

Linguaggio Java

- Testo adottato: S. Mizzaro, **Introduzione alla programmazione con il linguaggio Java.**
- Altri testi consigliati: G. Pighizzini, M. Ferrari, **Dai fondamenti agli oggetti Addison Wesley, 2005.**
- Introduzione a Java (dispensa) (www.dimi.uniud.it/demis)
- trasparenze sul linguaggio Java (www.dimi.uniud.it/demis)
- Manuale Java online (java.sun.com)

Linguaggio Java: sintassi

Gli elementi costitutivi di un programma Java sono chiamati **token**

I token sono suddivisi in 5 categorie:

- parole chiave
- letterali
- identificatori
- operatori
- separatori

N.B. oltre ai token un programma Java può includere spazi e commenti che il compilatore ignora.

Parole chiave

Le **parole chiave** hanno un uso e un significato speciale in Java.

abstract	boolean	break	byte	case	catch	char	class	const	goto
default	do	double	else	extends	final	finally	float	for	package
if	implements	import	instanceof	int	interface	long	native	new	synchronized
private	protected	public	return	short	static	strictfp	super	switch	
this	throw	throws	transient	try	void	volatile	while	continue	

Letterali

I **letterali** si dividono in

- numeri interi (es: `-679`)
- numeri decimali o in virgola mobile (es: `+23.789`)
- valori booleani (`true` e `false`)
- caratteri UNICODE (es: `'a'`)
- stringhe (es: `"pippo"`)
- il riferimento nullo `null`

Identificatori

Gli **identificatori** rappresentano nomi che possono essere assegnati a variabili, metodi, classi, ecc.

Un identificatore è composto esclusivamente da lettere, numeri e caratteri `_` e `$` e deve cominciare con una lettera, con `_`, oppure con `$`.

Un identificatore non può essere una parola chiave, un letterale booleano, un letterale nullo.

Identificatori validi: PIPPO, H2So4, \$bank, nome_var

Identificatori illegali: 2PIPPO, nome-var, int, true

N.B. Java distingue fra lettere maiuscole e minuscole pertanto i due identificatori Pippo e PIPPO sono da considerarsi distinti.

Operatori

Gli **operatori** indicano un tipo di calcolo da eseguire su uno o piú dati, detti operandi, che possono essere letterali, variabili o risultati di valutazioni di espressioni

=	>	<	!	?	:	~	
==	<=	>=	!=	&&		++	
--	+	-	*	/	&		
^	+=	-=	*=	/=	&=	=	^=

Separatori

I **separatori** sono i “segni di punteggiatura” del linguaggio

() { } [] ; , .

Commenti

<pre>/* questo e un commento */</pre>	tutti i caratteri da /* a */ sono ignorati
<pre>/** questo è un commento */</pre>	come /* */ ed inoltre permette di creare automaticamente la documentazione Javadoc
<pre>// questo è un commento</pre>	tutti i caratteri da // fino a fine linea sono ignorati

Esempio

```
/** Esempio
    di classe
 */
public class VotiEsame
{
    private int voti[]; // variabile di tipo vettore di interi

    public VotiEsame(int n) { voti = new int[n]; // inizializza un vettore di lunghezza n }

    // genera un vettore contenente interi pseudo-casuali compresi tra 15 e 31
    public void generavoti()
    {
        int i=0;
        Random r=new Random();
        for (i=0;i<voti.length;i++)
            voti[i]=15+r.nextInt(16);
        System.out.println("Vettore generato!");
    }

    // stampa vettore
    public void stampavoti()
    {
        int i=0;
        for (i=0;i<voti.length;i++)
            System.out.println("Elemento in posizione "+i+": "+voti[i]);
    }
}
```

Esempio

```
/** Esempio
  di classe
  */
public class VotiEsame
{
    private int voti[]; // variabile di tipo vettore di interi

    public VotiEsame(int n) { voti = new int[n]; /* inizializza un vettore di lunghezza n*/ }

    // genera un vettore contenente interi pseudo-casuali compresi tra 15 e 31
    public void generavoti()
    {
        int i=0;
        Random r=new Random();
        for (i=0;i<voti.length;i++)
            voti[i]=15+r.nextInt(16);
        System.out.println("Vettore generato!");
    }

    // stampa vettore
    public void stampavoti()
    {
        int i=0;
        for (i=0;i<voti.length;i++)
            System.out.println("Elemento in posizione "+i+": "+voti[i]);
    }
}
```

Esempio

```
/** Esempio  
di classe
```

```
*/
```

```
public class VotiEsame  
{
```

```
    private int voti[]; // variabile di tipo vettore di interi
```

```
    public VotiEsame(int n) { voti = new int[n]; /* inizializza un vettore di lunghezza n*/ }
```

```
    // genera un vettore contenente interi pseudo-casuali compresi tra 15 e 31
```

```
    public void generavoti()
```

```
    {
```

```
        int i=0;
```

```
        Random r=new Random();
```

```
        for (i=0;i<voti.length;i++)
```

```
            voti[i]=15+r.nextInt(16);
```

```
        System.out.println("Vettore generato!");
```

```
    }
```

```
    // stampa vettore
```

```
    public void stampavoti()
```

```
    {
```

```
        int i=0;
```

```
        for (i=0;i<voti.length;i++)
```

```
            System.out.println("Elemento in posizione "+i+": "+voti[i]);
```

```
    }
```

5 commenti

Esempio

```
/** Esempio
  di classe
  */
public class VotiEsame
{
    private int voti[]; // variabile di tipo vettore di interi

    public VotiEsame(int n) { voti = new int[n]; /* inizializza un vettore di lunghezza n*/ }

    // genera un vettore contenente interi pseudo-casuali compresi tra 15 e 31
    public void generavoti()
    {
        int i=0;
        Random r=new Random();
        for (i=0;i<voti.length;i++)
            voti[i]=15+r.nextInt(16);
        System.out.println("Vettore generato!");
    }

    // stampa vettore
    public void stampavoti()
    {
        int i=0;
        for (i=0;i<voti.length;i++)
            System.out.println("Elemento in posizione "+i+": "+voti[i]);
    }
}
```

Esempio

17 parole chiave

```
/** Esempio
  di classe
  */
public class VotiEsame
{
    private int voti[]; // variabile di tipo vettore di interi

    public VotiEsame(int n) { voti = new int[n]; /* inizializza un vettore di lunghezza n*/ }

    // genera un vettore contenente interi pseudo-casuali compresi tra 15 e 31
    public void generavoti()
    {
        int i=0;
        Random r=new Random();
        for (i=0;i<voti.length;i++)
            voti[i]=15+r.nextInt(16);
        System.out.println("Vettore generato!");
    }

    // stampa vettore
    public void stampavoti()
    {
        int i=0;
        for (i=0;i<voti.length;i++)
            System.out.println("Elemento in posizione "+i+": "+voti[i]);
    }
}
```

Esempio

```
/** Esempio
    di classe
*/
public class VotiEsame
{
    private int voti[]; // variabile di tipo vettore di interi

    public VotiEsame(int n) { voti = new int[n]; /* inizializza un vettore di lunghezza n*/ }

    // genera un vettore contenente interi pseudo-casuali compresi tra 15 e 31
    public void generavoti()
    {
        int i=0;
        Random r=new Random();
        for (i=0;i<voti.length;i++)
            voti[i]=15+r.nextInt(16);
        System.out.println("Vettore generato!");
    }

    // stampa vettore
    public void stampavoti()
    {
        int i=0;
        for (i=0;i<voti.length;i++)
            System.out.println("Elemento in posizione "+i+": "+voti[i]);
    }
}
```

Esempio

```
/** Esempio
  di classe
  */
public class VotiEsame
{
    private int voti[]; // variabile di tipo vettore di interi

    public VotiEsame(int n) { voti = new int[n]; /* inizializza un vettore di lunghezza n*/ }

    // genera un vettore contenente interi pseudo-casuali compresi tra 15 e 31
    public void generavoti()
    {
        int i=0;
        Random r=new Random();
        for (i=0;i<voti.length;i++)
            voti[i]=15+r.nextInt(16);
        System.out.println("Vettore generato!");
    }

    // stampa vettore
    public void stampavoti()
    {
        int i=0;
        for (i=0;i<voti.length;i++)
            System.out.println("Elemento in posizione "+i+": "+voti[i]);
    }
}
```

9 letterali

Esempio

```
/** Esempio
  di classe
  */
public class VotiEsame
{
    private int voti[]; // variabile di tipo vettore di interi

    public VotiEsame(int n) { voti = new int[n]; /* inizializza un vettore di lunghezza n*/ }

    // genera un vettore contenente interi pseudo-casuali compresi tra 15 e 31
    public void generavoti()
    {
        int i=0;
        Random r=new Random();
        for (i=0;i<voti.length;i++)
            voti[i]=15+r.nextInt(16);
        System.out.println("Vettore generato!");
    }

    // stampa vettore
    public void stampavoti()
    {
        int i=0;
        for (i=0;i<voti.length;i++)
            System.out.println("Elemento in posizione "+i+": "+voti[i]);
    }
}
```


