Programmazione Orientata agli Oggetti in Linguaggio Java

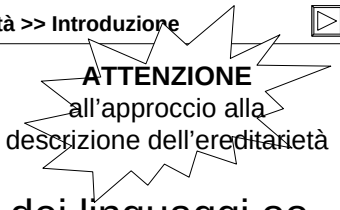
Sintassi e Semantica: Introduzione all'Ereditarietà

versione 2.1

G. Mecca – Università della Basilicata – mecca@unibas.it

Sommario

- Introduzione
- Perchè Esiste l'Ereditarietà
- Regole Meccaniche
- La Classe Object



ATTENZIONE
all'approccio alla descrizione dell'ereditarietà

Introduzione

- Una delle caratteristiche dei linguaggi oo
  - ⇨le classi possono "estendere" altre classi
  - ⇨da cui "ereditano" le caratteristiche
- Ereditarietà
  - ⇨meccanismo importante del linguaggio
  - ⇨ne parliamo sommariamente perchè è una caratteristica della prog. orientata agli oggetti
  - ⇨per ora ci interessa solo dare alcune regole meccaniche (>>)

Perché Esiste l'Ereditarietà

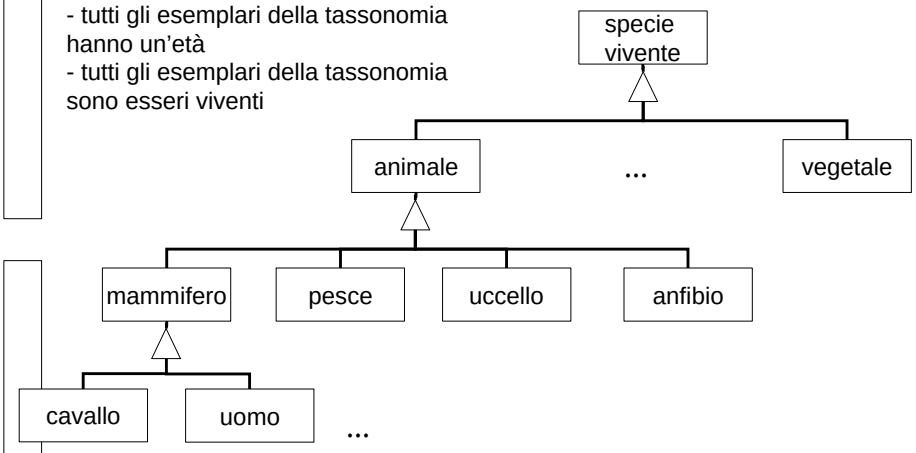
- Perché questi meccanismi ?
  - ⇨perchè rispecchiano aspetti della realtà
- Infatti
  - ⇨gli oggetti software sono fatti per rappresentare oggetti del mondo reale
  - ⇨e tra oggetti del mondo reale si stabiliscono frequentemente relazioni di simili a quelle descritte

Perché Esiste l'Ereditarietà

- Le relazioni nel mondo reale
  - ⇨i concetti del mondo reale sono spesso organizzati in "tassonomie"
  - ⇨una tassonomia è una rappresentazione di concetti – ovvero classi di oggetti – in cui ci sono concetti più generali e concetti più specifici
- Un esempio tipico
  - ⇨la biologia

Esempio di Tassonomia

NOTA
- tutti gli esemplari della tassonomia hanno un'età
- tutti gli esemplari della tassonomia sono esseri viventi



Perché Esiste l'Ereditarietà

- Supponiamo
  - ⇨di dover rappresentare oggetti della tassonomia in un'applicazione
- I requisito utile
  - ⇨vorrei definire una classe EssereVivente con la proprietà eta ed il metodo getEta()
  - ⇨e poi definire tutte le altre classi senza dover replicare la proprietà eta ed il metodo getEta()

Perché Esiste l'Ereditarietà

- II requisito utile
  - ⇨sarebbe utile se – dovendo, ad esempio, sviluppare un'applicazione su uno zoo – potessi mettere assieme in una collezione (es: array) riferimenti ad animali vari
  - ⇨trattandoli semplicemente come Animali
  - ⇨es: Animale a = new Uccello();
  - ⇨es: Animale b = new Cavallo();

Perché Esiste l'Ereditarietà

- L'ereditarietà
  - ⇨esiste nei linguaggi di programmazione per poter soddisfare questi due requisiti
- Idea
  - ⇨si definiscono le "superclassi" che rappresentano i concetti più in alto nella tassonomia
  - ⇨si definiscono le "sottoclassi" estendendo le superclassi esistenti

