Final

È una keyword

- FINAL + ATTRIBUTO → es: Private final int Valore = 10;
  - ↓ trasforma un attributo (variabile) in una costante
- FINAL + METODO → Public final int getAnno(){ }
  - non può fare override del metodo (ossia ridefinire il metodo di una superclasse)
  - final segnala sottoclasse non può fare override di questo metodo definito nella superclasse
- FINAL + CLASSE → public final class DATA{ ... }
  - la classe non può essere estesa

Upcasting

Fare casting vuol dire assegnare un certo tipo a una variabile e poi utilizzarlo come se fosse di un altro tipo.

es2 int value = 10

(double) value ⟹ operazione di casting trasforma il tipo di dato

Vogliamo fare upcasting

Supponiamo di avere

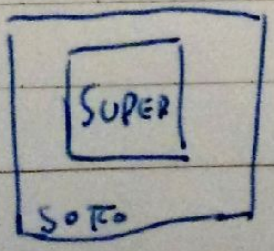
CLASSE (SUPER)

↑ EXTENDS

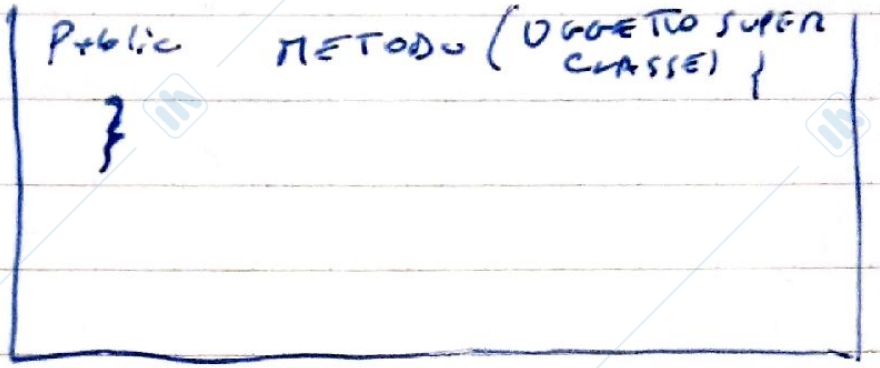
CLASSE (SOTTO)

e abbiamo un oggetto X della sottoclasse

questo X vogliamo farlo diventare un oggetto della superclasse



Immaginiamo di avere



Questa operazione di upcasting permette al metodo di agire qualsiasi oggetto di una classe che viene estesa

es
```
public class STRUMENTOMUSICALE {
    // ATTRIBUTI
    public void Play() {
        // IMPLEMENTAZIONE METODO
    }
    public static void Tune (STRUMENTOMUSICALE x) {
        X.Play();
    }
}
```

4

Mo Tu We Th Fr Sa Su

No. SISTEMI INF.

Date 11 · 12 · 19

```
public class STRUMENTOAFIATO extends STRUMENTOMUSICALE{
    public static void main (STRING[] args){
        STRUMENTOAFIATO FLAUTO= NEW STRUMENTOAFIATO();
        STRUMENTOMUSICALE.TUNE(FLAUTO);
    }
}
```

Il tipo di dato richiesto da tune è STRUMENTOMUSICALE mentre noi qui stiamo usando un tipo diverso (STRUMENTOAFIATO)

↓

in Java si può fare, in C assolutamente no

non si può fare tuttavia che tune richieda un tipo di una sottoclasse (es. STRUMENTOAFIATO) poiché potrebbe usare metodi che non sono presenti nella superclasse

Se un metodo di una superclasse viene ridefinito in molte sottoclassi (OVERRIDE)

↓

è possibile chiamarli tutti uguali senza perdere alcuna informazione

↓

POLIMORFISMO

Polimorfismo

Supponiamo di avere: tipo enumerativo

```
public enum NOTA {
        DO, DO_D, RE, RE_D;
}
```

```
public class STRUMENTOMUSICALE {
    public void play (NOTA N) {
        System.out.println ("LO STRUMENTO SUONA" +N);
    }
}

public class STRUMENTOAFIATO extends STRUMENTOMUSICALE {
    public void play (NOTA N) {
        System.out.println ("LO STRUMENTO A FIATO SUONA" +N);
    }
}

public class STRUMENTOACORDE extends STRUMENTOMUSICALE {
    public void play (NOTA N) {
        System.out.println ("LO STRUMENTO A CORDE SUONA" +N);
    }
}
```

```
public class musica {
    public static void main (String[] args){
        STRUMENTOAFIATO FLAUTO = new STRUMENTOAFIATO();
        STRUMENTOACORDE CHITARRA = new STRUMENTOACORDE();

        TUNE(FLAUTO);
        TUNE(CHITARRA);
    }
    public static void TUNE (~~STRUMENTOAFIATO~~ STRUMENTOMUSICALE x){
        x.PLAY(NOTA.DO);
    }
    public static void TUNE (STRUMENTOACORDE x){
        x.PLAY(NOTA.DO);
    }
}
```

metodo polimorfico

Modifiche sul codice in rosso

UPCASTING e POLIMORFISMO

play

X.PLAY può eseguire il metodo dello STRUMENTOAFIATO o dello STRUMENTOACORDE

dipende da cosa passo a TUNE