Metodi static

- I metodi static possono usare gli attributi static
- I metodi static devono creare una istanza per poter usare attributi o metodi non-static
- Esempio

```
public class Test {
   private static int contaSta = 0;
   private int conta = 0;
   public static void main(String[] args) {
      incrementa();
      Test t = new Test();
      t.increm();
   }
   public static void incrementa() { contaSta++; }
   public void increm() { conta++; }
}
```


Classe Math

- La classe di libreria Math ha solo attributi e metodi static
  - Attributi E e PI
  - Metodi abs, max, log, pow, sin, cos, ...

Costruttore

- Un costruttore è un metodo “speciale” di una classe
  - Tipicamente è usato per fare delle inizializzazioni
  - E’ chiamato al momento della creazione di una nuova istanza (con new)
  - Può prendere dei parametri in ingresso
  - Non ha tipo e parametro di ritorno
  - Ha un nome prefissato (uguale al nome della classe)
  - Non viene chiamato esplicitamente
  - Può non essere unico
    - Più costruttori con lo stesso nome e con parametri diversi, ovvero faccio overloading
- Nota: anche per i metodi posso usare overloading
  - Concetto di late binding

Costruttore

- Una classe con un costruttore

```
public class Account {
   private int balance;
   public Account(int openingBalance) {  // un costruttore di Account
       setBalance(openingBalance);
   }
   public void setBalance(int amount) {
       if (check())
          balance = amount;
   }
   public boolean check() { … }
   …
}
```

Istanza corrente

- La parola chiave `this` riferisce l'istanza corrente
- Può essere utile
  - Per indicare un attributo in caso di omonimia con parametri o variabili locali

```
this.nome = nome;
```

  - Per chiamare un costruttore diverso (solo come prima istruzione del costruttore)

```
this(1); // chiama costruttore con parametro int
```


Passaggio parametri

- I parametri passati nelle chiamate a metodi
  - Sono passati per valore (ovvero viene effettuata una copia)
  - Non esiste il passaggio per riferimento
  - Le modifiche sui parametri vengono perse alla fine del metodo chiamato
- Nelle chiamate a metodi, il passaggio di un riferimento ad un oggetto
  - Avviene tramite copia, ovvero passiamo una copia del riferim
  - Le modifiche sull'oggetto rimangono (poiché l'oggetto è lo stesso anche se abbiamo copiato il riferimento)

Confronto tra oggetti

- Le variabili contengono riferimenti ad oggetti
- Se eseguo `s1 == s2`, con `s1` e `s2` riferimenti ad oggetti
  - Verifico se `s1` e `s2` hanno lo stesso riferimento, cioè puntano allo stesso oggetto
  - Non verifico se gli oggetti puntati sono uguali
- Il metodo predefinito `equals()` (della classe `Object`) permette di verificare se due oggetti contengono gli stessi dati

```
s1.equals(s2)  // ritorna un boolean
```


Array

- Gli array sono oggetti, tuttavia non esiste il nome della classe
  - Un array va creato con new
  - La lunghezza dell'array è indicata dall'attributo public `length`

Riuso di classi

- Spesso si ha bisogno di classi simili
  - Riusare cioè classi esistenti per gestire attributi e metodi leggermente diversi
- Copiare la classe originaria e modificarne attributi o metodi non è pratico
  - Proliferazione di classi
  - Il programmatore deve fare tante attività
- Il riuso delle classi esistenti deve avvenire
  - Senza dover modificare codice esistente (e funzionante)
  - In modo semplice per i programmatore

Ereditarietà

- Attraverso l'ereditarietà è possibile
  - Definire una nuova classe indicando solo cosa ha in più rispetto ad una classe esistente
    - E' possibile aggiungere attributi e metodi nuovi
    - E' possibile modificare metodi esistenti
- Esempio
  - Una classe `Persona` ha nome e cognome (più vari metodi)
  - La classe `Studente` dovrebbe avere tutto ciò che `Persona` fornisce (attributi e metodi) ed inoltre nuovi attributi e metodi
  - `Studente` aggiunge `esami`, `voti`, etc.
  - La classe `Studente` eredita da `Persona`

```
public class Studente extends Persona { ... }
```

  - `Studente` è sottoclasse di `Persona`
  - `Persona` è superclasse di `Studente`

Ereditarietà

- La sottoclasse
  - Eredita tutti i metodi e gli attributi della superclasse e può usarli come se fossero definiti localmente
  - Aggiunge altri metodi
  - Può ridefinire i metodi della superclasse
  - Non può eliminare metodi o attributi della superclasse
- Esempio
  - La classe `Studente`
    - Può usare tutti i metodi della classe `Persona`, es. `setName`
    - Può aggiungere metodi, es. `media`

Ereditarietà

- Visibilità
  - Ciò che è private è visibile solo alla classe, non alla sottoclasse
  - Ciò che è public è visibile a tutti, anche alla sottoclasse
- Se voglio far vedere qualche metodo o attributo alle sottoclassi ma non a tutti
  - Uso `protected`