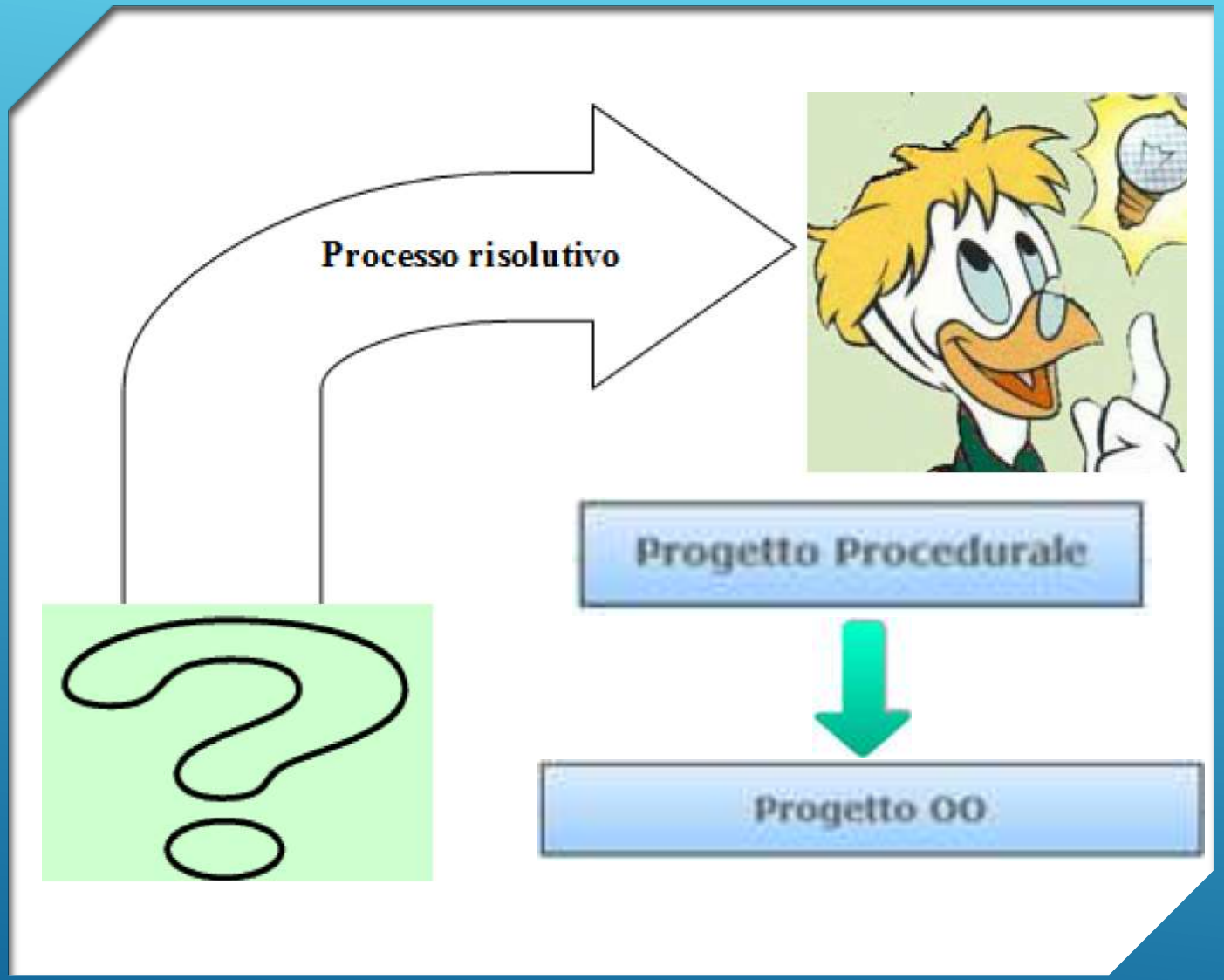
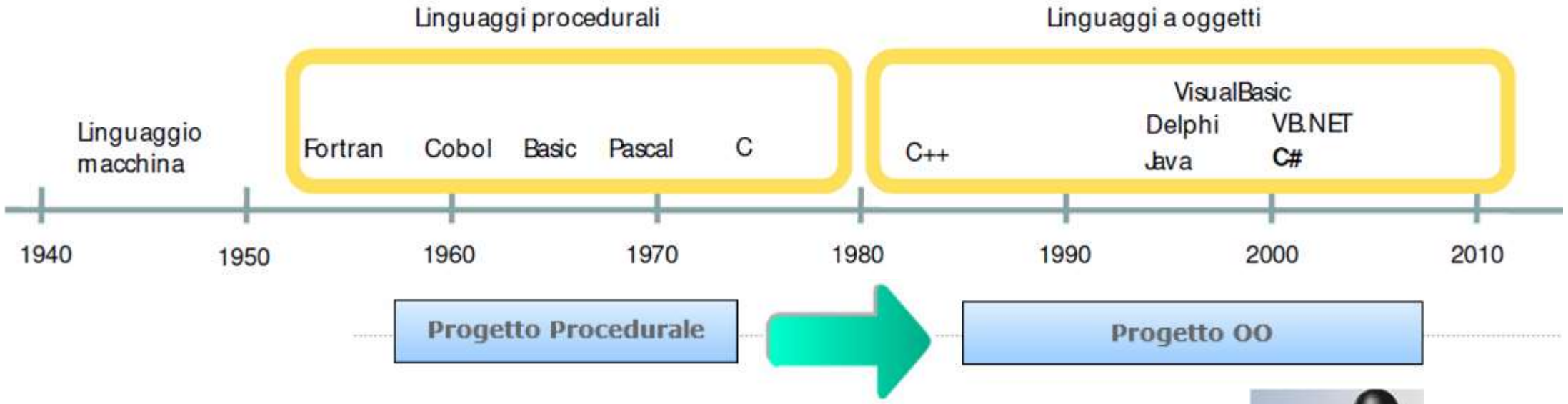


PARADIGMA DI PROGRAMMAZIONE

VS OOP

Paradigmi di programmazione

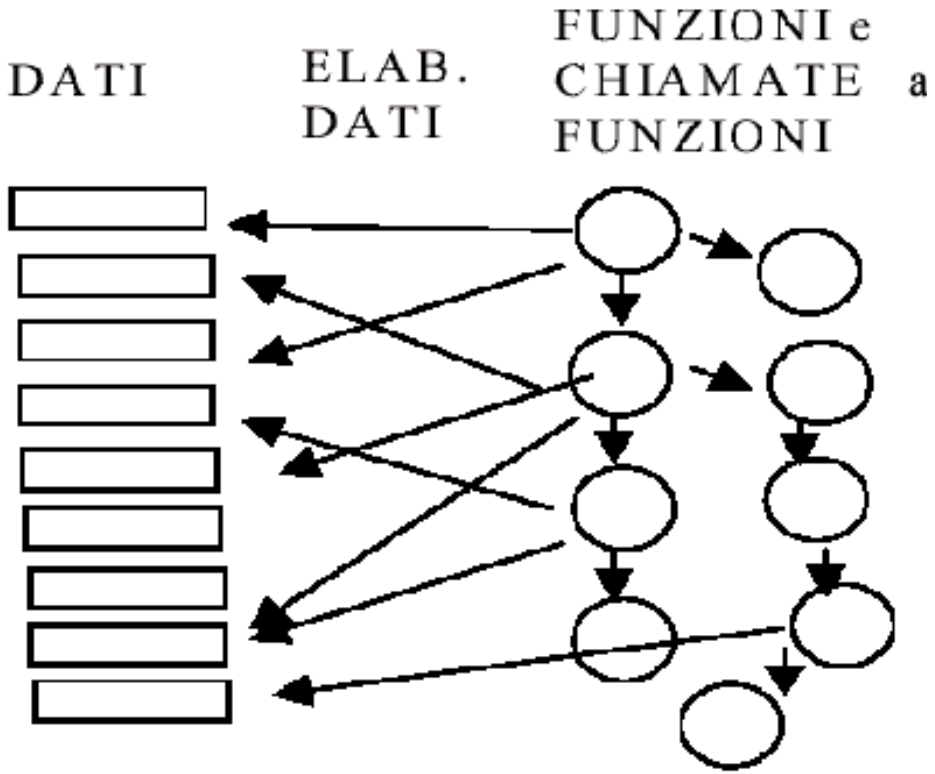


programmazione imperativa

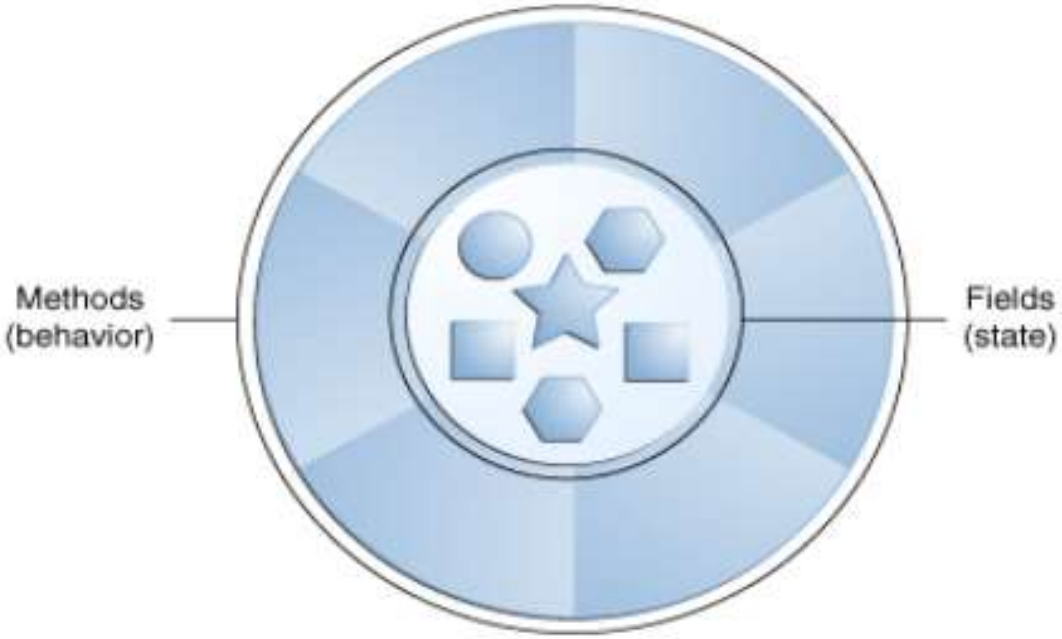
si basa su istruzioni-comando:
un istruzione impartisce ordini veri e propri

