## Identificatori

```
<identificatore> ::= <prefisso> [<suffisso>]

<prefisso> ::= _ | <lettera>

<suffisso> ::=
     <elemento> |
     <elemento> <suffisso>

<elemento> ::=
     <cifra> |
     <lettera> |
     _

<lettera> ::= A | B | ... | Z | a | b | ... | z

<cifra> ::= 0 | 1 | ... | 9
```

```
// Identificatori leciti
int _p = 10;
int Q9x = 34;
int questoIdentificatoreVaBene = 48;

// Identificatori scorretti
int 9a = 0;
int lo spazioNonVaBene = 4;
```

Java consente identificatori (nomi per variabili, metodi, classi) di lunghezza arbitraria contenenti lettere (incluse le accentate), cifre e il carattere "_" (sottolineatura), a patto che il carattere iniziale non sia una cifra. Gli identificatori non possono contenere altri caratteri (spazi e segni di interpunzione). Per convenzione gli identificatori sono scritti come segue:

- variabili (inclusi gli attributi delle classi): lettere minuscole (es. test) e utilizzo di maiuscole per separare più parole all'interno di un identificatore (es. primoTest).
- costanti (inclusi gli attributi delle classi): lettere maiuscole (es. TEST) e utilizzo di "_:" per separare più parole all'interndo di un identificatore (es. PRIMO_TEST)
- classi: prima lettera maiuscola (es. Deposito) e utilizo delle maiuscole per la separazione delle parole (es. DepositoBancario)
- metodi: stessa convenzione delle variabili

## Tipi

```
<tipo> ::=
     <tipo_primitivo> |
     <tipo_reference>

<tipo_primitivo> ::=
     byte | short | int | long | float | double |
     boolean |
     char

<tipo_reference> ::=
     String | StringBuffer |
     <vettore> |
     <identificatore>

<vettore> ::= <tipo> [ ]
```

```
// Dichiarazione di variabili con tipo primitivo
byte x = 10;         // Un valore numerico da -128 a +127
boolean flag = true; // Un valore logico vero/falso
char c = 'A'         // Singoli caratteri Unicode

// Dichiarazione di variabili con tipo reference
String s = "Java";             // Seq. di caratteri (immutabile)
StringBuffer sb =
     new StringBuffer("Java"); // Seq. di caratteri (modificabile)
int[] v = new int[10];         // Vettore di int
String[] w = new String[10]    // Vettore di String
ContoCorrente cc =
     new ContoCorrente(1000.0) // Oggetto della classe ContoCorrente
```

Il tipo definisce le caratteristiche della variabile (o della costante) all'atto della sua dichiarazione. Java ha 8 tipi primitivi di cui sei tipi numerici: byte (8 bit), short (16 bit), int (32 bit) e long (64 bit) che possono contenere numeri interi, mentre float (32 bit) e double (64 bit) possono contenere numeri in virgola mobile. Il tipo "logico" boolean può contenere il risultato di espressioni logiche, e le costanti true e false. Infine, il tipo char che può contenere singoli caratteri.. Vi sono poi i tipi reference, tra cui i tipi predefiniti String, StringBuffer e i vettori che consentono, rispettivamente, di rappresentare sequenze di caratteri alfanumerici (immutabili nel caso di String, modificabili nel caso di StringBuffer) e sequenze di oggetti omogenei (sia di tipo primitivo, sia di tipo reference). Infine, data una definizione di classe (ad es. ContoCorrente) si ha a tutti gli effetti la definizione di un nuovo tipo reference che può essere utilizzato analogamente agli altri predefiniti.

## Espressioni

```
<espr> ::=
     <espr_numerica> | <espr_logica> | <espr_stringa> |
     <creazione_oggetto> |<inizializzatore_array>

<espr_numerica> ::=
     ( <espr_numerica> ) |
     <op_num_un> <espr_numerica> |
     <espr_numerica> <op_num_bin> <espr_numerica> |
     <valore_numerico> | <valore_carattere>

<valore_numerico> ::=
     <invocazione_metodo> | <elemento_array> | <identificatore> |
     <costante_numerica>

<valore_carattere> ::=
     <invocazione_metodo> |<elemento_array> |<identificatore> |
     ' <carattere> '

<carattere> ::= {generico carattere Unicode}

<op_num_un> ::= - | +

<op_num_bin> ::= + | - | * | / | %

<espr_logica> ::=
     ( <espr_logica> ) |
     <espr_relazionale> |
     <op_log_un> <espr_logica> |
     <espr_logica> <op_log_bin> <espr_logica> |
     <valore_logico>

<valore_logico> ::=
     <invocazione_metodo> |<elemento_array> |<identificatore> |
     true |false

<espr_relazionale> ::=
     ( <espr_relazionale> ) |
     <espr_numerica> <op_rel> <espr_numerica>

<op_log_un> ::= !

<op_log_bin> ::= && | ||

<op_rel> ::= < | > | <= | >= | == | !=
```

**(Continua alla pag. successiva)**

```
<espr_stringa> ::=
     ( <espr_stringa> ) |
     <espr_stringa> + <espr_stringa> |
     <invocazione_metodo> |<elemento_array> |<identificatore> |
     " <seq_caratteri> "

<seq_caratteri> ::=
     <carattere> |
     <carattere> <seq_caratteri>

<invocazione_metodo> ::=
     [<identificatore> . ] <identificatore> ( [<seq_argomenti>] )

<seq_argomenti> ::=
     <espr> |
     <espr> , <seq_argomenti>

<elemento_array> ::= <identificatore> [ <espr_numerica> ]

<costante_numerica> ::= <intero> | <virgola_mobile>

<intero> ::= <seq_cifre>

<virgola_mobile> ::= <seq_cifre> . <seq_cifre>

<seq_cifre> ::=
     <cifra> |
     <cifra> <seq_cifre>

<creazione_oggetto> ::=
     new <identificatore> ( [<seq_argomenti>] ) |
     new <identificatore> [ <espr_numerica> ]

<inizializzatore_array> ::= { <seq_argomenti> }
```

## Istruzioni

```
<istruzione> ::=
     <dichiarazione> | <assegnamento> | <inizializzazione> |
     <scelta_condizionale> |
     <iterazione> |
     <invocazione_metodo>
     return <espr> ;

<dichiarazione> ::=
     <tipo> <seq_identificatori> ; |

<assegnamento> ::=
     <identificatore> = <espr> ; |
     <identificatore> <op_abbrev> <espr> ;

<op_abbrev> := += | -= | *= | /= | %=

<inizializzazione> ::=
     <tipo> <identificatore> = <espr>;|
     final <tipo> <identificatore> = <espr>;

<seq_identificatori> ::=
     <identificatore> |
     <identificatore> , <seq_identificatori>
```

```
// Dichiarazione, assegnamento e inizializzazione
int p;
p = 10;
int x= 34;

// Operatori di somma, sottrazione, ecc. abbreviati
x += 20;  // equivale a x = x + 20
p /= 2;   // equivale a p = p / 2

// Dichiarazioni multiple
int a, b, c;
String s1, s2, s3;
```

Le istruzioni di scelta condizionale, di iterazione, di invocazione di procedura e l'istruzione return sono esaminate in schede separate. Nel caso di assegnamento e inizializzazione è necessarioche il tipo di <espr> sia compatibile con il tipo della variabile a cui si assegna il valore.

## Istruzioni di scelta condizionale

```
<scelta_condizionale> ::=
     <if_then_else> |
     <if_then> |
     <if_else_if> |
     <switch>

<if_then_else> ::=
     if (<espr_logica>) {
          <seq_istruzioni>
     } else {
          <seq_istruzioni>
     }

<switch> ::=
     switch (<espr>) {
          <seq_case>
          [default :
           <seq_istruzioni>]
     }

<seq_case> ::=
     <case> |
     <case> <seq_case>

<case> ::=
     case <valore>:
          [<seq_istruzioni>]
          [break;]
```

```
<if_then> ::=
     if (<espr_logica>) {
          <seq_istruzioni>
     }

<if_else_if> ::=
     if (<espr_logica>) {
          <seq_istruzioni>
     }
     <seq_else_if>
     else {
          <seq_istruzioni>
     }

<seq_else_if> ::=
     <else_if> |
     <else_if> <seq_else_if>|

<else if> ::=
     else if (<espr_logica>) {
          <seq_istruzioni>
     }

<seq_istruzioni> ::=
     <istruzione> |
     <istruzione> <seq_istruzioni>
```

```
// i e j sono variabili di tipo int
if (i < j) {
     i = j + 1;
} else {
     j = i + 1;
}

// c è una variabile char, continua è una variabile di tipo boolean
switch (c) {
     case 's' : case 'S' :
          continua = true; break;
     case 'n' : case 'N' :
          continua = false; break;
     default :
          System.out.println("Errore!");
}
```

**Istruzioni di iterazione**

```
<iterazione> ::=
     <while_do> |
     <do_while> |
     <for>

<while_do> ::=
     while (<espr_logica)) {
          <seq_istruzioni>
     }

<do_while> ::=
     do{
          <seq_istruzioni>
     } while (<espr_logica));

<for> ::=
     for (<inizio>; <espr_logica> ; <passo>) {
          <seq_istruzioni>
     }

<inizio> ::=
     <tipo> <identificatore> = <espr>;|
     <identificatore> = <espr>;

<passo> ::=
     <assegnamento> |
     ++ <identificatore> |
     -- <identificatore>
```

```
int x= 34;       // x è inizializzata a 34
while (x > 0) {
     x = x - 1;
}                // al termine del primo ciclo x == 0

do {
     x = x + 1;
} while (x < 34) // al termine del secondo ciclo x == 34

int s = 0;
for (int i = 1; i < x; i *= 2) {
     s += i;
}                // s == 1 + 2 + 4 + 8 + 16 + 32
```

La definizione di <assegnamento> rispetta quella data nella scheda delle istruzioni.. I cicli vengono ripetuti fintanto che la condizione dettata da <espr_logica> è verificata.

**Definizione di classe**

```
<def_classe> ::=
     class <identificatore> {
          [<dich_attributi>]
          [<def_costruttori>]
          [<def_metodi>]
          [<def_metodo_main>]
     }

<dich_attributi> ::=
     <dich_attributo> |
     <dich_attributo> <dich_attributi>

<def_costruttori> ::=
     <def_costruttore> |
     <def_costruttore> <def_costruttori>

<def_metodi> ::=
     <def_metodo> |
     <def_metodo> <def_metodi>
```

```
class Persona {
     private String nome;
     private int    anni;

     public Persona( String n, int a ) {
          nome = n;
          anni = aM
     }

     public String restituisciNome() {
          return nome;
     }
     public int restituisciAnni() {
          return anni;
     }
}
```

La definizione di una classe consta di tre elementi (opzionali): la dichiarazione di uno o più attributi, cioè degli alementi dello stato degli oggetti della classe, la definizione di uno o più costruttori che sovraintendono alla corretta inizializzazione dello stato all'atto della creazione del singolo oggetto, e la definizione di uno o più metodi che definiscono il comportamento degli oggetti della classe. Per convenzione l'identificatore che consegna la classe si scrive con la prima lettera maiuscola (ad es. Persona, Libro, Automezzo) e, qualora l'identificatore consti di più parole, si utilizzano le maiuscole per identificarle (ad es., GestioneStrumenti, ContoCorrente).

**Definizione di attributo**

```
<def_attributo> ::=
     <modificatore> <tipo> <seq_identificatori> ; |
     <modificatore> static <tipo> <identificatore> = <espr>;|
     <modificatore> static final <tipo> <identificatore> = <espr>;

<modificatore> ::= public | private

<seq_identificatori> ::=
     <identificatore> |
     <identificatore> , <seq_identificatori>
```

```
class ContoCorrente {
     // Definizione di attributo costante
     public static final double SPESE FISSE = 50.0

     // Definizione di attributo a livello di classe
     private static int totaleContiAperti = 0;

     // Definizioni di attributi (a livello di oggetto)
     private int    numeroConto;
     private double saldo, commissioni;

     public ContoCorrente() {
          numeroConto = totaleContiAperti;
          totaleContiAperti += 1;
          saldo = 0.0;
          commissioni = 0.0;
     }

}
```

Si possono definire tre tipi di attributo: standard (a livello di oggetto), static (a livello di classe) e costante (sempre static). Gli attributi standard possono cambiare valore nel tempo e assumono valori indipendenti in ogni singola istanza; gli sttributi static possono cambiare valore nel tempo ma il loro valore è condiviso fra tutte le istanze; infine, gli attributi costanti sono condivisi fra tutte le istanze e il loro valore è immutabili nel tempo. La dichiarazione di attributi static e costanti deve essere contestuale all'inizializzazione: `<espr>` è un'espressione compatibile con `<tipo>` che deve essere valutabile al momento della compilazione della classe, ossia deve dipendere solo da costanti letterali o simboliche, a patto che queste ultime siano definite ed accessibili all'interno della classe in cui si dichiara l'attributo.

**Definizione di costruttore**

```
<def_costruttore> ::=
     <modificatore> <identificatore> ( [<dich_parametri>] ) {
          [<seq_istruzioni>]
     }

<modificatore> ::= public | private

<dich_parametri> ::=
     <tipo> <identificatore> |
     <tipo> <identificatore>, <dich_parametri>

<seq_istruzioni> ::=
     <istruzione> |
     <istruzione> <seq_istruzioni>
```

```
class ContoCorrente {
     private double saldo;

     ...

     public ContoCorrente() {
          saldo = 0.0;
     }
     public ContoCorrente( double s ) {
          saldo = s;
     }

     ...
}
```

Il costruttore è un particolare metodo invocato all'atto della costruzione di un oggetto (istanza) della classe. Il suo scopo è quello di inizializzare correttamente lo stato dell'oggetto, eventualmente utilizzando parametri. L'identificatore del costruttore coincide obbligatoriamente con l'identificatore della classe. Si possono definire più costruttori per una classe a patto che questi differiscano per il numero e il tipo dei parametri accettati (vedi esempio).

**Definizione di metodo**

```
<def_metodo> ::=
     <def_procedura> |
     <def_funzione>

<def_procedura> ::=
     <modificatore> void <identificatore> ( [<dich_parametri>] ) {
          [<seq_istruzioni>]
     }

<def_funzione> ::=
     <modificatore> <tipo> <identificatore> ( [<dich_parametri>] ) {
          [<seq_istruzioni>]
     }

<modificatore> ::= public | private

<dich_parametri> ::=
     <tipo> <identificatore> |
     <tipo> <identificatore>, <dich_parametri>

<seq_istruzioni> ::=
     <istruzione> |
     <istruzione> <seq_istruzioni>
```

```
class ContoCorrente {
     private double saldo;

     public ContoCorrente() {
          saldo = 0.0;
     }

     public void deposita( double somma ) {
          saldo = saldo + somma;
     }
     public double restituisciSaldo() {
          return saldo;
     }
}
```

I metodi definiscono i comportamenti comuni ai singoli oggetti, eventualmente definiti su uno o più parametri di ingresso. Le procedure eseguono un elaborazione senza restituire alcun valore (deposita nell'esempio), mentre le funzioni restituiscono un valore come risultato della loro elaborazione (restituisciSaldo nell'esempio). In particolare, per le funzioni è necessario che la sequenza delle istruzioni contenga un'istruzione **return** seguita da un'espressione di tipo compatibile con quello dichiarato nell'intestazione della funzione.

**Definizione di metodo principale (main)**

```
<def_metodo_main> ::=
     public static void main( String[] args ) {
          [<seq_istruzioni>]
     }

<seq_istruzioni> ::=
     <istruzione> |
     <istruzione> <seq_istruzioni>
```

```
class CiaoMondo {
     public static void main( String[] args ) {
          // Scrive Ciao a tutti! in console
          System.out.println("Ciao a tutti!");
     }
}
```

Il metodo main è un particolare metodo invocato dal sistema operativo all'atto dell'esecuzione di un'applicazione Java. Pertanto, ogni applicazione Java consta di un insieme di classi di cui almeno una, detta classe principale, è dotata della definizione del metodo main. Un insieme di classi privo di una classe principale non da luogo ad una applicazione eseguibile.

## Metodi della classe String

### charAt
x.charAt(n) restituisce il carattere avente indice n in x

```
String s = "Viva Java!";
char r = s.charAt(1);   // contiene 'i'
```

### length
x.length() restituisce la lunghezza di x (numero di caratteri)

```
String s = "Viva Java!";
int r = s.length();   // contiene 10
```

### equals
x.equals(y) restituisce true se il contenutodi x è identico al contenuto di y

```
String s = "Viva Java!", t = "Viva Java!";
boolean r = s.equals(t);   // contiene true
```

### equalsIgnoreCase
x.equalsIgnoreCase(y) ignora le differenze tra maiuscole e minuscole

### compareTo
x.compareTo(y) restituisce un valore minore, uguale o maggiore di 0 a seconda che x sia (lessicograficamente) minore, uguale o maggiore di y. Pertanto x.compareTo(y) è 0 solo se x.equals(y) è true.

### substring
x.substring(n,m) restituisce una stringa corrispondente alla porzione di x dalla posizione n alla posizione m

```
String s = "Viva Java!";
String t = substring(1,3); // contiene "iva"
```

### trim
x.trim() restituisce una stringa con lo stesso contenuto di x privato di eventuali spazi all'inizio e alla fine

### startsWith
x.startsWith(y) restituisce true se x comincia con y

### endsWith
x.endsWith(y) restituisce true se x termina con y

### indexOf
x.indexOf(c) restituisce la posizione di c in x e x.indexOf(c,n) restituisce la posizione di c in x cominciando la ricerca dalla posizione n; entrambe restituiscono –1 se c non è presente in x (c di tipo char)

## Metodi della classe StringBuffer

### charAt
Equivalente all'omonimo metodo di String

### length
Equivalente all'omonimo metodo di String

### setCharAt
x.setCharAt(n, c) imposta il carattere in posizione n al valore c (c è di tipo char)

### deleteCharAt
x.deleteCharAt(n) cancella il carattere avente indice n

### append
x.append(y) concatena a x il contenuto di y (y può essere String o StringBuffer)

### reverse
x.reverse() rovescia il contenuto di x

### insert
x.insert(n,y) inserisce y all'interno di x a partire dalla posizione n

### delete
x.delete(n,m) cancella dalla posizione n alla posizione m

### toString
y = x.toString() restituisce il contenuto di x sotto forma di String (y è un oggetto String)

## Metodi della classe Math

sin, cos, tan, asin, acos, atan
Funzioni trigonometriche (argomento di tipo double, risultato di tipo double)

exp, log
Esponenziale e logaritmo in base e (argomento di tipo double, risultato di tipo double)

pow
pow(x,y) eleva x.alla y (x,y sono di tipo double, il risultato è di tipo double)

sqrt
Radice quadrata dell'argomento (argomento e risultato di tipo double)

ceil, floor, rint
Calcolano rispettivamente, l'intero inferiore, l'intero superiore, l'arrotondamento a intero (argomento e risultato sono di tipo double)

round
Converte un double in un long eseguendo un arrotondamento

abs
Calcola il valore assoluto (esistono quattro versioni: int, long, float e double; il risultato e l'argomento hanno sempre lo stesso tipo)

PI, E
Sono costanti (non metodi) corrispondenti ai valori predefiniti di pi-greco (3.14...) e della costante Neperiana e (2,71..)

## Riepilogo delle classi del package Javabook

MainWindow
Oggetti di tipo "frame" a cui possono essere agganciati dialoghi (InputBox, OutputBox, ecc.).
Ha due costruttori (uno senza argomenti e uno che accetta il titolo della finestra) e un metodo show che constente di rendere visibile la finestra su schermo.

OutputBox
Oggetto di tipo "dialog" che serve per mostrare risultati delle elaborazioni. Per essere costruiti tali oggetti necessitano di una MainWindow a cui "agganciarsi".Ha due costruttori, entrambi che richiedono un oggetto MainWindow, di cui uno accetta il titolo della finestra. Ha un metodo show che constente di rendere visibile la finestra su schermo. Inoltre ha i seguenti metodi:

- print(s) che consente di visualizzare il contenuto della stringa s (senza andare a capo al termine)
- printLine(s), come print ma con andata a capo finale
- skipLine(n) che consente di saltare n linee nella visualizzazione

InputBox
Oggetto di tipo "dialog" che serve per mostrare risultati delle elaborazioni, con caratteristiche analoghe a OutputBox.. Ha diversi metodi del tipo get<Tipo> dove <Tipo> può essere Integer, Double, Float, String. Tutti questi metodi assomigliano nella forma al seguente esempio:

```
int x = inputBox.getInteger("Inserisci un intero");
```

ResponseBox
Visualizza un dialogo che consente di inserire una risposta positiva o negativa mediante due appositi bottoni. Esempio di utilizzo:

```
MainWindow  finestra = new MainWindow( );
ResponseBox yesNoBox = new ResponseBox( finestra );int selection =
yesNoBox.prompt("Premi un bottone");
switch (selection) {    case ResponseBox.YES:
                messageBox.show("Hai premuto Yes");
                break;  case ResponseBox.NO:
                messageBox.show("Hai premuto No");
                break;
}
```

ListBox
Visualizza un dialogo che consente di effettuare una selezione da una lista. Esempio di utilizzo:

```
MainWindow finestra = new MainWindow( );
ListBox colorList = new ListBox(finestra, "Select Color" );
colorList.addItem("Magenta");
colorList.addItem("Cyan");
colorList.addItem("Red");
colorList.addItem("Blue");
colorList.addItem("Green");
int selection = colorList.getSelectedIndex(); // da 0 (Magenta) a 4 (Green)
```