LEZIONE 6

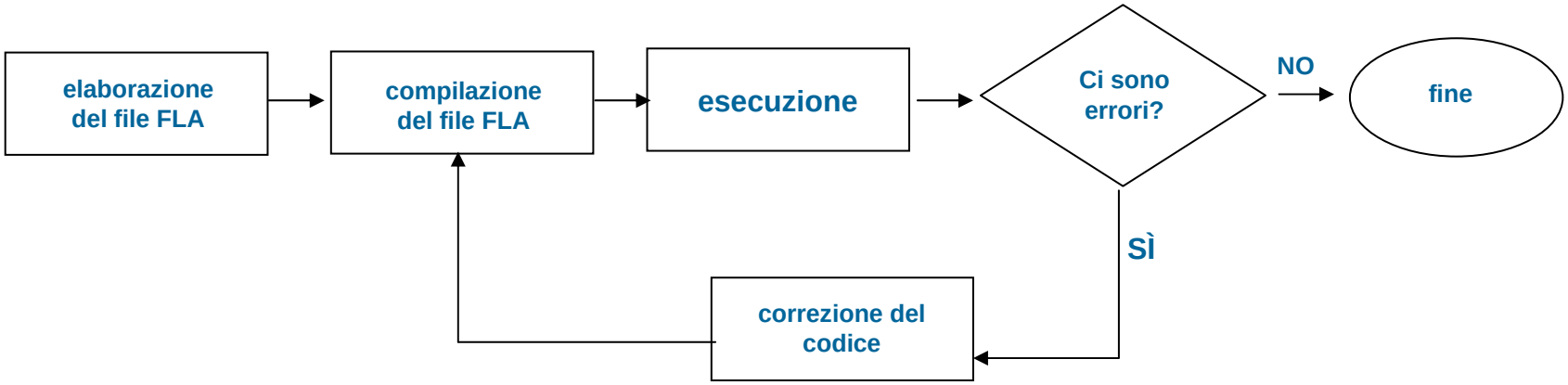
CREAZIONE DI CLASSI

LE CLASSI

- Nella programmazione orientata agli oggetti una classe definisce una *categoria di oggetto*.
- Le classi in pratica sono nuovi tipi di dati che il programmatore può creare per definire un nuovo tipo di oggetto. Una classe descrive le *proprietà* (*dati*) e i *metodi* (*comportamenti*) di un oggetto in modo molto simile a come un progetto di architettura descrive le caratteristiche di un edificio.
- Le proprietà (variabili definite all'interno di una classe) e i metodi (funzioni definite all'interno di una classe) di una classe sono detti *membri* della classe.
- In generale per utilizzare le proprietà e i metodi definiti da una classe, è necessario creare prima un'istanza di tale classe. La relazione tra un'istanza e la relativa classe è simile a quella che intercorre tra un edificio e il relativo progetto.

IL PROCESSO

- Come abbiamo gia detto il processo di realizzazione di un progetto flash è strutturata in un flusso simile a questo:



- Con la programmazione con le classi introduciamo un nuovo tipo di file: il file **Action Script (.as)** ma il processo rimane lo stesso.
- Se il progetto comprende anche dei file .as il compilatore li unirà ai dati contenuti nel .fla e creerà un unico file shockwave (filmato flash compilato e compresso).

I FILE DI CLASSE

- Il file .as è un normale file di testo (tipo blocco note) che contiene codice Action Script.
- Una classe in *ActionScript* viene sempre definita in un file esterno (un normale file di testo con estensione *.as*) che ha lo stesso nome della classe e che viene chiamato *file di classe*.
- Quando un filmato flash viene compilato (utilizzando Controllo > Prova filmato o File > Pubblica) per generare il file *.swf*, il codice contenuto nei file di classe necessari viene compilato e aggiunto al file *.swf*..