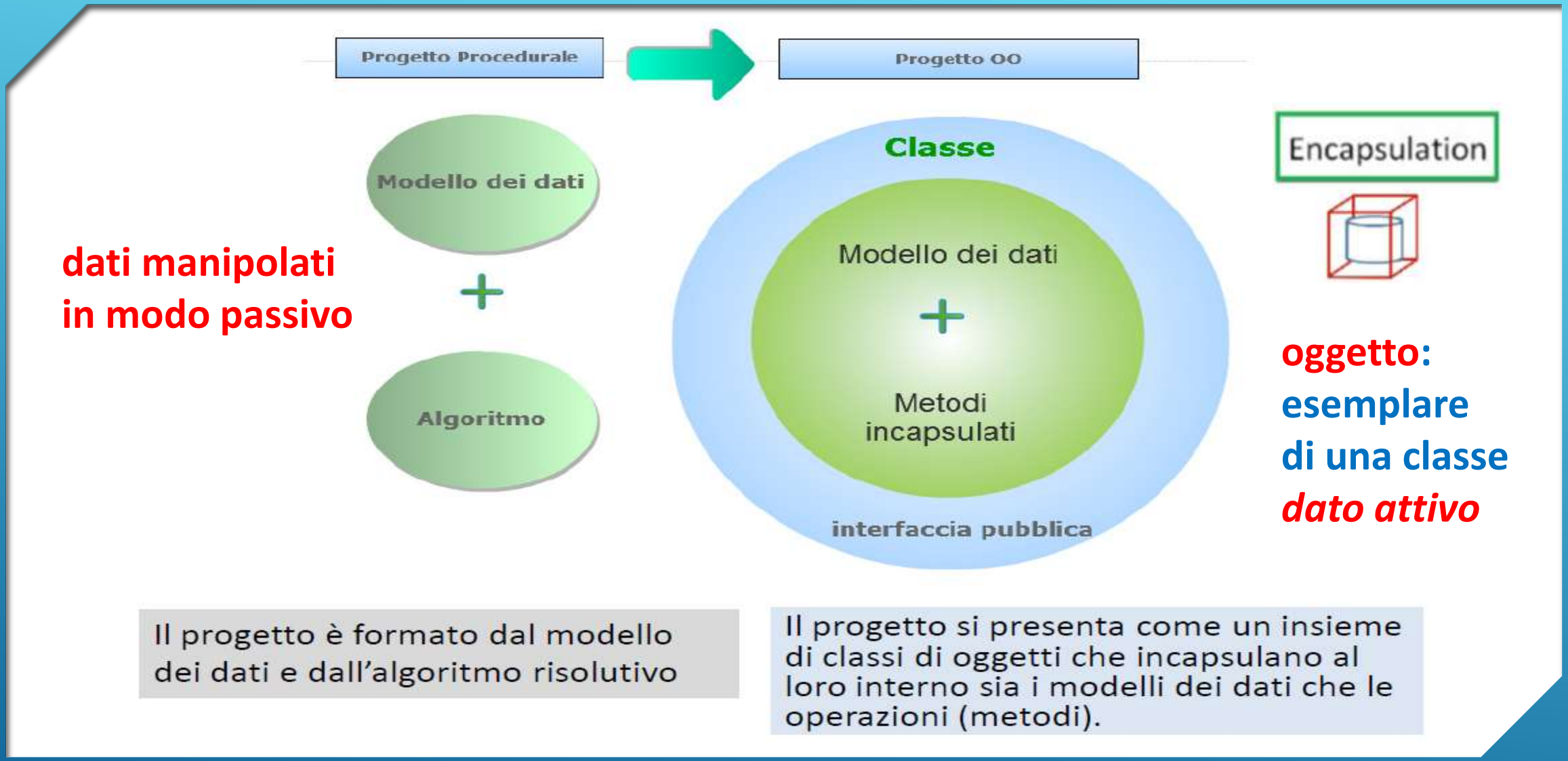
Paradigmi di programmazione: procedurale tradizionale vs Orientato agli Oggetti

Tradizionale:



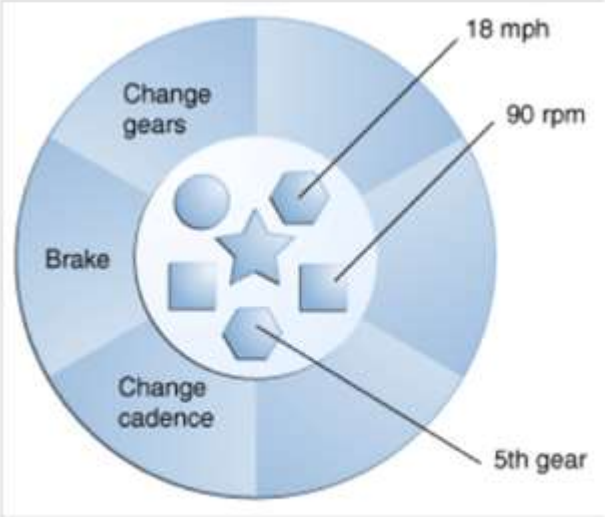
OOP:
modello di *dati attivi*





Paradigmi di programmazione: : procedurale tradizionale vs Orientato agli Oggetti

Esempio



Oggetto

Stato attuale*: *18 miglia all'ora, 90 giri al minuto, 5^ marcia

Comportamenti* : *cambia marcia

frena

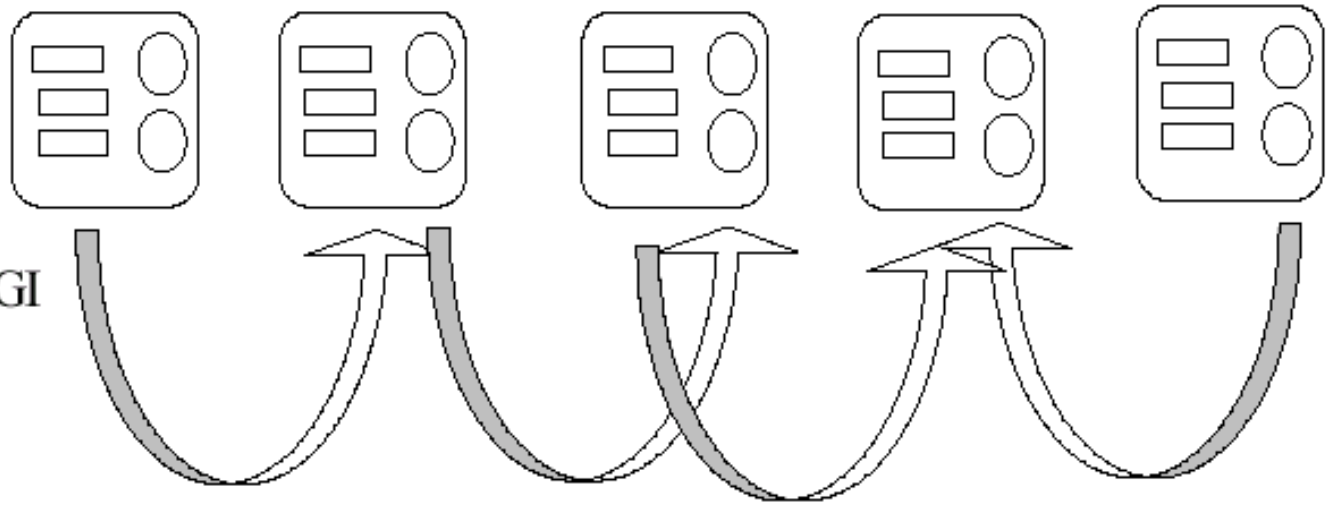
cambia cadenza

Oggetto bicicletta descritta da un modello (dato attivo: incapsula proprietà nascoste e comportamenti)

Ad Oggetti:

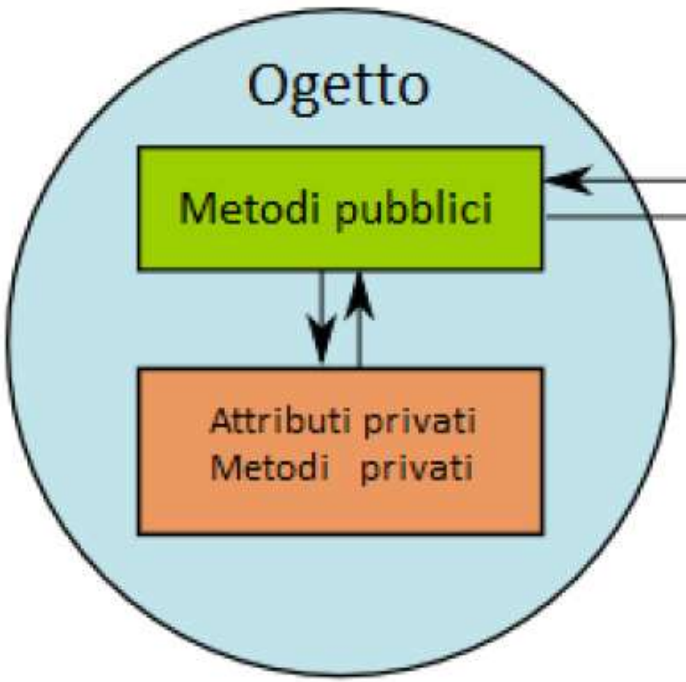
OGGETTI =
dati + funzioni
che vi operano

SCAMBIO DI MESSAGGI
(richieste di servizio)
TRA OGGETTI



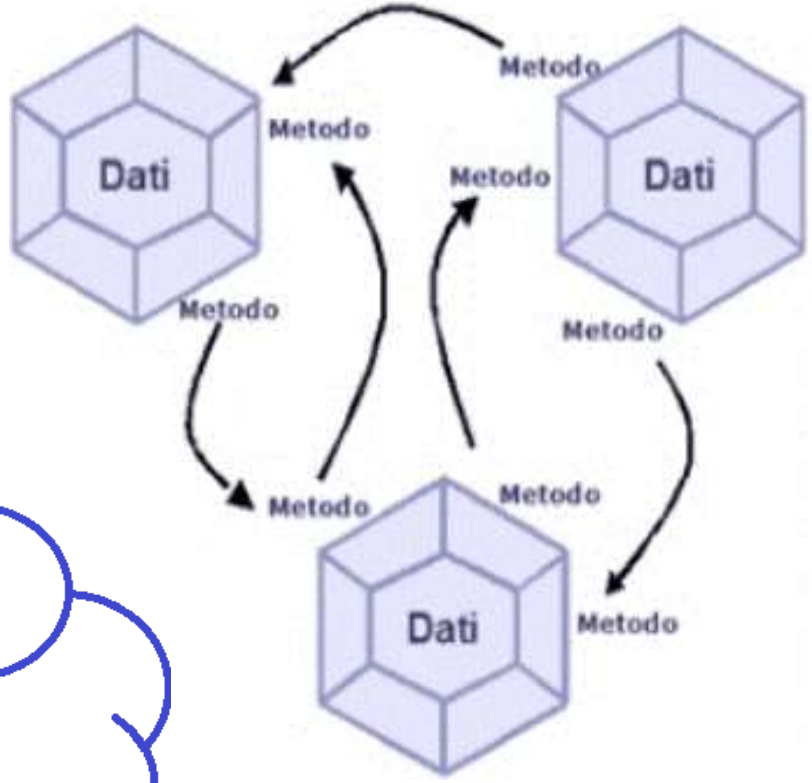
Oggetto: può chiedere o fornire un *servizio*

Oggetti che incapsulano dati e funzioni e scambiano messaggi tra loro



SW di qualità:

- Protetto
- Modulare
- Riusabile
- Documentato
- Incrementale ed Estensibile



Oggetti: dati attivi che incapsulano attributi (informazioni nascoste) prevedendo metodi (interfacce) per accesso controllato e comportamenti (servizi che l'oggetto può chiedere/fornire)

Classe:* *descrizione astratta

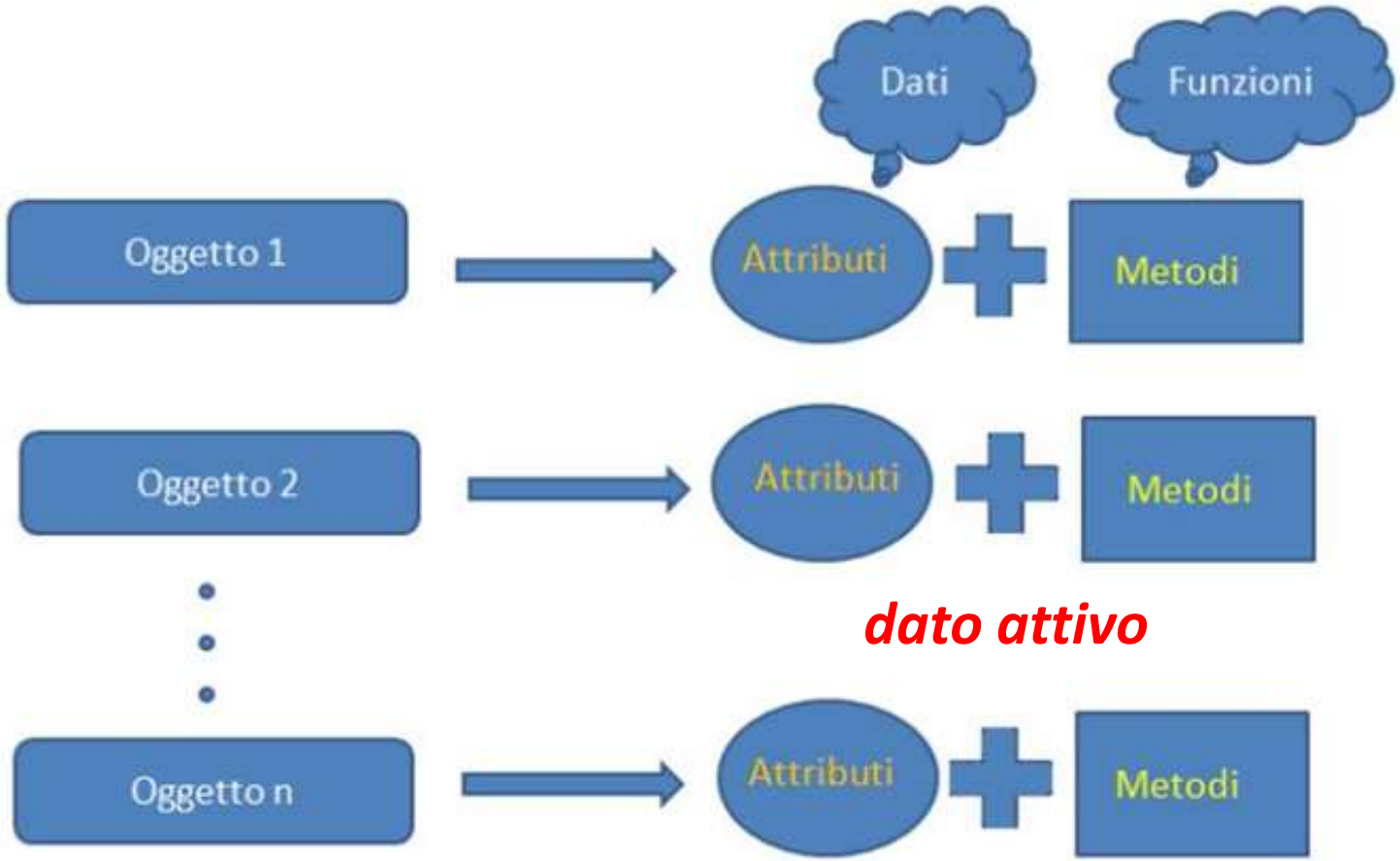
di una categoria di oggetti …..

Paragonabile al progetto di un'infrastruttura
che può poi essere realizzata o meno
con
l'istanziazione dei suoi oggetti.

Definizione del tipo

Classe

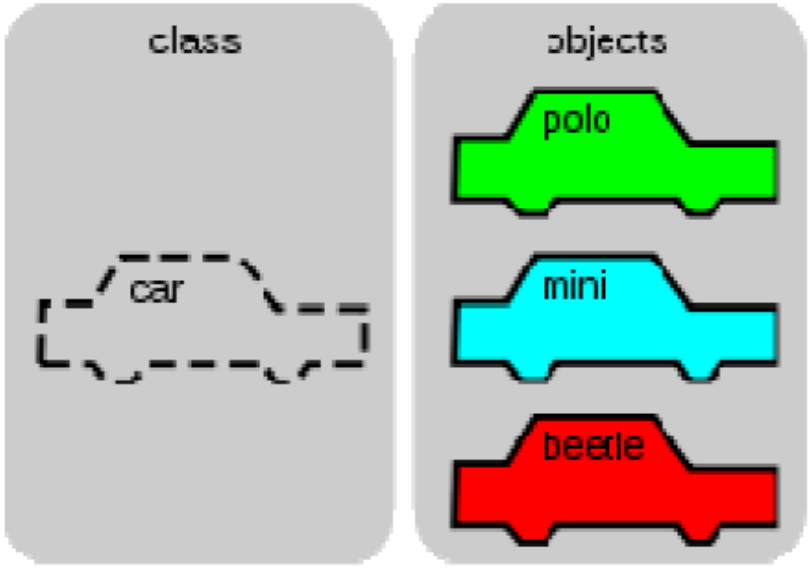
oggetto:
esemplare
di una classe



dato attivo

Oggetti : dati attivi che incapsulano dati (attributi) e comportamenti (metodi)
Classe: tipo di un insieme di oggetti con attributi e metodi comuni.

CLASSE
cioè
TIPO



oggetti:
esemplari
di una classe
dati attivi

Nella **OOP** quindi si descrivono gli oggetti (dati + comportamenti), cioè **si creano nuovi tipi di dato,** poi **si istanziano gli oggetti** stessi.

