



Oggetti, Classi, Array e dizionari

Start:

In questo tutorial ci occuperemo dei concetti di *oggetto*, *classe* degli *array*, dei *dizionari* e della *scrittura e lettura dei file*, anche questa volta daremo molto spazio a piccoli script, perché il modo migliore per imparare è buttare giù codice.

Per prima cosa introdurremo il concetto di oggetto, cosa importante da conoscere se si vuol scrivere codice più complesso.

Per capire cosa sia un oggetto, senza perderci in giri di parole, vediamo il seguente codice:

```
%myobject = $new(object,0,oggetto);  
if(%myobject)echo "Oggetto creato correttamente!";  
else echo "Creazione dell'oggetto fallita!";
```

Mettetelo nello "script tester" (quello con l'icona della bomba a mano nera) ed eseguitelo per vederne il risultato.

Adesso cominciamo subito l'esame di quello che abbiamo scritto:

```
%myobject = $new(object,0,nome_oggetto);
```

Per creare un oggetto bisogna usare la funzione `$new()` la quale richiede 3 parametri:

- La classe dell'oggetto (dopo vedremo cosa si intende per classe)
- L'ID dell'oggetto padre, (che può essere 0 per gli oggetti top-level (cioè a dire oggetti che non hanno padre, questi concetti verranno spiegati in seguito).
- Il nome dell'oggetto (può anche essere vuoto).

Gli oggetti possono essere trattati come pseudo-strutture (tipo le strutture del c per intendersi), non preoccupatevi se non avete la più pallida idea di cosa sia una struttura con il codice si chiarirà tutto; mettiamo il caso che vogliamo creare un oggetto utente, che abbia all'interno di se vari campi contenenti Nick, Username, etc. etc.:

```
# Creiamo l'oggetto utente  
%Utente = $new(object,0,descrizione_utente)  
# Settiamo i campi  
%Utente->%nickname = Grifisx  
%Utente->%username = mDm-Team  
%Utente->%hostname = non.disturbatemi.net  
%Utente->%info = Grifisx ha preso questo es dal manuale del KVirc
```

```
%Utente->%info << Tnx 2 Pragma
```

Mettiamo il codice nello script-tester ed eseguiamo, dopodiché posizioniamoci sulla riga di input, di qualsiasi finestra, e digitiamo:

```
/echo %Utente->%nickname
```

come vedrete ci ha dato l'echo del campo nickname, ovviamente farebbe lo stesso con tutti gli altri campi, mentre se facciamo un echo dell'oggetto in se tipo:

```
/echo %Utente
```

darebbe come output l'id dell'oggetto (nel mio caso 1710.1128236637). Come potete vedere è molto comodo usare gli oggetti, soprattutto se si devono creare script molto complessi.

Adesso passiamo ad esaminare le classi.

Una classe, come anche si legge sul manuale ufficiale, è una collezione di metodi che definiscono le caratteristiche di un oggetto, però così detto non è molto chiaro, quindi chiamiamo in aiuto il codice:

```
class (calc,object)
{
    constructor()
    {
        echo Usage\:
        echo Per calcolare la somma di 2 numeri:
        echo \/%Calc-\->\$somma2\(\x\,\y\)
        echo Per calcolare il quoziente di 2 numeri:
        echo \/%Calc-\->\$div\(\x\,\y\)
    }
    somma2()
    {
# $0 e $1 sono i valori che noi passeremo alla funzione richiamandola
# ad esempio in "\%Calc->$somma2(8,6)" $0=8 e %1=6
        echo $($0+$1)
    }
    div()
    {
        echo $($0/$1)
    }
}
%Calc=$new(calc)
```

L'esecuzione di questo script avrà l'effetto di darvi la possibilità di sommare o dividere 2 numeri tra di loro utilizzando le funzioni che voi avrete creato \$somma2() e \$div().

