TECNICHE E STRUMENTI PER LA VISUALIZZAZIONE DI DATI SCIENTIFICI

Giulio Botturi
Davide Quaglia

Contenuto

1. Visualizzazione grafica di serie di dati

Spesso è necessario visualizzare graficamente delle serie di dati, memorizzate e organizzate in basi di dati o file testuali, per una più chiara e immediata interpretazione.

1.1. Tecniche e strumenti

A questo scopo introduciamo ora alcune tecniche e strumenti utili ad automatizzare questa fase di analisi dei dati, ciascuno con le sue potenzialità, vantaggi e svantaggi. La scelta della tecnica dipende dal risultato che si vuole ottenere e dal grado di integrazione che si vuole avere a livello applicativo.

Principalmente i dati grezzi vengono rappresentati come tradizionali grafici per punti sul piano cartesiano, istogrammi, diagrammi circolari, a bolle e molti altri in parte derivati dai precedenti.

Gli strumenti illustrati in questa esercitazione sono tre: il software Gnuplot, la libreria Java JFreeChart e le primitive grafiche fornite da Java stesso.

In particolare Gnuplot non consente la realizzazione diretta di diagrammi circolari (grafico a torta). Per fare ciò è necessario realizzare complessi script che disegnano tutta la struttura della torta tramite funzioni matematiche oppure ricorrere, come segnalato nelle FAQ online sul sito del progetto Gnuplot, a una libreria C che sfrutta le GNU plotutils.

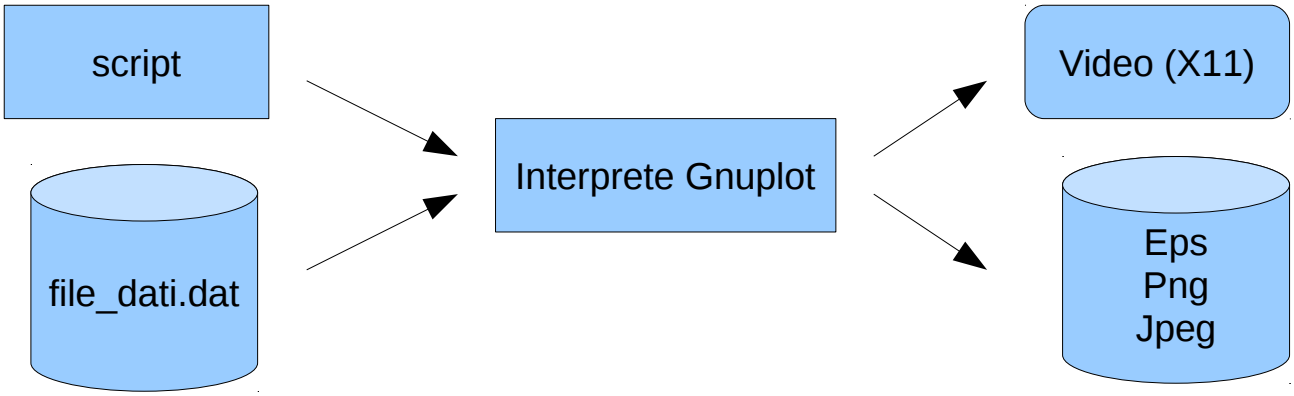
Vedremo nei due capitoli successivi, e in particolare in quello relativo alle primitive grafiche Java, come realizzare manualmente grafici in un linguaggio di programmazione. Questo ci consentirà di avere la massima libertà di personalizzazione, pagando il prezzo di dover gestire a livello geometrico e implementativo molti dettagli che strumenti automatici quali Gnuplot nascondono.

2. Gnuplot

Gnuplot è un versatile programma, distribuito gratuitamente in rete, per la realizzazione di grafici di funzioni matematiche e la rappresentazione grafica di dati grezzi.

Ai fini dell'esercitazione ci limitiamo ad analizzare la gestione dei dati grezzi memorizzati in semplici file di testo.

Lo schema di funzionamento di Gnuplot è il seguente:



L'interprete esegue i comandi di uno script che elabora i dati presenti in un file di testo. L'output viene mostrato a video oppure può essere dirottato su file nei più comuni formati di immagini.

Per avviare l'interprete, posizioniamoci nella directory in cui teniamo i file dati e script degli esempi e digitiamo il comando `gnuplot`.
I comandi possono essere direttamente impartiti all'interprete attraverso un prompt oppure essere letti da un file di script (ad esempio filescript.gp) con il comando

```
load “filescript.gp”
```

oppure è possibile direttamente passare il nome dello script in linea di comando alla chiamata di `gnuplot`

```
gnuplot filescript.gp
```

Per uscire dall'interprete si scrive il comando `quit`.

Nei seguenti paragrafi vediamo alcuni esempi di file di dati con i relativi script gnuplot per la loro visualizzazione. Tali sorgenti di dati verranno poi riutilizzati nei due capitoli successivi, per essere elaborati con altre tecniche e strumenti.

2.1. Coppie XY sul piano

Vediamo per prima cosa un semplice esempio in cui si vuole rendere graficamente l'andamento di due serie di dati affini, quali le temperature massime e minime giornaliere del mese di gennaio.
Tali informazioni sono memorizzate nel file *esempio1.dat* nel seguente formato:

```
# Temperature del mese di gennaio 2010
# giorno min max
01 -2 7
02 -2 6
# commento
03 0 7
04 -1 6
05 1 8
...
...
```

I valori sono memorizzati riga per riga, separati da uno o più spazi (o caratteri di tabulazione).

Abbiamo dunque bisogno di rappresentare su un piano in due dimensioni due serie di coppie di valori, (giorni – temperatura minima) e (giorno – temperatura massima).

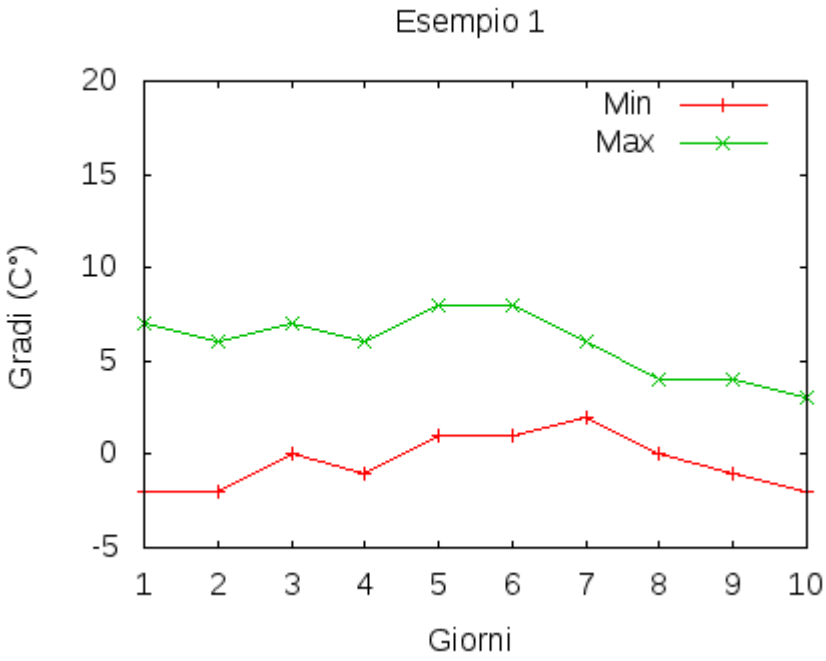
Analizziamo riga per riga lo script utilizzato per visualizzarli (*script1.gp)*

```
# questo è un commento
set title "Esempio 1"
set xlabel 'Giorni'
set ylabel 'Gradi (C°)'
set yrange [-5:20]
set style data linespoints
plot 'esempio1.dat' using 1:2 title 'Min', 'esempio1.dat' using 1:3 title 'Max'
pause -1
```

La prima riga indica il titolo da mostrare in cima al grafico.
La seconda e terza riga impostano le etichette da dare rispettivamente all'asse delle ascisse e delle ordinate.
La quarta riga imposta manualmente il range dei valori dell'asse delle ordinate. In mancanza di tale riga Gnuplot imposta automaticamente tale range a partire dal valore minimo visualizzato fino al massimo riscontrato. Nel nostro caso ampliare tale range ci consente una migliore resa grafica delle due spezzate che Gnuplot andrà a disegnare.

Le ultime due righe contengono i comandi più importanti.
In particolare il comando `set style data […]` permette di definire il tipo di grafico che si vuole ottenere. Molti sono gli stili possibili, qui ci limiteremo a vedere `linespoints e histogram`.
Il comando `plot` infine legge i dati grezzi dal file `esempio1.dat`, con la direttiva `using` indichiamo di utilizzare le colonne di dati 1 e 2, mentre con `title` diamo il nome `Min` alla curva che verrà mostrata sul grafico.
In questo esempio ricordiamo che sono due le serie di dati da mostrare insieme sullo stesso grafico, quindi dopo la virgola seguono le direttive per leggere i dati, questa volta dalle colonne 1 e 3, assegnando il nome `Max` alla curva generata.

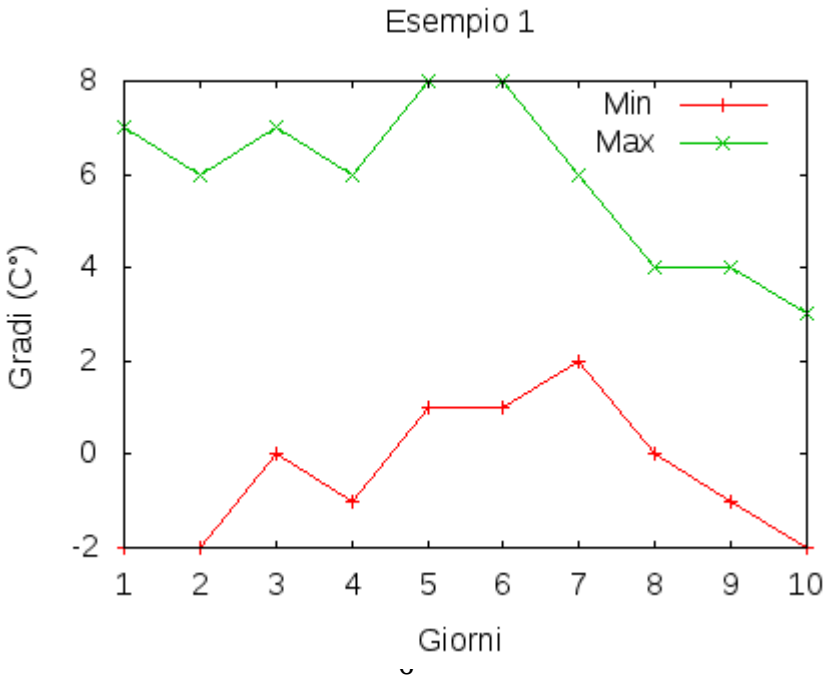
Di seguito il grafico che gnuplot genera dall'esecuzione dello script



Si noti il range ampliato sull'asse Y e la legenda in alto a destra.

