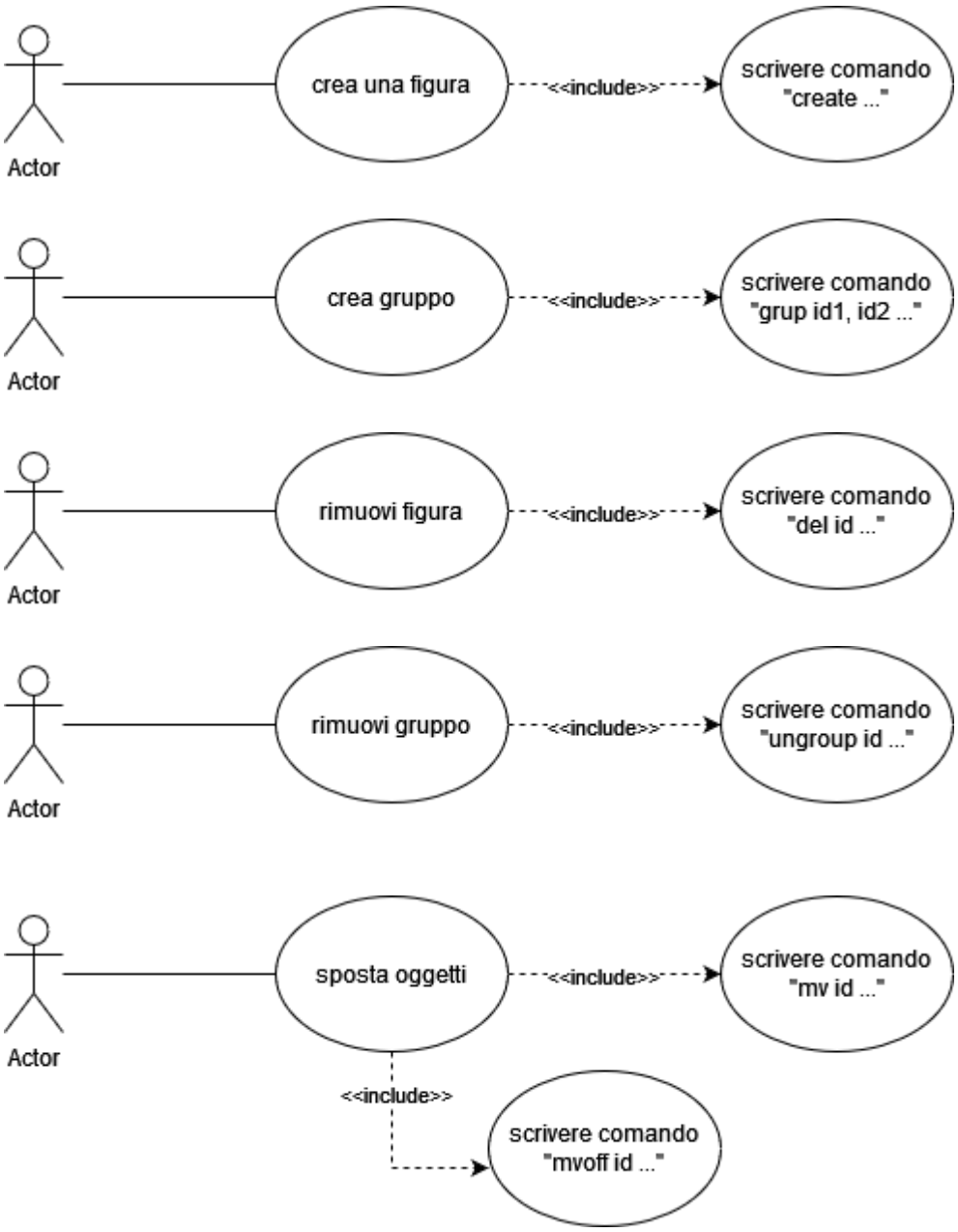
Caso D'uso	Visualizzare la lista di tutti i comandi.
Precondizione	L'applicazione è aperta e l'utente è nella vista principale.
Svolgimento Normale	L'utente preme sul bottone 'help' per ricevere le informazioni che necessita
Postcondizione	Si apre una finestra con tutti i comandi con esempi e spiegazioni.
Descrizione	