Oggetti : dati attivi che incapsulano dati e forniscono metodi

oggetto → automobile

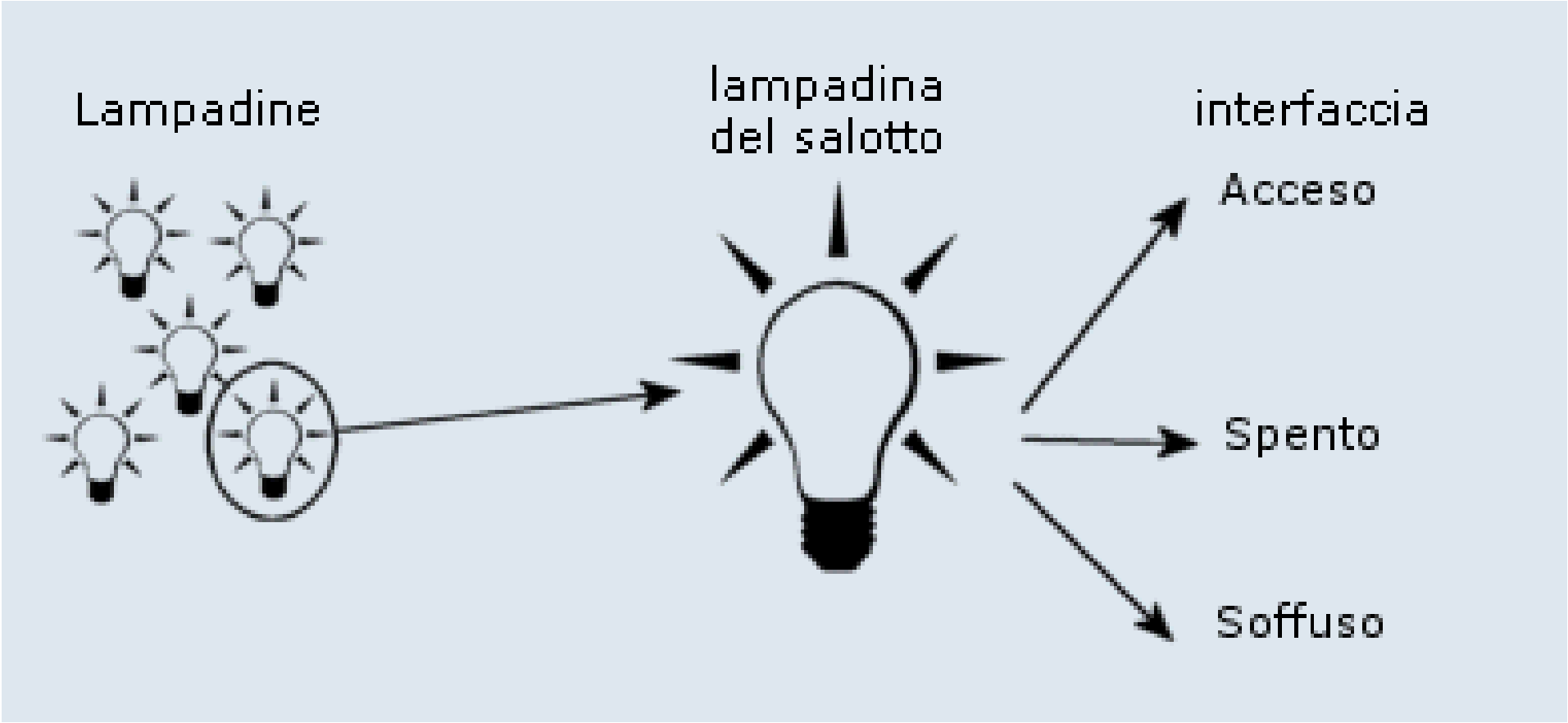
attributi → Velocità, colore, prezzo, dimensioni

metodi → Girare, muoversi, fermarsi, tamponare

istanza → 60 km/h, rosso, 15000 €,
4(lung) x 1,60(h) x 1,50(larg)

Oggetti : dati attivi che incapsulano dati e forniscono metodi.
Un oggetto è un'istanza di una classe cioè la realizzazione concreta (in RAM) di un esemplare

Classe cioè TIPO di oggetti: dati attivi che incapsulano dati e forniscono metodi

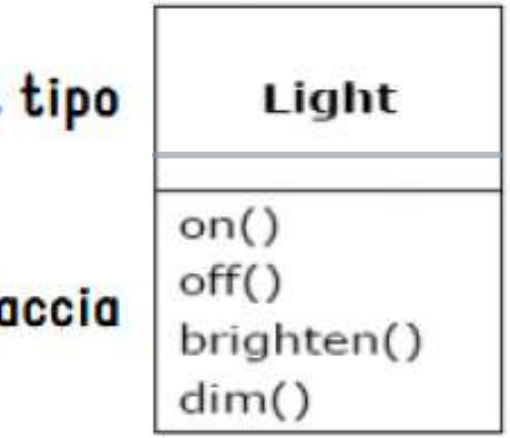


Oggetti : dati attivi che incapsulano metodi (interfacce)

Oggetto e interfaccia

Nome del tipo

Interfaccia

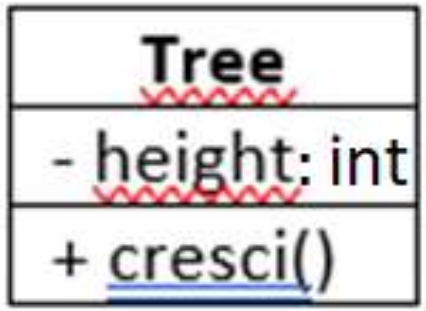


```
Light lt = new Light();
lt.on();
```

uso del metodo costruttore

Oggetti : dati attivi che incapsulano metodi (interfaccia).
La classe è il tipo di oggetti con attributi e metodi comuni

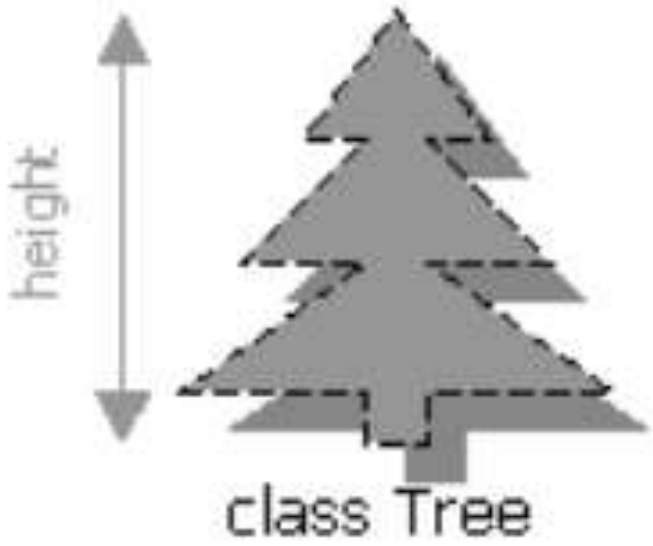
UML



nome classe

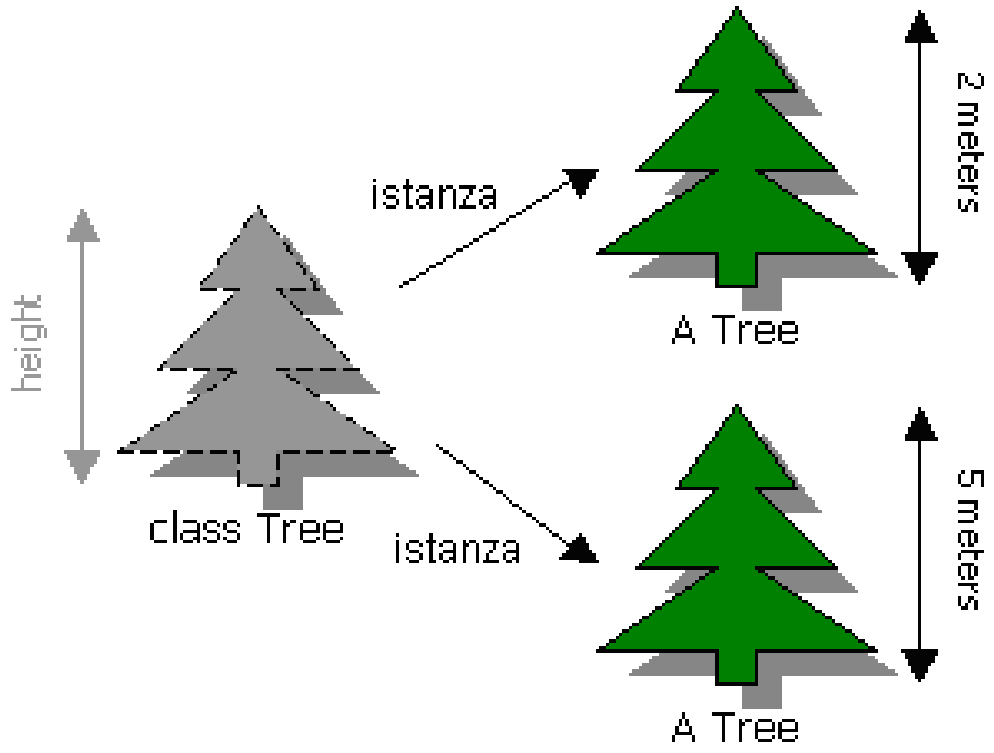
attributo nascosto

metodo pubblico



Classe: ***descrizione astratta*** *di una categoria di oggetti*
che ne illustra le caratteristiche (attributi)
ed i comportamenti comuni (metodi)

UML: diagramma statico di una classe che illustra il nome della classe, gli attributi e i metodi (diagramma di implementazione se dettaglia le specifiche per un dato linguaggio)



Oggetto: ***istanza*** *di una classe*
cioè ***realizzazione concreta***
con occupazione di memoria
(RAM)

I costruttori non svolgono *funzioni*, ma inizializzano un'istanza della classe

- sono **specifici di quella particolare classe**

NomeClasse ( )

- ha visibilità public
- non ha tipo di ritorno
- ha lo stesso nome della classe

Costruttori: metodi per creare oggetti di quel tipo classe

```
class Tree {
    /** altezza dell'albero */
    private int height;     // attributo nascosto
                            // inizializzato di default a zero
    /** Crescita di un metro */
    public void cresci() {
        height = height + 1;
    }
    public static void main(String [] args) {
        // Creazione di un oggetto della classe Tree
        Tree tree1 = new Tree();   // uso costruttore di default
        // richiesta di crescere di un metro
        tree1.cresci(); //accesso controllato ad attributo nascosto
        System.out.println("altezza: " + tree1.height);
    }
}
```

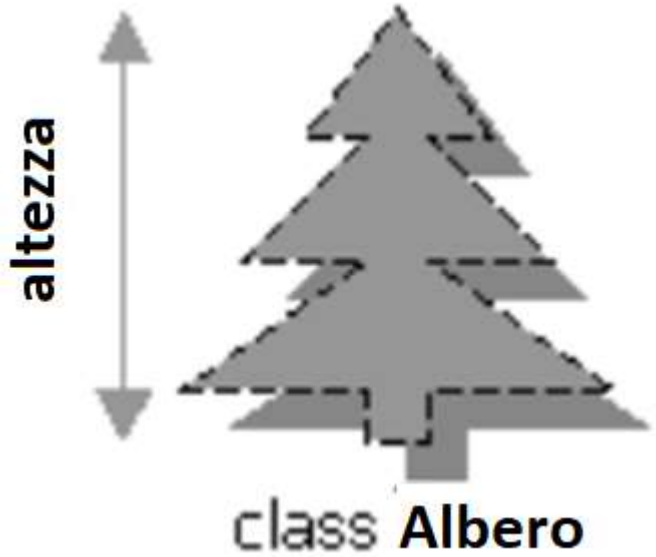
Codice Java: applicazione (gli attributi sono visibili nei metodi della classe)

```
class Albero {
  /** altezza dell'albero */
  private int altezza;          // attributo nascosto
                                // inizializzato di default a zero

  /** costruttore di default */
  public Albero (){
        altezza = 1;      // inizializza l'attributo ad un metro
  }

 /** costruttore parametrico
    @param val altezza dell'albero
  */
  public Albero (int val){
        altezza = val;          // inizializza l'attributo al valore passato come parametro
  }

  ……..
```



Codice Java: classe Albero con costruttori ….


```
/** per modificare il valore*/
public void setAttributo (tipoAtt valNew) {
                    attributo = valNew;
}
```

Metodi di accesso: metodi per usare in modo controllato gli attributi privati


```
/** per vedere il valore*/
public tipoAtt getAttributo () {
                    return attributo;
}
```

Metodi di accesso: metodi per usare in modo controllato gli attributi privati

```
/** metodo di accesso per modificare l'attributo nascosto
    @param val altezza dell'albero
 */
 public void setAltezza (int val){
          altezza = val;
 }
 /** metodo di accesso per visualizzare l'attributo nascosto
    @return altezza dell'albero
 */
 public int getAltezza ( ){
          return altezza ;
  }
 /** crescita di un metro */
 public void cresci() {
          altezza = altezza + 1;
 }
} // fine class
```

Codice Java: …. classe Albero con metodi di accesso agli attributi privati

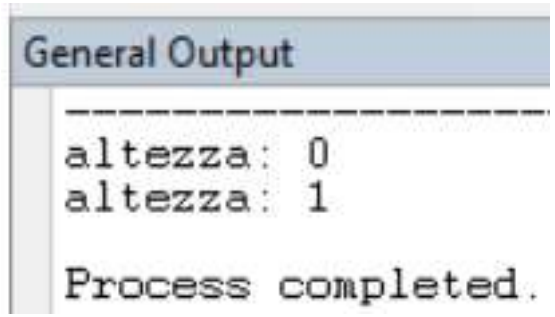
```
public class UsaAlbero {

    public static void main(String[] args) {

    // Creazione di un oggetto della classe Albero
    Albero tree1 = new Albero();   // uso costruttore di default


    System.out.println("altezza: " + tree1.getAltezza() );
    // richiesta di crescere di un metro
    tree1.cresci();
    System.out.println("altezza: " + tree1.getAltezza() );
    }
}
```

// accessi controllati
// ad attributo nascosto



Codice Java: applicazione che usa la classe Albero

*Sintassi per creare un'**istanza** di una classe (cioè un oggetto memorizzato in RAM):*

```
NomeClasse nomeOggetto = new NomeClasse(); // uso metodo costruttore
                                           // con lo stesso nome della classe
```

tipo

Sintassi per chiedere, come servizio, ad un oggetto di eseguire un suo metodo:

```
nomeOggetto.nomeMetodo(); // operatore dot per accedere al metodo
                          // chiedendo all'oggetto
                          // di eseguire quell'azione
```

*Sintassi per creare un'**istanza** di una classe (cioè un oggetto memorizzato in RAM):*

NomeClasse *nomeOggetto* = **new NomeClasse ()**; **// uso metodo costruttore di default**
// con lo stesso nome della classe

tipo

*Sintassi per creare un'**istanza** di una classe inizializzando attributi:*

NomeClasse *nomeOggetto* = **new NomeClasse (*tipo nome*)**; **// uso metodo costruttore parametrico**
// con lo stesso nome della classe
// per inizializzare un attributo private
// con attenzione all'information hiding

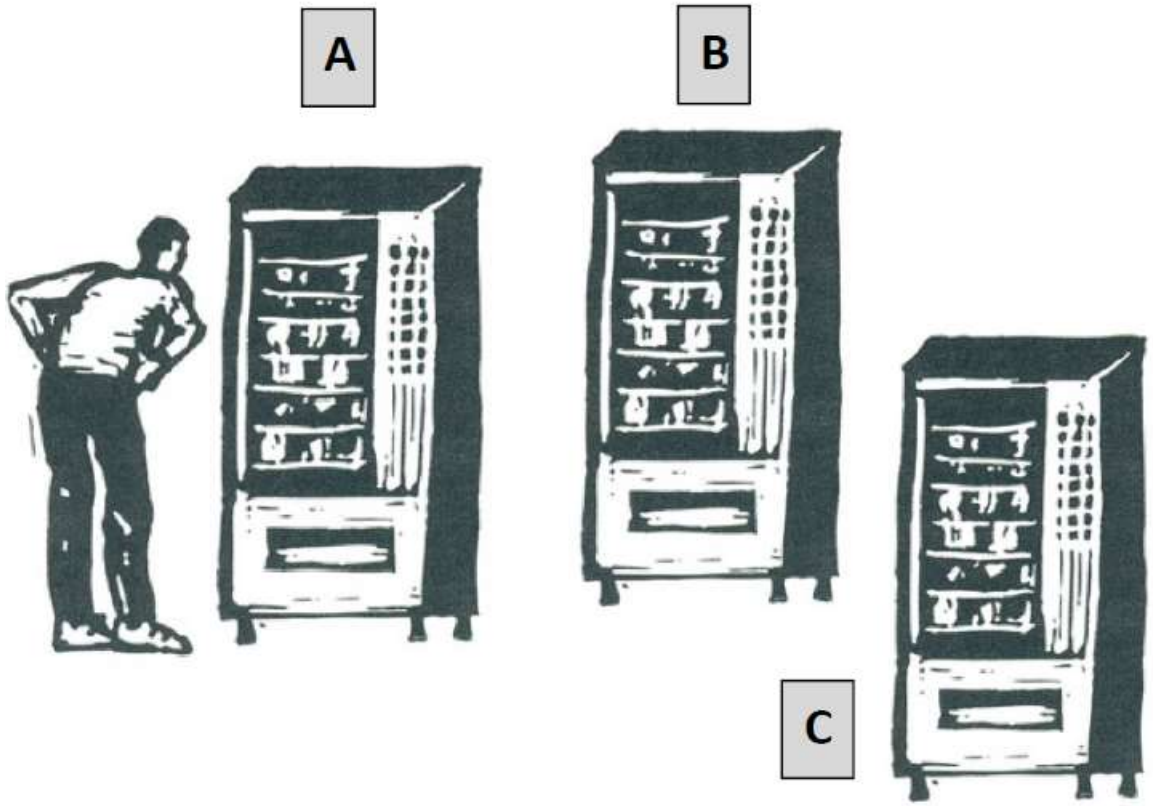
tipo

NomeClasse *nomeOggetto* = **new NomeClasse (*tipo nome1, tipo nome2*)**; **// uso altro costruttore parametrico**
// con lo stesso nome della classe

tipo

Polimorfismo : metodi con lo stesso nome

Costruttori e overloading: stesso nome ma diverso numero di parametri o tipo di parametri o diverso ordine dei parametri