Senza il comando `set yrange [-5:20]` il grafico risulterebbe così

```
# questo è un commento
set title "Esempio 1"
set xlabel 'Giorni'
set ylabel 'Gradi (C°)'
#set yrange [-5:20]
set style data linespoints
plot 'esempio1.dat' using 1:2 title 'Min',
     'esempio1.dat' using 1:3 title 'Max'
pause -1
```



2.2. Istogramma

Come secondo esempio vediamo la realizzazione di un istogramma a partire dal file di dati *esempio2.dat* strutturato nel seguento modo:

```
# Mese Entrate Uscite
Gennaio 4500 4350
Febbraio 4325 4100
Marzo 5200 4450
Aprile 4900 4600
Maggio 4620 4580
Giugno 4150 4450
```

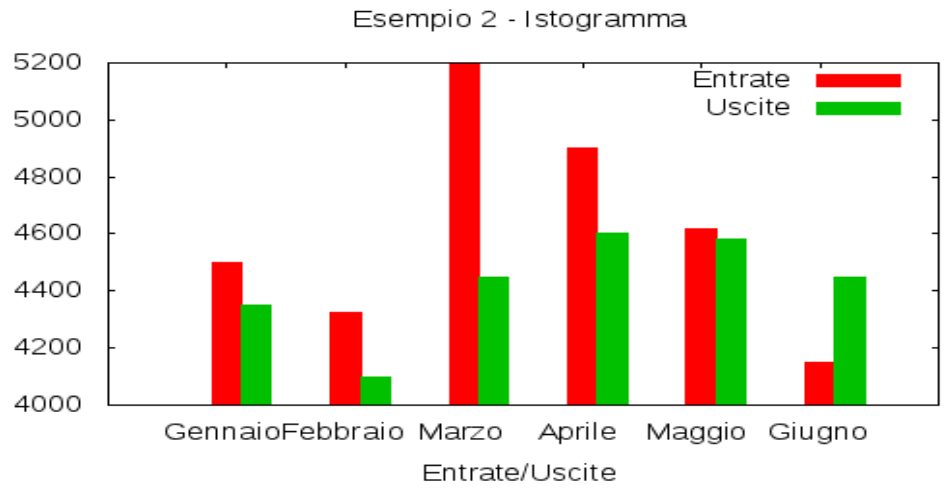
Sono memorizzate delle ipotetiche entrate e uscite (in euro) relative ai diversi mesi dell'anno e le vogliamo mettere a confronto graficamente. Anche in questo caso entrate e uscite rappresentano due serie di dati distinte, (mese – entrate) e (mese – uscite).

Analizziamo riga per riga lo script utilizzato (*script2.gp)*

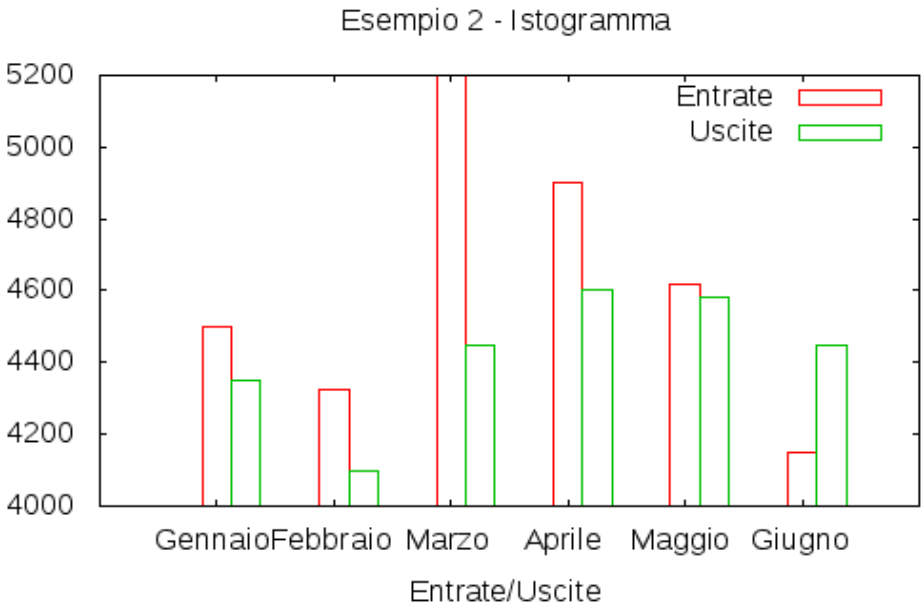
```
set title "Esempio 2 - Istogramma"
set xlabel 'Entrate/Uscite'
set style data histogram
set style fill solid
plot 'esempio2.dat' using 2:xticlabels(1) title 'Entrate', 'esempio2.dat' using
3:xticlabels(1) title 'Uscite'
pause -1
```

Come per l'esempio precedente, con la prima e seconda riga definiamo il titolo del grafico e l'etichetta da assegnare all'asse delle ascisse.
Il comando set style data histogram della terza riga imposta lo stile istogramma, specificando nella quarta riga la resa grafica da dare alle colonne (colorate al loro interno e non soltanto riquadrate come di default).
Il comando plot nuovamente legge due serie di dati dal file di testo, la prima dalle colonne 2 e 1, la seconda dalla colonne 3 e 1, rispettivamente con il nome 'Entrate' e 'Uscite'.
In questo caso i valori presente sull'asse X sono i mesi dell'anno, quindi vogliamo che sia indicato il nome testuale. Per questo motivo è necessario specializzare la direttiva `using` istruendola a piazzare i dati della colonna 1 come etichette di testo sotto l'asse X.

Di seguito il grafico che gnuplot genera dall'esecuzione dello script.



Senza il comando *`set style fill solid`* le colonne apparirebbero con la resa grafica di default



2.3. Gestire l'output degli script

L'output di default per gli script gnuplot è, in ambiente Linux, il terminale grafico X11 cioè una finestra a video.
E' possibile in alternativa redirigere tale output su file al fine di esportare i grafici sotto forma di immagini in vari formati per essere successivamente riutilizzati in altri ambiti. Per fare questo occorre inserire all'inizio dello script:

```
set terminal png size 450, 350
set output "esempio1bis.png"
```

Il primo comando indica a gnuplot che il terminale di output non è più X11 (valore di default), ma che deve provvedere a generare un file immagine in formato png (keyword alternative sono `jpeg gif eps`) di dimensione 450x350 pixel. Nella seconda riga indichiamo invece il nome del file di output in cui salvare tale immagine.
Di seguito è possibile richiamare gli altri comandi visti prima.

2.4. Esercizi

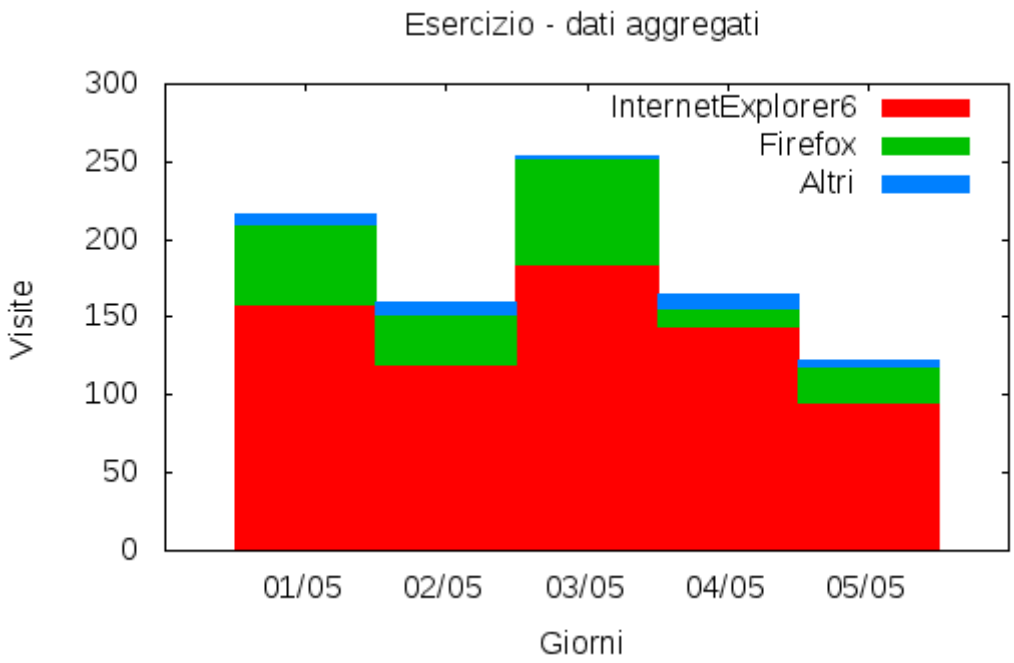
Per esercizio si chiede di realizzare un particolare tipo di istogramma utile per visualizzare dati aggregati.

Il file dati di esempio è il seguente (*esercizio.dat*):

```
# giorno IE6 firefox altri
01/05 158 52 6
02/05 120 32 8
03/05 184 68 2
04/05 144 12 9
05/05 95 23 4
```

in cui è memorizzato il numero di accessi giornaliero, ad un ipotetico sito web, suddiviso per la tipologia di browser.
Aggiungere la direttiva `set style histogram rowstacked` che costruisce delle barre verticali che aggregano i dati dei tre valori relativi al numero di visite. L'altezza della barra raggiunge la somma dei tre valori.
Aggiungere al grafico le etichette col nome dei due assi, “Giorni” per l'asse X e “Visite” per l'asse Y, oltre alla legenda relativa ai tre valori aggregati. Trattere i valori sull'asse X come stringhe testuali.