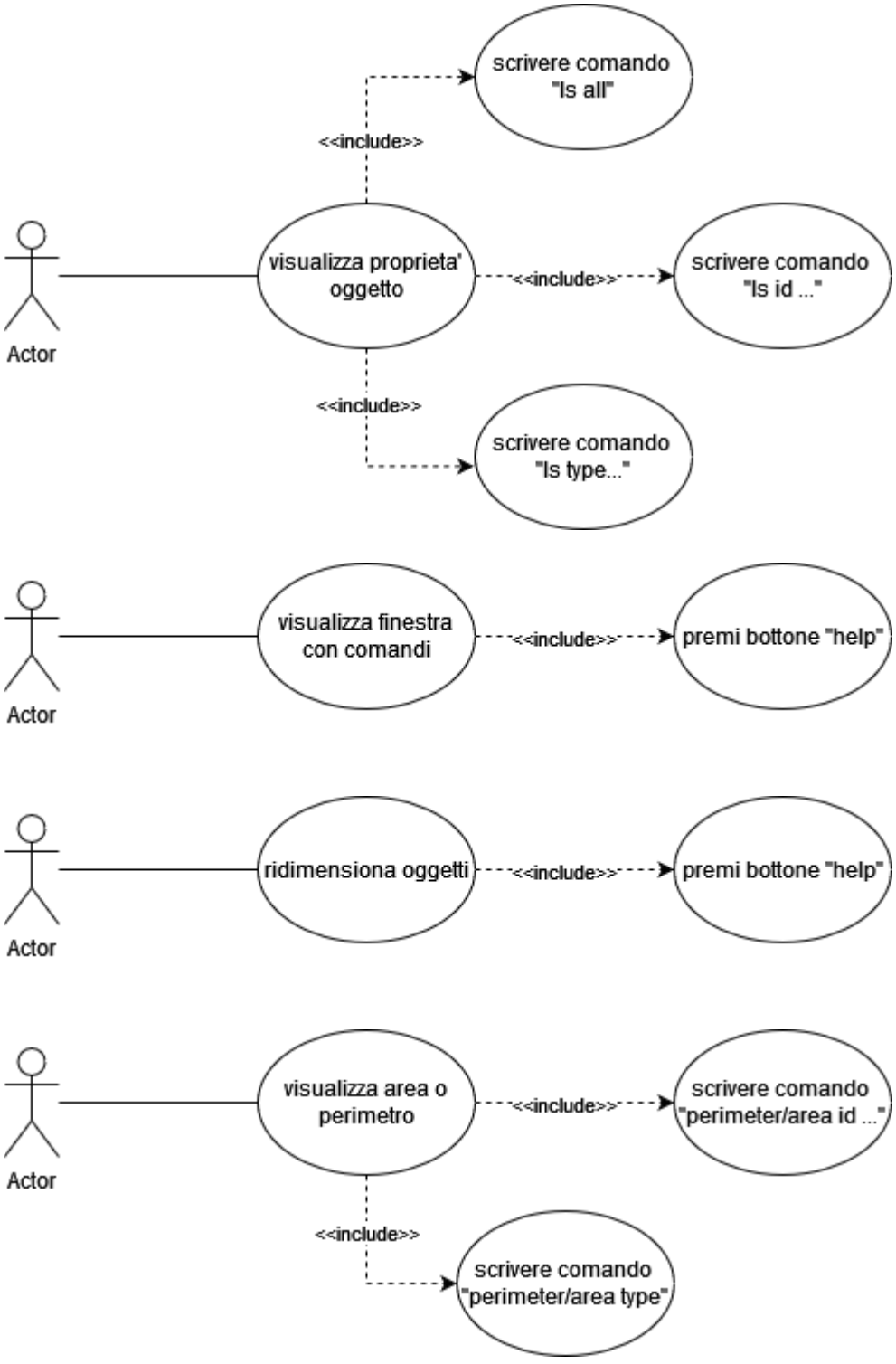
A.5 Assunzioni

La traccia non esprime un numero massimo o minimo di figure che si possono creare, ho quindi deciso di impostare gli id delle figure come numeri sequenziali da 1 a 999 per le figure e da 1000 in poi per i gruppi, per una questione di comodita' ed ulteriore sicurezza durante i controlli degli id dei comandi. Esiste quindi un tetto massimo figurativo di 999 figure.

Il pannello non ha dimensioni precise, ho quindi fatto in modo che si potesse allargare tramite mouse ingrandendo la finestra, in modo da poter mettere le figure dove vogliamo con le coordinate.

