LE CLASSI

PROGRAMMAZIONE OOP

- Rivoluzione copernicana
  - L'approccio procedurale definisce i passi necessari per risolvere un problema
  - L'approccio ad oggetti definisce gli strumenti (oggetti) di cui ho bisogno per risolvere un problema e ne definisce le proprietà e i comportamenti.

LE PAROLE CHIAVE DELLA PROGRAMMAZIONE OOP

OGGETTI

- Un oggetto del mondo reale, ad esempio un gatto, possiede delle *proprietà* (o stati), quali nome, età e colore, e presenta delle caratteristiche comportamentali, ad esempio dormire, mangiare e fare le fusa.
- Analogamente, nel mondo della programmazione orientata agli oggetti, gli oggetti presentano proprietà e comportamenti.
- Utilizzando le tecniche orientate agli oggetti, è possibile modellare un oggetto del mondo reale (come un gatto) per utilizzarlo, ad esempio, come protagonista di un videogioco oppure un oggetto più astratto (come, ad esempio, un processo chimico).

ISTANZE E MEMBRI DI CLASSE

- Nel mondo reale, è possibile considerare che esistono gatti di colore, età e nomi differenti che allo stesso tempo presentano modi diversi di mangiare o di fare le fusa. Tuttavia, indipendentemente dalle differenze tra i singoli animali, tutti i gatti appartengono alla stessa categoria che, in termini di programmazione orientata agli oggetti, è detta classe. Appartengono cioè alla classe Gatto. Nella terminologia orientata agli oggetti,ogni singolo gatto viene considerato un'*istanza* della classe Gatto.
- Una classe definisce un modello per un tipo di oggetto. Tutte le caratteristiche e i comportamenti che appartengono a una classe sono detti *membri* di tale classe. In particolare, le caratteristiche (nell'esempio del gatto, il nome, l'età e il colore) sono dette *proprietà* della classe e sono rappresentate da variabili, mentre i comportamenti (mangiare, dormire) sono detti *metodi* della classe e sono rappresentati da funzioni.

EREDITARIETÀ

- Uno dei vantaggi principali della programmazione orientata agli oggetti è rappresentato dalla creazione di **sottoclassi** (tramite estensione di una classe); una **sottoclasse** eredita tutte le proprietà e i metodi della rispettiva classe. La sottoclasse definisce generalmente ulteriori metodi e proprietà o sostituisce metodi o proprietà definiti nella superclasse. Le sottoclassi possono inoltre sostituire i metodi definiti in una superclasse, ovvero fornire definizioni proprie per tali metodi.

EREDITARIETÀ

MAMMIFERI

GATTO

SIAMESE

INTERFACCE

- Le interfacce nella programmazione orientata agli oggetti possono essere descritte come modelli di definizioni di classe; le classi che implementano le interfacce sono necessarie per implementare quel modello di metodo.
- Usando l'analogia del gatto, un'interfaccia è simile al progetto di un gatto: Il progetto evidenzia quali parti sono necessarie, ma non necessariamente come vengono assemblate le parti o qual è il funzionamento delle parti.

