Designer

ISTITUTO TECNICO TECNOLOGICO STATALE
"S. Fedi - E. Fermi"

Guglielmo Bartelloni, Francesco Bellezza

24 Dicembre, 2017
10 marzo 2018

4IB
Luigi Vestri e Davide Caramelli

Laboratorio di Informatica

Indice

1 Scopo dell'esercitazione

Lo scopo dell'esercitazione e quella di realizzare un programma che consenta di disegnare figure geometriche e grafici di funzioni permettendo di salvare le figure sul un file di testo.

2 Cenni Storici

2.1 Ereditarieta'

L'ereditarietà è una relazione di tipo is-A, dove una superclasse mette a disposizione di una sottoclasse i suoi metodi e attributi non privati (a eccezione del costruttore).

In Java, per estendere una superclasse e per creare quindi la sottoclasse, si utilizza la parole chiave extends accanto al nome della sottoclasse e accanto alla parola extends si inserisce il nome della superclasse.

Nel caso delle interfacce, ovvero particolari classi che al loro interno contengono solamente metodi astratti, si utilizza la parola chiave implements. Fiorenzo, Giorgio e Ivan, (*Corso di Informatica*)

2.2 Polimorfismo

Il polimorfismo è una tecnica che consente di utilizzare metodi polimorfici, ovvero metodi che hanno lo stesso nome, ma implementazioni diverse. Esistono due tipi di polimorfismi principali: per overloading e per overriding.

Nel polimorfismo per overloading si opera a un livello locale, ovvero all'interno di una classe.

Il metodo polimorfico in questo caso deve:

- Avere lo stesso nome degli altri metodi polimorfici.
- Avere numero di parametri diversi o avere tipi di parametri diversi o avere ordine di parametri diversi rispetto agli altri metodi polimorfici.
- Può avere un tipo di ritorno diverso se i due punti qua sopra sono rispettati.

Nel polimorfismo per overriding si opera a un livello di superclassi e sottoclassi.

Nelle sottoclassi viene ridefinito il metodo presente nelle superclassi.

Il metodo polimorfico in questo caso deve:

- Avere la stessa signature del metodo della superclasse.
- Avere lo stesso tipo di ritorno della superclasse o un sottotipo del tipo di ritorno della superclasse (stesso discorso vale per i parametri).
- I metodi private non vengono ereditati alla classe figlia, quindi non si può effettuare l'overriding del metodo.
- Le clausole come native, strictfp possono essere incluse nel metodo della classe figlia.
- Un metodo statico può essere solo adombrato e non sovrascritto.

Wikipedia, (*Polimorfismo*)

2.3 Classi Astratte

Le classi astratte sono direttamente connesse con il concetto di ereditarietá. Infatti queste sono classi che sono create appositamente all'unico scopo di essere estese, quindi servono per avere caratteristiche comuni alle classi che la esendono, logicamente non possono neanche essere istanziate.

Per dichiarare una classe astratta si puó ricorrere alla parola chiave *abstract* accanto al nome della classe, oppure se si dichiara dei metodi astratti, la classe diventa automaticamente astratta. HTML.it, (*Classi Astratte*)

2.4 Swing e AWT

L'AWT é una libreria grafica che permette di creare componenti grafici per i programmi Java, é stato sostituito dalla Swing per il fatto che utilizzava un primitive del sistema operativo su cui il programma girava. Questo non rispettava il motto di Java ("write once, run everywhere") perché, usando le primitive di sistema, il programma risultava avere un aspetto diverso a seconda della piattaforma su girava.

La Swing é un'estensione dell'AWT con componenti scritti interamente in Java, i programmi hanno cosí uguale aspetto su ogni piattaforma. Nonostante l'avvento della Swing, l'AWT viene utilizzata per la gestione degli eventi e per altre classi come la *Dimension*.

