Ereditarietà e Polimorfismo

Motivazioni e vantaggi

- riutilizzo codice (con un senso logico vicino)
  - accesso funzionalità superclassi
- specializzazione codice esistente
  - ridefinizione metodi
- evita duplicazione codice
  - = creazione di classi tutte separate
- semplificare la costruzione di nuove classi
- facilitare la manutenzione
  - es. modifica a superclasse va alle sottoclassi
- garantire la consistenza delle interfacce
  - = logicamente i dati sono divisi bene

```
class Car{
        String motore;
        int cilindrata;
};

class SUV extends Car{
        // aggiungiamo delle caratteristiche
};
```

Terminologia

- Classe *base*
  - Offre delle caratteristiche comuni
    - (Metodi e proprietà)
  - Chiamata anche classe genitore/superclassi
- Classe *derivata*
  - Offre più funzionalità
    - Ha almeno le funzionalità base
    - Ma aggiunge le proprie

    - Può ridefinire il comportamento di metodi
        - Polimorfismo = Almeno le caratteristiche base + caratteristiche figlie
        - Posso usare sia "il sopra" che "il sotto"
- Esempio polimorfismo

```
// Classe base Animale
class Animale {
    public void emettiSuono() {
        System.out.println("L'animale emette un suono");
    }
}

// Classe derivata Cane
class Cane extends Animale {
    public void emettiSuono() {
        System.out.println("Il cane abbaia");
    }
}

// Classe derivata Gatto
class Gatto extends Animale {
    public void emettiSuono() {
        System.out.println("Il gatto miagola");
    }
}

public class EsempioPolimorfismo {
    public static void main(String[] args) {
        Animale animale1 = new Cane();
        Animale animale2 = new Gatto();

        animale1.emettiSuono(); // Output: Il cane abbaia
        animale2.emettiSuono(); // Output: Il gatto miagola
    }
}
```

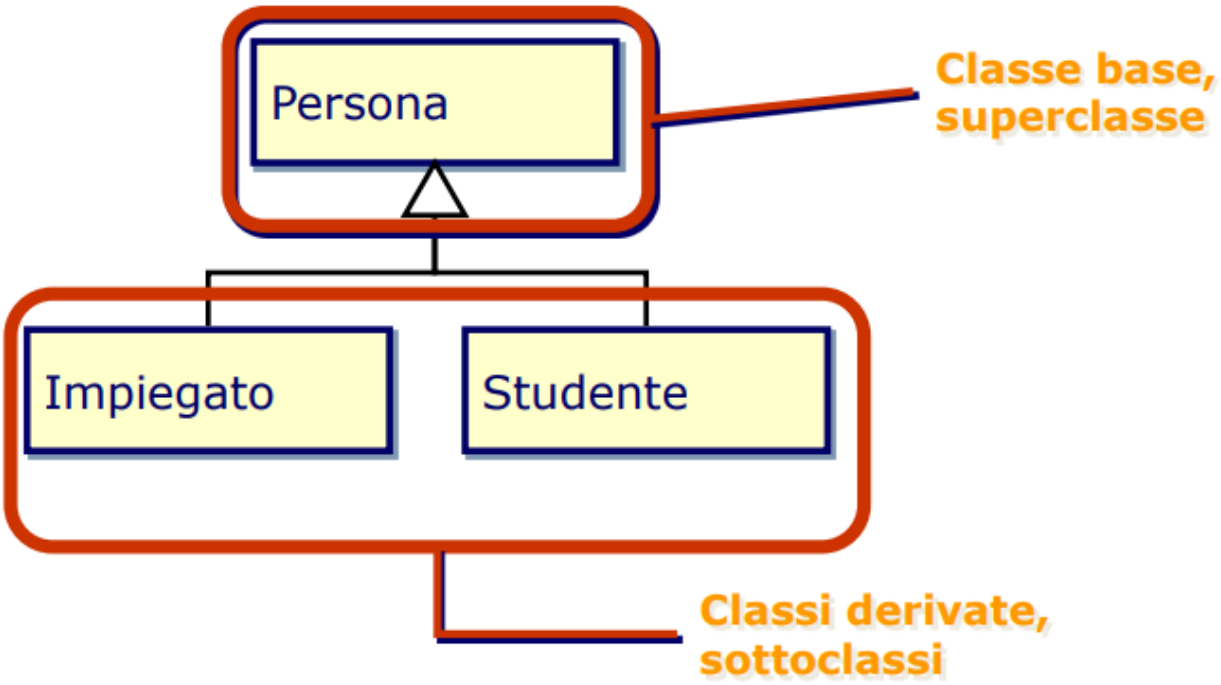
In questo esempio:

1. Abbiamo due classi derivate, `Cane` e `Gatto`, che estendono la classe `Animale` e ridefiniscono il metodo `emettiSuono()` con il proprio comportamento specifico. Il

cane abbaia e il gatto miagola.

2. Nella classe `EsempioPolimorfismo`, nel metodo `main()`, creiamo due oggetti `animale1` e `animale2` di tipo `Animale`, ma li istanziamo rispettivamente con un oggetto `Cane` e un oggetto `Gatto`.
3. Quando chiamiamo il metodo `emettiSuono()` su `animale1` e `animale2`, viene invocato il metodo specifico della classe effettiva dell'oggetto (rispettivamente `Cane` e `Gatto`), grazie al polimorfismo.

Esempio ereditarietà:



Astrazione = più significati alle classi derivate (= non dipendere dai parametri)

Nota bene:

- Da sotto passiamo facilmente a sopra
  - Sottoclasse è facilmente la superclasse
  - SUV può essere Macchina
- Ma non è vero il contrario
  - Quindi, che sottoclasse possa essere superclasse
  - SUV non può prendere il posto di macchina
- Esempio
  - Un'automobile è un veicolo

- Un veicolo non è (necessariamente) un'automobile
- Hanno significati diversi!

Ereditarietà in codice

- Uso la keyword `extends`

```
class Animal{}; //superclasse

class Cat extends Animal{}; //sottoclasse = estensione della classe base
```

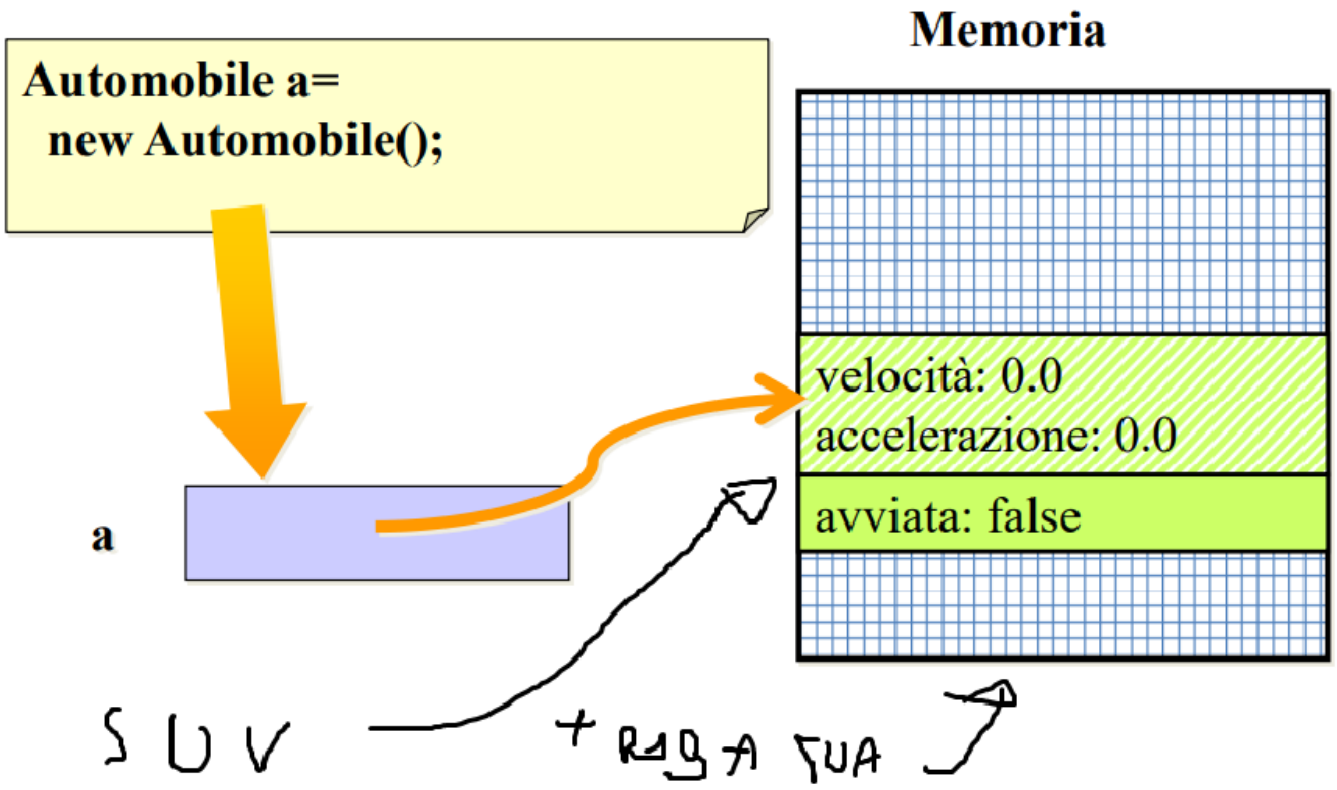
Esempio:

Veicolo.java

```
public class Veicolo {
        private double velocità;
        private double accelerazione;
        public double getVelocità() {…}
        public double getAccelerazione() {…}
}
```

Automobile.java

```
public class Automobile
    extends Veicolo {
          private boolean avviata;
          public void avvia() {…}
}
```



= risparmio memoria

Costruttori

- Dare un valore agli attributi della classe
- = costruirli

```
class Animal{
        int age;
        String gender;

        // di default esiste il costruttore anonimo (o di default)
        // istanzia i campi a nullo
        Animal(){
                age = 0;
                gender = "";
        }

        Animal(int age, String gender){
                this.age = age;
                this.gender = gender;
        }
};
```

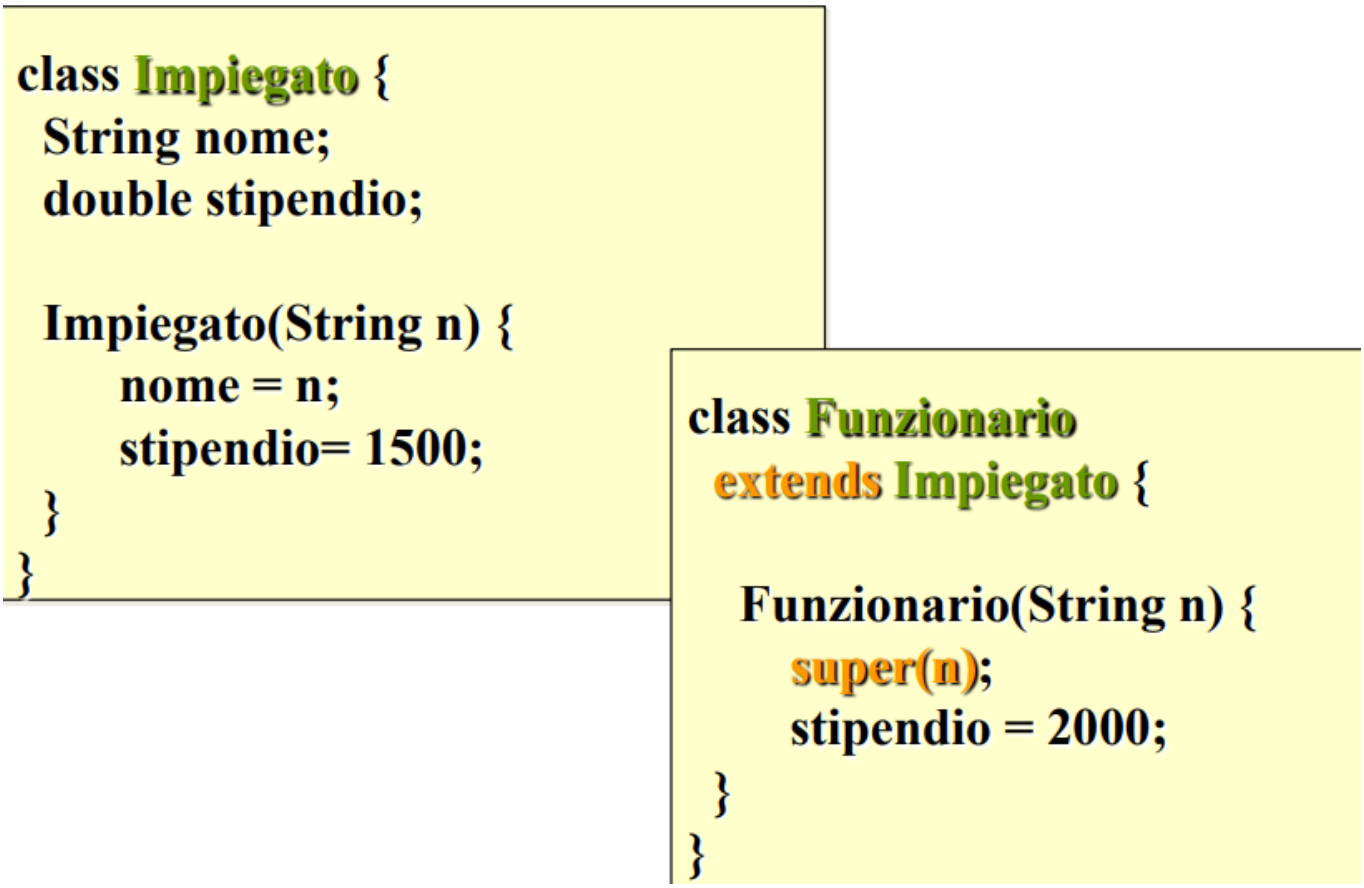
```
class Duck{
        String color;

        Duck(int age, String gender, String color){
                super(age, gender);
                // usi i campi della superclasse per costruire
                this.color = color;
        }
};

// esempio di utilizzo

Animal a(40, "F");
Duck d(50, "M", "White");
```

Esempio:



Visibilità:

- le derivate vedono tutti i componenti della base
    - anche se private

- `protected` = Vedono solo le superclassi, ma non all'esterno (via di mezzo tra `public` e `private`)

