Dai TDA agli oggetti

Stefano Mizzaro

Dipartimento di matematica e informatica
Università di Udine
http://www.dimi.uniud.it/mizzaro/
mizzaro@uniud.it

Programmazione, lezione 14
21 aprile 2015

Riassunto

- "Mattoni"
- Sequenza, Selezione, Iterazione
- Array
- Sottoprogrammi (metodi)
- Ricorsione
- TDA

Riassunto TDA

- Astrazione sui tipi, non sulle istruzioni
  - "Aggiungo al Java i tipi (istruzioni) che non ha"
- Dichiarazione
  - Come è fatto (`class`, `private`)
  - Implementazione (attributi) + Operazioni (metodi)
- Uso
  - Allocazione (costruttore, `new`)
  - Invocazione (chiamata metodi, notazione puntata)
- `this`, `toString`
- Occultamento delle informazioni
  - Incapsulamento = unitarietà + inaccessibilità
  - >Comprensibilità, modificabilità, riusabilità

Oggi

- Fine TDA
  - Esempio / riassunto
  - Interazione fra TDA
- Da TDA a OO: lo scambio messaggi
  - Concetto ed Esempi
- Il `this` rivisitato
- Il `toString` rivisitato
- Funzionale, procedurale, OO
- Altri esempi

Esempio / riassunto

- 1. Tipo Strutturato (TS)
- 2. Tipo Strutturato e Metodi (TSM)
  - Più comodo con i metodi ma no info hiding
- 3. Tipo Strutturato e Private (TSP)
  - Info hiding ma non compila
- 4. Tipo Strutturato e Metodi e Private (TSMP)
  - Info hiding e metodi comodi ma non compila
- 5. Tipo di Dato Astratto (TDA)
- Implementiamo le classi minimali ~Punto e ~UsaPunto nei 5 casi

1. Tipo Strutturato (TS)

```
class PuntoTS {
  int x;
  int y;
}
```

```
class UsaPuntoTS {
  public static void main(String[] args) {
    PuntoTS p = new PuntoTS();
    p.x = 5;
    p.y = 10;
    System.out.println(
                "Punto: (" + p.x + ", " + p.y + ")");
  }
}
```

```
>javac UsaPuntoTS.java
>java UsaPuntoTS
Punto: (5, 10)
>
```

2. Tipo Strutturato e Metodi (TSM)

```
class PuntoTS {
  int x;
  int y;
}
```

```
class UsaPuntoTSM {
  public static void main(String[] args) {
    PuntoTS p = new PuntoTS();
    set(p,5,10);
    print(p);
  }

  static void set(PuntoTS p, int x, int y) {
    p.x = x;
    p.y = y;
  }
  static void print(PuntoTS p) {
    System.out.println("Punto: (" + p.x +", " + p.y + ")");
  }
}
```

```
>javac UsaPuntoTSM.java
>java UsaPuntoTSM
Punto: (5, 10)
>
```


3. Tipo Strutturato e Private (TSP)

```
class PuntoTSP {
  private int x;
  private int y;
}
```

```
class UsaPuntoTSP {
  public static void main(String[] args) {
    PuntoTSP p = new PuntoTSP();
    p.x = 5;
    p.y = 10;
    System.out.println("Punto: (" + p.x+ ", " + p.y + ")");
  }
}
```

```
>javac UsaPuntoTSP.java
UsaPuntoTSP.java:4: error: x has private access in
PuntoTSP
    p.x = 5;
      ^
UsaPuntoTSP.java:5: error: y has private access in
PuntoTSP
...
```

4. Tipo Strutturato e Metodi e Private (TSMP)

```
class PuntoTSP {
  private int x;
  private int y;
}
```

```
class UsaPuntoTSMP {
  public static void main(String[] args) {
    PuntoTSP p = new PuntoTSP();
    set(p,5,10);
    print(p);
  }
  static void set(PuntoTSP p, int x, int y) {
    p.x = x;
    p.y = y;
  }
  static void print(PuntoTSP p) {
    System.out.println("Punto: (" + p.x +", " + p.y + ")");
  }
}
```

```
>javac UsaPuntoTSMP.java
UsaPuntoTSMP.java:9: error: x has private access in PuntoTSP
    p.x = x;
      ^
...
```

