UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
DIPARTIMENTO DI ECCELLENZA

Università di degli Studi di Salerno
Dipartimento di Informatica

Programmazione ad Oggetti

a.a. 2023-2024

Upcasting

Docente: Prof. Massimo Ficco

E-mail: *mficco@unisa.it*

Upcasting

La funzionalità più importante dell'ereditarietà non è la possibilità di poter aggiungere funzionalità ad una classe

La cosa più importante è che **un oggetto della *classe derivata è anche un oggetto della classe base***

Tale definizione è supportata dal linguaggio nella pratica!!!!!

Upcasting

```
// Un messaggio inviabile alla classe base può essere inviato anche alla classe derivata
class Instrument {
    public void play() {System.out.println("Play Instrument");}
    static void tune(Instrument i) {i.play();}
}

// Un oggetto Wind eredita l'interfaccia della classe base
public class Wind extends Instrument {
        public  void play(){System.out.println("Play Wind");
        static void tune(Wind i) {i.play();}

    public static void main(String[] args) {
        Wind flute1 = new Wind();
        flute1.play(); → "Play Wind"
        flute1.tune(flute1); → "Play Wind"

        Instrument flute2 = new Wind(); // Upcasting
        flute2.play();   // Upcasting → "Play Wind"
        Wind.tune(flute2); // Upcasting → "Play Wind"
        Instrument.tune(flute2); // Upcasting → "Play Wind"

Instrument pfd = new Instrument();
        Instrument.tune(pfd); // → "Play Instrument
}}
```

La classe java.lang.Object

In Java:

- Gerarchia di ereditarietà semplice
- Ogni classe ha una sola super-classe

Se non viene definita esplicitamente una super-classe, il compilatore usa la classe predefinita **Object**

- Object non ha super-classe!

Metodi di Object

Object definisce un certo numero di **metodi pubblici**

- Qualunque oggetto di qualsiasi classe li eredita
- La loro implementazione base è spesso minimale
- La tecnica del polimorfismo permette di ridefinirli

public boolean **equals**(Object o)

- Restituisce "vero" se l'oggetto confrontato è identico (ha lo stesso riferimento) a quello su cui viene invocato il metodo
- Per funzionare correttamente, ogni sottoclasse deve fornire la propria implementazione polimorfica

Metodi di Object

public String **toString**()

- Restituisce una rappresentazione stampabile dell'oggetto
- L'implementazione base fornita indica il nome della classe seguita dal riferimento relativo all'oggetto (java.lang.Object@10878cd)

public int **hashCode**()

- Restituisce un valore intero legato al contenuto dell'oggetto
- Se i dati nell'oggetto cambiano, deve restituire un valore differente
- Oggetti "uguali" **devono** restituire lo stesso valore, oggetti diversi **possono** restituire valori diversi
- Utilizzato per realizzare tabelle hash

Controllo Override

- Pertanto molti linguaggi supportano opportuni controlli introducendo nuove parole chiavi. La soluzione di Java è quello di usare un'annotazione:

```
public class Animale{
  private String nome;

  @Override
  public boolean equals(Animale animale){
    return nome.equals(animale.nome);
  }
}
```

- In questo caso il compilatore ci restituisce un errore:

  The method equals(Animale) of type Animale must override or implement a supertype method

Controllo Override

- Ecco la soluzione corretta:

```
public class Animale{
  private String nome;

  @Override
  public boolean equals(Object animale){
    return nome.equals(((Animale) animale).nome);
  }
}
```

- Come evitare che un metodo di una classe base possa essere sovrascritto in quella derivata?