Relazione Progetto

Matteo Toma, Matricola 116781

Per realizzare un programma che permette di disegnare con un linguaggio LOGO, sono state utilizzate Java 17.0.1 e per quanto riguarda Gradle, il plugin org.openjfx.javafxplugin con versione 0.0.13.

È stato inoltre definito un file module-info.java contenente le componenti necessarie ad avviare l'applicazione grafica: javafx.controls e javafx.fxml. Contiene anche la componente per visualizzare i logger: java.logging .

Gestione delle responsabilità

Come pattern di progettazione si è scelto l'MVC (**Model-View-Controller),** in quanto è abbastanza semplice da applicare al progetto non essendo particolarmente complesso.
Inoltre, risulta più agevolato l'utilizzo della grafica in JavaFx, che interagisce con gli elementi del model.

Model

In questo package bisogna definire tutte le classi e le interfacce necessarie per rappresentare l'applicazione d'interesse.

Servono classi che rappresentino e gestiscano il piano cartesiano, altre per la rappresentazione della penna, altre per la sua direzione.

Fondamentale sarà la classe che si occupa di implementare tutti quelli che saranno le istruzioni dei comandi da interpretare.

Servirà una classe per poter importare tutti i colori, della penna e dello sfondo del piano.

Serviranno classi per poter memorizzare/calcolare informazioni quali direzione del cursore e informazioni sull'area.
Per poter capire come sono configurati e come rappresentare i punti delle rette nel piano del disegno, verrà usata una classe per rappresentare dei grafi non orientati.

View

Le classi del package View si occuperanno di istanziare tutte le componenti grafiche dell'interfaccia, inizializzarle e resettarle, infine dovranno gestire gli eventi associati ad ogni pulsante.

Controller

Le classi e interfacce del package controller, costruiranno il piano di lavoro in base alle dimensioni specificate.
Dovranno permettere di poter mostrare la precedente configurazione del piano e quindi permettere di mostrare il piano DOPO e PRIMA di un'istruzione letta dal file.
Dovrà inoltre gestire i casi in cui le istruzioni nel file non esistano tra quelle esistenti, e controllare che la sintassi sia giusta.

Il controller si occuperà anche di caricare i file in linguaggio LOGO da interpretare, e di salvare, tramite un file e su una path a scelta, le informazioni del disegno costruito sulla base delle informazioni interpretate.

Interfacce

Closed Area

Interfaccia contenente metodi che ritornano la lista delle linee, il colore e la stringa che compongono un'area chiusa.

Metodi:

- List<L> getArea();
- RGBColor getColor();
- String toString();

Cursor

Interfaccia contenente metodi relativi alla rappresentazione di un cursore, necessari a disegnare sul piano. Sono:

1. Posizione
2. Direzione
3. Colore Linea
4. Colore area
5. Plot (tracciato)

Metodi:

1. C getPosition(); Restituisce le coordinate X e Y della posizione del cursore.
2. setPosition(C position); Imposta il cursore in una nuova posizione nel piano.
3. D getDirection(); Restituisce la direzione verso cui il cursore punta.
4. setDirection(D direction); Imposta la direzione verso cui il cursore dovrà puntare.
5. RGBColor getLineColor(); Restituisce il colore della linea prodotta del cursore come conseguenza di uno spostamento.
6. setLineColor(RGBColor color); Imposta il colore della linea prodotta dal cursore come conseguenza di uno spostamento.
7. RGBColor getAreaColor(); Restituisce il colore dell' area formata da una serie di linee.
8. setAreaColor(RGBColor color); Imposta il colore dell'area prodotta quando una serie di linee producono un'aria chiusa.
9. isPlot(); indica se durante uno spostamento, il cursore genera o meno un tracciato.
10. setPlot(boolean plot); Imposta il plot a seconda della generazione di un tracciato.
11. Plane<C> getPlane(); Restituisce il piano in cui è contenuto il cursore.
12. isPen(); Restituisce true o false a seconda se la penna è attaccata o meno al piano.
13. penUp(); Imposta la penna attaccata al piano del cursore.
14. penDown(); Imposta la penna staccata dal piano del cursore.
15. getPenSize(); Restituisce la size del tratto della penna.
16. setPenSize(int size); Imposta la size alla penna.

Direction

Interfaccia contenente metodi per definire dove un oggetto può essere direzionato.

Metodi:

1. GenericDirection genericDirection(int angle) : Ritorna l'angolo verso cui l'oggetto punta la direzione.
2. D getDirectionWay(): direzione attuale
3. setDirectionWay(D direction) : direzione da impostare

Graph

Interfaccia contenente metodi per rappresentare i dati tramite dei grafi, per cercare quelli che formano un ciclo e rimuoverli.
Metodi:

1. void cycleDFS(GraphNode<Integer, D> u, GraphNode<Integer, D> p); Visita tutto il grafo tramite Depth-first Search per poter trovare uno o più cicli.
2. addArc(GraphNode<Integer, D> u, GraphNode<Integer, D> v); aggiunge un arco al grafo.
3. List<D> getCycle(GraphNode<Integer, D> u, GraphNode<Integer, D> p); Ritorna una lista e rimuove i nodi che formano un ciclo in un determinato grafo.
4. Clear(); per resettare il grafo
5. List<List<GraphNode<Integer, D>>> getMatrix(); Ritorna la matrice delle adiacenze di questo grafo, quindi tutti i nodi adiacenti a ciascun nodo.
6. List<List<Integer>> getCycles(); Ritorna la lista di tutti i cicli presenti nel grafo.
7. Map<Integer, D> getNodes(); Ritorna la mappatura tra l' indice intero di un nodo e l'oggetto contenuto in quel nodo.
8. getArcs(); Ritorna il numero totale di archi presenti nel grafo.
9. int getCycleNumber(); Ritorna il numero del ciclo attuale nel grafo.
10. List<Integer> getParents(); Ritorna la lista degli indici di tutti i nodi padre all'interno del grafo.
11. List<Integer> getMarkedNodes(); Restituisce, tramite una lista d'interi, gli indici dei nodi riguardanti un certo ciclo.
12. List<Integer> getColors(); Restituisce la lista contenente gli indici dei nodi colorati.

Instruction

Interfaccia contenente metodi necessari a rappresentare ed eseguire un' istruzione in linguaggio LOGO nel piano di lavoro.

Contiene i metodi per interpretare i diversi comandi:

- LEFT<angolo> ruota il cursore in senso antiorario dei gradi descritti dal parametro, nel range [0, 360]).
- RIGHT<angolo> ruota il cursore in senso antiorario dei gradi descritti dal parametro, nel range [0, 360]).
- CLEARSCREEN cancella cio che è disegnato.
- HOME muove il cursore nella posizione di default (centro le foglio).
- PENUP stacca la penna dal foglio.
- PENDOWN attacca la penna al foglio.
- SETPENCOLOR <byte> <byte> <byte> imposta il colore della penna al colore, rappresentato dal colore RGB rappresentato dai tre byte dati, nel range che va da 0 a 255.
- SETFILLCOLOR <byte> <byte> <byte> imposta il colore del riempimento di un'area chiusa.

- SETSCREENCOLOR <byte> <byte> <byte> imposta il colore di background dell'area di disegno.
- SETPENSIZE indica la grandezza del tratto della penna, <size> è un intero di grandezza >= 1;
- RIPETI<num> [<cmds>] ripete num volte la sequenza di comandi presenti nella lista dei comandi <cmds>.

Metodi:

1. **Plane<Point<Double>> move(char operator, Plane<Point<Double>> plane, Object distance):**
   Muove il cursore in avanti o indietro rispetto la sua posizione e direzione.
2. **checkCursorAtBorder(Plane<Point<Double>> plane, Point<Double> newCursorPosition, Point<Double> oldCursorPosition) :**
   posiziona il cursore al bordo del piano se supera i limiti di esso.
3. **Plane<Point<Double>> forward(Plane<Point<Double>> plane, Object... args) :**
   Metodo che implementa l' istruzione FORWARD. Sposta il cursore in avanti dalla posizione attuale, di una certa distanza passata in args[0]. Se il cursore supera l' altezza del piano per andare in avanti, si ferma al bordo del piano.
4. **Plane<Point<Double>> backward(Plane<Point<Double>> plane, Object... args):**
   Metodo statico che implementa l' istruzione BACKWARD. Sposta il cursore indietro verso la sua direzione di una certa distanza passata in args[0]. Se il cursore supera l' altezza del piano all' indietro, si ferma al bordo.
5. **Plane<Point<Double>> left(Plane<Point<Double>> plane, Object... args):**
   Metodo che implementa l' istruzione 'LEFT'. Ruota il cursore in senso antiorario rispetto ai gradi specificati.
6. **Plane<Point<Double>> right(Plane<Point<Double>> plane, Object... args):**
   Metodo statico che implementa l' istruzione 'RIGHT'. Ruota il cursore in senso orario rispetto ai gradi specificati.
7. **Plane<Point<Double>> clearScreen(Plane<Point<Double>> plane, Object... args):**
   Metodo che implementa l' istruzione 'CLEARSCREEN'. Cancella il disegno, liberando il piano di lavoro.
8. **Plane<Point<Double>> home(Plane<Point<Double>> plane, Object... args):**
   Metodo che implementa l' istruzione 'HOME'. Sposta il cursore in posizione di default nel piano, ossia quelle di coordinate (base/2,altezza/2).
9. **Plane<Point<Double>> penUp(Plane<Point<Double>> plane, Object... args):**
   Metodo che implementa l' istruzione 'PENUP'. Stacca la penna dal foglio.
   Non verrà generata nessuna linea anche se il cursore si sposta.
10. **Plane<Point<Double>> penDown(Plane<Point<Double>> plane, Object... args):**
    Metodo che implementa l' istruzione 'PENDOWN'. Attacca la penna al foglio, quindi se il cursore si sposta verrà generata una linea.
11. **Plane<Point<Double>> setPenColor(Plane<Point<Double>> plane, Object... args):**
    Metodo statico che implementa l' istruzione 'SETPENCOLOR'.
    Imposta il colore della penna con colori RGB, da usare in una combinazione specifica per impostare il colore.
12. **Plane<Point<Double>> setFillColor(Plane<Point<Double>> plane, Object... args):**
    Metodo statico che implementa l' istruzione 'SETFILLCOLOR'.
    Imposta il colore RGB per riempire l' area chiusa dalle linee, in base ai parametri specificati.
13. **Plane<Point<Double>> setScreenColor(Plane<Point<Double>> plane, Object... args):**

Metodo che implementa l' istruzione 'SETSCREENCOLOR'. Imposta il colore RGB di background del piano in base ai colori specificati.

14. **Plane<Point<Double>> setPenSize(Plane<Point<Double>> plane, Object... args):**
    Metodo che implementa l' istruzione 'SETPENSIZE'.
    Imposta lo spessore della penna in base alla dimensione specificata.
15. **Plane<Point<Double>> repeat(Plane<Point<Double>> plane, Object... args):**
    Metodo che implementa l' istruzione 'REPEAT'. Ripete la sequenza di comandi [cmds] per N volte.
16. **Plane<C> execute(Plane<C> plane, Object... args):**
    Esegue una delle istruzioni LOGO nel piano di disegno.

Line

Interfaccia contenente metodi necessari a rappresentare una generica linea per disegnare sul piano.

Metodi:

1. C getStartingPoint();
   Ritorna le coordinate dell'estremo iniziale della linea.
2. C getEndPoint(); Ritorna le coordinate dell' estremo finale della linea.
3. RGBColor getColor(); Ritorna il colore associato alla linea.
4. int getSize(); Ritorna lo spessore del tratto della linea.

Pane

Interfaccia contenente metodi necessari a rappresenta un generico piano bidimensionale per poter disegnare.
Metodi:

1. **Optional<Point<Double>> intersect(Line<Point<Double>> a, Line<Point<Double>> b):**
   Calcola il punto d'intersezione di due linee.
2. **double getLength();** Ritorna la lunghezza del piano.
3. **double getHeight();** Ritorna l' altezza del piano.
4. **C getOrigin();** Ritorna la coordinata di origine del piano, dove si intersecano gli assi.
5. **C getHome();** Ritorna la coordinata del punto centrale del piano.
6. **Queue<Line<C>> getLines();** Ritorna l' insieme delle linee presenti nel piano in modo FIFO (first-in-first-out).
7. **int getNumLines();** Ritorna il numero totale delle linee nel piano.
8. **RGBColor getBackgroundColor();** Ritorna il colore dello sfondo del piano.
9. **void setBackgroundColor(RGBColor backgroundColor);** Imposta un certo colore RGB dello sfondo piano.
10. **void addLine(Line<C> line);** Aggiunge una linea al piano.
11. **int getNumPoints();** Ritorna il numero di punti appartenenti alle linee presenti nel piano.
12. **Cursor<C, GenericDirection> getCursor();** Ritorna il cursore che si trova attualmente nel piano. GenericDirection per la direzione del cursore (l'angolo).
13. **Queue<ClosedArea<Line<C>>> getClosedAreas();** Ritorna l' insieme di tutte le aree chiuse presenti nel piano in ordine FIFO.
14. **int getNumClosedAreas();** Ritorna il numero totale di aree chiuse nel piano.
15. **boolean isPartOfPlane(C point);** Verifica se un punto specificato appartiene o meno al piano.

