Aspetti dell’ereditarietà

Sandro Morasca

Università degli Studi dell'Insubria

Dipartimento di Scienze Teoriche e Applicate

Via Mazzini 5

21100 Varese

sandro.morasca@uninsubria.it

Ereditarietà

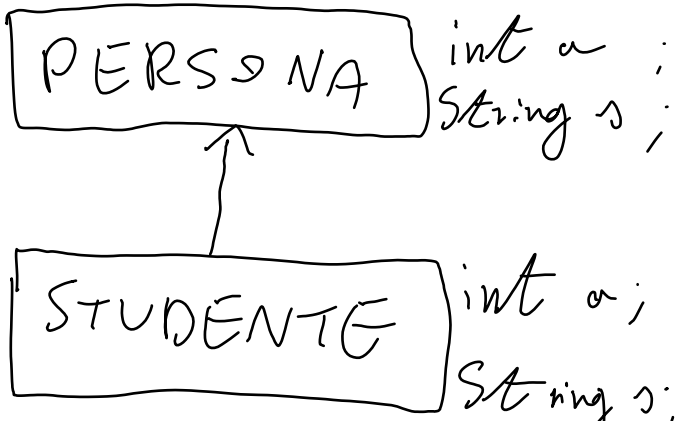
- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Gli oggetti di una classe possono essere oggetti anche di un'altra classe
  - oggetti di classe `Studente` sono casi particolari degli oggetti di classe `Persona`
    - la classe `Studente` si dice sottoclasse della classe `Persona`
    - la classe `Persona` si dice superclasse della classe `Studente`

Ereditarietà

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Gli oggetti della classe `Studente`
  - hanno automaticamente tutte le proprietà (dati e metodi membro) degli oggetti della classe `Persona`
  - possono avere altre proprietà (dati e metodi membro) in più degli oggetti della classe `Persona`
  - possono ridefinire alcune proprietà (dati e metodi membro) degli oggetti della classe `Persona` per meglio adattarle alla natura degli studenti
- Si dice che la sottoclasse "eredita" dalla superclasse le proprietà



class persona{
int a;
String s;
}

class studente extends persona{
char t;
}

Ereditarietà

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Sintassi

```
class SottoClasse extends SuperClasse
{
   ...
}
```

- Semantica

  `SottoClasse` eredita da `SuperClasse`

Vantaggi e svantaggi

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- L'ereditarietà è un meccanismo di programmazione molto potente
  - permette di non dover riscrivere codice
    - riuso del codice
    - riusabilità
    - estendibilità
  - esiste una sola copia del software riusato
    - modificabilità del software
  - aiuta ad organizzare i componenti software mostrando le relazioni tra di loro
    - comprensibilità
- Tuttavia, l'ereditarietà deve essere utilizzata con particolare cautela per evitare di commettere errori che possono rivelarsi in fase di compilazione o, peggio, di esecuzione

Costruzione di una sottoclasse

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Nella costruzione di una sottoclasse si possono
  - aggiungere nuovi elementi (dati e metodi) rispetto alla superclasse
  - ridefinire gli elementi (dati e metodi) protetti e pubblici della superclasse
- La costruzione della sottoclasse non comporta alcuna modifica della superclasse
- Nell'aggiunta e nella ridefinizione di elementi non si usa nella sottoclasse alcuna sintassi particolare

Riferimento alla superclasse

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Se necessario, per riferirsi alla superclasse si utilizza la parola chiave **`super`**

  `super``.metodo(...)`

- è la chiamata al metodo `metodo` della superclasse
  - si effettua spesso all'interno del metodo `metodo` della sottoclasse
  - il metodo della superclasse effettua le operazioni necessarie sui dati membro ereditati dalla superclasse
  - la parte restante del metodo della sottoclasse effettua le altre operazioni specializzate per la sottoclasse