UML

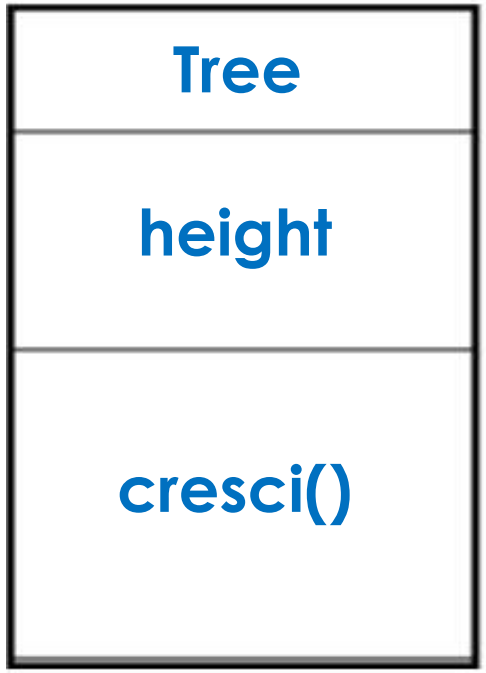
Distributore_bibite
numero_bottigliette_acqua numero_lattine-aranciata numero_lattine-the
Inserisci-monete Scegli_bibita Prendi_bibita Prendi_resto

diagramma di specifica …

***UML**: diagramma statico di una classe che illustra il nome della classe, gli attributi e i metodi*
Oggetti : istanze di quel tipo classe

diagramma di specifica …

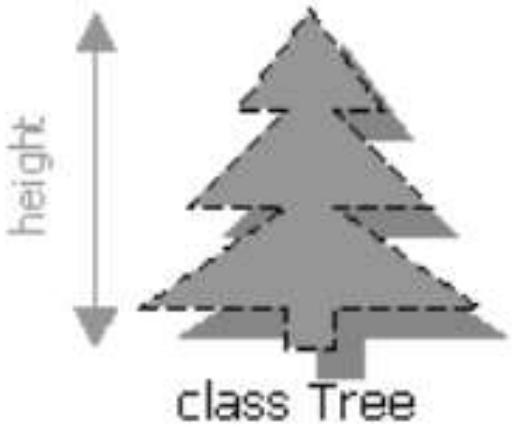
UML



nome della classe

attributi senza esplicitare tipo

metodi senza esplicitare tipo



***UML**: diagramma statico di una classe. Nel progetto: diagramma di specifica …*

diagramma di implementazione esplicitando i tipi

… ed i modificatori di accesso (visibilità)

ClassName
- Attributo1 : tipo1 - Attributo2 : tipo2 = "Valore di Default" …. …..
+ operazione1() + operazione2(Lista di parametri) + operazione3() : Tipo restituito

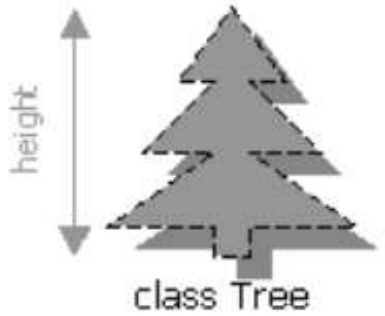
UML

Tree
- height: int
+ cresci()

nome della classe

attributi *esplicitando tipo e visibilità*

metodi *esplicitando tipo e visibilità*



***UML**: diagramma statico di una classe. Nel progetto: dal diagramma di specifica al diagramma di implementazione*

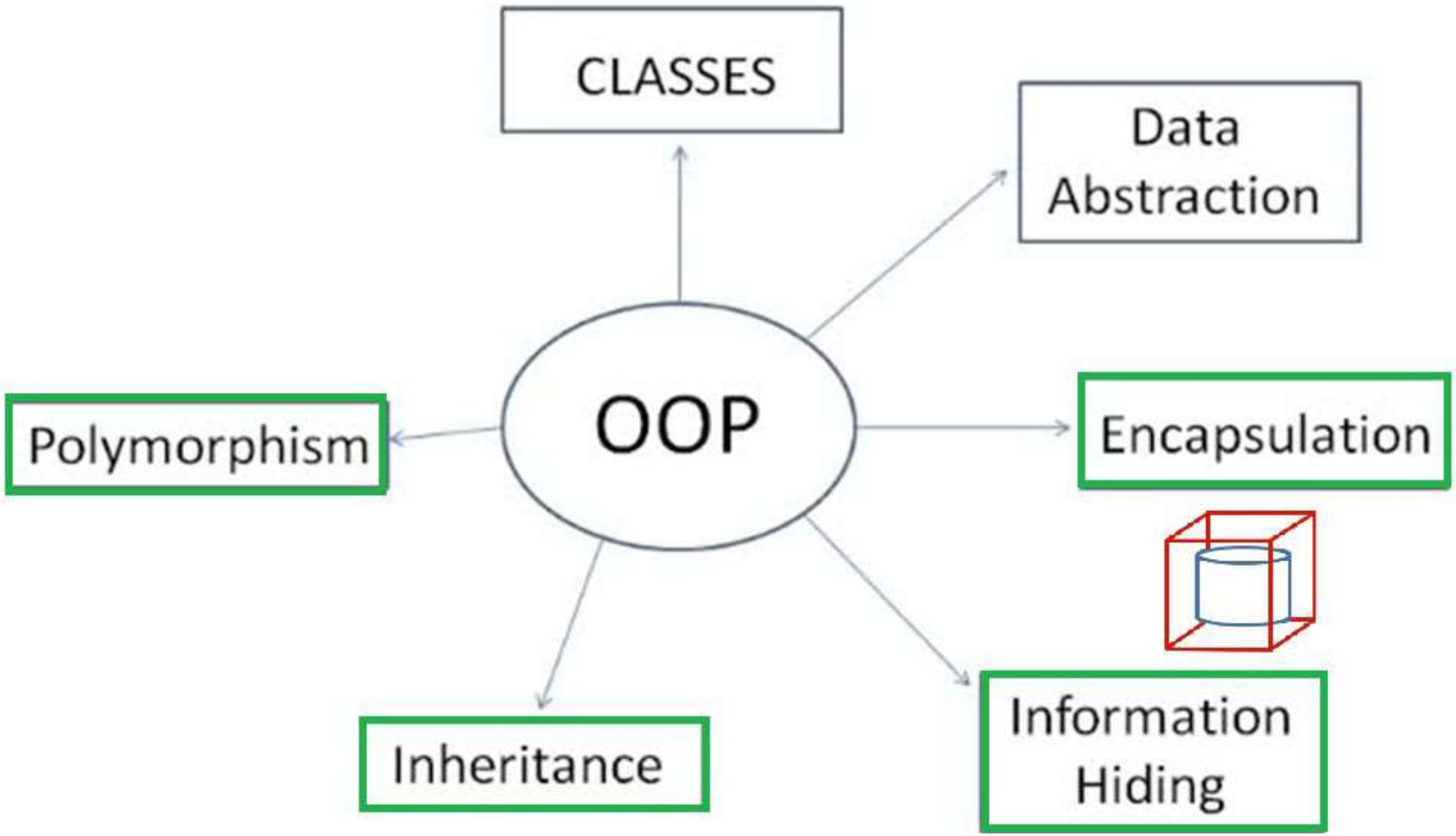
SITOGRAFIA

Introduzione …

… concetto di classe e oggetto nella OOP

Approfondimenti: tecnologia OO
e OOP (slides in formato pdf)

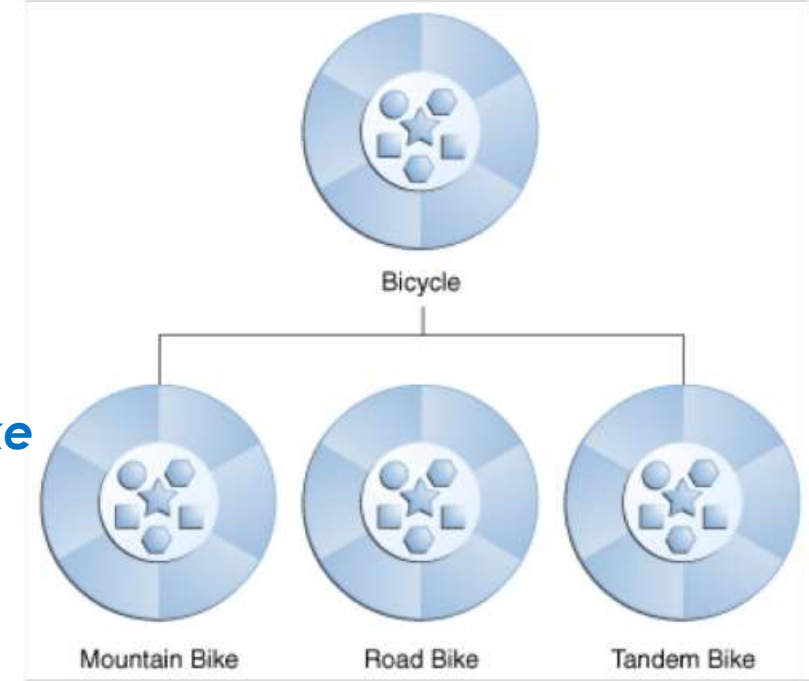
Esercizi: **UML** nell'illustrare class diagram
(slides in formato pdf)



