LEZIONE 8

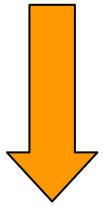
EREDITARIETÀ

EREDITARIETÀ

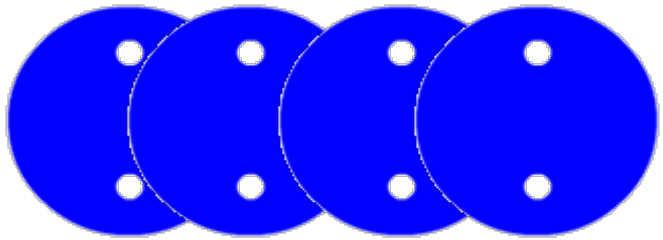
- Uno dei vantaggi assicurati dalla programmazione orientata agli oggetti è la possibilità di creare *sottoclassi* di una classe.
- La sottoclasse eredita tutte le proprietà e i metodi di una ***superclasse***.
  - Posso creare una classe estendendo una classe predefinita.
  - Ma posso anche creare un set di classi che estenda una superclasse sempre creata da me.

EREDITARIETÀ

MovieClip



PallaMov



animMoveTo(x:Number, y:Number)

EREDITARIETÀ

Vedura

EREDITARIETÀ

- La sottoclasse può aggiungere dei metodi alla superclasse e ridefinire i metodi ereditati creando così nuovi comportamenti per gli stessi metodi.
- L'uso di sottoclassi consente di riutilizzare il codice, in quanto estendendo una classe esistente si evita di riscrivere tutto il codice comune a entrambe le classi.

LE SOTTOCLASSI IN FLASH

- Nella programmazione orientata agli oggetti, una sottoclasse può ereditare le proprietà e i metodi di un'altra classe, detta ***superclasse***. È possibile estendere le classi personalizzate e molte delle classi di ActionScript. Non possono essere estese la classe ***TextField*** e le classi statiche, quali ***Math***, ***Key*** e ***Mouse***.
- Per creare questo tipo di relazione tra due classi, è necessario utilizzare la clausola ***extends*** dell'istruzione *class*:

```
class SubClass extends SuperClass {
     //corpo della classe
}
```

AGGIUNTA DI METODI E PROPRIETÀ

LA CLASSE JUKEBOX

- Ora definiremo una classe personalizzata ***JukeBox*** che estende la classe ***Sound***:
- Vengono aggiunti la proprietà
  - ***song_array*** di tipo *Array*
  - ***titles_array*** sempre di tipo *Array*,
- Vengono aggiunti i metodi
  - ***playSong***() che riproduce un brano musicale richiamando il metodo *loadSound*() ereditato dalla classe Sound,
  - ***addSong***() che aggiunge un brano al JukeBox
  - ***getTitleList***() che restituisce la lista dei titoli inseriti nel JukeBox.

CREIAMO IL FILE DI CLASSE

```
/**
      Classe JukeBox
      autore: Bruno Migliaretti
      versione: 1.0
      data modifica: 01/12/2006
      copyright: Accademia di Belle Arti di
      Urbino
      Questo codice definisce una classe JukeBox
      che estende la classe prdefinita Sound e
      consente di gestire una lista di brani
      musicali.
*/

class JukeBox extends Sound {
}
```

LE PROPRIETÀ AGGIUNTE

- Definire e inizializzare ora nel codice le proprietà ***song_array*** e ***titles_array*** aggiungendo al corpo della classe:

```
class JukeBox extends Sound {
	private var song_array:Array = new Array();
		//lista dei brani del JukeBox
	private var titles_array:Array = new Array();
		//lista dei titoli del JukeBox
}
```

IL METODO ADDSONG

- Aggiungere ora il codice per il metodo ***addSong***():

```
class JukeBox extends Sound {
.......
      public function addSong (titolo:String,
                  nomeFile:String):Number {
            /* aggiungo in coda ai due array il
               titolo del brano e il file musicale
               corrispondente*/
            this.titles_array.push(titolo);
            this.song_array.push(nomeFile);

            //restituisco l'indice del brano inserito
            return (song_array.length - 1);
      }
}
```

IL METODO PLAYSONG

- Il metodo *addSong*() accetta due parametri entrambi di tipo *String*: *titolo* e *nomeFile*. I due elementi vengono aggiunti rispettivamente agli array *song_array* e *titles_array*. Infine il metodo restituisce l'indice dell'elemento appena inserito.
- Aggiungere ora il codice per il metodo *playSong*()

```
class JukeBox extends Sound {
.......
      public function playSong (songId:Number) {
            /* il file memorizzato nella proprietà
               song_array con indice songId viene
               caricato ed eseguito */
            this.loadSound(song_array[songId], true);
      }
}
```

