



Oggetti grafici e layout

Start:

Questa volta parleremo della grafica, impareremo a costruirci delle interfacce grafiche gradevoli per poter godere a pieno delle comodità che un sistema grafico può offrirci.

Credo sia doveroso partire dalla spiegazione di alcuni concetti di base, come "widget" e "layout" che incontreremo spesso d'ora in poi, per poi parlare di alcune tra le tante classi grafiche che il KVIrc ci offre.

Abbiamo visto fino ad adesso la creazione di classi nostre, utilizzando la classe base `object` come negli esempi del volume precedenti, eccone un'altro:

```
class(test,object)
{
    constructor()
    {
        echo $k(4,8) Test
    }
}
%Test=$new(test)
```

Ovviamente questo script non fa altro che dare l'output a schermo della scritta `Test` in pratica abbiamo creato una *classe test*, che "eredita" dalla *classe object* del KVIrc.

Cerchiamo di capire in parole povere cosa significa ereditare, lo spiegherò in modo semplice ed elementare perché chi scripta non necessariamente deve essere un programmatore, e qualora lo sia non ha certamente bisogno che io gli spieghi cosa vuol dire ereditare =).

Che Tizio eredita da Caio significa che Tizio entra in possesso di tutto ciò che ha Caio, ed ad esso aggiungerà tutto ciò che ha lui.

Che una classe eredita da un'altra significa che entra in possesso di tutte le sue capacità (quindi le funzioni e le caratteristiche di quest'ultima) ed in più, ovviamente, a queste potrà aggiungere le proprie caratteristiche e funzioni.

Proviamo a fare 2 classi che ereditano una le funzioni dell'altra così capiremo meglio:

```

class(padre,object)
{
    testEredita()
    {
        echo $k(4,8) Faccio parte della classe \"padre\"
    }
}
class(figlio,padre)
{
    testFiglio()
    {
        echo $k(4,8) Faccio parte della classe \"figlio\"
    }
}

%Prova=$new(figlio)
%Prova->$testEredita()
%Prova->$testFiglio()
echo %Prova->$classname()

```

Vediamo un po' di capire questo semplice codice:

intanto abbiamo creato una classe chiamata "padre", a questa classe, come possiamo vedere dal codice, abbiamo dato solo una funzione cioè a dire "\$testEredita()", notiamo che questa classe "padre" eredita dalla classe object, in questo modo "**class(padre,object)**", quindi, secondo quanto abbiamo detto sul concetto di ereditarietà, essa avrà a disposizione tutte le funzioni della classe object più le proprie funzioni (che nel nostro caso sono una soltanto cioè \$testEredita()), cioè nel momento in cui creassimo un oggetto con la nostra classe "padre" esso godrebbe delle sue funzioni più quelle ereditate dalla classe object, ma continuiamo con l'esempio per capire ancora meglio. Poi abbiamo creato una classe "figlio" e l'abbiamo fatta ereditare dalla classe "padre", in questo modo "**class(figlio,padre)**", a questo punto, secondo il concetto di ereditarietà, la classe "figlio" godrà delle funzioni proprie e di quelle della classe padre.

In seguito, infatti, abbiamo creato un oggetto %Prova utilizzando la classe "figlio"(%Prova=\$new(**figlio**)) e poi usato le funzioni \$testEredita() che in realtà appartiene alla classe "padre" e la funzione \$testFiglio() che invece appartiene ad essa stessa.

Non basta, come vedete abbiamo anche usato la funzione \$classname() che serve a ritornare il nome di una classe e che appartiene alla classe "object", infatti, dato che la classe "padre" eredita dalla classe "object", la classe "figlio" godrà anche delle funzioni di questa (geneticamente sarebbe un nonno =P).

A questo punto sarebbe d'obbligo andare a vedere, sul manuale del KVIrc le funzioni della classe object, anche per sapere un po' cosa ereditiamo =P (dal manuale *Object Classes-> object class*).

adesso passiamo alla grafica.

Intanto dobbiamo capire cosa è una "widget", diciamo semplicemente che per widget si intende un oggetto grafico, facciamo subito una prova:

```

%oggettoGrafico = $new(widget)
%oggettoGrafico->$show()

```

incolliamo nello script tester ed eseguiamo.

Come vedrete apparirà come una finestra, più precisamente appare un oggetto grafico vuoto, se andiamo a vedere la classe widget nel

manuale notiamo che è forse quella più ricca di funzioni (Dal manuale dell' *Help Object Classes->widget class*) dato che questa è la classe base per gli oggetti grafici e, come vedremo, tutte le classi grafiche ereditano da questa, quindi ciascuna classe grafica, *button*, *listview*, *label* o *lineedit* che sia, godrà oltre che delle proprie funzioni anche di quelle della classe *widget*.

Cominciamo a prendere un po' di familiarità con l'oggetto grafico *widget* e le sue funzioni arricchendo il codice di prima con qualche altra funzione di quelle che si trovano sull'help del KVIrc.

```
%oggettoGrafico = $new(widget)
%oggettoGrafico->$setGeometry(250,150,500,155)
%oggettoGrafico->$setCaption("Qt Eccomi\!")
%oggettoGrafico->$setIcon(217)

%oggettoGrafico->$show()
```

Come possiamo vedere abbiamo usato 4 delle tantissime funzioni che ci da la classe *widget* per "modellare" il nostro oggetto grafico:

`$setGeometry(<x>,<y>,<larghezza>,<altezza>)`

che ci permette di settare la posizione e le dimensioni del nostro oggetto,

`$setCaption(<text>)`

che ci permette di settare il titolo del nostro oggetto

`$setIcon(<image_id>)`

che ci permette di settare l'icona, il cui numero lo ricaviamo dalla tabella delle icone del kvinc (*Strumenti->Mostra tabella delle icone*)

`$show()`

che è la funzione più importante poiché mostra l'oggetto e tutti i suoi eventuali figli.

I figli di un oggetto sono gli oggetti che ad esso sono legati e non dobbiamo confonderci col concetto di ereditarietà visto prima, perché se un oggetto è figlio di un'altro non eredita nessuna funzione da quest'ultimo, gli è solamente legato ...un esempio chiarirà le idee.

```
%oggettoGrafico = $new(widget)
%oggettoGrafico->$setGeometry(250,150,200,155)
%oggettoGrafico->$setCaption("Qt Eccomi\!")
%oggettoGrafico->$setIcon(217)
%bottone=$new(button,%oggettoGrafico)
%bottone->$setGeometry(50,50,90,40)
%bottone->$setText("Cliccami\!")
%oggettoGrafico->$show()
```

Come al solito le aggiunte sono in grassetto, andiamo ad esaminare cosa abbiamo fatto a parte ridurre da 500 a 200 la larghezza della widget principale:

`%bottone=$new(button,%oggettoGrafico)`

qui abbiamo creato un oggetto di tipo "button" (andate a vedere la classe *button* nel manuale del KVIrc per vedere di quali funzioni essa gode) e come vedete l'abbiamo creata come figlio di "%oggettoGrafico", cioè l'abbiamo legata alla widget principale, infatti se eseguiamo il codice il pulsante si troverà all'interno di quest'ultima.

Le altre 2 funzioni usate credo che si spieghino da sole.

Notate che lo `$show()` è stato fatto solo dell'oggetto principale, non è stato necessario farlo anche del figlio.

Adesso però dobbiamo far fare qualche cosa al pulsante, cioè dobbiamo

farlo reagire ai nostri click.

Per fare questo useremo la funzione *privateimpl* la cui sintassi è:

```
privateimpl (<oggetto>, <funzione>) { <implementazione>; }
```

In pratica questa funzione serve per reimplementare una funzione che appartiene all'oggetto, ovvero a fargli eseguire la serie di comandi **<implementazione>** (in parole povere viene eseguita la funzione reimplementata al posto di quella "originale"), facciamo un esempio per capire meglio:

```
%oggettoGrafico = $new(widget)
%oggettoGrafico->$setGeometry(250,150,200,155)
%oggettoGrafico->$setIcon(217)
```

```
privateimpl(%oggettoGrafico, setCaption) {
echo Prova
}
```

```
%oggettoGrafico->$setCaption("Qt Eccomi\!")
%oggettoGrafico->$show()
```

come potete vedere abbiamo reimplementato la funzione *setCaption* dell'oggetto *%oggettografico*, infatti se eseguiamo il codice noteremo che quello che farà adesso la funzione *\$setCaption()* è dare l'output a schermo della parola prova e non la sua funzione normale, cioè quella di settare il titolo nella barra dell'oggetto (vi ricordo che l'echo verrà emesso sulla finestra della console del KVIrc quindi darà li che lo vedrete).

