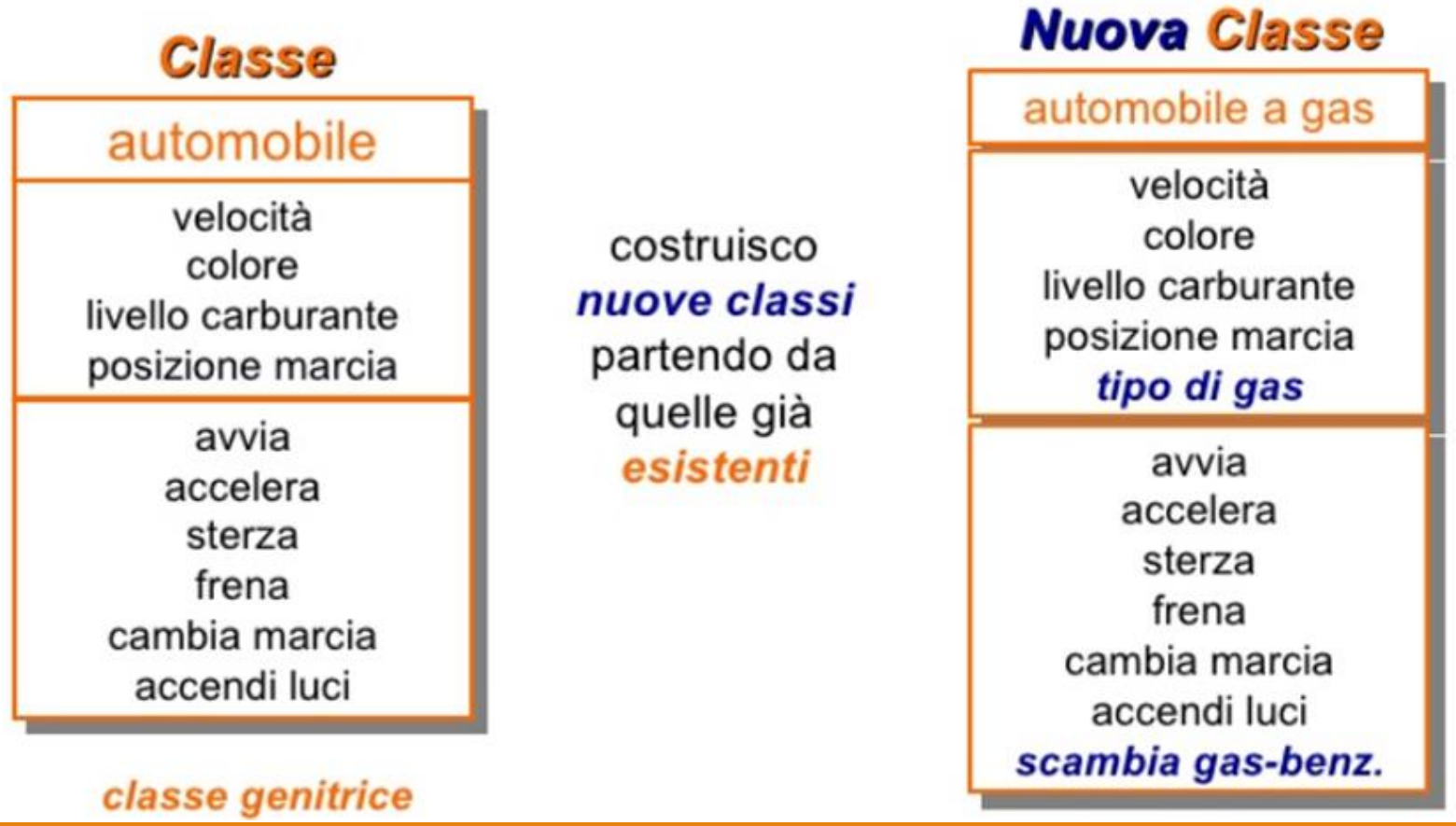
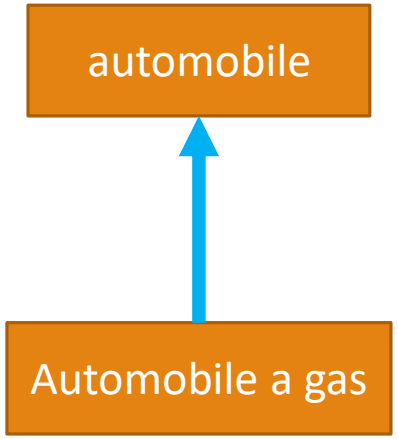
Classi

