Subtyping

Università
Ca'Foscari
Venezia

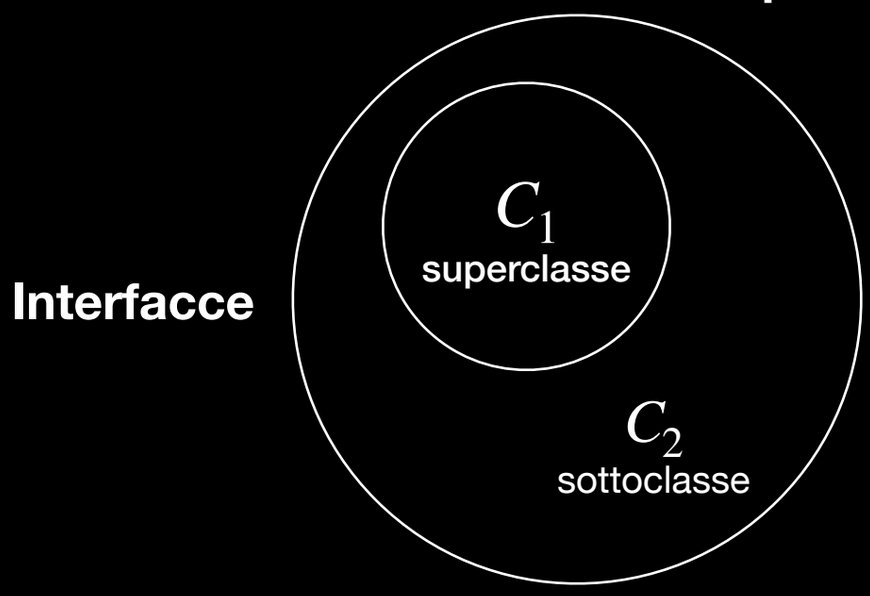
Tipi in Java

- Fortemente tipato: il tipo dell'oggetto deve essere compatibile con il tipo dichiarato nella funzione chiamata (anche per gli assegnamenti)
- Tipato staticamente: ogni espressione ha un tipo (dichiarato o inferito) conosciuto a compile time

Principio di sostituzione

Un oggetto o_1 istanza della classe C_1 può essere sostituito da un oggetto o_2 di tipo C_2 se quest'ultima offre un interfaccia più ampia o uguale di C_1

- Quindi se C_2 estende C_1, un'istanza di C_1 può essere sostituita da C_2, in quanto offre un'interfaccia più ampia



Sottotipi e polimorfismo

- Se la classe C_2 estende C_1, C_2 è un sottotipo di C_1
- Grazie ai sottotipo e l'ereditarietà è possibile avere **polimorfismo**: quando lo stesso simbolo (classe) rappresenta più tipi e possiede comportamenti diversi.
- L'override è un esempio di polimorfismo: C_1 ne conferma l'esistenza, ma a runtime viene eseguita l'implementazione di C_2

  Motivazione nelle prossime slide

```
Classe c =  new Sottoclasse(…)
c.metodo(…)
```

Classe assicura l'esistenza di *metodo*

L'esecuzione è quella sovrascritta da *Sottoclasse*

Tipi dinamici e statici

- I tipi **statici** sono quelli indicati quando si dichiara una variabile, un parametro o il tipo di ritorno di un metodo
- Sono conosciuti a tempo di compilazione e utilizzati per inferire il tipo delle operazioni

- I tipi **dinamici** sono conosciuti solamente a runtime
- Possono essere sottotipi del tipo statico (Quindi avere un'interfaccia uguale o più ampia)
- Possono variare ad ogni esecuzione

Il tipo dinamico determina l'implementazione dei metodi da utilizzare:

Partendo dalla classe, si utilizza l'implementazione definita nella superclasse più vicina al tipo dinamico. Questa esiste sempre perché assicurato dal tipo statico

Type casting

`(type) expression`

- Un’espressione può essere castata ad un sottotipo del suo tipo statico
- Inutile farlo per un supertipo perché ne eredita già tutte le caratteristiche
- Se il tipo dinamico non è compatibile viene lanciato un errore a runtime
- Per evitare errori a runtime e verificare il tipo dinamico di un espressione si utilizza la keyword `instanceof`

`expression instanceof Type`

→ **Ritorna** true **sse** *expression* **è un sottotipo di** *Type*

Università
Ca’Foscari
Venezia

Interfacce

`class Class extends … implements Interface1, Interface2, …`

- Risolvono il problema dell'ereditarietà singola
- Definiscono solamente le firme dei metodi, senza campi e implementazioni (come per le classi astratte)
- Una classe può implementare più di una interfaccia e deve implementare tutti i loro metodi, o essere astratta
- Non possono essere istanziate

Una classe puo' implementare un interfaccia e di conseguenzas implementare tutti i metodi di tale interfaccia

Interfacce

- Da Java 8 è possibile aggiungere implementazioni di default nei metodi
- Se più interfacce implementano lo stesso metodo di default il programma non compila (Quale implementazione usare?)
- Possono utilizzare gli altri metodi dell'interfaccia solamente se pubblici (per poter accedere ai campi è necessario dichiarare dei getter e dei setter, che verranno poi implementati dalla classe)

```
interface Interface{
  public Type1 method1(…)
  default public Type2 method2(…){
    …
    this.method1(…)
    …
  }
}
```

Università
Ca'Foscari
Venezia

Interfacce

- Con le interfacce si viene a creare una relazione di sottotipo ma non di ereditarietà
- In questo modo una classe è un sottotipo della classe che estende o delle interfacce che implementa
- Le gerarchie non sono più alberi, ma grafi non direzionati aciclici
- Inoltre le interfacce possono esterne altre

Interfacce o classi astratte

- Le interfacce sono solitamente utilizzate per modellare una proprietà che più oggetti possono possedere
- Le classi astratte modellano un’entità generica che può possedere uno stato