**“Ingegneria del Software” 2021-2022**

# <MiniCAD>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data** | <16/11/2022> | | | |
| **Documento** | Documento Finale – D3 | | | |
| **Team Members** | | |  |
| **Nome e Cognome** | | **Matricola** | **E-mail address** |
| Antonino Vaccarella | |  |  |
|  | |  |  |
|  | |  |  |
|  | |  |  |
|  | |  |  |

## Sommario

### 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** |
| Definire l'interpretazione dei comandi | 14/05/2022 | 20/05/2022 | I comandi fanno parte di una grammatica EBNF e quindi la challenge è stata definire una gerarchia che rappresentasse la grammatica e tutti i vincoli che essa descrive.  Questa challenge è stata risolta con l'ausilio del Design Pattern Interpreter. |
| Creazione dei comandi veri e propri | 21/05/2022 | 29/05/2022 | I comandi si presentano in varie tipologie, alcuni molto specifici (con id, tipo dell'oggetto, ecc.), altri invece molto generici (*ls groups, ls all*).  La challenge è stata individuare la struttura adeguata per garantire la corretta implementazione, utilizzando il Design Pattern Command. |
| Il metodo per identificare il giusto comando utilizzando l'interpretazione della grammatica | 30/05/2022 | 02/06/2022 | Ogni comando interpretato corrisponde ad uno specifico comando da eseguire sull'applicativo.  La challenge è stata mettere in piedi un meccanismo che garantisse l'esecuzione del comando corretto da parte del sistema, con l'ausilio del Factory Method. |
| Gestire la reversibilità di alcuni comandi (*undo*) | 03/06/2022 | 07/06/2022 | Alcuni dei comandi devono prevedere l'*undo (Create, Remove, Move, Scale, Group, Ungroup*).  Per effettuare ciò, attraverso il Design Pattern Memento viene salvata un'istantanea del sistema che ne garantisce la reversibilità. |

## A. Stato dell’Arte

MiniCAD nasce come estensione dell'applicazione per la manipolazione di oggetti grafici bidimensionali vista a lezione.

Esso permette, tramite dei comandi inseriti in console, la creazione di cerchi, rettangoli e immagini con la possibilità di fare alcune operazioni su di essi.

L'utente può dunque creare un nuovo oggetto, rimuoverlo, spostarlo e ridimensionarlo.

Dopo aver creato degli oggetti, l'utente può verificare la lista di quelli presenti, calcolare l'area e il perimetro di un singolo oggetto, di oggetti della stessa tipologia oppure calcolare la somma delle aree o perimetri di tutti gli oggetti.

Vi è inoltre la possibilità di creare gruppi di oggetti sui quali è possibile fare anche le operazioni sopra descritte.

## B. Raffinamento dei Requisiti

### B.1 Servizi (con prioritizzazione)

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **SERVIZIO** | **IMPORTANZA** | **COMPLESSITA’** |
| #1 | Inserimento del comando | Media | Bassa |
| Tramite console è possibile specificare un comando che l'utente andrà ad inserire. | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **SERVIZIO** | **IMPORTANZA** | **COMPLESSITA’** |
| #2 | Interpretazione comando inserito in input dall'utente | Alta | Alta |
| Il sistema deve saper interpretare il comando inserito, secondo la grammatica EBNF definita e rifletterne a video i cambiamenti dettati dall'esecuzione. | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **SERVIZIO** | **IMPORTANZA** | **COMPLESSITA’** |
| #3 | Possibilità di annullare i comandi inseriti | Media | Media |
| L'utente può richiedere l'annullamento di alcuni comandi inseriti tramite console.  I comandi reversibili sono: *Create, Remove, Move, Scale, Group, Ungroup.* | | | |

### B.2 Requisiti non Funzionali

1. Validazione dell'input dell'utente (deve essere rispettata la grammatica nell'inserimento dei comandi).
2. Validazione del path delle immagini (il path per la definizione di un'immagine deve essere valido).

