TIPI DI DATO E OPERATORI

Tipi primitivi

- 8 tipi semplici per numeri, testo, etc.

Nome	Descrizione	Esempio
`int`	interi	`42, -3, 0, 926394`
`double`	numeri reali	`3.1, -0.25, 9.4e3`
`char`	carattere singolo	`'a', 'X', '?', '\n'`
`boolean`	valori logici	`true, false`

  - Quale è la differenza fra interi e numeri reali?

Tipi Primitivi

Tipo	Dimensione	Valori
boolean	1-bit	true/false
char	16-bit Unicode	‘\n’
byte	signed 8-bit	-128…+127
short	signed 16-bit	-32768 … +32767
int	signed 32-bit	-2^{31} … + 2^{31}-1
long	signed 64-bit	-2^{63} … + 2^{63}-1
float	32-bit (IEEE-754)	3.4e+38 (7 decim.)
double	64-bit (IEEE-754)	1.7e+308 (15 decim)

Esplicitare il tipo

- il tipo va esplicitato:
  - 10 è un int (*default*)
  - 10 è anche uno short
  - 10 è anche un byte
  - 10L è un long
  - 10F è un float
  - 10D è un double
  - 1.0F è un float
  - 1.0 è un double (*default*)

Conversioni di tipo

- Conversioni automatiche se non c'è perdita di precisione:
  - da numeri interi a numeri in virgola mobile
  - fra interi di cardinalità minore a interi di cardinalità maggiore (int >> long)
  - da float a double
- Negli altri casi devo usare un "*casting*" esplicito
  - double >> int
  - (int)(5.5+0.4) vale 5

Divisione intera /

- Il risultato della divisione è intero
  - `14 / 4` è `3`, non è `3.5`

```
   3          4          52
4 ) 14    10 ) 45    27 ) 1425
   12         40         135
    2          5          75
                          54
                          21
```

- Altri esempi:
  - `32 / 5` è `6`
  - `84 / 10` è `8`
  - `156 / 100` è `1`
  - La divisione per 0 genera errore al Runtime

Operatore %

- % calcola il resto della divisione fra interi.
  - `14 % 4` is `2`
  - `218 % 5` is `3`

```
   3
4 ) 14
   12
   2
```

```
   43
5 ) 218
   20
   18
   15
   3
```

Quali sono i risultati?

```
45 % 6
2 % 2
8 % 20
11 % 0
```

- Altri usi di % :
  - Ultima cifra di un numero: `230857 % 10` is `7`
  - Ultime 4 cifre: `658236489 % 10000` is `6489`
  - Pari o dispari?: `7 % 2` is `1`, `42 % 2` is `0`

Precendenza operatori

- L'ordine con cui gli operatori sono valutati.
  - Normalmente da sinistra a destra.
    `1 - 2 - 3` è `(1 - 2) - 3` he vale `-4`
  - Ma `*/%` hanno un livello di precedenza maggiore rispetto a `+-`
    `1 + 3 * 4` è `13`

    `6 + 8 / 2 * 3`
    `6 + 4 * 3`
    `6 + 12` è `18`
  - Le parentesi forzano l'ordine di valutazione
    `(1 + 3) * 4` è `16`
  - Gli spazi non contano nulla
    `1+3 * 4-2` è `11`

Incremento e decremento

- *Shortcut per incrementare o decrementare le variabili di 1*

Versione corta

```
a++;
a--;
```

Versione estesa

```
a= a + 1;
a= a - 1;
```

```
int x = 2;
x++;
```

```
// x = x + 1;
// x vale 3
```

```
double gpa = 2.5;
gpa--;
```

```
// gpa = gpa - 1;
// gpa vale 1.5
```

Modifica ed assegna

- *Shortcut per modificare il valore di una variabile*

Versione breve	Versione estesa
`a += v;`	`a = a + v;`
`a -= v;`	`a = a - v;`
`a *= v;`	`a = a * v;`
`a /= v;`	`a = a / v;`
`a %= v;`	`a = a % v;`

```
x += 3;                    // x = x + 3;
gpa -= 0.5;                // gpa = gpa - 0.5;
number *= 2;               // number = number * 2;
```

COSTRUTTI BASE

Istruzioni if, else

```
if (espressione-booleana) istruzione;
```

```
if (numeroLati == 3)
    System.out.println("Questo è un triangolo");
```

```
if (espressione-booleana) istruzione1;
else istruzione2;
```

```
if (numeroLati == 3)
    System.out.println("Questo è un triangolo");
else
    System.out.println("Questo non è un triangolo");
```