A.6 Use Case Diagrams

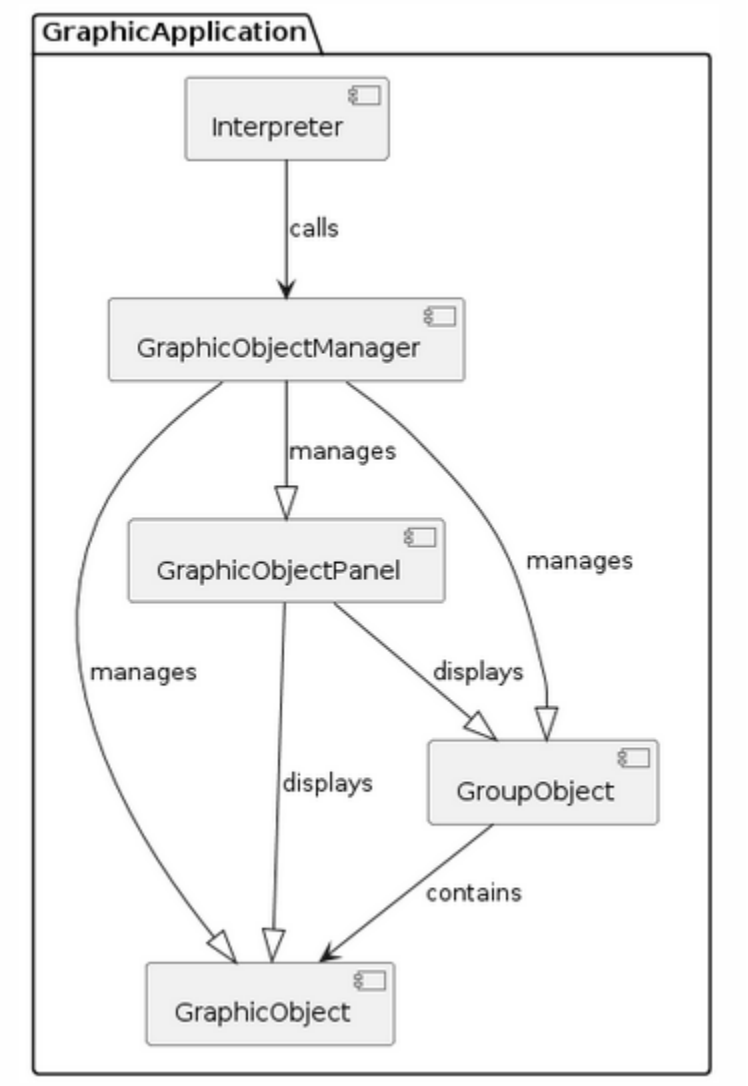



scrivere comando "ls all"
<<include>>
visualizza proprieta' oggetto
<<include>>
scrivere comando "ls id ..."
Actor
<<include>>
scrivere comando "ls type..."
visualizza finestra con comandi
<<include>>
premi bottone "help"
Actor
ridimensiona oggetti
<<include>>
premi bottone "help"
Actor
visualizza area o perimetro
<<include>>
scrivere comando "perimeter/area id ..."
Actor
<<include>>
scrivere comando "perimeter/area type"

C. Architettura Software

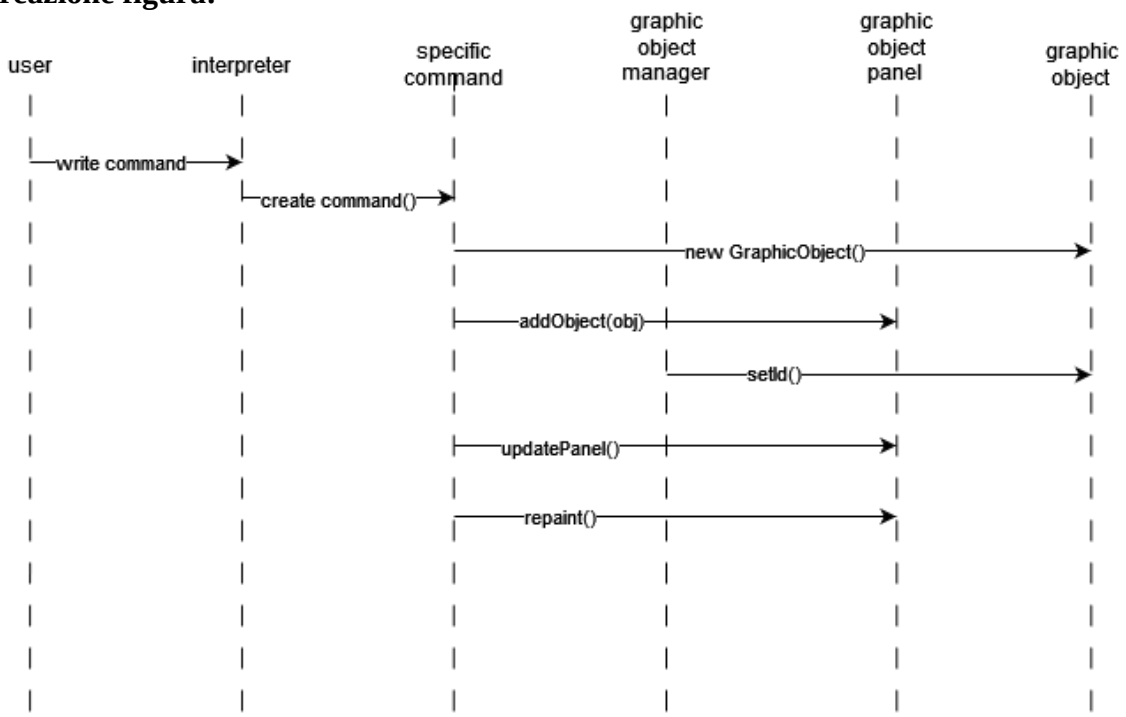
<IF RELEVANT, Report here both the static and the dynamic view of your system design, in terms of a Component Diagram, and their related Sequence Diagrams >

