



AA 2015-2016

PROGRAMMAZIONE 2

3b. Primi passi in Java



Why Java?

- ✎ Java offre tantissime cose utili
 - Usato a livello industriale
 - Librerie vastissime
 - Complicato ma necessariamente complicato
- ✎ Obiettivo di Programmazione II
 - Presentare le caratteristiche essenziali della programmazione Object-Oriented
 - Illustrare 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

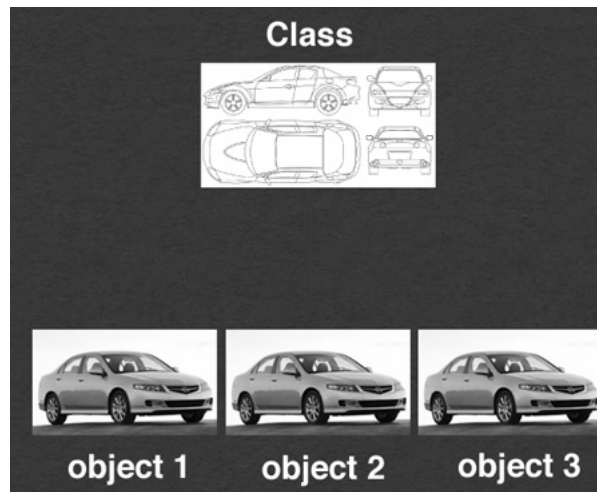
OBJECT
STATO (NASCOSTO)
METODI (PUBBLICO)

Oggetti e classi

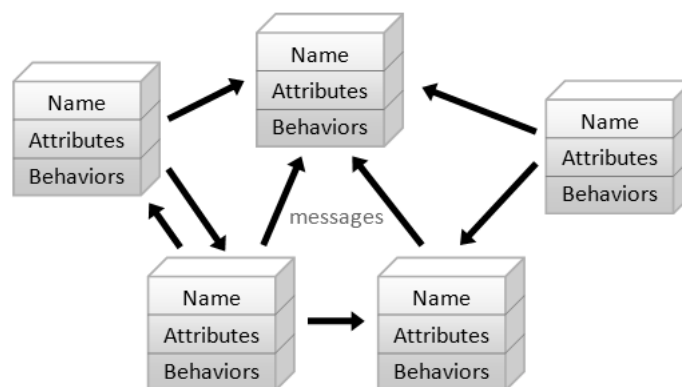


- La definizione di una classe specifica
 - Tipo e valori iniziali dello stato locale degli oggetti (le variabili di istanza)
 - Insieme delle operazioni che possono essere eseguite (metodi)
 - Costruttori (uno o più): codice che deve essere eseguito al momento della creazione di un oggetto
- Ogni oggetto è una istanza di una classe e può (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++; return cnt; }
    // metodo
    public int dec ( ) { cnt--; return cnt; }
}
```

DICHIARAZIONE DI CLASSE

public = visibile fuori
dell'oggetto

private = visibile solo
all'interno dell'oggetto

Esecuzione di Java



un programma Java è mandato in esecuzione
invocando un metodo speciale chiamato main

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

Compilare ed eseguire



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

Viene creato il bytecode Counter.class

```
prompt$ javac First.java
```

Viene creato il bytecode First.class

```
prompt$ java First
```

```
1
```

```
0
```

```
prompt$
```

Cosa è il Java bytecode?



- È 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)
- Visualizzabile con javap !!

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++; return cnt; }  
    // metodo  
    public int dec ( ) { cnt--; return cnt; }  
}
```

**DICHIARAZIONE
 DI CLASSE**

public = visibile fuori
 dell'oggetto

private = visibile solo
 all'interno dell'oggetto

Costruttori con parametri



```
public class First {  
    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 entità mutabile

```
Counter C;  
C = new Counter(5);  
C = new Counter(10);  
C.inc( );  
  
//quale è il valore dello stato locale?
```

Il valore 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...



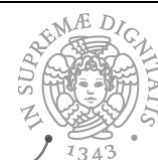
- ☛ Gli oggetti Java sono memorizzati nello heap
- ☛ Nello heap vengono allocate
 - variabili di istanza, quando si crea un oggetto
 - variabili statiche (o di classe), quando è caricata una classe
- ☛ Le variabili allocate nello heap sono inizializzate dal sistema
 - con **0** (zero), per le variabili 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 assegnar loro un valore prima di leggerle

Stato locale



- ☞ Modificatori: meccanismo per controllare l'accesso allo stato locale dell'oggetto
 - Public: visibile/accessibile da ogni parte del programma
 - Private: visibile/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 richiama da vicino la sintassi del 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
- λ // dichiara un oggetto C di tipo Counter e lo inizializza con
- λ // il costruttore
- λ **Counter c = new Counter();**
- λ **Counter d;** // dichiara d e il suo valore di default è **null**
- λ **d = c;** // assegna a d l'oggetto denotato da c => **Aliasing!**

Alcuni comandi...



Condizionali

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

Iterativi

- **while** (exp) { stmt1; stmt2; }
- **do** { stmt1; stmt2; } **until** (exp);
- **for** (init; term; inc) { stmt1; stmt2; }

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);
    }
}
```

Demo 2



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

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



INTERFACCE IN JAVA

Tipi in Java



- ✎ Java è un linguaggio fortemente tipato (ogni entità 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: le interfacce

Java Interface



- Una interfaccia definisce il tipo degli oggetti in modo dichiarativo: non viene presentato il dettaglio della implementazione
- Interfaccia = Contratto d'uso dell'oggetto

Esempio



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

Nome

Dichiarazione
dei tipi dei metodi

Una implementazione...



```
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;
    }
}
```

Devono essere implementati tutti i metodi dell'interfaccia

Un'altra!!



```
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; }
}
```

Oggetti che implementano la stessa interfaccia possono avere stato locale differente

Delega all'oggetto point

Numero maggiore di metodi di quelli previsti dal contratto

Tipi e interfacce



Dichiarare variabili che hanno il tipo di una interfaccia

```
Diplacabile 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

Interfacce 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; center = new Point(x0, y0);
    }

    public double getArea ( ) { return Math.PI * 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 d'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;

```