I FILE DI CLASSE

- Quando il compilatore trova che nel filmato da compilare viene utilizzata una classe **DEVE** trovare il file che contiene il codice relativo a quella classe:
- Il file di classe deve avere esattamente lo stesso nome della classe (case sensitive).
- Il compilatore deve sapere in che cartelle cercare.

I FILE DI CLASSE

1. Esiste un elenco globale di cartelle che contengono classi che si può modificare andando in:
   Modifica>Preferenze>ActionScript
   e scegliendo il bottone “Impostazioni Action Script 2
2. Il compilatore oltre che nelle cartelle globali cerca anche nella cartella in cui è stato salvato il file .fla.

LE CLASSI AGGIUNTIVE

- Per default l'elenco globale delle cartelle contiene la cartella in cui si trovano le classi aggiuntive fornite con flash 8 (filtri grafici, funzioni geometriche, ecc)
- Queste classi sono organizzate in sottocartelle.
- In java e in Action Script le sottocartelle in cui sono organizzate le classi si chiamano **pakages** (pacchetti).

I PACCHETTI

- L’uso dei pacchetti serve esclusivamente ad organizzare il meglio il mio lavoro. Nella mia vita di programmatore svilupperò sempre nuovi progetti e presumibilmente creerò nuove classi, che per loro natura sono codice riciclabile, utilizzabile cioè in nuovi progetti.
- Se organizzerò queste classi in gruppi sarà più facile per me ritrovarle e riutilizzarle.

I PACCHETTI

- Per identificare una classe in un pacchetto (o in un sottopacchetto inserito in un pacchetto) devo usare la sintassi del punto.
- Per utilizzare la classe **BlurFilter** dovro scivere:

```
// senza importazione
var myBlur:flash.filters.BlurFilter =
  new flash.filters.BlurFilter(10,10,3);
```

IL COMANDO IMPORT

- Per semplificare la scrittura del codice posso usare il comando **import**.
- Utilizzando il comando eviterò di dovere scrivere sempre l'intero percorso della classe:

```
// con importazione
import flash.filters.BlurFilter
var myBlur:BlurFilter = new BlurFilter(10,10,3);
```

CREAZIONE DI CLASSI

- Per definire una classe devo creare un file action script esterno. Vediamo di definire le regole (alcune obbligatorie, altre solo consigliate) che devo seguire nello scrivere un file di classe saranno presenti questi elementi:
  - Il file di classe inizierà con commenti di documentazione, tra cui una descrizione generale del codice e informazioni sull'autore e sulla versione.
  - Seguiranno le istruzioni import (se necessarie).
  - Scriverò poi la dichiarazione di classe come segue:

```
class User {
...
}
```

CREAZIONE DI CLASSI

- Inserirò gli eventuali commenti relativi all'implementazione della classe.
- Dichiarerò, quindi, le variabili statiche.
- Dichiarerò le variabili di istanza rispettando questo ordine: prima le variabili pubbliche, poi quelle private.
- Aggiungerò l'istruzione relativa alla funzione di costruzione, come indicato nell'esempio seguente:

```
public function User(username:String,
password:String)
{
  ...
}
```

CREAZIONE DI CLASSI

- Scriverò i metodi, raggruppandoli per funzionalità. Questo tipo di organizzazione dei metodi consente di migliorare la leggibilità e la chiarezza del codice.
- Scriverò i metodi getter/setter.

LA CLASSE USER

```
/*
   Classe User
   autore: John Doe
   versione: 0.8
   data modifica: 08/21/2005
   copyright: Macromedia, Inc.
   Questo codice definisce una classe User personalizzata per
   Creare nuovi utenti e specificare le relative informazioni di login.
*/
class User {
   // Variabili di istanza private
   private var __username:String;
   private var __password:String;
   // Funzione di costruzione
   public function User(p_username:String, p_password:String) {
            this.__username = p_username;
            this.__password = p_password;
   }
   public function get username():String {
            return this.__username;
   }
   public function set username(value:String):Void {
            this.__username = value;
   }
   public function get password():String {
            return this.__password;
   }
   public function set password(value:String):Void {
            this.__password = value;
   }
}
```