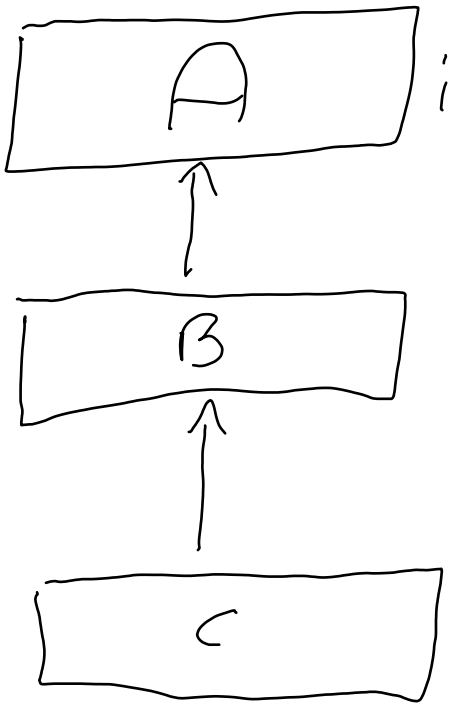
Riferimento alla superclasse

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Non dimenticarsi della parola chiave **`super`**, altrimenti

  `metodo(...)`

  è la chiamata al metodo `metodo` della sottoclasse
  - una chiamata ricorsiva se effettuata all'interno del metodo metodo della sottoclasse



int a;

- Non si può andare alla superclasse di una superclasse:

  `super.super.``metodo(...)`

  è un errore di sintassi.

```
Class C {
  super.super.a;   Vietato
}
```

Costruttori e finalizzatori di una sottoclasse

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- All'inizio dell'esecuzione di un costruttore di una sottoclasse viene SEMPRE chiamato un costruttore della superclasse.
- Se c'è una chiamata esplicita

  ```
  super(lista_parametri_effettivi)
  ```

  viene chiamato il costruttore corrispondente della superclasse individuato dalla lista dei parametri effettivi
- Questa chiamata deve SEMPRE essere la prima istruzione del costruttore della sottoclasse: non ci può essere nemmeno una dichiarazione di variabile prima della chiamata

  ```
  super(lista_parametri_effettivi)
  ```

Costruttori e finalizzatori di una sottoclasse

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Se non c'è una chiamata esplicita a un costruttore della superclasse
  - viene chiamato implicitamente e automaticamente il costruttore per default della superclasse
- È buona norma chiamare il finalizzatore della superclasse **`super.`**`finalize()` come ultima istruzione del finalizzatore della sottoclasse
  - Il finalizzatore della sottoclasse non chiama automaticamente il finalizzatore della superclasse

Gerarchia di ereditarietà

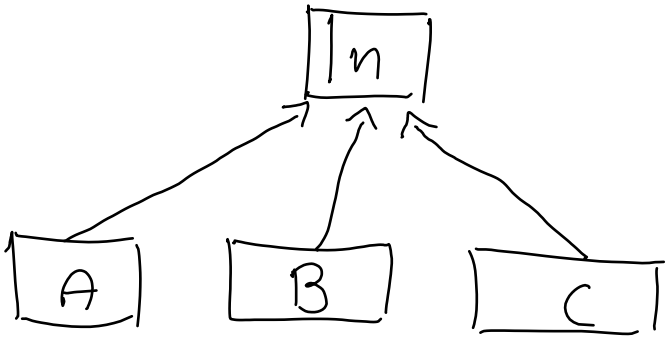
- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

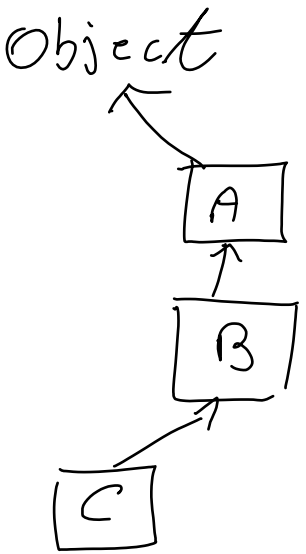
- Il meccanismo dell'ereditarietà può essere replicato in cascata a piacere
  - `A` superclasse di `B` *AND* `B` superclasse di `C` ⇒
    - `A` superclasse di `C`
  - ovvero `C` è indirettamente sottoclasse di/eredita da `A`
- In Java, la gerarchia di ereditarietà tra le classi è un albero
  - una classe può ereditare da una sola classe
  - l'ereditarietà multipla da classi NON è consentita