### B.3 Scenari d’uso dettagliati

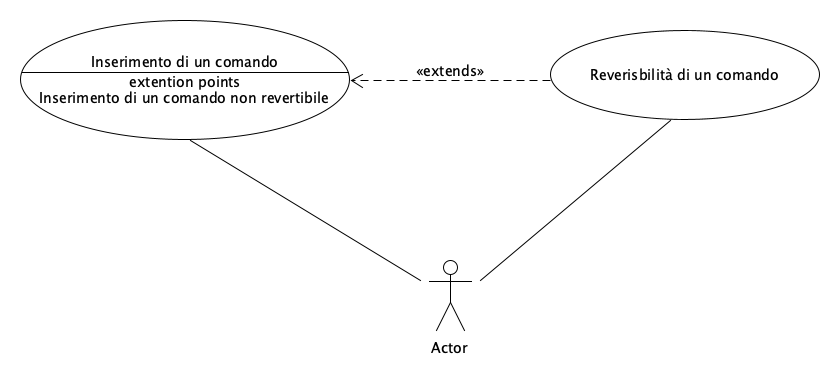
|  |  |
| --- | --- |
| **Inserimento di un comando** | |
| Tipo | Primario |
| Svolgimento normale | **1.** L’utente inserisce tramite la console un comando di tipo: *Create, Remove, Move, Scale, Group, Ungroup*.  **2.** Il sistema interpreta il comando inserito verificandone la validità.  **3.** Il sistema identifica il comando adeguato e lo esegue.  **4.** Il sistema mostra a video gli effetti del comando.  **5.** Il sistema mostra a video un pulsante *undo* che permette la reversibilità dell'ultimo comando inserito. |
| Svolgimento alternativo | **1a.** L'utente inserisce tramite la console un comando di tipo: *Area, Perimeter, List.*  **4a.**Il sistema stampa in console gli effetti del comando eseguito.  **5a.** Il sistema non mostra a video il pulsante *undo* in quanto il comando non è reversibile. |
| Descrizione | Questa operazione permette all’utente di inserire il comando che il sistema dovrà eseguire. |

|  |  |
| --- | --- |
| **Reversibilità di un comando** | |
| Tipo | Primario |
| Svolgimento normale | **1.** L’utente clicca sul pulsante *undo*.  **2.** Il sistema ripristina lo stato precedente all'ultimo comando inserito.  **3.** Il sistema ripercuote a video le modifiche.  **4.** Il sistema elimina il pulsante *undo*. |
| Descrizione | Questa operazione permette all’utente di annullare l'ultimo comando inserito. |

### B.4 Assunzioni

* Si assume esistano solo tre tipi di oggetti grafici: *Rettangolo, Immagine* e *Cerchio.*