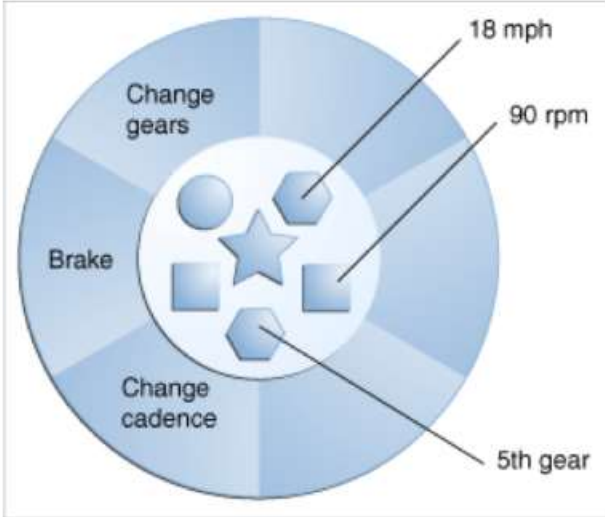
Meccanismi della OOP

Gli oggetti sono descritti da modelli

Una Montain Bike
è
una Bicycle



Classi e superclasse



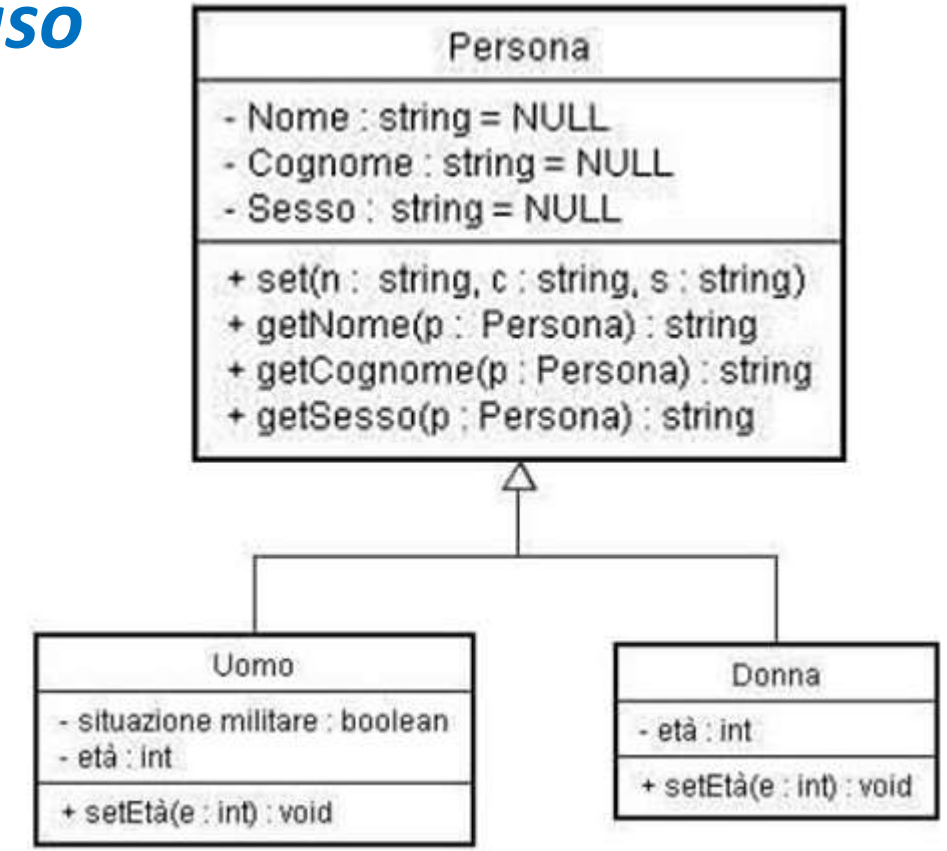
Oggetto

http://docs.oracle.com/javase/tutorial/java/

Classi e superclasse: ereditarietà singola in Java

UN MECCANISMO RISOLUTIVO:
EREDITARIETÀ

L'ereditarietà serve per *non ripartire da zero* quando serve una nuova classe, permettendo di ***poterla definire a partire da una già esistente*** nella logica del ***riuso***



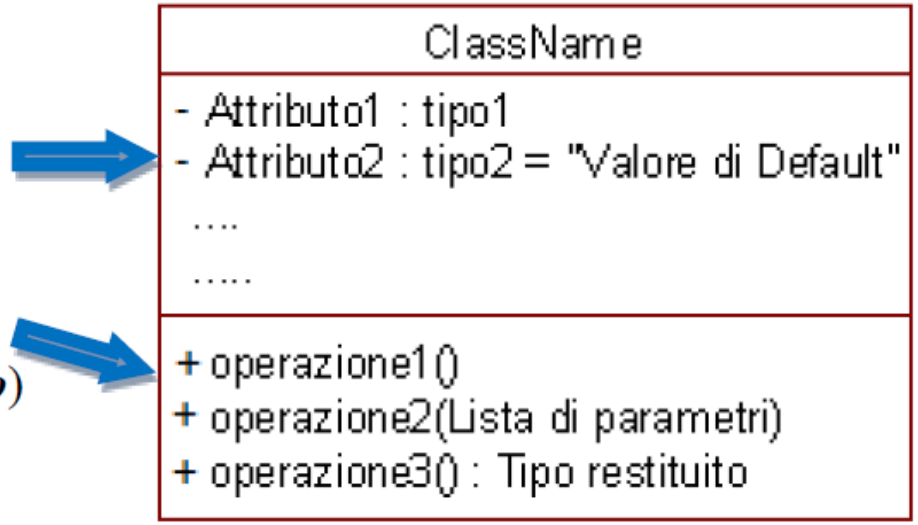
Con uso **UML** si progetta la **classe nomeClasse** con:

- **proprietà private** (incapsulate nella classe)
- **metodi pubblici**:
  - che permettono di accedere alle proprietà private (*metodi di accesso*)
  - che definiscono l'elaborazione (operazioni proprie delle varie *istanze*)

Descrizione con UML **diagramma statico di una classe:**

Il simbolo – specifica che l'accesso è privato
(***specificatore di accesso privato***)

Il simbolo + specifica che l'accesso è pubblico
(***specificatore di accesso pubblico***)



UML : diagramma statico di una classe che illustra il nome della classe, gli attributi (informazioni nascoste) e i metodi (miniguida introduttiva al linguaggio Java pg.20)

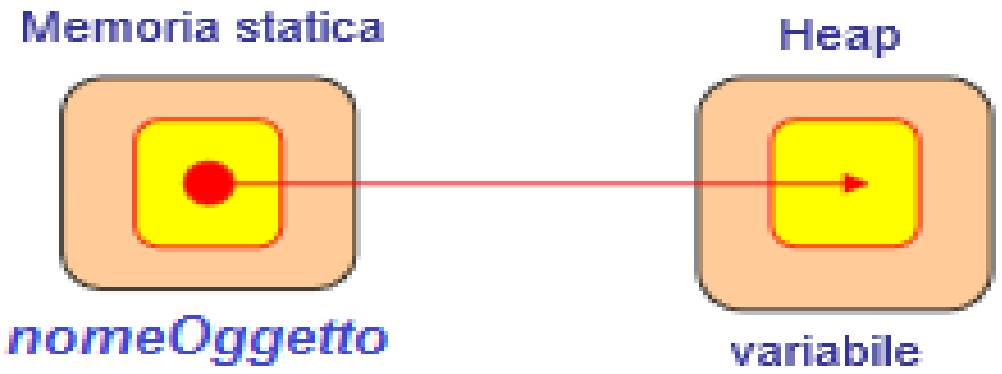
modificatori di accesso (visibilità)

- **Pubblica (+)**: l'attributo è accessibile da qualsiasi altro oggetto dotato di riferimento all'oggetto che contiene l'attributo in questione;
- **Privata (-)**: l'attributo è accessibile solo all'interno della classe di appartenenza (dichiarante);
- **Protetta (#)**: l'attributo è accessibile da tutte le istanze delle classi che "ereditano" da quella in cui l'attributo è definito;
- **Package (~)**: l'attributo è accessibile da qualsiasi altro oggetto istanza di classi appartenenti allo stesso package o in un altro ad esso annidato a qualsiasi livello.

stessa sottocartella (default)

Costruttori: dove e cosa memorizzano

*NomeClasse nomeOggetto = **new** NomeClasse();*

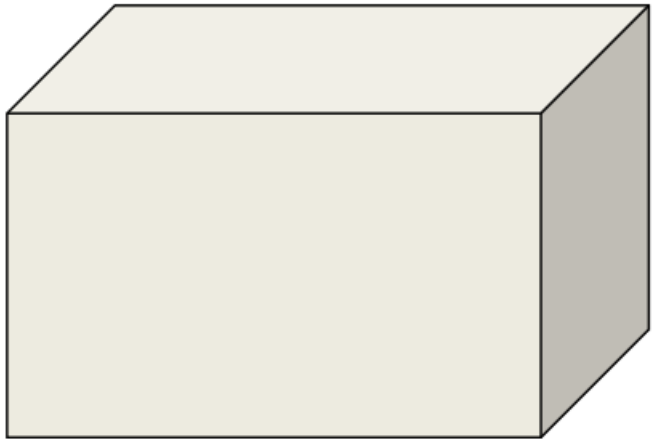


nomeOggetto
indica la posizione in RAM
cioè il riferimetno all'oggetto

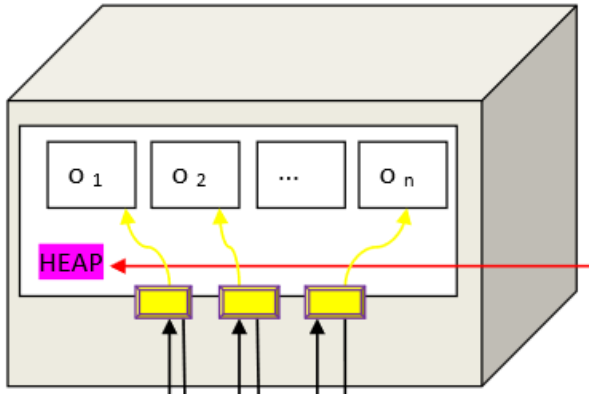
Memoria statica (STACK) ed Heap *(miniguida introduttiva al inguaggio Java pg.24)*