Gerarchia di ereditarietà

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- L'ereditarietà multipla è consentita solo per "ereditare" dalle **`interface`**
- In Java, la classe `Object` è la radice dell'albero della gerarchia di ereditarietà
  - si trova nel package `java.lang`, importato implicitamente in tutte le applicazioni Java
- Ogni classe implicitamente o esplicitamente eredita direttamente o indirettamente da `Object`





Metodi e classi `final`

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Con la parola chiave **`final`** si designano
  - metodi che non possono essere ridefiniti nelle sottoclassi di una classe
  - classi che non possono essere ridefinite in sottoclassi
- Esempi

```
public final void metodo()
```

```
public final class SottoClasse extends SuperClasse
```

- I metodi **`static`** sono anche **`final`** e non possono essere ridefiniti
- I metodi **`private`** possono essere ridefiniti
  - comunque non sono visibili nelle sottoclassi
  - perciò i metodi private **`static`** possono essere ridefiniti

Metodi e classi `abstract`

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Con la parola chiave **`abstract`** si designano
  - metodi che devono essere ridefiniti nelle sottoclassi di una classe
  - classi che devono essere ridefinite in sottoclassi
- Non è possibile istanziare oggetti di classi **`abstract`**
  - solo oggetti di sue sottoclassi non **`abstract`**
- È sufficiente che un metodo sia **`abstract`** perché l'intera classe sia **`abstract`**
  - la classe **deve** essere dichiarata **`abstract`**
- Esempi

```
public abstract void metodo()

public abstract class SuperClasse
```

Interfacce

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Con la parola chiave **`interface`** si designano "scheletri di classi" in cui
  - i metodi sono tutti **`public abstract`**,
    - anche se non è scritto esplicitamente, e
    - devono essere ridefiniti nelle classi e sottoclassi che la implementano
  - i dati sono obbligatoriamente **`public final static`**
    - anche se non è scritto esplicitamente

Interfacce

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo


```
public interface Interfaccia
{
    public final static int COSTANTE1 = 30;
    public double COSTANTE2 = 3.14;
      //e' comunque final static
    public abstract void m( int x );
    public int q( double y );
      //e' comunque abstract

 }
```

Interfacce, ereditarietà e implementazione

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Le interfacce sono legate tra di loro dalla relazione di ereditarietà

```
public interface Interfaccia1 extends
Interfaccia
{
   ...
}
```

- Un'interfaccia può ereditare da più di un'interfaccia

```
public interface Interfaccia2 extends
Interfaccia, Interfaccia1
{
   ...
}
```

Interfacce, ereditarietà e implementazione

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Le interfacce sono legate alle classi dalla relazione di implementazione

```
public class Classe1 implements Interfaccia
{
   ...
}
```

- Una classe può estendere una classe e implementare più interfacce

```
public class Classe2 extends Classe1
implements Interfaccia1, Interfaccia2
{
   ...
}
```

Interfacce, ereditarietà e implementazione

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- In Java una classe non può ereditare da più classi
  - ESEMPIO NON CORRETTO

```
public class ClasseSbagliata extends
PrimaClasse,SecondaClasse
{
   ...
}
```

- Una classe non è sostituibile a più classi
  - al massimo a una sola classe

- Una classe è però sostituibile a più interfacce

Migrazione di oggetti nella gerarchia di ereditarietà

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Un oggetto di una sottoclasse può essere trattato come un oggetto di una delle sue superclassi
  - se si tratta un oggetto di una superclasse come un oggetto di una delle sue sottoclassi si possono avere errori
- Un riferimento ad un oggetto di una sottoclasse può essere assegnato al riferimento ad un oggetto di una delle sue superclassi
  - un riferimento ad un oggetto di una superclasse NON può essere assegnato al riferimento ad un oggetto di una delle sue sottoclassi

