OOP

- **O**bject **O**riented **P**rogramming : Programmazione Orientata agli Oggetti
- Problema: crescente complessità dei progetti software
- Vantaggi: facilità di comprensione e modifica del programma, robustezza e riusabilità del codice
- Ottenuti grazie a: Incapsulamento, Ereditarietà, Polimorfismo

Programmazione Strutturata e OOP

- Nella programmazione strutturata l'enfasi è posta sull'algoritmo ovvero sulla sequenza di operazioni (procedure o funzioni) da svolgere per risolvere il problema
- Secondo il paradigma OO un programma corrisponde ad un' interazione tra "oggetti"

Paradigma OOP

- Ogni oggetto…
  - Appartiene ad una classe di oggetti simili
  - Ha degli attributi che ne descrivono lo stato
  - Può eseguire delle azioni che ne modificano lo stato (ovvero il valore dei propri attributi)
  - Può richiedere ad altri oggetti di eseguire determinate azioni
- Vedi programmazione ad oggetti e Luca

Capre digitali…

- **Definizione della Classe Capra**
- File capra.java:

```
public Class Capra {
  private string nome;
  private float peso;
  ……
  public void bruca(){  …    }
  public  void salta_staccionata(){  …  }
  ……
}
```

Attributi

Metodi

Capre digitali…

- **Creazione di *Oggetti*** *(invocazione metodo costruttore della Classe)*:

  Achille = new Capra(“Achille”,120,”bianco”,3);

  Ulisse = new Capra(“Ulisse”,130,”grigio”,5);
- **Invocazione di metodi** “su” *Oggetti*:

  Achille.salta_staccionata();

  Achille.bruca();

  Ulisse.spostati_in(5,3);

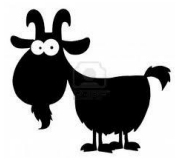
  Ulisse.saluta(Achille); // interazione tra oggetti

Classe Capra

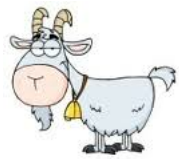
- Modello astratto di capra
  - Attributi: nome, peso, colore_mantello, età, x, y, energia, umore,…
  - Metodi: bruca(), salta_staccionata(), dormi(), saluta(Capra c), spostati_in(int x, int y),…
- La scelta di attributi e metodi da incorporare dipende dal problema e dal programmatore
- Attributi = variabili, Metodi = funzioni

Classe = Tipo di Oggetto

Classe = Capra



Oggetto = Achille
= «istanza» di Capra

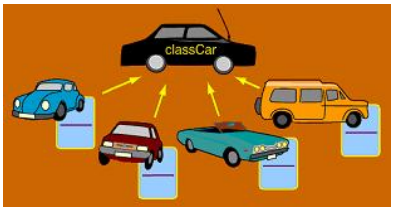


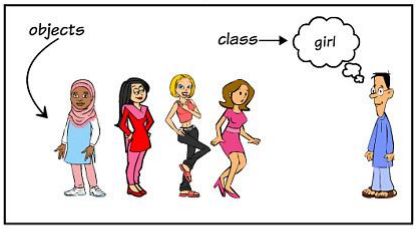
nome	=	Achille
peso	=	120kg
colore_mantello	=	bianco
età	=	3 anni

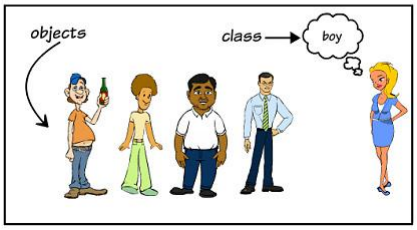
Relazione Metodi - Attributi

- In relazione ai metodi invocati il valore di alcuni attributi viene modificato
- Esempio:
  - bruca() → peso aumenta
  - salta_staccionata() → energia diminuisce
  - dormi() → energia aumenta
  - saluta(Capra c) → umore di c varia
  - spostati_in(int x, int y) → x,y variano

Altri esempi







Classi e Oggetti

- Una classe definisce un nuovo tipo di dato.
- Individua una categoria di oggetti che posseggono determinati attributi(proprietà) e su cui possono essere effettuate solo determinate operazioni(metodi)
- Gli oggetti sono istanze(ovvero elementi) della classe
- La creazione di un oggetto comporta l'assegnazione di valori specifici agli attributi della classe
- Analogia:
  - Classe = Tipo di record,
  - Oggetto = record particolare

Tipo di dato – Dato(valore)