16. **C getDownLeftPoint();** Ritorna le coordinate del punto sull' angolo del in basso a sinistra piano.
17. **C getDownRightPoint();** Ritorna le coordinate del punto sull' angolo in basso a destra del piano.
18. **C getUpLeftPoint()**; Ritorna le coordinate del punto sull' angolo in alto a sinistra del piano.
19. **C getUpRightPoint();** Ritorna le coordinate del punto sull'angolo in alto a destra del piano.
20. **Map<C, Integer> getPoints();** Ritorna la mappa con tutti i punti (e le loro caratteristiche).
21. **Graph<C> getGraph();** Restituisce il grafo dei punti nel piano.
22. **PlaneUpdateSupport<C> getPlaneUpdateSupport();** Permette di gestire i cambiamenti delle proprietà del piano. C è il tipo parametrico per le coordinate del punto nel piano.
23. **default C getCursorPosition()** Ritorna la posizione attuale del cursore nel piano con le coordinate X e Y
24. **default Set<Line<C>> getLinesAt(C point)** Ritorna l' insieme delle linee aventi come estremo un punto passato come parametro.
25. **addPlaneUpdateListener(PlaneListener<Point<Double>> listener);**
    Aggiunge il listener per gli aggiornamenti al piano.
26. **removePlaneUpdateListener(PlaneListener<Point<Double>> listener);**
    Rimuove il listener per gli aggiornamenti al piano.

PlaneListener

Interfaccia contenente metodi necessari far interagire la view e il model.
Metodi:

1. **void MovedCursor(C point);** Metodo che indica che il cursore si è spostato in un punto specificato.
2. **void GeneratedLine(Line<C> line);** Metodo che indica che è stata generata la linea specificata.
3. **void GeneratedArea(ClosedArea<Line<C>> area);** Metodo che indica che è stata generata un' area chiusa.
4. **void ScreenColor(RGBColor color);** Metodo che indica che il colore del piano è cambiato.
5. **void ScreenCleaned();** Metodo che indica che è stato pulito tutto ciò che si vede a schermo.

Point

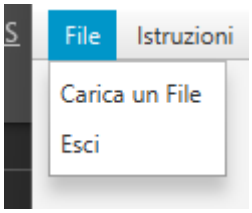
Interfaccia contenente metodi necessari a rappresentare un generico punto nel piano.
Metodi:

1. **static <N extends Number> CartesianPoint cartesianPoint(N x, N y) :** crea un punto cartesiano in base alle coordinate specificate.
2. **N getX();** Ritorna l' ascissa di un punto.
3. **N getY():** Ritorna l' ordinata di un punto.

Funzionamento del programma ed esecuzione

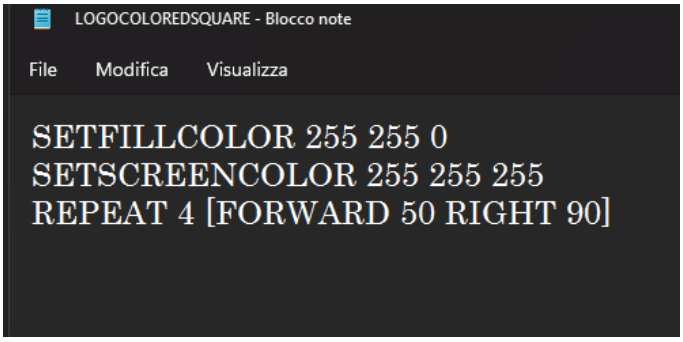
Per avviare il programma (MainFX , quindi la grafica), basterà eseguire il comando gradle build e application run.

Una volta avviata l'interfaccia grafica, premendo su File e scegliendo un file contenente le istruzioni necessarie a creare un'immagine, verranno mostrati i seguenti comandi:

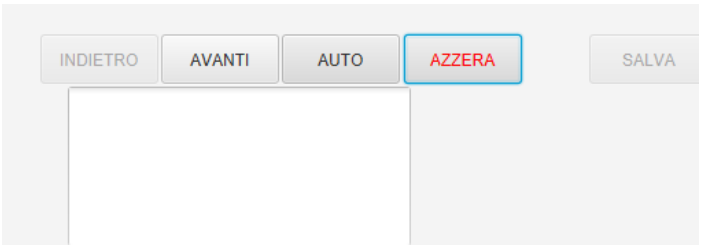


Premendo carica File, si potrà scegliere il file .txt contenente le informazioni in linguaggio Logo.

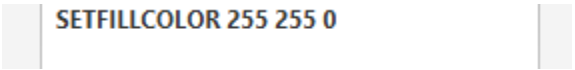
Un esempio di file è questo:



Una volta caricato il file, verrà mostrata la seguente schermata:



Premendo "avanti", verranno mostrate le istruzioni step by step (contemporaneamente, verranno interpretate ed eseguite), mentre premendo auto verranno interpretate ed eseguite tutte insieme.



Il risultato finale è questo:

Dopo aver completato la lettura del file, il pulsante "salva" sarà abilitato per salvare, sotto forma di file di testo, il contenuto del piano di disegno.

Premendo “salva”, verrà salvato un file con le informazioni riguardanti il disegno:

1) Area del disegno (base per altezza), seguita dai codici dei colori RGB che corrispondono al colore di riempimento dell’area.
2) Il poligono, descritto da N che corrispondono al numero totale di linee (tratti) che lo formano, il suo colore di riempimento e la lista di linee che formano il poligono, le coordinate dei punti, il colore del punto e lo spessore del tratto della linea.

Esempi di comandi

Ad esempio, per disegnare un cerchio colorato, si possono scrivere i seguenti comandi:

SETFILLCOLOR 255 102 255 #per impostare il colore di riempimento dell’area di rosa

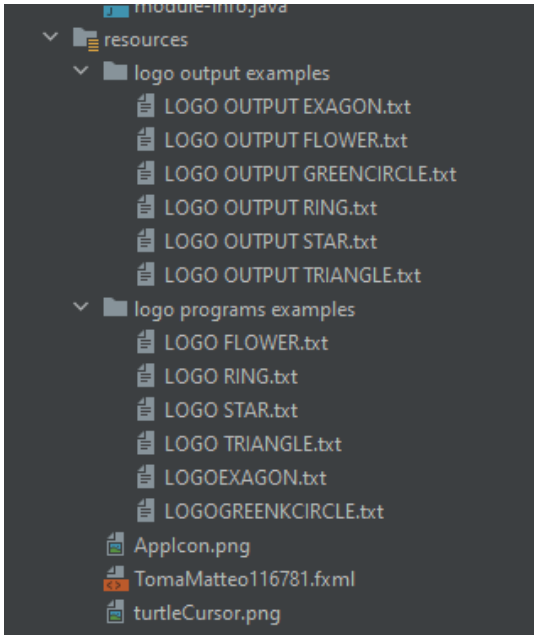
SETPENCOLOR 255 255 255 #per impostare il colore del tratto della penna di bianco

SETSCREENCOLOR 153 153 255 #per impostare il colore di sfondo di azzurro.

LEFT 90 #per ruotare il cursore verso sinistra (di 90 gradi)

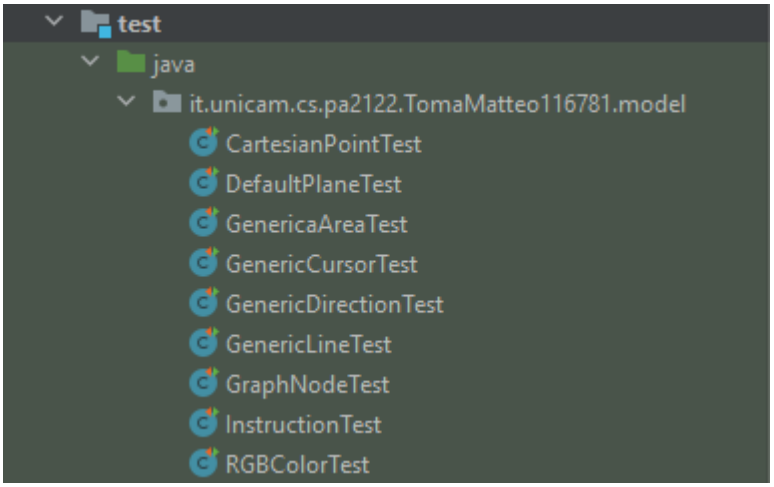
REPEAT 360 [FORWARD 2 RIGHT 1] #per ripetere l’operazione 360 volte, andando in avanti di due e una volta a destra. Infatti, aumentando, ad esempio, il forward a 3, aumenterà il raggio del cerchio.

File presenti nella cartella “resources”



1) I programmi Logo di esempio per testare l'applicazione
2) L'output dei programmi logo generati, contenenti le informazioni sui disegni.
3) L'icona dell'app
4) Il file fxml che rappresenta la grafica.
5) L'icona del cursore.

File presenti nella cartella "test"



Descrizione delle classi usate per rappresentare i programmi Logo

Anche se alcune classi possono essere convertite in record (perché contengono dati immutabili), per favorire l'estendibilità del codice si è scelto di mantenerle come classi.

Per rappresentare i programmi Logo sono state create le Classi **Default Controller**, che serve per controllare tutta la logica dell'applicazione. Implementa i metodi dell'interfaccia Controller.

Nel Default Controller, attraverso i metodi *loadInstructions,* che permette di caricare un file con istruzioni Logo.

Nella classe **Default Controller si trovano:**

Il metodo getAllInstructions per prendere tutte le istruzioni lette*, getConfigurationInstructions* per ottenere le istruzioni di configurazione, *createLogoFile* per creare il file LOGO contenente le informazioni del disegno attuale nel piano.

Infine, il metodo *execute,* per eseguire tutti i comandi letti da un file.

Descrizione delle classi usate per rappresentare il disegno prodotto da un programma Logo

Per rappresentare i disegni prodotti da un programma con istruzioni in Logo sono state usate le classi:

Cartesian Point: Rappresenta il punto cartesiano nel piano di disegno.

Generic Area: Classe per implementare un' area chiusa dell' interfaccia *ClosedArea*.
La classe è parametrizzata da una linea, e da un punto di tipo Double per aumentare la precisione.

In questo modo, è possibile rappresentare una semplice area tramite linee chiuse, formate da coppie di punti rappresentati attraverso Double per aumentare la precisione.

Generic Cursor: Classe che implementa un cursore, con le coordinate de punto in cui il cursore si trova, e una direzione, data da un angolo che deve essere in un range dì 0-360. (gradi)

Generic Direction: Classe usata per implementare i metodi per la direzione del cursore, data dall'angolo e nell'intervallo [0,360].

Generic Line: Classe necessaria a implementare una linea. Serve per poter creare una linea ogni volta che si riposiziona il cursore. Un insieme di linee può formare un' area chiusa, quindi potenzialmente un poligono.

RGBColor: Per rappresentare i colori (dell'interno di un'area, e/o di una linea) , è un Record: è una forma ristretta di una classe. È ideale per "supporti per dati semplici", classi che contengono dati che non devono essere modificati e solo i metodi più fondamentali come costruttori e metodi accessori.

Graph: Questa classe permette di generare cicli nel grafo per poter trovare le aree chiuse.

Possibili Estensioni:

La struttura del progetto è aperta a modifiche.

Per aggiungere nuovi comandi, ad esempio, basterà creare un metodo nell'interfaccia *Instruction* del package *Model*. Poi si aggiungerà tra i casi del metodo *Repeat*.

L'uso dei tipi generici inoltre permette di poter personalizzare i tipi di dati in base alle necessità.

Non utilizzando mai la clausola "final class" nelle classi (tranne in Undirected Graph, poiché è l'algoritmo base per trovare i grafi), si è evitato che una classe non possa essere più estesa, quindi il progetto è aperto ad estensioni di gerarchie riguardanti altre forme geometriche.

Non sono stati utilizzati nemmeno gli enum, solo interfacce, in questo modo il progetto è più aperto a modifiche.

Inoltre, se ad esempio si volesse ottenere dei disegni di più di due dimensioni, basterebbe aggiungere un altro parametro all'interfaccia *Point*.