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

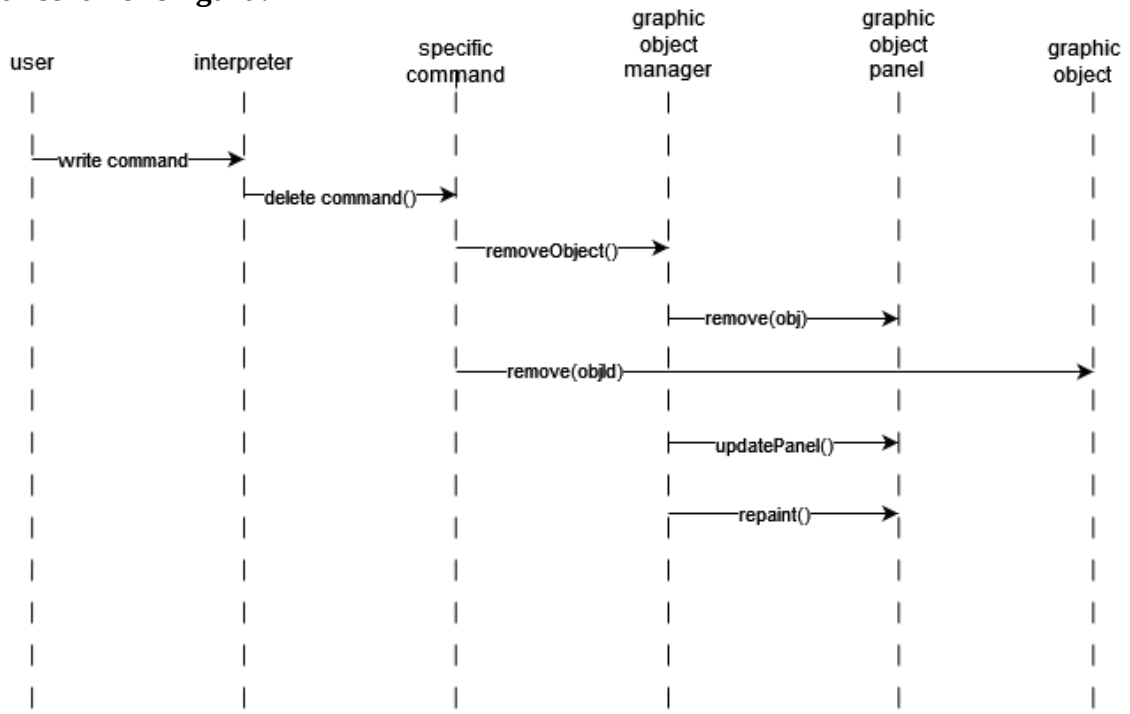


C.2 The dynamic view of the software architecture: Sequence Diagram

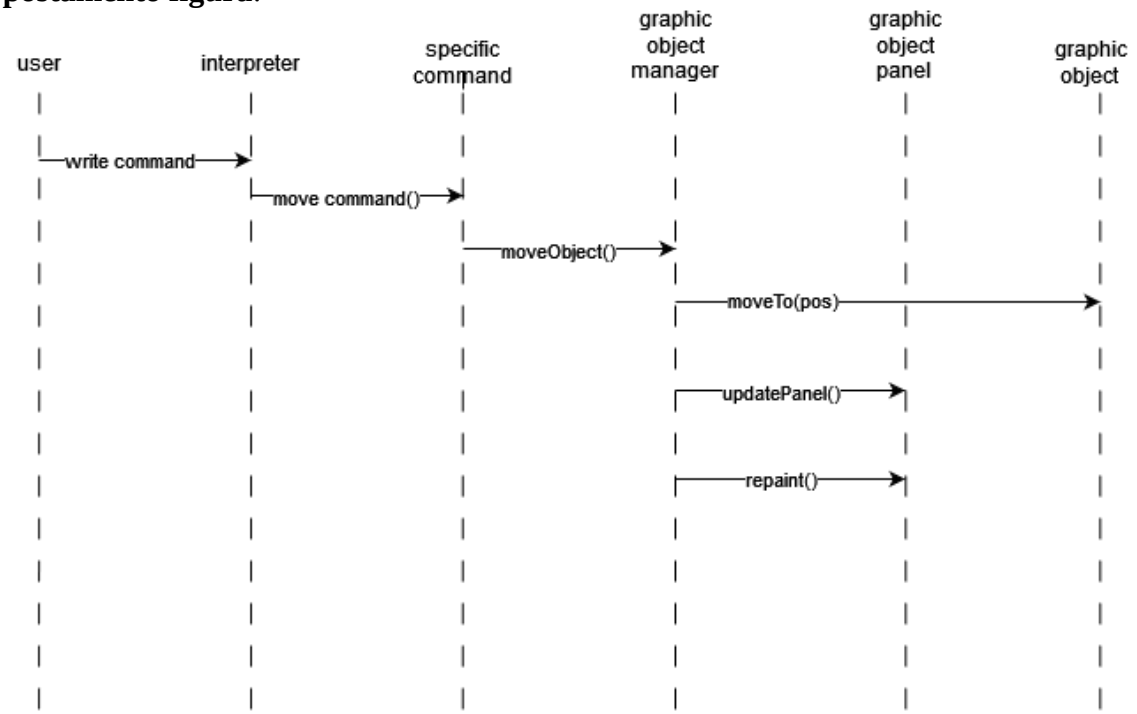
Creazione figura:



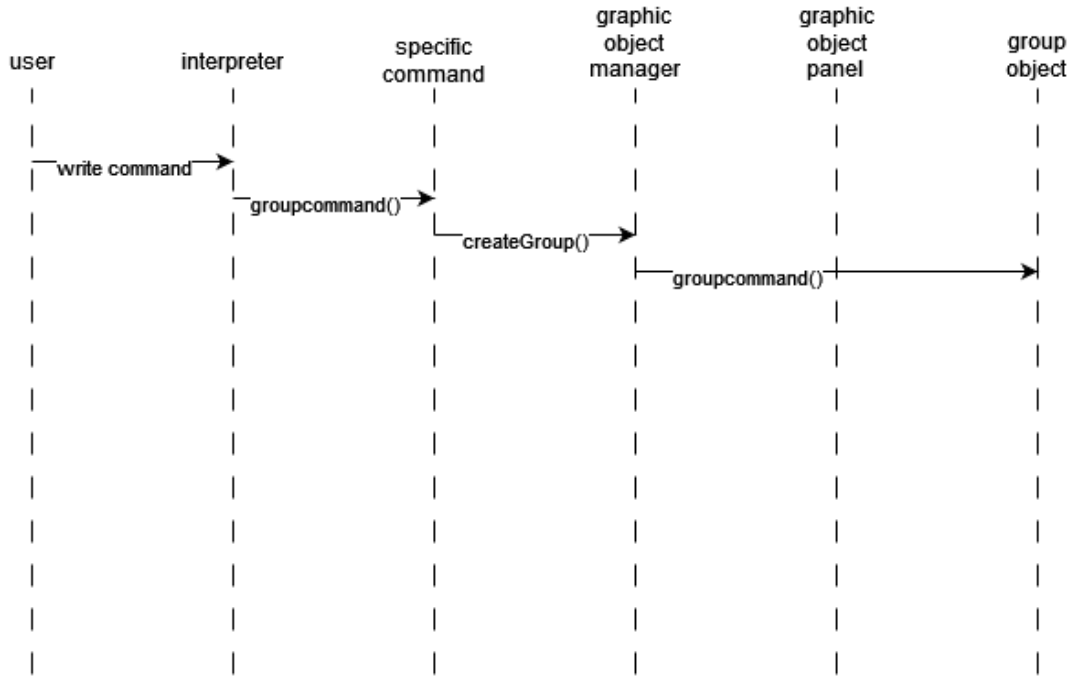
Cancellazione figura:



Spostamento figura:



Creazione Gruppo:



E. Scelte Progettuali (Design Decisions)

COMMAND PATTERN

Descrizione: Incapsula una richiesta come un oggetto, permettendo di parametrizzare i client con richieste diverse, fare logging e implementare undo/redo.

Implementazione:

- **Interfaccia Command**: Command.
- **Concrete Command**: CreateCircleCommand, CreateRectangleObject, CreateImgCommand, MoveCommand, MoveOffsetCommand, ScaleCommand, GroupCommand, UngroupCommand, AreaCommand, PerimeterCommand.
- **Invoker**: HistoryCommandHandler gestisce la cronologia dei comandi

FACTORY METHOD PATTERN

Descrizione: Definisce un'interfaccia per creare oggetti, ma lascia che siano le sottoclassi a determinare quale classe istanziare.

Implementazione:

- **Factory**: GraphicObjectViewFactory fornisce le viste per ogni tipo di oggetto grafico.
- **Concrete Product**:
  - CircleObjectView
  - RectangleObjectView
  - ImageObjectView

COMPOSITE PATTERN

Descrizione: Permette di comporre oggetti in strutture ad albero e trattare gli oggetti singoli e composti in modo uniforme.

Implementazione:

- **Component**: GraphicObject.
- **Composite**: GroupObject, che contiene una lista di GraphicObject.
- **Leaf**:
  - CircleObject
  - RectangleObject
  - ImageObject

OBSERVER PATTERN

Descrizione: Definisce una dipendenza uno-a-molti tra oggetti, in modo tale che quando un oggetto cambia stato, tutti i suoi osservatori vengono notificati.

Implementazione:

- **Subject**: GraphicObject, che notifica i cambiamenti.
- **Observer**: GraphicObjectListener, implementato da GraphicObjectPanel

INTERPRETER PATTERN

Descrizione: Definisce una rappresentazione per la grammatica della traccia e definisce un interprete in grado di interpretare i comandi della textbox.