Il grafico che si dovrebbe ottenere è il seguente:



Per ottenere un aiuto su un comando è sufficiente scrivere sulla prompt dei comandi di Gnuplot

```
help nomecomando
```

Per ulteriori dettagli riguardo lo style histogram si rimanda al paragrafo 34 della documentazione ufficiale gnuplot, reperibile in formato pdf al link *http://www.gnuplot.info/documentation.html*.

3. Libreria JFreeChart

Per creare grafici di varia natura in modo automatico all'interno di nostri programmi occorre usare una libreria all'interno del linguaggio di programmazione usato, ad esempio la libreria java **JFreeChart**.

Tale libreria, all'apparenza complessa per il gran numero di classi che fornisce, lavora nel seguente modo:

- le classi del package *org.jfree.data* gestiscono le serie di dati da rappresentare
- la classe ChartFactory del package *org.jfree.chart* fornisce i metodi statici per generare grafici di varia natura e ritornano tutti oggetti del tipo JFreeChart
- con il metodo getPlot() di un oggetto JFreeChart si può agire su numerose proprietà del grafico, quali il range di valori degli assi
- infine da un oggetto JfreeChart si ottengono oggetti ChartFrame, ChartPanel che ereditano dalle classi java.awt.Frame e java.awt.Panel. Tali oggetti si possono maneggiare tramite le primitive grafiche java fornite dal packate *java.awt*

Vediamo in dettaglio alcuni esempi dei grafici più comuni che possiamo realizzare.

3.1. Grafici a coordinate XY sul piano

Vogliamo rappresentare ad esempio le temperature massime e minime di ciascun giorno del mese di gennaio 2010.

```
import org.jfree.data.xy.*;
import org.jfree.chart.*;
import org.jfree.chart.axis.*;
import org.jfree.chart.plot.*;

// creo la serie di dati per le temperature massime
XYSeries serieMax = new XYSeries("Massima");
// aggiungo le coppie di valori XY
serieMax.add(1,7);
serieMax.add(2,6);
serieMax.add(3,6);
serieMax.add(4,4);
serieMax.add(5,5);
serieMax.add(6,8);
// ....

// creo la serie di dati per le temperature minime
XYSeries serieMin = new XYSeries("Minima");
serieMin.add(1,0);
serieMin.add(2,-2);
serieMin.add(3,-2);
serieMin.add(4,-3);
serieMin.add(5,-1);
serieMin.add(6,1);


// creo il dataset (con una o piè serie di dati), che rappresenta
// la collezione dei dati da visualizzare
XYSeriesCollection dataSet = new XYSeriesCollection();
dataSet.addSeries(serieMax);
dataSet.addSeries(serieMin);

// creo l'oggetto JFreeChart col metodo statico dedicato
// al il tipo di grafico in esame
JFreeChart grafo1 = ChartFactory.createXYLineChart("Grafo1",
                                    "Giorni di Gennaio",
                                    "Gradi (C°)",
                                    dataSet, PlotOrientation.VERTICAL,
                                    true,
                                    false,
                                    false );

// modifico il range di valori sull'asse delle ordinate
XYPlot plot = grafo1.getXYPlot();
ValueAxis asseY = plot.getRangeAxis();
asseY.setRange(-5, 20);

// creo un oggetto ChartFrame per visualizzare finalmente il grafico
// ChartFrame eredita da java.awt.Frame ed è quindi trattabile in seguito
// con le normali primitive grafiche fornite dal package java.awt
ChartFrame frame1 = new ChartFrame("Temperature di Gennaio 2010", grafo1);
// scelgo la dimensione in pixel della finestra
frame1.setSize(400,350);
// e la visualizzo
frame1.setVisible(true);
```

Il range di valori sugli assi viene gestito in automatico in base ai valori minimi e massimi rilevati. In alcuni casi è però necessario intervenire su tali valori per una visualizzazione più accurata.

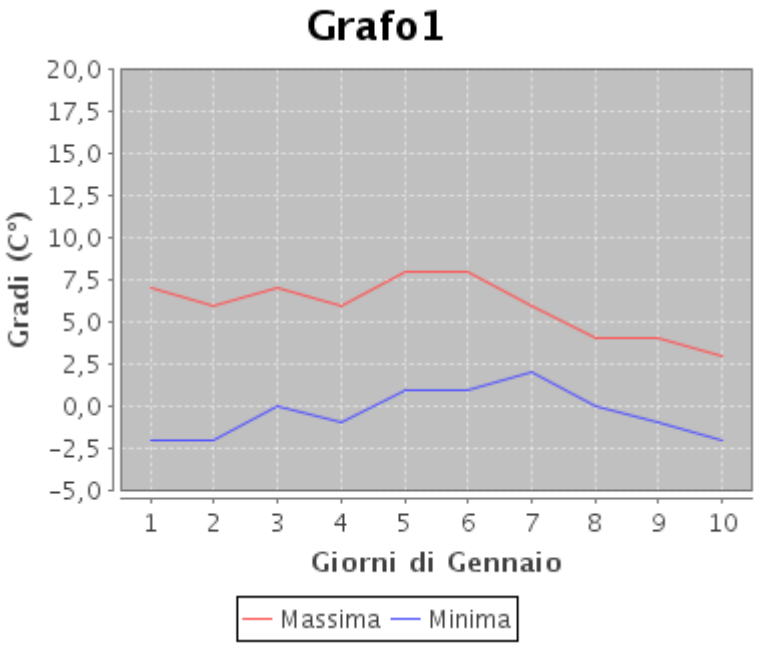
Consideriamo ora la situazione in cui le temperature del mese di gennaio siano memorizzate in un file di testo, chiamato `esempio1.dat`, con il seguente formato (ogni colonna è separata da una tabulazione):

```
# Temperature del mese di gennaio 2010
# giorno	min	max
01	-2	7
02	-2	6
03	0	7
04	-1	6
...
```

Ecco una funzione che ritorna il dataset caricato dal file.

```
static XYSeriesCollection caricaDati1()
{
	String s, a[];

	// creo la serie di dati per le temperature massime
	XYSeries serieMax = new XYSeries("Massima");
	// creo la serie di dati per le temperature minime
	XYSeries serieMin = new XYSeries("Minima");

	try
	{
		FileReader in = new FileReader("temp.dat");
		BufferedReader reader = new BufferedReader(in);

		while (reader.ready())
		{
			s = reader.readLine();
			if (s.length() > 0 && s.charAt(0) != '#')
			{
				// la funzione split suddivide la stringa s in
				// sottostringhe riconoscibili da un delimitatore
				// (in questo caso una tabulazione)
				a = s.split("\t");
				serieMin.add(Double.parseDouble(a[0]),
						Double.parseDouble(a[1]));
				serieMax.add(Double.parseDouble(a[0]),
						Double.parseDouble(a[2]));
			}
		}

		reader.close();
		in.close();

	}
	catch (IOException ioe)
	{
		System.out.println(ioe.getMessage());
	}
	// creo il dataset (con una o più serie di dati), che rappresenta
	// la collezione dei dati da rappresentare
	XYSeriesCollection dataSet = new XYSeriesCollection();
	dataSet.addSeries(serieMax);
	dataSet.addSeries(serieMin);

	return dataSet;
}
```

Ecco il grafico che si ottiene

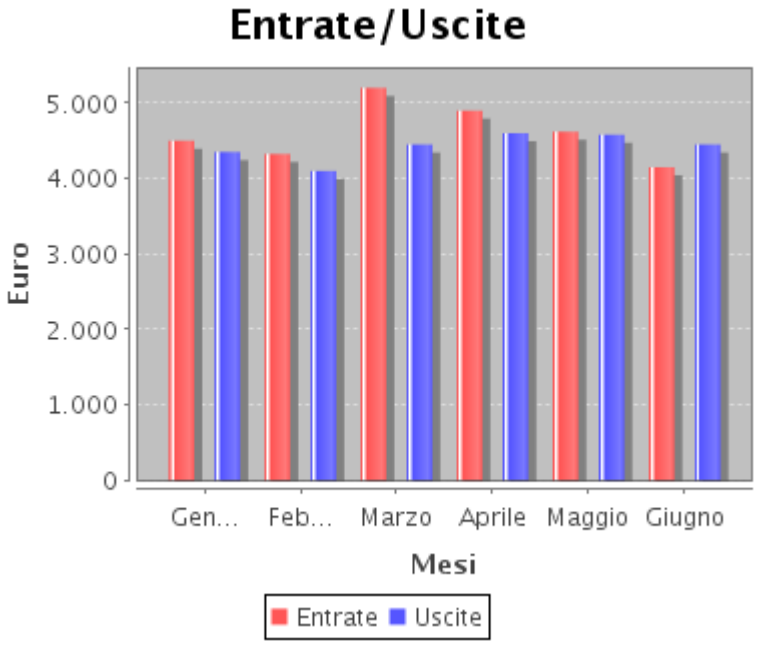


3.2. Istogrammi

Vogliamo ora mettere a confronto le entrare e le uscite di ogni mese dell'anno corrente. Tali informazioni sono memorizzate nel file di esempio *esempio2.dat* già visto con Gnuplot.
In questo caso torna utile un grafico a istogramma, realizzato in JFreeChart col dataset CategoryDataset.

```
// creo un CategoryDataset
DefaultCategoryDataset dataSet = caricaDati2();
// creo l'oggetto JFreeChart col metodo statico dedicato
// per il tipo di grafico in esame
JFreeChart grafo = ChartFactory.createBarChart("Entrate/Uscite",
                                               "Mesi",
                                               "Euro",
                                               dataSet,
                                               PlotOrientation.VERTICAL,
                                               true, false, false);

// creo un oggetto ChartFrame per visualizzare finalmente il grafico
// ChartFrame eredita da java.awt.Frame ed Ã¨ quindi trattabile in seguito
// con le normali primitive grafiche fornite dal package java.awt
ChartFrame frame1 = new ChartFrame("Esempio3", grafo);
// scelgo la dimensione in pixel della finestra
frame1.setSize(400,350);
// e la visualizzo
frame1.setVisible(true);
```

