Classi e oggetti

Sandro Morasca

Università degli Studi dell'Insubria

Dipartimento di Scienze Teoriche e Applicate

Via Mazzini 5

22100 Como

sandro.morasca@uninsubria.it

Problema 1: Modificabilità del Software

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static

- Un insieme di numeri interi può essere rappresentato in molti modi diversi

```
public class InsiemeDiInteri
{
  public final static int CAPACITA = 10;
  public int n;
  public int elenco[] = new int [CAPACITA];
```

questo è solo un esempio …

NON mettere i dati public!

- Le applicazioni che usano l'insieme di interi vengono implementate in base alla rappresentazione

Problema 1: Modificabilità del Software

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public class GestioneTarghe
{
  public InsiemeDiInteri insiemeTarghe;
  ...
  public int cercaTarga(int numeroTarga)
  {
    int i;
    while (i < insiemeTarghe.n &&
           numeroTarga != insiemeTarghe.elenco[i] )
    {
      i++;
    }
    if (numeroTarga != insiemeTarghe.elenco[i] )
    { return -1; }
    else
    { return i; }
  }
  ...
}
```

Problema 1: Modificabilità del Software

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public class GestioneMatricole
{
  public InsiemeDiInteri insiemeMatricole;
  ...
  public int leggiMatricola() //legge numero dall'esterno
  {
    int numeroLetto;
    ...
    return numeroLetto;
  }
  public int cercaMatricola( int numeroMatricola) { ... }
  public void aggiungiMatricola()
  { int numero = leggiMatricola();
    int posizione = cercaMatricola(numero);
    if( posizione == -1 )
    { insiemeMatricole.elenco[insiemeMatricole.n] =
 numero; insiemeMatricole.n++; }
    ...
  }
  ...
}
```

Problema 1: Modificabilità del Software

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public class GestioneContiCorrenti
{
  public InsiemeDiInteri insiemeConti;
  ...
  public int cercaConto(int numeroConto)
  { ... }

  public void eliminaConto(int numeroConto)
  {
    int posizione = cercaConto(numeroConto);
    if (posizione != -1)
    {
      insiemeConti.elenco[posizione] = 0;
    }
  }
  ...
}
```

Problema 1: Modificabilità del Software

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static

- Che cosa succede se si cambiano solo i dettagli della struttura dati e si mantiene il significato inalterato?

```
public class NodoListaInteri
{
  public int dato;
  public NodoListaInteri next;
  ...
}

public class InsiemeDiInteri
{
  public NodoListaInteri primoNodo;
  ...
}
```

- Saranno inevitabili cambiamenti anche nelle altre applicazioni

Problema 1: Modificabilità del Software

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public class GestioneTarghe
{
  public InsiemeDiInteri insiemeTarghe;
  ...
  public NodoListaInteri cercaTarga(int numeroTarga)
  {
    NodoListaInteri rifNodo = insiemeTarghe.primoNodo;
    while ( rifNodo != null &&
            numeroTarga != rifNodo.dato )
    {
      rifNodo = rifNodo.next;
    }
    return rifNodo;
  }
  ...
}
```

Problema 1: Modificabilità del Software

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public class GestioneMatricole
{
  public InsiemeDiInteri insiemeMatricole;
  ...
  public int leggiMatricola() { }
  public NodoListaInteri cercaMatricola(
                    int numeroMatricola) { ... }
  public void aggiungiMatricola()
  {
    int numero = leggiMatricola();
    NodoListaInteri rifNodo = cercaMatricola(numero);
    if( rifNodo != null )
    {
      NodoListaInteri nuovoNodo =
                              new NodoListaInteri(numero);
      nuovoNodo.next = insiemeMatricole.primoNodo;
      insiemeMatricole.primoNodo = nuovoNodo;
    }
  }
  ...
}
```

Problema 1: Modificabilità del Software

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public class GestioneContiCorrenti
{
  public InsiemeDiInteri insiemeConti;
  ...
  public NodoListaInteri cercaConto(int numeroConto)
  { ... }

  public void eliminaConto(int numeroConto)
  {
    NodoListaInteri rifNodo = cercaConto(numeroConto);
    if (rifNodo != null)
    {
      <cerca il nodo precedente prec ...>
     prec.next = rifNodo.next;
    }
  }
  ...
}
```