Implementazione:

- **Analizzatore lessicale**: AnalizzatoreLessicale, che trasforma una sequenza di caratteri in sequenze di token.
- **Parser**: Parser, organizza i token e richiama i metodi del pattern Command.

F. Progettazione di Basso Livello

COMMAND PATTERN

CommandHandler.java

```java
package command;

public interface CommandHandler {
    void handle(Command cmd);
}
```

Command.java

```java
package command;

public interface Command {

    boolean doIt();

    boolean undoIt();
}
```

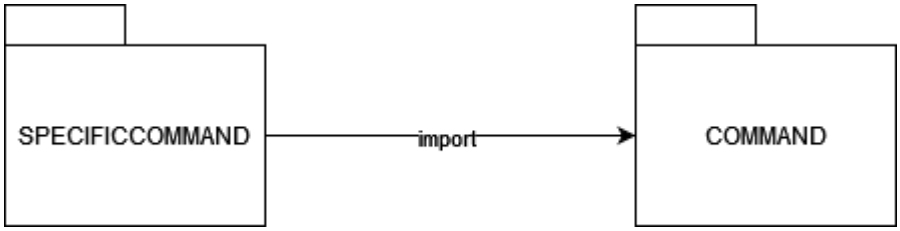
HistoryCommandHandler.java

```java
package command;

import java.util.LinkedList;
import view.GraphicObjectPanel;

public class HistoryCommandHandler implements CommandHandler {
    private int maxHistoryLength = 100;

    private final LinkedList<Command> history = new LinkedList<>();

    private final LinkedList<Command> redoList = new LinkedList<>();

    GraphicObjectPanel gpanel;    "gpanel": Unknown word.

    public HistoryCommandHandler(GraphicObjectPanel panel) {
        this(maxHistoryLength:100);
        this.gpanel = panel;    "gpanel": Unknown word.
    }

    public HistoryCommandHandler(int maxHistoryLength) {

        if (maxHistoryLength < 0)
            throw new IllegalArgumentException();
        this.maxHistoryLength = maxHistoryLength;
    }

    public void handle(Command cmd) {

        if (cmd.doIt()) {
            addToHistory(cmd);
        } else {
            history.clear();
        }
        if (redoList.size() > 0)
            redoList.clear();
    }

    public void redo() {
        if (redoList.size() > 0) {
            Command redoCmd = redoList.removeFirst();
            redoCmd.doIt();
            gpanel.repaint();    "gpanel": Unknown word.
            history.addFirst(redoCmd);

        }
    }

    public void undo() {
        if (history.size() > 0) {
            Command undoCmd = history.removeFirst();
            undoCmd.undoIt();
            gpanel.repaint();    "gpanel": Unknown word.
            redoList.addFirst(undoCmd);
        }
    }

    private void addToHistory(Command cmd) {
        history.addFirst(cmd);
        if (history.size() > maxHistoryLength) {
            history.removeLast();
        }
    }
}
```

NaiveCommandHandler.java

```java
package command;

public class NaiveCommandHandler implements CommandHandler {

    @Override
    public void handle(Command cmd) {
        cmd.doIt();
    }

}
```

Il pattern composite mi è stato molto d'aiuto per memorizzare tutti i comandi a cui era possibile fare poi un redo oppure un undo, grazie alla funzione doIt ed UndoIt è stato molto piu' semplice ed ordinato creare dei metodi per annullare alcune azioni, che molto spesso non sono affatto banali come una delete di un gruppo. Le classi specifiche per i comandi che implementano Command, sono all'interno del package Specificcommand, esiste quindi una classe per ogni comando, con doIt ed UndoIt.



FACTORY METHOD PATTERN

GraphicObjectViewFactory.java

```java
package view;

import java.util.HashMap;
import java.util.Map;

import shapes.GraphicObject;

public enum GraphicObjectViewFactory {

    FACTORY;

    private final Map<Class<? extends GraphicObject>, GraphicObjectView> viewMap = new HashMap<>();

    GraphicObjectView createView(GraphicObject go) {
        return viewMap.get(go.getClass());
    }

    public void installView(Class<? extends GraphicObject> clazz, GraphicObjectView view) {
        viewMap.put(clazz, view);
    }
}
```

CircleObjectView.java

```java
package view;

import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Point2D;

import shapes.CircleObject;
import shapes.GraphicObject;

public class CircleObjectView implements GraphicObjectView {

    @Override
    public void drawGraphicObject(GraphicObject go, Graphics2D g) {
        CircleObject co = (CircleObject) go;
        Point2D position = co.getPosition();
        double r = co.getRadius();
        double x = position.getX() - r;
        double y = position.getY() - r;
        g.draw(new Ellipse2D.Double(x, y, r * 2.0, r * 2.0));
    }

}
```