LA CLASSE USER

- Un *commento di documentazione* standard che specifica il nome della classe, l'autore, la versione, la data di modifica, le informazioni di copyright e una breve descrizione dello scopo della classe. Scriverò i metodi getter/setter.
- funzione di costruzione della classe User
- le proprietà getter e setter per le variabili di istanza private

LA CLASSE STUDENT

PROVARE UNA CLASSE

- Per creare e usare una classe è necessario:
  - Definizione di una classe in un file di classe *ActionScript* esterno.
  - Salvataggio del file di classe nella directory specificata per il percorso della classe (o nel percorso in cui Flash cerca le classi) oppure nella stessa directory del file FLA dell'applicazione.
  - Creazione di un'istanza della classe in un altro script, ossia un documento FLA o un file di script esterno, oppure tramite creazione di una sottoclasse basata sulla classe originale.

USARE UNA CLASSE

- Per creare un'istanza di una classe *ActionScript*, si utilizza l'operatore ***new*** per richiamare la funzione di costruzione della classe. Tale funzione ha sempre lo stesso nome della classe e restituisce un'istanza della classe che generalmente viene assegnata a una variabile.

```
var firstUser:User = new User();
```

- Usando l'operatore punto (.) si accede al valore di una proprietà di un'istanza.

```
firstUser.username;
```

PROPRIETÀ E METODI PUBBLICI E PRIVATI

- La parola chiave *public* specifica che una variabile o una funzione è disponibile per qualsiasi chiamante. Poiché le variabili e le funzioni sono pubbliche per impostazione predefinita, la parola chiave *public* viene utilizzata principalmente per ragioni di stile e leggibilità.
- La parola chiave *private* specifica che una variabile o una funzione è disponibile solo per la classe che la dichiara o la definisce o per le relative sottoclassi.

METODI E PROPRIETÀ STATICI

- La parola chiave *static* specifica che una variabile o una funzione viene creata solo una volta per ogni classe anziché in ogni oggetto basato sulla classe. È possibile accedere a un membro di classe statico senza creare un'istanza della classe. I metodi e le proprietà statici possono essere sia pubblici che privati.
- In ActionScript ci sono classi predefinite che hanno solo metodi e proprietà statiche.

METODI GETTER E SETTER

- I metodi getter e setter sono un modo alternativo per definire una proprietà. Quando creo una proprietà definendo una variabile pubblica all'interno della classe consento a chi usa la classe di interagire direttamente con la variabile. Quando leggerà la proprietà leggerà il contenuto della variabile, quando la modificherà il contenuto della variabile.
- Scrivere metodi setter e getter consente di controllare quanto viene scritto (o letto) ed eventualmente modificarlo.
- La sintassi dei metodi getter e setter è la seguente:
  - Un metodo getter non accetta parametri e restituisce sempre un valore.
  - Un metodo setter accetta sempre un parametro e non restituisce mai valori
- Per definire tali metodi, utilizzare gli attributi dei metodi ***get*** e ***set***. I metodi creati ottengono o impostano il valore di una proprietà e aggiungono la parola chiave *get* o *set* prima del nome del metodo

CREAZIONE DELLA FUNZIONE DI COSTRUZIONE

- Per funzioni di costruzione si intendono funzioni che consentono di inizializzare (*definire*) le proprietà e i metodi di una classe. Per definizione, le funzioni di costruzione sono funzioni incluse in una definizione di classe, con la stessa denominazione della classe
- La funzione di costruzione di una classe è una funzione speciale che viene chiamata quando si crea un'istanza di una classe utilizzando l'operatore new e presenta lo stesso nome della classe che la contiene.