Questo è il grafico che si ottiene



La funzione di lettura dei dati dal file *esempio2.dat* necessita di alcune modifiche rispetto a quella vista in *esempio1.dat*.

```
static DefaultCategoryDataset caricaDati2()
{
	String s, a[];
	DefaultCategoryDataset dataSet = new DefaultCategoryDataset();
	try
	{
		FileReader in = new FileReader("esempio2.dat");
		BufferedReader reader = new BufferedReader(in);

		while (reader.ready())
		{
			s = reader.readLine();
			if (s.length() > 0 && s.charAt(0) != '#')
			{
				// la funzione split suddivide la stringa s in
				// sottostringhe riconoscibili da un delimitatore
				// (in questo caso una tabulazione)
				a = s.split("\t");
				// a[0] mese
				// a[1] entrate
				// a[2] uscite
				dataSet.addValue(Double.parseDouble(a[1]), "Entrate", a[0]);
				dataSet.addValue(Double.parseDouble(a[2]), "Uscite", a[0]);
			}
		}

		reader.close();
		in.close();
	}
	catch (IOException ioe)
	{
		System.out.println(ioe.getMessage());
	}
	return dataSet;
}
```

In particolare analizziamo il metodo addValue della classe DefaultCategoryDataset.

```
public void addValue(double value,
                     java.lang.Comparable rowKey,
                     java.lang.Comparable columnKey)
```

Nel nostro esempio ogni “categoria” è rappresentata da un mese, e per ciascuna mostriamo due barre a indicare il valori rispettivamente delle entrate e delle uscite.
JFreeChart intende il dataset rappresentato come una tabella in cui ogni colonna sta per una categoria e ogni riga sta per uno dei valori associati alla categoria (una barra dell'istogramma).

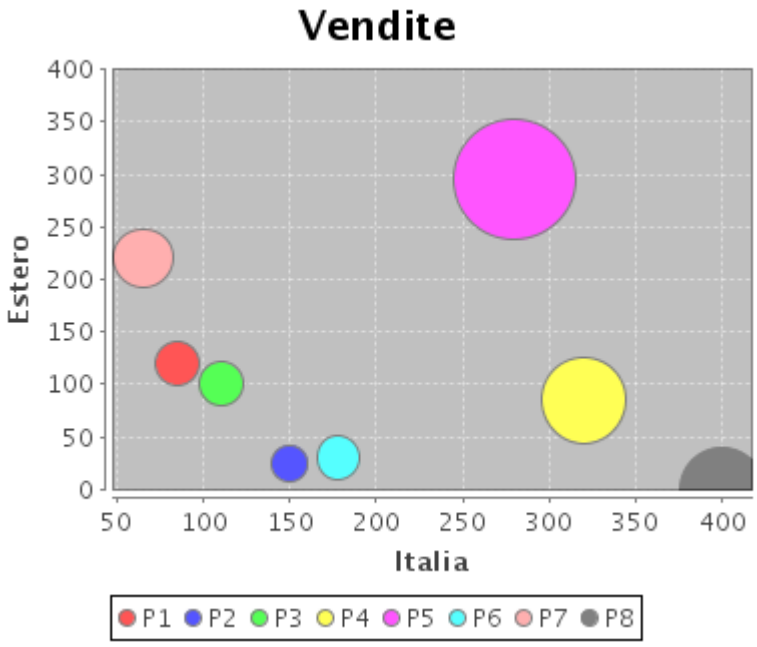
	Gennaio	Febbraio	Marzo	Aprile	Maggio
Entrate					
Uscite					

3.3. Grafici a bolle

Come ultimo esempio analizziamo il grafico a bolle che è caratterizzato da elementi definiti su tre parametri. Risulta piuttosto utile per analizzare la relazioni tra i parametri giocando, ad esempio, sulla dimensione delle bolle, o il colore delle stesse.
JFreeChart gestisce questo tipo di grafico con il dataset DefaultXYZDataset e il metodo createBubbleChart.

```
DefaultXYZDataset dataSet = caricaDati3();
JFreeChart grafo = ChartFactory.createBubbleChart("Vendite",
                                                   "Italia",
                                                   "Estero",
                                                   dataSet,
                                                   PlotOrientation.VERTICAL,
                                                   true, false, false);
XYPlot plot = grafo.getXYPlot();

// imposto il range di valori sull'asse delle ordinate sull'interno [0, 400]
ValueAxis asseY = plot.getRangeAxis(0);
asseY.setRange(0, 400);

ChartFrame frame = new ChartFrame("Esempio3", grafo);
frame.setSize(400,350);
frame.setVisible(true);
```

Questo il grafico che ne esce:



Attenzione che il valore della dimensione delle bolle disegnate sul grafo segue il range dei valori sugli assi.
Nell'esempio normalizziamo tale valore in un range pari a 1/5 (scelta arbitraria) quello dell'asse Y, in modo che le bolle non risultino enormi e soprapposte, rendendo impossibile la loro interpretazione.

```
static DefaultXYZDataset caricaDati3()
{
	DefaultXYZDataset dataSet = new DefaultXYZDataset();

	String s, a[];
	double d[][];
	double tot, max = 0, t;

	try
	{
		FileReader in = new FileReader("esempio3.dat");
		BufferedReader reader = new BufferedReader(in);

		// scorro tutto il file per determinare il valore massimo del terzo campo
		// (Serve per la successiva normalizzazone dei valori)
		while (reader.ready())
		{
			s = reader.readLine();
			if (s.length() > 0 && s.charAt(0) != '#')
			{
				a = s.split("\t");
				tot = Double.parseDouble(a[1]) + Double.parseDouble(a[2]);
				if (tot > max) max = tot;
			}
		}

		reader.close(); in.close();

		// inizio una nuova scansione del file per caricare i dati
		in = new FileReader("esempio3.dat");
		reader = new BufferedReader(in);

		while (reader.ready())
		{
			s = reader.readLine();
			if (s.length() > 0 && s.charAt(0) != '#')
			{
				d = new double[3][1];
				// la funzione split suddivide la stringa s in
				// sottostringhe riconoscibili da un delimitatore
				// (in questo caso una tabulazione)
				a = s.split("\t");
				d[0][0] = Double.parseDouble(a[1]);
				d[1][0] = Double.parseDouble(a[2]);
				// sommo le vendite di italia e estero
				tot = d[0][0] + d[1][0];
		/*
		 * Normalizzo la dimensione delle bolle nel range [0-M] con M che vale 1/5 del valore

		 * rilevato tra i dati
		 * Questa operazione va compiuta perchè la dimensione delle bolle di default
		 * è proporzionale ai valori sugli assi.
		 * A noi interessa la dimensione relativa delle bolle per l'immediata comparazione
		 */

				t = (max/5) * tot / max;
				d[2][0] = t;
				dataSet.addSeries(a[0], d);
			}
		}

		reader.close();
		in.close();
	}
	catch (IOException ioe)
	{
		System.out.println(ioe.getMessage());
	}
	return dataSet;
}
```

3.4. Esportazione dei grafici ottenuti

La libreria in esame offre la possibilità di salvare come file immagine i grafici che abbiamo generato.
La classe *org.jfree.chart.ChartUtilities* mette a disposizione alcuni metodi statici per l'esportazione di oggetti JFreeChart nei più comuni formati di immagini.

Vediamo brevemente un esempio

```
DefaultXYZDataset dataSet = caricaDati3();
JFreeChart grafo = ChartFactory.createBubbleChart("Vendite",
                                                   "Italia",
                                                   "Estero",
                                                   dataSet,
                                                   PlotOrientation.VERTICAL,
                                                   true, false, false);
XYPlot plot = grafo.getXYPlot();

ValueAxis asseY = plot.getRangeAxis(0);
asseY.setRange(0, 400);

File f = new File ("esempio3.jpeg");
try
{
        ChartUtilities.saveChartAsJPEG(f, grafo, 400, 350);
}
catch (IOException ioe)
{
        System.err.println(ioe.getMessage());
}
```

In questo caso non abbiamo creato un oggetto ChartFrame da visualizzare a video ma si è creato un file *esempio3.jpeg* nel quale esportare in formato compresso JPEG il grafico.

3.5. Esercizi

Un'altro tipo di grafico gestibile da JFreeChart è il diagramma circolare (comunemente indicato con grafico a torta).
Le classi e i metodi necessari a gestire tale rappresentazione sono i seguenti:

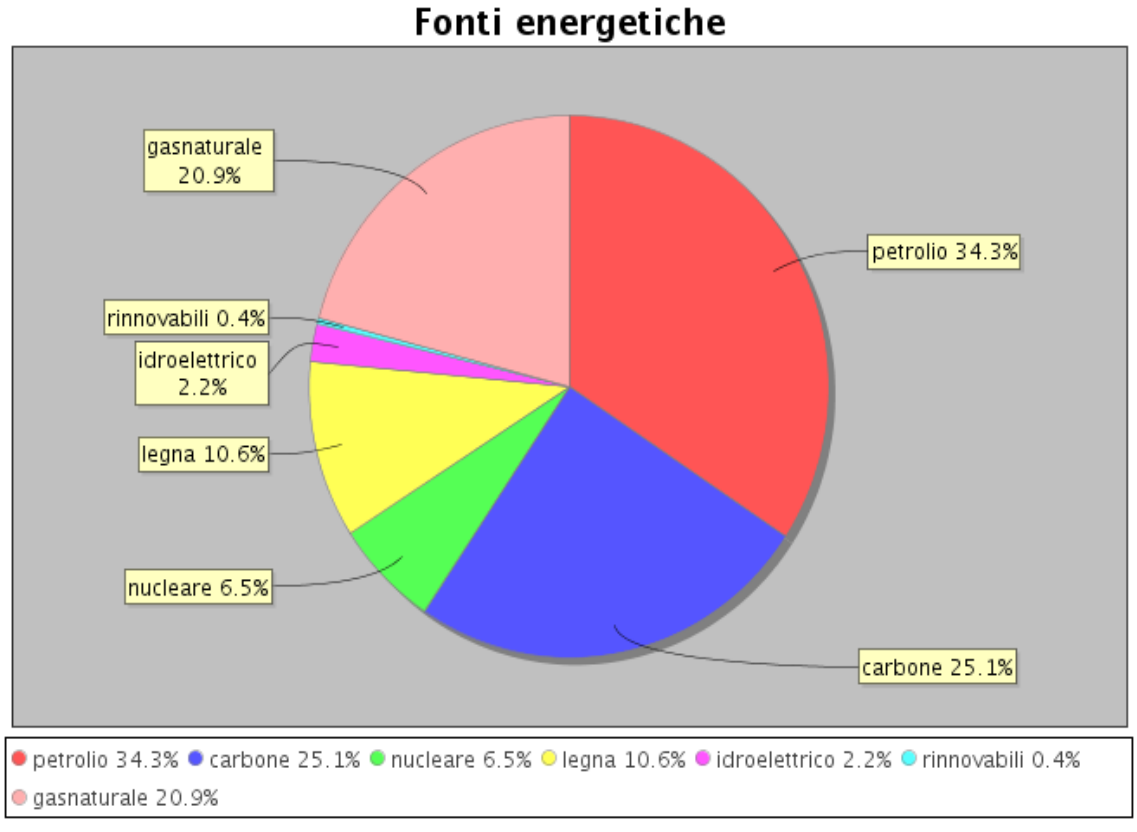
- org.jfree.data.general.DefaultPieDataset per il dataset
- public static JFreeChart createPieChart(java.lang.String title, PieDataset dataset, boolean legend, boolean tooltips, boolean urls) per generare il grafico

Si rimanda al javadoc per i dettagli del metodo setValue(..) della classe DefaultPieDataset per l'inserimento dei dati.