Dopo questa piccola digressione, andiamo a usare questa funzione per reimplementare la funzione

```
$mousePressEvent(<bottone_premuto>, <x>, <y>)
```

(notate che questa funzione di default non fa niente, essa viene richiamata in modo automatico quando il pulsante del mouse viene premuto ed il mouse si trova nella widget *<bottone_premuto>* sarà 0 se è stato premuto il pulsante sinistro del mouse, 1 per quello destro e 2 per quello centrale, *x* e *y* sono le coordinate relative alla widget su cui abbiamo premuto il pulsante e sono espresse in pixel)

ora modifichiamo il nostro codice:

```
%oggettoGrafico = $new(widget)
%oggettoGrafico->$setGeometry(250,150,200,155)
%oggettoGrafico->$setIcon(217)
%oggettoGrafico->$setCaption("Qt Eccomi\!")
%bottone=$new(button,%oggettoGrafico)
%bottone->$setGeometry(50,50,90,40)
%bottone->$setText("Cliccami\!")
%Label=$new(label,%oggettoGrafico)
%Label->$setText("Sono una label")
%Label->$setGeometry(50,20,120,20)
privateimpl(%bottone,mousePressEvent)
{
    if($0==0)
    {
        %Label->$setText("Tasto sinistro premuto");
    }
    if($0==1)
    {
        %Label->$setText("Tasto destro premuto");
    }
}
```

```

    if($0==2)
    {
        %Label->$setText("Tasto centrale premuto");
    }
}
%oggettoGrafico->$show()

```

Ecco che abbiamo messo in pratica il tutto =D, come potete vedere abbiamo aggiunto una label, e l'abbiamo chiamata %Label (w la fantasia) e come potete notare l'abbiamo creata globale (comincia per lettera maiuscola) poiché, dato che dobbiamo cambiarla tramite la pressione del tasto, ci serve che "persista" e non muoia dopo la creazione (provate a chiamarla %label e vedetene gli effetti).

Dopo abbiamo usato la funzione *privateimpl* e, ricordandoci che 0 è il tasto sinistro del mouse, 1 quello destro e 2 quello centrale, abbiamo creato la nostra implementazione della funzione *mousePressEvent*, che se il tasto premuto sarà quello sinistro (**if(\$0==0)**) setterà il testo della label a **"Tasto sinistro premuto"** se sarà quello destro a **"Tasto destro premuto"** mentre in caso sia quello centrale a **"Tasto centrale premuto"**.

Adesso dobbiamo introdurre il concetto di layout, il layout è una classe fantastica, essa ci permette di non impazzire dietro le coordinate geometriche e ci permette di allineare e posizionare in modo pulito gli elementi di una widget.

Come dice il manuale (tradotto ed adattato):

"Il layout è una classe per gestire la geometria degli oggetti figli di una widget. Tu crei un layout, gli dai degli oggetti da gestire, e lui li dispone in modo geometricamente pulito.

*Il padre del layout deve essere la widget di cui si vogliono sistemare geometricamente i figli. Esso (il layout) è una griglia virtuale di NxM celle nella quale puoi inserire gli oggetti grafici con la funzione **\$addWidget()**.*

*Oppure se essi devono occupare più di una cella con la funzione **\$addmulticellwidget()**."*

Vediamone subito un esempio:

```

# Creo i tre oggetti
%oggettoGrafico = $new(widget)
%layout=$new(layout,%oggettoGrafico)
%bottone=$new(button,%oggettoGrafico)
# Oggetto principale
%oggettoGrafico->$setIcon(217)
%oggettoGrafico->$setCaption("Qt")
# Label
%Label=$new(label,%oggettoGrafico)
%Label->$setText("Sono una label")
# Bottone
%bottone->$setText("Cliccami!\")
privateimpl(%bottone,mousePressEvent)
{
    if($0==0) {%Label->$setText("Tasto sinistro premuto");}
    if($0==1) {%Label->$setText("Tasto destro premuto");}
    if($0==2) {%Label->$setText("Tasto centrale premuto");}
}
# Aggiungo gli oggetti figli al layout
%layout->$addWidget(%Label,0,0)
%layout->$addWidget(%bottone,1,0)

```

```
# Mostro il tutto
%oggettoGrafico->$show()
```

Come potete vedere abbiamo eliminato tutti i `$setGeometry`, demandando al layout la gestione delle geometrie.

La funzione utilizzata è `$addWidget()` che ha la sintassi:

```
$addWidget(<oggetto_grafico>,<riga>,<colonna>)
```

Ricordando che il layout è come una griglia virtuale, nel nostro esempio, abbiamo disposto la label alla riga 0 e alla colonna 0 mentre il bottone proprio sotto, cioè alla riga 1 della colonna 0 della nostra griglia virtuale, notate anche che se allarghiamo o stringiamo la finestra, gli oggetti grafici si adatteranno, mantenendo le proporzioni e la posizione.

