



## IN VIAGGIO DA OCAML A JAVA

### OCAML



- ✎ Programmazione I + Programmazione II = caratteristiche importanti di un linguaggio funzionale e i fondamenti della programmazione
- ✎ Caratteristiche di OCAML non considerate
  - La "O" in Ocaml: la parte Object-Oriented del linguaggio (simila a quella presente in Java, C#, C++)
  - La struttura complessiva del sistema dei moduli (ad esempio i funtori)
  - Il sistema di inferenza di tipo e tutti gli aspetti legati all'analisi statica di programmi

## Value-oriented programming



- ✦ Aspetto chiave che abbiamo considerato:  
programmare con strutture dati immutabili via  
pattern matching e ricorsione
- ✦ ML e i suoi fratelli (LISP, SCHEME, HASKELL) hanno  
avuto un impatto significativo nello sviluppo dei  
linguaggi di programmazione
  - funzioni anonime recentemente introdotte in Java e C#,
  - Linguaggi ibridi (Scala)
  - ma molte altre cose le scopriremo nella seconda parte del  
corso

## Verificare vuotezza di un albero



```

O
C
A
M
L
type 'a tree =
| Empty
| Node of ('a tree) * 'a * ('a tree)

let is_empty (t:'a tree) =
  begin match t with
  | Empty -> true
  | Node(_,_,_) -> false
  end

let t : int tree = Node(Empty,3,Empty)
let ans : bool = is_empty t
  
```

## La versione di Java



```
interface Tree<A> {
    public boolean isEmpty();
}
```

```
class Empty<A> implements Tree<A> {
    public boolean isEmpty() { return true;
    }
}
```

```
class Node<A> implements Tree<A> {
    private final A v;
    private final Tree<A> lt;
    private final Tree<A> rt;
}
```

```
Node(Tree<A> lt, A v, Tree<A> rt) {
    this.lt = lt; this.rt = rt; this.v = v;
}
```

```
public boolean isEmpty() { return false;
}
}
```

```
class mainProgram {
    public static void main(String[] args) {
        Tree<Integer> t =
            new Node<Integer>(new Empty<Integer>( ),
                3, new Empty<Integer>( ));
        boolean ans = t.isEmpty();
    }
}
```

## E' una provocazione



### Java offre tantissime cose utile

- Usato a livello industriale
- Librerie vastissime
- Complicato ma necessariamente complicato

### Obiettivo di Programmazione II

- Presentare le caratteristiche essenziali della programmazione Object-Oriented
- Come le tecniche OO aiutano nella soluzione di problemi
- Sperimentare con Java

## Oggetti e classi



- ✦  *Oggetto*= insieme strutturato di *variabili di istanza* (stato) e *metodi* (operazioni)
- ✦  *Classe*: *modello (template)* per la creazione di oggetti

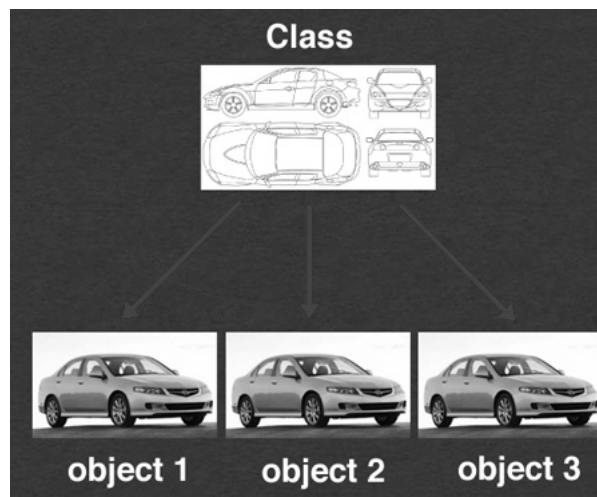
|                   |
|-------------------|
| OBJECT            |
| STATO (NASCOSTO)  |
| METODI (PUBBLICO) |

