

Sistemi informativi applicati (reti di calcolatori): appunti delle lezioni

Da JavaScript a Java

Versione 3.4.05

Per la prima lezione in aula e la prima esercitazione in laboratorio

Da JavaScript a Java

Contestualizziamo la tecnologia HTML/JavaScript, per come sperimentata nel corso di Informatica Applicata, rispetto a tre chiavi di lettura:

- l'ambiente di esecuzione dell'applicazione
- il modello di esecuzione
- l'interazione con l'utente

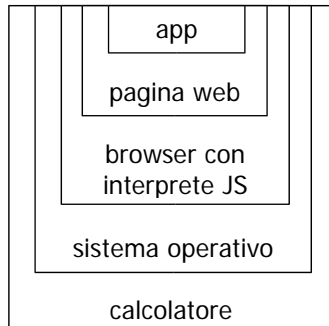
In questo modo documenteremo che:

- da un punto di vista *sintattico*, Java può essere inteso come un'estensione di JS
- da un punto di vista *applicativo*, Java è invece significativamente diverso da JS

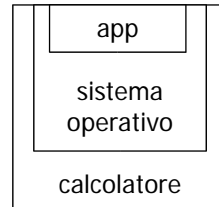
Ambienti di esecuzione

Ogni applicazione software viene sviluppata per essere eseguita in un determinato **ambiente**, costituito di componenti software e hardware

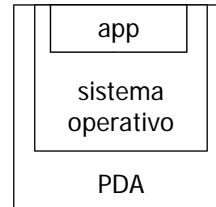
Per esempio:



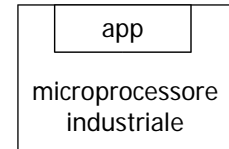
applicazioni JavaScript sviluppate nel corso di Informatica Applicata



tipica applicazione Windows



applicazione per calcolatore palmare



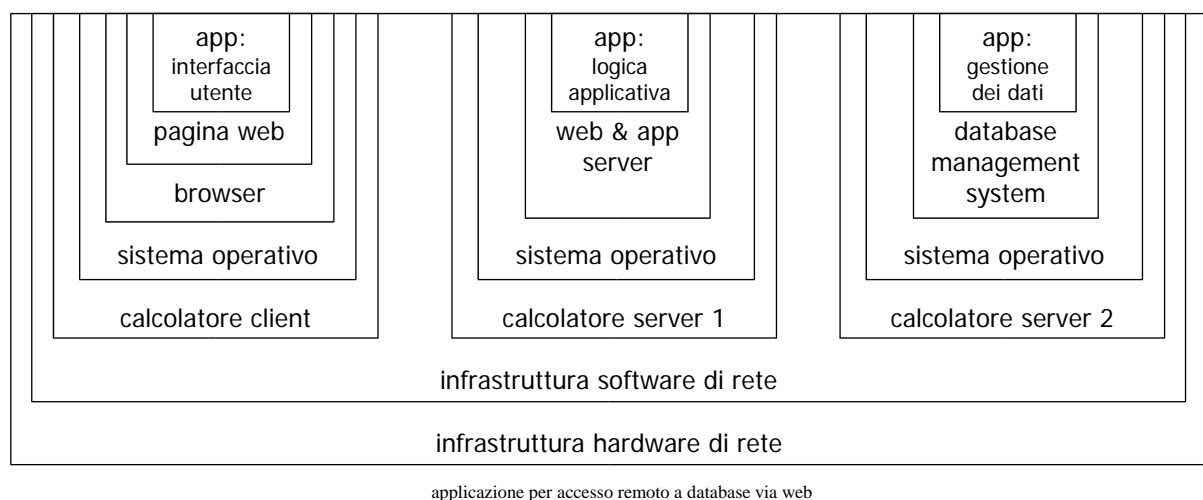
applicazione di controllo di processo

... tutti casi di *esecuzione stand alone*

Ambienti di esecuzione /2

D'altra parte, in molti casi le applicazioni sono **strutturate come sistemi**, con sottosistemi in esecuzione su componenti hardware diverse e remote

Per esempio:



... un ambiente sensibilmente più sofisticato (e complesso...) del precedente

Ambienti di esecuzione e linguaggi di programmazione

Di principio la struttura dell'ambiente di esecuzione non dovrebbe porre vincoli rispetto al linguaggio di programmazione adottato nello sviluppo...

... cosa che *in pratica* richiederebbe però la disponibilità di interpreti/compilatori di ogni linguaggio per ogni ambiente di esecuzione...

Interpreti JS sono effettivamente disponibili in vari ambienti di esecuzione (nei browser per scripting DHTML, negli application server ASP e ASP.NET per scripting web server-side, in Windows Scripting Host per scripting Windows desktop, in Acrobat per scripting in documenti PDF, ...), ma JS non è comunque sufficientemente sofisticato per essere utilizzato come linguaggio per lo sviluppo di applicazioni complesse

Java non ha i limiti di JS, e in particolare può essere utilizzato:

- per applicazioni desktop (Java 2 Standard Edition: J2SE)
- per applicazioni enterprise (Java 2 Enterprise Edition: J2EE)
- per applicazioni per dispositivi mobili (Java 2 Mobile Edition: J2ME)

Modelli di esecuzione

Confrontiamo queste due semplici pagine DHTML:

```
<html><body>
<h1>Pagina uno</h1>
<script>
var x1 = window.prompt();
var x2 = window.prompt();
document.write(x1+x2);
</script>
</body></html>
```

```
<html><body>
<h1>Pagina due</h1>
<input id="x1">
<input id="x2">
<span id="y"></span>
<button onclick="y.innerText=x1.value+x2.value;">Click!</button>
</body></html>
```

Benché il risultato che si ottiene sia sostanzialmente lo stesso (la visualizzazione sulla pagina della concatenazione delle stringhe scritte dall'utente), il modo con cui si ottiene questo risultato è profondamente diverso nei due casi:

- nel primo caso, le istruzioni vengono interpretate ed eseguite in modo sequenziale, e quindi quando la pagina è stata interamente visualizzata l'interprete conclude il suo lavoro: si tratta dunque di un modello di **esecuzione batch**
- nel secondo caso, anche quando la pagina è stata interamente visualizzata l'interprete rimane attivo, nell'attesa dell'input dell'utente: si tratta dunque di un modello di **esecuzione interattiva**

Modelli di esecuzione /2

E' evidente il maggior grado di sofisticazione e di complessità dell'esecuzione interattiva rispetto all'esecuzione batch

Grazie alla presenza del DOM di Internet Explorer, JS può essere facilmente adattato a un modello di esecuzione interattiva

Anche Java può operare in modo sia batch sia interattivo, ma gli strumenti per sviluppare applicazioni interattive in Java non sono così semplici come quelli disponibili per JS

D'altra parte, con Java è possibile sviluppare applicazioni che operano in rete, con una componente client e una componente server che comunicano secondo un modello di **esecuzione interattiva distribuita**

Interazione con l'utente

Una caratteristica certamente peculiare di JS è di non disporre di strumenti propri per la gestione delle funzioni di input e output con l'utente (e infatti anche `window.prompt()` e `window.alert()` sono parte del DOM di IE, *non* di JS...)

L'adozione di pagine DHTML per ospitare script JS garantisce perciò a JS un'interfaccia con l'utente **di tipo grafico** ("*graphical user interface*", GUI)

Java è dotato di strumenti per la gestione dell'interfaccia con l'utente sia di tipo grafico sia basati **sulla linea dei comandi** ("*command line*")

Da JavaScript a Java, operativamente

Per generare una pagina DHTML occorre creare un file di testo, assegnargli un nome con estensione **.htm** oppure **.html**, e inserire al suo interno una struttura del tipo:

```
<html>
<head>
<script>
...
</script>
<body>
...
</body>
</html>
```

... che stabilisce un contesto (un "framework") appropriato per lo script

Anche per generare un'applicazione Java occorre creare un file di testo, assegnargli un nome:

- senza spazi né caratteri speciali
- per convenzione con la prima lettera maiuscola
- con estensione **.java**

(il file che conterrà il nostro primo programma, per esempio, si chiamerà

PrimoProgramma.java)

e inserire al suo interno una struttura in grado di ospitare le istruzioni che dovranno essere eseguite

La struttura di base di un'applicazione Java

Ecco cosa è necessario scrivere in un file di testo di nome **PrimoProgramma.java** perché esso possa essere eseguito come un'applicazione:

```
class PrimoProgramma {
    public static void main(String[] args) {
        ...
    }
}
```

Qualche commento:

- Java è *case-sensitive*, e quindi i vari termini devono essere scritti esattamente come indicato (**class** e non **Class**, **main** e non **Main**, ...)
- ogni applicazione è costituita da (almeno) una "classe", identificata dal termine **class**, il cui nome deve essere uguale al nome del file senza l'estensione **.java**
- proprio come in JS, anche in Java per indicare un blocco di istruzioni lo si delimita con parentesi graffe; dunque questa struttura si può interpretare così: la classe **PrimoProgramma** include una funzione **main** (riconosciamo che è una funzione dalle parentesi che racchiudono i suoi argomenti), che ha sua volta potrà contenere ...
- proprio come in JS, anche in Java è una buona idea rendere il più possibile leggibile un programma indentando opportunamente le sue istruzioni

Il primo programma...

(come in JS, anche in Java si possono introdurre dei commenti in un programma:

- `// questo è un commento che termina alla fine della linea`
- `/* questo è invece un commento
che può proseguire anche su più linee */`

li useremo sistematicamente, scrivendoli in blu per chiarezza)

Ecco la più semplice applicazione Java:

```
//file: PrimoProgramma.java
class PrimoProgramma {
    public static void main(String[] args) {
        System.out.println("il mio primo programma");
    }
}
```

il cui scopo è, semplicemente, di visualizzare sullo schermo “il mio primo programma”

Il primo programma: qualche altro commento

A proposito dell'istruzione `public static void main(String[] args)`

- **main** è il nome predefinito della funzione attivata quando un'applicazione viene messa in esecuzione
- **public static void** sono dei costrutti (obbligatori nel caso di **main**) che qualificano le caratteristiche della funzione: per ora consideriamoli necessari, senza entrare in dettagli sul loro significato
- **String[] args** è la dichiarazione dell'argomento (cioè della variabile di input) della funzione; come vedremo, in Java è sempre necessario dichiarare le variabili *con il loro tipo*: in questo caso, quindi, viene introdotto un argomento di nome **args** e di tipo array di **String**

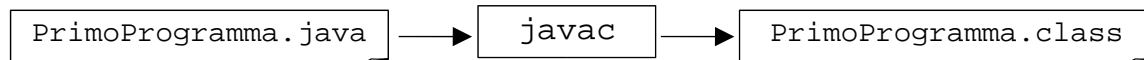
A proposito dell'istruzione `System.out.println("il mio primo programma");`

- nel suo complesso, il costrutto `System.out.println` ha lo scopo di visualizzare il suo argomento, in questo caso la stringa `il mio primo programma`, sullo schermo
- in particolare `System.out` specifica il dispositivo di uscita (per default appunto lo schermo), un oggetto predefinito e preinizializzato, a cui si applica il metodo `println` che ha lo scopo di visualizzare il suo argomento e di forzare poi un ritorno a capo (mentre il metodo `print` avrebbe prodotto la stessa visualizzazione ma senza il ritorno a capo)

Il primo programma: il ciclo di sviluppo

L'applicazione contenuta nel file `PrimoProgramma.java` non è ancora eseguibile, dato che è scritta in un linguaggio di cui non disponiamo di un interprete

Occorre perciò convertire il programma in una sua versione equivalente ma scritta in un linguaggio eseguibile; l'**ambiente di sviluppo di Java** (il *Java Development Kit*, JDK) mette a disposizione un programma, un "compilatore Java" (chiamato `javac`), che ha esattamente questa funzionalità:



Fornendo in input a `javac` il file `PrimoProgramma.java` viene prodotto il file `PrimoProgramma.class`, finalmente eseguibile da parte dell'interprete Java (la *Java Virtual Machine*, JVM)

Il primo programma: operativamente

Dalla linea di comando (supponendo che la shell sia aperta nella directory che contiene il file `PrimoProgramma.java`) occorre dunque:

1. compilare il programma, con il comando: `javac PrimoProgramma.java`
2. eseguire il programma, con il comando: `java PrimoProgramma`

(si noti il dettaglio che nel comando di esecuzione occorre specificare il nome del file senza l'estensione `.class`)

```
cmd
C:\prova>javac PrimoProgramma.java
C:\prova>java PrimoProgramma
il mio primo programma
C:\prova>
```

← Nessuna segnalazione: la compilazione "è andata a buon fine"...

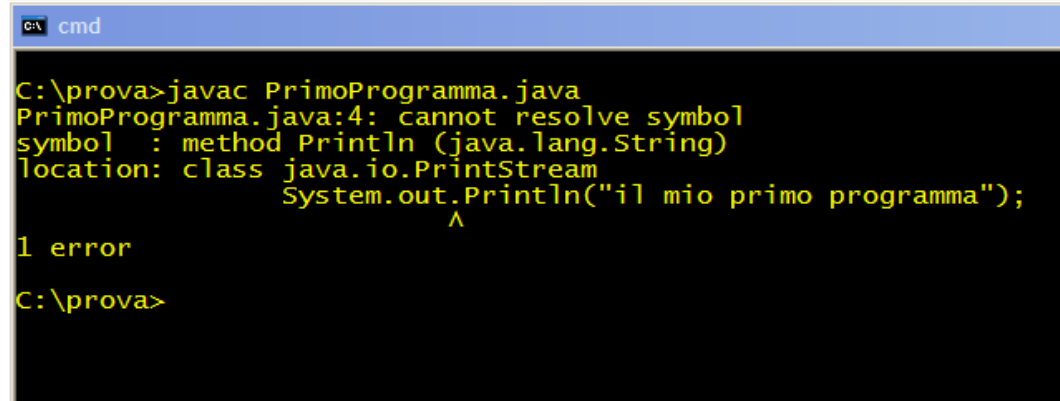
← Il risultato dell'esecuzione...

Il primo programma: errori e debugging

Se avessimo commesso uno o più errori sintattici nella scrittura del codice, per esempio:

```
System.out.Println("il mio primo programma"); // Println non esiste...
```

... al momento della compilazione avremmo ottenuto un messaggio di errore da parte del compilatore, che ci segnala che tipo di errore si è generato e in quale linea:



```
C:\prova>javac PrimoProgramma.java
PrimoProgramma.java:4: cannot resolve symbol
symbol  : method Println (java.lang.String)
location: class java.io.PrintStream
        System.out.Println("il mio primo programma");
                ^
1 error
C:\prova>
```

La capacità di “fare debugging”, gestendo appropriatamente situazioni di questo genere (e quindi in particolare di sfruttare al meglio l’informazione fornita dal compilatore) è fondamentale...

La gestione dell’input e dell’output

Purtroppo la gestione dell’I/O in Java non è esattamente semplice (e per questa ragione prossimamente ci doteremo di una libreria per rendere agevole questo compito)

Abbiamo già usato l’istruzione `system.out.println(x)`; per la visualizzazione, il cui argomento, proprio come in JS, `x` può essere sia una stringa sia un valore numerico

Per dotare le nostre prime applicazioni della capacità di acquisire gli input che fornisce l’utente, ricorriamo al fatto che la funzione `main()` prevede (e in effetti richiede) come argomento un array di stringhe, corrispondenti agli argomenti che l’utente indica nella linea di comando successivamente al nome del programma

Se dunque il nostro programma include la dichiarazione:

```
public static void main(String[] args)
```

e lo eseguiamo con il comando:

```
java PrimoProgramma uno due "tre e quattro"
```

in modo automatico l’interprete Java effettuerà gli assegnamenti:

```
args[0] ← "uno"      args[1] ← "due"      args[2] ← "tre e quattro"
```

da cui si nota che:

- come in JS, anche in Java il primo elemento degli array ha indice 0
- nella lista degli argomenti sulla linea di comando lo spazio è un separatore e le virgolette sono un delimitatore

La gestione del tipo delle variabili

Una importante differenza tra JS e Java riguarda la gestione del tipo delle variabili

In JS le variabili si dichiarano senza specificarne il tipo, che viene determinato automaticamente e dinamicamente dall'interprete in base al tipo del valore che viene assegnato alle variabili stesse:

```
var x;           // la variabile non ha ancora un tipo
x = 1;           // la variabile diventa di tipo numerico...
x = "1";         // ... e ora diventa di tipo stringa
```

Java, al contrario, richiede che le variabili vengano dichiarate direttamente con il loro tipo, che non può essere più cambiato durante l'esecuzione:

```
int x1;          // la variabile viene dichiarata di tipo numero intero
String x2;        // la variabile viene dichiarata di tipo stringa
x1 = 1;           // un'assegnazione corretta...
x2 = "1";         // ... e un'altra assegnazione corretta...
x1 = "1";         // ... mentre sia questa...
x1 = x2;          // ... generano errori che il compilatore segnala
```

Questa differenza tra JS e Java si sintetizza specificando che:

- JS è un linguaggio "a tipi deboli" (*weakly typed*)
- Java è un linguaggio "a tipi forti" (*strongly typed*)

Tipi di variabili e conversione tra tipi

Come vedremo nel seguito, una delle caratteristiche che rendono Java così sofisticato è che in pratica ogni classe che un programmatore crea può essere utilizzata come tipo per variabili

Java ci giunge comunque già dotato di un ricco insieme di tipi, e in particolare:

- **int** : numeri interi
- **double** : numeri decimali
- **String** : stringhe

Come in JS, anche in Java tra variabili numeriche e stringhe è possibile effettuare delle conversioni:

- da stringhe a numeri:

```
String x1 = "1";
int x2 = Integer.parseInt(x1);
double x3 = Double.parseDouble(x1);
```

- da numeri a stringhe:

```
int x1 = 1;
double x2 = 1.2;
String x3 = Integer.toString(x1);
String x4 = Double.toString(x2);
```

Tipi di variabili e conversione tra tipi /2

Per la conversione da numeri a stringhe si può anche sfruttare il fatto che l'operatore polimorfo `+` (che somma numeri e concatena stringhe, proprio come in JS) se applicato a un numero e una stringa converte automaticamente il numero nella stringa corrispondente. Dunque:

```
int x1 = 1;
String x2 = Integer.toString(x1);
```

e:

```
int x1 = 1;
String x2 = "" + x1;
```

producono esattamente lo stesso effetto

Per la conversione da `int` a `double` e viceversa, si usa una notazione diversa, basata su un'operazione chiamata *casting*:

```
int x1 = 1;
double x2 = (double)x1;
```

e analogamente:

```
double x1 = 1.2;
int x2 = (int)x1;
```

Strutture di controllo

Le strutture di controllo sono forse la caratteristica di Java in cui esso è più simile a JavaScript (e, se è per questo, al C e al C++)

Come in JS, si trovano in Java la struttura condizionale:

```
if(condizione) {
    blocco1
} else {
    blocco2
}
```

e le strutture iterative:

```
for(inizializzazione; condizione; incremento) {
    blocco
}
```

```
while(condizione) {
    blocco
}
```

```
do {
    blocco
} while(condizione);
```

Esercizi / problemi

1. Creare un'applicazione che visualizza **Ciao a tutti**
2. Creare un'applicazione che visualizza la somma dei numeri interi da **1** a **n**, dove **n** è specificato nel codice
3. Creare un'applicazione che visualizza **Ciao xxx**, dove **xxx** è l'input dell'applicazione, specificato sulla linea di comando all'atto di eseguire l'applicazione
4. Modificare l'applicazione al punto 2 in modo da specificare il valore **n** sulla linea di comando
5. Creare un'applicazione che visualizza la somma di due numeri in input
6. Modificare l'applicazione precedente introducendo un controllo sul fatto che gli input siano effettivamente due (usa `args.length`), visualizzando un messaggio di errore in caso contrario (nota che rimane il problema degli input non riconoscibili come numeri)
7. Modificare l'applicazione precedente visualizzando non la somma ma la media dei due numeri (attenzione al tipo delle variabili...)
8. Modificare l'applicazione precedente mettendo il calcolo della media in una funzione (per esempio `static double media(int x1, int x2) { ...}`)
9. Creare una gestione di conto corrente, con possibilità di specificare versamenti e prelievi, visualizzazione del saldo dopo ogni operazione e controllo di conto non scoperto

Soluzioni: problema 1

```
class Esercizio1 {  
  
    public static void main(String[] args) {  
        System.out.println("Ciao a tutti");  
    }  
  
}
```

Soluzioni: problema 2

```
class Esercizio2 {  
  
    public static void main(String[] x) {  
        int somma = 0, n = 20;  
        for(int i = 1; i <= n; i++) {  
            somma += i;  
        }  
        System.out.println("somma=" + somma);  
    }  
}
```

Soluzioni: problema 3

```
class Esercizio3 {  
  
    public static void main(String[] x) {  
        System.out.println("Ciao " + x[0]);  
    }  
}
```

Soluzioni: problema 4

```
class Esercizio4 {  
  
    public static void main(String[] x) {  
        int n = Integer.parseInt(x[0]);  
        int somma = 0;  
        for(int i = 1; i <= n; i++) {  
            somma += i;  
        }  
        System.out.println("somma=" + somma);  
    }  
}
```

Soluzioni: problema 5

```
class Esercizio5 {  
  
    public static void main(String[] x) {  
        int n1 = Integer.parseInt(x[0]);  
        int n2 = Integer.parseInt(x[1]);  
        int n3 = n1 + n2;  
        System.out.println("somma=" + n3);  
    }  
}
```

Soluzioni: problema 6

```
class Esercizio6 {  
  
    public static void main(String[] x) {  
        if(x.length != 2) {  
            System.out.println("Errore: ci vogliono 2 parametri...");  
            return;  
        }  
        int n1 = Integer.parseInt(x[0]);  
        int n2 = Integer.parseInt(x[1]);  
        int n3 = n1 + n2;  
        System.out.println("somma=" + n3);  
    }  
}
```

27

Soluzioni: problema 8

```
class CalcolaMedia {  
  
    public static void main(String[] args) {  
        if(args.length != 2) {  
            System.out.println("Numero di valori in input non corretto...");  
            return;  
        }  
        int a1 = Integer.parseInt(args[0]);  
        int a2 = Integer.parseInt(args[1]);  
        System.out.println(media(a1,a2));  
    }  
  
    static double media(int x1, int x2) {  
        return (x1 + x2) / 2.0;  
    }  
}
```

28

Soluzioni: problema 9

```
class EsercizioConto {  
  
    public static void main(String[] args) {  
        int saldo = 0;          // conto iniziale  
  
        int versamento = 100;  
        saldo += versamento;  
        System.out.println("Saldo attuale: " + saldo);  
  
        int prelievo = 130;  
        saldo -= prelievo;  
        System.out.println("Saldo attuale: " + saldo);  
  
        if(saldo<0) System.out.println("Conto scoperto!");  
    }  
}
```