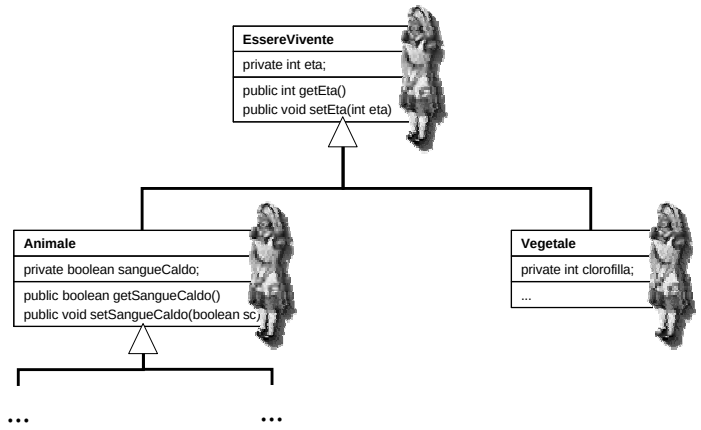
Regole Meccaniche

- Per estendere una superclasse
  - ⇨la sottoclasse deve utilizzare la parola chiave extends
- Esempio

```
public class EssereVivente  {
    private int eta;
    public int getEta() { return this.eta; }
    public void setEta(int eta) { this.eta = eta; }
    ...  }

public class Animale extends EssereVivente {
    private boolean sangueCaldo;
    public void setSangueCaldo(boolean b) {this.sangueCaldo = sangueCaldo; }
    public boolean getSangueCaldo() {return this.sangueCaldo;}
    ...
}
```

Regole Meccaniche



Regole Meccaniche

ATTENZIONE
alle regole meccaniche sull'ereditarietà

- Ipotesi
  - ⇨supponiamo che una classe F ("figlia") estenda una classe P ("padre")
  - ⇨in questo caso F si chiama anche "sottoclasse", P si chiama anche "superclasse"
  - ⇨definiamo le regole meccaniche secondo cui deve essere interpretata la semantica dell'estensione

Regole Meccaniche

- Regola n. 1: eredità delle caratteristiche
  - ⇨gli oggetti di tipo F ereditano tutte le proprietà ed i metodi degli oggetti di tipo P
  - ⇨possono estenderli aggiungendo ulteriori proprietà e ulteriori metodi
  - ⇨possono ridefinire i metodi nel caso in cui i metodi ereditati non siano adeguati
  - ⇨NOTA: non vale il viceversa

Regole Meccaniche

- In altri termini
  - ⇨la regola n. 1 dice che posso scrivere

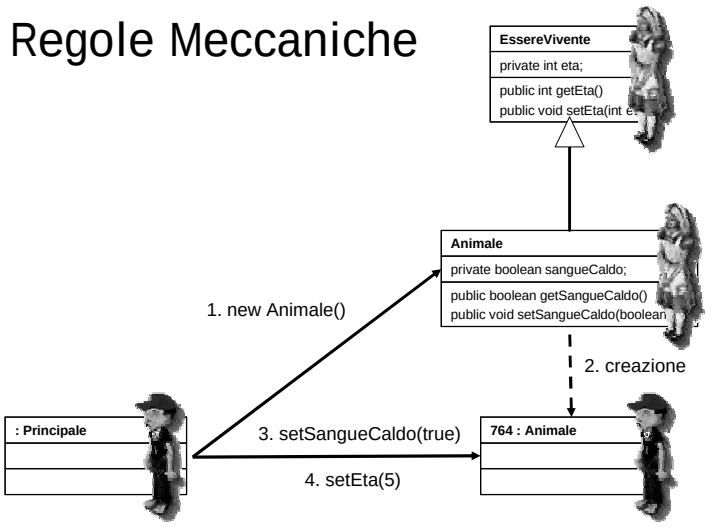
```
Animale a = new Animale();
a.setSangueCaldo(true);
System.out.println(a.getSangueCaldo());
a.setEta(5);
System.out.println(a.getEta());
```

Animale estende EssereVivente e ne eredita i metodi

```
EssereVivente e = new EssereVivente();
e.setEta(5);
System.out.println(e.getEta());
e.setSangueCaldo(true);
System.out.println(e.getSangueCaldo());
```

Attenzione: non vale il viceversa (EssereVivente non eredita i metodi di Animale)

Regole Meccaniche



Regole Meccaniche

- Regola n. 2: sostituibilità del tipo
  - ⇨gli oggetti di tipo F possono essere considerati anche oggetti di tipo P
  - ⇨infatti gli oggetti di tipo F hanno tutte le caratteristiche degli oggetti di tipo P
  - ⇨di conseguenza, gli oggetti di tipo F sono sostituibili a quelli di tipo P (possono essere utilizzati dove servono gli altri)
  - ⇨NOTA: non vale il viceversa

Regole Meccaniche

- In altri termini
  - la regola n. 2 dice che posso scrivere

```
EssereVivente e = new Animale();
e.setEta(5);
System.out.println(e.getEta());
```

un oggetto di tipo Animale è accettabile per un riferimento di tipo EssereVivente

```
Animale a = new EssereVivente();
a.setEta(5);
System.out.println(a.getEta());
```