## Oggetti e classi

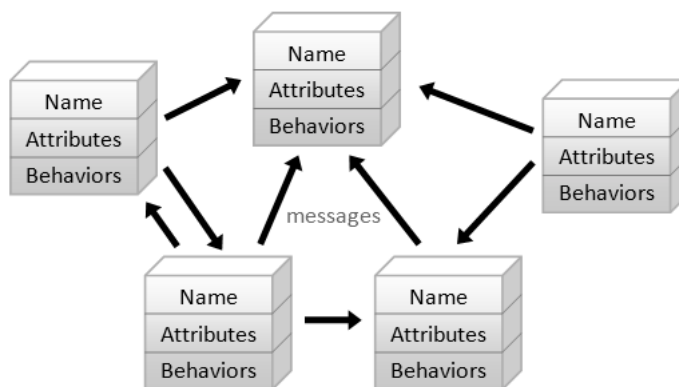


- ✦ La definizione di una classe specifica
  - Tipo e valori iniziali dello stato locale degli oggetti (le variabili di istanza)
  - Insieme delle operazioni che possono essere eseguite (metodi)
  - Costruttori (uno o piu'): codice che deve essere eseguito al momento della creazione di un oggetto
- ✦ Ogni oggetto e' una istanza di una classe e puo' (opzionalmente) implementare una interfaccia

## Oggetti e classi



## Oggetti e classi



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

## Un primo esempio



```
public class Counter { //nome della classe

private int cnt; // lo stato locale

// metodo costruttore
public Counter () { cnt = 0;
}
// metodo
public int inc () { cnt = cnt + 1; return cnt;
}
// metodo
public int dec () { cnt = cnt - 1; return cnt;
}
}
```

### DICHIARAZIONE DI CLASSE

public = visibile fuori  
dell'oggetto

private = visibile solo  
all'interno dell'oggetto

## Esecuzione di Java



un programma Java viene mandato in esecuzione  
Invocando un metodo speciale chiamato il  
metodo main.

```
public class Main {
public static void main(String[] args) {
Counter c = new Counter();
System.out.println(c.inc());
System.out.println(c.dec());}
}
```

## Compilare e eseguire



```
prompt$ javac Counter.java
```

**Viene creato il bytecode Counter.class**

```
prompt$ javac Main.java
```

**Viene creato il bytecode del Main Main.class**

```
prompt$ java Main
```

```
1
```

```
0
```

```
prompt$
```

## Cosa e' il bytecode



- ✎ Java bytecode e' il linguaggio della java virtual machine
- ✎ Load & store (e.g. aload\_0, istore)
- ✎ Arithmetic & logic (e.g. ladd, fcmpl)
- ✎ Object creation & manipulation (new, putfield)
- ✎ Operand stack management (e.g. swap, dup2)
- ✎ Control transfer (e.g. ifeq, goto)
- ✎ Method invocation & return (e.g. invokespecial, areturn)

## Riusciamo a vederlo



- 👁 Il comando javap
- 👁 Demo!!

## Creare oggetti



Dichiarare una variabile di tipo Counter  
Invocare il costruttore per creare l'oggetto di tipo Counter

```
Counter c;  
c = new Counter();
```

Soluzione alternativa: fare tutto in un passo!!

```
Counter c = new Counter();
```



## Costruttori con parametri



```
public class Counter { //nome della classe

private int cnt; // lo stato locale

// metodo costruttore
public Counter (int v0) { cnt = v0;
}
// metodo
public int inc () { cnt = cnt + 1; return cnt;
}
// metodo
public int dec () { cnt = cnt - 1; return cnt;
}
}
```

### DICHIARAZIONE DI CLASSE

public = visibile fuori  
dell'oggetto

private = visibile solo  
all'interno dell'oggetto

## Costruttori con parametri



```
public class Main {
public static void main(String[] args) {
Counter c = new Counter(25);
System.out.println(c.inc());
System.out.println(c.dec());}
}
```

## Strutture mutabili



Ogni variabile di oggetto in Java denota una entita' mutabile

```
Counter C;
C = new Counter(4);
C = new Counter(5);
C.inc();

//quale e' il valore dello stato locale?
```

## Valori NULL



Il valore **null** è generico e può essere assegnato a qualunque variabile di tipo riferimento.

Restituisce un oggetto di tipo Counter  
o **null** se non lo trova

```
Counter c = cercaContatore();
if (C == null)
    System.out.println("contatore non trovato");
```

Attenzione: come in C,  
= **singolo**: assegnamento  
== **doppio**: test di uguaglianza

Nello heap vengono allocate

- variabili di istanza, quando si crea un oggetto
- variabili statiche (o di classe), quando viene caricata una classe



Le variabili allocate nello heap, vengono inizializzate dal sistema,

- con **0** (zero), per le variabili di tipi numerici
- con **false** per le variabili di tipo **boolean**
- con **null** per le variabili di tipo riferimento

Le variabili dichiarate localmente in metodi/costruttori non vengono inizializzate per default: bisogna assegnargli un valore prima di leggerle.

## Stato locale



- ✎ **Modificatori: meccanismo per controllare l'accesso allo stato locale dell'oggetto**
  - **Public:** visibile e accessibile da ogni parte del programma
  - **Private:** visibile e accessibile solo all'interno della classe
- ✎ **Design Pattern (suggerimento grossolano)**
  - Tutte le variabili di istanza: private
  - Costruttori e metodi: public

## Riassunto



Il “frammento imperativo” di Java è molto simile al C

```
int x = 3; // dichiara x e lo inizializza al valore 3
```

```
int y; // dichiara y e gli viene assegnato il valore di default 0
```

```
y=x+3; // assegna a y il valore di x incrementato di 3
```

```
// dichiara un oggetto C di tipo Counter e lo inizializza con  
// il costruttore
```

```
Counter c = new Counter();
```

```
Counter d; // dichiara d e il suo valore di default è null
```

```
d = c; // assegna a d l'oggetto denotato da c => Aliasing!
```

## Comandi condizionali



```
if (cond) {  
    stmt1;  
    stmt2;  
    stmt3;  
}
```

```
if (cond) {  
    stmt1;  
    stmt2;  
} else {  
    stmt3;  
    stmt4;  
}
```

## Cicli



```
while (expression){ // your code goes here}

do { statement(s)} while (expression);

for (initialization; termination; increment) { statement(s)}
```

## demo



```
class WhileDemo {
    public static void main(String[] args){
        int count = 1;
        while (count < 11){
            System.out.println("Count is: " + count);
            count++;
        }
    }
}

class ForDemo {
    public static void main(String[] args){
        for(int i=1; i<11; i++){
            System.out.println("Count is: " + i);
        }
    }
}
```

```
class BreakDemo {
    public static void main(String[] args) {

        int[] arrayOfInts =
            { 32, 87, 3, 589,
              12, 1076, 2000,
              8, 622, 127 };
        int searchfor = 12;

        int i;
        boolean foundIt = false;

        for (i = 0; i < arrayOfInts.length; i++) {
            if (arrayOfInts[i] == searchfor) {
                foundIt = true;
                break;
            }
        }

        if (foundIt) {
            System.out.println("Found " + searchfor + " at index " + i);
        } else {
            System.out.println(searchfor + " not in the array");
        }
    }
}
```



## Tipi primitivi

**int** // *standard integers*

**byte, short, long** // *other flavors of integers*

**char** **float**, // *unicode characters*

**double** // *floating-point numbers*

**boolean** // *true and false*

***String non sono tipi primitivi!!!***





| OCaml     | Java      | description         |
|-----------|-----------|---------------------|
| =         | ==        | equality test       |
| <> !=     | !=        | inequality          |
| < <= > >= | < <= > >= | comparisons         |
| +         | +         | addition            |
| -         | -         | subtraction         |
| /         | /         | division            |
| *         | *         | multiplication      |
| mod       | %         | remainder (modulus) |
| not       | !         | logical "not"       |
| &&        | &&        | logical "and"       |
|           |           | logical "or"        |

| Built-In Types |         |
|----------------|---------|
| int            | double  |
| long           | String  |
| char           | boolean |

| System               |
|----------------------|
| System.out.println() |
| System.out.print()   |
| System.out.printf()  |

| Math Library |            |
|--------------|------------|
| Math.sin()   | Math.cos() |
| Math.log()   | Math.exp() |
| Math.sqrt()  | Math.pow() |
| Math.min()   | Math.max() |
| Math.abs()   | Math.PI    |

| Flow Control |       |
|--------------|-------|
| if           | else  |
| for          | while |

| Parsing              |
|----------------------|
| Integer.parseInt()   |
| Double.parseDouble() |

| Boolean |       |
|---------|-------|
| true    | false |
|         | &&    |
| !       |       |

| Punctuation |   |
|-------------|---|
| {           | } |
| (           | ) |
| ,           | ; |

| Assignment |
|------------|
| =          |

| Primitive Numeric Types |    |    |
|-------------------------|----|----|
| +                       | -  | *  |
| /                       | %  | ++ |
| --                      | >  | <  |
| <=                      | >= | == |
| !=                      |    |    |

| String   |             |
|----------|-------------|
| +        | ""          |
| length() | compareTo() |
| charAt() | matches()   |

| Arrays   |
|----------|
| a[i]     |
| new      |
| a.length |

| Objects |            |
|---------|------------|
| class   | static     |
| public  | private    |
| final   | toString() |
| new     | main()     |



## INTERFACE IN JAVA



### Tipi in Java

- ✦ Java e' un linguaggio fortemente tipato (ogni entita' ha un tipo)
- ✦ Le classi definiscono il tipo degli oggetti
  - Classe come definizione del contratto di uso degli oggetti che appartengono a quella classe
- ✦ Java prevede un ulteriore meccanismo per associare il tipo agli oggetti: interface



## Java Interface



- Una interface definisce il tipo degli oggetti in modo dichiarativo: non viene presentato il dettaglio della implementazione
- Interface = Contratto d'uso dell'oggetto
- Analogia: le Signature di OCaml


## Esempio




```
public interface Displaceable {  
    public int getX ();  
    public int getY ();  
    public void move(int dx, int dy);  
}
```

Nome

Dichiarazione  
dei tipi dei metodi

|  |   |
|--|---|
| <pre>public class Point implements Displaceable {     private int x, y;     public Point(int x0, int y0) {         x = x0;         y = y0; }      public int getX() { return x; }      public int getY() { return y; }      public void move(int dx, int dy) {         x = x + dx;         y = y + dy;     } }</pre> |  |
|  | <p>Devono essere implementati tutti i metodi dell'interfaccia</p>                   |

|   |   |
|---|---|
| <h2>Un'altra implementazione</h2>   |    |
| <pre>class ColorPoint implements Displaceable {     private Point p;     private Color c;      ColorPoint (int x0, int y0, Color c0) {         p = new Point(x0,y0); c = c0;     }      public void move(int dx, int dy) {         p.move(dx, dy);     }      public int getX() { return p.getX(); }      public int getY() { return p.getY(); }      public Color getColor() { return c; } }</pre> | <p>Oggetti che implementano la stessa interface possono avere stato locale differente</p> <p>Delega all'oggetto point</p> <p>Numero maggiore di metodi di quelli previsti dal contratto</p> |

## Tipi e interface



Dichiarare variabili che hanno il tipo di una interface

```
Diplacable d;  
d = new Point(1,2);  
d.move(-1,1)
```

Assegnare una implementazione

```
d = new ColorPoint(1,2, new Color("red"));  
d.move(-1,1);
```

## Sottotipi



La situazione descritta illustra il fenomeno del subtyping (sottotipo): Un tipo A è un sottotipo di B se un oggetto di tipo A in grado di soddisfare tutti gli obblighi che potrebbero essere richiesti dall'interfaccia o una classe B.

Intuitivamente, un oggetto di tipo A può fare qualsiasi cosa che un oggetto B può fare.

Maggiori dettagli in seguito

## Interface multiple



```

public interface Area {
    public double getArea();
}

public class Circle implements Displaceable, Area {
    private Point center;
    private int radius;

    public Circle (int x0, int y0, int r0) {
        radius = r0; p = new Point(x0,y0);
    }

    public double getArea () { return 3.14159 * r * r; }

    public int getRadius () { return r; }
    public getX() { return center.getX(); }
    public getY() { return center.getY(); }
    public move(int dx, int dy) { center.move(dx,dy);}
}

```

## Esempi di uso



```

Circle c = new Circle(10,10,5);
Displaceable d = c;
Area a = c;

Rectangle r = new Rectangle(10,10,5,20);
d = r;
a = r;

```