Tipo di dato	Dato(valore)
Integer	3
Float	2.5
Char	b

Paradigma OOP

- Un oggetto è dotato di:
  - Comportamento: metodi (operazioni) che si possono invocare sull'oggetto
  - Stato: valori assunti dagli attributi
  - Identità: possibilità di distinguere un oggetto da altri(anche se ha gli stessi valori per gli attributi)
- Appartiene ad una classe costituita da oggetti dello stesso tipo
- Attributi e metodi sono specificati nella classe

In conclusione…

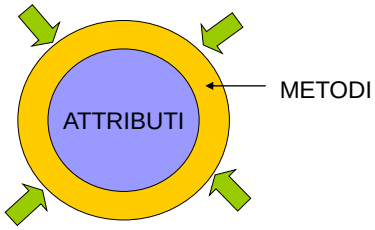
- Definisco la classe con attributi e metodi
- Creo un oggetto dichiarando che appartiene alla classe definita (di cui condivide attributi e metodi) e assegnando dei valori agli attributi
- Invoco i metodi della classe sull'oggetto in questione

Incapsulamento

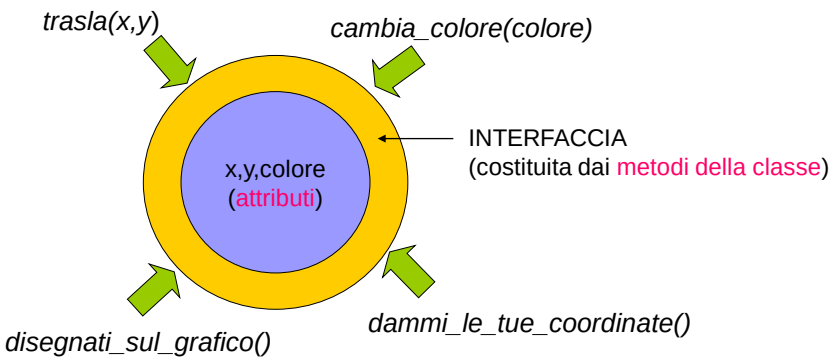
- Con questo termine si indica il fatto che all'interno di una classe sono racchiusi sia i dati che le funzioni che operano su di essi.
- Le classi pertanto hanno un elevato grado di indipendenza dal resto del codice
- Ogni oggetto conserva al proprio interno tutto ciò che gli serve per poter operare…è un'entità autonoma

Information hiding

- E' consentita la modifica degli attributi solo attraverso i metodi «esposti» dalla classe
- Tali metodi rappresentano l'interfaccia della classe mentre l'implementazione dei vari metodi è nascosta al programmatore



Esempio: Classe Punto



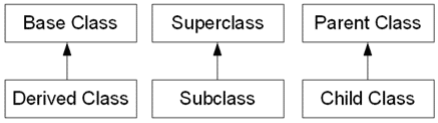
Vantaggi
Incapsulamento e Information Hiding

- E' possibile utilizzare una classe senza conoscere la sua implementazione → *tutela copyright + facilità di utilizzo*
- Eventuali modifiche alla classe non hanno alcun effetto sulle classi che la utilizzano purché non vari la dichiarazione dei metodi → *modularità, manutenibilità, riusabilità*
- E' possibile verificare che agli attributi della classe vengano assegnati valori corretti attraverso opportuni metodi set() e get() → *robustezza del codice*

Ereditarietà

- E' possibile creare sottoclassi che estendono e/o ridefiniscono le funzionalità della classe originaria
  - Nuovi attributi e metodi in aggiunta a quelli della superclasse
  - Diversa implementazione di alcuni metodi
- Vantaggi: Riutilizzabilità e Manutenibilità del codice

Terminologia:



Ereditarietà

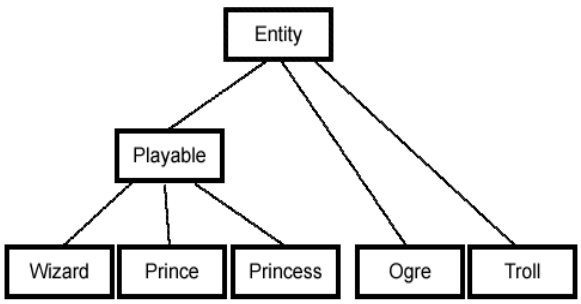
Guerriero

Moschettiere

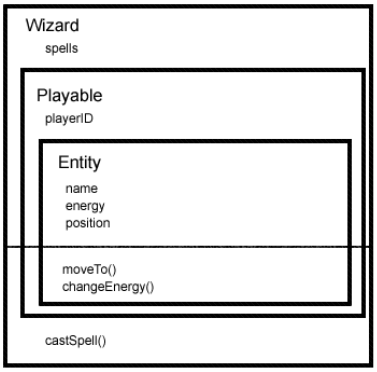
Arciere

Corazzato

Ereditarietà

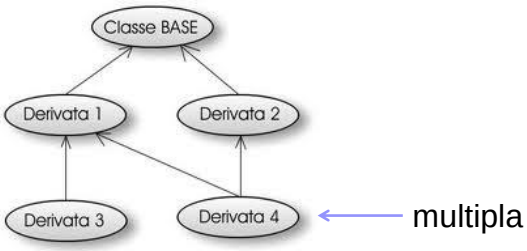


Ereditarietà

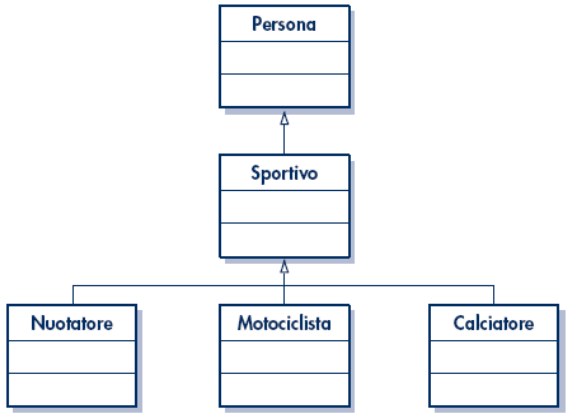


Ereditarietà

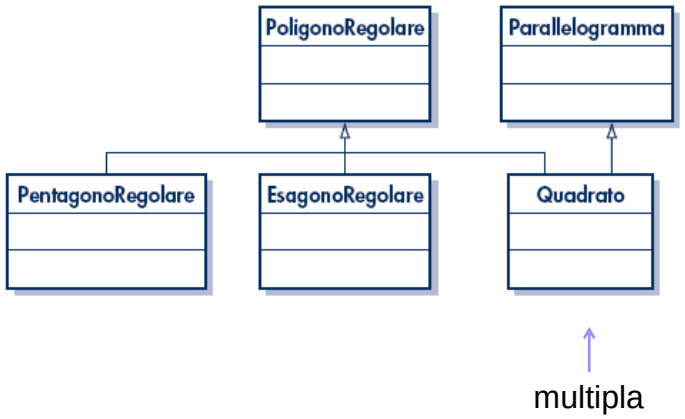
- Singola: un solo «genitore» (superclasse)
- Multipla: più genitori (ovvero una classe può ereditare attributi e metodi da più classi)



Ereditarietà Singola



Ereditarietà Multipla



Interfaccia

- Java: solo ereditarietà singola
- In alcuni casi potrebbe essere utile avere qualcosa di simile all'ereditarietà multipla
- In Java è possibile utilizzare il costrutto interface

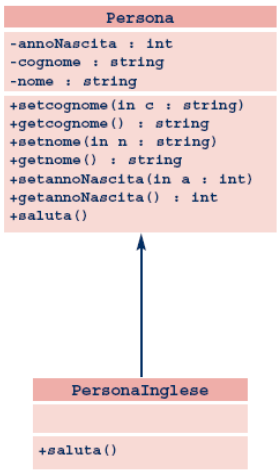
Interfaccia

- Un'interfaccia specifica quali metodi dovranno essere presenti nelle classi che «implementano» l'interfaccia
- Una classe può «implementare» più interfacce oltre ad estendere una superclasse
- Vedi Java-Interfacce

Polimorfismo

- Overriding: cambio implementazione di un metodo nella sottoclasse
  - Esempio: metodo attacca() diverso per le varie sottoclassi di guerriero
- Overloading: definisco più metodi con lo stesso nome ma con parametri diversi nella stessa classe
  - attacca(),
  - attacca(int nemicoX),
  - attacca(string tipo_nemico),…

Overriding




```
class Persona {
  …
…
  public void saluta(){
    String saluto="Salve sono ";
    saluto+=cognome+" "+nome;
    System.out.print(saluto);
}
  …
  …
}
```

```
class PersonaInglese extends Persona {
  public void saluta(){
    String saluto="Hallo I'm ";
    saluto+=cognome+" "+nome;
System.out.print(saluto);
  }
}
```