IL METODO GETTITLELIST

- Aggiungere ora il codice per il metodo ***getTitleList***()
- Il metodo ***getTitleList***() consente l'accesso in lettura alla proprietà *title_array* che devo poter essere modificata solo dal metodo ***addSong***() e quindi e definita come privata.

```
class JukeBox extends Sound {
.......
      public function getTitleList():Array {
            /* in questo modo titles_array è
               accessibile solo in lettura e può
               essere modificato solo usando il metodo
               addSong */
            return titles_array;
      }
}
```

USARE CLASSE JUKEBOX

- Ora creeremo un nuovo file flash per provare la classe JukeBox:
- Creiamo un testo statico e scriviamo Lista dei brani.
- Collochiamo poi sullo Stage un’istanza della componente List trascinandone l’icona dalla finestra componenti allo stage. Usiamo il pannello proprietà per dare alla componente il nome song_list.
- Creiamo un nuovo livello Azioni. Selezionimao il primo fotogramma del nuovo livello per scrivere il cosice che farà funzionare la nostra classe

USARE CLASSE JUKEBOX

- Creiamo un'istanza della classe JukeBox e poi aggiungiamo i brani:

```
//creo un'istanza di JukeBox
var jb:JukeBox = new JukeBox();

// aggiungo le informazioni sui brani musicali
jb.addSong("Blue", "0169_Blue.mp3");
jb.addSong("Cafe caldo", "0217_Cafe Caldo.mp3");
jb.addSong("Cello sweet", "0246_Cello Sweet.mp3");
jb.addSong("Conto alla rovescia",
"0328_Countdown.mp3");
jb.addSong("Vai facile", "0459_Easy Does It.mp3");
……
```

USARE CLASSE JUKEBOX

- E riempiamo la list Song List:

```
//riempio la lista song_list
var titoli:Array = jb.getTitleList();

for (var i = 0; i < titoli.length; i++) {
	song_list.addItem({label:titoli[i],data:i});
}

//gestisco l'evento change delle lista
var listHandler = new Object();

listHandler.change = function(evt:Object) {
	var index = evt.target.selectedItem.data;
	jb.playSong(index);
}
this.song_list.addEventListener("change",listHandler);
```

SOSTITUZIONE DI METODI E PROPRIETÀ

LA CLASSE DATAIT

- Uno dei vantaggi offerti dall'uso delle classi e dalla loro estensione è la possibilità non solo di garantire nuove funzionalità a una classe esistente aggiungendo metodi e proprietà, ma anche di modificare le funzionalità già presenti.
- Come esempio modificheremo il comportamento del metodo ***toString***() nella classe predefinita ***Date*** creando una classe che chiameremo ***DataIt***.
- Il medito ***toString***() è comune a tutte le classi predefinite e viene invocato automaticamente quando un'istanza di una qualsiasi classe viene usata come una stringa. Se scrivo:

```
var now:Date = new Date();
trace(now);
```

- Otterrò il seguente output:

  Tue Nov 28 22:07:32 GMT+0100 2006

LA CLASSE DATAIT

- Quando passo un oggetto come parametro alla funzione ***trace***() (che manda un messaggio sulla finestra di output) di fatto lo converto in stringa e quindi applico implicitamente il metodo ***toString***() all'oggetto.
- Modificando quindi il comportamento standard del metodo ***toString***() per un oggetto personalizzarò il modo con cui quell'oggetto verrà convertito in stringa.

IL CODICE

```
/**
		Classe DataIt
		autore: Bruno Migliaretti
		versione: 1.0
		data modifica: 01/12/2006
		copyright: Accademia di Belle Arti di Urbino
		Questo codice definisce una classe DataIt che estende la classe predefinita Date
		traducendone l'output in italiano.
*/
class DataIt extends Date {
		//creo una proprietà mesi che contiene i nomi dei mesi
		private var mesi:Array = new Array("gennaio", "febbraio", "marzo", "aprile", "maggio", "giugno", "luglio", "agosto",
					"settembre", "ottobre", "novembre", "dicembre");
		//e una proprietà giorni che contiene i nomi dei giorni della setimana
		private var giorni:Array = new Array("domenica", "lunedì", "martedì", "mercoledì","giovedì", "venerdì", "sabato");

		//scrivo una semplice metodo che aggiunge uno "0"
		//davanti ad un numero se è composto da una sola cifra
		private function zeroPrima(n:Number):String {
				var s:String = n.toString();
				if (s.length == 1) {
					s = "0" + s;
				}
				return (s);
		}

		//ridefinisco il metodo toString()
		public function toString():String {
				var giorno_sett:String = giorni[this.getDay()];
				var giorno:Number = this.getDate();
				var mese:String = mesi[this.getMonth()];
				var anno:Number = this.getFullYear();
				var ora:String = zeroPrima(this.getHours());
				var minuti:String = zeroPrima(this.getMinutes());
				var secondi:String = zeroPrima(this.getSeconds());
				return( giorno_sett + " " + giorno + " " + mese + " " + anno +
					" " + ora +":" + minuti + ":" + secondi);
		}
}
```