### B.5 Use Case Diagrams

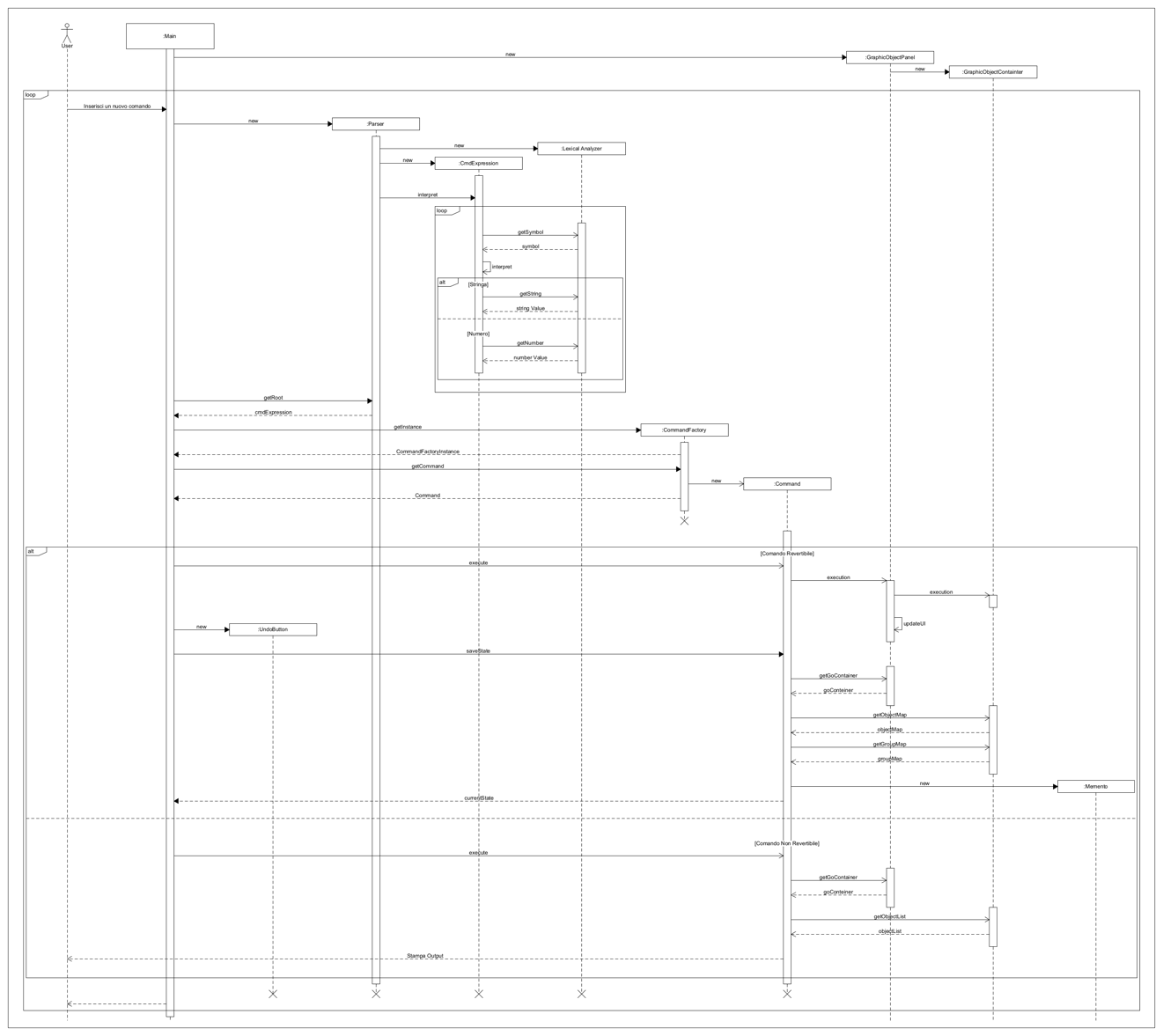


## C. Architettura Software

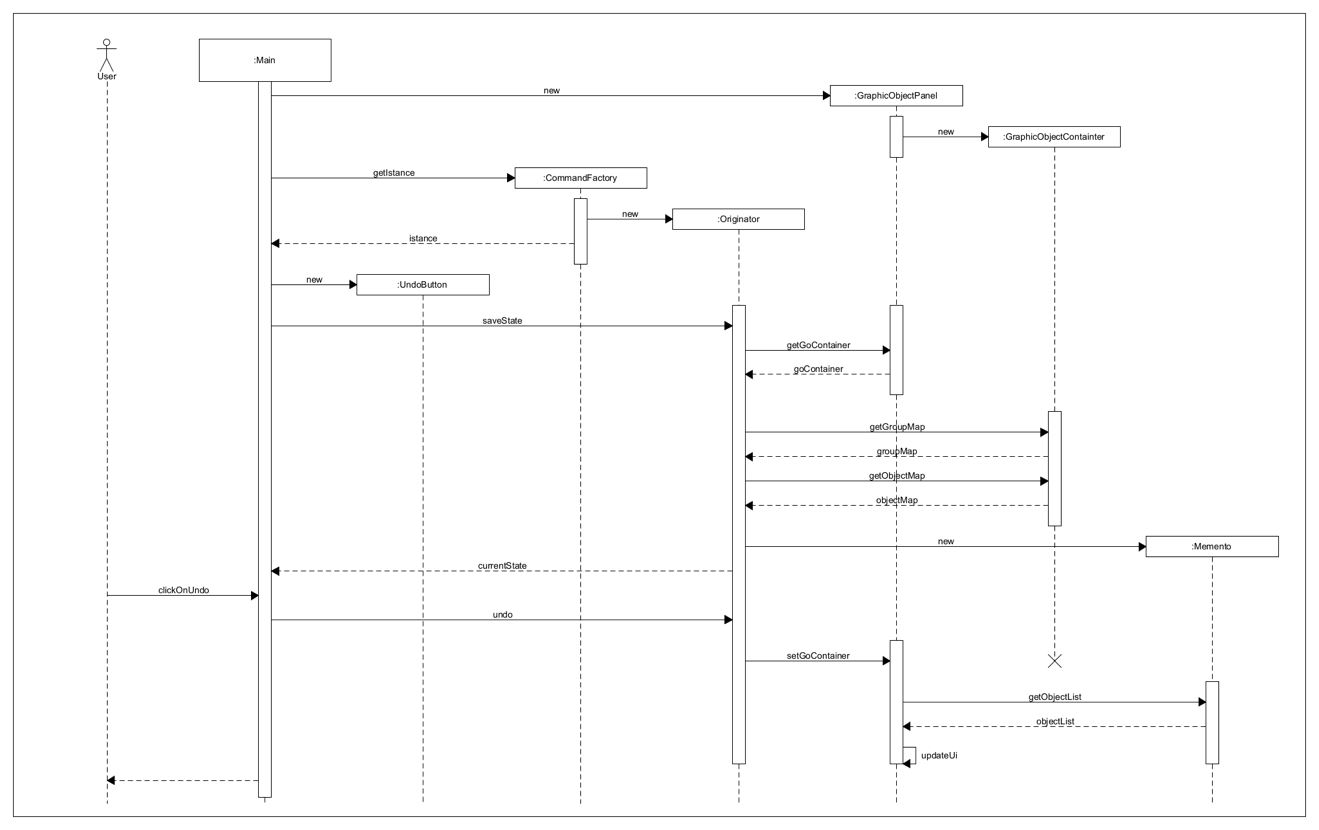
### C.1 The static view of the system: Component Diagram