Esempio

36 identificatori

```
/** Esempio
  di classe
  */
public class VotiEsame
{
    private int voti[]; // variabile di tipo vettore di interi

    public VotiEsame(int n) { voti = new int[n]; /* inizializza un vettore di lunghezza n*/ }

    // genera un vettore contenente interi pseudo-casuali compresi tra 15 e 31
    public void generavoti()
    {
        int i=0;
        Random r=new Random();
        for (i=0;i<voti.length;i++)
            voti[i]=15+r.nextInt(16);
        System.out.println("Vettore generato!");
    }

    // stampa vettore
    public void stampavoti()
    {
        int i=0;
        for (i=0;i<voti.length;i++)
            System.out.println("Elemento in posizione "+i+": "+voti[i]);
    }
}
```

Esempio

```
/** Esempio
  di classe
  */
public class VotiEsame
{
    private int voti[]; // variabile di tipo vettore di interi

    public VotiEsame(int n) { voti = new int[n]; /* inizializza un vettore di lunghezza n*/ }

    // genera un vettore contenente interi pseudo-casuali compresi tra 15 e 31
    public void generavoti()
    {
        int i=0;
        Random r=new Random();
        for (i=0;i<voti.length;i++)
            voti[i]=15+r.nextInt(16);
        System.out.println("Vettore generato!");
    }

    // stampa vettore
    public void stampavoti()
    {
        int i=0;
        for (i=0;i<voti.length;i++)
            System.out.println("Elemento in posizione "+i+": "+voti[i]);
    }
}
```

Esempio

15 operatori

```
/** Esempio
  di classe
  */
public class VotiEsame
{
    private int voti[]; // variabile di tipo vettore di interi

    public VotiEsame(int n) { voti = new int[n]; /* inizializza un vettore di lunghezza n*/ }

    // genera un vettore contenente interi pseudo-casuali compresi tra 15 e 31
    public void generavoti()
    {
        int i=0;
        Random r=new Random();
        for (i=0;i<voti.length;i++)
            voti[i]=15+r.nextInt(16);
        System.out.println("Vettore generato!");
    }

    // stampa vettore
    public void stampavoti()
    {
        int i=0;
        for (i=0;i<voti.length;i++)
            System.out.println("Elemento in posizione "+i+": "+voti[i]);
    }
}
```

Esempio

```
/** Esempio
  di classe
  */
public class VotiEsame
{
    private int voti[]; // variabile di tipo vettore di interi

    public VotiEsame(int n) { voti = new int[n]; /* inizializza un vettore di lunghezza n*/ }

    // genera un vettore contenente interi pseudo-casuali compresi tra 15 e 31
    public void generavoti()
    {
        int i=0;
        Random r=new Random();
        for (i=0;i<voti.length;i++)
            voti[i]=15+r.nextInt(16);
        System.out.println("Vettore generato!");
    }

    // stampa vettore
    public void stampavoti()
    {
        int i=0;
        for (i=0;i<voti.length;i++)
            System.out.println("Elemento in posizione "+i+": "+voti[i]);
    }
}
```

Esempio

```
/** Esempio  
di classe
```

```
*/
```

```
public class VotiEsame
```

```
{
```

```
    private int voti[]; // variabile di tipo vettore di interi
```

```
    public VotiEsame(int n) { voti = new int[n]; /* inizializza un vettore di lunghezza n*/ }
```

```
    // genera un vettore contenente interi pseudo-casuali compresi tra 15 e 31
```

```
    public void generavoti()
```

```
{
```

```
    int i=0;
```

```
    Random r=new Random();
```

```
    for (i=0;i<voti.length;i++)
```

```
        voti[i]=15+r.nextInt(16);
```

```
    System.out.println("Vettore generato!");
```

```
}
```

```
    // stampa vettore
```

```
    public void stampavoti()
```

```
{
```

```
    int i=0;
```

```
    for (i=0;i<voti.length;i++)
```

```
        System.out.println("Elemento in posizione "+i+": "+voti[i]);
```

```
}
```

52 separatori

Tipi di dato in Java

- Java è un linguaggio fortemente tipato in quanto ogni variabile ed ogni espressione hanno associato un tipo.
- I tipi limitano l'insieme di valori che una variabile o un'espressione possono assumere.
- Tipo di dato = (Supporto, Operazioni)
- Tipi di dato primitivi e Tipi di dato astratti

Tipi di dato primitivi

- il tipo booleano `boolean` (valori: `true` e `false`)
- i tipi per i numeri interi `byte` (8 bit), `short` (16 bit), `int` (32 bit), `long` (64 bit).
- i tipi per i numeri decimali `float` (32 bit), `double` (64 bit)
- il tipo carattere `char`

Tipi di dato astratti

- In Java sono anche chiamati tipi **referimento**
- I tipi referimento sono classi, vettori, interfacce.

Variabili

- Una **variabile** è una locazione della memoria referenziabile mediante un identificatore.
- Una variabile di un tipo primitivo può contenere solo valori del supporto di tale tipo.
- Una variabile di tipo T, dove T è un tipo riferimento, può contenere un riferimento a qualunque istanza di T oppure il valore null.

Dichiarazione di variabili

- L'associazione di una variabile a un tipo ha luogo al momento della dichiarazione di tale variabile.

```
<tipo> <identificatore>;
```

oppure

```
<tipo> <identificatore1>, ..., <identificatoreN>;
```

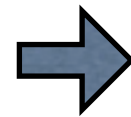
Esempi dichiarazioni

```
int X, pippo;  
double radice;  
float r,c;
```



Tipi primitivi

```
String Codice_fiscale;  
Persona p1,p2,p3;
```



Tipi riferimento

Assegnamento (=)

L'operatore di **assegnamento (=)** permette di memorizzare in una variabile un valore ottenuto dalla valutazione di una espressione.

```
<nome_variabile> = <espressione>;
```

- Una variabile per essere assegnata deve prima essere stata dichiarata;
- È lecito combinare dichiarazioni con assegnamenti. La sintassi è la seguente:
- `<tipo> <nome_variabile> = <espressione>;`
- **In generale, <identificatore> e <espressione> devono avere lo stesso tipo.**

Esempi di assegnamento

```
String parola="pippo";  
int x,y;  
double d;
```

⇒ dich. e ass. combinati

⇒ dichiarazioni

```
d = 3.1459;  
x = 1;  
y = (x*5)+4;  
x = parola;  
y = 3.29;  
z = 6;
```

⇒ errori di assegnamento

Conversioni di tipo

- A volte è necessario assegnare espressioni di tipo T1 a variabili di tipo T2, con T1 e T2 tipi di dato distinti. È sempre possibile?
- L'operazione di conversione di tipo è chiamata **casting** e può essere **implicita** o **esplicita**.

Casting implicito

- Il casting implicito è lecito quando non comporta nessuna perdita di precisione.

```
int i=5;  
double j=3.1456;
```

```
j = i; /* casting implicito lecito */
```

```
i = j; /* casting implicito illegale  
perdita di precisione */
```

Casting esplicito

- nel casting esplicito il tipo in cui si vuole convertire una variabile deve essere fornito esplicitamente.
- Può comportare perdita di precisione.

```
int i = 65;
```

```
char c;
```

```
double j = 3.1456;
```

```
i = (int) j; /* casting esplicito legale  
             con perdita di precisione */
```

```
c = (char) i; /* assegna a c il codice UNICODE i */
```

```
j = (double) i; /* casting esplicito legale  
                senza perdita di precisione */
```


Le classi

Un programma Java è un insieme di definizioni di classi. La sintassi per definire una **classe** è

```
public class <nome classe>
{
    <corpo della classe>
}
```

il corpo di una classe è costituito da dichiarazione di variabili (**attributi** della classe) e da definizioni di metodi (**comportamento** della classe)

Attributi di classe

Gli attributi sono variabili che permettono di modellare lo stato degli oggetti.

```
public class <nome classe>
{
    <dichiarazione attributi>
    <dichiarazione metodi>
}
```

Metodi

Un **metodo** pubblico ha la seguente sintassi:

```
public <tipo> <nome_metodo> (<lista_parametri>)  
{  
    <corpo del metodo>  
}
```

dove `tipo` è il tipo del valore di ritorno del metodo, `nome_metodo` è un identificatore, `lista_parametri` è una sequenza (eventualmente vuota) di coppie `tipo parametro` che rappresenta i parametri di input del metodo, infine `corpo del metodo` è la definizione del metodo vera e propria.

Se il tipo di ritorno è diverso da `void` (parola chiave che denota l'assenza di un valore di ritorno), il corpo del metodo deve terminare con la parola chiave `return` seguita dal valore che deve essere restituito.

Metodo costruttore

Un metodo **costruttore** pubblico è un metodo speciale che ha lo stesso nome della classe e che permette di creare oggetti (istanze) della classe. Nella definizione di tale metodo non compare il tipo di ritorno. Tipicamente un metodo costruttore definisce una lista di parametri attraverso i quali si assegnano dei valori agli attributi della classe.

```
public <nome_costruttore> (<lista_parametri>)  
{  
    <corpo del metodo>  
}
```

Un **oggetto** O di una **classe** C si crea invocando il metodo costruttore della classe e istanziando gli opportuni parametri. La sintassi è la seguente:

```
new <nome_costruttore>(<valori_parametri>);
```

Esempi

```
public class Matrice
{

    char [][] matrix;
    int numero_righe;
    int numero_colonne;

    public Matrice(int n, int m)
    {
        matrix=new char [n][m];
        numero_righe=n;
        numero_colonne=m;
    }

    public void stampaMatrice()
    {
        int i,j;
        for(i=0;i<numero_righe;i++){
            for(j=0;j<numero_colonne;j++)
                System.out.print(matrix[j][i]+" ");
            System.out.println("");
        }
    }
}
```

Esempi di metodi

tipo

nome

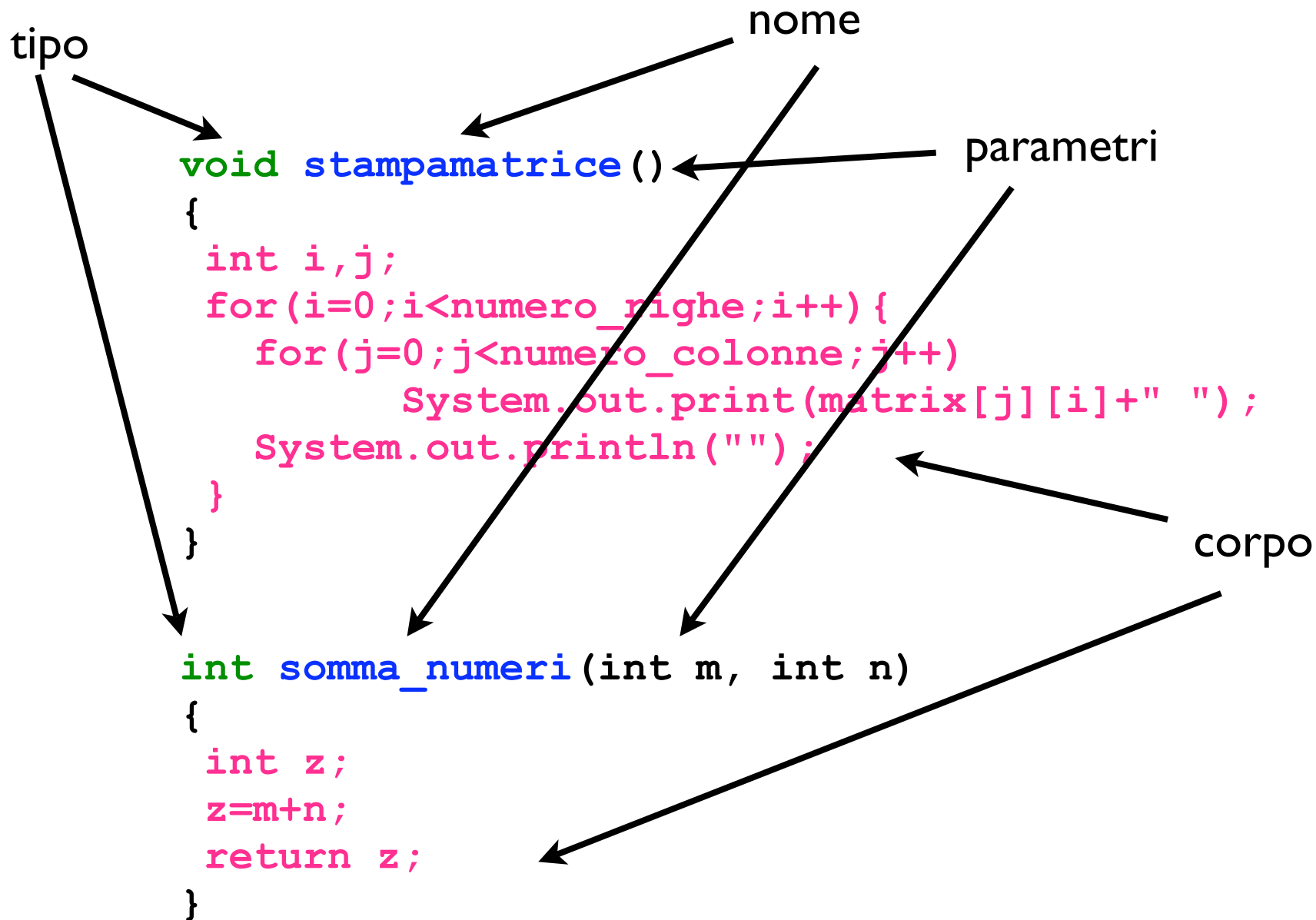
parametri

```
void stampamatrice ()  
{  
    int i,j;  
    for(i=0;i<numero_righe;i++){  
        for(j=0;j<numero_colonne;j++){  
            System.out.print(matrix[j][i]+" ");  
            System.out.println("");  
        }  
    }  
}
```

corpo

```
int somma_numeri(int m, int n)  
{  
    int z;  
    z=m+n;  
    return z;  
}
```

Esempi di metodi



Invocazione di metodi

Dato un **oggetto** O di una **classe** C, creato invocando il metodo costruttore della classe C, e' possibile invocare i metodi pubblici di O con la seguente sintassi :

```
<nome_oggetto>.<nome_metodo>(<lista_parametri>)
```

<lista_parametri> rappresenta la lista dei parametri attuali sui quali eseguire il metodo.

```
{  
    Matrice M = new Matrice(5,10);  
  
    // codice per caricare i dati nella matrice  
    // omezzo...  
  
    M.stampaMatrice();  
}
```


Osservazione

- Per invocare un metodo definito in una classe **C** all'interno di qualche metodo della stessa **C** si utilizza la notazione

`<nome_metodo> (<lista_parametri>)`

o in alternativa la notazione

`this.<nome_metodo> (<lista_parametri>)`

Esempio

```
public Class Esempio
{

    public int max(int u,int v)
    {
        if (u>v)
            return u;
        else
            return v;
    }

    public int usa_max(int x,int y)
    {
        int m;

        // invocazione metodo max
        // definito nella classe
        m = max(x,y) ;

        System.out.println(m) ;
    }
}
```

Overloading

In una classe è possibile definire metodi con nome uguale ma parametri distinti (e pertanto comportamento distinto). In questo caso si parla di **overloading**.

```
public class Esempio_overloading
{

    public void stampaDati(int n)
    {
        System.out.println("stampa numero "+n);
    }

    public void stampaDati(String s)
    {
        System.out.println("stampa stringa "+s);
    }

}
```

Esempi

```
public class Matrice  
{
```

```
    char [][] matrix;  
    int numero_righe;  
    int numero_colonne;
```

attributi

```
    public Matrice(int n, int m)  
    {  
        matrix=new char [n][m];  
        numero_righe=n;  
        numero_colonne=m;  
    }
```

costruttore

```
    public void stampaMatrice()  
    {  
        int i,j;  
        for(i=0;i<numero_righe;i++){  
            for(j=0;j<numero_colonne;j++){  
                System.out.print(matrix[j][i]+" ");  
                System.out.println("");  
            }  
        }  
    }
```

metodo

Esempi

```
class Matrice  
{
```

```
    char [][] matrix;  
    int numero_righe;  
    int numero_colonne;
```

← attributi

```
    Matrice(int n, int m)
```

```
    {  
        matrix=new char [n][m];  
        numero_righe=n;  
        numero_colonne=m;  
    }
```

← costruttore

```
    void stampaMatrice()
```

```
    {  
        int i,j;  
        for(i=0;i<numero_righe;i++){  
            for(j=0;j<numero_colonne;j++){  
                System.out.print(matrix[j][i]+" ");  
                System.out.println("");  
            }  
        }  
    }
```

← metodo

```
}
```

Esempi

```
class Esempio_Uso_Classe_Matrx
{

    void creaScacchieradispari ()
    {
        int i,j;
        char colore='b' ;
        Matrix scacchiera= new Matrix(3,3) ;

        for(i=0;i<numero_righe;i++)
        {
            for(j=0;j<numero_colonne;j++){
                scacchiera[i][j]=colore;
                if (colore=='b' )
                    colore='n' ;
                else
                    colore='b' ;
            }
        }
    }
}
```

Blocchi

Il codice Java è diviso in **blocchi** delimitati da parentesi graffe aperte e chiuse.

I **blocchi** possono essere annidati gli uni dentro gli altri.

```
Public Class Esempio
```

```
{  
    // def. attributi  
    int x;  
    String s;  
  
    // def. metodi  
    public Esempio(int y, String s)  
    {  
        // corpo del costruttore  
    }  
  
    public void metodo()  
    {  
        // corpo del metodo  
    }  
}
```

Ambito delle variabili

È possibile dichiarare ed usare variabili in ogni blocco del programma. L'**ambito** (o scope) in cui tali variabili possono essere utilizzate è limitato al blocco in cui sono state dichiarate e ai sottoblocchi eventualmente annidati.

Le variabili dichiarate nel corpo di una classe definiscono gli **attributi** della classe.

Le variabili dichiarate nel corpo di un metodo sono dette variabili **locali**, in quanto il loro ambito è ristretto al corpo del metodo.

Le variabili locali vengono create al momento dell'esecuzione di un metodo e distrutte quando esso termina.

Esempio

```
Public Class Esempio
{
    int x=5;

    // def. metodi
    public void metodoA(int y, String s)

    {
        char c='B';

        System.out.println("valore di x: "+x);
        System.out.println("valore di c: "+c);

    }

    public void metodoB()
    {

        String s="pippo";

        System.out.println("valore di s: "+s);
        System.out.println("valore di c: "+c);

    }

}
```

Esempio

```
Public Class Esempio
{
    int x=5;

    // def. metodi
    public void metodoA(int y, String s)

    {
        char c='B';

        System.out.println("valore di x: "+x);
        System.out.println("valore di c: "+c);

    }

    public void metodoB()
    {

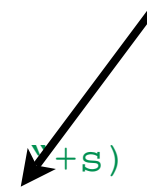
        String s="pippo";

        System.out.println("valore di s: "+s)
        System.out.println("valore di c: "+c);

    }

}
```

Errore: la variabile c non è visibile in questo blocco!



Ambito (cont.)

Che succede se dichiaro variabili con lo stesso nome?

Non è possibile dichiarare variabili con lo stesso nome all'interno dello stesso blocco.

```
public metodo()  
{  
    int i;  
    char i; /* errore i è già definita! */  
    ...  
}
```

È possibile dichiarare variabili con lo stesso nome all'interno di blocchi distinti anche se sono annidati. Ciò non crea alcun conflitto, in quanto la dichiarazione più interna nasconde le eventuali dichiarazioni più esterne.

Esempio

Che valore stampa l'esecuzione del metodo metodoA()? E metodoB()?

```
Public Class Esempio
{
    private int x=5;

    public void metodoA()
    {
        char x='A';

        System.out.println("valore di x: "+x);
    }

    public void metodoB()
    {
        System.out.println("valore di x: "+x);
    }
}
```

Esempio

Che valore stampa l'esecuzione del metodo metodoA()? E metodoB()?

```
Public Class Esempio
{
    private int x=5;

    public void metodoA()
    {
        char x='A';

        System.out.println("valore di x: "+x);
    }

    public void metodoB()
    {
        System.out.println("valore di x: "+x);
    }
}
```

Esempio

Che valore stampa l'esecuzione del metodo metodoA()? E metodoB()?

```
Public Class Esempio
{
    private int x=5;

    public void metodoA()
    {
        char x='A';

        System.out.println("valore di x: "+x);
    }

    public void metodoB()
    {
        System.out.println("valore di x: "+x);
    }
}
```

Stampa il carattere 'A'



Esempio

Che valore stampa l'esecuzione del metodo metodoA()? E metodoB()?

```
Public Class Esempio
{
    private int x=5;

    public void metodoA()
    {
        char x='A';

        System.out.println("valore di x: "+x);
    }

    public void metodoB()
    {
        System.out.println("valore di x: "+x);
    }
}
```

Stampa il carattere 'A'



Stampa l'intero 5



Espressioni

Un'**espressione** permette di calcolare un risultato applicando opportuni operatori a un certo numero di operandi, che possono essere variabili, letterali, valori di ritorno di metodi o altre espressioni.

`2*X+Y //esempio di espressione`

N.B. il tipo degli operandi deve essere compatibile con il tipo degli operatori. (Es. non posso moltiplicare un carattere e un numero!)

Tipicamente il risultato di una espressione è memorizzato in una variabile mediante l'operatore di assegnamento.

`Z = 2*X+Y;`

Operatori aritmetici

+	addizione
-	sottrazione
*	moltiplicazione
/	divisione
%	resto della divisione intera

Questi operatori si applicano sia a numeri interi che a numeri frazionari. L'operatore / produce come risultato un intero se applicato a operandi interi, mentre ritorna un valore frazionario se applicato a operandi frazionari.

Il risultato che si ottiene applicando tali operatori è di tipo numerico (intero o frazionario).

$X \% Y$, produce il resto della divisione intera fra gli interi X e Y .
Es. $12 \% 7$ ritorna come valore 5.

Esempi di espressioni aritmetiche

```
int X=4, Y=12;  
double Z = 3.14;  
double W;
```

```
W= ((2*X-Z*Z) / 2) + 1;
```

```
Y = (Y/X) % 2;
```

Esempi di espressioni aritmetiche

```
int X=4, Y=12;  
double Z = 3.14;  
double W;
```

W ha valore -0.0702

```
W = ((2*X - Z*Z) / 2) + 1;
```

```
Y = (Y/X) % 2;
```

Y ha valore 1

Operatori relazionali

<	minore
<=	minore o uguale
>	maggiore
>=	maggiore o uguale
==	uguale
!=	diverso

Questi operatori confrontano due espressioni e ritornano un valore booleano (**true** oppure **false**). Permettono di costruire espressioni booleane.

N.B. non si confonda l'operatore di uguaglianza == con l'operatore di assegnamento =.

Esempi di espressioni booleane

```
int X=3,Y=5;  
double Z = 3.14;  
boolean W;  
boolean K;  
  
W = ((X+1) == (Y*2)%6);  
  
K = Z < X;
```

Esempi di espressioni booleane

```
int X=3, Y=5;  
double Z = 3.14;  
boolean W;
```

W prende il valore true

```
W = ((X+1) == (Y*2) % 6);
```

```
Y = Z < X;
```

Y prende il valore false

Operatori logici

&&	and logico o congiunzione
	or logico o disgiunzione
!	not o negazione

Questi operatori permettono di combinare assieme più espressioni booleane. Ritornano sempre un risultato di tipo booleano.

Operatori logici

e₁	e₂	e₁ && e₂	e₁ e₂	!e₁
true	true	true	true	false
true	false	false	true	--
false	true	false	true	true
false	false	false	false	--

Esempi di espressioni booleane

```
int X=3,Y=5;  
double Z = 3.14;  
boolean W1,W2;
```

```
W1 = ((X+1) == ((Y*2)%6)) && (3<4) || !(9>=Z);
```

```
W2 = ((X+1) != ((Y*2)%6)) && (3<4) || !(9>=Z);
```

Esempi di espressioni booleane

```
int X=3,Y=5;
```

```
double Z = 3.14;
```

```
boolean W1,W2;
```

W1 prende il valore true

```
W1 = ((X+1) == ((Y*2)%6)) && (3<4) || !(9>=Z);
```

```
W2 = ((X+1) != ((Y*2)%6)) && (3<4) || !(9>=Z);
```

W2 prende il valore false

Operatori ++ e --

Questi operatori sono utilizzati, in notazione sia prefissa che postfissa, per **incrementare** o **decrementare** di 1 il valore di una variabile.

Gli usi prefisso e postfisso di tali operatori non sono equivalenti.

Esempio:

```
{
int K, X=3, Y, W, Z=4;

Z++;           // postincremento
Y = X++;      // postincremento
W = ++X;      // preincremento
K = W--;      // postdecremento
}
```

Operatori ++ e --

Questi operatori sono utilizzati, in notazione sia prefissa che postfissa, per **incrementare** o **decrementare** di 1 il valore di una variabile.

Gli usi prefisso e postfisso di tali operatori non sono equivalenti.

Esempio:

```
{  
int K, X=3, Y, W, Z=4;  
  
Z++; // postincremento  
Y = X++; // postincremento  
W = ++X; // preincremento  
K = W--; // postdecremento  
}
```

Z prende il valore 5

Y prende 3 e X diventa 4

K prende 5 e W diventa 4

W prende 5 e X diventa 5

Blocco sequenziale

Permette di eseguire un insieme di istruzioni in sequenza.

È delimitato da una coppia di parentesi graffe.

Nel caso la sequenza si riduca a una sola istruzione le parentesi graffe possono essere omesse (tranne il caso di blocchi denotanti corpi di classi o metodi).

Può contenere dichiarazioni di variabili in qualsiasi punto.

Definizione:

```
{  
  <istruzione 1>  
  <istruzione 2>  
  <istruzione 3>  
  ...  
  <istruzione n>  
}
```

Esempio:

```
{  
  int x=1;  
  double z;  
  boolean b;  
  z=x*4;  
  b= !(z==x);  
  System.out.println(b);  
}
```

Strutture di controllo

Sono istruzioni che permettono di modificare il flusso di esecuzione di un metodo

- istruzioni **condizionali**
- istruzioni **iterative** (cicli)

Istruzioni condizionale

La più comune istruzione condizionale è il blocco **if-else**, la cui sintassi è la seguente:

```
if (<condizione>
    <blocco di istruzioni 1>
else
    <blocco di istruzioni 2>
```

La condizione deve essere un'espressione booleana. Se la condizione è valutata **true** allora si esegue il blocco di istruzioni 1, altrimenti si esegue il blocco di istruzioni 2

Le parentesi graffe delimitanti i blocchi possono essere omesse quando i blocchi di istruzioni sono costituiti da un'unica istruzione.

Istruzioni condizionali - esempio

```
{  
int X =-5;  
if (X>0)  
    System.out.println("X è positivo");  
else  
    {  
        System.out.print("X è negativo ");  
        System.out.println("oppure nullo");  
    }  
}
```


Istruzioni condizionali

A volte il ramo **else** di una istruzione condizionale può essere assente. In tal caso l'istruzione assume la seguente forma:

```
if (<condizione>
    <blocco di istruzioni>
```

Esegui il blocco di istruzioni se la condizione è valutata **true**.

```
{ int X=4; if (X%2==0)
  System.out.println("X è pari");
}
```

Istruzioni condizionali

È possibile unire più istruzioni if-else per esprimere più possibilità condizionali. La sintassi è la seguente

```
if (<condizione 1>
    <blocco di istruzioni 1>
else
    if (<condizione 2>
        <blocco di istruzioni 2>
        ...
    else
        <blocco di istruzioni n>
```

Se la condizione 1 è valutata **true**, esegui il blocco di istruzioni 1, altrimenti se la condizione 2 è valutata **true**, esegui il blocco di istruzioni 2, ... altrimenti se nessuna condizione è verificata, esegui il blocco di istruzioni n.

Istruzioni condizionali - esempio

```
{
int X=44;
if (X>0 && X<=10)
    System.out.println(X);
else
    if (X==11)
        System.out.println("Jack");
    else
        if (X==12)
            System.out.println("Regina");
    else
        System.out.println("Re");
}
```

Istruzioni condizionali:

È possibile esprimere più possibilità condizionali, in maniera molto intuitiva, utilizzando l'istruzione **switch**.

```
switch (<espressione>) {  
    case <letterale 1>: <blocco istruzioni 1>  
    case <letterale 2>: <blocco istruzioni 2>  
    ...  
    default: <blocco istruzioni n>  
}
```

L'**espressione** deve produrre un risultato **byte**, **short**, **int** o **char**.

Viene valutata l'**espressione**. Il risultato viene confrontato con ciascun **case** nell'ordine in cui questi compaiono. Se il risultato dell'i-esimo **case** coincide con il risultato dell'espressione allora tutti i blocchi di istruzioni da i a n vengono eseguiti. Se il risultato non coincide con nessun **case** allora si esegue il blocco di istruzioni n relativo al **default**.


Istruzioni condizionali:

```
{int X=2;  
  
  switch(X) {  
    case 1: System.out.println("Uno");  
    case 2: System.out.println("Due");  
    default: System.out.println("Default");  
  }  
}
```

Istruzioni condizionali:

```
{int X=2;  
  
switch(X) {  
    case 1: System.out.println("Uno");  
    case 2: System.out.println("Due");  
    default: System.out.println("Default");  
}  
}
```

Questo codice stampa le
stringhe Due e Default



Istruzioni condizionali:

Nell'istruzione **switch**, se si vuole eseguire solo il blocco di istruzioni corrispondente al letterale uguale al risultato dell'espressione, si deve terminare il blocco di istruzioni con la parola chiave **break**.

```
{int X=2;

  switch(X) {
    case 1: System.out.println("Uno");break;
    case 2: System.out.println("Due");break;
    default: System.out.println("Default");
  }
}
```

Istruzioni condizionali:

Nell'istruzione **switch**, se si vuole eseguire solo il blocco di istruzioni corrispondente al letterale uguale al risultato dell'espressione, si deve terminare il blocco di istruzioni con la parola chiave **break**.

```
{int X=2;

switch (X) {
    case 1: System.out.println("Uno");break;
    case 2: System.out.println("Due");break;
    default: System.out.println("Default");
}
}
```

Questo codice stampa la
stringa **Due**



Istruzioni iterative: while

L'istruzione **while** è una istruzione iterativa che permette di ripetere più volte l'esecuzione di una porzione di codice. La sintassi è la seguente:

```
while (<condizione>)  
    <blocco di istruzioni da ripetere>
```

la condizione deve essere un'espressione booleana. Finché il risultato dell'espressione è **true**, si ripete il blocco di istruzioni successivo.

While (esempio)

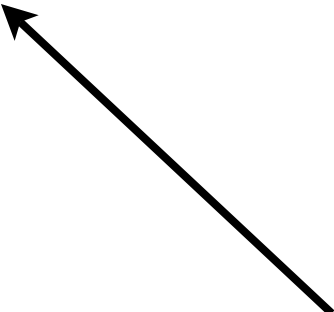
```
{int i=0;

while(i<10) {
    System.out.println(i);
    i++;
}
}
```

While (esempio)

```
{int i=0;

while(i<10) {
    System.out.println(i);
    i++;
}
}
```



Questo codice stampa gli
interi da 0 a 9 e alla fine del
ciclo i vale 10

Istruzioni iterative: do-

L'istruzione **do-while** è una istruzione iterativa che permette di ripetere più volte l'esecuzione di una porzione di codice. A differenza dell'istruzione **while**, le istruzioni da ripetere sono poste prima della condizione da valutare. Pertanto il blocco di istruzioni si ripete sempre almeno una volta!

Sintassi:

```
do
    <blocco di istruzioni da ripetere>
while (<condizione>);
```

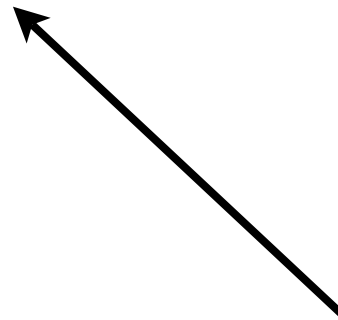
la condizione deve essere un'espressione booleana. Finché il risultato dell'espressione è **true**, si ripete il blocco di istruzioni.

Do-while (esempio)

```
{int i=0;  
  do  
  {  
    i++;  
    System.out.println(i);  
  }  
  while (i<10);  
}
```

Do-while (esempio)

```
{int i=0;
do
{
    i++;
    System.out.println(i);
}
while(i<10);
}
```



Questo codice stampa gli interi da 1 a 10 e alla fine del ciclo i vale 10

Istruzioni iterative: for

L'istruzione **for** è una istruzione iterativa che permette di ripetere più volte l'esecuzione di una porzione di codice.

Sintassi:

```
for (<inizializzazione>; <condizione>; <espressione>)  
  <blocco di istruzioni da ripetere>
```

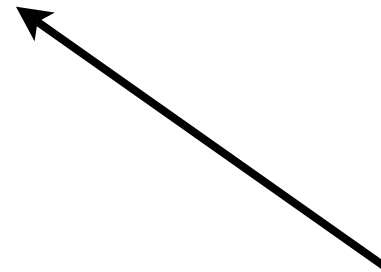
L'espressione **inizializzazione** è tipicamente un'istruzione di assegnamento, che attribuisce ad una variabile di controllo un valore iniziale. La **condizione** è un'espressione booleana che tipicamente confronta il valore della variabile di controllo con un valore limite. L'**espressione** specifica il modo in cui la variabile di controllo deve essere modificata al termine di ogni iterazioni del ciclo.

For (esempio)

```
{int i;  
  
    for (i=0; i<11; i++)  
        System.out.println(i);  
}
```


For (esempio)

```
{int i;  
  
  for (i=0; i<11; i++)  
    System.out.println(i);  
}
```



Questo codice stampa gli
interi da 0 a 10 e alla fine del
ciclo i vale 11

I vettori (o array)

Il **vettore** (o **array**) è un oggetto che fornisce lo spazio di memoria per un elenco di elementi tutti dello stesso tipo componente T.

Un vettore deve essere innanzitutto dichiarato mediante la seguente sintassi:

```
<tipo>[] <nome_vettore>;
```

```
int [] x,y; //dichiara x e y array di interi
```

```
String [] frase; // dichiara un array di  
                // stringhe di nome frase
```

Creazione di un vettore

Il tipo vettore è un tipo riferimento. Un riferimento ad un array si crea mediante l'operatore new. Quando si crea un vettore è necessario specificare la dimensione (ovvero il numero di celle allocate).

```
new <tipo> [<dimensione>];
```

```
x = new int [10]; //alloca un vettore di 10
                  //celle e memorizza il
                  //riferimento nella
                  //variabile vettore X
```

```
frase = new String[3]; //la variabile frase
                       //può contenere 3
                       stringhe
```

Creazione di un vettore

Un altro modo per creare un riferimento a un array consiste nel dare esplicitamente l'elenco dei valori da memorizzare in fase di dichiarazione.

```
int[] x={14,12,1,24,7,1,2,9,9,11};
```

```
String[] frase={"questa","e'una","frase"};
```

Accesso e scrittura di un vettore

Le componenti di un array sono indicizzate mediante un numero intero progressivo. Se l'array contiene n celle, la prima componente ha indice 0 e l'ultima ha indice n-1.

La componente i-esima di un array può essere reperita o scritta usando la seguente sintassi:

```
<variabile di tipo array>[i]
```

```
char[] lettere={'a','b','c'};  
char j;
```

```
System.out.println(lettere[2]);  
j=lettere[0];  
lettere[1]= 'k';
```

Accesso e scrittura di un vettore

Le componenti di un array sono indicizzate mediante un numero intero progressivo. Se l'array contiene n celle, la prima componente ha indice 0 e l'ultima ha indice n-1.

La componente i-esima di un array può essere reperita o scritta usando la seguente sintassi:

```
<variabile di tipo array>[i]
```

char[] lettere={'a','b','c'};

char j;

System.out.println(lettere[2]), stampa 'c'

j=lettere[0]; mette 'a' in j

lettere[1]= 'k'; scrive 'k' nella seconda componente di lettere[]

Matrici

Una **matrice** è un vettore multidimensionale, in cui gli elementi memorizzati sono riferiti attraverso delle n-uple di indici.

Ci limiteremo ad usare matrici di dimensione 2.

Nel caso di dimensione 2, una matrice può essere interpretata come una griglia di valori (vettore di vettori). Ogni elemento di una matrice di dimensione 2 è indicizzato mediante una coppia di indici (**numero di riga, numero di colonna**).

	0	1	2
0	5	1	0
1	1	3	7
2	10	9	8

Matrici

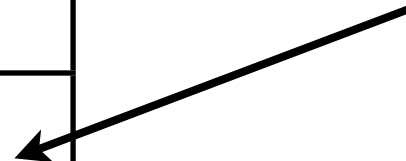
Una **matrice** è un vettore multidimensionale, in cui gli elementi memorizzati sono riferiti attraverso delle n-uple di indici.

Ci limiteremo ad usare matrici di dimensione 2.

Nel caso di dimensione 2, una matrice può essere interpretata come una griglia di valori (vettore di vettori). Ogni elemento di una matrice di dimensione 2 è indicizzato mediante una coppia di indici (numero di riga, numero di colonna).

	0	1	2
0	5	1	0
1	1	3	7
2	10	9	8

elemento 7 identificato
dalla coppia di indici (1,2)



Matrici

Una matrice deve essere innanzitutto dichiarata mediante la seguente sintassi:

```
<tipo> [][] <nome_matrice>;
```

```
int [][] x;           //dichiara x matrice  
                      //bidimensionale di interi
```

```
String [][] parole; // dichiara una matrice  
                   // di stringhe di nome  
                   // parole
```

Creazione di una matrice

Il tipo matrice è un tipo riferimento. Un riferimento ad una matrice si crea mediante l'operatore `new`. Quando si crea una matrice è necessario specificarne le dimensioni (i.e. numero di righe e numero di colonne)..

```
new <tipo> [<dimensione1>][<dimensione2>;
```

```
x = new int [10][3]; //alloca una matrice di  
                    //10 righe e 3 colonne e  
                    //memorizza il riferimento  
                    //in x.
```

```
parole = new String[2][2]; //parole può  
                            //contenere una da  
                            //tabella di stringhe  
                            //di 2 righe e 2  
                            //colonne.
```

Creazione di una matrice

Un altro modo per creare un riferimento a una matrice consiste nel dare esplicitamente l'elenco dei valori da memorizzare (**linearizzato per righe**) in fase di dichiarazione.

```
int[][] x={{14,12},{1,24},{7,1},{2,9}};
```

```
String[][] frase={{\"ab\",\"cd\"},{\"ef\",\"gh\"}};
```

Accesso e scrittura di una matrice

Le componenti di una matrice sono indicizzate mediante coppie di numeri interi. Se la matrice ha n righe e m colonne, essa conterrà $n*m$ celle. L'indice sulle righe può variare da 0 a $n-1$, mentre quello sulle colonne da 0 a $m-1$.

La componente di riga i e colonna j di una matrice può essere reperita o scritta usando la seguente sintassi:

```
<variabile di tipo matrice>[i][j]
```

```
char[][] l={{'a','b'},{'c','d'}};  
char j;
```

```
System.out.println(l[1][0]);  
j=lettere[0][0];  
lettere[1][1]= 'k';
```

Accesso e scrittura di una matrice

Le componenti di una matrice sono indicizzate mediante coppie di numeri interi. Se la matrice ha n righe e m colonne, essa conterrà $n*m$ celle. L'indice sulle righe può variare da 0 a $n-1$, mentre quello sulle colonne da 0 a $m-1$.

La componente di riga i e colonna j di una matrice può essere reperita o scritta usando la seguente sintassi:

```
<variabile di tipo matrice>[i][j] stampa 'c'
```

```
char[][] l={{'a','b'},{'c','d'}};
```

```
char j;
```

```
System.out.println(l[1][0]); mette 'a' in j
```

```
j=lettere[0][0];
```

```
lettere[1][1]='k'; scrive 'k' nella seconda componente della  
seconda riga e seconda colonna
```

La classe Random

la classe `Random` è una classe predefinita che permette di generare numeri pseudocasuali. È contenuta nel package `java.util` e pertanto per poterla utilizzare è necessario importare tale package con l'istruzione

```
import java.util.Random;
```

che deve essere posta all'inizio della definizione della classe che userà la classe `Random`.

Per maggiori informazioni...

<http://java.sun.com/j2se/1.4.2/docs/api/java/util/Random.html>

La classe Random

Alcuni metodi della classe Random:

- `Random()` costruisce un oggetto della classe Random
- `nextInt()` genera un numero intero pseudo casuale
- `nextInt(int n)` genera un numero casuale compreso tra 0 e n-1
- `nextDouble()` genera un numero di tipo double compreso tra 0.0 e 1.0.

Uso della classe Random

Tipicamente per poter utilizzare i metodi della classe si seguono i seguenti passi:

- 1) si dichiara una variabile R della classe Random
- 2) si istanzia un oggetto della classe Random e si memorizza nella variabile R
- 3) Si invoca il metodo desiderato attraverso R
- 4) si memorizza il numero casuale generato in una variabile del tipo opportuno.

Esempio

```
{  
Random R; //Dichiarazione della variabile  
          //di tipo Random  
int X;  
  
R = new Random(); //Generazione di un  
                  //oggetto di tipo Random  
                  //e suo assegnamento alla  
                  //variabile R  
  
X = R.nextInt(15); //invocazione del metodo  
                  //nextInt, che genera un  
                  //numero pseudocasuale  
                  //compreso fra 0 e 14 e  
                  //assegnazione del numero  
                  //generato alla variabile  
                  //di tipo intero X  
}
```

La classe String

- La classe **String** permette di rappresentare sequenze di caratteri (i.e. stringhe).
- Tale classe è dotata di numerosi metodi per la manipolazione delle stringhe.
- Maggiori informazioni al link

<http://java.sun.com/j2se/1.4.2/docs/api/java/lang/String.html>

Alcuni metodi della classe String

- Costruttori

String() - costruisce una stringa vuota

Es. **String s=new String();**

String(String s) - costruisce una stringa e assegna ad essa il contenuto della stringa s.

Es. **String st=new String("pippo");**

Stesso effetto con: **String st="pippo";**

length() - calcola la lunghezza di una stringa

```
es. String s=new String("pippo");  
    int l=s.length(); // l=5
```

charAt(int i) - seleziona l'i-esimo carattere della stringa.

```
es. String s=new String("pippo");  
    char c=s.charAt(4); // c='o'
```

equals(Object s) - confronta la stringa con l'oggetto s. Se sono uguali torna true altrimenti false

```
es. String s1=new String("pippo");  
    String s2=new String("pappa");  
    boolean uguali=s1.equals(s2); // uguali=false
```

toUpperCase() - Converte i caratteri di una stringa in caratteri maiuscoli.

```
es. String s=new String("PipPo");  
    String s1= new String(s.toUpperCase());  
    // s1="PIPPO"
```

toLowerCase() - Converte i caratteri di una stringa in caratteri minuscoli.

```
es. String s=new String("PipPo");  
    String s1= new String(s.toLowerCase());  
    // s1="pippo"
```

substring(int i, int j) - seleziona la sottostringa dalla posizione i (inclusa) alla posizione j (esclusa).

```
es. String s=new String("Cappotto");  
    String s2= new String(s.substring(4,s.length()));  
    // s2="otto"
```

concat(String s) - Concatena la stringa con la stringa s.

```
es. String s1=new String("No");  
    String s2= new String("Where");  
    s1=s1.concat(s2); //s1="NoWhere"
```

Nota: per concatenare due stringhe potete sempre usare l'operatore +. Es. "No"+"Where" -> "NoWhere"

Ereditarietà

È possibile derivare una classe B (chiamata **sottoclasse**) a partire da una classe A (chiamata **superclasse**). La sottoclasse B eredita tutti i metodi e gli attributi della superclasse. Inoltre in B possiamo definire nuovi metodi e nuovi attributi non presenti in A. Tale meccanismo è chiamato **ereditarietà**.