Provate ad esempio a fare dalla riga di input

```
/%Calc->$div(10,5)
```

Come si può vedere per prima cosa è stata creata una classe *calc* che è di tipo *object* (potrebbe anche essere di tipo *widget*, ad esempio, per gli oggetti grafici, ma questo lo vedremo più avanti) e , all'interno della classe, sono state create 3 funzioni (o metodi come preferite) *\$constructor()*, *\$somma2()* e *\$div()*; dopo questo creiamo l'oggetto con il codice "*%Calc=\$new(calc)*" e a questo punto possiamo usare le funzioni che ho realizzato all'interno della classe, come vedete basta richiamarle tramite il simbolo "->".

Ricordiamo tra l'altro che la funzione *\$(<espressione_aritmetica>)* esegue il calcolo dell'espressione contenuta all'interno delle

parentesi.

Inoltre si noti che il codice dentro `$constructor()` viene eseguito alla creazione della classe in modo automatico, questo è il posto migliore dove, ad esempio, inizializzare variabili.

Altro esempio:

```
class(rubrica,object)
{
  addName()
  {
    $$->%name=$0-
  }
  addSurname()
  {
    $$->%surname=$0-
  }
  addTel()
  {
    $$->%tel=$0-
  }
}
%Rubrica=$new(rubrica)
```

Per prima cosa spieghiamo cosa sono quel `$$->` e il `$0-`:

il `$0-` è il parametro che la funzione riceve (come gli alias ricordate? Funziona allo stesso modo, cioè quando chiameremo la funzione `$addName()` dovremo passargli il nome da aggiungere).

`$$`, invece, sta per "Variabile appartenente a questa classe", cioè se dobbiamo creare una variabile di classe (visibile solo all'interno della classe ed ad essa appartenente), dobbiamo crearla con `$$->` oppure con `$this->`, allo stesso modo, se dovessimo chiamare una funzione della stessa classe all'interno di un'altra funzione dovremmo chiamarla con `$$->$funzione()`.

Prima di passare a fare del codice più complesso dobbiamo chiarire alcuni piccoli concetti, per l'esattezza quello di `array` e quello di `dizionario` sono concetti importanti, perché ci permettono di creare delle "collezioni" di valori, di stringhe o di oggetti, insomma di dati.

Un array è una collezione di dati variabili indicizzati per numero, il primo indice dell'array è 0 mentre l'ultimo è uguale alla grandezza dell'array meno uno (poiché si parte da zero).

Per ottenere il numero di elementi che è contenuto in un array possiamo usare l'espressione `%ArrayEsempio[]#`.

Non è necessario dichiarare la grandezza dell'array come in altri linguaggi di programmazione, a mano a mano che si aggiungerà un numero, la grandezza del nostro array varierà automaticamente e se il primo elemento che assegneremo, lo assegneremo ad un indice maggiore di 0, tutte le posizioni precedenti saranno vuote.

Proviamo ad esempio:

```
%Array[0]=Grifisx
%Array[1]=Noldor
%Array[2]=Pragma
#Stampo il contenuto di tutto l'array
echo %Array[]
#Stampo la grandezza dell'array
```

```
echo %Array[]#  
#Stampo solo il primo elemento  
echo %Array[0]
```

oppure proviamo questo codice:

```
%Array[0]=Grifisx  
%Array[1]=Non mostrare questo  
%Array[2]=Noldor  
%Array[5]=Segreto shhhh..  
%Array[8]=Pragma  
for(%i=0;%i < %Array[]#;%i+=2)echo Entry %i: \"%Array[%i]\";
```

Come vedete è abbastanza semplice crearsi delle collezioni indicizzate per numero, così come lo è anche muoversi all'interno di esse, qui si è voluto usare un ciclo *for* ma ovviamente avremmo anche potuto usare un *foreach(%item,%Array[])echo %item =D*, oppure un ciclo *while*.

Un array potremmo anche inizializzarlo in questo modo
%Array[]=\$array(Grifisx,Noldor,Pragma,Madero) ;

cioè utilizzando la funzione *\$array(<el1>,<el2>,<el3>,<el4>,..)*.

Adesso è il momento di esaminare il fratello maggiore dell'array: il dizionario.