Istruzioni if, else

```
if (espressione-booleana) {
    istruzione_1;
    ..........;
    istruzione_k;
} else if (espressione-booleana){
    istruzione_k+1;
    ..............;
    istruzione_j;
} else if (espressione-booleana){
    istruzione_j+1;
    ..............;
    istruzione_h;
} else {
    istruzione_h+1;
    ..............;
    istruzione_n;
}
```

Operatore Ternario

- Può sostituire il costrutto if else

```
variabile = (espr-booleana) ? espr1 : espr2;
```

- true assegna expr1
- false assegna expr2

- Esempio

```
String query = "select * from table " +
    (condition != null ? "where " + condition : "");
```

Istruzione while

```
[inizializzazione;]
while (espr. booleana) {
    corpo;
    [aggiornamento iterazione;]
}
```

```
public class WhileDemo {
    public static void main(String args[]) {
        int i = 1;
        while (i <= 10) {
            System.out.println(i);
            i++;
        }
    }
}
```

Istruzioni do while

```
[inizializzazione;]
do {
    corpo;
    [aggiornamento iterazione;]
} while (espr. booleana);
```

```
public class DoWhile {
  public static void main(String args[]) {
    int i = 10;
    do {
      System.out.println(i);
    } while(i < 10);
  }
}
```

Istruzione for

```
for (inizializzazione; espr. booleana; aggiornamento)
{
    istruzione_1;
    ..........;
    istruzione_n;
}
```

```
public class ForDemo
{
    public static void main(String args[])
    {
        for (int n = 10; n > 0; n--)
        {
            System.out.println(n);
        }
    }
}
```

Istruzioni switch case

```
switch (variabile di test) {
    case valore_1:
        istruzione_1;
    break;
    case valore_2: {
        istruzione_2;
        ..........;
        istruzione_k;
    }
    break;
    case valore_3:
    case valore_4: {  //blocchi di codice opzionale
        istruzione_k+1;
        ..........;
        istruzione_j;
    }
    break;
    [default: {    //clausola default opzionale
        istruzione_j+1;
        ..........;
        istruzione_n;
    }]
}
```

Istruzioni break e continue

```
int i = 0;
while (true) //ciclo infinito
{
    if (i > 10)
        break;
    System.out.println(i);
    i++;
}
```

```
int i = 0;
do
{
    i++;
    if (i == 5)
        continue;
    System.out.println(i);
}
while(i <= 10);
```

La classe `Math`

Method name	Description
`Math.abs(`*value*`)`	absolute value
`Math.round(`*value*`)`	nearest whole number
`Math.ceil(`*value*`)`	rounds up
`Math.floor(`*value*`)`	rounds down
`Math.log10(`*value*`)`	logarithm, base 10
`Math.max(`*value1*`, `*value2*`)`	larger of two values
`Math.min(`*value1*`, `*value2*`)`	smaller of two values
`Math.pow(`*base*`, `*exp*`)`	*base* to the *exp* power
`Math.sqrt(`*value*`)`	square root

`Math.sin(`*value*`)`
`Math.cos(`*value*`)`
`Math.tan(`*value*`)`
	sine/cosine/tangent of an angle in radians

`Math.toDegrees(`*value*`)` `Math.toRadians(`*value*`)`	convert degrees to radians and back
`Math.random()`	random `double` between 0 and 1

Constant	Description
`Math.E`	2.7182818...
`Math.PI`	3.1415926...

Strings

- Un oggetto che memorizza una sequenza di caratteri

```
String name = "text";
String name = expression;
```

```
String name = "P. Diddy";
```

index	0	1	2	3	4	5	6	7
char	P	.		D	i	d	d	y

  - Il primo indice è lo 0
  - L'ultimo carattere è nella posizione lunghezza stringa-1
  - Ogni carattere è un `char`

Concatenare String

- Si usa il + tra due stringhe

```
"hello" + 42       è "hello42"
1 + "abc" + 2      è "1abc2"
"abc" + 1 + 2      è "abc12"
1 + 2 + "abc"      è "3abc"
"abc" + 9 * 3      è "abc27"
"1" + 1            è "11"
4 - 1 + "abc"      è "3abc"
```

- Si usa anche per concatenare una stringa con altri tipi.
  - `System.out.println("Grade: " + (95.1 + 71.9) / 2);`
  - Output: `Grade: 83.5`

Metodi di String

Method name	Description
`indexOf(`**`str`**`)`	index where the start of the given string appears in this string (-1 if it is not there)
`length()`	number of characters in this string

`substring(`**`index1`**`, `**`index2`**`)`
or
`substring(`**`index1`**`)`
	the characters in this string from *index1* (inclusive) to *index2* (exclusive);
	if *index2* omitted, grabs till end of string

`toLowerCase()`	a new string with all lowercase letters
`toUpperCase()`	a new string with all uppercase letters

- Sono chiamati con la notazione col punto

```
String gangsta = "Dr. Dre";
System.out.println(gangsta.length());   // 7
```

Metodi di test

Method	Description
`equals(`**`str`**`)`	whether two strings contain the same characters
`equalsIgnoreCase(`**`str`**`)`	whether two strings contain the same characters, ignoring upper vs. lower case
`startsWith(`**`str`**`)`	whether one contains other's characters at start
`endsWith(`**`str`**`)`	whether one contains other's characters at end
`contains(`**`str`**`)`	whether the given string is found within this one

```
String name = console.next();
if (name.startsWith("Dr.")) {
    System.out.println("Are you single?");
} else if (name.equalsIgnoreCase("LUMBERG")) {
    System.out.println("I need your TPS reports.");
}
```

Metodo equals

- Confronta due String.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Barney")) {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

  - Ritorna un valore boolean

Type `char`

- `char` : A primitive type representing single characters.
  - Each character inside a `String` is stored as a `char` value.
  - Literal `char` values are surrounded with apostrophe (single-quote) marks, such as `'a'` or `'4'` or `'\n'` or `'\''`
  - It is legal to have variables, parameters, returns of type `char`

```
char letter = 'S';
System.out.println(letter);                    // S
```

- `char` values can be concatenated with strings.

```
char initial = 'P';
System.out.println(initial + " Diddy");  // P Diddy
```

char vs. String

- "h" is a String
  'h' is a char(the two behave differently)
- String is an object; it contains methods

```
String s = "h";
s = s.toUpperCase();          // 'H'
int len = s.length();         //  1
char first = s.charAt(0);     // 'H'
```

- char is primitive; you can't call methods on it

```
char c = 'h';
c = c.toUpperCase();    // ERROR: "cannot be dereferenced"
```

  - What is s + 1 ? What is c + 1 ?
  - What is s + s ? What is c + c ?

System.out.print

- Prints without moving to a new line
  - allows you to print partial messages on the same line

```
int highestTemp = 5;
for (int i = -3; i <= highestTemp / 2; i++) {
    System.out.print((i * 1.8 + 32) + "  ");
}
```

- Output:

```
26.6  28.4  30.2  32.0  33.8  35.6
```

System.out.printf

`System.out.printf("`**format string**`",` **parameters**`);`

- A format string contains *placeholders* to insert parameters into it:
  - `%d` an integer
  - `%f` a real number
  - `%s` a string
  - `%8d` an integer, 8 characters wide, right-aligned
  - `%-8d` an integer, 8 characters wide, left-aligned
  - `%.4f` a real number, 4 characters after decimal
  - `%6.2f` a real number, 6 characters wide, 2 after decimal
  - Example:

```
int x = 3, y = 2;
System.out.printf("(%d, %d)\n", x, y);  // (3, 2)
System.out.printf("%4d %4.2f\n", x, y); //    3 2.00
```

Scanner

- `System.out`
  - An object with methods named `println` and `print`
- `System.in`
  - not intended to be used directly
  - We use a second object, from a class `Scanner`, to help us.

- Constructing a `Scanner` object to read console input:

```
Scanner name = new Scanner(System.in);
```

  - Example:

```
Scanner console = new Scanner(System.in);
```

Scanner methods

Method	Description
`nextInt()`	reads a token of user input as an `int`
`nextDouble()`	reads a token of user input as a `double`
`next()`	reads a token of user input as a `String`
`nextLine()`	reads a *line* of user input as a `String`

- Each method waits until the user presses Enter.
  - The value typed is returned.

```
System.out.print("How old are you? ");     // prompt
int age = console.nextInt();
System.out.println("You'll be 40 in " +
        (40 - age) + " years.");
```

  - **prompt**: A message telling the user what input to type.

Testing for valid input

- `Scanner` methods to see what the next token will be:

Method	Description
`hasNext()`	returns `true` if there are any more tokens of input to read *(always true for console input)*
`hasNextInt()`	returns `true` if there is a next token and it can be read as an `int`
`hasNextDouble()`	returns `true` if there is a next token and it can be read as a `double`
`hasNextLine()`	returns `true` if there are any more lines of input to read (always true for console input)

- These methods do not consume input;
  they just give information about the next token.
  - Useful to see what input is coming, and to avoid crashes.

Input tokens

- **token**: A unit of user input, separated by whitespace.
  - A `Scanner` splits a file's contents into tokens.

- If an input file contains the following:

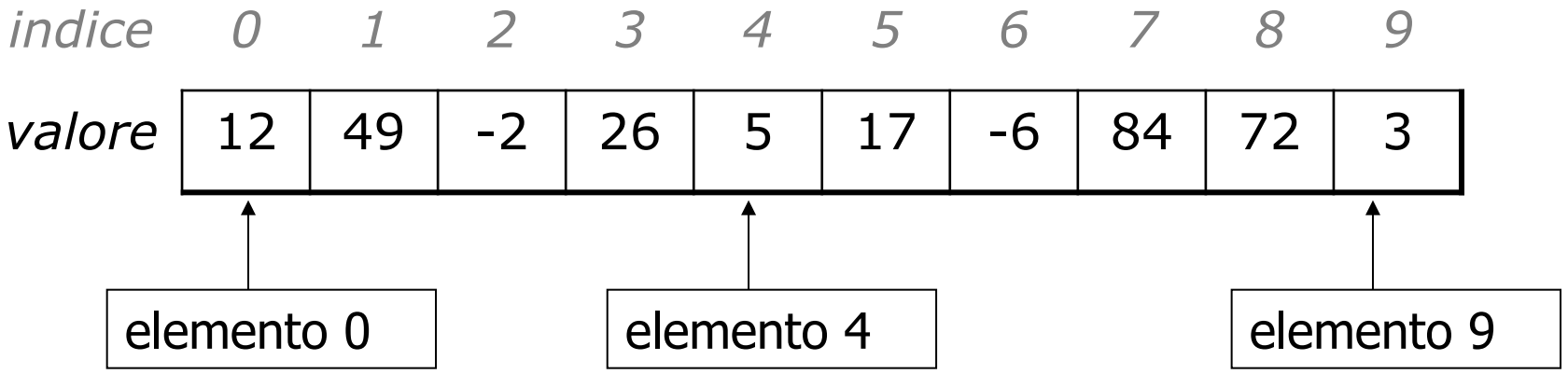
```
23   3.14
  "John Smith"
```

The `Scanner` can interpret the tokens as the following types:

Token	Type(s)
`23`	`int`, `double`, `String`
`3.14`	`double`, `String`
`"John`	`String`
`Smith"`	`String`

Array

- **Oggetti che memorizzano valori di un tipo**

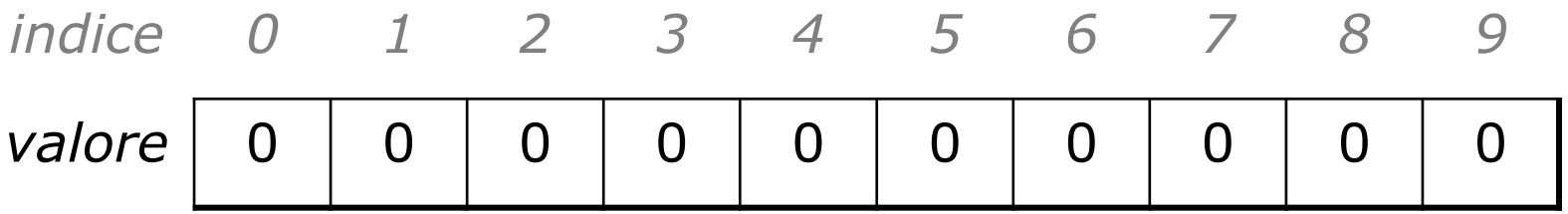


Dichiarazione

type`[]` **name** `=` `new` **type**`[`**length**`];`

- Esempio:

```
int[] numbers = new int[10];
```



Accesso agli elementi

```
name[index]                    // accesso
name[index] = value;           // modifica
```

- Esempio:

```
numbers[0] = 27;
numbers[3] = -6;

System.out.println(numbers[0]);
if (numbers[3] < 0) {
    System.out.println("Element 3 is negative.");
}
```

indice	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
valore	27	0	0	-6	0	0	0	0	0	0

Out-of-bounds

- Indici validi: tra **0** e **length - 1**.
  - Eccezione: `ArrayIndexOutOfBoundsException`.
- Esempio:

```
int[] data = new int[10];
System.out.println(data[0]);        // okay
System.out.println(data[9]);        // okay
System.out.println(data[-1]);       // exception
System.out.println(data[10]);       // exception
```

Il campo `length`

- Il campo `length` memorizza il numero degli elementi nell'array.

```
name.length

for (int i = 0; i < numbers.length; i++) {
    System.out.print(numbers[i] + " ");
}
// output: 0 2 4 6 8 10 12 14
```

  - Non è con le parentesi `.length()`.

Inizializzazione rapida

type`[]` **name** `=` `{`**value**`,` **value**`,` … **value**`};`

– Esempio:

```
int[] numbers = {12, 49, -2, 26, 5, 17, -6};
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	12	49	-2	26	5	17	-6

La classe `Arrays`

- Metodi statici di utilità per array:

Method name	Description
`binarySearch(`**array**`, `**value**`)`	returns the index of the given value in a sorted array (< 0 if not found)
`equals(`**array1**`, `**array2**`)`	returns `true` if the two arrays contain the same elements in the same order
`fill(`**array**`, `**value**`)`	sets every element in the array to have the given value
`sort(`**array**`)`	arranges the elements in the array into ascending order
`toString(`**array**`)`	returns a string representing the array, such as `"[10, 30, 17]"`

CLASSI E OGGETTI

Classi ed Oggetti

- Una classe è una astrazione indicante un insieme di oggetti che condividono le stesse funzionalità

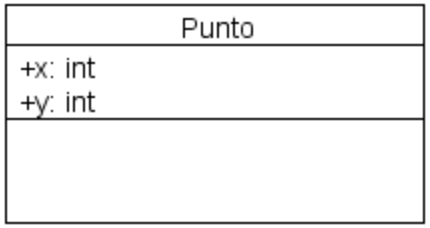
- Un oggetto è una istanza (fisica) di una classe

La classe Punto

- Astrazione del punto cartesiano bidimensionale

```
public class Punto
{
    public int x;
    public int y;
}
```

- Possiamo compilarlo
  - javac Punto.java
- Non possiamo eseguirlo
  - java Punto
- Abbiamo definito il “template” ma non l'oggetto
  - La sua realizzazione fisica nel PC



Gli oggetti Punto

```
1     public class Principale
2     {
3         public static void main(String args[])
4         {
5             Punto punto1;
6             punto1 = new Punto();
7             punto1.x = 2;
8             punto1.y = 6;
9             Punto punto2 = new Punto();
10            punto2.x = 0;
11            punto2.y = 1;
12            System.out.println(punto1.x);
13            System.out.println(punto1.y);
14            System.out.println(punto2.x);
15            System.out.println(punto2.y);
16        }
17    }
```

La main per poter eseguire il codice

Creo un oggetto

Creo un altro oggetto

Osservazioni

- Classe Punto
  - Definisce la struttura dati
  - La usiamo in compilazione
  - Sono gli oggetti che hanno un ruolo attivo
- A rigore
  - Le classi non dovrebbero possedere membri
    - Nota: sono gli oggetti che possiedono x e y
  - Infatti per accedere alla locazione di memoria
    - nomeOggetto.nomeVariabile
    - La variabile appartiene a punto1 e non a Punto

Osservazioni

- Prima Eccezione: La classe Principale
  - Esegue del codice nella classe
  - Non su un oggetto creato dalla classe
- Per storia:
  - Java non usa programmi “chiamanti” come il C++
  - Java avvia i programmi un metodo “statico” della classe: la main
  - Va sempre dichiarata così:

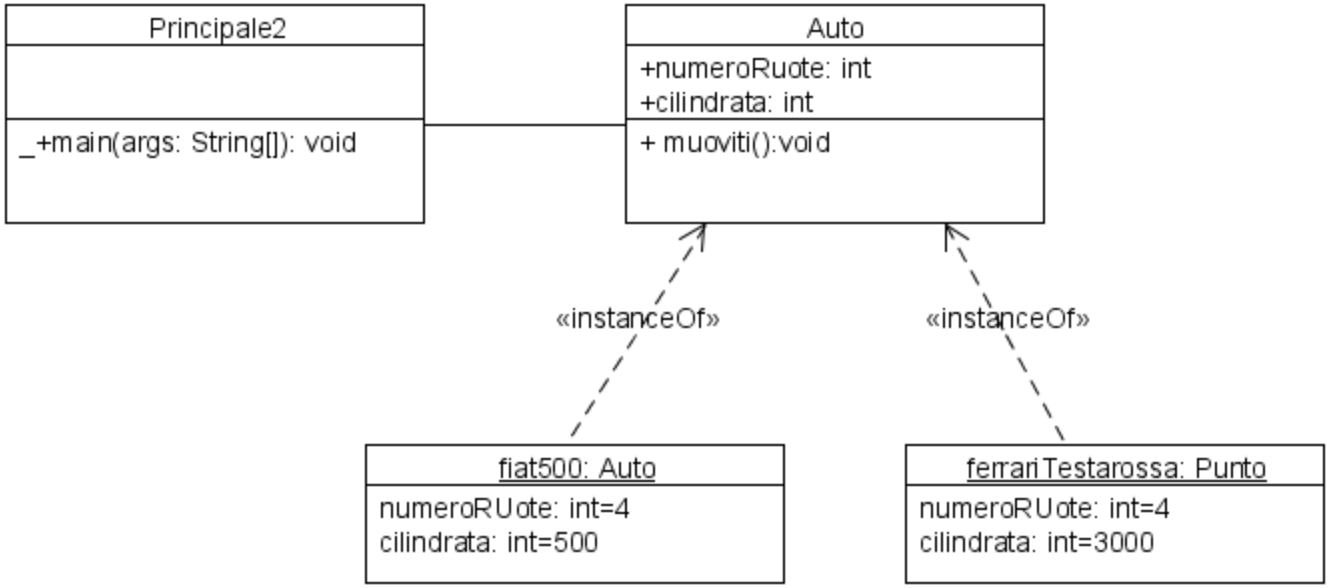
```
public static void main(String args[])
```

Astrarre la realtà

```
public class Auto
{
    public int numeroRuote = 4;
    public int cilindrata; // quanto vale?
    public void muoviti()
    {
        // implementazione del metodo...
    }
}
```

```
public class Principale2
{
    public static void main(String args[])
    {
        Auto fiat500;
        fiat500 = new Auto();
        fiat500.cilindrata = 500;
        fiat500.muoviti();
        Auto ferrariTestarossa = new Auto();
        ferrariTestarossa.cilindrata = 3000;
        ferrariTestarossa.muoviti();
    }
}
```

UML


Principale2
_+main(args: String[]): void
Auto
+numeroRuote: int
+cilindrata: int
+ muoviti():void
«instanceOf»
«instanceOf»
fiat500: Auto
numeroRUote: int=4
cilindrata: int=500
ferrariTestarossa: Punto
numeroRUote: int=4
cilindrata: int=3000

Dichiarare Metodi

```
[modificatori] tipo_di_ritorno nome_del_metodo
([parametri]) {corpo_del_metodo}
```

- Modificatori
  - cambiano le caratteristiche del metodo
  - Esempi: public, static
- Tipo di ritorno
  - Il tipo di dato che il metodo restituisce
  - Può essere un tipo primitivo (int) od un oggetto (String) o nulla (void)
- Nome del metodo
- Parametri
  - Dichiarazione di variabili che possono essere passate al metodo
  - Possono non esserci
  - Se più di uno vanno separati dalla virgola
- Corpo del metodo
  - Le istruzioni da eseguire

Esempio

```
public class Aritmetica
{
    public int somma(int a, int b)
    {
        return (a + b);
    }
}
```

```
1   public class Uno
2   {
3       public static void main(String args[])
4       {
5           Aritmetica oggetto1 = new Aritmetica();
6           int risultato = oggetto1.somma(5, 6);
7       }
8   }
```

Accesso a Metodi e Variabili

- Metodi:
  - nomeOggetto.nomeMetodo()

```
int risultato = oggetto1.somma(5, 6);
```

  - Nota: l'oggetto1 va creato con un new
- Variabile
  - nomeOggetto.nomevariabile

Esempio

```
public class Saluti
{
    public void stampaSaluto()
    {
        System.out.println("Ciao");
    }
}
```

```
1     public class Due
2     {
3         public static void main(String args[])
4         {
5             Saluti oggetto1 = new Saluti();
6             oggetto1.stampaSaluto();
7         }
8     }
```

Dichiarare Variabili

```
[modificatori] tipo_di_dato nome_della_variabile [ =
inizializzazione];
```

- Modificatori:
  - cambiano le caratteristiche della variabile
- Tipo di dato
  - Il tipo di dato della variabile
- Nome della Variabile
- Inizializzazione
  - Il valore a cui viene impostata la memoria di default

```
public class Quadrato
{
    public int altezza, larghezza;
    public final int NUMERO_LATI = 4;
}
```

Variabili

- Variabili d'istanza
  - Dichiarate nella classe ma fuori da un metodo
  - Fanno parte dell'oggetto
    - Vengono allocate con il new dell'oggetto
    - Vengono de-allocate quando l'oggetto non esiste più
- Variabili locali
  - Sono dichiarate all'interno dei metodi
  - Vengono allocate quando si esegue il metodo

Parametri o Argomenti

- Compaiono nella dichiarazione dei metodi

```
public int somma(int x, int y)
{
    int z = x + y;
    return z;
}
```

- Sono creati quando chiamiamo il metodo

```
int risultato = oggetto1.somma(5, 6);
```

```
int a = 5, b = 6;
int risultato = oggetto1.somma(a, b);
```

I Metodi Costruttori

- Metodi speciali con le seguenti proprietà
  - Hanno lo stesso nome della classe
  - Non hanno tipo di ritorno
  - Sono chiamati automaticamente se creo un oggetto della classe

```
public class Punto
{
    public Punto() //metodo costruttore
    {
        System.out.println("costruito un Punto");
    }
    int x;
    int y;
}
```

Creare oggetti

- Dichiarazione ed istanza

```
Punto punto1; //dichiarazione
punto1 = new Punto();  // istanza
```

```
Punto punto1 = new Punto(); //dichiarazione ed istanza
```

- Solo istanza

```
new Punto();
```

  - Non è utilizzabile mi manca il riferimento all'oggetto

Costruttori con parametri