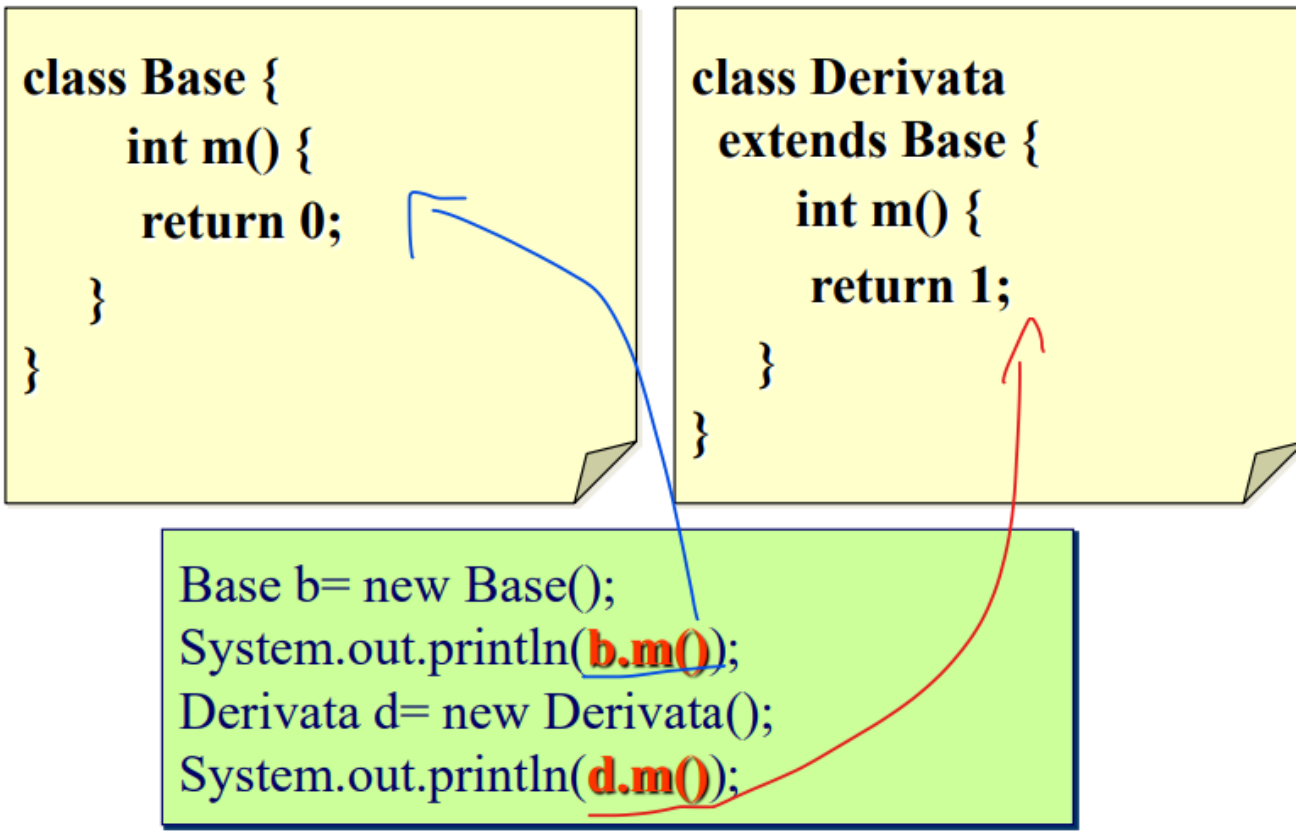
Ridefinizione metodi

```
class Shape{
        public:
                void draw();
};

class Square extends Shape(){
        public:
                // ridefinizione (overriding) = stesso nome/stessi parametri/stesso tipo di ritorno