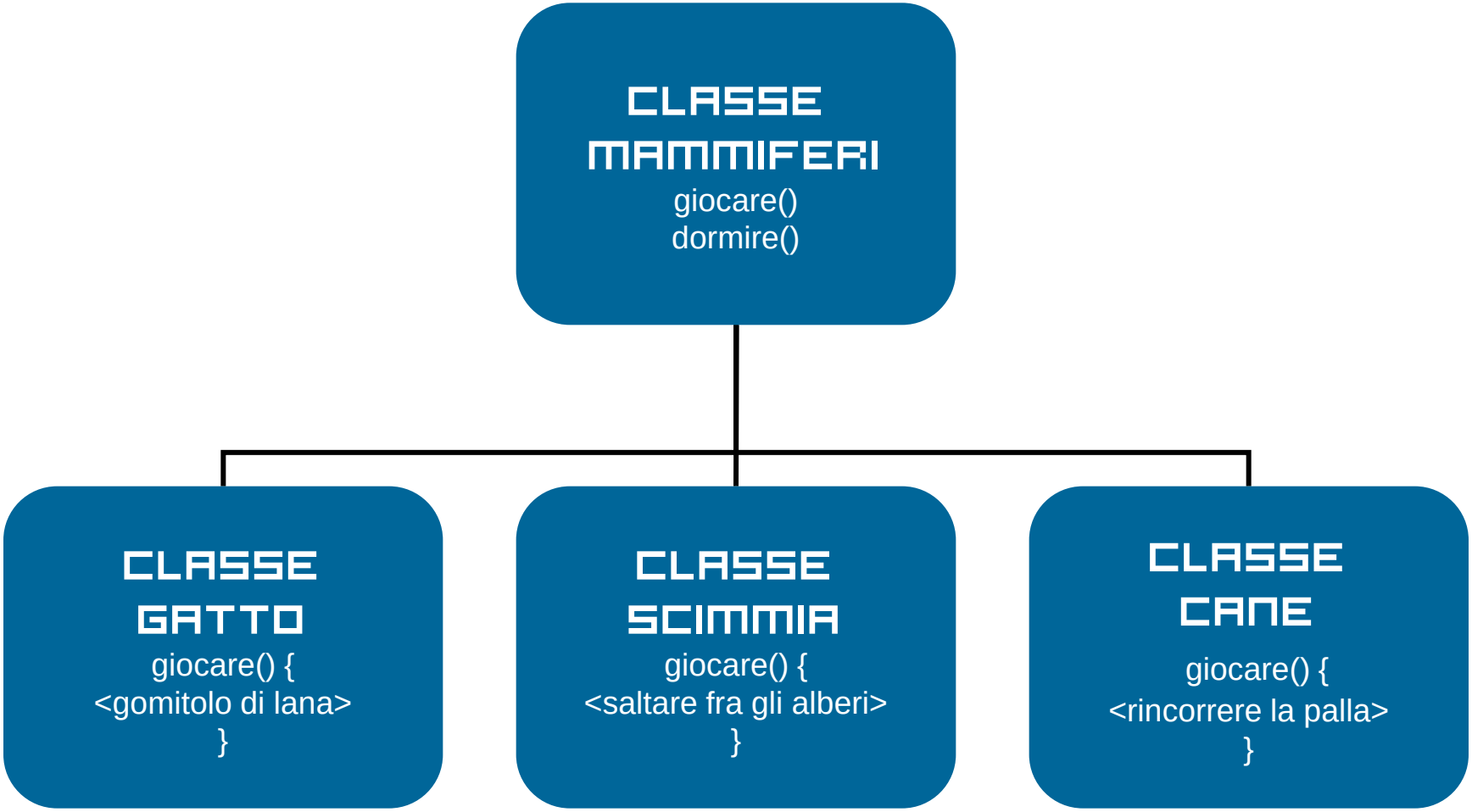
INCAPSULAMENTO

- Nella progettazione orientata agli oggetti, gli oggetti sono considerati scatole nere che contengono o *incapsulano* funzionalità.
- Un programmatore dovrebbe essere in grado di interagire con un oggetto conoscendone solo le proprietà, i metodi e gli eventi (l'interfaccia di programmazione), ma senza conoscerne i dettagli di implementazione.

POLIMORFISMO

- La programmazione orientata agli oggetti permette di esprimere differenze tra le singole classi con l'ausilio di una tecnica detta ***polimorfismo***, grazie alla quale le classi possono sostituire i metodi delle relative superclassi e definire implementazioni specializzate di tali metodi. .

POLIMORFISMO



CLASSI E TIPI

- Quando abbiamo parlato dei tipi di dati abbiamo visti che esistono ***tipi di dati primitivi*** e ***tipi di dati derivati*** *o* ***complessi***.
- In linea di principio in un linguaggio orientato agli oggetto tutti i tipi di dati sono classi.
- Creando una nuova classe il programmatore crea un nuovo ***tipo di dati complesso***, un ***modello,*** cioè, delle proprietà e delle caratteristiche che ha un nuovo tipo di dato.