L'esercizio richiede di creare il grafico, secondo lo schema visto per gli esempi precedenti, partendo dal file di testo qui sotto riportato (i valori si intendono espressi in forma percentuale):

```
# Consumi energetici mondiali, per fonte
# nell'anno 2004
petrolio    34.3
carbone     25.1
nucleare    6.5
legna 10.6
idroelettrico     2.2
rinnovabili 0.4
gasnaturale 20.9
```

Di seguito il grafico che ne dovrebbe uscire:



3.6. Note sulla compilazione e riferimenti alla documentazione

Per la compilazione dei sorgenti, è necessario ricordare alcune opzioni del compilatore *javac* per quanto riguarda il classpath e i file di libreria .jar.
Le classi JFreeChart che ci interessano sono presenti nella cartella lib/ in due file .jar, *jcommon-1.0.16.jar* e *jfreechart-1.0.13.jar*

```
drwxr-xr-x 2 giulio giulio 4,0K 2010-04-02 10:41 demoGrafici
-rw-r--r-- 1 giulio giulio  132 2010-03-31 10:10 esempio1.dat
-rw-r--r-- 1 giulio giulio  127 2010-03-31 18:48 esempio2.dat
-rw-r--r-- 1 giulio giulio  117 2010-04-01 14:06 esempio3.dat
-rw-r--r-- 1 giulio giulio  161 2010-04-02 10:17 esercizio.dat
drwxr-xr-x 2 giulio giulio 4,0K 2010-04-02 10:31 lib
```

Occorre posizionarsi nella cartella che contiene i file .dat.

Col seguente comando è possibile compilare i sorgenti degli esempi:

```
javac -classpath lib/jcommon-1.0.16.jar:lib/jfreechart-1.0.13.jar
demoGrafici/Grafici.java
```

Con l'opzione -classpath vengono indicati i due file .jar che contengono le classi della libreria (non è sufficiente indicare la sola directory che li contiene!).

Col seguente comando è possibile eseguire gli esempi:

```
java -classpath ./:lib/jcommon-1.0.16.jar:lib/jfreechart-1.0.13.jar
demoGrafici/Grafici
```

Per eseguire il nostro codice è necessario aggiungere all'opzione -classpath anche la directory contente le classi compilate dai sorgenti.

Riferimenti e documentazione ufficiale:

http://www.jfree.org/
http://www.jfree.org/jfreechart/api/javadoc/index.html

4. Java e le primitive grafiche

Questo capito introduce le primitive che Java mette a disposizione del programmatore per la gestione della grafica elementare.
Queste primitive consentono da un lato di implementare GUI (Graphical User Interface) interattive con l'utente, dall'altro di giocare con i costrutti geometrici fondamentali e realizzare ogni sorta di disegno e rappresentazione grafica.
Vedremo dunque come implementare a livello applicativo la visualizzazione di un semplice grafico a partire sempre da un file di dati, come visto nei capitoli precedenti.
Con questo esempio vengono illustrate alcune problematiche che si incontrano in questo tipo di realizzazioni. In particolare la manipolazione dei sistemi di riferimento e la normalizzazione dei dati da rappresentare, in quanto devono essere scalati in relazione alle dimensioni dell'area grafica in cui si voglio mostrare.

4.1. Panoramica sul package AWT

Di seguito faremo riferimento all'Abstract Window Toolkit, la libreria contenente le classi e le interfacce fondamentali per la creazione di elementi grafici, inserita nelle API standard di Java.
I package fondamentali in cui si articola sono i seguenti:

- java.awt package principale che contiene le classi di tutti i componenti che possiamo utilizzare nella GUI, quali ad esempio Frame, Panel, Button, Label, Checkbox, Canvas, Menu, MenuBar, TextArea e molti altri
- java.awt.event fornisce le classi per la gestione degli “eventi”, ovvero il sistema che awt utilizza per passare il controllo al programmatore in seguito ad azioni avvenute sui componenti, come la pressione di un bottone, il passaggio del mouse su un componente, l'apertura di una finestra

La classe che rappresenta una “finestra” di interazione grafica con l'utente è la `java.awt.Frame`. Essa presenta fondamentalmente la classica barra del titolo e un bordo.
Per la disposizione dei componenti al suo interno sono definibili diversi layout.

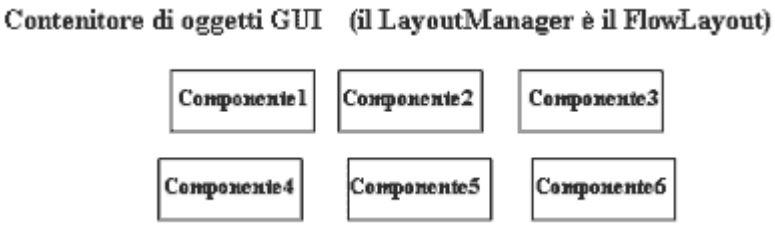
Ecco una breve descrizioni dei più comuni layout:

- BorderLayout dispone cinque oggetti secondo i diversi punti cardinali. La loro posizione si sceglie con le cinque costanti della classe borderLayout:

```
borderLayout.NORTH
borderLayout.SOUTH
borderLayout.EAST
borderLayout.WEST
borderLayout.CENTER
```

- GridLayout consente di disporre i componenti come fossero posizionati su una griglia. Tutti i componenti assumeranno la stessa dimensione. Al costruttore della classe viene passato come parametro la dimensionee della griglia, inteso come numero di righe e di colonne

- FlowLayout permette di disporre i componenti da sinistra verso destra su una stessa linea.



Quando è necessario realizzare complesse GUI con numerosi componenti, gestire tutto ciò manualmente risulta difficile. In tali situazioni è maggiormente produttivo ricorrere a tool progettati per disegnare in modo visuale l'aspetto dell'interfaccia.

4.2. Esempio dimostrativo

Scopo di questo esempio è presentare l'implementazione di un semplice grafico per punti su un piano XY rappresentanti, come visto nel primo esempio dei capitoli precedenti, le temperature rilevate giornalmente.
Iniziamo ad analizzare il codice sorgente dell'esempio, per apprendere i metodi basilari per creare la nostra GUI e disporvi alcuni componenti.

```
import java.awt.*;
import java.awt.event.*;

public class Esempio extends Frame implements ActionListener
{
	private String fileDati;

	// componenti presenti sul frame
	MenuBar barraMenu;
	Menu menuFile;
	MenuItem fileEsci;
	PianoXY piano;
	Panel pannelloBase;
	Label labelStato;

	public Esempio(String s)
	{
		fileDati = s;

		inizializzaComponenti();
		inizializzaAscoltatoriEventi();

		setSize(650, 550);
		setLocationRelativeTo(null);
		setVisible(true);
	}
	...
	...
```

La classe estende per prima cosa Frame e implementa l'interfaccia ActionListener che definisce i metodi necessari ad intercettare alcuni eventi generati dall'utente tramite il click del mouse.
Tra le dichiarazioni di variabili troviamo subito alcuni componenti che disporremo successivamente all'interno della finestra.
In particolare l'oggetto `PianoXY` è una classe realizzata ad hoc per ospitare il piano di lavoro su cui disegneremo il grafico. Essa estende il componente `java.awt.Canvas`, che rappresenta un'area vuota rettangolare dentro la quale l'applicazione può disegnare elementi grafici primitivi.
Abbiamo poi un oggetto Panel, per il quale definiamo in seguito il layout per disporre alcuni componenti con ordine.
Nel costruttore si chiamano due metodi per aggiungere i componenti al frame e catturare gli eventi.
L'ultima istruzione rende visibile la finestra.

```
	private void inizializzaComponenti()
	{
		...
		piano = new PianoXY(fileDati);
		pannelloBase = new Panel(new BorderLayout());
		labelStato = new Label("Stato");
		...

		add(pannelloBase);
		pannelloBase.add(piano, BorderLayout.CENTER);
		pannelloBase.add(labelStato, BorderLayout.SOUTH);
	}
```

Con queste istruzioni viene creato il pannello base a cui vengono aggiunti il componente PianoXY

al centro e nella parte bassa il componente etichetta di testo per visualizzare alcuni messaggi.