I dizionari non sono altro che array associativi di stringhe, per capire bene, riprendo l'esempio del manuale ufficiale:

```
%Songs{Jimi Hendrix} = Voodoo child  
%Songs{Shawn Lane} = Gray piano's flying  
%Songs{Mina} = Brava  
%Songs{Greg Howe} = "Full Throttle"  
# Mostra tutto in una stringa  
echo %Songs{  
# Mostra tutti gli elementi del dizionario  
foreach(%var,%Songs{ })echo %var
```

Ovviamente anche qui, come negli array *%Songs{ }#* restituirà il numero degli elementi del dizionario.

Mentre *%Songs{ }@* restituirà una lista degli elementi separata da virgole.

Ovviamente potremmo unire dizionari e array insieme, per avere un dizionario di array ad esempio.

Insomma possiamo gestirci le nostre collezioni come più ci piace.

Capisco che sono concetti un po' complicati ma vedrete che alla fine diventerà "naturale" usarli.

Adesso è necessario fare cenno su come si possa salvare e leggere da un file di testo, poiché se abbiamo delle collezioni, e vogliamo conservarle, la prima cosa da fare è metterle su file.

Vediamo:

```
$file.read("file_da_leggere")  
file.write("file_da_scrivere")
```

la prima funzione ci permette di leggere da un file di testo mentre il secondo (è un comando non una funzione, lo vedere dal fatto che non inizia per \$) ci permette di scriverci dentro

Quindi mettiamo il caso che avessimo un file di testo (tipo "userdatabase") con dentro una lista di nomi separati da virgole (tipo "Grifisx,Noldor,Pragma,Madero") e che volessimo mettere questi nick in un array, come dovremmo fare? Niente di più semplice:

```
%users[] = $split(",",$file.read($file.localdir(usersdatabase)))
```

la funzione `$split` (tra non molto verrà cambiata in `$str.split()` quindi occhio, se vi da errore) serve per dividere una stringa in base ad un separatore che gli diamo (in questo caso la virgola ","); mentre la funzione `$file.localdir()` restituisce la directory delle configurazioni locali del KVIrc (provate subito uno `"/echo $file.localdir()"`):

quindi, in questo modo, avremo creato l'array `%users[]`, che al suo interno ha i nick che abbiamo letto dal file.

Se volessimo scrivere sul file avremmo invece:

```
file.write $file.localdir(usersdatabase) %users[]
```

Vediamo di testare il tutto nello script tester:

```
%Users[]=$array(Grifisx,Noldor,Pragma,Madero);
file.write $file.localdir(usersdatabase) %Users[];
%UsersNew[] = $split(",",$file.read($file.localdir(usersdatabase)));
%idx=0;
while(%idx!=%UsersNew[]#)
{
    echo Utente: %UsersNew[%idx];
    %idx++;
}
file.remove $file.localdir(usersdatabase)
```

Come avrete potuto notare, eseguendo lo script, il file è stato creato, letto e successivamente rimosso con il comando `file.remove` (per vedere gli altri comandi disponibili per la gestione dei file, controllate nel manuale alla voce *Comandi* e poi lettera *f*).