Se volessimo ad esempio che il pulsante fosse la metà della label, cioè prendesse 1 cella virtuale, mentre la label ne prendesse 2, potremmo usare la funzione:

```
$addMultiCellWidget(<oggetto>,<from_row>,<to_row>,<from_col>,<to_col>)
```

E il codice risulterebbe essere così:

```
%oggettoGrafico = $new(widget)
%layout=$new(layout,%oggettoGrafico)
%bottone=$new(button,%oggettoGrafico)

%oggettoGrafico->$setIcon(217)
%oggettoGrafico->$setCaption("Qt")

%Label=$new(label,%oggettoGrafico)
%Label->$setText("Prendo 2 Celle!!!!!!!!!!!!!!!!!!!!!!!!!!!!")

%bottone->$setText("Cliccami!\")
privateimpl(%bottone,mousePressEvent)
{
    if($0==0) {%Label->$setText("Tasto sinistro premuto");}
    if($0==1) {%Label->$setText("Tasto destro premuto");}
    if($0==2) {%Label->$setText("Tasto centrale premuto");}
}

%layout->$addMultiCellWidget(%Label,0,0,0,1)
%layout->$addMultiCellWidget(%bottone,1,1,0,0)

%oggettoGrafico->$show()
```

Come vedete, adesso il pulsante occupa 1 cella (`%bottone,1,1,0,0` una riga [la 1,1] ed una colonna [la 0,0]) mentre la label ne occupa 2 (`%Label,0,0,0,1` una riga [la 0,0] ma due colonne vicine [la 0,0 e la 1,1 infatti (`%Label,0,0,0,1`)).

Proviamo a fare una finestrella più complessa.

```
# Creo i tre oggetti
```

```
%oggettoGrafico = $new(widget)
%layout=$new(layout,%oggettoGrafico)
# Oggetto principale
%oggettoGrafico->$setIcon(217)
%oggettoGrafico->$setCaption("Qt Mirror")
```

```
# Creo le Label
```

```
%Labelfrasebase=$new(label,%oggettoGrafico)
%Labelfrasebase->$setText("Frase Base")
```

```

%Labelfrasereversata=$new(label,%oggettoGrafico)
%Labelfrasereversata->$setText("Frase Reversata")
# Creo i Quadranti
%QuadranteFraseBase=$new(multilinedit,%oggettoGrafico)
%QuadranteFraseReversata=$new(multilinedit,%oggettoGrafico)
%QuadranteFraseReversata->$setReadOnly(1)
# Creo i pulsanti
%bottone=$new(button,%oggettoGrafico)
%bottone->$setText("Reversa\!")
privateimpl(%bottone,mousePressEvent)
{
    if($0==0)
    {
        %testoDareversare = %QuadranteFraseBase->$text()
        %idx=$( $str.len(%testoDareversare)+1)
        while(%idx!=0)
        {
            %sztmp=$str.section(%testoDareversare,"",%idx,%idx)
            %szTestoreversato=%szTestoreversato%sztmp;
            %idx--;
        }
        %QuadranteFraseReversata->$setText(%szTestoreversato)
    }
}
# Aggiungo gli oggetti figli al layout
%layout->$addMultiCellWidget(%Labelfrasebase,0,0,0,6)
%layout->$addMultiCellWidget(%Labelfrasereversata,0,0,7,13)
%layout->$addMultiCellWidget(%QuadranteFraseBase,1,1,0,6)
%layout->$addMultiCellWidget(%QuadranteFraseReversata,1,1,7,13)
%layout->$addMultiCellWidget(%bottone,2,2,0,0)

%oggettoGrafico->$show()

```

Non credo ci sia molto da aggiungere a quanto già detto, il codice parla chiaro e la routine per capovolgere la frase è la solita che abbiamo visto negli altri tutorial e tutte le altre funzioni usate le abbiamo già viste, a parte `$setReadOnly()` che come potete immaginare serve per settare il secondo quadrante come di sola lettura, la funzione vuole un valore booleano (la sua sintassi è `$setReadOnly(<bool_value>)`) quindi 1(VERO) oppure 0(FALSO) a seconda che vogliamo impostare la proprietà di sola lettura oppure no, considerando che di default è falso.

Anche per questa volta abbiamo finito.
Digitiamo con soddisfazione il nostro...

/ECHO STOP

" Tu vedi cose e ne spieghi il perché, io invece immagino cose che non sono mai esistite e mi chiedo perché no." (George Bernad Shaw)

Grifisx