4.3. Sistema di coordinate e primitive grafiche

Il sistema di coordinate è l'aspetto più delicato da gestire, in quanto l'origine del sistema, in tutti i componenti di awt, è fissato nell'angolo in alto a sinistra del componente stesso.
Questo rende necessario implementare alcuni metodi per convertire le coordinate poste nell'usuale sistema di riferimento (con l'origine in basso a sinistra) nel sistema awt.
Un'ulteriore complicazione nasce dal fatto che in questo esempio l'origine degli assi del grafico che vogliamo realizzare risulterà spostato dal bordo della finestra di un certo numero di pixel per lasciare spazio sottostante alle etichette dei valori.

```
public class PianoXY extends Canvas
{

	// distanza dell'origine degli assi dal bordo del canvas
	static final int OFFSET_ASSE_X = 40;
	static final int OFFSET_ASSE_Y = 40;
	// distanza delle etichette dagli assi
	static final int OFFSET_LABEL_ASSE_X = 20;
	static final int OFFSET_LABEL_ASSE_Y = 30;

	private AsseX asseX;
	private AsseY asseY;

	public PianoXY(String f)
	{
		this.setBackground(Color.green);
		caricaDati(f);
	}

	public void paint(Graphics g)
	{
		// metodo paint che viene invocato ogni qualvolta sai necessario
		// aggiornare le componenti grafiche dell'oggetto
		disegnaAssi();
		disegnaEtichette();
		disegnaValori();
	}
	...
	...
```

Nelle dichiarazioni troviamo alcune variabili che definiscono il numero di pixel di offset degli assi rispetto al bordo, oltre a quanti pixel distanziare le etichette dei valori dagli assi.
Notare l'importanza del metodo `paint(Graphics g)` (sovrascritto), che viene invocato ogni qualvolta sia necessario, da parte del sistema grafico AWT, ridisegnare i componenti. (Questo avviene ad esempio quando viene ridimensionata la finestra o nel momento in cui è visualizzata)
In tale metodo dunque richiamiamo tre sottoprocedure, che disegnano rispettivamente gli assi del grafico, le etichette dei valori sugli assi e i punti all'interno dell'area del grafico.

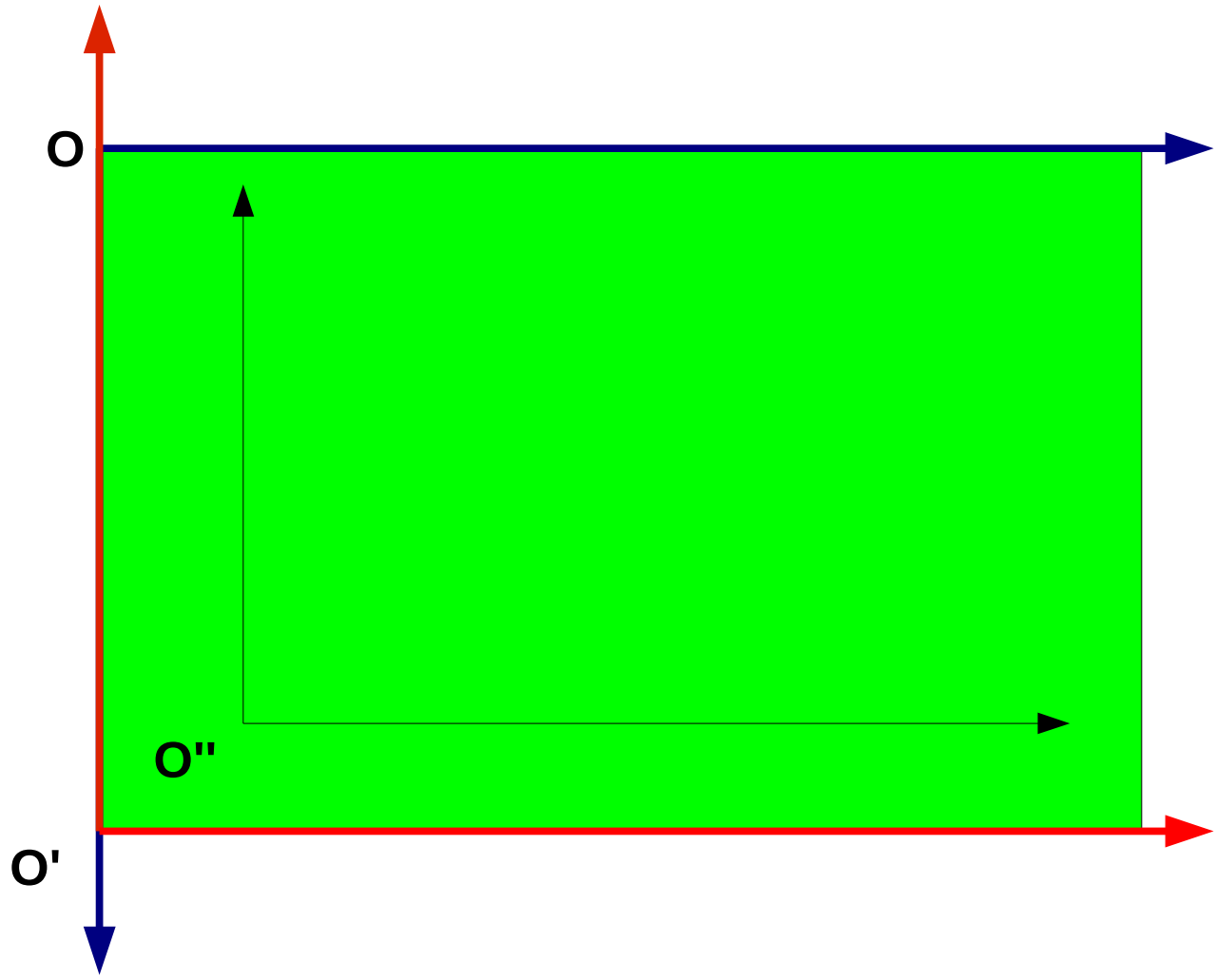
Prima di entrare nei dettagli è bene soffermarsi sull'importante metodo per la conversione delle coordinate.

```
	private Point convertiPlot2Canvas(Point p, Point o)
	{
		int x = p.x + o.x;
		int y = this.getHeight() - (p.y + o.y);

		return new Point(x,y);
	}
```

La classe java.awt.Point rappresenta un punto in uno spazio a due dimensioni con le sue coordinate.
Questo metodo converte le coordinate di un punto, visto nel sistema di riferimento con origine nell'angolo in basso a sinistra, nel sistema di riferimento di awt ancorato nell'angolo in alto a

sinistra. Inoltre il parametro Point o fornisce l'informazione sull'offset dell'origine degli assi rispetto all'angolo in basso a sinistra come precedentemente spiegato.

La seguente illustrazione chiarisce meglio la relazione tra i tre sistemi di riferimento:



Nell'origine O è ancorato il sistema di riferimento di AWN. I O' l'usuale sistema di riferimento che siamo abituati a maneggiare e il O" il sistema di riferimento del grafico che dobbiamo realizzare all'interno dell'area di lavoro.
Dato un punto P(x',y') in O' la coordinata x in O resterà dunque invariata, mentre Y in O è data dall'altezza dell'area verde meno Y' in O'.
Se invece prendiamo un punto P(x", y") in O" applichiamo lo stesso calcolo ma tenendo conto dell'offset dell'origine O" rispetto a O'.
X diventerà X" più l'offset lungo X' e Y diventerà l'altezza dell'area verde meno Y" più l'offset lungo Y'.
Per esercizio provare ad eseguire alcuni cambi di coordinate per fare un po' di pratica ed evitare confusione in seguito.

Detto ciò passiamo ad esaminare i metodi che implementano le primitive grafiche e gli oggetti su cui si possono applicare.
Ogni componente awn ha un metodo getGraphics() che ritorna un oggetto di tipo Graphics che rappresenta la sua area grafica sulla quale è possibile disegnare.

Ecco alcuni dei metodi di tale classe:

- **drawLine**(int x1, int y1, int x2, int y2)
- **drawOval**(int x, int y, int width, int height)
- **drawRect**(int x, int y, int width, int height)
- **drawRoundRect**(int x, int y, int width, int height, int arcWidth, int arcHeight)
- **drawString**(String str, int x, int y)
- **fillOval**(int x, int y, int width, int height)
- **fillRect**(int x, int y, int width, int height)
- **setColor**(Color c)

E' possibile quindi disegnare line, cerchi, rettangoli, stringhe di testo e cambiare il colore degli elementi prima di disegnarli.
Per un elenco completo e maggiori dettagli sui parametri si rimanda al javadoc.

4.4. Implementare il grafico

Torniamo ad analizzare il codice dell'esempio per vedere come creare il grafico.
Con una procedura simile a quella già vista negli esempi in JFreeChart, carichiamo i dati dal file testuale nel quale sono memorizzati.

```
public void caricaDati(String fileDati)
{
	...
	...
	try
	{
		FileReader in = new FileReader(fileDati);
		BufferedReader reader = new BufferedReader(in);

		// con una prima scansione determino il range dei dati
		// da visualizzare (Serve per dimensionare gli assi X e Y)
		while (reader.ready())
		{
			s = reader.readLine();
			if (s.length() > 0 && s.charAt(0) != '#')
			{
				// la funzione split suddivide la stringa s in
				// sottostringhe riconoscibili da un delimitatore
				// (in questo caso una tabulazione)
				a = s.split("\t");
				// conto il numero di record nel file
				conta++;
				col1 = Integer.parseInt(a[0]); col2 = Integer.parseInt(a[1]);
				if (primaIter)
				{
					// inizializzo alcune variabili per determinare i
					// valori min e max
					maxX = col1; minX = col1;
					maxY = col2; minY = col2;
					primaIter = false;
				}
				else
				{
					if (col1 > maxX ) maxX = col1;
					if (col1 < minX) minX = col1;
					if (col2 > maxY) maxY = col2;
					if (col2 < minY) minY = col2;
				}
			}
		}
		reader.close();
		in.close();

		...
		...

		asseX = new AsseX(maxX, minX, mySerieX);
		asseY = new AsseY(maxY, minY, mySerieY);
```

E' necessario però memorizzare i valori minimi e massimi da rappresentare su ciascun asse, per poterli in seguito dimensione opportunamente.
Per questo definiamo due classi, AsseX e AsseY, che accolgono tre campi, due per i valori minimo e massimo e uno dato da un array contenente la serie dei dati per quell'asse.

Il passo successivo è disegnare gli assi con il metodo `disegnaAssi()`. Tale metodo deve disegnare due segmenti tra l'origine del sistema di riferimento del grafico (O" nella illustrazione precedente) e il bordo della finestra (fermandosi un po' prima).