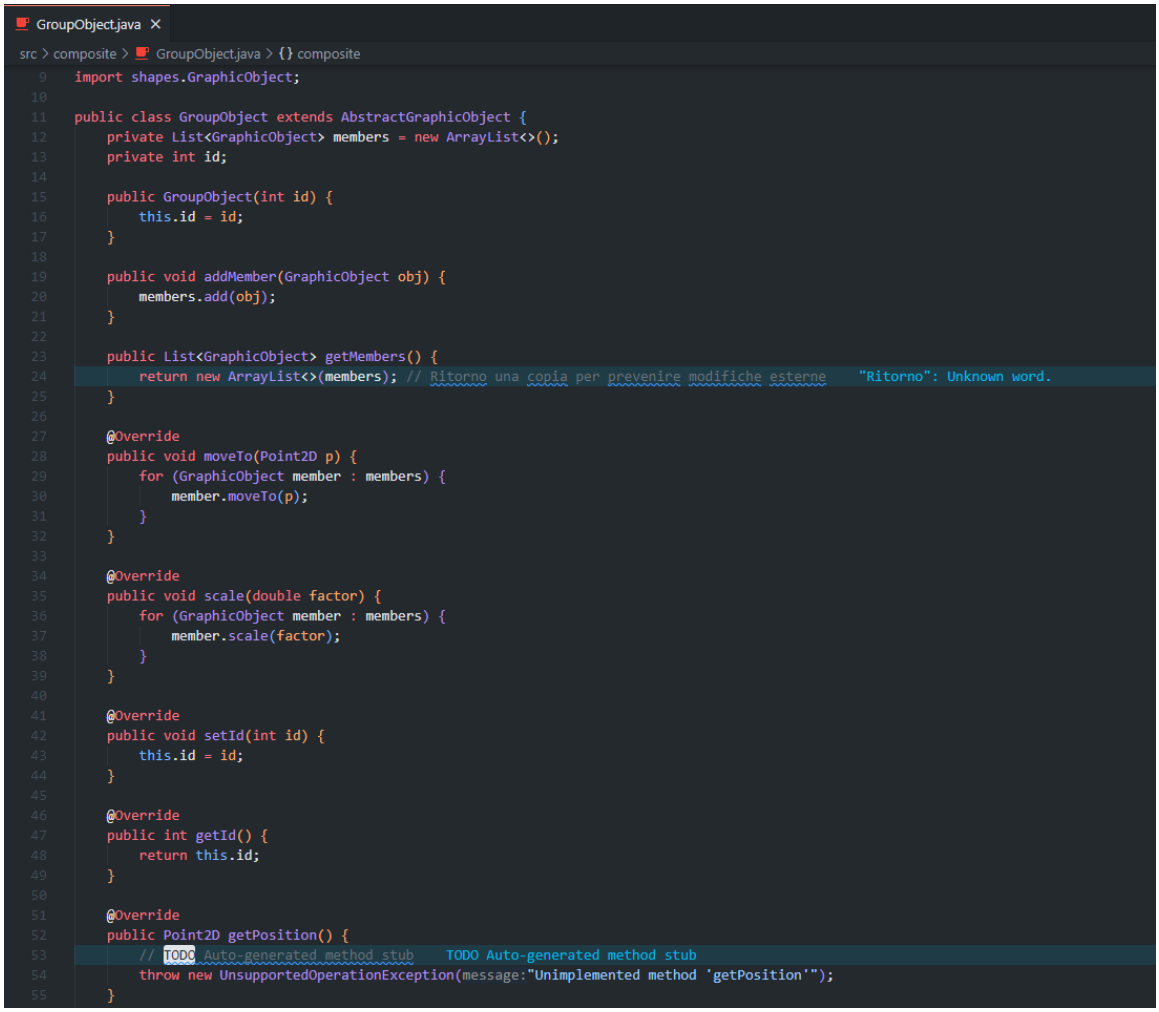
CircleObjectView.java | ImageObjectView.java

```java
package view;

import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.geom.Dimension2D;
import java.awt.geom.Point2D;

import shapes.GraphicObject;
import shapes.ImageObject;

public class ImageObjectView implements GraphicObjectView {
    @Override
    public void drawGraphicObject(GraphicObject go, Graphics2D g) {
        ImageObject io = (ImageObject) go;
        Dimension2D dim = io.getDimension();
        Point2D position = io.getPosition();
        Image image = io.getImage();
        int w = (int) (dim.getWidth());
        int h = (int) (dim.getHeight());
        int x = (int) (position.getX()) - w / 2;
        int y = (int) (position.getY()) - h / 2;

        g.drawImage(image, x, y, w, h, observer:null);

    }
}
```

RectangleObjectView.java

```java
package view;

import java.awt.Graphics2D;
import java.awt.geom.Rectangle2D;
import shapes.GraphicObject;
import shapes.RectangleObject;

public class RectangleObjectView implements GraphicObjectView {

    @Override
    public void drawGraphicObject(GraphicObject go, Graphics2D g) {
        if (go instanceof RectangleObject) {
            RectangleObject rect = (RectangleObject) go;

            double x = rect.getPosition().getX();
            double y = rect.getPosition().getY();
            double width = rect.getWidth();
            double height = rect.getHeight();

            g.draw(new Rectangle2D.Double(x, y, width, height));
        }
    }
}
```