- **Ereditarietà e gerarchie**
- **Tipi di ereditarietà**
- **POLIMORFISMO**
  - **Overloading**
  - **Override**

EREDITARIETA’:

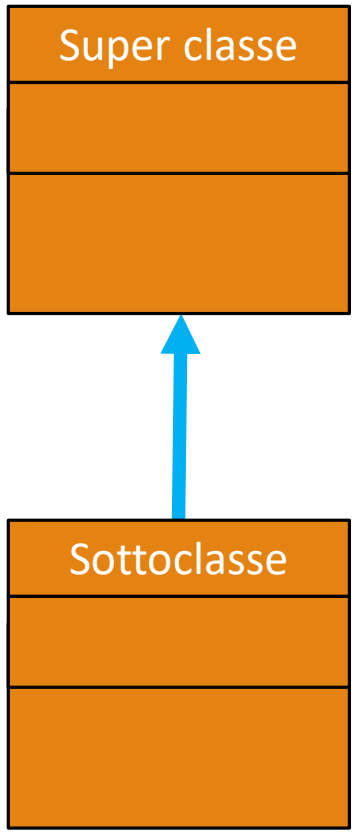
L’ereditarietà **è un concetto OOPS in cui un oggetto acquisisce le proprietà e i comportamenti dell’oggetto genitore**. Sta creando una relazione genitore-figlio tra due classi. Offre un meccanismo robusto e naturale per l’organizzazione e la struttura di qualsiasi software.





la classe AUTOMOBILE A GAS
si dice **EREDITA (O DERIVA)**
dalla classe **AUTOMOBILE**

L'ereditarietà è un meccanismo fondamentale per il **riutilizzo del codice nella programmazione a oggetti**. Infatti grazie a questo meccanismo è possibile ***estendere e potenziare classi già esistenti***



In Java si usa l'ereditarietà quando occorre definire una classe i cui **oggetti realizzano delle funzionalità aggiuntive** (ovvero hanno una struttura più ricca di quella di una classe già definita)

La classe più "generale" (quella preesistente) si dice **Superclasse**.
La nuova classe più "specializzata" che eredita i campi e i metodi dalla classe preesistente viene detta **Sottoclasse**

La **sottoclasse** si dice **ESTENDE LA SUPERCLASSE**

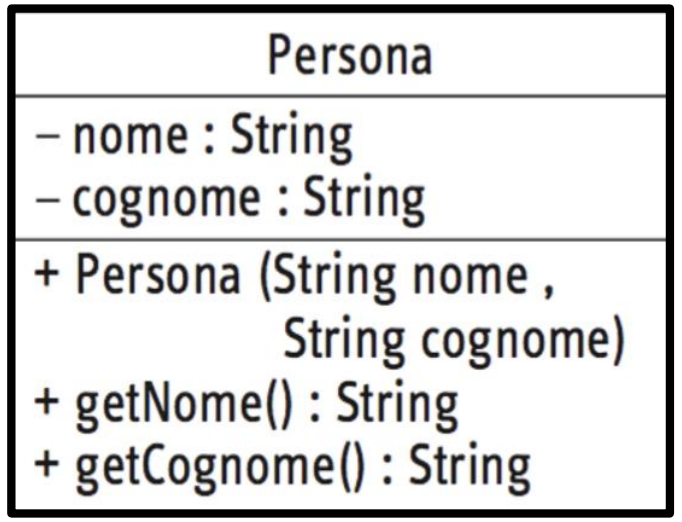
ESERCIZIO

Creare una classe di nome Persona con le variabili di istanza: nome, cognome di tipo stringa. La classe deve contenere un costruttore parametrico; i metodi ***getNome()*** e ***getCognome()***.

```
class Persona {
    private String nome;
    private String cognome;

    public Persona(String nome,String cognome){
        this.nome=nome;
        this.cognome=cognome;
    } //costr

    public String getNome(){
        return nome;
    }
    public String getCognome(){
        return cognome;
    }
} //class
```