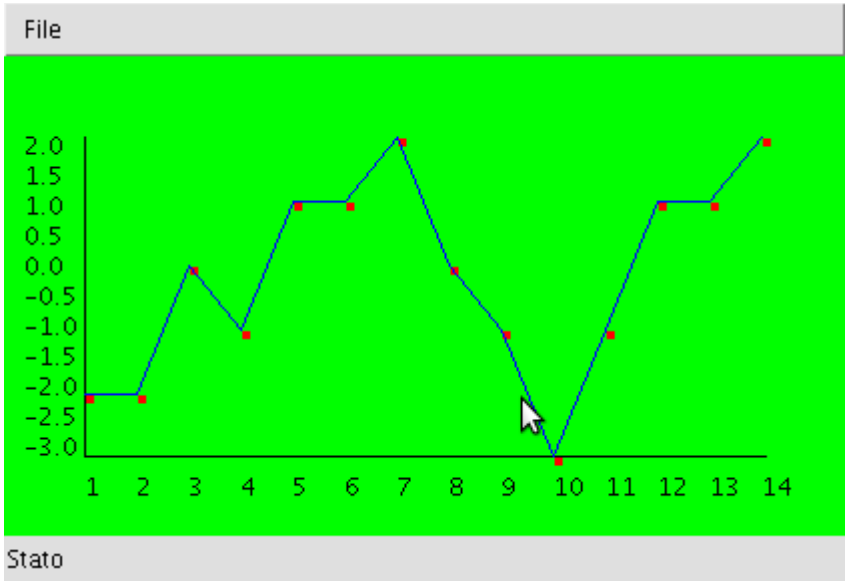
Successivamente, con il metodo `disegnaEtichette()`, devono essere piazzate le etichette dei valori sugli assi. Sotto l'asse delle ascisse il numero del giorno cui la temperatura fa riferimento, mentre sull'asse delle ordinate i valori delle temperature in un range che va dal minimo al massimo rilevato nel file (precedentemente memorizzato nella classe AsseY).
Questa operazione non è semplice perchè bisogna determinare le coordinate dei punti in cui andare a disegnare l'etichetta di testo. Per fare questo bisogna determinare la spaziatura da tenere tra una etichette e l'altra in base al numero di etichette e alla lunghezza dell'asse.
Sull'asse Y vanno scritti invece dei valori a intervalli regolari che vanno dal minimo al massimo. (Si è definito tale numero di valori arbitrariamente a 10).

Infine si disegnano i punti corrispondenti ai valori caricati dal file.
Per determinare la coordinata Y in pixel relativa all'area di disegno, è necessario normalizzare i valori del grafico sulla lunghezza dell'asse (in pixel)
Poniamo ad esempio di avere sull'asse Y valori da -2 e +5 gradi (minimo e massimo) e di voler disegnare il punto a temperatura 3 gradi. L'asse Y è lungo, ad esempio, 450pixel nell'area di lavoro, dunque il valore 3 dall'intervallo [-2:+5] va normalizzato nell'intervallo [0:450] così da trovare la giusta posizione rispetto all'asse.
Inoltre, dato che trattiamo anche valori negativi, il primo range di valori va traslato a partire da zero prima di eseguire la normalizzazione.
Il metodo che implementa la normalizzazione è molto semplice:

```
// normalizza il valore v (compreso tra 0 e valueMax) nell'intervallo
// [0 : rangeMax]
// Per valori negativi operare uno shift degli stessi nell'intervallo di partenza
// [0 : valueMax + shift]
private double normalizza(double rangeMax, double valueMax, double v)
{
	return v * (rangeMax/valueMax);
}
```

Per i dettagli delle tre fasi appena descritte si veda il codice sorgente e i numerosi commenti che spiegano in dettaglio ogni operazione.

Alla fine il grafico che si ottiene appare così:



La finestra è liberamente ridimensionabile. Il metodo `paint(Graphics g)` viene invocato a ogni modifica della finestra e quindi il grafico viene ridisegnato opportunamente scalato.

4.5. Esercizi

- Il primo esercizio richiede di creare un nuovo Frame semplice, senza barra dei menu. Al suo interno aggiungere un oggetto Panel come pannello base definito con il BorderLayout.
  Come per l'esempio visto creare una nuova classe che estende java.awt.Canvas e aggiungere tale componente nella posizione centrale (BorderLayout.CENTER) del pannello base.
  L'obiettivo della nuova classe è disegnare un cerchio al centro della finestra. Tale cerchio dovrà avere una diametro pari al 50% della dimensione della finestra, e dovrà naturalmente mantenere tale proporzione quando al finestra viene ridimensionata.
  Colorare inoltre con colori contrastanti piano della finestra e area del cerchio.

- Il secondo e terzo esercizio prevedono di modificare l'esempio visto nel capitolo.

  Aumentare l'offset dell'origine degli assi nell'area di lavoro, così da lasciare più spazio tra l'asse delle ascisse e il bordo della finestra. In tale spazio disegnare, a metà della lunghezza dell'asse, una stringa con la scritta “Giorni” così da dare un nome all'asse stesso.

- Convertire ora il tipo di grafico disegnato in un grafico a barre. Al posto dei puntini rossi e delle spezzate che li uniscono disegnare, per ogni valore, una barra rossa larga 5pixel, che parte dall'asse X fino all'altezza Y del punto in questione. (E' necessario modificare il metodo `disegnaValori()` e individuare la primitiva grafica opportuna tra i metodi della classe `Graphics`). *Suggerimento: le coordinate relative al sistema del Canvas dei punti del grafico sono già calcolate, quindi la modifica prevede solo di disegnare opportunamente un rettangolo per ogni barra.*

Si ricorda che per posizionare un componente o disegnare ad esempio una stringa vanno indicate le coordinate dell'angolo in alto a sinistra del componente stesso.

5. Elaborazione elementare di immagini con Gimp

Nell'ambito dell'acquisizione e memorizzazione di dati scientifici e di laboratorio può essere necessario manipolare file di immagini per adattarle alle proprie esigenze o modificarne le caratteristiche fondamentali.
Quando trattiamo una immagine memorizzata con un certo formato, ci troviamo davanti ad un segnale digitale ottenuto da un segnale analogico in due dimensioni spaziali.
Tale trasformazione passa per le fasi di campionamento e quantizzazione che influenzano la qualità dell'immagine che si ottiene.

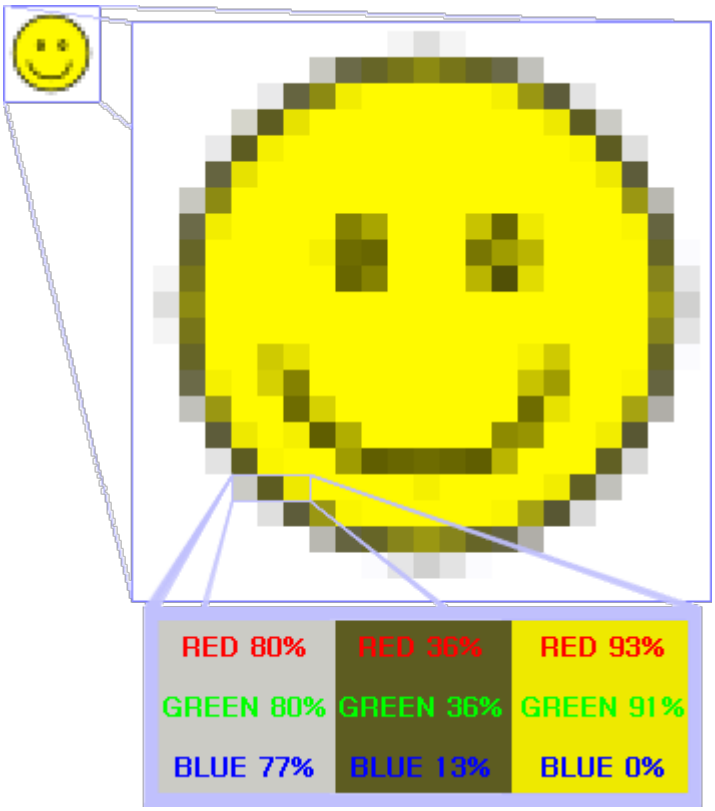
Una immagine digitale è quindi caratterizzata da una dimensione, ovvero il numero di pixel totale che la compongono, e dalla profondità di colore, cioè il numero di bit che rappresentano il colore di ciascun punto.
La dimensione in pixel è determinata dalla fase di campionamento mentre la profondità di colore dalla successiva fase di quantizzazione.

Un'altro aspetto da tenere in considerazione quando si trattano immagini, acquisite da dispositivi quali scanner o che si devono successivamente stampare, è la risoluzione, che collega il numero di pixel dell'immagine con le sue dimensioni di visualizzazione (su carta, su monitor).
Tale parametro è spesso indicato con l'unità di misura dpi (dot per inch) o ppi (point per inch) dove inch significa “pollice” e vale 2.54 centimetri.
Una risoluzione, ad esempio, di 300 dpi indica che il dispositivo è in grado di rilevare 300 campioni dall'immagine analogica per pollice. Stesso discorso vale per le stampanti, dove 300 dpi indicano che è in grado di stampare fedelmente in un pollice 300 pixel dell'immagine digitale.
Con risoluzione si intende dunque il numero di pixel contenuti nell'unità di misura considerata.

5.1. Gimp

Gimp (GNU Image Manipulation Program) è un programma di fotoritocco che permette di creare e modificare immagini raster. Immagine raster, termine che si significa trama/reticolo, si contrappone a immagine vettoriale, dove l'immagine stessa è definita con la descrizione degli elementi geometrici elementari che la compongono.

Questo è un esempio che mostra come invece è rappresentata una immagine raster.



Una volta aperto, il programma Gimp presenta la sua interfaccia con una finestra principale dove verrà caricata l'immagine da elaborare, e con due finestre secondarie contenti gli strumenti di disegno e altri box per la gestione dei colori, dei livelli e delle proprietà degli strumenti stessi.