5. Tipo di Dato Astratto (TDA)

```
class PuntoTDA {
  private int x;
  private int y;

  static void set(PuntoTDA p, int x, int y) {
    p.x = x; p.y = y;
  }

  static void print(PuntoTDA p) {
    System.out.println("Punto: (" + p.x +", " + p.y + ")");
  }
}
```

```
class UsaPuntoTDA {
  public static void main(String[] args) {
    PuntoTDA p = new PuntoTDA();
    PuntoTDA.set(p,5,10);
    PuntoTDA.print(p);
  }
}
```

```
>javac UsaPuntoTDA.java
>java UsaPuntoTDA
Punto: (5, 10)
```


Interazione fra TDA

- Finora:
  - un unico TDA
  - una classe che lo usa
- In generale:
  - più TDA
  - che si usano a vicenda
  - una classe con il main

Esempio

- Programma di "grafica"
  - Ultrasemplificato
  - Solo punti e cerchi
  - Solo rappresentazione
  - Visualizzazione solo testuale
- Classe `Punto`: rappresenta punti
- Classe `Cerchio`: rappresenta cerchi
- Classe `Grafica`: con main

Punto.java

```
class Punto {
  private double x; private double y;
  Punto() {this(0,0);}
  Punto(double xCoord, double yCoord) {
    x = xCoord; y = yCoord;
  }
  static void set(Punto p, double xC, double yC){
    p.x = xC; p.y = yC;
  }
  static void setX(Punto p, double xCoord) {
    p.x = xCoord;
  }
  static void setY(Punto p, double yCoord) {
    p.y = yCoord;
  }
  static double getX(Punto p) {return p.x;}
  static double getY(Punto p) {return p.y;}
  static String toString(Punto p) {
    return "(" + p.x + ", " + p.y + ")";
  }
}
```


Cerchio.java

```
class Cerchio {
  private Punto centro; private double raggio;
  Cerchio() {this(new Punto(), 0);}
  Cerchio(double x, double y, double r) {
    this(new Punto(x,y), r);
  }
  Cerchio(Punto c, double r){centro = c; raggio = r;}
  static void setCentro(Cerchio c, Punto p) {
    c.centro = p;
  }
  static void setRaggio(Cerchio c, double r) {
    c.raggio = r;
  }
  static Punto getCentro(Cerchio c) {
    return c.centro;
  }
  static double getRaggio(Cerchio c) {
    return c.raggio;
  }
  static String toString(Cerchio c) {
   return "cerchio: centro "+Punto.toString(c.centro)
          +" raggio " + c.raggio;
  }
}
```

Grafica.java

```
/** Programma per la
 * prova di cerchi e punti */
class Grafica {
  public static void main (String[] args) {
    Punto p1 = new Punto();
    Punto p2 = new Punto(1,1);
    Cerchio c1 = new Cerchio();
    System.out.println(Cerchio.toString(c1));
    Cerchio c2 = new Cerchio(p1,0);
    System.out.println(Cerchio.toString(c2));
    Cerchio c3 = new Cerchio(1,1,1); // o (p2,1)
    System.out.println(Cerchio.toString(c3));
  }
}
```


Osservazioni

- Sovraccarico costruttori
  - Più comodo creare istanze
- Cosa succede quando viene invocato il costruttore (uno dei primi 2) di `Cerchio`?
  - Viene invocato anche il costruttore di `Punto`
- La relazione fra `Cerchio` e `Punto` è diversa da quella con `Grafica`:
  - Uso
  - Composizione (una parte di un cerchio è un punto)

Struttura programma Java?

```
class ... {
  static <tipo> <id> (<parametri>) {
    ...
  }
  static <tipo> <id> (<parametri>) {
    ...
  }
  public static void main (String[] args) {
    ...
  }
  static <tipo> <id> (<parametri>) {
    ...
  }
  static <tipo> <id> (<parametri>) {
    ...
  }
}
```

