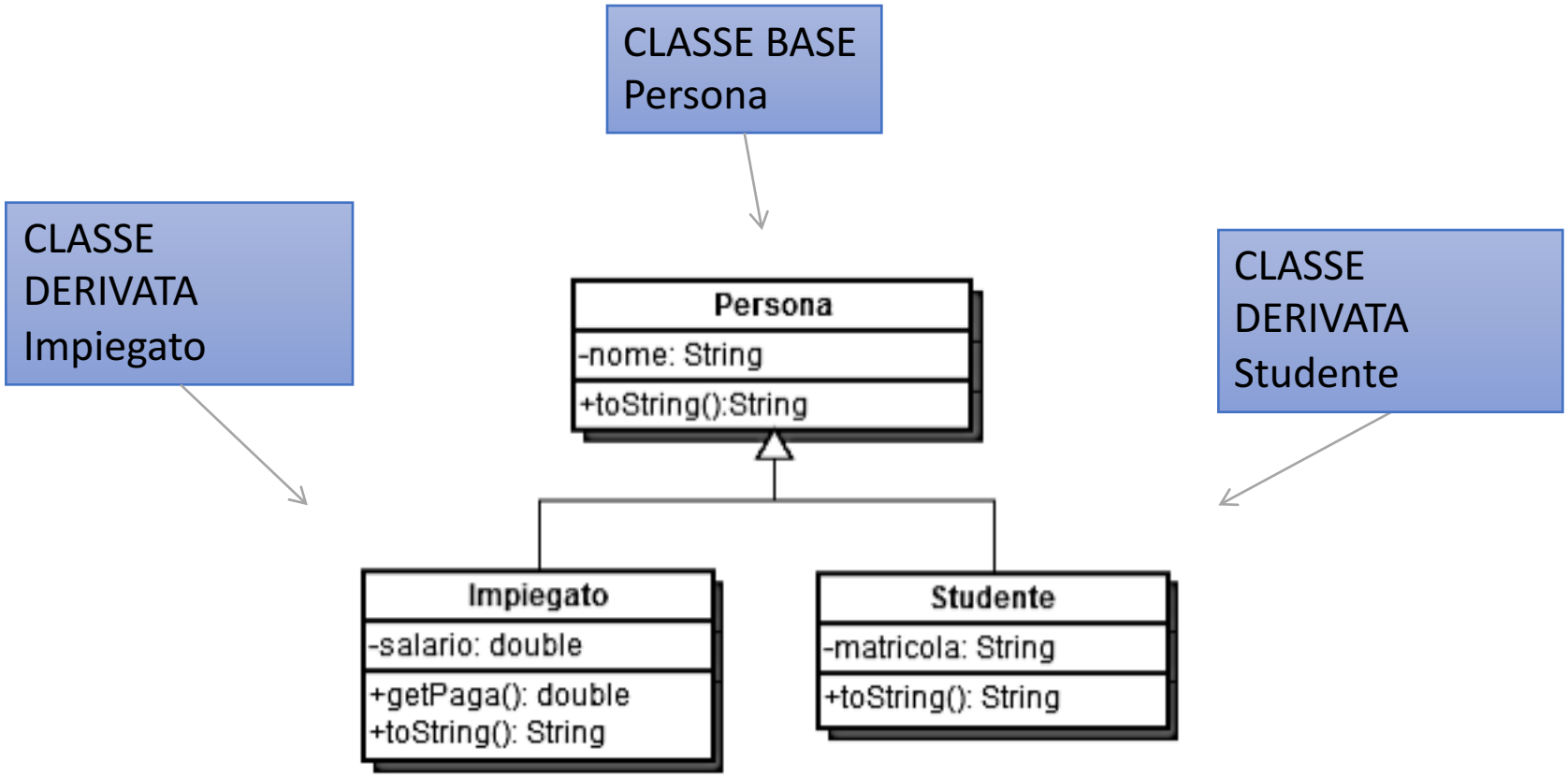
Ereditarietà e Polimorfismo

Fondamenti di Informatica A-K

Esercitazione 6

- Introduzione al calcolatore e Java
- Linguaggio Java, basi e controllo del flusso
- I metodi: concetti di base
- Stringhe ed array
- Classi e oggetti, costruttori, metodi statici, visibilità
- Eclipse, ereditarietà e polimorfismo

Ereditarietà di base



Ereditarietà in teoria

- **Ogni classe definisce un tipo:**
  - *Un oggetto, istanza di una sottoclasse, è compatibile con il tipo della classe base*

IL CONTRARIO NON È VERO!