```
public class Punto
{
    public Punto(int a, int b)
    {
        x = a;
        y = b;
    }
    public int x;
    public int y;
}
```

- Non posso più usare

```
Punto punto1 = new Punto();
```

- Creo l'oggetto con

```
Punto punto1 = new Punto(5,6);
```

Stile di Codifica e Commenti

```
public class Classe {
    public int intero;
    public void metodo() {
        intero = 10;
        int unAltroIntero = 11;
    }
}
```

```
// Questo è un commento su una sola riga
```

```
/*
   Questo è un commento
   su più righe
*/
```

```
/**
   Questo commento permette di produrre
   la documentazione del codice
   in formato HTML, nello standard Javadoc
*/
```

Regole per gli identificatori

- Identificatori:
  - nomi di classi, metodi, variabili, package, etc.
- Primo carattere
  - A-Z, a-z, _, $
- Altri caratteri
  - A-Z, a-z, _, $, 0-9
- Non possono essere una keyword java!!
  - ex `new`, `class`, etc
  - Elenco delle keyword sul libro

Riferimenti ad Oggetti

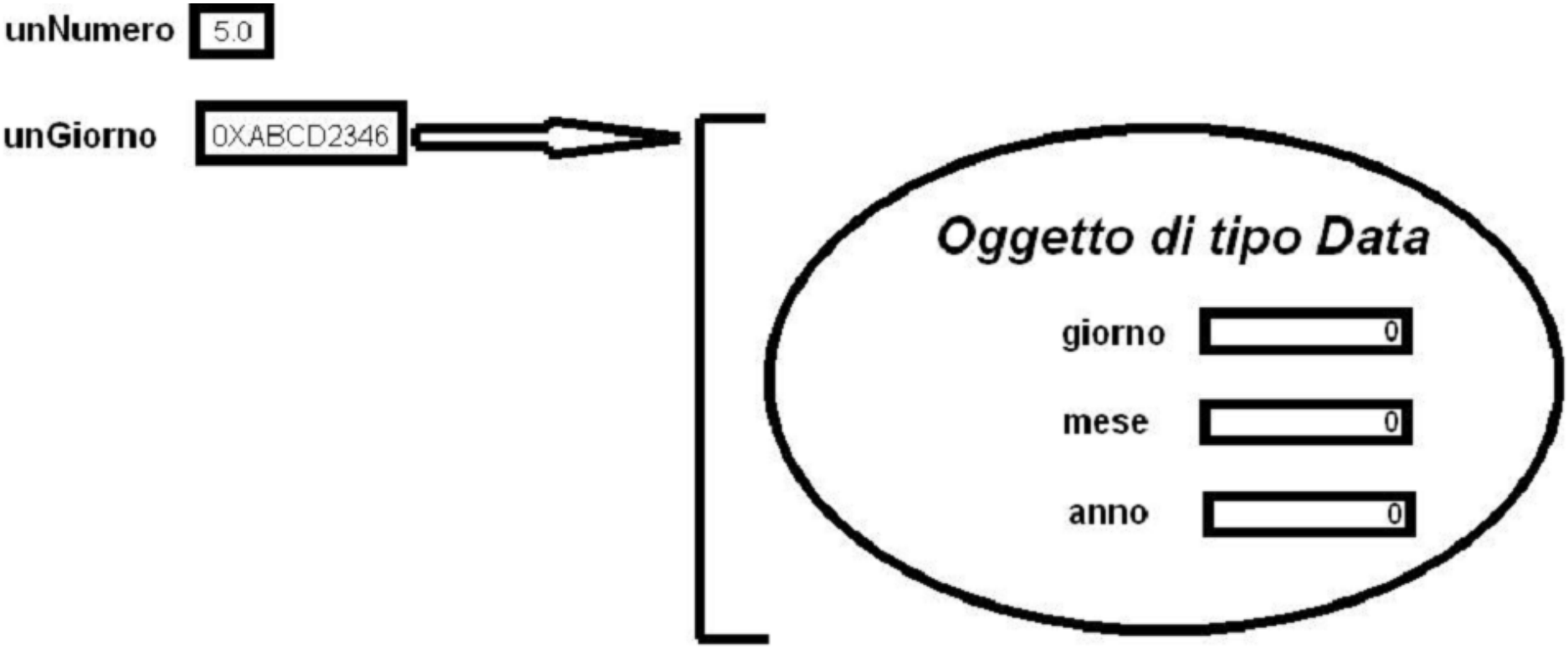
```
NomeClasse nomeOggetto;
```

- Simile alla dichiarazione di tipi primitivi
  - Il nomeOggetto è detto reference (riferimento)
  - Contiene un indirizzo di memoria