Polimorfismo

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Un oggetto di una sottoclasse può essere trattato come un oggetto di una delle sue superclassi
- Il riferimento a un oggetto di una superclasse è una variabile
  - può puntare a oggetti diversi durante l'esecuzione di un programma
- Il riferimento a un oggetto di una superclasse può in ogni momento puntare a un oggetto della superclasse oppure a un oggetto di una qualunque delle sue sottoclassi
  - ogni riferimento può puntare a un oggetto che assume "molte forme" (polimorfismo)

Polimorfismo

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Il polimorfismo è un meccanismo di programmazione molto potente, che viene sfruttato bene soprattutto insieme con l'ereditarietà
  - permette il riuso del software
  - permette di utilizzare sempre il metodo o il dato membro corretto per ogni oggetto, tramite il binding dinamico (*late* binding)
  - favorisce una vera estendibilità del software
  - favorisce la modificabilità del software
  - diminuisce la leggibilità del software
  - diminuisce la verificabilità del software

Binding dinamico

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Quando si deve eseguire un metodo su un oggetto

  `rifOggetto.metodo(...)`

  non è detto che il metodo sia stato dichiarato nella classe cui appartiene l'oggetto
  - il metodo può essere stato dichiarato in una superclasse della classe dell'oggetto
- Il metodo che viene eseguito è quello con la stessa segnatura dichiarato nella classe stessa oppure nella superclasse più vicina

Binding dinamico

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo


```
class ClasseA
{
    void metodo1( int par ) { … }
    void metodo2( int par ) { … }
    ...
 }
 class ClasseB extends ClasseA
 {
    void metodo1( int par ) { … }
    //ridefinizione di metodo
    ...
}
class ClasseC extends ClasseB
{
    void metodo3( int par ) { … }
    ...
}
```

Binding dinamico

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo


```
ClasseA rifA = new ClasseA();
ClasseB rifB = new ClasseB();
ClasseC rifC = new ClasseC();


rifA.metodo1(12); //è metodo1 dichiarato in ClasseA
rifA.metodo2(32); //è metodo2 dichiarato in ClasseA
rifB.metodo1(36); //è metodo1 dichiarato in ClasseB
rifB.metodo2(87); //è metodo2 dichiarato in ClasseA
rifC.metodo1(43); //è metodo1 dichiarato in ClasseB
rifC.metodo2(54); //è metodo2 dichiarato in ClasseA
rifC.metodo3(65); //è metodo3 dichiarato in ClasseC
```

Binding dinamico

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- In generale
  - non è noto al momento della compilazione quale metodo verrà eseguito durante l'esecuzione
  - il collegamento (binding) tra il nome del metodo e il metodo che viene effettivamente eseguito è "tardivo" (late), in quanto è noto solo durante l'esecuzione, poiché i riferimenti possono puntare a oggetti di tipi diversi (purché siano classi, sottoclassi e superclassi compatibili) in momenti diversi

Conversione di tipo dei riferimenti agli oggetti: da sottoclasse a superclasse

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Quando serve, il riferimento a un oggetto di una sottoclasse viene convertito IMPLICITAMENTE in un riferimento a oggetto di una delle sue superclassi
- Quando si assegna il riferimento a un oggetto di una sottoclasse a un riferimento a un oggetto di una superclasse, bisogna stare attenti a quali dati e metodi si usano/si possono usare quando si usa il riferimento all'oggetto della superclasse
  - Dati e metodi ORIGINARI della superclasse, né aggiunti né ridefiniti nella sottoclasse
    - il riferimento all'oggetto della superclasse usa i dati e i metodi ORIGINARI della superclasse
  - Dati e metodi RIDEFINITI nella sottoclasse
    - il riferimento all'oggetto della superclasse usa i dati e i metodi RIDEFINITI nella sottoclasse