Problema 1: Modificabilità del Software

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- `static`

- Sarà necessario cambiare
  - tutte le istruzioni che facevano riferimento alla vecchia struttura dati
  - solo le istruzioni che facevano riferimento alla vecchia struttura dati
- Problemi
  - tempo e sforzo mal spesi
  - possibilità di introdurre errori

Problema 2: Integrità dei dati

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- `static`

- Se più funzioni possono accedere ai dati liberamente, crescono le possibilità di avere problemi di correttezza
  - nell'**uso** delle informazioni
  - nella **modifica** delle informazioni

Problema 2: Integrità dei dati

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public class InsiemeDiInteri
{
  public final static int CAPACITA = 10;
  public int n;
  public int elenco[] = new int [CAPACITA];
}
```

Problema 2: Integrità dei dati

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public class GestioneContiCorrenti
{
   ...
   public int cercaConto(int numeroConto, InsiemeDiInteri
    insiemeConti)
   { ... }

   public void eliminaConto(int numeroConto,
    InsiemeDiInteri insiemeConti)
   {
     int posizione = cercaConto(numeroConto,
    insiemeConti);
     if (posizione != -1)
     {
       insiemeConti.elenco[posizione] = 0;
     }
   }
   ...
}
```

significato convenzionale di 0
per dire che un conto non esiste

Problema 2: Integrità dei dati

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public class GestioneContiSpeciali
{
  public int cercaConto(int numeroConto, InsiemeDiInteri
   insiemeConti)
  { ... }
  public void togliConto(int numero, InsiemeDiInteri
   insiemeConti)
  {
    int posizione;
    posizione = cercaConto(numero, insiemeConti);
    if (posizione != -1)
    {
      insiemeConti.n--;
      insiemeConti.elenco[posizione] =
insiemeConti.elenco[insiemeConti.n];
    }
  }
  ...
}
```

altro modo per eliminare

Problema 2: Integrità dei dati

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public class GestioneContiEsteri
{
  public int cercaConto(int numeroConto, InsiemeDiInteri
   insiemeConti)
  { ... }
  public void eliminazione(int numero, InsiemeDiInteri
   insiemeConti)
  {
    int posizione;
    posizione = ricerca(numero);
    if (posizione != -1)
    {
      insiemeConti.elenco[posizione] =
   insiemeConti.elenco[insiemeConti.n];
    }
  }
  ...
}
```

altro modo ancora per eliminare, non coerente con gli altri

Problema 2: Integrità dei dati

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public class GestioneConti
{
  InsiemeDiInteri insieme;
  GestioneContiCorrenti corrente;
  GestioneContiSpeciali speciale;
  GestioneContiEsteri estero;

  public static void main()
  {
    int numero;
    ...
    corrente.eliminaConto(numero, insieme);
    speciale.togliConto(numero, insieme);
    estero.eliminazione(numero, insieme);
}
```

uso inconsistente dei dati

Problema 3: Riuso del Software

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- `static`

- Tante rappresentazioni diverse …
  - … un unico tipo
- Un tipo di dato astratto
  - i metodi devono portare agli stessi risultati indipendentemente dalla rappresentazione scelta
- Un tipo di dato che può essere
  - riusato da tanti programmi senza modifiche

Soluzione

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- `static`

- Separare
  - il contenuto
    - l'insieme dei dati cui si vuole accedere
  - l'interfaccia
    - ovvero modalità di accesso ai dati di un tipo di dato astratto
- Ulteriori benefici, grazie al riuso :
  - maggiore affidabilità
  - minore costo
  - minore tempo di sviluppo

Esempio: Insieme di interi

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public class InsiemeDiInteri
{
  private final static int CAPACITA = 10;
  private int n;
  private int elenco[] = new int [CAPACITA];

  private int ricerca(int numero)
  {
    int i = 0;
    if ( eVuoto() )
    { return -1; }
    while ( (i < n -1) && (numero != elenco[i]) )
    { i++;}
    if ( numero != elenco[i] )
    { return -1; }
    else
    { return i; }
  }
```

Esempio: Insieme di interi

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public InsiemeDiInteri()
{
  n = 0;
}

public InsiemeDiInteri( InsiemeDiInteri altroInsieme )
{
  copia( altroInsieme );
}

public void finalize()
{
  n = 0;
}
```

Esempio: Insieme di interi

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public boolean eVuoto()
{
  return n == 0;
}

public boolean ePieno()
{
  return n == CAPACITA;
}

public int cardinalita()
{
  return n;
}

public boolean appartiene(int numero)
{
  return ricerca(numero) != -1;
}
```