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

|  |
| --- |
| **Inserimento di un nuovo comando** |



|  |
| --- |
| **Annullamento di un comando reversibile** |



## E. Scelte Progettuali (Design Decisions)

|  |  |
| --- | --- |
| **N°** | **SCELTA PROGETTUALE** |
| 1 | Design Pattern Interpreter  Per l'interpretazione dei comandi, considerando che fanno parte di una grammatica EBNF, è stato scelto l'utilizzo del Design Pattern Interpreter in modo tale da garantire il rispetto dei vincoli che la stessa comporta.  L'analisi dell'input dell'utente è stata svolta attraverso la classe *Lexical Analyzer.*  Il parsing dell'input dell'utente è stato svolto dalla classe Parser che restituisce al termine dell'interpretazione il comando interpretato.  I simboli terminale sono stati racchiusi in una *enum* chiamata *TerminalExpression*.  Tutte le espressioni non terminali sono state, invece, definite ognuna con la propria classe (*AreaExpression, PerimeterExpression, ecc.*). |
| 2 | Design Pattern Composite  Alcuni comandi non terminali sono dati dalla composizione di altri simboli non terminali.  Per gestire correttamente questa gerarchia è stato utilizzato il pattern Composite, inglobando all'interno di un'espressione non terminale, l'insieme delle espressioni non terminali che la compongono.  es. **new rectangle (1.0,1.0) (5.0,5.0)**  Questa espressione è formata dal simbolo terminale **new** e dal simbolo non terminale *typeconstr*  e dal simbolo non terminale *pos*.  A sua volta, il simbolo non terminale *typecontr* è formato dal simbolo terminale **type**  e dal simbolo non terminale *pos*. |
| 3 | Design Pattern Command  Per gestire la catena di comandi da eseguire è stato scelto il pattern Command.  Ognuna delle classi definite, rappresenta un comando atomico disponibile nell'applicativo.  Per la gestione dei comandi attinenti, è stata costruita una gerarchia con un oggetto astratto contenente le funzioni comuni e i suoi figli che contengono la logica del comando concret.  Ad esempio:  *AbstractAreaCommand* rappresenta il comando astratto dell'area contenente la formula per calcolare l'area.  *AreaCommand*, *AreaObjectCommand* e *AreaTypeCommand* sono delle implementazioni complete che permettono, rispettivamente, di eseguire il calcolo dell'area su tutti gli oggetti, sugli oggetti di un particolare tipo, su un oggetto specifico. |
| 4 | Design Pattern Factory Method  Il set delle espresssioni ed il set dei comandi concreti sono due gerarchie gestite separatamente all'interno dell'applicativo.  Per permettere la creazione del corretto comando concreto, partendo dall'interpretazione dell'espressione inserita dall'utente, è stato scelto l'utilizzo del Factory Method.  Questo ci permette di separare il comando vero e proprio da ciò che inserisce l'utente in input. |
| 5 | Design Pattern Memento  Alcuni comandi (*Create, Remove, Move, Scale, Group, Ungroup)* devono garantire la reversibilità.  Per far ciò è stato scelto l'utilizzo del pattern Memento, in modo tale che all'esecuzione di uno di essi, viene salvata un'istantanea dell'applicativo ripristinabile qualora l'utente clicchi sul tasto *undo*.  La classe *GraphicObjectContainer*, che definisce l'insieme dei nostri oggetti, è un'estensione dell'interfaccia Memento e rappresenta lo stato del nostro applicativo.  L'*Originator* è dato dal comando reversibile stesso. |