Conversione di tipo dei riferimenti agli oggetti: da sottoclasse a superclasse

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Dati e i metodi AGGIUNTI nella sottoclasse, non esistenti nella superclasse
  - il riferimento all'oggetto della superclasse NON PUÒ usare i dati e i metodi AGGIUNTI nella sottoclasse

```
SuperClasse rifSopra;
SottoClasse rifSotto = new SottoClasse();
...
rifSopra = rifSotto;
rifSopra.metodoNonRidefinito();
//Si esegue il metodo originario di Superclasse
rifSopra.metodoRidefinito();
//Si esegue il metodo ridefinito in Sottoclasse
rifSopra.metodoAggiunto();
//Attenzione! Non è legale: errore in compilazione!
```

Conversione di tipo dei riferimenti agli oggetti: da sottoclasse a superclasse

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Se si effettua la conversione esplicita di un riferimento a un oggetto di una sottoclasse a un riferimento a un oggetto di una superclasse tramite una conversione (cast) di tipo, non si cambia comunque il risultato

```
SuperClasse rifSopra;
SottoClasse rifSotto = new SottoClasse();
...
rifSopra = rifSotto;

((SuperClasse)rifSopra).metodoNonRidefinito();
//Si esegue il metodo originario di

// Superclasse
```

Conversione di tipo dei riferimenti agli oggetti: da sottoclasse a superclasse

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo


```
((SuperClasse)rifSotto).metodoNonRidefinito();
//Si esegue il metodo originario di Superclasse

((SuperClasse)rifSopra).metodoRidefinito();
//Si esegue il metodo ridefinito in Sottoclasse

((SuperClasse)rifSotto).metodoRidefinito();
//Si esegue il metodo ridefinito in Sottoclasse

((SuperClasse)rifSopra).metodoAggiunto();
//Attenzione! Non è legale: errore in compilazione!

((SuperClasse)rifSotto).metodoAggiunto();
//Attenzione! Non è legale: errore in compilazione!
```

Conversione di tipo dei riferimenti agli oggetti: da superclasse a sottoclasse

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Quando dovesse servire, il riferimento a un oggetto di una superclasse PUÒ ESSERE CONVERTITO SOLO ESPLICITAMENTE in un riferimento a oggetto di una delle sue sottoclassi tramite una conversione (cast) di tipo.

```
SuperClasse rifSopra = new SuperClasse();
SottoClasse rifSotto = new SottoClasse();
...
//Cast ridondante

rifSopra = (SottoClasse)rifSotto;
...
//Cast necessario

rifSotto = (SottoClasse)rifSopra;
...
```

Conversione di tipo dei riferimenti agli oggetti: da superclasse a sottoclasse

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Attenzione! Questo genere di conversione può portare a molti problemi!

```
SuperClasse rifSopra = new SuperClasse();
SottoClasse rifSotto;


((SottoClasse)rifSopra).metodoNonRidefinito();
//Attenzione! Non è legale: eccezione in esecuzione!


((SottoClasse)rifSopra).metodoRidefinito();
//Attenzione! Non è legale: eccezione in esecuzione!


((SottoClasse)rifSopra).metodoAggiunto();
//Attenzione! Non è legale: eccezione in esecuzione!


rifSotto = (SottoClasse)rifSopra;
//Attenzione! Non è legale: eccezione in esecuzione!
```

Conversione di tipo dei riferimenti agli oggetti: da superclasse a sottoclasse

- Concetti base
- Sottoclassi
- Gerarchia
- Classi speciali
- Interfacce
- Polimorfismo
- Cast di tipo

- Bisogna sempre essere sicuri di poter eseguire le operazioni richieste
- Ad esempio, si può utilizzare l'operatore **`instanceof`**

```
SuperClasse rifSopra = new SuperClasse();
SottoClasse rifSotto;

if ( rifSopra instanceof SottoClasse )
{
   ((SottoClasse)rifSopra).metodoNonRidefinito;
   ((SottoClasse)rifSopra).metodoRidefinito;
   ((SottoClasse)rifSopra).metodoAggiunto;
   rifSotto = (SottoClasse)rifSopra;
}
```