

# LE CLASSI

## GLI ELEMENTI PRINCIPALI DI OGNI LINGUAGGIO

- **valori costanti**
  - Sono valori dati che non vengono modificati durante lo svolgimento del programma
- **variabili**
  - Un valore appartenente a un determinato tipo di dati (primitivo o derivato) viene associato ad un nome
- **funzione**
  - Un gruppo di istruzioni viene associato ad un nome
- **condizione**
  - La verifica di una condizione modifica il comportamento del programma o di una funzione
- **iterazione**
  - Un gruppo di istruzioni viene ripetuto fino a che una determinata condizione risulta vera

# TIPI DI DATO

- Per rappresentare (codificare) dati semplici uso i **tipi primitivi** (fissati dalle specifiche del linguaggio). In Action Script: **Number**, **String** e **Boolean**.
- Per rappresentare **dati complessi** (ad esempio un elenco di valori, i dati che compongono un indirizzo, una data, ecc.) utilizzo **tipi complessi** che il linguaggio mi offre oppure ne posso creare ad hoc.

# COSTANTI

- Costanti in Action script sono semplicemente i valori che inserisco in un programma.
- Se definiscono anche letterali perché rappresentano letteralmente dei valori.
- Anche se letterali i dati costanti devono essere scritti rispettando precise convenzioni o regole grammaticali.

# COSTANTI

- In Action Script possono essere scritti cinque tipi di valori costanti o letterali:
  - Boolean
  - Number
  - String
  - Object
  - Array
- Tutti gli altri tipi di valori (ad esempio Date, MovieClip, Sound, ecc.) possono essere rappresentati solo utilizzando una variabile

# COSTANTI

- Boolean
  - true o false
- Number
  - per separare la parte decimale bisogna utilizzare il punto:  
0; 34; 1.223;
- String
  - Una serie di caratteri tra apici semplici o doppi:  
'Hello world!';  
"Hello world";
- Object:
  - Coppie chiave:valore letterale separate da virgole e racchiuse tra parentesi graffe:  
{nome:"Paolo",cognome:"Rossi",telefono:"555-1234"};
- Array
  - Valori letterali separati da virgole e racchiusi tra parentesi quadre:  
[1,5,"Pippo",'1987',52.44];

# VARIABILI

- **Dichiarazione**: per dichiarare una variabile in ActionScript si usa la parola riservata **var** seguita dal nome della variabile, dai due punti e dal tipo:  
`var pippo:String;`
- **Inizializzazione**: creazione della variabile e attribuzione di un valore iniziale.
  - Automatica: assegnazione di un valore letterale  
`var pippo:String = "Hello World";`  
`var my_array:Array = ['gen', 'feb', 'mar'];`
  - Utilizzando l'operatore new e la funzione di costruzione del tipo:  
`var pippo:Array = new Array();`  
`var now:Date;`

# COSA È UNA FUNZIONE

- Una funzione (o procedura o metodo) è una costrutto presente in tutti i linguaggi di programmazione che consente di associare un gruppo di comandi ad un **nome**.
- Quando nel programma scriverò l'identificatore saranno eseguiti tutti i comandi che compongono la funzione
- Una funzione può restituire un valore (ad esempio il risultato di un calcolo o di una ricerca).
- La funzione è anche un tipo di dato.



# Sintassi dell'istruzione if

- L'istruzione if consente di tradurre in un linguaggio di programmazione i ragionamenti fatti parlando della logica Booleana.
- L'istruzione if può avere due forme:
  - if** ( espressione ) blocco di istruzioni
  - if** ( espressione ) blocco di istruzioni **else** blocco di istruzioni
- L'espressione che compare dopo la parola chiave **if** deve essere di tipo logico, se la condizione risulta vera viene eseguita l'istruzione subito seguente; nel secondo caso, invece, se la condizione risulta vera si esegue l'istruzione seguente, altrimenti si esegue l'istruzione subito dopo la parola chiave **else**.
- Per più scelte invece si può usare l'**else if** che permette di porre una condizione anche per le alternative, lasciando ovviamente la possibilità di mettere l'**else** (senza condizioni) in posizione finale.

# while

- L'istruzione **while** viene schematizzata come segue:

```
while ( condizione )  
    blocco istruzioni;
```

- Con questa istruzione viene prima valutata l'espressione <condizione>, se l'espressione risulta vera viene eseguito <blocco istruzioni> e il blocco **while** viene ripetuto, altrimenti si esce dal ciclo e si procede con il resto del programma.