Il ciclo di vita di un oggetto in java inizia con il new e finisce con il garbage collector.

new ----------> **garbage collector**



RAM



RAM

1. HEAP è lo spazio della memoria RAM dove vengono memorizzati gli oggetti

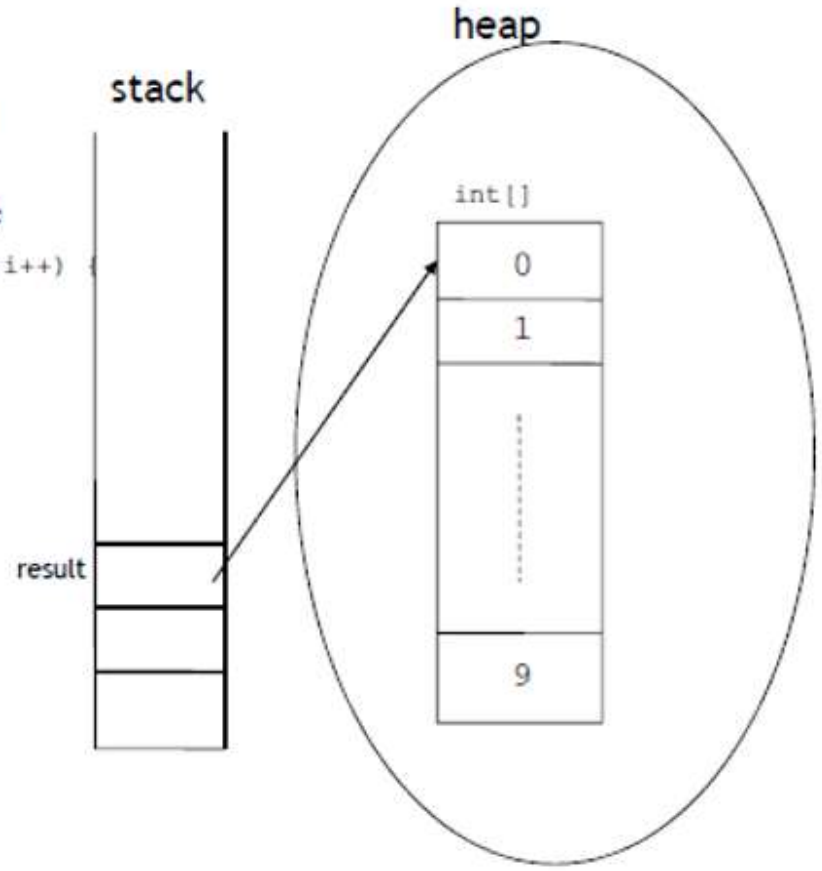
Garbage Collector

Garbage Collector si occupa di togliere gli oggetti dalla memoria HEAP.

Memoria Heap e Garbage Collector per eliminare automaticamente i references (*approfondimenti*)

Array : Rappresentazione

```
...
public int[] iniVettore()  {
    int[] result;
    result = new int[ 10 ];
    for( int i=0; i<result.length;i++) {
         result[i] = i;
    }
    return result;
}
```

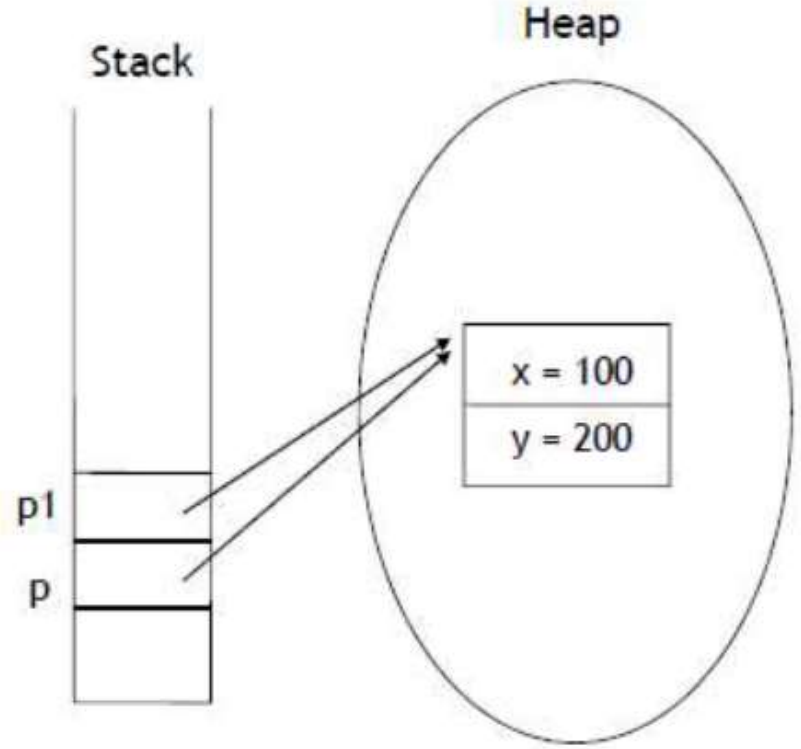


Un vettore in Java è un oggetto (scandire un vettore per inizializzarlo: esercizi risolti pg.20)

In generale per qualsiasi oggetto:

- *il nome è l'indirizzo di RAM dove è allocato cioè il valore del reference*
- *l'istruzione nomeOggetto1 = nomeOggetto significa che entrambi i nomi puntano alla stessa area di RAM*

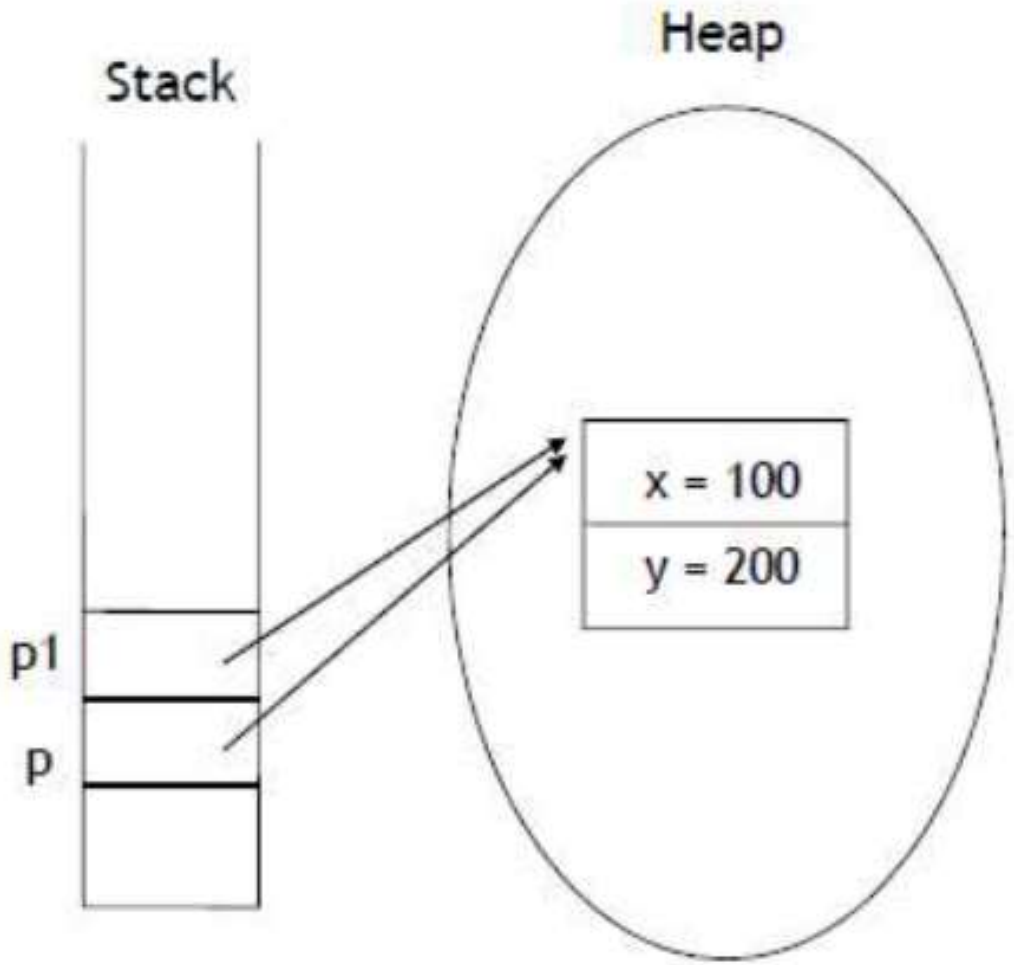
```
Point p = new Point( 100, 200 );
Point p1;
p1 = p;
```



*Riferimenti allo stesso oggetto (**esercizi** risolti pg.9)*

```
public class Point {
    private int x;
    private int y;

    public Point(int dx, int dy) {
        x = dx;
        y = dy;
    }
}
public class Test {
    public static void main(String[] args) {
        Point p = new Point( 100, 200 );
        Point p1;
        p1 = p;
    }
}
```



*Riferimenti allo stesso oggetto (**esercizi** risolti pg.9)*