COMPOSITE PATTERN



```java
import shapes.GraphicObject;

public class GroupObject extends AbstractGraphicObject {
    private List<GraphicObject> members = new ArrayList<>();
    private int id;

    public GroupObject(int id) {
        this.id = id;
    }

    public void addMember(GraphicObject obj) {
        members.add(obj);
    }

    public List<GraphicObject> getMembers() {
        return new ArrayList<>(members); // Ritorno una copia per prevenire modifiche esterne    "Ritorno": Unknown word.
    }

    @Override
    public void moveTo(Point2D p) {
        for (GraphicObject member : members) {
            member.moveTo(p);
        }
    }

    @Override
    public void scale(double factor) {
        for (GraphicObject member : members) {
            member.scale(factor);
        }
    }

    @Override
    public void setId(int id) {
        this.id = id;
    }

    @Override
    public int getId() {
        return this.id;
    }

    @Override
    public Point2D getPosition() {
        // TODO Auto-generated method stub    TODO Auto-generated method stub
        throw new UnsupportedOperationException(message:"Unimplemented method 'getPosition'");
    }
```

GroupObject non è un oggetto normale come sono ad esempio circle, rectangle o image, è un oggetto che è in grado di “incapsularne” altri semplici, quelli scritti sopra, immagazinandoli in una struttura dati di tipo ArrayList, contenente appunto solo oggetti di tipo GraphicObject.

INTERPRETER PATTERN

La classe Parser è responsabile dell'interpretazione dei comandi analizzati dalla classe AnalizzatoreLessicale, che utilizza uno streamtokenizer per riconoscere lettere, simboli, numeri ecc.. tramite il metodo prossimoSimbolo. Il parser quindi inizializza l'analizzatore lessicale e gestisce i comandi in base a cosa trova scritto, indirizzando poi il tutto tramite i metodi appositi come parseMoveCommand, ParseCreateCommand ecc… che servono poi a richimare le classi dentro il package Specificcommand.

G

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

Requisiti funzionali:

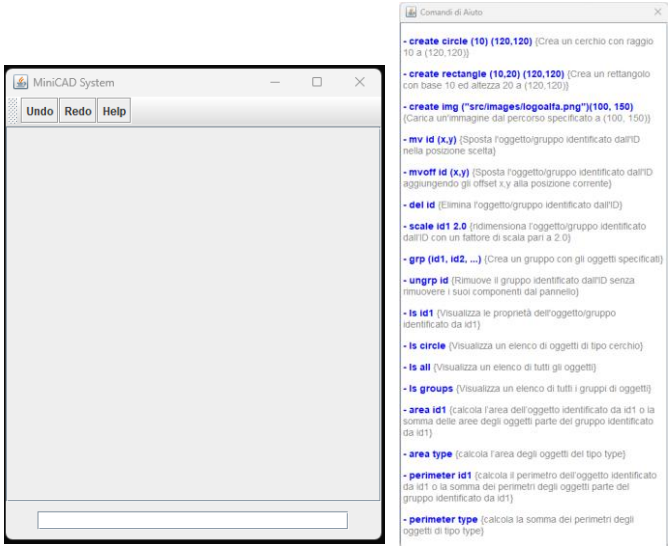
Sono stati tutti soddisfatti tramite i comandi appositi, elencati nella finestra che si apre tramite pulsante “help” nella interfaccia grafica.

Requisiti non funzionali:

- **Usabilita’**: l’interfaccia molto semplice ed intuitiva garantisce una esperienza utente semplice anche per chi non ha esperienza nel campo, è un buon mix che rende semplice l’utilizzo sia ad un utente esperto che non, grazie anche al bottone help che aiuta a chiarire eventuali dubbi sulle funzionalita dei comandi o sulla sintassi stessa.
- **Performance:** il sistema sfrutta l’inserimento in apposite hashmap, questo permette di essere leggero e rapido, qualsiasi comando è istantaneo, con un tempo di risposta immediato.
- **Affidabilità:** il sistema è stato testato in moduli, quindi ogni parte è stata posta a diversi test pratici e di stress, risolvendo qualsiasi tipo di bug trovato. Il sistema risulta quindi difficile da bloccare o far crashare.
- **Manutenibilità ed Estensibilità:**
  l’ordine dei vari package e della struttura complessiva, fa si che la manutenzione e l’eventuale aggiunta di un nuovo oggetto, risulti semplice e veloce. Il tutto testato, in quanto ho inserito prima img e circle, una volta finito tutto ho aggiunto il rectangle, è bastato aggiungere la classe per l’oggetto e quella della view, automaticamente tutti i metodi funzionano senza bisogno di adattarsi a nuovi aggiornamenti.

Appendix. Prototype

Questa è come si presenta l'interfaccia grafica assieme alla finestra help:



Questa è come si presenta l'interfaccia con alcuni oggetti creati sopra: