A

PARAMETRI IMPLICITI: oggetto sul quale un metodo è chiamato

PARAMETRI ESPLICITI: costituiscono l'INPUT per un metodo. Non tutti i metodi hanno parametri espliciti.

ES. System.out.println(greeting)

P. IMPLICITO — System.out
P. ESPLICITO — greeting

VOID se un metodo non ritorna alcun valore

METODO D'ACCESSO: non modifica lo stato del parametro implicito

METODO MODIFICATORE: modifica lo stato del parametro implicito

THIS usato per indicare il PARAMETRO IMPLICITO

FINAL variabile COSTANTE, va indicata con le MAIUSCOLE (es. QUARTER_VALUE), una volta fissato il suo valore esso non può più essere modificato.

STATIC FINAL nel caso: valori costanti siano necessari in parecchi metodi. Vanno dichiarati assieme alle variabili d'istanza.

METODO STATIC non opera su oggetti ma su CLASSI (Es. Math, System.out)

NULL: indica la mancanza di riferimento ad alcun oggetto

CLASSI

CLASSE COESA se tutte le caratteristiche sono correlate al concetto che rappresenta

ACCOPPIAMENTO misura il numero di DIPENDENZE che vi hanno tra le classi

CLASSI IMMUTABILI: Classi senza alcun metodo modificatore

EFFETTO COLLATERALE: qualunque modifica di dati osservabile esternamente

CHIAMATA PER VALORE: i parametri del metodo sono copiati nelle variabili parametro quando il metodo inizia

CHIAMATA PER RIFERIMENTO: i metodi possono modificare i parametri (vengono dei parametri gli indirizzi dei parametri) NO IN JAVA!

PRECONDIZIONE: requisito che deve essere soddisfatto da chi lancia un metodo → in caso lancia ECCEZIONE (facoltativo)

POSTCONDIZIONE: Condizione che è vera dopo che un metodo è terminato.

METODI STATICI: ogni metodo deve appartenere ad una classe, non sono chiamati su oggetti.

VARIABILI STATICHE: ogni var. statica deve appartenere ad una classe e non ad alcun oggetto della classe.
↳ solitamente vengono dichiarate PRIVATE ad eccezione delle COSTANTI (PUBLIC STATIC FINAL)

VARIABILI LOCALI: ambito di visibilità è il blocco o il metodo che la contiene.

VISIBILITÀ SOVRAPPOSTE: una variabile locale può mettere in ombra una variabile d'istanza con lo stesso nome; in tal caso si usa THIS.nomevariabile

)╥╢pGò¢♠ΔêZ╕
↓X@╪j▸Mj`4Ö*}▐ ßE╞hú╜Pt∩Ñ9▐fó♥æ30╓m╟V≤ ¼"v/┬τ╡î¹⁄₄n˜αIÖ╪Σ I♥»■ⓐ

PACKAGE: insieme di classi correlate.

INTERFACCIA: elenco metodi (es. intestazioni) che la classe deve avere, per IMPLEMENTS l'interfaccia.
↳ È simile ad una classe ma con le seguenti differenze:
- Tutti: METODI SONO ASTRATTI (non hanno un'implementazione)
- Tutti: METODI SONO PUBBLICI
- Non ha DATI D'ISTANZA

Es.
```
public interface Measurable {
    double getMeasure();
}
```

POLIMORFISMO: Comportamento che varia in base al tipo effettivo dell'oggetto.
Chiamato anche LATE BINDING (SELEZIONE POSTICIPATA): la selezione viene effettuata IN ESECUZIONE.
Differente dall'OVERLOADING (SOVRACCARICAMENTO - SELEZ. ANTICIP.): selezione svolta dal COMPILATORE.

EREDITARIETÀ: estende le classi aggiungendo metodi e variabili.
Es. Savings EXTEND BankAccount
↳ CLASSE ESTESA (SUPERCLASS)
↳ CLASSE CHE ESTENDE (SUBCLASS)

- EREDITARE METODI
  → SOVRASCRIVERE UN METODO: stessa firma metodo superclasse, nuova implementazione, viene lanciato il metodo che sovrascrive se applicato ad un oggetto della sottoclasse.
  → EREDITARE UN METODO: non do nuova implementazione, posso usare il metodo della superclasse sugli oggetti della sottoclasse.
  → AGGIUNGERE UN METODO: creo un nuovo metodo che non esiste nella superclasse, esso può essere applicato solo ad oggetti della sottoclasse.

- EREDITARE VARIABILI DI ISTANZA — N.B. NON si possono SOVRASCRIVERE VARIABILI
  → EREDITARIETÀ DELLE VARIABILI: tutte le variabili di istanza della superclasse sono ereditate automaticamente
  → AGGIUNGERE UNA VARIABILE: fornire una nuova variabile che non esiste nella superclasse
  → SE CREO UNA VARIABILE CON LO STESSO NOME creo due variabili distinte, una per classe.

SUPER: usato per chiamare il metodo della superclasse (Es. super.deposit(amount);)

SUPER (variabili superclasse): indica una chiamata al costruttore della superclasse.

CONVERSIONE:
- riferimento sottoclasse → riferimento superclasse ✓
- riferimento superclasse → riferimento sottoclasse PERICOLOSO! SI USI INSTANCEOF (instanceof)
- INSTANCE OF: verifica se un oggetto appartiene ad un specifico tipo
  (Es. anObject instanceof BankAccount → rende TRUE)

B

CONTROLLI DI ACCESSO:

- PUBLIC: accesso consentito ai metodi di TUTTE LE CLASSI
- PRIVATE: accesso consentito ai metodi della CLASSE STESSA
- PROTECTED: accesso consentito ai metodi della CLASSE STESSA, SOTTOCLASSI e STESSO PACKAGE
- PACKAGE ACCESS: accesso consentito a TUTTE LE CLASSI NELLO STESSO PACKAGE

LIVELLI DI ACCESSO CONSIGLIATI:

- VARIABILI DI ISTANZA E STATICHE: sempre PRIVATE, eccetto COSTANTI (PUBLIC STATIC FINAL)
- METODI: PUBLIC o PRIVATE
- CLASSI o INTERFACCE: PUBLIC o PACKAGE

OBJECT: è la superclasse UNIVERSALE, tutte le classi usano "extend" estendo Object
↳ contiene metodi toString(), equals(), clone().

(1)

- REPLACE — string. REPLACE ("com", "con le Gra")
- LENGTH () — string. LENGTH () → lunghezza stringa
- TO UPPERCASE() — string. TO UPPERCASE() → tutta la stringa in MAIUSCOL
- To LowerCase() — — — — — — MINUSCOL
- RECTANGLE — new RECTANGLE (x, y, lato1, lato2) — PARAMETRI DI ISTANZIAZIONE
- TRANSLATE — box. TRANSLATE (x, y) ← METODO MODIFICATORE

COSTRUIRE FRAME

1) JFrame frame = new JFrame(); — CREO OGGETTO CLASSE JFRAME
2) frame. setSize (300,400); — IMPOSTO DIMENSIONI
3) frame. setTitle ("An Empty Frame"); — IMPOSTO TITOLO
4) frame. setDefaultCloseOperation (JFrame. EXIT_ON_CLOSE); — IMPOSTO OPERAZ. DI CHIUSURA FRAME
5) frame. setVisible (true); — RENDO IL FRAME VISIBILE

DISEGNARE

1) Definire una classe che estende la classe JComponent.
2) Istruzione le istruzioni dentro il metodo paintComponent

```
public class RectComponent extends JComponent
{
   public void paintComponent (Graphics g) {...}
}
```

→ CONVERSIONE GRAPHICS 2D :

```
public class RectComponent extends JComponent {
    public void paintComponent (Graphics g) {
        Graphics2D g2 = (Graphics2D) g;
        ...
    }
}
```

- DRAW — g2. draw (box) dentro paintComponent
- ADD — frame. add (component) aggiunge componente al frame
- ELLIPSE 2D. DOUBLE — new Ellipse2D. Double (x, y, width, height)
- LINE 2D. DOUBLE → new Line2D. Double (x_1, y_1, x_2, y_2)
  → oppure new Line 2D. Double (punto1, punto2)
- SetColor — g2. setColor (magenta)
- FILL — g2. fill (rectangle) ← riempito con il colore corrente

NUMERI

- MATH. ROUND — Math. round (balance) ← Converte un FLOATING-POINT all' INTERO più vicino
- CONVERSIONE — int dollars = (int) balance ← converte balance a INTERO
- DIVISIONE / — int / int dà DIVISIONE INTERA
  double/int dà DIVISIONE IN DOUBLE
- RESTO (MODULO % — int % int dà il resto della DIVISIONE INTERA
- POTENZA — MATH.POW — Math. pow (x, n) dà x^n
- RADICE QUADRATA — MATH. SQRT — Math. sqrt (x) dà $\sqrt[2]{x}$

CONVERSIONE TRA STRINGHE E NUMERI

- CONVERTIRE AD UN NUMERO `int n = Integer.parseInt(stringa);`
  `double x = Double.parseDouble(x);`
- CONVERTIRE A STRINGA `String str = "" + n;`
  `str = Integer.toString(n);`

- SUBSTRING `oggetto.substring(0, 5)` estrae una sottostringa da una stringa
  (0 = posizione iniziale, 5 = numero di lettere da estrarre)

LEGGERE L'INPUT (da tastiera)

```
Scanner in = new Scanner(System.in);
int quantity = in.nextInt();     legge INTERO
               nextDouble();     legge un DOUBLE
               nextLine();       legge una LINEA (finché l'utente non preme INVIO)
               nextWord()        legge una PAROLA (finché l'utente non preme SPAZIO)
```

BOX DI DIALOGO

- `JOptionPane.showInputDialog(prompt)` — INPUTBOX
- `JOptionPane.showMessageDialog(frame, message)` ← Crea altra roba MESSAGE BOX

IF

```
IF (condition1) {istruzioni;}
ELSE IF (condition2) {istruzioni;}
ELSE {istruzioni}
```

Se le istruzioni da compiere da quando voglio raggruppale in BLOCCO con parentesi GRAFFE, per una riga di istruzione non serve.

CONFRONTO TRA OGGETTI

- `==` verifica l'IDENTITÀ fra oggetti
- `EQUALS` verifica l'IDENTICO CONTENUTO (Es. box1.equals(box3) → TRUE)

CONFRONTO VALORI

N.B. ATTENZIONE AGLI ARROTONDAMENTI DEI FLOATING-POINT, SI USA EPSILON: $(x_1 - x_2) < \varepsilon$

CONFRONTO STRINGHE

N.B. NON USARE `==` MA USARE EQUALS

- `str.compareTo(str2) < 0` str viene prima di str2 nel dizionario
  `== 0` str ha lo stesso contenuto di str2
  `> 0` str viene dopo di str2 nel dizionario

METODI PREDICATIVI NELLA CLASSE SCANNER

`hasNextInt()`, `hasNextDouble()` } restituiscono TRUE o FALSE se il prossimo numero è INT o DOUBLE

LOGICA BOOLEANA

`&&`	AND
`||`	OR
`!`	NOT

CICLI

- **WHILE** `while (condizione) { istruzioni; }` — N.B. INCREMENTARE UN CONTATORE O MODIFICARE QUALCOSA ALL'INTERNO DELLE ISTRUZIONI
  ↳ esegue le istruzioni finché la condizione è vera

(2)

- **DO** esegue il corpo del ciclo almeno una volta; `do { istruzioni; } while (condizione);`
  - ALTERNATIVA UNTIL: `boolean done = false; while (!done) { ... if (valore > 0) done = true; ... }`
- **FOR** `for (inizializzazione; condizione; aggiornamento) { istruzioni; }`
  (Es. `for (i=1; i<10; i++) {..}`)

NUMERI CASUALI

```
Random generatore = new Random();
int n = generatore.nextInt(a);      0 ≤ n < a   (intero)
double x = generatore.nextDouble(); 0 ≤ x < 1   (double)
```

ARRAY — DIMENSIONE FISSA

Es. `double[] data = new double[10];`
↳ LUNGHEZZA ARRAY

`data.length` non è un metodo ma una COSTANTE PUBBLICA !!!

Posso scrivere un array da 0 a (length − 1)

ARRAYLIST — DIMENSIONE PUÒ VARIARE

`ArrayList<TipoOggetti> nome = new ArrayList<TipoOggetti>();`

- ADD(a) `nome.add(cheCosa);` aggiunge in fondo
- ADD(i,a) `nome.add(indice, cheCosa);` aggiunge prima nella posizione i
- SIZE `nome.size();` restituisce il numero di elementi
- SET(i,a) `nome.set(indice, cheCosa);` SOVRASCRIVE in posizione i
- REMOVE(i) `nome.remove(indice);` RIMUOVE l'oggetto in posizione i
- GET(i) `nome.get(indice);` RITORNA ciò che trova in posizione i

NB Meglio: ARRAYLIST non si possono inserire valori di tipo PRIMITIVO, come oggetti vanno le CLASSI INVOLUCRO

CLASSI INVOLUCRO

double (primitivo) ⟶ Double (classe involucro)
boolean (") ⟶ Boolean (")

Da JAVA 5.0 l'IMPACCHETTAMENTO è automatico (non servono CONVERSIONI)

CICLO FOR GENERALIZZATO (FOR EACH)

`for (double e : arrayDiDouble) { istruzioni; }`

↳ può essere utilizzato anche con ARRAYLIST

ARRAY 2-DIMENSIONALI

Bisogna specificare il numero di righe e di colonne:

```
FINAL INT ROWS = 3;
FINAL INT COLUMNS = 3;
STRING [][] board = NEW STRING [ROWS] [COLUMNS];
```