SINTASSI:

```
class <nome_sottoclasse> extends <nome_superclasse>
{
    <corpo del sottoclasse>
}
```

Alcune proprietà

È possibile definire il metodo costruttore della classe derivata a partire dal costruttore della superclasse utilizzando il costrutto

```
super(<lista valori>);
```

dove `lista valori` contiene i valori da passare come parametri al costruttore della superclasse.

È possibile definire nella sottoclasse metodi che hanno lo stesso nome di alcuni metodi della superclasse, ma parametri e comportamento distinti (**overriding**).

Esempio

```
class Persona
{
    String nome;
    String cognome;
    int eta;

    Persona(String n, String c, int e){
        nome=n;
        cognome=c;
        eta=e;
    }
    void stampaDati()
    {
        System.out.println("Salve, sono una persona e mi chiamo "+nome);
        System.out.println("Ho "+eta+" anni");
    }

    public void modificaNome(String n)
    {
        nome=n;
    }

    public String ottieniNome()
    {
        return "Mr."+nome;
    }
}
```

Esempio

```
public class Studente extends Persona
{
    // sottoclasse derivata dalla classe Persona

    int matricola;

    // costruttore della sottoclasse Studente
    // si richiama il costruttore della sopraclasse
    // Persona mediante l'istruzione super
    public Studente(String n, String c, int e, int m)
    {
        super(n,c,e);
        matricola=m;
    }

    public void stampaDati()
    {
        System.out.println("Salve, sono uno studente e mi chiamo "+
            nome+" "+cognome);
        System.out.println("Ho "+eta+" anni"+ " e il mio numero di
            matricola e' "+matricola);
    }

    public void modificaMatricola(int matr)
    {
        matricola=matr;
    }
}
```

Modificatori di accesso

I metodi e gli attributi di una classe possono essere dotati dei seguenti modificatori di accesso che ne limitano la visibilità.

1. **private**: specifica che l'attributo, o il metodo, che segue può essere acceduto solo dall'interno della classe;
2. **protected**: consente l'accesso anche alle sottoclassi e alle classi nello stesso package;
3. **public**: consente l'accesso a chiunque. In assenza di modificatori d'accesso, si applica il criterio public.

Es: `private void stampaDati()`

Interfacce

Un'**interfaccia** permette di definire una serie di prototipi di metodi che saranno successivamente implementati da una o più classi.

Un'interfaccia si definisce nel seguente modo:

```
interface <nome_interfaccia>
{
    <corpo dell'interfaccia>
}
```

dove nel corpo dell'interfaccia si definisce una lista di signature di metodi (i.e. intestazioni di metodi).

Esempio

```
interface Misura
{
    double Area();
    double Perimetro();
}
```

Interfacce

Una classe può implementare una o più interfacce usando la seguente sintassi:

```
class <nome_classe> implements <nome_interfaccia1>, <nome_interfaccia2>
{
    <corpo della classe>
}
```

```
Class Rettangolo implements Figura
{
    double base; double altezza;
    double xcoord; double ycoord;
    Rettangolo(x,y,a,b) {
        base=b;altezza=a;xcoord=x;ycoord=y;
    }
    double Area() {
        return base*altezza;
    }
    double Perimetro() {
        return 2*base+2*altezza;
    }
}
```

Input/Output Standard

L' I/O è gestito mediante **flussi** (stream) di dati che permettono di collegare un'applicazione con i dispositivi di I/O.

- **System.out** rappresenta lo standard output (tipicamente il monitor). È dotato dei metodi **print** e **println** che permettono di stampare oggetti a video (rispettivamente senza e con newline a fine stampa).

Esempi:

```
System.out.print("questa è una stringa");  
System.out.println("questa è una stringa");
```

- **System.in** rappresenta lo standard input (tipicamente la tastiera). Permette di gestire un flusso di byte in input (vedi slides successive).

Standard Input

Ingredienti:

E' necessario importare il pacchetto java per la gestione dell'I/O (**java.io.***) nel quale sono definite le seguenti classi:

- 1) **IOException** (classe che permette di gestire le anomalie e gli errori che si possono verificare durante la lettura/scrittura dei dati);
- 2) **InputStreamReader** (permette di convertire un flusso in input di byte in un flusso in input di caratteri);
- 3) **BufferedReader** (permette di prelevare/leggere da un flusso di caratteri in input una stringa di caratteri o un singolo carattere attraverso i metodi **readline()** e **read()** rispettivamente.

Input di una stringa

```
import java.io.*;

class EsempioLetturaStringa
{
    String leggiStringa() throws IOException
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        String s;
        System.out.print("Inserisci Stringa: ");
        s= br.readLine();
        return s;
    }
}
```

Input di un carattere

```
import java.io.*;

class EsempioLetturaCarattere
{
    char leggiChar() throws IOException
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        char c;
        System.out.print("Inserisci Carattere: ");
        c= (char)br.read();
        return c;
    }
}
```

Input di int, float, double e boolean

- Per leggere valori dei tipi di dato primitivi int, float, double e boolean si legge una stringa di caratteri e la si converte nel tipo richiesto.
- La conversione da stringa al tipo di dato primitivo richiesto avviene attraverso i metodi:
 - `Integer.parseInt(<stringa da convertire>)`
 - `Double.parseDouble(<stringa da convertire>)`
 - `Float.parseFloat(<stringa da convertire>)`
 - `Boolean.parseBoolean(<stringa da convertire>)`

Input di un intero

```
import java.io.*;

class EsempioLetturaCarattere
{
    int leggiIntero() throws IOException
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        int i;
        System.out.print("Inserisci Intero: ");
        i= Integer.parseInt(br.readLine());
        return i;
    }
}
```

Input di un double

```
import java.io.*;

class EsempioLetturaCarattere
{
    double leggiDouble() throws IOException
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        double d;
        System.out.print("Inserisci valore Double: ");
        d= Double.parseDouble(br.readLine());
        return d;
    }
}
```

Applicazioni Java stand-alone

Ogni applicazione Java stand-alone, deve contenere un metodo principale dal quale parte l'esecuzione dell'applicazione.

Tal metodo si deve obbligatoriamente chiamare Main e la sua intestazione standard è la seguente:

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

`args` rappresenta un'array di stringhe contenente gli argomenti dati da linea di comando

Per creare un'applicazione stand-alone:

1) fase di compilazione: `javac <nomefile>.java`

2) fase di interpretazione: `java <nomefile> <argomenti>`

(nota: nel caso non ci siano argomenti si esegue semplicemente

`java <nomefile>`)

Esempio

```
class EsempioStandAloneconargomenti
{
    // stampa i valori interi compresi tra args[0] e args[1]

    public static void main(String[] args)
    {
        int i;
        for (i=Integer.parseInt(args[0]); i<=Integer.parseInt(args[1]); i++)
            System.out.println(i);
    }
}
```

Applet

- Sono programmi scritti in Java che possono
 - essere scaricati dalla rete
 - essere eseguiti direttamente nei browser

Applet in Java

- Ogni applet è implementato come sottoclasse della classe predefinita `java.applet.Applet`. Ogni applet deve implementare (ridefinire) alcuni metodi ereditati dalla classe `java.applet.Applet`
 - **init()** - invocato al caricamento dell'applet. Deve essere definito solo in caso siano necessarie specifiche operazioni di inizializzazione.
 - **start()** - invocato al momento dell'attivazione dell'applet (momento in cui l'area dell'applet diventa visibile nella finestra del browser).
 - **paint(Graphics g)** - output grafico dell'applet
 - **stop()** - invocato quando l'applet viene sospeso (momento in cui l'area dell'applet non è più visibile nella finestra del browser).
 - **destroy()** - invocato in concomitanza con la chiusura della pagina web contenente l'applet

il metodo paint

- Permette di disegnare degli oggetti grafici nella finestra del browser. Per utilizzare le primitive grafiche di java è necessario importare il pacchetto AWT.
- Prende in input come parametro un oggetto della classe **Graphics** rappresentante l'area grafica dell'applet. `public void paint(Graphics g)`
- **Graphics** è una classe predefinita che permette di disegnare delle figure geometriche.