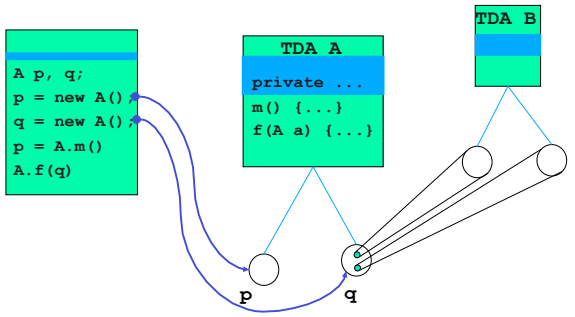

Programma con TDA (1/2)

```
class C {
  ... main ... {
    A p, q;
    p = new A();
    q = new A();
    A.m();
    p = A.f(q);
  }
}
```

```
class A {
  ... private ...
  ... static ...
  ... void  m() {...}
  ... A f(A a) {...}
}
```

```
class C {
class B {
  ... private ...
  ... static ...
  ... void  m() {...}
  ... B f(B b) {...}
}
```

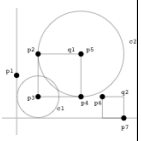
Programma con TDA (2/2)



Vantaggi TDA

- Scomposizione del problema
  - Avere un tipo in più rende più semplice la scrittura del programma che risolve un problema. Divide et impera
- Riuso
  - Una volta creato, il TDA sarà probabilmente riusabile in altre situazioni (non vero per un metodo "da solo")
- Librerie (da "software libraries", eh…)
  - Mica detto che il TDA lo dobbiamo creare noi!
  - Possiamo trovarlo, comprarlo, farlo fare su commissione da qualcun altro, ecc. ecc.
  - Nelle librerie (API) del Java ci sono migliaia (!) di TDA pronti per l'uso

Esercizi sull'esempio



- Aggiungere `Quadrato.java` (e modificare `Grafica.java`)
- Modificare `Grafica.java` x "disegnare" -->
- Confrontare il codice sui lucidi con quello sul testo. Osservare come l'uso del `this` evita duplicazioni
- Cercare e invocare/collaudare versioni alternative dei costruttori
- Modificare i nomi dei parametri formali per
  - Renderli uguali agli attributi
  - Usare il `this`
- Disegnare una figura con classi e istanze

Quali TDA? Esempi

- Programma di grafica
  - Punti, cerchi, segmenti, …
- Programma per pilota automatico d'aereo
  - Piano di volo, tratta, quota, coordinata, …
- Programma di gestione di un magazzino
  - Magazzino, merce, scaffale, ordine, …
- Programma per robot in un labirinto
  - Robot, griglia, spostamento, traiettoria, …
- N.B. Eh, non è facile. L'esperienza aiuta. E ci sono tecniche (vedrete)

Scaletta

- Interazione fra TDA
  - (Fine cap. 7)
- Da TDA a OO: lo scambio messaggi
  - Concetto ed Esempi
- Il `this` rivisitato
- Il `toString` rivisitato
- Funzionale, procedurale, OO
- Altri esempi

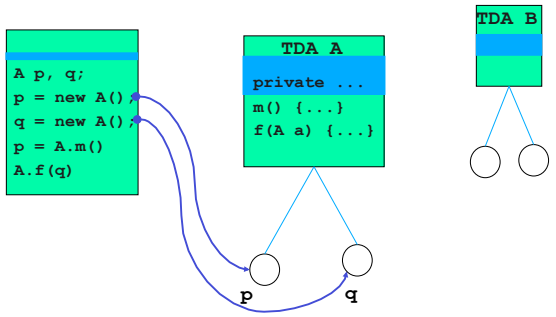
OO è…

- (O-O, Object Oriented, Orientato agli Oggetti)
- OO è:
  - TDA +
  - Scambio messaggi +
  - Eredità +
  - Polimorfismo

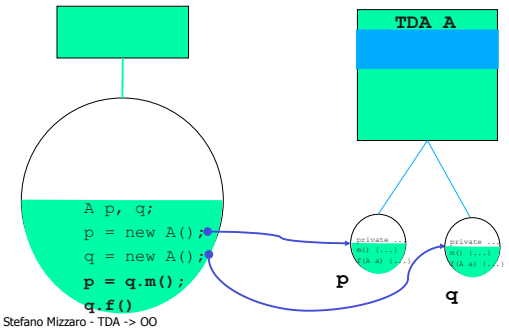
Scambio messaggi

- Da TDA e funzioni/procedure definite nel TDA a oggetti attivi che si scambiano messaggi
- Da "esegui il metodo **`getX()`** della classe **`Punto`** con argomento **`q`**":
  - `double a = Punto.getX(q);`
- a "manda il messaggio **`getX`** all'oggetto **`q`**":
  - `double a = q.getX();`
- "Approccio TDA":
  - variabili (passive) contengono stato e basta
  - i metodi sono nella classe
- "Approccio OO":
  - Oggetti/istanze (attive) contengono stato e metodi