Esempio: Insieme di interi

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public void inserimento(int numero)
{
  if (!ePieno() && (!appartiene(numero)))
  {
    elenco[n] = numero;
    n++;
  }
}

public void eliminazione(int numero)
{
  int posizione;
  posizione = ricerca(numero);
  if (posizione != -1)
  {
    n--;
    elenco[posizione] = elenco[n];
  }
}
```

Esempio: Insieme di interi

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public boolean contiene( InsiemeDiInteri altroInsieme )
{
  for (int i = 0; i < altroInsieme.n; i++)
  {
    if (!appartiene(altroInsieme.elenco[i]))
    {
      return false;
    }
  }
  return true;
}

public boolean equals( InsiemeDiInteri altroInsieme )
{
  return contiene( altroInsieme ) &&
altroInsieme.contiene( this );
}
```

Esempio: Insieme di interi

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public void insiemeVuoto()
{
  n = 0;
}

public void copia( InsiemeDiInteri altroInsieme )
{
  n = 0;
  for (int i = 0; i < altroInsieme.n; i++)
  {
    elenco[n] = altroInsieme.elenco[i];
    n++;
  }
}
```

Esempio: Insieme di interi

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
 public void unione( InsiemeDiInteri altroInsieme )
{
  for (int i = 0; i < altroInsieme.n; i++)
  {
    if (ePieno())
    { return; }
    inserimento(altroInsieme.elenco[i]);
  }
}

public void differenza( InsiemeDiInteri altroInsieme )
{
  for (int i = 0; i < altroInsieme.n; i++)
  {
    if (eVuoto())
    { return; }
    eliminazione(altroInsieme.elenco[i]);
  }
}
```

Esempio: Insieme di interi

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public void intersezione( InsiemeDiInteri altroInsieme )
 {
   int i = 0;

   while(i < n)
   {
     if (!altroInsieme.appartiene(elenco[i]))
     {
       eliminazione(elenco[i]);
     }
     else
     {
       i++;
     }
   }
 }
```

Esempio: Insieme di interi

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static


```
public String toString()
  {
    String risultato = "";

    for (int i = 0; i < n; i++)
    {
      risultato += (elenco[i] + "\n");
    }

    return risultato;
  }
}
```

Classi vs. oggetti

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- `static`

- Una classe definisce un tipo, ossia
  - l'insieme dei valori di dati
  - l'insieme delle operazioni ammesse su quei dati
  - la dimensione di quei dati

- Una classe fornisce servizi ai clienti della classe, ovvero modalità per la definizione e l'uso dei dati membro

- Un oggetto è una variabile il cui tipo è una classe

Istanziazione

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static

- In Java, la dichiarazione

  ```
  identificatore_classe identificatore
  ```

  è la dichiarazione di un **riferimento**, non di un oggetto

- Gli oggetti devono essere **creati**, es.