Un semplice esempio

```
import java.applet.Applet;  
import java.awt.Graphics;  
  
public class HelloWorld extends Applet {  
    public void paint(Graphics g) {  
        g.drawString("Hello world!", 50, 25);  
    }  
}
```

Applet nelle pagine Web

Per inserire un applet in una pagina Web è necessario usare il tag HTML **Applet** nel seguente modo:

```
<APPLET  
  code="<nomeApplet.class>"  
  codebase="<percorso/URL dell'applet>"  
  width="<larghezza dell'area grafica dell'applet>"  
  height="<altezzadell'area grafica dell'applet>"  
>
```

Esempio:

```
<APPLET  
  code="HelloWorld.class"  
  codebase="."  
  width="500"  
  height="500"  
>
```

La classe Graphics

alcuni metodi

Graphics() - è il costruttore della classe e permette di istanziare un nuovo oggetto della classe

drawLine(int x1, int y1, int x2, int y2) - disegna un segmento di retta i cui estremi sono i punti (x1,y1) e (x2,y2)

drawOval(int x, int y, int width, int height) - disegna un ovale con angolo superiore sinistro di coordinate (x,y), largo width e alto height.

drawRect(int x, int y, int width, int height) - disegna un rettangolo il cui vertice in alto a sinistra ha coordinate (x,y), largo width e alto height.

La classe Graphics

`drawArc(int x, int y, int width, int height, int startangle, int arcangle)` - disegna un arco che origina in (x,y), largo width, alto height, il cui angolo iniziale e' startangle e l'angolo finale e' arcangle.

`drawString(String s,int x, int y)` - stampa s alle coordinate (x,y).

`setColor(Color c)` - setta il colore al colore Color.c

Alcuni valori possibili per c: `black`, `blue`, `cyan`, `darkGray`, `orange`, `pink`, `red`, `yellow`, `magenta`, `white`.

Es. `g.setColor(Color.green)`, dove g e' di tipo Graphics

`fillRect(int x, int y, int width, int height)` - disegna un rettangolo e lo riempie con colore selezionato

La classe Graphics

alcuni metodi

`fillOval(int x, int y, int width, int height)` - disegna un ovale e lo riempie con il colore selezionato.

`fillArc(int x, int y, int width, int height, int startangle, int arcangle)` - disegna un arco e lo riempie fino alla corda con il colore selezionato.

`drawPolygon(int[] x, int[] y, int n)` - disegna un poligono di n vertici. Le coordinate dei vertici sono date mediante gli array di interi `x[]` e `y[]`.

`fillPolygon(int[] x, int[] y, int n)` - disegna un poligono di n vertici e lo riempie con il colore selezionato.

```
import java.applet.*;
import java.awt.*;

public class grafica extends Applet
{
    public void init()
    {
    }
    public void start()
    {
    }
    public void stop()
    {
    }
    public void paint(Graphics g)
    {
        g.setColor(Color.white);
        g.fillRect(0, 0, 200, 100);
        g.setColor(Color.black);
        g.drawString("Questa
o e' un", 20, 20);
        g.setColor(Color.blue);
        g.drawString("applet!", 20, 40);
    }
    public void destroy()
    {
    }
}
```


Gestione eventi: il mouse

- Un evento indica che una determinata azione è occorsa in un dato componente (es. in un applet).
- La classe **MouseEvent** gestisce gli eventi legati al mouse (pressione di un tasto, movimenti del mouse, rilascio di un tasto, etc.)

Alcuni metodi di MouseEvent

- `int getX()` - ritorna la coordinata X del mouse
- `int getY()` - ritorna la coordinata Y del mouse
- `int getButton()` - ritorna quale bottone del mouse e' stato premuto
(NOBUTTON,BUTTON1,BUTTON2,BUTTON3)

La classe MouseListener

E' una classe astratta utilizzata per ricevere gli eventi del mouse. Per utilizzarla è necessario importare `java.awt.event.*`

Per poter gestire tali eventi è necessario definire una sottoclasse di `MouseListener` e ridefinire alcuni dei suoi metodi. Tale sottoclasse viene chiamata `Ascoltatore`.

Alcuni metodi di MouseListener

void mouseClicked(MouseEvent e) - è un metodo che si invoca quando clicchiamo un bottone del mouse

void mousePressed(MouseEvent e) - è un metodo che si invoca quando manteniamo premuto un bottone del mouse

void mouseReleased(MouseEvent e) - è un metodo che si invoca quando si rilascia il bottone del mouse

La classe MouseMotionAdapter

Permette di ricevere e gestire i movimenti del mouse (movimento senza pressione di bottoni, trascinamento)

Per poter gestire tali eventi è necessario definire una sottoclasse di **MouseMotionAdapter** e ridefinire alcuni dei suoi metodi. Tale sottoclasse viene chiamata **Ascoltatore**.

Alcuni metodi di MouseAdapter

void mouseDragged(MouseEvent e) - è un metodo che si invoca quando muoviamo il mouse mantenendo premuto un bottone.

void mouseMoved(MouseEvent e) - è un metodo che si invoca quando muoviamo il mouse senza premere bottoni.

Gestione degli eventi in un applet

- Per poter gestire gli eventi legati al mouse in java è necessario:
 - definire una classe ascoltatore per gli eventi che si vogliono gestire (può essere una sottoclasse di **MouseListener** oppure una sottoclasse di **MouseMotionListener**)
 - collegare l'ascoltatore all'applet attraverso il metodo **addMouseListener** oppure **addMouseMotionListener** a seconda dell'ascoltatore implementato

Esempio

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class EsempioMouse extends Applet
{
    Color c;
    int base;
    int altezza;
    int x;
    int y;
    // Ascoltatore per la gestione dei click
    public class AscoltatoreClick extends MouseAdapter{
        public void mousePressed(MouseEvent e)
        {
            base=200;
            altezza=100;
            c=Color.orange;
            repaint();
        }
        public void mouseReleased(MouseEvent e)
        {
            base=100;
            altezza=200;
            c=Color.green;
            repaint();
        }
    }
}
```

Continua...

Esempio

```
// Ascoltatore per la gestione del trascinamento
public class AscoltatoreDrag extends MouseMotionAdapter{
    public void mouseDragged(MouseEvent e){
        x=e.getX();
        y=e.getY();
    }
}

public void init()
{
    c=Color.green;
    base=100;
    altezza=200;
    x=0;
    y=0;
    //collega l'ascoltatore per la pressione dei tasti del mouse all'applet
    addMouseListener(new AscoltatoreClick());
    //collega la classe ascoltatore per la gestione dei movimenti del mouse all'applet
    addMouseMotionListener(new AscoltatoreDrag());
}

public void paint(Graphics g){
    g.setColor(c);
    g.fillRect(x,y,base,altezza);
}
}
```

Metodo `getGraphics()`

È un metodo della classe `Applet` utilizzato per poter disegnare sul contenitore grafico associato senza passare per il metodo `paint(Graphics g)` (o `repaint()`).

Ritorna l'oggetto della classe `Graphics` associato all'applet.

A differenza di `repaint()`, una volta invocato non reinizializza il contenitore grafico associato.

Per simulare il comportamento di `getGraphics()` con `repaint()` ed ottenere un metodo di repaint incrementale, si deve ridefinire il metodo `update(graphics g)` nell'applet nel seguente modo

```
public void update(Graphics g)
{
    paint(g);
}
```

Visualizzare immagini in un applet

Per visualizzare un immagine (.gif, .jpg, .png,....) è necessario:

1) Caricare l'immagine in un oggetto di tipo **Image** con il metodo:

```
Image getImage(String <percorso immagine>,  
               String <nome immagine>);
```

Es:

```
Image picture;  
picture = getImage(getDocumentBase(), "pippo.gif");
```

getDocumentBase() ritorna il percorso del documento.

Visualizzare immagini in un applet

2) Visualizzare l'immagine caricata in un oggetto di tipo **Image** con il metodo della classe Graphics:

```
drawImage(Image Pic, int Xcoord, int Ycoord,  
          this);
```

Es:

```
Image picture;  
picture = getImage(getDocumentBase(), "pippo.gif");  
  
g.drawImage(picture, 0, 0, this);
```

dove g è un oggetto della classe Graphics.

Suoni e applet

Per poter riprodurre un suono le formato .au è necessario:

1) Caricare il suono desiderato in un oggetto di tipo **AudioClip** con il metodo:

```
AudioClip getAudioClip(String <percorso suono>,  
                        String <nome suono>);
```

Es:

```
AudioClip sound;  
sound = getAudioClip(getDocumentBase(), "sirena.au");
```

`getDocumentBase()` ritorna il percorso del documento.

Riproduzione di suoni in un applet

2) Controllare la riproduzione del suono caricato in un oggetto di tipo **AudioClip** mediante i metodi della classe AudioClip:

```
void play();  
void stop();  
void loop();
```

Es:

```
AudioClip sound;  
sound = getAudioClip(getDocumentBase(), "sirena.au");  
sound.play();
```