Programma con TDA (di nuovo)



Programma con scambio messaggi



`Punto.java` (TDA, non a oggetti)

```
class Punto {
  private double x; private double y;
  ...
  static void set(Punto p, double x, double y){
    p.x = x; p.y = y;
  }
  static void setX(Punto p, double x) {
    p.x = x;
  }
  static void setY(Punto p, double y) {
    p.y = y;
  }
  static double getX(Punto p) {return p.x;}
  static double getY(Punto p) {return p.y;}
  ...
}
```


`UsaPunto.java` (TDA, non a oggetti)

```
/** Programma per la prova di cerchi e punti */
class UsaPunto {
  public static void main (String[] args) {
    Punto p1 = new Punto();
    Punto p2 = new Punto(1,1);
    Punto.setX(p1,5);
    System.out.println(Punto.getX(p2));
    ...
  }
}
```


La classe `Punto` "a oggetti"

```
class Punto {
  private double x;
  private double y;
  public Punto() {this(0,0);}
  public Punto(double x, double y){
    this.x = x;
    this.y = y;
  }
  public void setX(double x) {this.x = x;}
  public void setY(double y) {this.y = y;}
  public double getX() {return x;}
  public double getY() {return y;}
}
```

Programma che usa `Punto`

```
class UsaPunto {
  public static void main(String[]args) {
    Punto p;
    p = new Punto(12.0, 34.9);
    Punto q = new Punto(0, 0);
    p = new Punto(2.3, 3.4);
    p.setX(2.5);
    double x = p.getX();
    p = q;  // Alias
  }
}
```


Confronto: TDA vs. OO

- Filosoficamente:
  - variabili passive vs. oggetti attivi
- In pratica: metodi d'istanza e un parametro in meno
  - `public static void setX(Punto p, double x)` vs. `public void setX(double x)`
  - `Punto.getX(q)` vs. `q.getX()`
- Non c'è più lo `static`
- Terminologia:
  - Attributi
  - Metodi

Cosa fa il main

- Inizia l'esecuzione
- Crea oggetti/istanze
- Manda messaggi a oggetti/istanze (invoca metodi d'istanza)
- Invoca direttamente metodi `static`

Altro esempio

- Di nuovo il TDA Ora,
- Però a oggetti

Ora / TDA

```
class Ora {
  private int secondi;
  static void impostaOra(Ora x, int o, int m, int s) {
    x.secondi = o*3600+m*60+s;
  }
  static int getOre(Ora x) {return x.secondi/3600;}
  static int getMinuti(Ora x) {
    return (x.secondi%3600)/60;
  }
  static int getSecondi(Ora x) {return x.secondi%60;}
}
```

```
class UsaOra {
  public static void main(String[] args) {
    ...
    Ora h = new Ora();
    Ora.impostaOra(h,12,22,31);
    System.out.println("Sono le ore " +Ora.getOre(h));
    ...
  }
}
```

Ora / OO

```
class Ora {
  private int secondi;
  void impostaOra(int o, int m, int s) {
    secondi = o*3600+m*60+s;
  }
  int getOre() {return secondi/3600;}
  int getMinuti() {
    return (secondi%3600)/60;
  }
  int getSecondi() {return secondi%60;}
}
```

```
class UsaOra {
  public static void main(String[] args) {
    ...
    Ora h = new Ora();
    h.impostaOra(12,22,31);
    System.out.println("Sono le ore" +h.getOre());
    ...
  }
}
```

Altro esempio

- Di nuovo la pila di interi limitata
- Però "a oggetti"
- (Ex:
  - Coordinate
  - FrecciaOrizzontale
-)

Pila.java

```
class Pila {
  private static final int MAX = 10;
  private int[] elementi;
  private int numElementi;
  Pila() {
    numElementi = 0;
    elementi = new int[MAX];
  }
  boolean vuota() {return numElementi == 0;}
  boolean piena() {return numElementi == MAX;}
  void push(int e) {
    if (!piena()) elementi[numElementi++] = e;
  }
  int top() {
    if (!vuota()) return elementi[numElementi-1];
    else return -1;
  }
  void pop() {if (!vuota()) numElementi--;}
}
```