ESERCIZIO

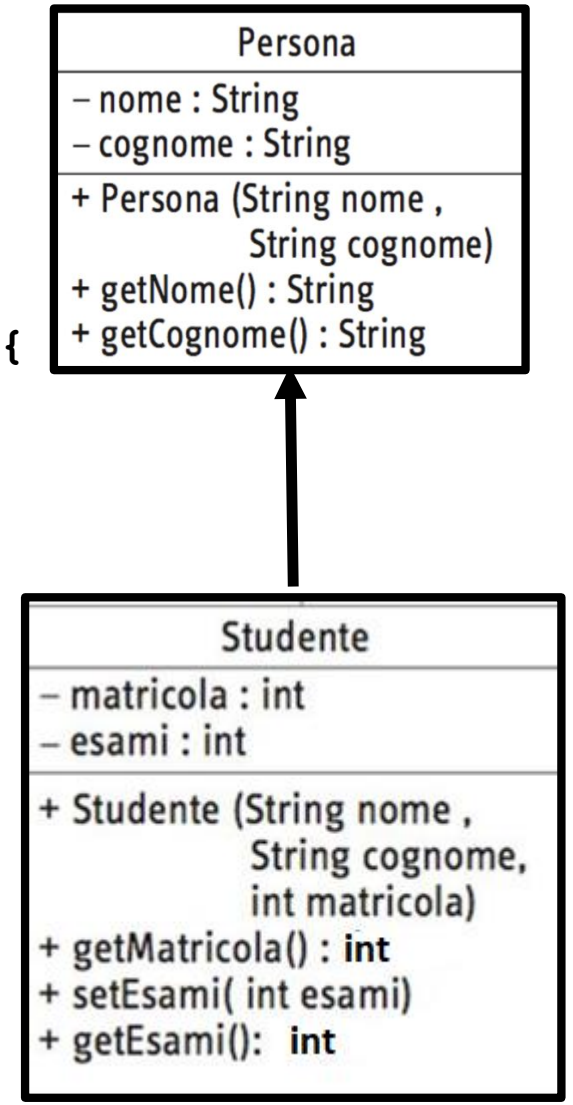
Costruire una **sottoclasse** di **Persona**, chiamata **Studente**, che contiene le variabili di istanza: **matricola** ed **esami**, *che registra il numero di esami sostenuti*, e devono essere entrambe di tipo intero. La sottoclasse deve contenere un **costruttore parametrico** ed i metodi ***setMatricola()*** e ***getMatricola()***

```
class Studente extends Persona{
    private int matricola;
    private int esami;

    public Studente(String nome, String cognome, int matricola){
        super (nome, cognome);
        this.matricola = matricola;
    } //costr
    public int getMatricola(){
        return matricola;
    }
    public int getEsami(){
        return esami;
    }
    public void setEsami(int esami){
        this.esami = esami;
    }
} //class
```

costruttore della classe genitrice

L'espressione ***super(...)*** invoca il costruttore della classe genitrice (***nel nostro caso Persona***), quindi deve avere i parametri attuali corrispondenti in numero e tipo a quelli formali.



I MODIFICATORI DI ACCESSO

I MODIFICATORI DI ACCESSO costituiscono le regole per accedere ai membri e ai metodi di una classe.

Il modificatore di accesso protected rende accessibile un campo a tutte le sottoclassi, presenti e future e ecostituisce perciò un permesso di accesso valido per ogni possibile sottoclasse che possa essere definita.

Ricapitolando:

private: disponibile solo dall'ambito della stessa classe
protected: disponibile nell'ambito delle sottoclassi
public: disponibile per qualsiasi altra classe

Il **default**… significa, disponibile per qualsiasi altra classe appartenente **allo stesso package**

La parola chiave super e this

La parola chiave **super**

1. nella forma `super(...),` invoca un costruttore della classe base
2. nella forma `super.val`, consente di accedere al campo val della classe base (*sempre che esso non sia private*)
3. nella forma `super.metodo(),` consente di invocare il metodo ***metodo()*** della classe madre (*sempre che questo metodo non sia private*)

super fa riferimento all'istanza stessa come se appartenesse alla superclasse (***si usa per evitare ambiguità e per specializ zare metodi***)

```
class ContoCorrente {
  float saldo;

  void deposita(float somma) {
    saldo += somma;
  }
}
```