# do

- L'istruzione **do** può essere considerato una variante dell'istruzione **while** ed è strutturato nella maniera seguente:

```
do
  blocco istruzioni
while ( condizione )
```

- Prima di tutto viene eseguito il blocco di istruzioni racchiusa tra **do** e **while** (quindi si esegue almeno una volta), poi si verifica il risultato dell'espressione, se è vero si riesegue il **do**, altrimenti si continua con l'esecuzione del resto del programma.

# for

- Il **for** inizializza una variabile, pone una condizione e poi modifica (normalmente incrementa o decrementa) la variabile iniziale.  

```
for (inizializzazione; condizione; modifica)  
    blocco istruzioni;
```
- Il codice <blocco istruzioni> viene eseguito fino a che l'espressione <condizione> risulta vera, poi si passa alla istruzione successiva al **for**.

# 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.

# APPROCCIO PROCEDURALE

- Un tipico approccio procedurale: il programma che stampa i primi  $n$  numeri primi.

# ANALISI

- Sinteticamente il programma:
  - Prenderà uno ad uno i numeri della serie dei numeri naturali
  - Stamperà sullo schermo solo quelli primi
  - Fino a che i numeri stampati saranno il numero richiesto dall'utente

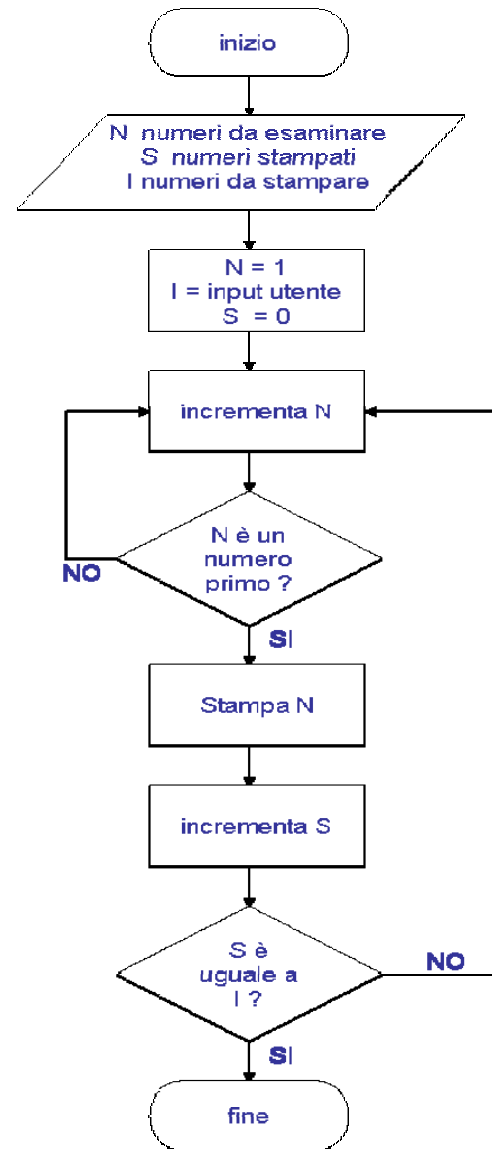


# ANALISI

- Possiamo quindi articolare i compiti del programma in due compiti principali:
  - Stampare sullo schermo i numeri primo fino a raggiungere il numero stabilito
  - Determinare se un numero è primo
- E quindi per risolvere il mio problema definiremo due algoritmi.