  ```
  identificatore = new identificatore_classe();
  ```

- Un oggetto è anche detto una istanza (= un esemplare) di una classe, anche se
  - non tutte le classi vengono poi effettivamente istanziate in oggetti
  - non tutte le classi possono essere istanziate in oggetti (classi astratte)

Elementi di classe e di istanza

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- `static`

- Classi e oggetti contengono come elementi
  - dati, detti dati membro
  - metodi, detti metodi membro
- Dati e metodi esterni a una classe si dicono non membro quando ci si riferisce alla classe o a un oggetto istanza della classe.
- Elementi (dati e metodi)
  - di istanza: ne esiste una copia indipendente per ogni oggetto istanza della classe
  - di classe (**`static`**): ne esiste una copia unica comune a tutti gli elementi della classe

Elementi pubblici di una classe/un oggetto

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- `static`

- Gli elementi pubblici di una classe/un oggetto sono
  - dati membro (**ASSOLUTAMENTE SCONSIGLIATO**)
  - metodi

  che sono visibili a tutte le classi e a tutti gli oggetti.

- Sono qualificati dalla parola chiave **`public`**
  - il meccanismo di incapsulamento serve per nascondere i dati e permettere di accedervi solo tramite le funzionalità dell'interfaccia

- La parte pubblica non dovrebbe perciò contenere dati membro

Elementi protetti di una classe/un oggetto

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- `static`

- Gli elementi protetti di una classe/un oggetto sono quelli a cui possono accedere soltanto i metodi
  - della classe/degli oggetti della classe
  - delle sue sottoclassi/degli oggetti delle sue sottoclassi
  - delle classi appartenenti allo stesso package della classe
- Sono qualificati dalla parola chiave **`protected`**
- A differenza degli elementi di una classe/un oggetto con visibilità limitata al package, viene esplicitata l'intenzione di rendere questi elementi riutilizzabili dalle sottoclassi che ereditano dalla classe

Elementi di una classe/un oggetto con visibilità limitata al package

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static

- Gli elementi di una classe/un oggetto con visibilità limitata al package sono
  - dati membro (**ASSOLUTAMENTE SCONSIGLIATO**)
  - metodi

  che sono visibili a tutte le classi appartenenti allo stesso package della classe e ai loro oggetti.

- Non sono qualificati da alcuna parola chiave
  - il meccanismo di incapsulamento serve per nascondere i dati e permettere di accedervi solo tramite le funzionalità dell'interfaccia

- La parte pubblica non dovrebbe perciò contenere dati membro

Elementi privati di una classe/un oggetto

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- `static`

- Gli elementi privati di una classe/un oggetto sono
  - dati membro: è meglio che i dati membro non siano pubblici
  - metodi

  che non sono visibili all'esterno della classe/dell'oggetto

- Sono qualificati dalla parola chiave **`private`**

Categorie di metodi membro

- Motivazioni
- Classi e oggetti
- Elementi
- **Metodi**
- Identità
- `static`

- Estensore (metodo "set")
  - Metodo membro non privato che modifica (ovvero definisce) il valore dei dati membro
- Selettore (metodo "get")
  - Metodo membro non privato che accede al valore (ovvero usa il valore) dei dati membro senza modificarli
- Metodo di servizio (metodo "utility")
  - Metodo membro privato
- Il tipo restituito da un metodo membro può essere qualsiasi, inclusa la classe stessa cui il metodo appartiene (caso molto frequente).

Costruttori

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static

- Metodi chiamati automaticamente durante l'esecuzione all'atto della creazione di un oggetto per inizializzare i dati membro dell'oggetto.

- Due tipi di costruttori:
  - costruttori con parametri "generici"
  - costruttori per default

Costruttori: sintassi

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static

- I costruttori sono metodi con lo stesso nome della classe, ma senza tipo ritornato (nemmeno void)

```
class identificatore_classe
{
   ...
   //Costruttore
   public identificatore_classe(lista_parametri_formali)
   {
   //Corpo del costruttore
   ...
   }
   ...
   //Altro costruttore
   public
      identificatore_classe(altra_lista_parametri_formali)
   {
   //Corpo del costruttore
   ...
   }
   ...
}
```

Chiamate di costruttori

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- `static`

- Il costruttore può essere attivato, solo una volta, all'atto della creazione di un nuovo oggetto
- I costruttori non possono essere chiamati esplicitamente dal programmatore su un oggetto dopo la creazione dell'oggetto
- I **dati membro** di una classe/un oggetto sono inizializzati automaticamente a un valore "neutro" in mancanza di un valore esplicito
  - ad esempio i dati membro `int` sono inizializzati a 0

Costruttore per default

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static

- Non ha parametri
- Chiamata di un costruttore per default

```
identificatore_classe oggetto;
puntatore_a_oggetto = new identificatore_classe();
```

- Se non viene dichiarato ***nessun*** costruttore per una classe, si può prendere quello della superclasse più vicina della gerarchia di ereditarietà che contiene il costruttore per default
  - eventualmente risalendo fino alla classe `Object`

Costruttore con parametri “generici”

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static

- Hanno parametri
- Possono essere sottoposti a overloading
- Chiamata di un costruttore con parametri "generici“