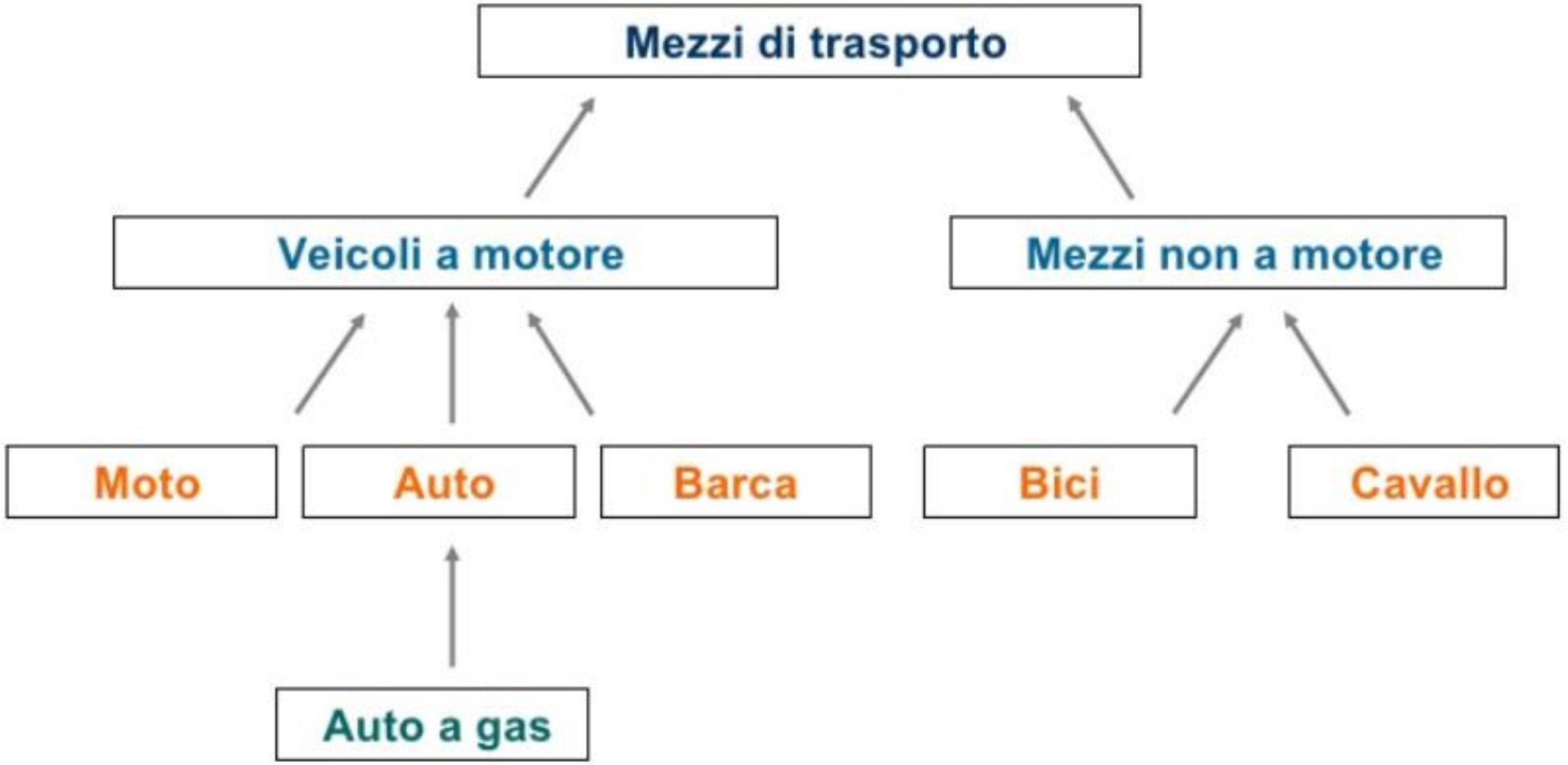
```
class ContoAziendale {
  RegistroCassa registro;

  void deposita(float somma) {
    super.deposita(somma);
    registro.annotaEntrata(somma);
  }
}
```