LE CLASSI IN ACTION SCRIPT

LE CLASSI INCORPORATE

- Abbiamo visto che nei linguaggi OOP le classi servono a rappresentare ***tipi di dati complessi***.
- Per gestire dati come le date, gli array, ecc Action Script fornisce un certo numero di classi incorporate nel linguaggio.
- Prima di passare alla programmazione di classi personalizzate vedremo di familiarizzare con il concetto di classe imparando ad utilizzare le classi incorporate in ActionScript.

LE CLASSI INCORPORATE

- *ActionScript* comprende numerose classi incorporate che servono a gestire diversi tipi di elementi: tipi di dati di base quali Array, Boolean, Date, gestione degli errori, gestione degli eventi, caricamento di contenuto esterno (XML, immagini, dati binari originari, ecc.).

LE CLASSI INCORPORATE

- Prima di passare all'uso delle classi è bene citare alcune regole di scrittura che ActionScript segue e alle quali, anche se non è sempre obbligatorio è bene adeguarsi:
  - i nomi delle variabili, delle proprietà, delle funzioni e dei metodi iniziano sempre con una lettera minuscola o, a volte, con "_" (underscore),
  - i nomi dei tipi di dati e, quindi, delle classi sempre con una lettera maiuscola.

COME USARE UNA CLASSE

- Nella maggior parte dei casi utilizzare una classe significa
  - creare un'istanza della classe stessa,
  - assegnare l'istanza creata ad una variabile e
  - usarne le proprietà e applicarne i metodi per ottenere i risultati desiderati.
- Per creare un'istanza di una classe devo applicare l'operatore ***new*** al ***costructor***, cioè alla funzione di costruzione della classe, una speciale funzione che ha lo stesso nome della classe.

ESEMPIO

```
// dichiaro “now” come variabile di tipo Date
var now:Date;
// assegno a now una istanza di Date che corrisponde alla
// data e all’ora corrente
now = new Date();
```

Dopo questa operazione now conterrà un’istanza della classe Date che rappresenta la data e l’ora corrente. A now si potranno applicare tutti i metodi e usare le proprietà della classe Date.

Da notare l’uso diverso di ***Date*** in ***var now:Date;*** e in ***now = new Date();***.
Nel primo caso ***Date*** indica il ***tipo Date*** e il costrutto dice al compilatore che la variabile now potrà contenere solo oggetti di tipo Date. Nel secondo caso **Date()** (seguito da parentesi) indica la funzione di costruzione della classe Date e il costrutto ***now = new Date()*** crea una nuova istanza della classe Date() e la memorizza in now.

ECCEZIONI

- L'operatore *new* non deve essere usato per i tipi primitivi: *Number*, *String*, *Boolean* (che comunque sono classi a tutti gli effetti, con le loro proprietà e i loro metodi). Le istanze delle classi sono create automaticamente quando maneggio questo tipo di dati.
- Ci sono delle classi per cui non si possono creare istanze e che rappresentano in quanto tali un oggetto su cui operare: *Accessibility*, *Camera*, *ContextMenu*, *CustmActions*, *Key*, *Math*, *Microphone*, *Mouse*, *Selection*, *SharedObject*, *Stage*, *System*. In questi casi proprietà e metodi si applicano direttamente alla classe e vengono detti *statici*.

LA CLASSE OBJECT

OBJECT

- Il tipo di dati ***Object*** rappresenta l'astrazione dell'idea di oggetto. Sta alla radice della gerarchia delle classi: tutte le classi sono estensioni di ***Object***.
- La classe *Object* mi consente di creare oggetti vuoti, senza proprietà né metodi.
- Creando un'istanza di *Object* creo un oggetto personalizzato, a cui posso aggiungere le proprietà che voglio e che mi può servire per organizzare le informazioni nell'applicazione Flash.