Attenzione: non vale il viceversa (EssereVivente non è accettabile dove serve un animale)

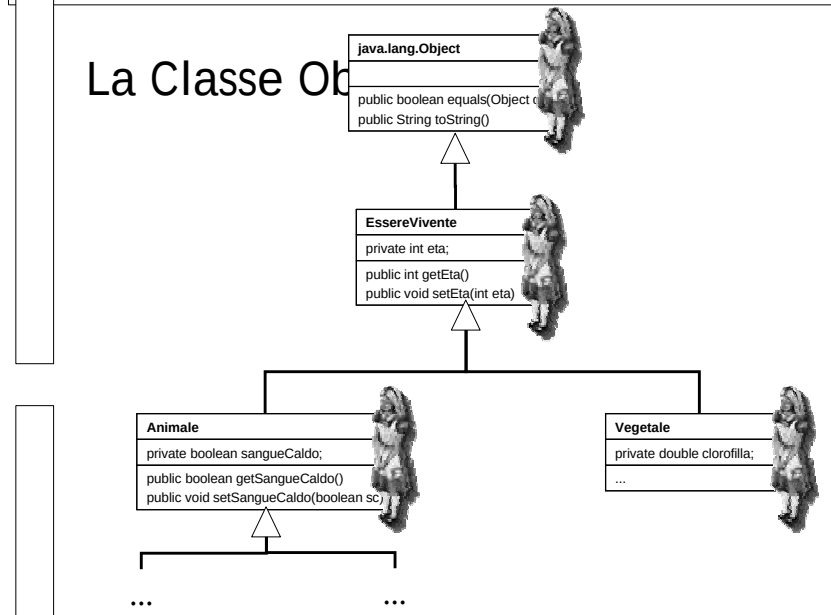
La Classe Object

- Nei linguaggi moderni (Java, C#)
  - una classe può dichiarare di estenderne al massimo un'altra con extends ("ereditarietà singola")
  - inoltre, se una classe non specifica la clausola extends, implicitamente estende una classe speciale denominata java.lang.Object
  - di conseguenza, ciascuna classe estende sempre esattamente una superclasse (o definita esplicitamente oppure Object)

La Classe Object

>> java.lang.Object

- In Java
  - ⇨tutte le classi estendono automaticamente java.lang.Object
- java.lang.Object
  - ⇨vari metodi pubblici
- I principali metodi
  - ⇨boolean equals(Object o)
  - ⇨String toString()



La Classe Object

NOTA: data la semantica del metodo println, basta scrivere: System.out.println(c);

- Esempio
  - ⇨calcolatrice.Circonferenza estende java.lang.Object
- Regola n. 1: eredità delle caratteristiche
  - ⇨le istanze di Circonferenza ereditano i metodi delle istanze di object (es: toString())
  - ⇨es: è possibile scrivere

```
Circonferenza c = new Circonferenza();
System.out.println(c.toString());
```


La Classe Object

- Ma...
  - ⇨il metodo toString() ereditato da Object è poco interessante; produce stringhe del tipo Circonferenza@42ba81
  - ⇨per avere un comportamento più interessante, gli oggetti di tipo Circonferenza possono ridefinire il metodo in modo da stampare le coordinate del centro e il raggio

La Classe Object

- Esempio

```
public class Circonferenza {
  ...
  public String toString () {
     return “Centro: (“ + this.ascissaCentro + “,” +
                                this.ordinataCentro  + “) - ” +
            “Raggio: “ + this.raggio;
   }
  ...
}
```


La Classe Object

- In questo caso
  - ⇨gli oggetti di tipo Circonferenza ereditano toString() dalla superclasse Object
  - ⇨ma lo ridefiniscono e la versione predefinita prevale su quella ereditata

```
Circonferenza c = new Circonferenza();
System.out.println(c);
// risultato: Centro: (2, -1.5) – Raggio: 5.7
```

La Classe Object

```
in java.lang.Object:
public boolean equals(Object obj) {
    return (this == obj);
}
```

- Lo stesso discorso vale per equals()
  - ⇨tutti gli oggetti hanno un metodo public boolean equals(Object o) ereditato da Object
  - ⇨che dovrebbe consentire di confrontarli con altri oggetti
  - ⇨ma il metodo è sostanzialmente inutile, perchè non fa altro che confrontare i due riferimenti
  - ⇨di conseguenza, se necessario, va ridefinito

La Classe Object

- Regola n. 2: sostituibilità del tipo
  - ⇨un oggetto di tipo Calcolatrice è anche un oggetto di tipo Object
  - ⇨può essere usato dovunque serva un oggetto di tipo Object
  - ⇨es: è possibile scrivere
    Object o = new Calcolatrice();

Riassumendo

- Introduzione
- Perchè Esiste l'Ereditarietà
- Regole Meccaniche
- La Classe Object

Termini della Licenza