RELAZIONE PER L'ESAME DI PROGRAMMAZIONE AVANZATA SETTEMBRE 21/22

DI HAMZA BRAHMI 109429

Progetto: Programma **LOGO**

RESPONSABILITÀ	COME VIENE IMPLEMENTATA
Tenere traccia delle informazioni del cursore, come la sua attuale posizione, direzione, i colori della linea che genera e dell'area chiusa se verrà creata, lo spessore del tratto e il plot (se la penna è attaccata al foglio o no).	Attraverso l'interfaccia Cursor e la sua implementazione DefaultCursor.
Tenere traccia delle informazioni dell'Area di disegno, come la sua larghezza e altezza, il colore dello sfondo e la sua HOME. In più memorizzare le varie figure che vengono create durante l'esecuzione del programma LOGO.	Attraverso l'interfaccia Canvas e la sua implementazione DefaultCanvas.
Rappresentare una locazione in uno spazio.	Attraverso l'interfaccia Location e la sua implementazione GridLocation, le quali mi identificano la prossima locazione in una data direzione e distanza.
Creare le Figure da almeno tre segmenti consecutivi e adiacenti.	Attraverso l'interfaccia CanvasHandler e la sua implementazione DefaultCanvashandler, le quali mi gestiscono la logica dietro la creazione delle figure e il loro salvataggio nella Canvas.
Far evolvere il programma LOGO che consiste in una serie di istruzioni, o comandi, che permettono ad esempio il movimento del cursore, il cambio del colore dello sfondo, ecc.	Attraverso l'interfaccia Command e le sue implementazioni, una per ogni comando desiderato, all'interno del folder commands del model.
Creare questi comandi.	Questi comandi vengono creati attraverso l'interfaccia CommandFactory e la sua implementazione SwitchCaseCommandFactory, la quale attraverso un semplice switch case mi crea il comando della classe corretta. Inizialmente doveva essere una ReflectionCommandFactory, la quale mi avrebbe permesso di creare dinamicamente gli "agenti", in questo caso i comandi, istanziandoli in base al nome della loro classe e al loro numero di parametri memorizzati all'interno del enum CommandAvaliable, però per una serie di impegni non ho avuto il tempo di implementarlo.
Decifrare i comandi da una stringa letta.	Attraverso l'interfaccia CommandParser e la sua implementazione DefaultCommandParser, le quali utilizzando un CommandFactory creano una lista di comandi.
Definire il concetto di colore.	Attraverso la classe Color.

Definire il concetto di Direzione.	Attraverso la classe Direction.
Definire il concetto di linea creata dal movimento del cursore.	Attraverso l'interfaccia Line e la sua implementazione StraightLine. Line estende l'interfaccia Figure (vedi sotto).
Definire il concetto di Figura disegnata, la quale può essere una semplice linea o un poligono ad esempio.	Attraverso l'interfaccia Figure, la quale viene implementata sia dalla classe Polygon, che rappresenta una figura limitata da linee chiuse, sia dall'interfaccia Line, poiché una singola linea è comunque una figura che può essere salvata nella Canvas.
Leggere un file in input	Attraverso l'interfaccia funzionale CommandReader e la sua implementazione DefaultCommandReader, la quale prende un File in input e lo decifra con il CommandParser e li istanzia con il CommandFactory.
Scrivere il risultato finale dell'esecuzione del programma in un file.	Attraverso l'interfaccia funzionale ResultWriter e la sua implementazione DefaultResultWriter, la quale presa una Canvas e un file di output, scrive il risultato finale del programma, ovvero le informazioni della Canvas, come la larghezza, altezza e colore dello sfondo, e una lista delle figure disegnate.
Scrivere l'esecuzione "in tempo reale" del programma verso un output, in questo caso in una textArea.	Attraverso l'interfaccia funzionale ExecutionWriter e la sua implementazione DefaultExecutionWrite, la quale viene utilizzata ad ogni esecuzione di un comando per scriverne il risultato in output.
Controllare l'esecuzione di un programma LOGO.	Attraverso l'interfaccia Controller e la sua implementazione DefaultController, i quali gestiscono tutta l'esecuzione di un programma LOGO.
Controllare l'esecuzione della parte di JavaFX.	Attraverso il JLogoFXController, il quale utilizza il controller della classe per gestire l'esecuzione dell'applicazione JavaFX.

La descrizione delle classi usate per rappresentare i programmi Logo:

Un programma Logo è costituito da una serie di comandi.
In questa implementazione questi comandi vengono rappresentati dall'interfaccia Command. Ogni comando dovrà estenderla seguendo il contratto definito in essa, ovvero dovranno implementare i metodi execute() e resultOfCommand(). I vari comandi verranno memorizzati nel folder commands all'interno del folder model. Questi comandi, una volta letti dal file, vengono decifrati dall'interfaccia CommandParser e la sua implementazione DefaultCommandParser e in seguito creati attraverso l'interfaccia CommandFactory e la sua implementazione SwitchCaseCommandFactory.

La descrizione delle classi usate per rappresentare il disegno prodotto da un programma:

I disegni prodotti dall'esecuzione del programma vengono definiti dall'interfaccia Figure. In questa implementazione queste figure possono essere delle semplici linee, rappresentate dall'interfaccia Line che estende Figure e la sua implementazione StraightLine, o dei poligoni, rappresentate dalla classe Polygon che estende anch'essa Figure.
Queste figure vengono creati attraverso una logica descritta all'interno dell'interfaccia CanvasHandler e la sua implementazione DefaultCanvasHandler, le quali hanno la responsabilità di creare le linee e, in questo caso, quando tre linee consecutive e adiacenti si chiudono creano una figura.
Infine, queste figure vengono memorizzate all'interno della Canvas che utilizza queste classi.

La descrizione di come dette gerarchie possano essere “estese” per aggiungere nuovi comandi o disegni:

- Per aggiungere nuovi **comandi** è sufficiente creare una nuova classe che estende l’interfaccia Command, aggiungere il nome del comando e il numero dei suoi parametri all’interno del enum CommandAvaliable e aggiungere un case nello switch all’interno della classe SwitchCaseCommandFactory (questo passo non sarebbe necessario nel caso avessi terminato l’implementazione del ReflectionCommandFactory, la quale avrebbe creato dinamicamente i comandi in base al nome della loro classe).
- Per aggiungere nuove **figure** è sufficiente creare una nuova classe che estende:
    - nel caso si aggiunga un nuovo tipo di linea come un arco o una curva, l’interfaccia Line;
    - nel caso si aggiunga un nuovo tipo di figura, ad esempio un “non poligono” ovvero una figura delimitata da una linea curva o mista, l’interfaccia Figure.

  Per poterle creare, avendo disaccoppiato questa responsabilità dalla classe Canvas, basterà modificare la logica della loro creazione all’interno della classe DefaultCanvasHandler, che estende CanvasHandler. Nel caso di un tipo di linea si potrebbe anche inserire un nuovo tipo di comando, ad esempio “Curve”, che presi come parametri oltre la distanza anche un angolo, disegna un nuovo tipo di Line, ad esempio “CurvedLine”.

Istruzioni necessarie ad eseguire gli esempi consegnati ed a valutare il progetto:

All’interno del folder “Test” del package “App” sono presenti diversi file che rappresentano degli esempi di programmi LOGO, i quali contengono vari comandi per la creazione di figure.
Una volta avviato il programma, dovrà inserire la base e l’altezza dell’area di disegno (di default sono 1000x 1000) e in seguito potrà premere sul pulsante Open con il quale sceglierà il File contenente i comandi per la creazione dei disegni.
Una volta fatto potrà scegliere se eseguire il programma in modo automatico o step by step (in questo caso il pulsante stepBackward non è abilitato poiché non ho avuto di implementarlo correttamente, nel codice ho lasciato comunque la sua implementazione commentata).
L’esecuzione del programma potrà essere seguita, sia in modo automatico che step by step nella textArea a destra.
Al termine del programma potrà, o resettare il programma con il pulsante RESET, o salvare il risultato finale dell’esecuzione del programma LOGO in un File con il pulsante SAVE.
All’interno del folder dove ci sono gli esempi, ci anche i risultati delle loro esecuzioni.

Nota: all’interno del progetto ho lasciato commentato sia la parte relativa alla Reflection, all’interno della classe SwitchCaseCommandFactory, sia l’implementazione del comando stepBackward all’interno del DefaultController e del JLogoFXController, entrambe incomplete in caso volesse darle un’occhiata (principalmente per la parte sulla reflection). Altrimenti le può anche ignorare.