ESEMPIO

```
var user:Object = new Object();
user.name = "Irving";
user.age = 32;
user.phone = "555-1234";
```

Viene creato un nuovo oggetto denominato user e tre proprietà: name, age e phone che sono tipi di dati String e Numeric.
Lo stesso oggetto può essere creato anche assegnando alla variabile il letterale di tipo *Object* corrispondente.

```
var user:Object;
user = {name:”Irving”,age:32,phone:”555-1234”};
```

Quando si assegna ad una variabile un valore in formato letterale non è necessario richiamare il costruttore della classe con l’operatore *new*. Queato vale sia per ***Object*** che per ***Array.***

for .. in

- Un ciclo **for..in** consente di eseguire iterazioni scorrendo gli elementi di un ogetto (un clip filmato, un oggetto generico, o un array).
- La struttura del comando è:

```
for variabile in oggetto
    blocco istruzioni;
```

- Il ciclo prende in esame tutti gli elementi presenti in <oggetto>. Ad ogni ciclo <variabile> assume il nome dell'elemento preso in esame e <blocco istruzioni> viene eseguito.

LA CLASSE ARRAY

ARRAY

- Un *array* è un oggetto le cui proprietà sono identificate da un numero (indice) che ne rappresenta la posizione nella struttura dell'array.
- Un array è un elenco di elementi recuperabile attraverso un indice.
- Non occorre che gli elementi dell'array abbiano lo stesso tipo di dati. È possibile inserire numeri, dati, stringhe, oggetti, array.

USO DEGLI ARRAY

- Gli array possono essere utilizzati in modi diversi.
- Uno degli utilizzi più tipici è quello di organizzare i dati di un database sotto forma di array di oggetti.
- La posizione di un elemento nell'array è detta *indice*. Tutti gli array sono con base zero, ovvero [0] è il primo elemento dell'array, [1] è il secondo, e così via.
- Normalmente i contenuti di un array vengono esaminati utilizzando un ciclo for che consente di scorrere tutti gli elementi di un array.

MODIFICA DI UN ARRAY

L'array può essere controllato e modificato tramite ***ActionScript***. È possibile spostare valori all'interno dell'array o modificarne la dimensione. Il seguente codice, ad esempio, scambia i dati di due indici di un array:

```
var buildingArr:Array = new Array();
buildingArr[2] = "Accounting";
buildingArr[4] = "Engineering";
trace(buildingArr);
//undefined,undefined,Accounting,undefined,Engineering
var temp_item:String = buildingArr[2];
buildingArr[2] = buildingArr[4];
buildingArr[4] = temp_item;
trace(buildingArr);
//undefined,undefined,Engineering,undefined,Accounting
```

Nell'esempio precedente è necessario creare una variabile temporanea perché se il contenuto dell'indice 4 dell'array fosse stato copiato nell'indice 2 dell'array senza salvarne prima il contenuto, il contenuto originale dell'indice 2 sarebbe andato perso.

LUNGHEZZA DI UN ARRAY

- Quando si lavora con gli array, è spesso necessario conoscere il numero degli elementi contenuti in un array, in particolare se si scrivono cicli for che eseguono iterazioni su ogni elemento dell'array ed eseguono una serie di istruzioni. La proprietà ***length*** restituisce la lunghezza dell'array.

AGGIUNTA E RIMOZIONE

- Un array contiene elementi e ogni elemento ha una posizione numerica (l'indice) che corrisponde alla modalità con cui si fa riferimento alla posizione di ogni elemento nell'array.
- Ogni elemento può contenere dati o essere vuoto. I dati contenuti possono essere del formato numerico, stringa, booleano o essere un array o un oggetto.
- Se si assegna un solo valore in un array all'indice 5, la lunghezza dell'array restituisce 6. Nell'array vengono quindi inseriti cinque valori non definiti.
- Possono essere aggiunti elementi in coda all’array utilizzando il metodo ***push().***