UsaPila.java

```
class UsaPila {
  public static void main(String[] args) {
    Pila p = new Pila ();
    while (!p.piena()) {
      System.out.print("Inserisci un elemento:");
      p.push(Leggi.unInt());
    }
    while (!p.vuota()) {
      System.out.println(p.top());
      p.pop();
    }
  }
}
```


Confronto con la versione TDA

- Il TDA
  - Parametro in meno
  - Senza **`static`**
- Uso
  - Parametro in meno
  - Nome istanza anziché nome classe
- Filosoficamente: si parla alle istanze
- Si scrive di meno (parametri, nome della classe)

TDA vs. OO

- TDA: classi e variabili
  - Variabili = implementazione (com'è fatto)
  - Classi forniscono metodi per lavorare sulle variabili
  - I metodi sono di classe, si parla alle classi
- OO: classi e istanze
  - Istanza = implementazione + operazioni
  - Le operazioni sono a livello di ogni istanza
  - I metodi sono d'istanza, si parla alle istanze

Scaletta

- Interazione fra TDA
- Da TDA a OO: lo scambio messaggi
  - Concetto ed Esempi
- Il **`this`** rivisitato
- Il **`toString`** rivisitato
- Funzionale, procedurale, OO
- Altri esempi

Il `this` rivisitato

- Ora si capisce: "this" = "questa istanza", "questo oggetto"
  - Gli attributi di questo oggetto (**`this.`**)
  - Il costruttore di questo oggetto (**`this()`**)
- Si potrebbe premettere sempre
  - a invocazioni di altri metodi dell'oggetto ("mando un messaggio a me stesso")
  - all'accesso agli attributi dell'oggetto
- … ma è implicito

`Pila.java` col `this`…

```
class Pila {
  ...
  Pila() {
    this.numElementi = 0;
    this.elementi = new int[MAX];
  }
  boolean vuota() {return this.numElementi == 0;}
  boolean piena() {return this.numElementi==MAX;}
  void push(int e) {
    if (!this.piena())
      this.elementi[this.numElementi++] = e;
  }
  int top() {
    if (!this.vuota()) return
      this.elementi[this.numElementi-1];
    else return -1;
  }
  void pop() {
    if (!this.vuota()) this.numElementi--;}
}
```

Il `toString` rivisitato

- Avevamo visto che se un TDA C ha il metodo **`toString`**, per visualizzare si fa:
  - **`System.out.println(C.toString(x))`**
- Ma ora sappiamo che **`toString`** può/deve essere d'istanza, non **`static`** => Lo si invocherà così:
  - **`System.out.println(x.toString())`**
  - È più comodo!
- Di più: l'invocazione è automatica!!
  - **`System.out.println(x)`**
- Se una classe non ha il **`toString`** ce n'è uno di default, ma non molto utile… (fare una prova!)
- Ex.: aggiungere il **`toString`** alle classi viste

Scaletta

- Interazione fra TDA
- Da TDA a OO: lo scambio messaggi
  - Concetto ed Esempi
- Il `this` rivisitato
- Il `toString` rivisitato
- Funzionale, procedurale, OO
- Altri esempi

Funzionale, procedurale, OO (1/2)

- Funzionale:

```
public static Pila push (Pila p, int e) {
  if (!piena(p))
    p.elementi[p.numElementi++] = e;
  return p;
}
...
p1 = Pila.push(p1,5);
```

- Procedurale:

```
public static void push (PilaP p, int e){
  if (!piena(p))
    p.elementi[p.numElementi++] = e;
}
...
Pila.push(p1,5);
```

- OO:

```
public void push (int e) {
  if (!piena())
    elementi[numElementi++] = e;
}
...
p1.push(5);
```


Funzionale, procedurale, OO (2/2)

- Programma funzionale in esecuzione =
  - funzioni che si chiamano a vicenda passandosi come parametri gli argomenti e restituiscono valori al chiamante
- Programma procedurale in esecuzione =
  - procedure che si chiamano a vicenda passandosi come parametri sia gli argomenti sia le variabili in cui memorizzare i risultati
- Programma OO in esecuzione =
  - oggetti che si scambiano messaggi, eventualmente passandosi come parametri altri oggetti