Per accedere: `board [i][j] = "x"`

COPIARE ARRAY

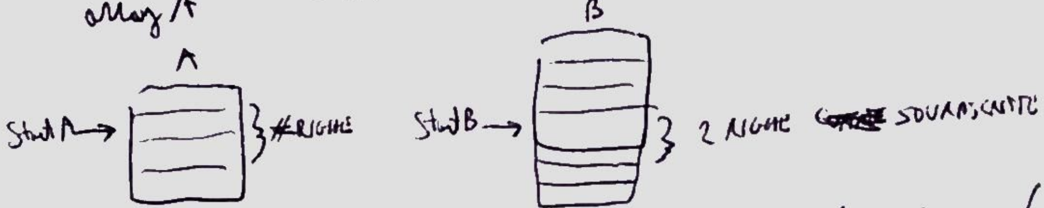
• CLONE `double[] prices = (double[]) data.clone();`

└ VA SPECIFICATO IL TIPO!!! Altrimenti array di oggetti generici

COPIARE DA ARRAY A ARRAY

• SYSTEM.ARRAYCOPY (arrayPartenza, indiceArrayPartenzaSTART, arrayArrivo, indiceArrayArrivoSTART, numero di righe da copiare)

(arrayPartenza = array A; indiceArrayPartenzaSTART = indexStart A; arrayArrivo = array B; indiceArrayArrivoSTART = indexStart B; numero di righe da copiare = quante righe)



↳ posso usare questo metodo per AGGIUNGERE/RIMUOVERE righe dallo stesso Array (N.B non ArrayList!)

ARRAY PARZIALMENTE RIEMPITI

Uso costante per lunghezza array e creo una variabile int per tenere traccia fino a che punto l'array è stato riempito.

EVENTI

– SORGENTE: oggetto che genera l'evento (Es. pulsante)

– ASCOLTATORE: oggetto che contiene un metodo che scatta quando si verifica l'evento.

• interfaccia ActionListener

```
public interface ActionListener {
    void actionPerformed (ActionEvent event); }
```

(parametro event contiene dettagli sull'evento)

↑ bisogna fornire una classe il cui metodo actionPerformed contiene istruzioni che vanno eseguite al verificarsi dell'evento

• Costruire un oggetto dell'ascoltatore ed aggiungerlo al pulsante:

```
ActionListener listener = new ClickListener();
button.addActionListener(listener);
```

• CLASSI INTERNE: semplici classi ascoltatore interne dove servono per non "infastidire" il resto del progetto.

TIMER

• Aggiungere l'oggetto ascoltatore all'oggetto TIMER:

```
MyListener listener = new MyListener();
Timer t = new Timer(interval, listener);
t.start();
```

LEGGERE FILE DI TESTO

```
FileReader reader = new FileReader("input.txt");
Scanner in = new Scanner(reader);
```

FILEREADER LEGGE FILE SU DISCO
SCANNER LEGGE I DATI DAL FILE
(NEXT, NEXTLINE, NEXTINT, NEXTDOUBLE)

SCRIVERE FILE DI TESTO

```
PrintWriter out = new PrintWriter("output.txt");
out.print(29.5);
out.println("Ciao!");
out.close()   <- IMPORTANTE!
```

CREA FILE O LO RISCRIVE
SCRIVE TESTO
SCRIVE NUOVA LINEA
CHIUDE IL BUFFER!

FINESTRE DI DIALOGO FILE

```
JFileChooser chooser = new JFileChooser();
```

↓ METODI `chooser.showOpenDialog` / `chooser.showSaveDialog` → `JFileChooser.APPROVE_OPTION` (UTENTE SCEGLIE FILE) / `JFileChooser.CANCEL_OPTION` (UTENTE FACCIA "CANCEL")

ECCEZIONI

- CHECKED (verificate): dovute a circostanze esterne che il programmatore non può completamente prevenire
- UNCHECKED (non verificate): dovute a difetti del programmatore

2 SCELTE: 1) GESTIRE L'ECCEZIONE
2) FAR TERMINARE IL METODO QUANDO AVVIENE L'ECCEZIONE

- THROWS: per lanciare un'eccezione verificata

CATTURARE ECCEZIONI

```
TRY { istruzioni da controllare }
CATCH (IOException exception) { istruzioni in caso di IOException }
CATCH ( ... ) { ... }
:
FINALLY { istruzioni da eseguire sempre }   <- ES. reader.close();
```