ARRAY ASSOCIATIVI

- Un array associativo è composto da chiavi e valori non ordinati e utilizza le chiavi alfanumeriche al posto degli indici numerici per organizzare i valori.
- Ogni chiave è una stringa univoca e viene utilizzata per accedere al valore a cui è associata. Il valore può essere un tipo di dati Number, Array, Object e così via.
- Array associativi e Object rappresentano due modi diversi per rappresentare gli stessi dati e sono intercambiabili.

```
// Definisce l'oggetto da utilizzare come array associativo
var someObj:Object = new Object();
// Definisce una serie di proprietà
someObj.myShape = "Rectangle";
someObj.myW = 480;
someObj.myH = 360;
someObj.myX = 100;
someObj.myY = 200;
someObj.myAlpha = 72;
someObj.myColor = 0xDFDFDF;
// Visualizza una proprietà utilizzando l'operatore punto e
// la sintassi di accesso agli array
trace(someObj.myAlpha); // 72
trace(someObj["myShape"]); // 72
```

LA CLASSE DATE

DATE

- La classe *Date* consente di recuperare i valori relativi alla data e all'ora del tempo universale (UTC) o del sistema operativo su cui è in esecuzione Flash Player.
- Per chiamare i metodi della classe Date, è prima necessario creare un oggetto Date mediante la funzione di costruzione della classe Date.

CREARE UN'ISTANZA DI DATE

- Posso passare alla funzione una data del passato o del futuro. Il questo caso Date() richiede da due a sette parametri (anno, mese, giorno, ora, minuti, secondi , millisecondi).
- In alternativa, posso costruire un oggetto *Date* passando un singolo parametro che rappresenta il numero di millisecondi trascorsi dal 1 gennaio 1970 alle 0:00:000 GMT.
- Se, infine, non specifico alcun parametro all'oggetto data Date() vengono assegnate la data e l'ora correnti.

USO DI DATE

- Una volta creato all'oggetto Date posso applicare varie proprietà che:
  - Mi restituiscono informazione sull'anno, il mese, il giorno, il giorno della settimana, ecc della data creata.
  - Mi consentono di confrontare, sommare e sottrarre date

CREAZIONE DI CLASSI

LE CLASSI

- Nella programmazione orientata agli oggetti una classe definisce una *categoria di oggetti*.
- 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 design descrive le caratteristiche di un oggetto, o a come una classificazione botanica descrive le caratteristiche di una pianta.
- 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 oggetto reale e il relativo progetto del designer o tra una pianta reale e la sua classificazione.

LA CLASSE ROSA

LA CLASSE ROSA

```
public class Rosa {
	// Proprietà della classe
	private var proprietario: String;
	private var appassita: Boolean;
	private var colore: String;

	// Metodo "Constructor" : viene chiamato automaticamente quando
	// si crea una instanza della classe
	public function Rosa() {
		proprietario = new String(); // valore iniziale : nessun proprietario
		appassita = false ; // valore iniziale : non appassita
		colore = new String("Rosa");
	}

	// Metodo per definire un nuovo proprietario
	public function regala(nuovoProprietario:String) {
		proprietario = nuovoProprietario;
	}

	public function diChi():String {
		if (proprietario == "") {
			return "nessun proprietario";
		} else {
			return proprietario;
		}
	}
}
```

LA CLASSE ROSA

```
public class Rosa {
    //definizione della classe
    ...
    ...
    ...
}
```

LA CLASSE ROSA

```
public class Rosa {
    // Proprietà della classe
    private var proprietario: String;
    private var appassita: Boolean;
    private var colore: String;

}
```

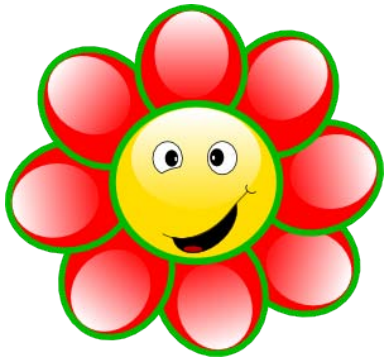
LA CLASSE ROSA

```
public class Rosa {
..
/* Metodo "Constructor" : viene chiamato
automaticamente quando si crea una
instanza della classe */
    public function Rosa() {
          proprietario = new String();
// valore iniziale : nessun proprietario
          appassita = false ;
// valore iniziale : non appassita
          colore = new String("Rosa");
    }
..
}
```