LA CLASSE MOVIECLIP

LA CLASSE MOVIECLIP

- La classe MovieClip è l'oggetto software che ci consente di gestire gli elementi visuali in Flash da ActionScript.
- Attraverso la classe Movie Clip posso gestire:
  - MovieClip creati runtime utilizzando ActionScript.
  - Simboli di tipo MovieClip memorizzati in libreria e trascinati durante lo sviluppo sullo stage.
  - Simboli di tipo MovieClip memorizzati in libreria e caricati sullo stage in fase runtime.
  - Filmati memorizzati su disco e caricati runtime all'interno del filmato principale.
  - Immagini bitmap (formato jpeg) memorizzate su disco.

LA CLASSE MOVIECLIP

- La classe MovieClip normalmente NON si crea utilizzando il comando **new**.
- Quando trascino un Simbolo di tipo MovieClip memorizzato in libreria sullo stage durante lo sviluppo automaticamente creo un oggetto MovieClip che è istanza del simbolo in libreria.
- Posso creare MovieClip runtime all'interno di altre MovieClip applicando il metodo **CreateEmptyMovieClip().**
- Posso inserire istanze di simboli di tipo MovieClip memorizzati in libreria in altre MovieClip applicando il metodo **AttachMovieClip().**
- Posso caricare filmati o immagini bitmap memorizzati su disco sostituendo il contenute di MovieClip esistenti o creandone di nuove.

MOVIECLIP CREATA DESIGN TIME

- Quando trascino un simbolo MovieClip sullo stage creo un nuova istanza di quel simbolo.
- Se a quella istanza design time do un nome, attraverso quel nome posso modificare le proprietà di quella istanza e applicare ad essa i metodi di MovieClip.
- Intermini di Programmazioone ad Oggetti quando creo un simbolo MovieClip creo un modello di filmato che condivide con la classe MovieClip tutte le proprietà e tutti i metodi, e che aggiunge alla MovieClip astratta i contenuti visuali presenti.
- L'istanza creata appartiene alla MovieClip (film principale o secondario) entro alla quale è stata inserita. Programmaticamente la si puo raggiungere utilizzando indifferentemente la sintassi degli Oggetti **contenitore.nomeistanza** che la sintassi degli array associativi **contenitore["nomeistanza"]**

MODIFICA DELLA POSIZIONE E DELL'ASPETTO DI UN CLIP FILMATO

- Per modificare le proprietà di un clip filmato durante l'esecuzione, creare un'istruzione che assegna un valore a una proprietà.
- Le proprietà _x, _y, _rotation, _xscale, _yscale, _height, _width, _alpha e _visible determinao l’aspetto geometrico del clip filmato di tutti gli elementi in esso contenuto.
- Le seguenti proprità sono a sola lettura: _currentframe, _droptarget, _framesloaded, _parent, _target, _totalframes, _url, _xmouse e _ymouse.

TRASCINAMENTO DI CLIP FILMATO

- Applicando ad un Movie Clip il metodo **startDrag()** si rende il clip filmato trascinabile.
- In questo modo è possobile il trasinare oggetti in giochi, barre di scorrimento e cursori.
- Un clip rimane trascinabile fino a quando il trascinamento non viene interrotto esplicitamente mediante il metodo **stopDrag()** o fino a quando il trascinamento non viene applicato ad un altro clip filmato mediante startDrag().
- È possibile trascinare in un solo clip filmato alla volta.

CREAZIONE DI CLIP FILMATO IN FASE DI RUNTIME

- Le istanze di clip filmato possono essere realizzate non solo nell'ambiente di creazione di Flash, ma anche in fase di runtime.
- Ogni istanza di clip filmato creata in fase di runtime deve avere:
  - un nome di istanza unico nel MovieClip in cui è contenuto
  - un valore di profondità (*z*-order) ugualmente unico
- La profondità specificata determina in che ordine il nuovo clip si sovrappone agli altri clip sulla stessa linea temporale.
- Per ogni livello di profondità avremo un unico clip. Creando un clip ad un livello già occupato da un altro clip quest'ultimo viene sostituito dal nuovo clip creato.