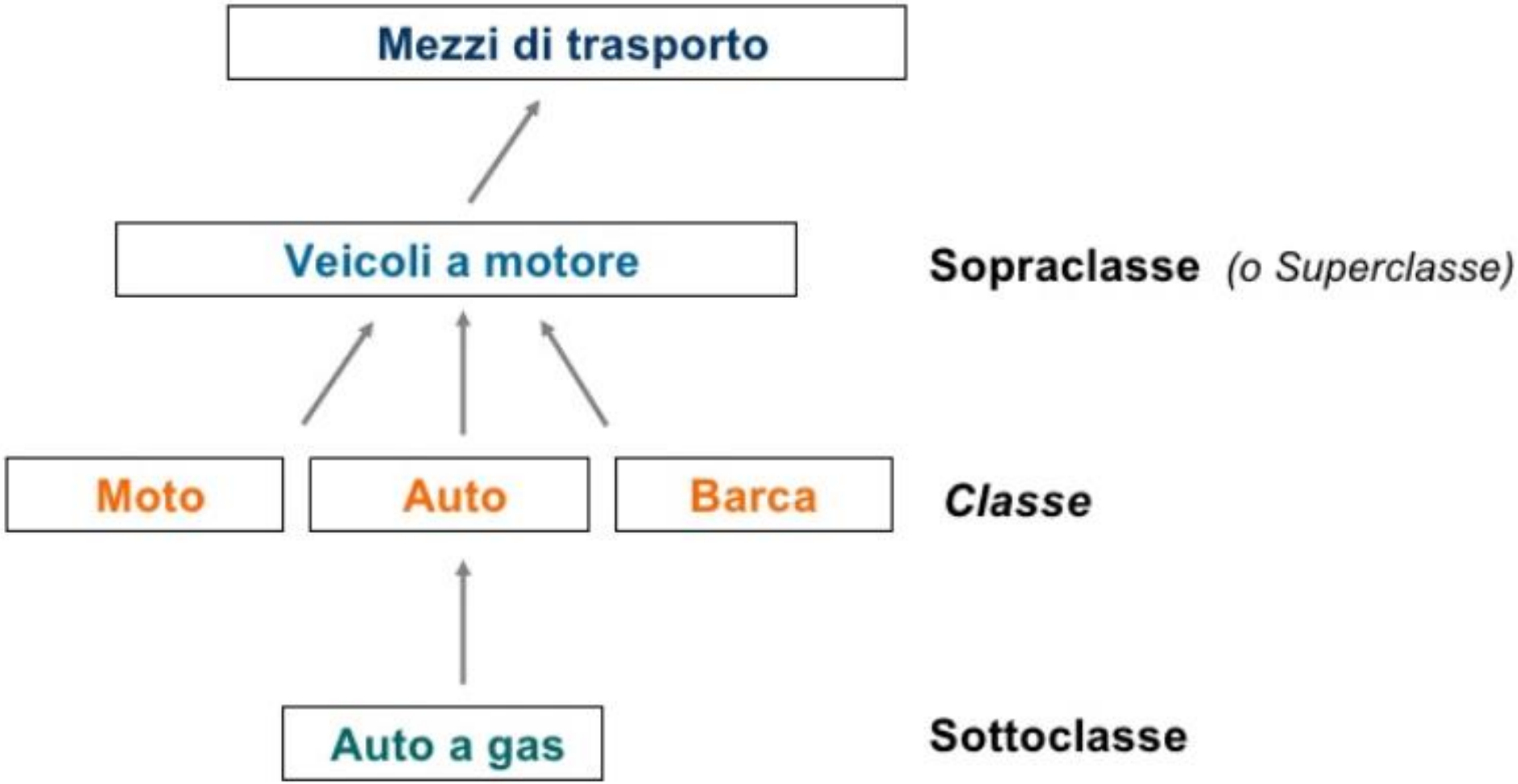
this fa riferimento all'istanza stessa (***si usa per evitare ambiguità***)

```
class Ciclista extends Persona {
        Bicicletta bici;
        boolean dimmiSeTua (Bicicletta bici) {
                return this.bici = bici;
        }
}
```