## F. Progettazione di Basso Livello

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

L'architettura del sistema software è caratterizzata dai seguenti componenti principali:

### INTERPRETER

L'interpreter racchiude le logiche di validazione dei comandi inseriti dall'utente, come definito dalla grammatica EBNF.

La classe *Parser* si occupa di effettuare il parsing dell'input ed espone il metodo getRoot() che permette di ottenere l'espressione interpretata (a valle dell'interpretazione).

La classe LexicalAnalyzer si occupa dell'analisi dei frammenti dell'input.

Espone Alcuni metodi funzionali, quali:

* nextSymbol() che permette di passare al frammento successivo ed interpretarlo restituendo il simbolo terminale analizzato.
* getString() che permette di ottenere la stringa del frammento analizzato.
* getNumber() che permette di ottenere il valore numerico, qualora sia stato riconosciuto come frammento di tipo Number, del frammento corrente.

L'interfaccia NonTerminalExpression rappresenta l'astrazione dell'espressione.

Viene implementata da tutti i simboli non terminali presenti ella grammatica. Ognuno di essi avrà una sua classe.

Per rappresentare la composizione di diversi simboli non terminali, quelli terminali potranno contenere, a seconda di quanto specificato dalla grammatica, altre classi non terminali.

L'interfaccia espone il metodo interpret() che restituisce l'interpretazione dello stesso simbolo non terminale.

La classe CmdExpression rappresenta il set di simboli non terminali che raffigurano il set di istruzioni eseguibili sull'applicativo, seguendo le regole della grammatica EBNF.

All'interno delle classi che compongono la gerarchia dell'Interpreter, troviamo la validazione dell'input dell'utente accettando solamente comandi ben formati.

### COMMAND

Per garantire che il sistema possa offrire i servizi definiti dalla grammatica EBNF, i comandi vengono definiti negli appositi oggetti che implementano l'interfaccia Command.

Questi oggetti interagiscono con il GraphicObjectContainer che rappresenta l'insieme degli oggetti presenti nell'applicativo, con il GraphicObjectPanel che svolge il ruolo di controller grafico ed ha la responsabilità di far visualizzare all'utente l'output dei comandi inseriti; e con gli oggetti presenti nel *model* che rappresentano l'insieme di oggetti costruibili all'interno dell'applicativo (Rettangolo, Immagine e Cerchio).

### MEMENTO

Per garantire la reversibilità dei comandi, il main avrà il ruolo di Care-Taker, cioè immagazzina lo stato dell'applicativo quando viene effettuato un nuovo comando reversibile.

I comandi reversibili implementano l'interfaccia *Originator* che espone i metodi saveState(), che si occupa di effettuare una copia dello stato attuale dell'applicativo e *undo()* che si occupa di ripristinare lo stato dell'applicativo.

Il *GraphicObjectContainer()* implementa la classe Memento e rappresenta quindi un'istantanea dell'applicativo.

Quando l'utente clicca su *undo* il comando reversibile si occuperà di ripristinare lo stato, mentre il GraphicObjectPanel (il controller) si occuperà di aggiornare le informazioni a video, secondo lo stato ripristinato.