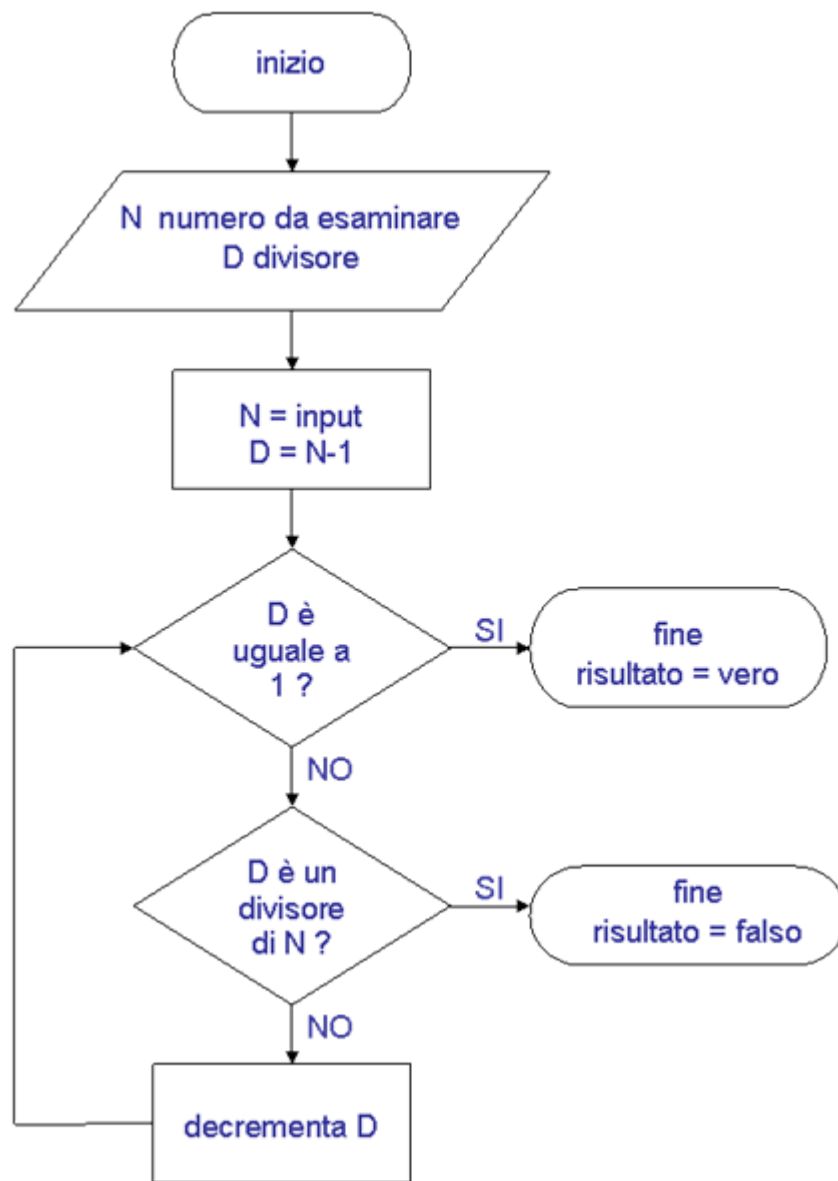
# STAMPA SCHERMO

Utilizziamo un diagramma di flusso per descrivere il semplice algoritmo che stampa i primi **n** numeri primi sullo schermo.



# CONTROLLO

Utilizziamo un secondo diagramma di flusso per descrivere il semplice algoritmo che controlla se un numero è primo.



# PROGRAMMAZIONE OOP

Nome	Tipo	Dimensioni totali	Spazio disponibile
ACER (C:)	Disco locale	26,3 GB	4,64 GB
ACERDATA (D:)	Disco locale	26,5 GB	21,7 GB
Unità DVD-RAM (E:)	Unità CD		
Server 2 su 'Mbsrver' (G:)	Unità di rete disconn...		
Dispositivo portatile	Cartella di sistema		
Documenti condivisi	Cartella di file		
Documenti - Bruno Migliaretti	Cartella di file		
BrunoLav (Z:)	Disco locale	37,3 GB	3,12 GB

## PROGRAMMAZIONE OOP

### Gestione utenti

Se vogliamo organizzare i diversi utenti che usano un'applicazione possiamo creare una classe User, ad esempio, che rappresenta gli utenti. Nella definizione della classe vengono definite le proprietà che hanno gli utenti (ad esempio nome, cognome, username, password, ecc.).

Per gestire i singoli utenti vengono create istanze della classe, che, per la classe User, rappresentano ogni singolo individuo.

Le istanze della classe User comprendono tutte le proprietà della classe User.

start | Risorse del computer | Microsoft ActiveSync | Lezione07-08.ppt | Lezione05-06.ppt | Zuma Deluxe 1.0.0.1 | IT | Norton | 6.52

COPYRIGHT BRUNO MIGLIARETTI

# PROGRAMMAZIONE OOP

- **Gestione utenti**

- Se vogliamo organizzare i diversi utenti che usano un'applicazione possiamo creare una classe *User*, ad esempio, che rappresenta gli utenti. Nella definizione della classe vengono definite le proprietà che hanno gli utenti (ad esempio nome, cognome, username, password, ecc.).
- Per gestire i singoli utenti vengono create *istanze* della classe, che, per la classe *User*, rappresentano ogni singolo individuo.
- Le istanze della classe *User* comprendono tutte le proprietà della classe *User*.

# PROGRAMMAZIONE OOP

- **La pallina animata**

- Il fatto di avere affrontato il problema in maniera orientata agli oggetti ha due vantaggi:
  - Il comportamento della pallina (in quanto è dell'oggetto e non del programma) è generale e non legato ad una procedura specifica
  - Il codice è immediatamente riutilizzabile.