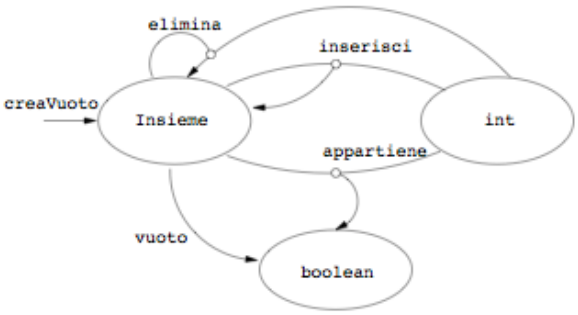
Di classe e d'istanza

	Attributi	Metodi
Di classe (`static`)	Informazioni sulla classe. Costanti	Approccio TDA
D'istanza	Informazioni sulle istanze	Approccio OO

Scaletta

- Interazione fra TDA
  - (Fine cap. 7)
- Da TDA a OO: lo scambio messaggi
  - Concetto ed Esempi
- Il `this` rivisitato
- Il `toString` rivisitato
- Funzionale, procedurale, OO
- Altri esempi

Esempio: Il TDA Insieme Segnatura



Esempio: Il TDA Insieme

- Rappresentazione: array di booleani, con valore `true` in posizione `i` se `i` appartiene all'insieme
- Pro: appartenenza, inserimento, eliminazione immediati
- Contro: Valore massimo limitato
- Versione OO

Insieme.java

```
class Insieme {
  private int maxElementi;
  private boolean[] elementi;
  /** Costruttore */
  public Insieme(int num) {
    maxElementi = num;
    elementi = new boolean[maxElementi];
  }
  /** Costruttore copia: costruisce (e restituisce) un
   * insieme uguale a quello passato come parametro */
  public Insieme(Insieme x) {
    maxElementi = x.maxElementi;
    elementi = new boolean[maxElementi];
    for (int i = 0; i < maxElementi; i++)
      elementi[i] = x.elementi[i];
  }
  /** Dice se l'insieme x e' vuoto */
  public boolean vuoto() {
    boolean trovato = false;
    for(int i = 0; i < maxElementi; i++)
      if (elementi[i]) {
        trovato = true;
        break;
      }
    return !trovato;
  }
```



```
  /** Inserisce l'elemento e */
  public void inserisci(int e) {
    elementi[e] = true;
  }
  /** Elimina l'elemento e */
  public void elimina(int e) {
    elementi[e] = false;
  }
  /** Predicato di appartenenza */
  public boolean appartiene(int e) {
    return elementi[e];
  }
  /** Restituisce una stringa del tipo {...} */
  public String toString() {
    String str = "";
    boolean primo = true;
    str = str + "{";
    for (int i = 0; i < maxElementi; i++) {
      if (elementi[i]) {
        if (primo) primo = false;
        else str = str + ",";
        str = str + i;
      }
    }
    str = str + "}";
    return str;
  }
}
```

```
class ProvaInsieme {
  // solo per testare l'implementazione
  public static void main(String[] args) {
    Insieme a = new Insieme(100);
    System.out.println(a.vuoto());
    a.inserisci(13);
    a.inserisci(11);
    System.out.println(a);
    Insieme b = new Insieme(a);
    System.out.println(a.appartiene(13));
    System.out.println(b.appartiene(11));
    System.out.println(a.appartiene(17));
    System.out.println(a.vuoto());
    a.elimina(11);
    System.out.println(a);
    a.elimina(13);
    System.out.println(a);
    System.out.println(b);
    System.out.println(a.appartiene(11));
    System.out.println(a.vuoto());
  }
}
```

Esercizi

- **`Punto`, `Cerchio`** e **`Quadrato`** con approccio OO
  - Usando **`this`** e **`toString`**
  - POI confrontare con il codice sul testo
- **`Insieme`** con approccio TDA e OO
  - Usando **`this`** e **`toString`**
  - POI confrontare con il codice sul testo
- "Array estendibile" (di, ad es., **`int`**)
- (fine cap. 8)

Riassunto

- Interazione fra TDA
- Da TDA a OO: lo scambio messaggi
  - Concetto ed Esempi
- Il **`this`** rivisitato
- Il **`toString`** rivisitato
- Funzionale, procedurale, OO
- Altri esempi

Prossima lezione

- Altri 2 ingredienti dell'OO:
  - Ereditarietà
  - Polimorfismo
- (Poi, OO in Java:
  - Varianti
  - Librerie (API)
-)

Credits