## H. Testing

Per testare l'applicativo ho creato due classi *InterpreterTest* e *CommandTest*, utilizzando JUnit 5, in modo tale da testare i due componenti principali del progetto: l'interprete e i comandi da eseguire.

H1. InterpreterTesting

Immagine che contiene testo

Descrizione generata automaticamente

All'interno di questa classe sono presenti due test: testCreateExpression() e testCreateWithWrongExpression().

Il primo ci permette di testare l'esecuzione di un comando di tipo create (es. "new circle (20.0) (200.0,200.0)").

Attraverso il metodo assertEquals(), fornito da JUnit, andiamo a verificare che la catena delle espressioni create per il pattern interpreter sia correttamente eseguita e che quindi il comando sia correttamente interpretato.

Il secondo metodo, testCreateWithWrongExpression(), ci permette di testare il comportamento dell'applicativo quando si immette un comando che viola la grammatica EBNF.

Con il metodo assertThrows(), fornito da JUnit, andiamo a verificare che il sistema vada a sollevare l'eccezione corretta, definita nell'applicativo.

Immagine che contiene testo

Descrizione generata automaticamenteRisultati

H2. CommandTesting

Immagine che contiene testo

Descrizione generata automaticamente

All'interno di questa classe sono presenti due test: createCommand() e createCommandWithException().

Il primo ci permette di testare l'esecuzione di un comando di tipo create (es. "area all").

Attraverso il metodo assertEquals(), fornito da JUnit, andiamo a verificare che il comando creato sia quello corretto.

Il secondo metodo, createCommandWithException(), ci permette di testare il comportamento dell'applicativo quando si immette un comando errato.

Con il metodo assertThrows(), fornito da JUnit, andiamo a verificare che il sistema vada a sollevare l'eccezione corretta, definita nell'applicativo.

Risultati

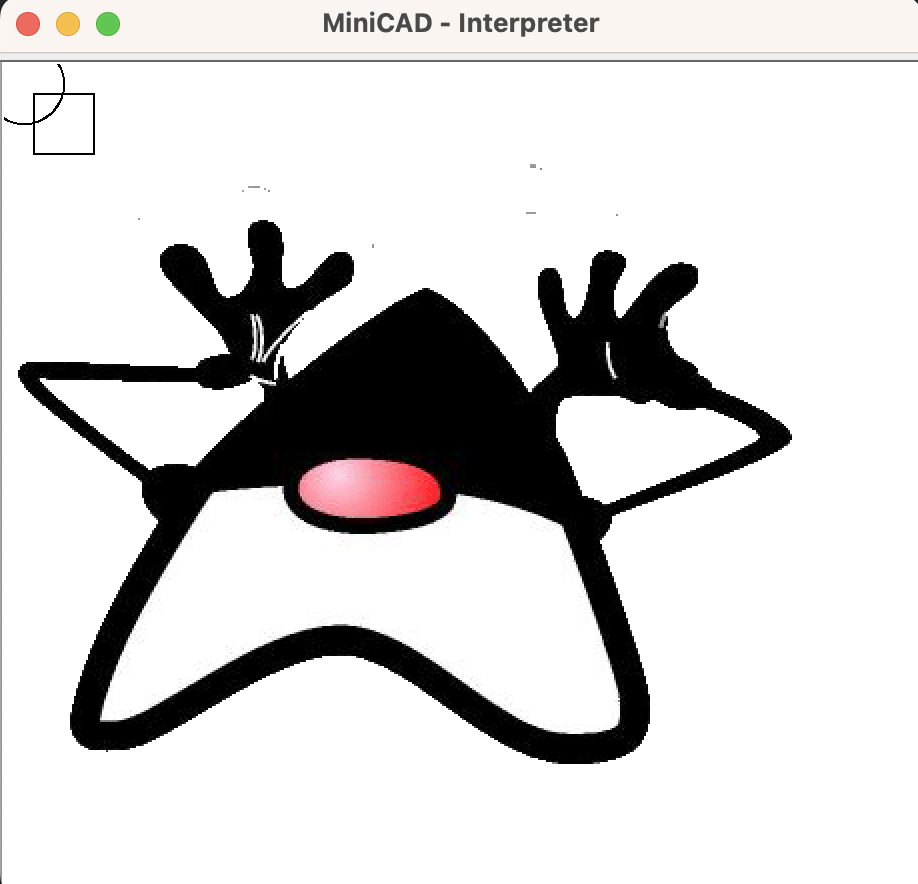
Immagine che contiene testo

Descrizione generata automaticamente

## Appendix. Prototype

Il sistema si presenta all'utente attraverso una grafica implementata dalla classe Main che costituisce un JFrame di Java Swing.