2.5 Disegno in Java

Per il disegno di figure geometriche in Java inizialmente veniva utilizzata la classe Canvas, questa peró é stata dichiarata deprecata. Adesso, anche con l'avvento del framework Swing, viene utilizzato un metodo chiamato *paintComponent(Graphics g)* a cui viene fatto l'overriding nella classe in cui si vuole disegnare, questo metodo é comune a tutti i JComponent. Per il disegno vero e proprio, il componente *Graphics g* contiene dei metodi appositi come il *DrawLine(...)* per disegnare una linea.

3 Analisi Funzionale

3.1 Ipotesi Risolutiva

Il programma deve riuscire a disegnare figure elementari su un pannello. Per farlo viene utilizzato l'oggetto graphics andando a reimplementare la classe

```
public void paintComponent(Graphics g);
```

3.2 Funzionalitá del programma

Le funzionalitá che si devono implementare sono:

1. Salvare su file
2. Caricare da file
3. Disegnare figure(quadrato, arco, linea, ellisse, rettangolo, grafico, cerchio e ellisse)

4. Disegnare Font(grandezza, tipo, stile)

L‘interfaccia si presenta in questo modo:

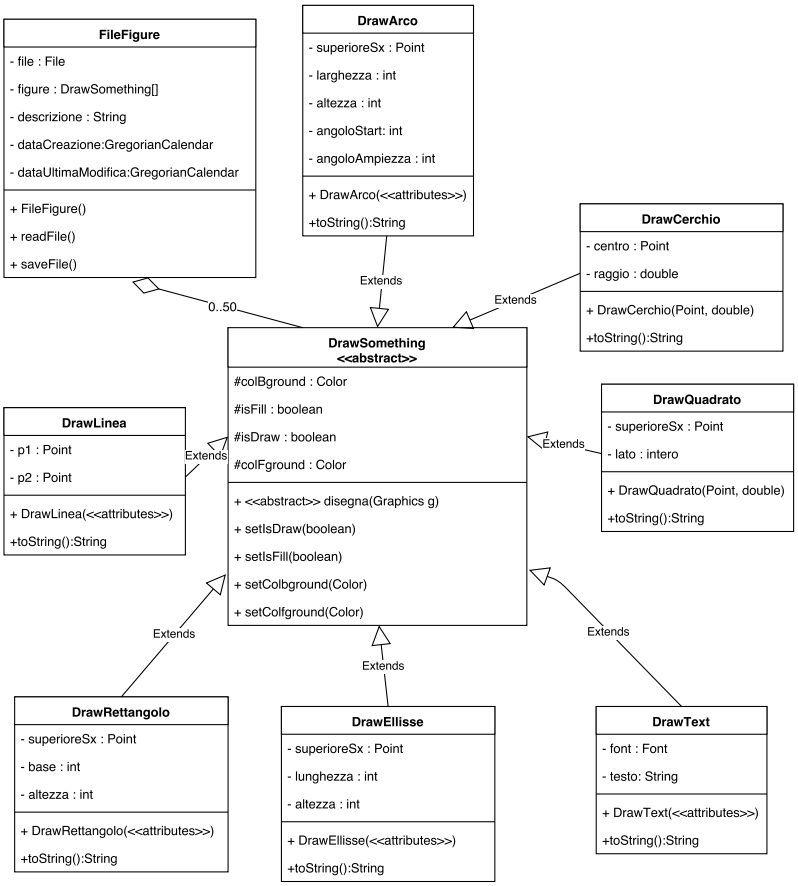
4 Analisi Tecnica

4.1 Scomposizione Top-Down

La scomposizione Top-Down del Main é la seguente:

```
INIZIO
   imposta le dimensioni del frame
   inizializza le icone
   crea i componenti della GUI
   aggiungi i listener
   imposta la finestra non ridimensionabile
FINE
```

4.2 UML



4.3 Descrizione Classi

Le classi del programma Designer sono divise per package a seconda del loro utilizzo all'interno del programma stesso, essi sono:

- *File*: contiene le classi che si occupano della gestione dei file.
- *GUI*: contiene le classi che si occupano della gestione della grafica insieme alle classi che rappresentano le varie Dialog e la tabella.
- *Grafico*: contiene la classe Funzione che rappresenta una funzione matematica.
- *UtilitiesForDrawing*: contiene le classi che si occupano del disegno delle figure geometriche compreso anche il grafico della funzione e il font.
- *VectorAndException*: contiene la classe che rappresenta il vettore di figure insieme alle eccezioni ad esso collegate.
- *icons*: contiene tutte le immagini delle icone utilizzate.
- *lookAndFeel*: contiene i .jar del Look And Feel utilizzato.