LA CLASSE ROSA

```
public class Rosa {
..
    // Metodo per definire un nuovo proprietario
    public function regala
    (nuovoProprietario:String) {
          proprietario = nuovoProprietario;
    }

    public function diChi():String {
          if (proprietario == "") {
                return "nessun proprietario";
          } else {
                return proprietario;
          }
    }
}
```

LA CLASSE FIORE



fiore

margherita

viola

rosa

LA CLASSE FIORE

```
public class Fiore {
    //proprità della classe Fiore
    private var numPetali:Number;
    private var profumo:Boolean;
    public var genere:String;

    //costruttore
    public function Fiore () {
    //inizialmente la proprietà genere contiene la
    //stringa "non definito"
            genere = "Non definito";
    }
}
```

LA CLASSE ROSA

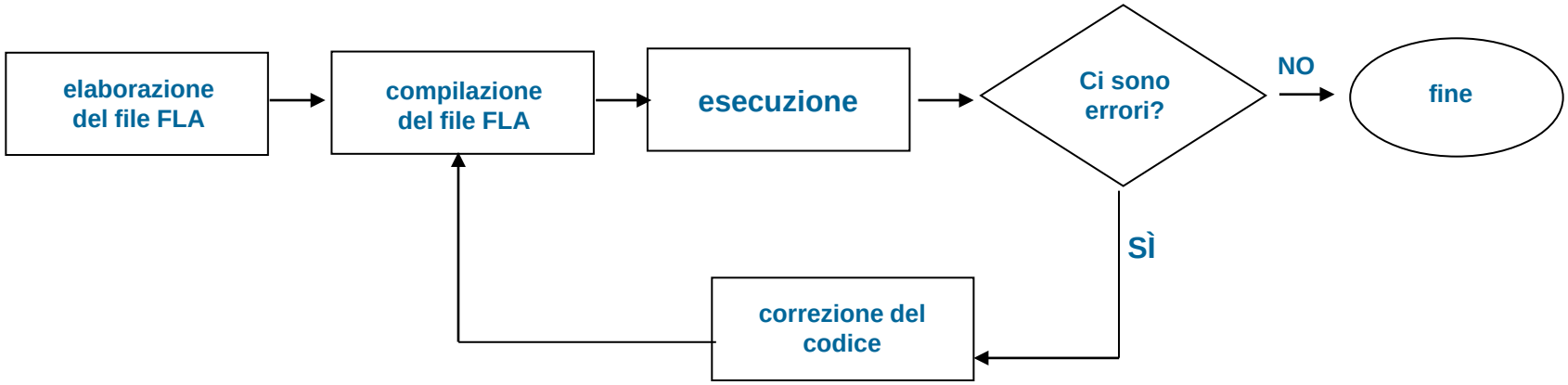
```
public class Rosa extends Fiore {
	// Proprietà della classe
	private var proprietario: String;
	private var appassita: Boolean;
	private var colore: String;

	// Metodo "Constructor" : viene chiamato automaticamente quando
	// si crea una instanza della classe
	public function Rosa() {
		proprietario = new String(); // valore iniziale : nessun proprietario
		appassita = false ; // valore iniziale : non appassita
		colore = new String("Rosa");
	}

	// Metodo per definire un nuovo proprietario
	public function regala(nuovoProprietario:String) {
		proprietario = nuovoProprietario;
	}

	public function diChi():String {
		if (proprietario == "") {
			return "nessun proprietario";
		} else {
			return proprietario;
		}
	}
}
```

LA CLASSE ROSA

```
public class Rosa extends Fiore {
    //definizione della classe
    ...
    ...
    ...
}
```

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. Il compilatore in primo luogo inizierà la sua ricerca dalla cartella in cui è stato salvato il file .fla.
2. 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 3”

LE CLASSI AGGIUNTIVE

- Queste impostazione definiscono le cartelle di partenza dell'organizzazione delle classi
- Le 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