# 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 MEMERI 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.

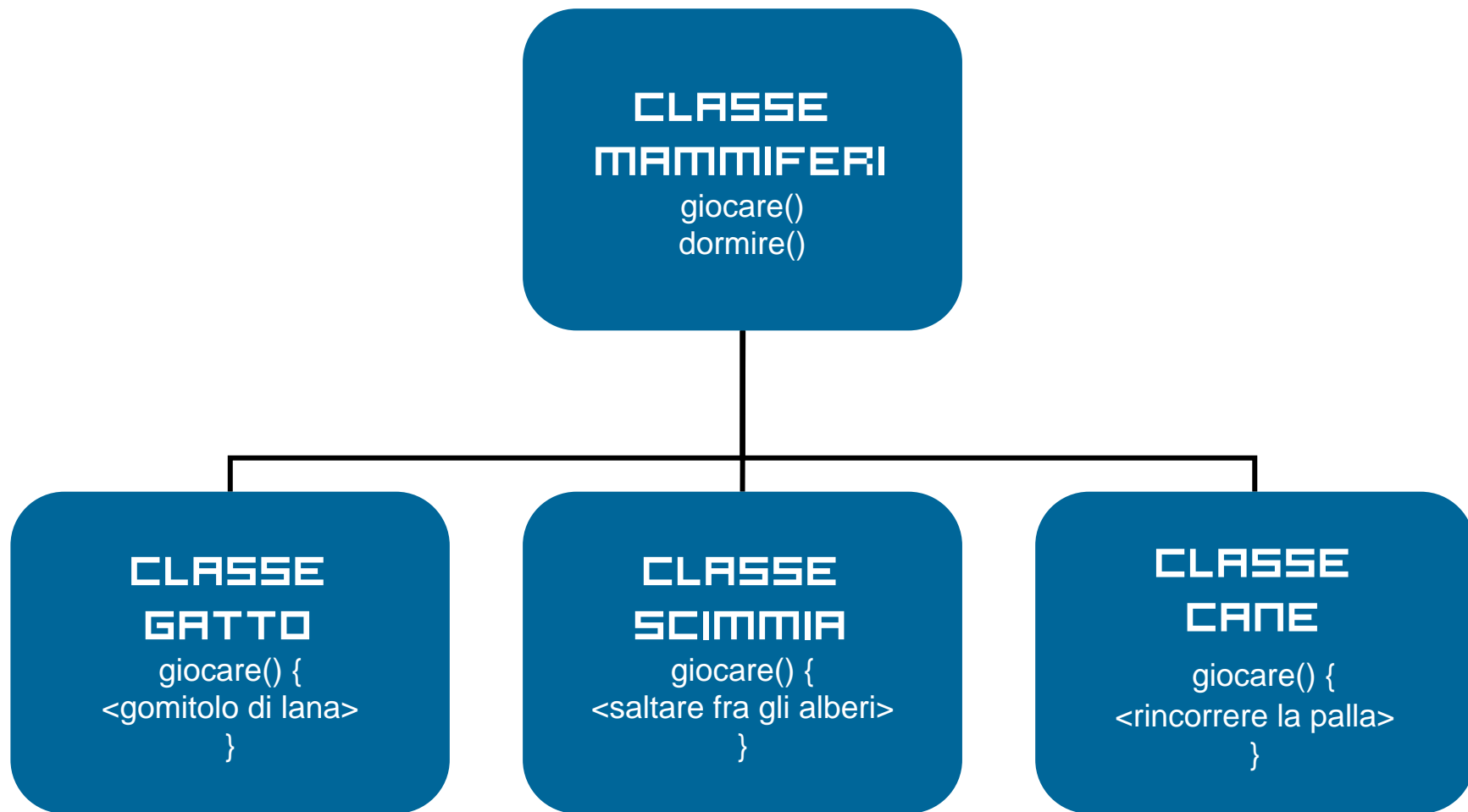
# INCAPSULAMENTO

- 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 oggetti 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 circa 65 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 **costruttore**, 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*.
- Infine ci sono classi la cui istanze vengono create automaticamente quando creo il o colloco il simbolo corrispondente in un fotogramma del filmato. Queste classi sono *Button*, *MovieClip*, *TextField* e *Video*.

# 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*. Questo vale sia per *Object* che per *Array*.

# ESEMPIO

Un altro esempio interessante dell'uso del tipo Object lo troviamo in metodi.fl:

```
var userArr:Array = new Array();
userArr.push({firstname:"Giorgio", age:39});
userArr.push({firstname:"Daniele", age:43});
userArr.push({firstname:"Luca", age:2});
userArr.sortOn("firstname");
var userArrayLenth:Number = userArr.length;
var i:Number;
for (i = 0; i < userArrayLenth; i++) {
    trace(userArr[i].firstname);
}
```

# 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 0:00:00 GMT.
- Se, infine, non specifico alcun parametro all'oggetto data `Date()` vengono assegnate la data e l'ora correnti.

# 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 0:00:00 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 vari metodi 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