```
riferimento_a_oggetto = new
identificatore_classe(lista_parametri_effettivi);
```

Finalizzatore

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- `static`

- Metodo membro chiamato automaticamente durante l'esecuzione all'atto della distruzione di un oggetto dal garbage collector, che è il processo che permette di rilasciare la memoria allocata per oggetti quando gli oggetti non servono più.

- Esprime le "ultime volontà" dell'oggetto prima che venga distrutto.

- La distruzione di un oggetto avviene tipicamente quando
  - l'oggetto perde significato perché si è usciti dall'ambito in cui è dichiarato l'oggetto
  - si è eliminata la possibilità di accedere all'oggetto
    - per esempio dopo aver posto a **`null`** l'unico riferimento esistente per un oggetto

Finalizzatore: sintassi e semantica

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- `static`

- Il finalizzatore è un metodo membro void il cui identificatore è `finalize`
- Il finalizzatore viene chiamato dal garbage collector
  - il momento di esecuzione non viene controllato dal programmatore
- Il finalizzatore può essere chiamato esplicitamente dal programmatore per distruggere un oggetto

Finalizzatore: sintassi e semantica

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- `static`

- Se non viene dichiarato il finalizzatore in una classe, si prende quello della superclasse più vicina della gerarchia di ereditarietà che contiene il finalizzatore
- L'identificatore `finalize` non è una parola chiave del linguaggio
- Il finalizzatore deve essere scritto con molta attenzione per non causare danni al funzionamento del programma, soprattutto nel caso di strutture dati dinamiche

La parola chiave `this`

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static

- Ogni oggetto è raggiungibile tramite un riferimento, indicato convenzionalmente nel linguaggio Java dalla parola chiave **`this`** *quando si scrive la dichiarazione della classe*
  - Si usa una parola chiave in quanto nel momento in cui si dichiara una classe, ossia un tipo, non è possibile usare nomi di oggetti della classe, che non esistono ancora
- La parola chiave **`this`** indica un riferimento che contiene l'indirizzo dell'oggetto stesso cui appartengono
  - i dati membro che si stanno manipolando
  - i metodi membro che si stanno utilizzando

La parola chiave `this`

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static

- La parola chiave **`this`** è implicita per i dati membro e i metodi membro, ossia si può omettere quando si menzionino dati membro e metodi membro qualora non vi siano pericoli di ambiguità.
- Utilizzi tipici
  - al termine di un metodo, restituire il riferimento all'oggetto che si è manipolato: **`return this;`**
    - Ad esempio ciò permette di concatenare più chiamate di metodi;
  - all'interno di un metodo, per permettere di accedere a quei dati membro di una classe/un oggetto che sono nascosti dai dati locali del metodo che hanno lo stesso nome

Assegnamento e uguaglianza

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- `static`

- Come per gli array, gli operatori predefiniti di assegnamento e uguaglianza
  - effettuano l'assegnamento
  - controllano l'uguaglianza

  dei riferimenti (assegnamento e uguaglianza *superficiali*) e **non degli oggetti** (assegnamento e uguaglianza *profondi*).
- Definire comunque metodi in modo tale che sia possibile effettuare
  - l'assegnamento
  - l'uguaglianza (metodo `equals`)

  profondi

Assegnamento e uguaglianza

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- static

- Non è necessario definire un metodo apposito per l'assegnamento, ma si può utilizzare un costruttore "per copia“

```
Oggetto o1, o2;
o2 = new Oggetto( o1 );
```

con la chiamata di

```
public Oggetto( Oggetto altroOggetto )
```

che costruisce un oggetto per copia di `altroOggetto`

Dati e metodi membro `static`

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- **`static`**

- Sono dati e metodi membro di cui deve esistere una sola copia per una classe
- Sono qualificati come **`static`**

```
class identificatore_classe
{
   ...
   private static int numeroOggetti;
   ...
   private static int intero = 14;
   ...
   private static AltraClasse oggetto1;
   ...
   private static AltraClasse oggetto2 =
                                    new AltraClasse();
   ...
}
```

Dati e metodi membro `static`

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- **`static`**

- Dato membro
  - `numeroOggetti` inizializzato con 0
  - `intero` inizializzato con 14
  - `oggetto1` inizializzato con **`null`**
  - `oggetto2` inizializzato con il costruttore per default della classe `AltraClasse`
- I dati **`static`** vengono allocati indipendentemente dall'esistenza di oggetti della classe non appena inizia l'elaborazione

Dati e metodi membro `static`

- Motivazioni
- Classi e oggetti
- Elementi
- Metodi
- Identità
- `static`

- I metodi **`static`** possono essere chiamati con

  `identificatore_classe.identificatore_metodo_statico(lista_parametri_effettivi)`

  (**consigliato**)

  `identificatore_oggetto.identificatore_metodo_statico(lista_parametri_effettivi)`

  (**meglio no**)

- Dati e metodi **`static`** non fanno riferimento a nessun oggetto, per cui non possono utilizzare il riferimento **`this`**
- I metodi **`static`** possono utilizzare solo dati e metodi **`static`** della stessa classe (oltre naturalmente a metodi non **`static`** di oggetti di altre classi)