                void draw(){
                        System.out.println("Sono un quadrato");
                }
}
```

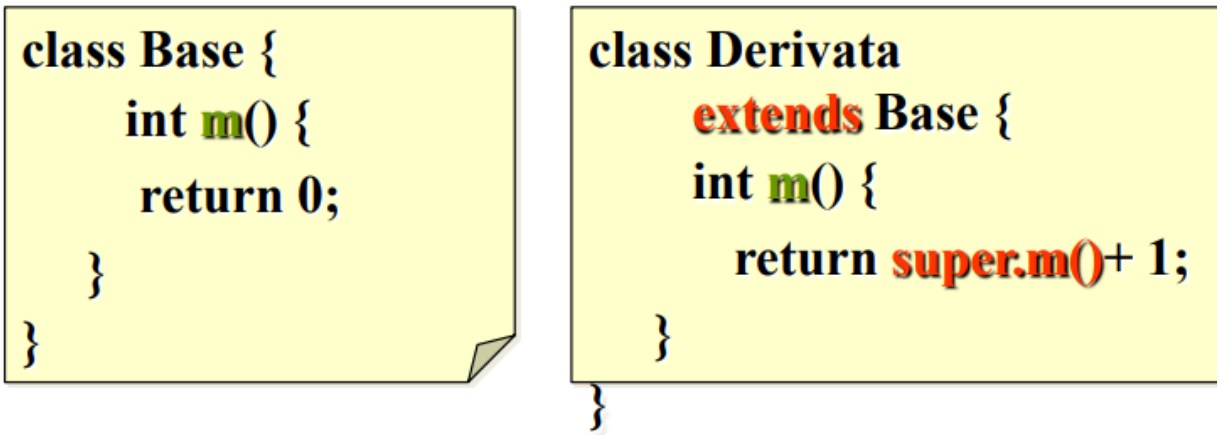
- Per le istanze della sottoclasse, il nuovo metodo nasconde l'originale
  - = tendenzialmente, se sei sotto, usi "il tuo"