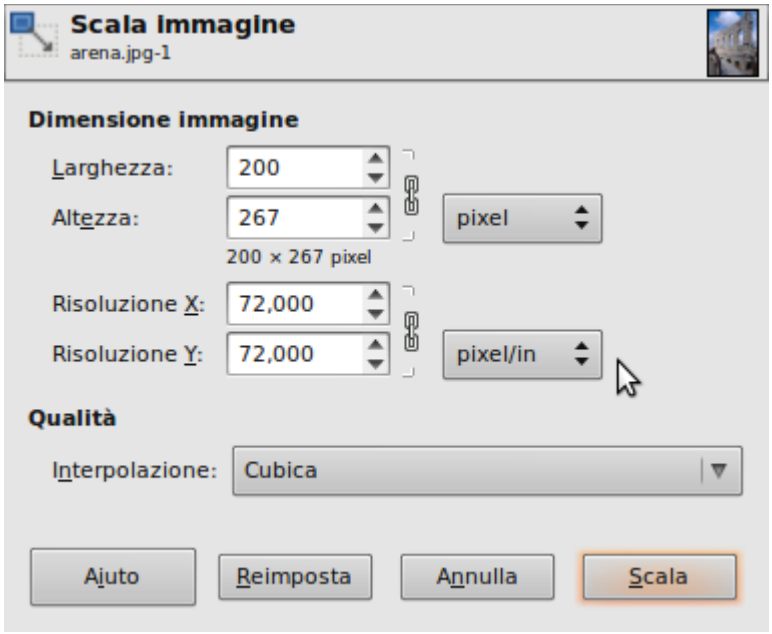
Una volta caricata una immagine la finestra ci mostra sulla barra del titolo, a fianco del nome, la sua dimensione in pixel.

Per vedere la risoluzione si può andare sul menu Immagine/Proprietà.

Nell'esempio l'immagine ha una dimensione di 768x1024 e una risoluzione di 72 ppi.

5.2. Scalare le dimensioni

Vogliamo ora ridurre la dimensione dell'immagine ad una larghezza di 200 pixel. Dal menu Immagine selezioniamo la voce “Scala immagine”.



Indicando nel campo “Larghezza” i 200 pixel desiderati, l'altezza viene aggiornata mantenendo la proporzione precedente tra le due misure. Applichiamo la modifica premendo “Scala”. Si noti che non viene alterati il contenuto della casella “Risoluzione” e quindi essa non cambia; il risultato stampato occuperà quindi un'area più piccola.

Si provi a caricare l'immagine dell'Arena di Verona, si riduca la larghezza da 768 a 200 pixel e si salvi il risultato in un altro file. Si provi a caricare il secondo file e a operare ora un aumento di dimensione, riportando l'immagine ai 768 pixel di larghezza come in origine; nel campo “Interpolazione” si selezioni “Nessuna”. Si confronti il risultato con l'immagine del file originale e si valuti le differenze. Da cosa son causati i difetti riscontrati ?

Attenzione! L'operazione di ridimensionamento di una immagine è irreversibile, in quanto nel momento in cui la si riduce si perde informazione che non si può più recuperare riportando l'immagine alla dimensione originale.

I difetti sono causati dalla perdita di informazione dovuta alla riduzione della dimensione da 768 a 200. Gimp prova a sopperire alla mancanza di informazione interpolando i dati sul colore dei pixel limitrofi (ecco perchè c'è l'opzione per scegliere l'algoritmo di interpolazione); l'interpolazione è comunque solo un rimedio parziale che funziona solo in certe parti dell'immagine prive di molti dettagli.

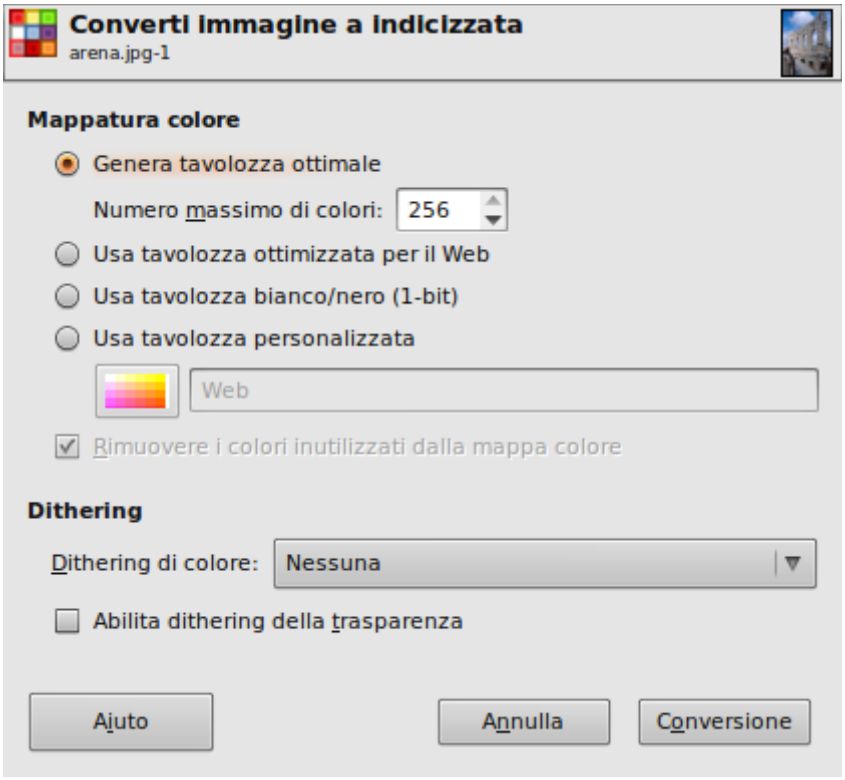
Si noti che un'immagine ridotta di dimensione e poi riportata alla dimensione originale senza mai variare la risoluzione, tornerà ad essere stampata su un'area grande ma con i difetti dovuti alla perdita di informazione.

Questa è l'immagine riportata alla dimensione originale senza alcuna interpolazione. Seppur non fedele a quella mostrata a video, si nota come i bordi degli elementi, e dove ci sono netti contrasti tra i colori, i pixel risultano molto sgranati e i tratti poco definiti.

Esercizio 1: *provare con diverse immagini a ridurre e riprostinare la dimensione specificando diversi tipi di interpolazione per notare le differenze sulla qualità dell'immagine risultante (si salvi sempre in un file a parte la versione ridotta/ripristinata in modo da poterla confrontare con l'originale). Riaprendo tutti i file cosa si può notare ?*

5.3. La profondità di colore

Lavoriamo ora sullo spazio dei colori e sulla profondità dei colori.
Lo spazio colore in cui è rappresentata l'immagine è l'RGB, ovvero un modello additivo basato sui tre colori fondamentali Red Green e Blue.
Dal menu Immagine → Modalità → Scala di grigi possiamo passare allo spazio in scala di grigi così da ottenere l'immagine i bianco e nero.
Possiamo inoltre, con la modalità "Indicizzata" (sempre dal menu Immagine → Modalità →), variare il numero di bit della profondità di colore.



Nel campo "Numero massimo di colori" di default troviamo 256, che corrisponde a 8bit di profondità colore (log base 2 di 256).

Esercizio 2: *provare, sulla stessa immagine di partenza, a variare la profondità di colore a 7, 6 e 4 bit. Si salvi l'immagine trasformata in un secondo file in modo da confrontarla con l'originale. Riaprendo tutti i file cosa si può notare ?*

Cosa accade se si prova, in seguito, a riportare la profondità al valore originale di 8 bit (256 colori) ?

Attenzione! L'operazione di riduzione della profondità di colore di una immagine è irreversibile, in quanto nel momento in cui la si riduce si perde informazione che non si può recuperare anche ritornando alla profondità originale.

5.4. Memorizzazione e compressione delle immagini

La memorizzazione di immagini richiede una grande quantità di spazio.
Facciamo l'esempio di una immagine di 2048x1024 pixel con profondità colore a 32 bit (cioè 8 bit per compomente RGB). Il numero totale di bit necessari a rappresentarne l'informazione è dunque 2048x1024x32, che corrispondono a 8388608 byte.
Questo spazio di memorizzazione può essere ridotto ricorrendo a formati immagine compressi, siano essi lossless che lossy. Con **lossless** si intende una compressione senza perdita di informazione, ovvero consente di decomprimere il file ottenendo gli stessi pixel dell'originale.
Per ottenere una maggior rapporto di compressione si ricorre ad algoritmi **lossy**, ovvero che introduzione perdita di informazione durante il processo di compressione. Uno di questi è adottato dallo standard JPEG.

Il passaggio da immagine Raster al formato JPEG avviene sostanzialmente in tre fasi:

- Utilizzo della trasformata discreta del coseno per la rappresentazione in frequenzia
- Quantizzazione mediante opportune matrici, che conservano maggiormente le basse frequenze spaziali, importanti ai fini della sintesi dell'immagine perchè sono quelle maggiormente percepite del sistema visivo umano
- codica entropica ed eliminazione delle ridondanze di tipo statistico tramite run-length encoding e codici di Huffman

In sostanza vi è un componente di elaborazione volta ad eliminare le informazioni analogiche che il sistema visivo umano percepisce con maggior difficoltà (fase di compressione lossy), e una componente di pura compressione lossless.
Seppur gli algoritmi lossy siano volti ad eliminare le informazioni analogiche che il sistema visivo umano percepisce con maggior difficoltà, se si abusa del fattore di compressione lossy, il degrado della qualità sarà percepibile.

JPEG è lo standard di compressione lossy utilizzato in ambito fotografico, pienamente supportato da GIMP. Altri formati lossless in cui possiamo memorizzare le nostre immagini sono PNG e BMP.

I formati di compressione lossy si utilizzano in applicazioni dove l'utilizzatore finale è un essere umano e l'applicazione non è critica dal punto di vista sanitario; ad esempio si possono usare per le foto delle vacanze o la foto tessera della carta di identità. Le immagini che devono/potrebbero venir analizzate da un sistema informatico non devono essere compresse con formati lossy che potrebbero rimuovere informazioni importanti per l'algoritmo di analisi (ad esempio ricerca automatica di tumori in immagini mediche).

Esercizio 3: *aprire il file dell'Arena di Verona e salvarlo sempre in formato JPEG (basta indicare come estensione .jpg) con valori decrescenti del coefficiente di qualità. Ad esempio salvare in arena60.jpg la versione ottenuta portando la barra della qualità al 60%, passando poi a 30, 15, 10, 1.*

- *Riaprendo tutti i file cosa si può notare ?*
- *Guardando la dimensione dei file cosa si può notare ?*
- *Osservare la dimensione in pixel, risoluzione e profondità di colore di tutti i file; è uguale per tutti ? Si potrebbe dire che il degrado di qualità sia dovuto ad un cambio di uno di questi parametri ?*