CREAZIONE DI CLIP FILMATO IN FASE DI RUNTIME

- Posso creare un nuovo MovieClip solo in MovieClip già esitente. Il MovieClip che contiene il MovieClip creato (runtime o design time) viene indicato dalla proprietà **_parent** del MovieClip.
- Ho tre metodi a disposizione per creare un nuovo MovieClip in un MovieClip esistente:
  - **MovieClip.createEmptyMovieClip()**
  - **MovieClip.attachMovie()**
  - **MovieClip.duplicateMovieClip()**

CREAZIONE DI UN CLIP FILMATO VUOTO

- Per creare una nuova istanza di clip filmato vuota sullo stage, utilizzare il metodo createEmptyMovieClip() della classe MovieClip. Questo metodo crea un clip filmato secondario del clip che richiama il metodo.

```
parent_mc.createEmptyMovieClip("new_mc", 10);
```

- Il seguente codice crea un nuovo clip filmato denominato canvas_mc nella linea temporale principale del file SWF in cui viene eseguito lo script e quindi chiama loadMovie() per caricare un file JPEG esterno.

```
this.createEmptyMovieClip("canvas_mc", 10);
canvas_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
```

DUPLICAZIONE O RIMOZIONE DI UN CLIP FILMATO

- Per duplicare o rimuovere istanze di clip filmato, utilizzare metodi della classe MovieClip **duplicateMovieClip()** o **removeMovieClip()**.
- Il metodo duplicateMovieClip() crea una nuova istanza da un'istanza di clip filmato esistente e le assegna un nuovo nome di istanza e una profondità, o *z*-order.
- Per eliminare un clip filmato creato con duplicateMovieClip(), utilizzare **removeMovieClip().**

ATTACH MOVIE

- Un ultimo modo per creare istanze di clip filmato in fase di runtime consiste nell'utilizzare il metodo **attachMovie()**. Il metodo attachMovie() associa allo stage un'istanza di un simbolo di tipo clip filmato nella libreria. Il nuovo clip viene inserito nel MovieClip a cui è stato applicato il metodo.
- Per poterlo utilizzare da ActionScript il simbolo deve essere esportatp nel file swf.
- Per eseguire questa operazione, utilizzare la finestra di dialogo Proprietà di concatenamento.
- Per impostazione predefinita, tutti i clip filmato esportati per l'utilizzo in ActionScript vengono caricati prima del fotogramma iniziale del file SWF che li contiene.

AGGIUNTA DI PARAMETRI A CLIP FILMATO CREATI DINAMICAMENTE

- Se si crea o si duplica dinamicamente un clip filmato utilizzando MovieClip.attachMovie() e MovieClip.duplicateMovie(), è possibile assegnare un valore iniziale ad alcune proprietà.
- Il metodo accetto un parametro opzionale di tipo Object attreaverso il quale posso inizializzare ilclip filmato creati in modo dinamico.

GESTIONE DELLE PROFONDITÀ NEI CLIP FILMATO

- Ogni clip filmato ha un proprio spazio *z*-order che determina in che modo gli oggetti si sovrappongono all'interno del file SWF o del clip filmato di origine. A ciascun clip filmato è associato un valore di profondità che determina se il filmato si trova in primo piano o dietro altri clip filmato della stessa linea temporale.
- Quando si crea un clip filmato in fase di runtime attraverso attachMovie, duplicateMovieClip o createEmptyMovieClip, occorre sempre specificare una profondità per il nuovo clip come parametro del metodo.
- Se si crea o si associa un nuovo clip filmato a una profondità in cui è già presente un altro clip filmato, il nuovo clip o il clip associato sovrascrive il contenuto esistente. Per evitare l'insorgere di questo problema, utilizzare il metodo **MovieClip.getNextHighestDepth**();
- Il valore intero restituito da questo metodo indica la successiva profondità disponibile che determina la posizione in primo piano rispetto a tutti gli altri oggetti del clip filmato.

CARICAMENTO DI UN FILMATO O DI UN IMMAGINE DA DISCO

- Per caricare dinamicamente un filmato (.swf) o un'immagine jpeg (.jpg) da disco ho a disposizione tre strategie:
  - Caricare il contenuto al posto del filmato principale
  - Caricare il contenuto nel filmato principale su un altro level
  - Utilizzare una MovieClip per caricare il contenuto esterno.
- L'utilizzo di una MovieClip mi dà poi due ulteriori alternative:
  - Utilizzazione del metodo MovieClip.loadMovie
  - Utilizzo dell'oggetto MovieClipLoader.

SOSTITUZIONE DEI CONTENUTI DEL FILMATO PRINCIPALE

- Se semplicemente voglio sostituire il contenuto del mio filmato principale con un altro filmato presente su disco posso usare il comando globale loadMovie() che accetta come parametro l'url del filmato da caricare.
- Devo tener presente che:
  - Tutti i contenuti presenti nello stage
  - Tutti i contenuti presenti nella libreria del filmato
  - Tutte le funzioni e le classi definite nel filmato

  andranno persi.

UTILIZZO DEI LIVELLI

- Se invece voglio aggiungere dei contenuti al filmato principale o sostituirne una parte posso utilizzare i livelli o le movie clip.
- Per utilizzare i livelli userò il comando loadMovieNum(url, livello)
- I contenuti verranno caricati nel livello specificato nel parametro
- Il livello è un'oggetto di tipo MovieClip che viene creato automaticamete (se già non esiste) e che si chiama _level*n*

UTILIZZO DI UNA MOVIECLIP

- Un'alternativa ai livelli (molto più flessibile e da noi preferita) è l'utilizzo di una MovieClip gia presente nel filmato o creata appositamente.
- Basta applicare il metodo loadMovie() alla MovieClip.
- Se la clip è vuota, dopo il caricamento il suo contenuto sarà costituito dal filamto o dall'immagine scaricata
- Se ha del contenuto il contenuto sarà sostituito dal contenuto caricato.

UTILIZZO DI MOVIECLIPLOADER

- MovieClipLoader è una classe presente in flash dalla versione 7 in poi molto utile per il controllo del caricamento di contenuti in un filmato.
- Come per tutte le classi prima di tutto dovrò creare un'istanza della per utilizzarla:

```
var myLoader:MovieClipLoader = new MovieClipLoader();
```

- Dopo di che potrò applicare all'istanza creata il metodo loadClip(url, movieClip) che mi consente di caricare un contenuto esterno in una movie clip

```
myLoader.loadClip("unfilmato.swf", my_clip_mc);
```

UTILIZZO DI MOVIECLIPLOADER

- Il vantaggio rispetto all'uso diretto di MovieClip è ch l'oggetto mi fornische 5 gestori eventi che mi consentono un pieno controllo del caricameto:
  - onLoadStart che mi comunica che il caricamento è iniziato
  - onLoadProgress che mi comunica continuamente lo stato del caricamento
  - onLoadComplete che mi dice che il caricamento è completato
  - onLoadInit che mi comunica che, terminato il caricamento, il filamto è stato inizializzato e quindi i suoi metodi e le sue proprietà sono disponibili
  - onLoadError che mi segnala errori

GESTIONE DEGLI EVENTI NEI CLIP FILMATO

- Un metodo di un gestore di eventi è un metodo di una classe che viene richiamato quando si verifica un evento su un'istanza di tale classe.
- La classe MovieClip, ad esempio, definisce numerosi gestori di eventi che vengono richiamati ogni volta che l'evento in questione si verifica.
- Gestire un evento significa scrivere una funzione anonima che contenga il codice da eseguire quando l'evento si verifica.
- Un gestore di eventi è costituito da tre elementi:
  - l'oggetto al quale si applica l'evento,
  - il nome del metodo del gestore di eventi dell'oggetto
  - la funzione assegnata al gestore di eventi.
- Ecco la struttura di base di un gestore di eventi:

```
object.eventMethod = function () {
// Inserire qui il codice che risponde all'evento.
}
```