Di seguito verranno descritte solo le classi principali.

4.3.1 MainFrame

La classe *MainFrame* è la classe che si occupa di gestire tutti gli oggetti del programma, infatti rappresenta il *controller* e la *vista* del Design Pattern MVC. Le sue funzioni principali sono state esplicitate in 4.1. La classe è stata scomposta in molti sottometodi per facilitare la lettura e la scrittura. Viene implementato il MouseListener registrato sul *pnlDisegno* e nel momento in cui l'utente preme sul pannello viene registrata la posizione (iniziale) del mouse poi, al rilascio, viene registrata la posizione (finale) del mouse inizializzando e inserendo nel vettoreFigure la figura precedentemente selezionata dall'utente.

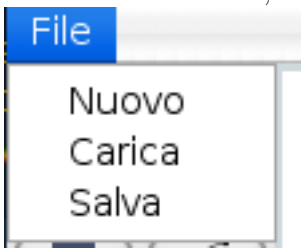
Per il Font abbiamo utlizzato tre pulsanti nel pannello di destra insieme a una *JComboBox* con la lista dei font di sistema e una *JTextField* per inserire la parola/frase da disegnare.

Per il disegno dell'arco e del grafico ci siamo serviti di due Dialog che permettono di ottenere i dati necessari al disegno dei due elementi.

Al rilascio del pulsante del mouse vengono fatti vari controlli delle coordinate per farsì che la figura sia disegnata correttamente: se per esempio l'utente trascina da in alto a sinistra fino in basso a sinistra dovranno essere scambiate le coordinate come segue

```
pInf.setLocation(pIniziale.x, pFinale.y);
pSup.setLocation(pFinale.x, pIniziale.y);
pIniziale.setLocation(pSup);
pFinale.setLocation(pInf);
```

Al suo interno è presente una inner class, che rappresenta una *JMenuBar* e di gestire la creazione di un file, il caricamento e il salvataggio. Il menú si presenta in questo modo:



setSelectedBotton() è un particolare metodo per riuscire a lasciare selezionati i pulsanti delle figure e dello stile dei font, semplicemente deseleziona tutti i pulsanti che non corrispondono a quello premuto.

4.3.2 DrawSomething

La classe *DrawSomething* é una classe astratta che dichiara alcuni metodi per impostare i colori e un metodo astratto per essere riscritto nelle classi figlie. É pensata all'unico scopo di essere estesa dalle altre classi *draw* che implementeranno il metodo *toString()* e il metodo *draw(Graphics g)* come mostrato in figura 4.2. Gli attributi di questa classe sono i seguenti:

```
protected Color colBground;
protected Color colFground;
```

4.3.3 FileFigure

La classe *FileFigure* é la classe che si occupa di gestire di gestire il ripristino e il salvataggio delle figure nel file di testo. Un file di testo, per essere letto, deve avere al suo interno:

1. Descrizione
2. Data Creazione
3. Data Ultima Modifica
4. Colore sfondo
5. Figure da disegnare

Al suo interno sono presenti i seguenti metodi:

Il metodo *setWrittable()* serve per creare una istanza della classe PrintWriter che scrive sul file. Se la funzione ritorna un valore uguale a false, allora c'è stato un qualche errore dovuto all'Input/Output.

Il metodo *setReadable()* serve per creare una istanza della classe BufferedReader che legge il file. Se la funzione ritorna un valore uguale a false, allora c'è stato un qualche errore dovuto all'Input/Output: ad esempio il file non è stato trovato.