Adesso, con le nuove nozioni che abbiamo imparato, andiamo a fare qualche cosa di più complesso ed allo stesso tempo più concreto: un piccolo gestore di appunti =)

```
class(appunti,object)
{
    constructor()
    {
# Nel costruttore faccio apparire a video le istruzioni d'uso,
# ricordando che il costruttore è la prima cosa che viene eseguita
# appena la classe viene creata
        echo $k(5,8) \-\-\-Appunti\-\-\-\
        echo $k(5,8) Comandi manuali:
        echo $k(5,8) \/%Appunti\-\>\$viewApp           $b
Visualizza gli appunti presenti
        echo $k(5,8) \/%Appunti\-\>\$addApp\ (appunto\ ) $b
Aggiunge un nuovo appunto
        echo $k(5,8) \/%Appunti\-\>\$delApp\ (appunto\ ) $b
Cancella un appunto
    }
# Funzione per aggiungere un appunto
    addApp()
    {
# Creo un array e lo riempio con il contenuto del file
        $$->%appuntos[] =
        $split(",",$file.read($file.localdir(kvircAppuntidb)))
        $$->%ap = 0
# Ora creerò un dizionario, perché è più semplice lavorare con
# stringhe e sono più facilmente individuabili, ma allo stesso tempo
# perché così avremo una applicazione pratica dell'uso dei dizionari,
```

```
# e dopo lo riempio con il contenuto dell'array, questo mi permetterà  
# di avere una cosa del tipo:  
# %Array{ciao}=ciao , in modo che, quando vorrò poi cancellarlo, non  
# dovrò preoccuparmi del suo indice numerico dato che il mio indice è  
# uguale all'elemento, in pratica lo rintraccerò sempre.
```

```
while($$->%ap < $$->%appuntos[]#)  
{  
    $$->%ElencoAppunti{$$->%appuntos[$$->%ap]} = $$->%appuntos[$$->%ap]  
    $$->%ap++;  
}  
$$->%ProvaAP = $0-  
$$->%ElencoAppunti{$$->%ProvaAP} = $$->%ProvaAP
```

```
# Visualizziamolo a schermo tutti gli appunti
```

```
foreach($$->%var,$$->%ElencoAppunti{ }) echo $k(5,8)$$->%var
```

```
# Conserviamo sul file
```

```
file.write $file.localdir(kvircAppuntidb) $$->%ElencoAppunti{ }  
echo $k(5,8) Fine Lista : $$->%ElencoAppunti{ }# \  
Appunti presenti  
}
```

```
# Funzione per cancellare un appunto
```

```
dellApp()  
{
```

```
# Come la funzione precedente
```

```
    $$->%appuntos[] =  
$split(",",$file.read($file.localdir(kvircAppuntidb)))  
    $$->%ap = 0  
    while($$->%ap < $$->%appuntos[]#)  
    {  
        $$->%ElencoAppunti{$$->%appuntos[$$->%ap]} = $$->%appuntos[$$->%ap]  
        $$->%ap++;  
    }  
    $$->%ProvaAP = $0
```

```
# Ecco che posso cancellarlo facilmente senza preoccuparmi dell'indice  
# dato che utilizzo il suo stesso nome come indice
```

```
    $$->%ElencoAppunti{$$->%ProvaAP} = ""
```

```
# svuotato =D
```

```
foreach($$->%var,$$->%ElencoAppunti{ }) echo $k(5,8)$$->%var  
file.write $file.localdir(kvircAppuntidb) $$->%ElencoAppunti{ }  
echo $k(5,8) Fine Lista : $$->%ElencoAppunti{ }# \  
Appunti presenti  
}
```

```
# Funzione per visualizzare gli appunti
```

```
viewApp()  
{
```

```
# Tutto come sopra
```

```
    $$->%appuntos[] =  
$split(",",$file.read($file.localdir(kvircAppuntidb)))  
    $$->%ap = 0  
    while($$->%ap < $$->%appuntos[]#)  
    {  
        $$->%ElencoAppunti{$$->%appuntos[$$->%ap]} = $$->%appuntos[$$->%ap]
```

```

>%appuntos[$$->%ap]
    $$->%ap++;
    }
    foreach($$->%var,$$->%ElencoAppunti{})echo $k(5,8) $$-
>%var
    echo $k(5,8) Fine Lista : $$->%ElencoAppunti{)# \)
Appunti presenti
    }
}
# Creo il mio oggetto %Appunti
%Appunti=$new(appunti)

```

Non credo che ci sia molto da spiegare, poiché alla fine, se avete capito bene i concetti di *array*, *dizionario* e di lettura e scrittura dei file (e ovviamente di *classe*), lo script si spiegherà da solo.

Si potrebbe fare senza ombra di dubbio uno script migliore (o utilizzare costrutti certamente più funzionali di quelli utilizzati), ma ci si è limitati ad usare tutti (e solo) gli argomenti trattati, cercando di riunirli in un unico script.

Lo farete sicuramente migliore quando avrete maturato tutti gli altri concetti che il linguaggio del KVIrc nasconde =).

/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