Gerarchia di ereditarietà

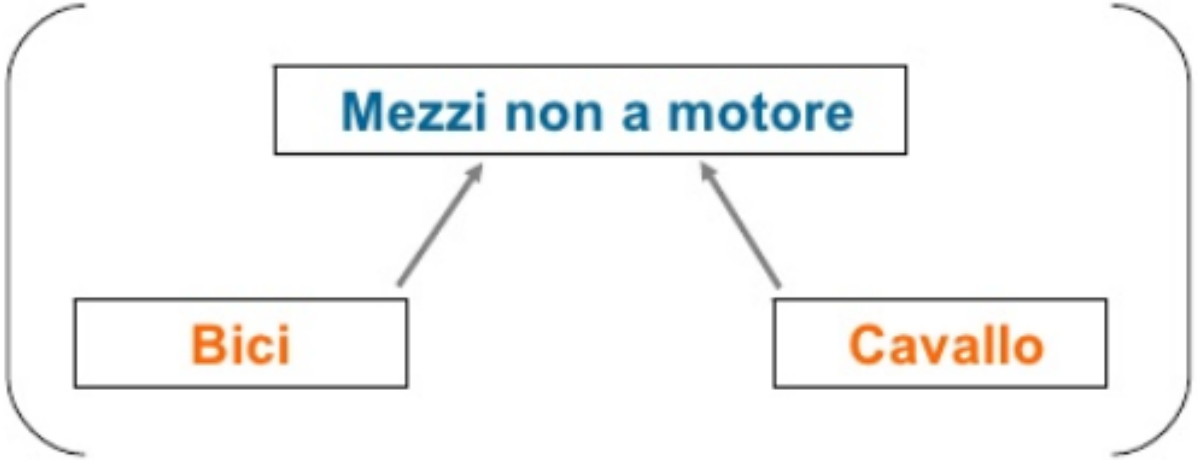


Gerarchia di ereditarietà



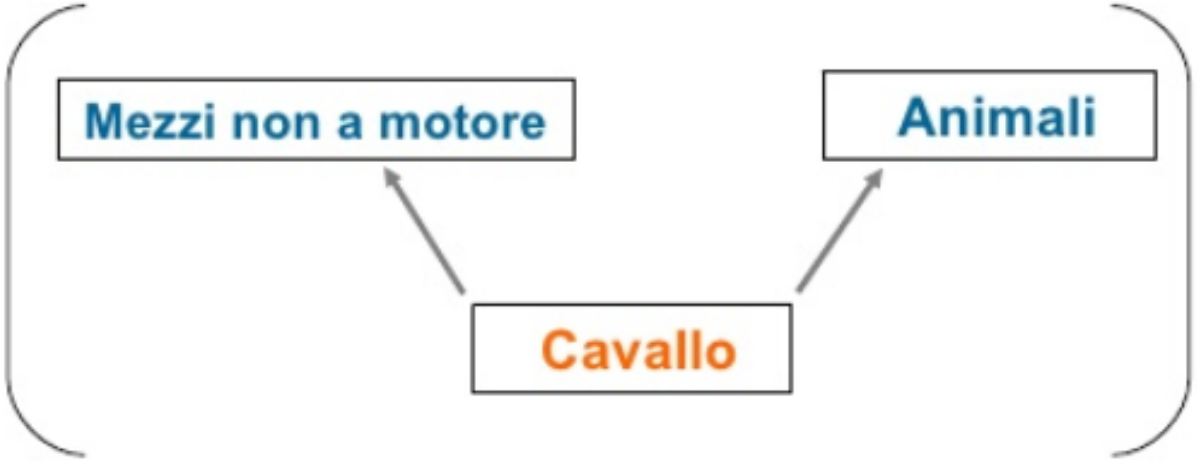
Tipi di ereditarietà

Eredità singola



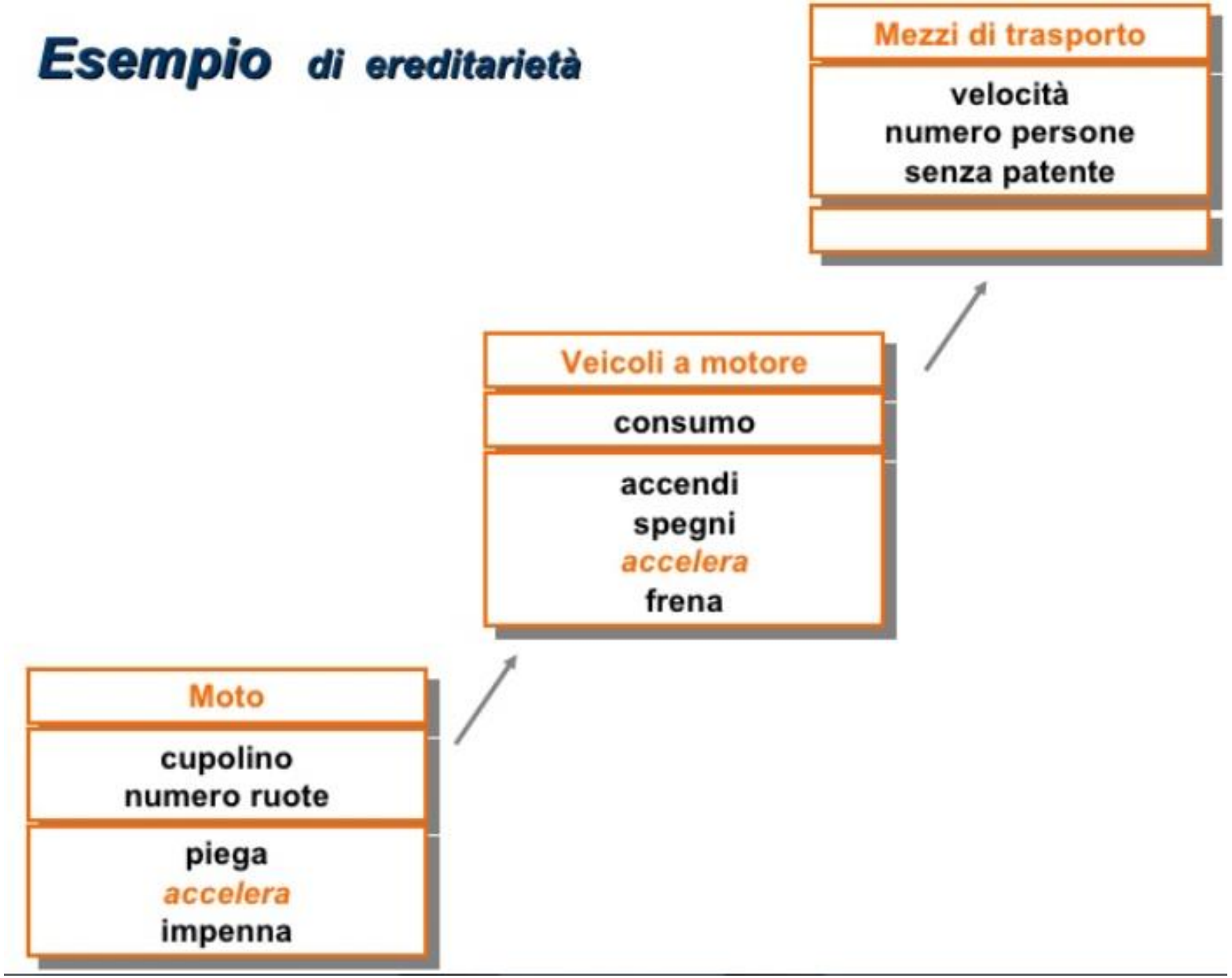
In JAVA esiste solo l'ereditarietà SINGOLA

Eredità multipla



Il C++ supporta l'ereditarietà MULTIPLA

Esempio di ereditarietà



IL POLIMORFISMO

Il termine polimorfismo in JAVA si riferisce alla possibilita che hanno i metodi ad assumere piu forme e quindi a comportarsi diversamente a seconda di come vengono chiamati.