Il metodo *readFile()* si occupa di leggere il file di testo e salvare le figure nell'array.

Il metodo *saveFile()* si occupa di salvare l'array di figure all'interno del file di testo. Per il salvataggio delle figure viene utilizzato il metodo *toString()* di ogni figura.

4.3.4 Funzione

La classe *Funzione* é una classe che rappresenta una funzione matematica contiene infatti l'insieme di coefficienti insieme al termine noto all'interno di un'array di *double*. Gli attributi di questa classe sono i seguenti:

```
private double[] coefficienti
```

Contiene un metodo per ottenere la y data una x come parametro:

```
public double getRisultato(double x);
```

e un metodo per aggiungere i coefficienti:

```
public boolean add(double coefficiente, int potenzaCoef);
```

4.3.5 Vector

La classe *Vector* é la classe che rappresenta un vettore di dimensione iniziale di 100 Object. Contiene i metodi per rendere dinamico il vettore, infatti ogni volta che verrá aggiunto un elemento ci sará un controllo che verificherá che il numero di elementi non superi la grandezza del vettore, se viene superata allora verrá lanciata l'eccezione *IsFullException* che deve essere catturata per poter richiamare il metodo *riorganizza()*. Il metodo *riorganizza()* si occupa di ricreare l'array con dimensione doppia rispetto a quella iniziale e di immettere gli elementi al suo interno. É presente anche un metodo per rimuovere gli elementi data una posizione facendo uno shift verso sinistra:

```
public void removeElement(int pos);
```

I parametri della classe sono:

```
protected Object[] items;
protected int index;
public static int DIMENSIONE;
```

4.3.6 VectorDraw

Classe che estende la classe *Vector* in modo da contenere elementi di tipo *DrawSomething*. Contiene due metodi che servono uno, per ritornare l'array di DrawSomething, l'altro, per ritornare la singola figure presente nella posizione data come parametro.

4.3.7 JColorChoicePane

Pannello che ha la funzione di visualizzare la scelta dei colori contiene un *ItemListener* per salvare il colore una volto che viene scelto.

4.3.8 JButtonColor

La classe *JButtonColor* é stata creata appositamente perché il programma utilizza un Look and Feel diverso da quello di default. Il problema che abbiamo riscontrato con questo, é che i componenti non possono essere cambiati di colore al di fuori del *paintComponent(Graphics g)*. Abbiamo dovuto quindi, per la scelta del colore, creare dei bottoni che disegnino un rettangolo al loro interno con il colore selezionato dall'utente nel *JColorChoicePane*.

4.3.9 JPanelDrawing

Rappresenta il pannello per il disegno, ovvero il foglio bianco su cui disegnare le figure. Presenta il metodo *paintComponent(Graphics g)* che disegna tutte le figure del vettore attraverso un *for* e richiamando il metodo *draw(Graphics g)* di ogni figura.

4.3.10 JDialogArc

Rappresenta una *Dialog* che si occupa di [rendere i dati necessari per costruire l'arco

4.3.11 JDialogGrafico

Rappresenta una *Dialog* che si occupa di prendere i dati necessari per la creazione del grafico. Permette di inserire i dati con un formato a matematico uno alla volta del tipo: $2x^2$
in questo

```
2x2
```

4.3.12 Utilities

É la classe che contiene metodi statici per il calcolo delle distanze tra due punti, i metodi sono i seguenti:

```
public static int getDistanzaX(Punto p1, Punto p2);
public static int getDistanzaY(Punto p1, Punto p2);
public static int getDistanza(Punto p1, Punto p2);
```

5 Test Data Set/Debug

Riferimenti bibliografici

Fiorenzo, Formichi, Meini Giorgio e Venuti Ivan. *Corso di Informatica*. Zanichelli, 2017.

HTML.it. *Classi Astratte*. http://www.html.it/pag/51820/classi-astratte-in-java/.

Wikipedia. *Polimorfismo*. https://it.wikipedia.org/wiki/Polimorfismo_(informatica).