```
public class Data {
    public int giorno;
    public int mese;
    public int anno;
}
```

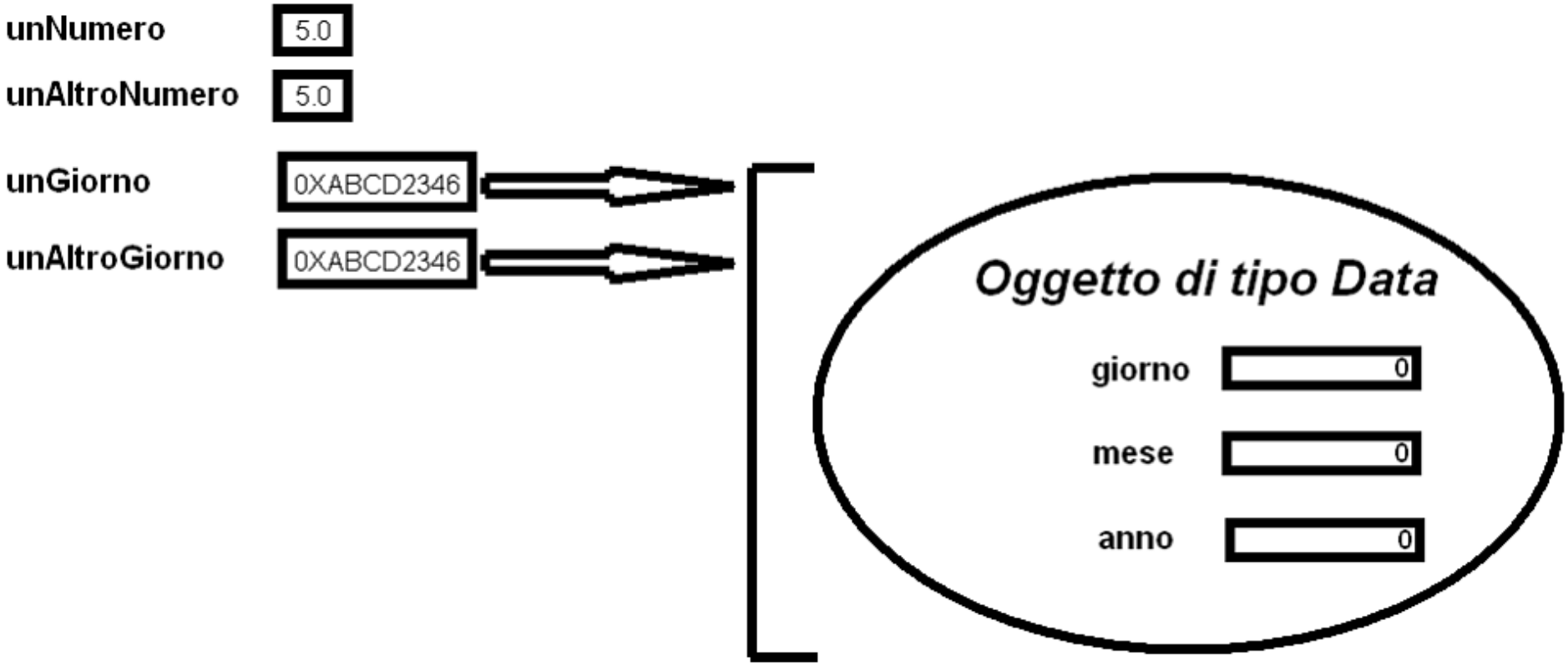
Schema di allocazione in memoria

```
double unNumero = 5.0;
Data unGiorno = new Data();
```



Schema di allocazione in memoria

```
double unNumero = 5.0;
double unAltroNumero = unNumero;
Data unGiorno = new Data();
Data unAltroGiorno = unGiorno;
```



Passaggio dei Parametri

- Il passaggio dei Parametri avviene per valore
  - Al metodo viene passato il valore della variabile

```
public class CiProvo
{
    public void cambiaValore(int valore)
    {
        valore = 1000;
    }
}
```

```
CiProvo ogg = new CiProvo();
int numero = 10;
ogg.cambiaValore(numero);
System.out.println("il valore del numero è " + numero);
```

```
il valore del numero è 10
```

Passaggio dei Parametri

```
public class CiProvoConIReference
{
    public void cambiaReference(Data data)
    {
        data = new Data();
        // Un oggetto appena istanziato
        // ha le variabili con valori nulli
    }
}
```

```
CiProvoConIReference ogg = new CiProvoConIReference();
Data dataDiNascita = new Data();
dataDiNascita.giorno = 26;
dataDiNascita.mese = 1;
dataDiNascita.anno = 1974;
ogg.cambiaReference(dataDiNascita);
System.out.println("Data di nascita = "
  + dataDiNascita.giorno + "-"+ dataDiNascita.mese
  + "-" +dataDiNascita.anno );
```