Esempio:



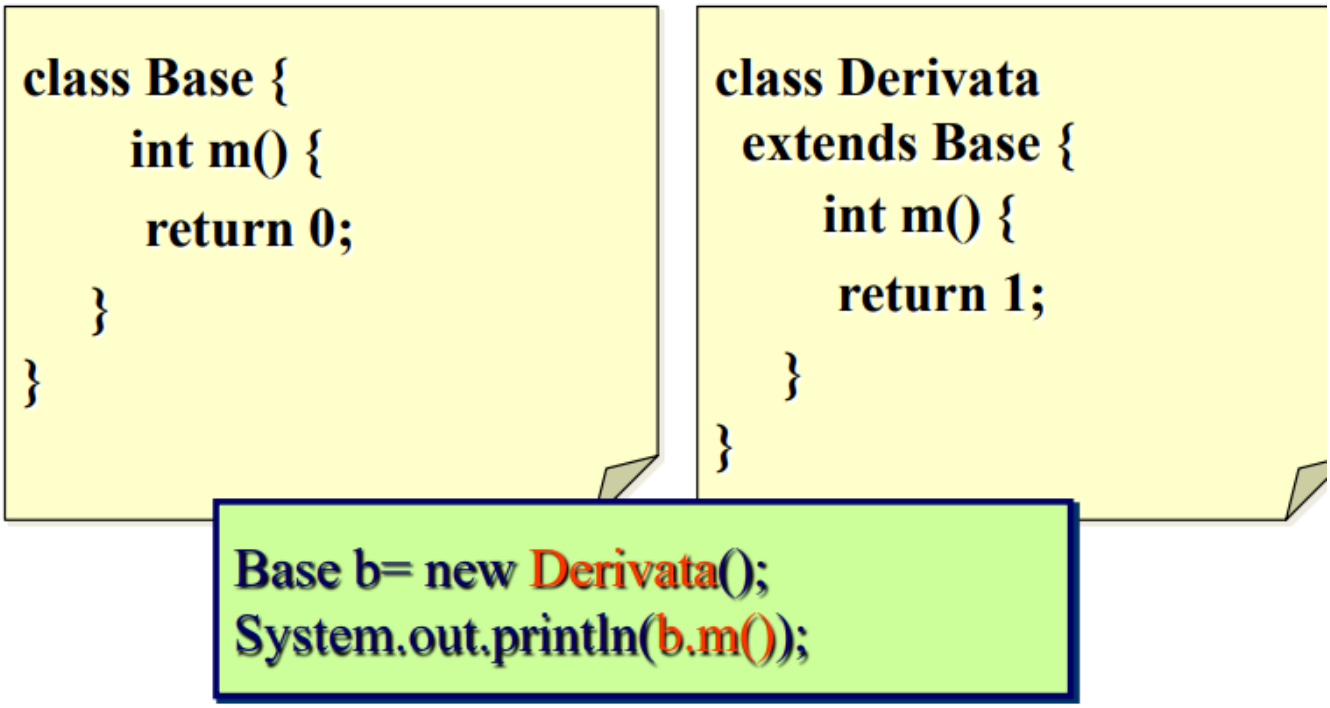
A volte, una sottoclasse vuole “perfezionare” un metodo ereditato (lo chiama con "super" e "aggiunge roba"):

– Per invocare l’implementazione presente nella super-classe, si usa il costrutto
super.<*nomeMetodo*> (…)



Polimorfismo

Esempio: posso assumere facilmente la forma della classe derivata



Una variabile:

- chiama sempre il tipo vicino
- ci possono essere più sottoclassi

Per sfruttare questa tecnica:

- Si definiscono, nella super-classe, metodi con implementazione generic
  - sostituiti, nelle sottoclassi, da implementazioni specifiche
- Si utilizzano variabili aventi come tipo quello della super-classe