Al suo interno troviamo un **JScrollPane** principale che conterrà gli oggetti dell'applicativo.

Immagine che contiene testo

Descrizione generata automaticamente

In alto troviamo una JToolbar, che conterrà un JButton che rappresenta l'*undoButton* che permette all'utente di annullare l'ultimo comando inviato.

Tale comando, insieme alla Toolbar, verranno visualizzati solamente quando viene inserito in input un comando di tipo reversibile.

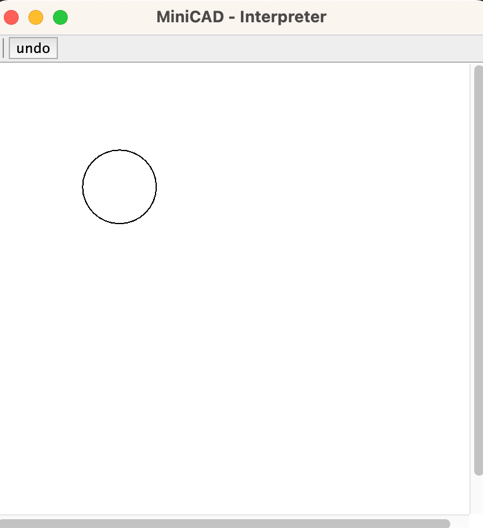
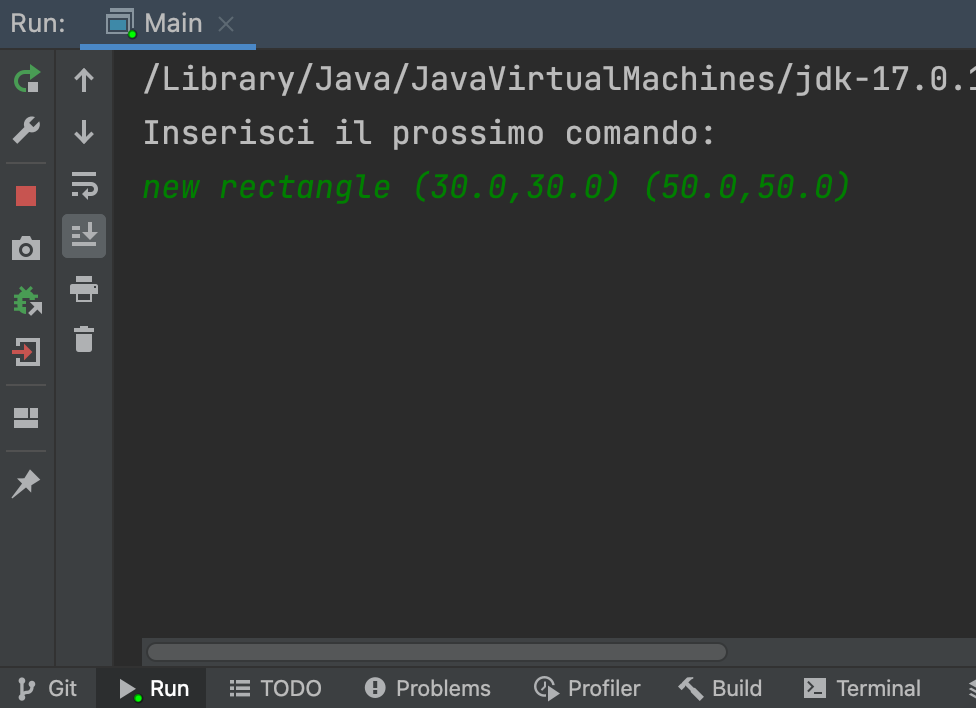


Immagine che contiene testo

Descrizione generata automaticamente

I comandi vengono immessi dall'utente attraverso la console:



I comandi di tipo *area, perimeter e list* avranno un output nella console:

