Relazione Progetto Programmazione Avanzata (jlogo110944)

Piero Hierro
MATRICOLA:110944

Introduzione

In questa relazione sarà illustrato il progetto di **Programmazione Avanzata**, sviluppato in Java.

Il progetto si basa sul **pattern MVC** ed è quindi suddiviso in Model, View e Controller. Per facilitare la lettura del codice il progetto è stato diviso in 3 packages (**Model, View e Controller**).

All'interno di questi packages c'è un ulteriore package denominato **"Implementation"** il quale, come il nome suggerisce, contiene le implementazioni delle interfacce contenute dei package sopra citati.

Tecnologie utilizzate

Per gestire il processo di compilazione si è fatto uso di **Gradle** (versione 7.1.1). La versione di java utilizzata è **Java 11** e per quanto riguarda l'interfaccia grafica si è fatto uso di **JavaFX**

(versione 11.0.2) e di **Scene Builder** per creare i layout più facilmente.

Descrizione delle responsabilità

Model

- (Interfaccia) **Line:** le classi che implementano questa interfaccia hanno la responsabilità di gestire le informazioni di una linea (estremi della linea, colore e spessore).
  - (Classe) **BasicLine:** implementazione base dell'interfaccia Line, parametrizzata con un tipo generico per rappresentare le locazioni degli estremi.
- (Interfaccia) **RGBColor:** le classi che implementano questa interfaccia hanno la responsabilità di rappresentare un colore RGB.
  - (Classe) **BasicRGBColor:** implementazione base dell'interfaccia RGBColor. Permette di ottenere uno dei tre colori RGB sotto forma di dato primitivo byte.
- (Interfaccia) **Area:** le classi che implementano questa interfaccia hanno la responsabilità di gestire le informazioni di un'area (colore e linee che la compongono).
  - (Classe) **BasicArea:** implementazione di base dell'interfaccia Area.
- (Interfaccia) **Cursor:** le classi che implementano questa interfaccia hanno la responsabilità di immagazzinare e poter modificare le caratteristiche di un cursore (posizione, colore di linea, angolo e plot).
  - (Classe) **BasicCursor:** implementazione base dell'interfaccia Cursor. Permette di accedere e modificare tutti i campi di un cursore.

- (Interfaccia) **Panel:** le classi che implementano questa interfaccia hanno la responsabilità di gestire il campo da disegno, potendo, ad esempio, aggiungere linee e aree. È anche possibile aggiungere o togliere dei listener.
  - (Classe) **CartesianPanel:** implementazione dell'interfaccia Panel la quale usa le coordinate cartesiane per individuare le locazioni nel campo da disegno. Permette di ottenere il cursore a esso associato, le aree individuate, le linee che possono formare un'area chiusa, il colore di sfondo, il colore di riempimento e anche muovere il cursore. Attraverso il metodo result() si ottiene una stringa rappresentativa dei comandi eseguiti sul pannello.
- (Interfaccia) **PanelUpdateListener:** le classi che implementano questa interfaccia hanno la responsabilità di gestire gli eventi scaturiti quando il pannello cambia il suo stato.
- (Classe final) **PanelUpdateSupport:** ha un solo compito, notificare tutti i listener quando il pannello cambia il proprio stato. Queste notifiche vengono fatte attraverso metodi synchronized per evitare problemi si concorrenza sull'interfaccia grafica.
- (Interfaccia) **LogoCommand:** interfaccia con vari metodi statici in cui ogni metodo rappresenta un comando Logo.
- (Interfaccia) **AreaFinder:** le classi che implementano questa interfaccia hanno il compito di trovare un'area chiusa nel pannello ad esso associato.
  - (Classe) **GraphAreaFinder:** implementazione dell'interfaccia AreaFinder la quale usa un grafo per individuare aree chiuse. Fa uso dell'algoritmo sui grafi DFS e il metodo di colorazione dei nodi in bianco, grigio e nero. Se trova un ciclo all'interno del grafo ritorna un Optional dell'area, altrimenti viene restituito un Optional.empty().
- (Classe final) **CartesianLocation:** classe usata per rappresentare una coordinata cartesiana, è usata per

parametrizzare le classi che usano un tipo generico come locazione.

Controller

- (Interfaccia) **Controller:** le classi che implementano questa interfaccia hanno la responsabilità di creare nuovi pannelli, fornire un risultato dell'esecuzione del programma, caricare i comandi come stringa di testo o da file, eseguire i comandi caricati e permettere di esportare i comandi caricati in un file.
  - (Classe) **BasicController**: implementazione di base dell'interfaccia Controller. Permette l'interazione tra Model e View.

View

- (Classe) **MainFXController:** classe che estende la classe astratta Application e implementa l'interfaccia PanelUpdateListener. controlla lo stage principale.
- (Classe) **NewPanelController:** controlla lo stage per la creazione di nuovi pannelli.
- (Enum) **PanelSize:** contiene le dimensioni (altezza e larghezza) di default dei pannelli.
- (Classe) **ResultController:** controlla lo stage in cui è possibile visualizzare il risultato del programma.
- (Classe) **WriteCommandsController:** controlla lo stage per scrivere i comandi.
- (Classe) **LoadCommandsController:** controlla lo stage per caricare i comandi, è possibile caricarli scrivendoli oppure selezionando un file di testo.

Risultato del Programma

Una volta eseguiti tutti i comandi caricati è possibile pulsare il bottone *result* per ottenere una descrizione dello stato finale del pannello. La descrizione contiene informazioni base come ad esempio il colore di sfondo del pannello fino a cose più dettagliate come il numero di aree chiuse individuate e il colore e i vertici che la compongono. È possibile inoltre, dopo aver eseguiti tutti i comandi caricati, esportare i comandi inseriti attraverso il menu item *Export Commands.*

Test

I test sono sviluppati secondo Junit Jupiter.

Ogni classe del testing usa il tag @BeforeEach per inizializzare le variabili utili ai vari metodi del testing.

Sono stati testati tutti i metodi pubblici presenti nelle classi.

Conclusioni

Il progetto è stato sviluppato in modo tale da rispettare i **principi SOLID** e usando i **generics,** ottenendo così una discreta estendibilità a nuove funzionalità. Per il momento il progetto contiene solo i comandi richiesti dal docente, ma nulla vieta il poter aggiungerne altri come la possibilità di creare linee curve o vari tipi di poligoni.

Comandi

I comandi supportati sono:

- **FORWARD <dist>** : sposta il cursore in avanti (con dist>=0);
- **BACKWARD <dist>** : sposta il cursore indietro (con dist>=0);
- **LEFT <angle>** : ruota il cursore in senso antiorario dei gradi descritti dal parametro (con angle>=0 e <=360);
- **RIGHT <angle>** : ruota il cursore in senso orario dei gradi descritti dal parametro (con angle>=0 e <=360);
- **CLEARSCREEN** : cancella quanto disegnato;
- **HOME** : muove il cursore sulla posizione home;
- **PENUP** : stacca la penna dal foglio;
- **PENDOWN** : attacca la penna al foglio;
- **SETPENCOLOR <byte> <byte> <byte>** : imposta il colore della penna al colore RGB rappresentato dai tre byte (con byte>=0 e <=255);
- **SETFILLCOLOR <byte> <byte> <byte>** : imposta il colore di riempimento di un'area chiusa al colore RGB rappresentato dai tre byte (con byte>=0 e <=255);
- **SETSCREENCOLOR <byte> <byte> <byte>** : imposta il colore di background del campo da disegno al colore RGB rappresentato dai tre byte (con byte>=0 e <=255);
- **SETPENSIZE <size>** : imposta lo spessore della linea (con size >=1);
- **REPEAT <num> [<cmds>]** : ripete la sequenza di comandi <cmds> fino a <num> volte.

Ecco qualche comando da inserire (size 650x650):

1. PENUP

   BACKWARD 50

   LEFT 90

   FORWARD 100

RIGHT 90

PENDOWN

REPEAT 18 [REPEAT 5 [RIGHT 40 FORWARD 300 RIGHT 120] RIGHT 20]

2. REPEAT 180 [FORWARD 200 BACKWARD 200 RIGHT 2]
3. SETFILLCOLOR 255 0 255

   SETPENSIZE 5

   SETPENCOLOR 255 132 0

   REPEAT 2 [REPEAT 6 [FORWARD 80 RIGHT 60] RIGHT 120]

   SETSCREENCOLOR 0 230 0

4. SETPENSIZE 4

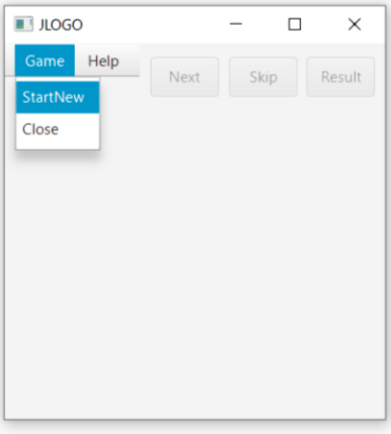
   REPEAT 12 [SETPENCOLOR 243 29 194 REPEAT 6 [FORWARD 100 LEFT 60] RIGHT 30]

Inoltre, all'interno della cartella resources (*JLogo>app>src>main>resources*) è possibile trovare un file di testo *(comandi.txt)* da cui importare ulteriori comandi.

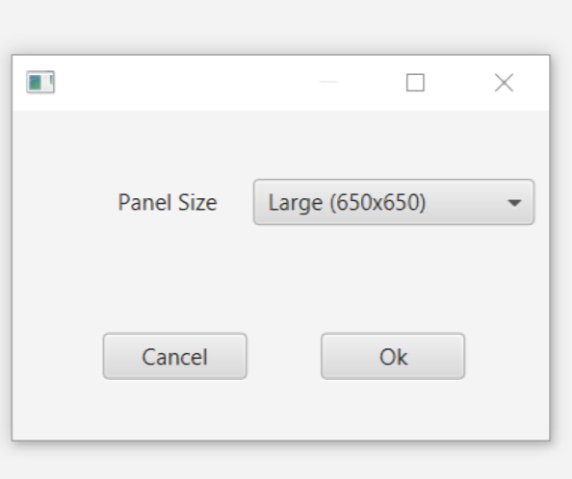
NOTA BENE: ogni linea deve cominciare con il nome di un comando (no spazi o numeri), il nome e gli argomenti del comando scelto devono essere separati da almeno uno spazio e ogni comando deve essere separato da un solo carattere di nuova riga (\n).

Tutorial

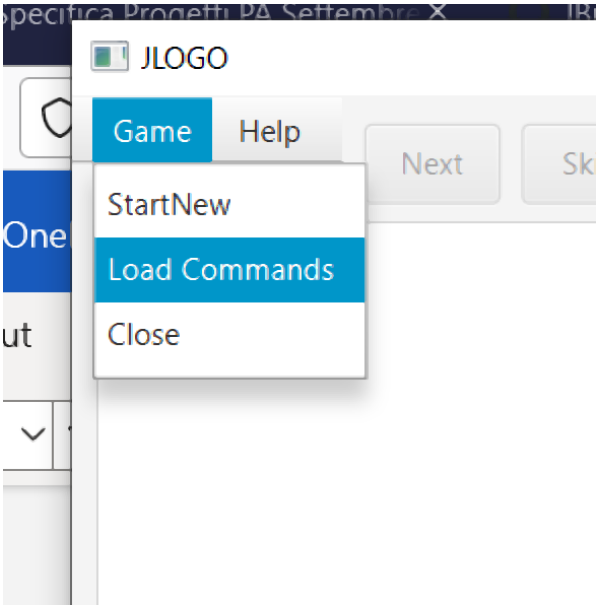
1. Creare un pannello



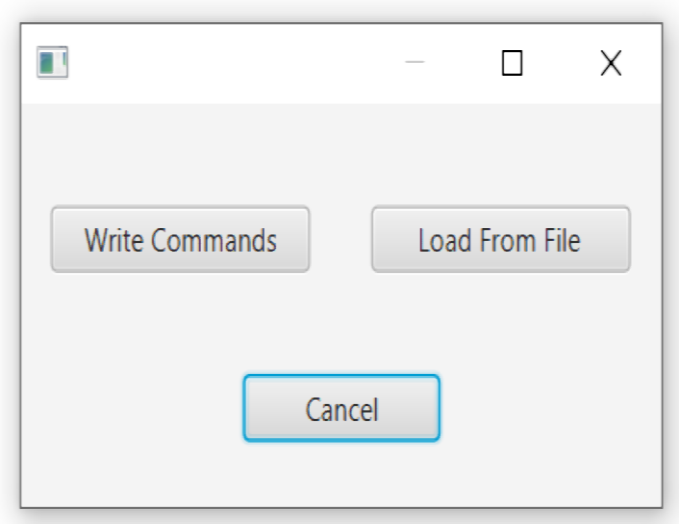
2. Scegliere le grandezze per il pannello



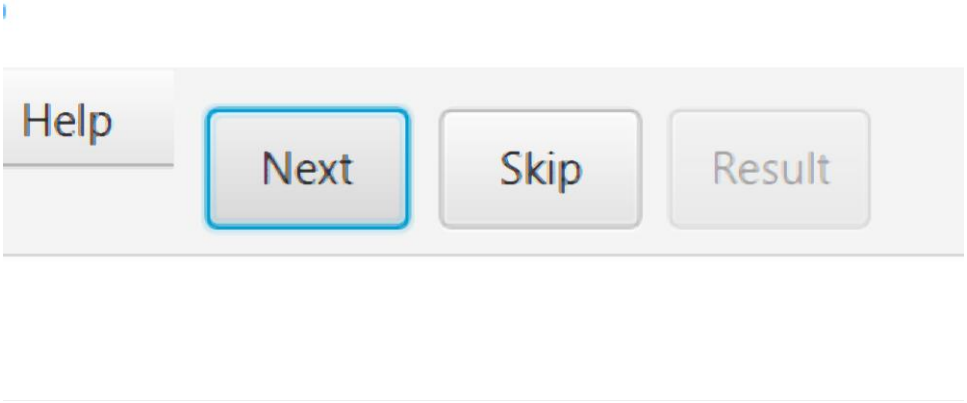
3. Caricare i comandi



4. Scegliere se scriverli o caricarli da file



5. Eseguire i comandi passo-passo o tutti di seguito



6. Esportare i comandi scegliendo la cartella su cui verrà poi creato il file denominato "commands.txt" (opzionale)