Ad esempio:

- Un impiegato è una persona ma una persona non è (necessariamente) un impiegato
- Un'automobile è un veicolo ma un veicolo non è (necessariamente) un'automobile

Esempio in codice

```java
public class Persona {

    private String nome;
    public Persona(String nome) { this.nome = nome; }
    public String toString()    {return "Mi chiamo " + this.nome;}
}
```

```java
public class Studente extends Persona{
    private String matricola;
    public Studente(String nome, String matricola) {
        super(nome);
        this.matricola = matricola;
    }
    @Override
    public String toString() {
        return super.toString() +
        ", numero di matricola: " + this.matricola;
    }
}
```

```java
public class Impiegato extends Persona {
    private double salario;
    public Impiegato(String nome, double salario) {
        super(nome);
        this.salario = salario;
    }
    public double getPaga() { return salario;}
    @Override
    public String toString() {
        return super.toString() + " e guadagno " + getPaga() + "€"; }
}
```

Terminologia di ereditarietà

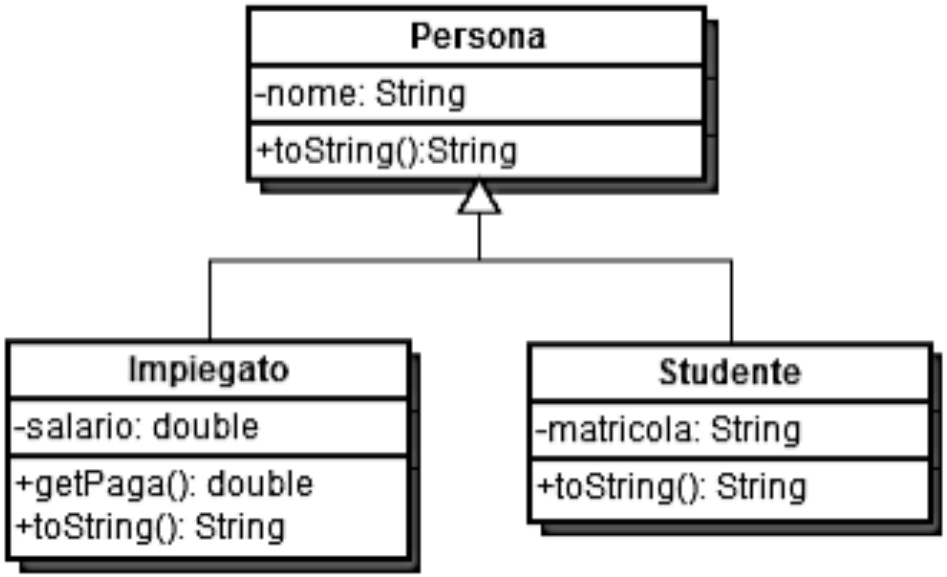
Parola chiave ***extends***
Specifica da quale classe eredita.
Nell'esempio, Studente eredita da Persona.

Parola chiave ***super***
Consente di invocare un metodo, un costruttore o un attributo della classe base purché non `private`.

Annotazione ***@Override***
Permette di ridefinire un metodo della superclasse a condizione che abbia stesso nome, parametri e tipo di ritorno.

```
public class Studente extends Persona{
    private String matricola;
    public Studente(String nome, String matricola) {
        super(nome);
        this.matricola = matricola;
    }
    @Override
    public String toString() {
        return super.toString() +
        ", numero di matricola: " + this.matricola;
    }
}
```

Polimorfismo



- OK

```
Persona p = new Studente("Mario", 123456);
System.out.println(p.toString());
```

- KO

```
Studente s = new Persona("Mario");
Studente s = new Impiegato("Mario", 1400);
```

Esercizio «`Bicicletta`»

- Creare una classe `Bicicletta` che abbia i seguenti attributi
  1. `double speed`
  2. `int marcia`
- Ed i seguenti metodi:
  1. Un costruttore `Bicicletta()` che inizializzi tutti i parametri a zero.
  2. Un metodo `speedUp()` per incrementare la velocità di 1 km/h
  3. Un metodo `speedDown()` per decrementare la velocità di 1 km/h
  4. Un metodo `setMarcia(int nuova_marcia)` per cambiare marcia.
  5. Un metodo `toString()` che stampi il valore di tutti i parametri in sequenza.