ESTENSIONE DELLA CLASSE MOVIECLIP

ASSEGNAZIONE DI UNA CLASSE A UN SIMBOLO CLIP FILMATO

- Normalmente per estendere il comportamento della classe MovieClip creo un simbolo di tipo MovieClip (che contiene tutti gli elementi grafici di cui ho bisogno) e associo a questo simbolo una classe che estenda il comportamento della classe MovieClip. filmato utilizzando la finestra di dialogo Proprietà del concatenamento.
- Ogni volta che viene creata un'istanza del clip filmato a cui la classe è assegnata, tale istanza assume le proprietà e i comportamenti definiti dalla classe assegnata.

AGGIUNTA DI UN METODO ALLA CLASSE MOVIECLIP

- Ora creeremo una classe che aggiunge alla classe MovieClip il metodo ***moveTo***(x:Number, y:Number) che sposta la MovieClip al punto indicato da x e y e la proprietà ***speed*** che determina la velocità di questo spostamento.

IL CODICE

```
/**
		Classe MyAnimClip
		autore: Bruno Migliaretti
		versione: 1.0
		data modifica: 01/12/2006
		copyright: Accademia di Belle Arti di Urbino
		Questo codice aggiunge il metodo moveTo alla classe MovieClip.
*/
class MyAnimClip extends MovieClip {
		private var totalTime:Number;
		private var start_x:Number;
		private var start_y:Number;
		private var x_delta:Number;
		private var y_delta:Number;
		private var startTime:Number;
		public var speed:Number = 100;

		public function  moveTo (x:Number,y:Number) {
				start_x = this._x;
				start_y = this._y;
				x_delta = x - this._x;
				y_delta = y - this._y;

				totalTime = (Math.sqrt((x_delta * x_delta) + (y_delta * y_delta))/speed) * 1000;

				startTime = getTimer();

				this.onEnterFrame = function() {
						var t:Number = getTimer() - startTime;
						if (t > totalTime) {
								t = totalTime;
						}
						var t_delta:Number = t/totalTime;
						this._x = start_x + (x_delta * t_delta);
						this._y = start_y + (y_delta * t_delta);
						if (t == totalTime) {
								delete this.onEnterFrame;
						}
				}
		}
}
```

CARICAMENTO DI CONTENUTI ESTERNI

TIPO DI CONTENUTI MULTIMEDIALI

- In Flash posso caricare contenuti multimediali presenti su disco. A secondo dei contenuti utilizzerò oggetti e sistemi di controllo diversi:
  - Filmati Flash
  - Immagini di tipo bitmap
  - File audio
  - Video

IMMAGINI BITMAP

- Le immagini bitmap vengono caricati utilizzando la classe MovieClip normalmente utilizzando MovieClipLoader per controllare il caricamento
- Flash Player versione 7 è in grado di visualizzare solo immagini in formato jpeg non interallacciate
- Flash Player versione 8 è in grado di visualizzare immagini in formato jpeg interallaciato e non interallacciate, immagini in formato GIF (con eventuale trasparenza) e immagini in formato PNG (con eventuale trasparenza)

FILMATI FLASH

- I filmati flash vengono caricati utilizzando la classe MovieClip.
- Possono essere ottimizzati:
  - Verificare la qualità del sonoro
  - Verificare la qualità delle immagini bitmap
  - Trasformare gli oggetti vettoriali molto complicati in immagini
- Per default i filmati Flash sono eseguiti progressive downloading, cioè l'esecuzione inizia non appena caricato il primo frame.
- Si può simulare la creazione di un buffer bloccandoli sul primo frame e facendoli ripartire dopo una certa percentuale di caricamento.

FILMATI FLASH

- Inserire uno stop() nel primo frame
- Caricare il filmato utilizzando il metodo **MovieClip.loadMovie**()
- Controllare, usando i metodi **MovieClip.getBytesLoaded**() e **MovieClip.getBytesTotal**() lo stato del caricamento.
- Una volta raggiunta la percentuale di caricamento voluto fare partire il filmato con il metodo **MovieClip.play**():

FILE AUDIO

- Oggetto Sound
- File in formato MP3 cbr (constant bit rate)
- Calcolo del bit rate (vale anche per il video):
  - (Bytes * 8) / durata (in secondi).

FILE VIDEO

- Oggetti Video e NetConnection
- File in formato **FLV** (FLash Video)
- Codecs Sorenson Spark (flash 6 e 7) e On2 VP6 (solo flash 8).
- Calcolo del bit rate:
  - (Bytes * 8) / durata (in secondi).