JAVA:

Una classe può essere vista come un TIPO, che specifica le azioni possibili su di esso.

COSTRUTTORI:

Metodo con lo stesso nome della classe.

Si usa `new <Nome_Classe>();`

- deve essere private

Esiste un costruttore di default se non ne viene implementato uno

ASSEGNAMENTO:

- Tipi primitivi: l'effettivo valore nella variabile viene copiato nell'altra.

```
int x, y;
x = 5;
y = x;
```

- Tipi referenziali: viene copiata la reference, non l'oggetto.

```
Date d1, d2;
d2 = new Date(4,2,2004);
d1 = d2;
// ora c'è solo UN OGGETTO, non 2
```

!! SHARING (Aliasing)

Per questo l'operatore == NON va bene per due variabili oggetto. Darà falso anche se i due oggetti sono nello stesso stato

⇓

`obj1.equals(obj2)`

STRINGHE:

Sono immutabili. Una volta inizializzate non si può più aggiungere/rimuovere caratteri

↳ bisogna creare una nuova stringa con il costruttore String()

```
String a = new String("Hello");
```

ARRAY:

```
int[] ai1, ai2;
float[] af1;
ai1 = {1,2,3};
```

Se non si inizializza non viene allocato spazio in memoria

Solo se sono primitivi

In assenza di inizializzazione, non viene allocata memoria.

Si può usare anche `int[] i = new int[10];`

ARRAY DI OGGETTI → VA usato new

```
Person[] person;
person = new Person[20]; // Array CREATO
person[0] = new Person();
```

ITERARE in un ARRAY:

```
for(int i : a) // "for each i in a"
```

↳ è anche il TIPO DELL'ARRAY su

THIS: this.<nome attributo>

serve per riferirsi ad un attributo della classe all'interno di un suo metodo che la vuole modificare

Può essere usato anche per ritornare la reference dell'oggetto.

es.: `return this;` // returns the modified set

Si può usare "this" per chiamare un costruttore all'interno di un altro

this(<list of parameters that identify the right constructor>)

STATIC:

Un metodo static può accedere solo ad attributi e metodi static.

Un metodo tradizionale può accedere a metodi static.

FINAL:

Per dichiarare attributi COSTANTI

Non potranno più essere modificati

OVERLOADING: chiamare metodi con lo stesso nome ma distinguerli con tipi e numero dei parametri in ingresso

ENUM:

```
enum Size {SMALL, MEDIUM, LARGE, X_LARGE}
Size s = Size.SMALL;
```

Size è effettivamente una classe.

Può avere anche attributi e metodi

```
enum Color {Red, White, Blue}
for (Color c : Color.values()) {...}
```

EREDITARIETÀ:

public class Class2 extends Class1
la sottoclasse eredita tutta l'implementazione della superclasse.

- Può aggiungere attributi e metodi ma anche ridefinire quelli della superclasse.

↓

si lascia uguale il numero di parametri ed il tipo del metodo

↓

OVERRIDING

Si utilizza l'annotazione @Override.

↓ il compilatore manderebbe errore altrimenti.

super.<nome metodo>(<lista par. corrente>) chiama il tale metodo della super classe.
da solo, super chiama il COSTRUTTORE

CLASSE OBJECT: ogni classe in assenza di specifiche ESTENDE Object.

- .toString(): ritorna una rappresentazione testuale dell'oggetto.

- Se vogliamo evitare che una classe venga estesa la dichiaro final
- Analogamente se voglio evitare OVERRIDING.

CLASSE ASTRATTA:

Classe la cui implementazione non è specificata.
↳ può contenere uno o più metodi astratti.

- Non si possono creare istanze di classi astratte.

```
abstract class Shape{
    ...
    abstract void show();
}

class Circle extends Shape{
    void show(){
    ...
    }
}

Shape s = new Shape(); // SBAGLIATO
Shape s = new Circle(); // Giusto, è un CERCHIO
```

• Una classe astratta può avere anche metodi NON ASTRATTI
↳ così verranno ereditati da tutte le sottoclassi SENZA BISOGNO di override.

Problema: se ho 2 classi molto diverse
es.: Car e Toy, è difficile implementare ToyCar.

INTERFACCE:

È una classe che
- può avere solo attributi static e final
- " " " metodi pubblici e astratti

```
interface <name> {
    ...
}
```

- PUÒ EREDITARE altre interfacce

Una classe può implementare più interfacce, ma estendere al massimo una classe.
(ecco perché ESISTONO LE INTERFACCE)

Se una classe NON È ASTRATTA, deve fornire le implementazioni delle interfacce che usa!

Un'interfaccia può avere metodi statici, metodi DEFAULT, che vengono ereditati a meno che la sottoclasse faccia @Override.

INFORMATION HIDING:

PACKAGE: classi raggruppate in pacchetti, un pacchetto raggruppa classi definendo regole di visibilità.
è una cartella

UNITÀ DI COMPILAZIONE: file che contiene dichiarazioni di una o più classi (o interfacce)
↳ UNA È PUBBLICA e ha il nome del file.
→ Si può specificare il pacchetto a cui appartiene.

Pacchetto = cartella con dentro una o più unità di compilazione

VISIBILITÀ:
- public: visibili a tutti
- private: - visibili solo all'interno della classe
  - invisibili alle sottoclassi
- protected: - visibili a classi nello stesso pacchetto
  - visibili a sottoclassi
- "friendly": - visibili a classi nello stesso pacchetto
  - visibili a sottoclassi SOLO nello stesso pacchetto

PUBLIC = promessa che la classe non cambierà mai
⇒ attributi MEGLIO PRIVATE, PROTECTED o FRIENDLY

solo quando classi nello stesso pacchetto devono avere accesso privilegiato