Esercizio «`MountainBike` »

- Creare la classe `MountainBike extends Bicicletta` aggiungendone i seguenti attributi:
  1. `boolean ammortizzatori`
- Aggiungere i seguenti metodi:
  1. Un costruttore `mountainBike`() che inizializzi tutti i parametri di bicicletta a zero ed il tipo di ammortizzatori a `false` (utilizzare `super`).
  2. Un metodo `getAmmortizzatori` per ottenere il valore di `ammortizzatori`.
- Ridefinire il metodo `toString()` per la stampa dei valori degli attributi della `Bicicletta` aggiungendo il valore di `ammortizzatori.`

Esercizio «`FatBike`»

- Creare la classe `FatBike extends Bicicletta` aggiungendo i seguenti attributi:
  1. `int spessorePneumatico`
- Ed i seguenti metodi:
  - Un costruttore `FatBike()` che inizializzi tutti i parametri di bicicletta e `spessorePneumatico` (utilizzare `super`).
  - Un metodo `getSpessorePneumatico` per ottenere il valore dello spessore dello pneumatico.
- Ed infine Ridefinisca il metodo `toString()` per la stampa dei valore degli attributi di `Bicicletta` aggiungendo il valore di `spessorePneumatico`.

Classi Astratte

- Abbiamo detto che le classi rappresentano entità del mondo reale, ma non tutte le entità sono reali:
  - Ad esempio: non esiste concretamente il generico animale ma esistono gli specifici animali!
- Una classe astratta si definisce in questi termini:

```
public abstract class Animale {
   private String nome;
   protected String verso;
   public abstract String verso();
   public abstract String si_muove();
   public abstract String vive();
}
```

- Il concetto così espresso è che:
  - ogni animale "reale" può fare un verso, può muoversi, e può dire in che ambiente vive
  - ma non si può, in generale, precisare come, perché questo ovviamente dipende dallo specifico animale

Classi astratte, precisazioni

- **Una classe avente anche solo un metodo `abstract` è astratta**, e deve essere dichiarata **`abstract`** essa stessa.
  - *Una classe* **`abstract`** *può però anche non avere metodi dichiarati* **`abstract`** (ma resta comunque **`abstract`**, quindi **è impossibile istanziarla**)
- **Una sottoclasse di una classe `abstract`, se non ridefinisce tutti i metodi che erano `abstract` nella classe base, è anch'essa `abstract`.**

Interfacce

- Una interfaccia è l'equivalente di una classe con tutti metodi **`abstract,`** cioè *non definiti, ma solo dichiarati.*
- Come tale :
  - Si limita a dichiarare i metodi, senza implementarli (tutti i metodi di un'interfaccia sono implicitamente abstract)
  - **Non può definire variabili (può definire costanti, cioè static final)**

Un'interfaccia importante, Comparable

- Quest'interfaccia :
  - ci è utile per definire dei criteri di ordinamento tra un oggetto ed un altro.
  - Contiene al suo interno un solo metodo da ridefinire (`compareTo`)
- API → http://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html

```
public interface Comparable<T>{
    int compareTo(T o);
}
```

Esercizio «`Personale`» 1/2

- Realizzare la seguente tassonomia:
  - Creare un'interfaccia `Impiegato` che abbia la dichiarazioni dei seguenti metodi:
    1. `public String getNome();`
    2. `public String getIndirizzo();`
    3. `public String getTelefono();`
    4. `public double getPaga();`
  - Creare una classe astratta `Impiegato` che implementi l'interfaccia ridefinendo:
    1. Tutti i metodi sopraelencati all'infuori di `getPaga();`
    2. Inserendo gli attributi:
       - `String nome;`
       - `String indirizzo;`
       - `String telefono;`
  - Creare le tre classi descritte nella foto alla slide 17 che estendano la classe `Impiegato`
    1. Specializzino il metodo astratto `getPaga()` della superclasse
    2. Aggiungano gli eventuali metodi e attributi come da diagramma

Esercizio «`Personale`» 2/2

- Il metodo `getPaga()` cambia in ognuna delle tre classi:
  1. Nella classe `Dipendente()` il metodo restituisce il valore della paga mensile del dipendente.
  2. Nella classe `Giornaliero()` il metodo restituisce il valore della paga, data dal prodotto tra la paga giornaliera ed il numero di giorni lavorativi.

     ```
     double getPaga() {
          return (paga_base * giorni_lavorativi);
     }
     ```
  3. Nella classe `Volontario()` il metodo restituisce zero.

Tassonomia «`Personale`»