- In ActionScript 3 è fondamentale capire il concetto di package.
- Con l'aggiunta del comando **package** (obbligatorio) per ogni classe viene definto un **package** di appartenenza che corrisponde a come i file di classe sono organizzati su disco
- Supponiamo che il mio progetto Rosa.fla si collocato in **D:\ProgettiFlash\Rosa**

I PACCHETTI

```
//package aninimo
package {
  public class Rosa {
    ...
  }
}
```

- Il file di classe si chiamerà **Rosa.as** e risiederà nella cartella del progetto:
  D:\progettiFlash\Rosa\Rosa.as

I PACCHETTI

```
//package con nome
package classiProgetto{
  public class Rosa {
    ...
  }
}
```

Il file di classe si chiamerà **Rosa.as** e risiederà nella cartella classiProgetto:

D:\progettiFlash\Rosa\classiProgetto\Rosa.as

I PACCHETTI

```
//package con nome
package classiProgetto.uno{
  public class Rosa {
    ...
  }
}
```

Il file di classe si chiamerà **Rosa.as** e risiederà nella cartella classiProgetto\uno:

D:\progettiFlash\Rosa\classiProgetto\uno\Rosa.as

I PACKAGES STANDARD DI FLASH CS3

I PACKAGES STANDARD DI FLASH CS3

- Vedi **Help di Flash**:
  - Guida di riferimento al Linguaggio e alla Componenti di ActionScript 3.0 -> Tutti i packages
- Oppure **Programmare con ActionScript 3.0**
  - Pagina 56

IL COMANDO IMPORT

- Quanado utilizzo un file di classe lo devo comunicare al compilatore utilizzando il comando **import**.

```
// con package anonimo
import Rosa
var r:Rosa = new Rosa();
```

```
// con package con nome
import classiProgetto.uno.Rosa
var r:Rosa = new Rosa();
```

IL FILE DI CLASSE

- Per definire una classe devo creare un file **actionScript** esterno.
- Prima di tutto dovrò definire il package a cui appartiene la classe
- Se il file di classe risiede nella stessa cartella il cui risiede il progetto userò un packege anonimo
- Altrimenti specificherò un package che ricalca il percorso in cui il file è memolrizzato

```
package {
  public class Rosa {
    ...
  }
}
```

IL FILE DI CLASSE

- Dovrò poi inserire i comandi import necessari.
- I comandi import dovranno essere definiti **prima** della definizione di classe
- Dovro definire i comandi import sia pe le classi flash che per quelle da me definite

```
package {
  import flash.display.MovieClip;
  import flash.display.TextField;
  public class Rosa {
    ...
  }
}
```

IL FILE DI CLASSE

- Inserirò quindi le dichiarazioni delle proprietà della classe
- Le proprietà di una classe sono variabili e vanno dichiarate nell stesso modo
- La dichiarazione è preceduta da un attributo che ne regola il **controllo di accesso** (scope)

```
package {
  ..
  public class Rosa {
   private var proprietario:String;
   ...
  }
}
```

IL FILE DI CLASSE

- Inserirò quindi il costruttore
- Il costruttore è una funzione con attributo **public** che ha lo stesso nome della classe, viene eseguito quando viene creata un'istnaza della classe e quindi deve contenere le istruzioni di inizializzazione

```
package {
  ..
  public class Rosa {
   ...
   public function Rosa() {
      proprietario = new String();
      appassita = false ;
      colore = new String("Rosa");
   }
  }
}
```

FILE DI CLASSE

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

```
package {
  ..
  public class Rosa {
    ...
    public function regala(nuovoProprietario:String) {
      proprietario = nuovoProprietario;
    }

    public function diChi():String {
       if (proprietario == "") {
         return "nessun proprietario";
       } else {
          return proprietario;
       }
    }
  }
}
```

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 r:Rosa = new Rosa();
```

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

```
r.regala(“Alessandra”);
```

GLI ATTRIBUTI DI CONTROLLO DI ACCESSO

- Gli attributi di controllo di accesso deteminano la visibiltà o scopo di classi, proprietà e metodi.
- La sintassi è la seguente
- <attributo> **class**….
- <attributo> **var** ….
- <attributo> **function** …..