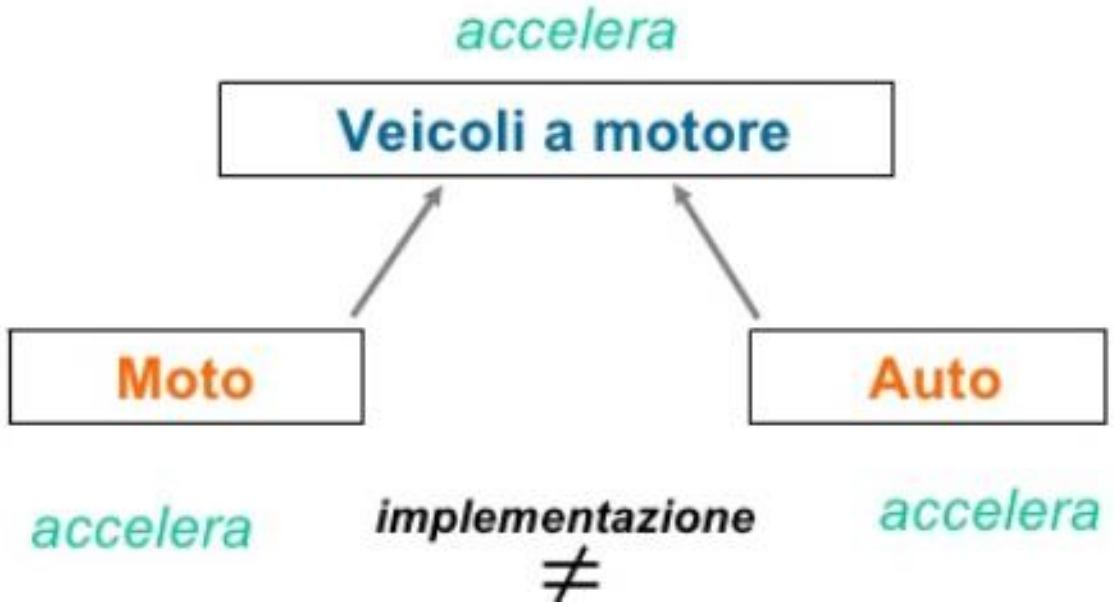
Il polimorfismo si realizza attraverso **l'OVERRIDING** e **l'OVERLOADING**

Con ***overload*** si intende *la possibilità*, consentita al programmatore, *di definire, in una stessa classe, più metodi col medesimo nome, ma* ***firma*** *differente*. Ciò è utile quando si desidera definire più metodi che possiedano la medesima funzionalità, ma con differente campo di applicazione (cioè con una diversa lista di parametri).

Per quanto riguarda l'**override**, invece, *si intende la caratteristica delle classi derivate (ereditarietà) di poter ridefinire un metodo pubblico ereditato dalla superclasse.*

Polimorfismo . 1°

Indica la possibilità per i ***metodi*** di ***assumere forme diverse*** (cioè *implementazioni diverse)* all'interno della gerarchia delle classi



Polimorfismo . 2°

Indica la possibilità per i ***metodi*** di ***assumere forme diverse*** (cioè *implementazioni diverse)* all'interno della stessa classe

Classe Auto

- ***Frena*** ( )
- ***Frena*** (mano, 3)
- ***Frena*** (motore)

= ***nome***

≠ ***parametri***

(overloading dei metodi)

L' OVERRRIDING di un metodo

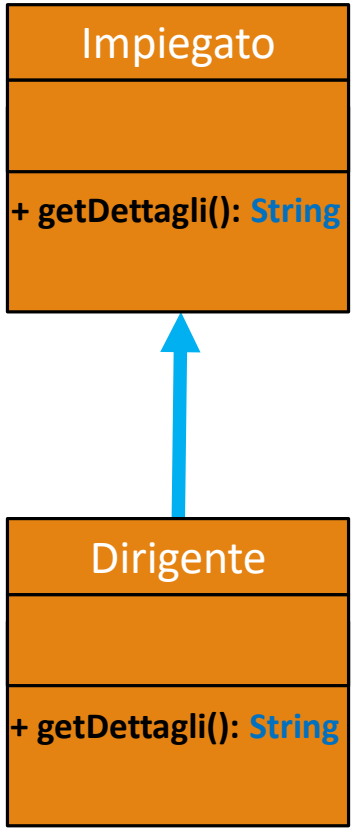
L'overriding rappresenta una importante risorsa della programmazione ad oggetti, indica la caratteristica delle sottoclassi di poter ridefinire un metodo ereditato da una superclasse.
Ovviamente non si puo' parlare di overriding senza che ci sia l'ereditarietà

Supponiamo di avere due classi, una classe **IMPIEGATO** e una **DIRIGENTE** cosi descritte:

```
public class Impiegato {
    proteced String nome;
    protected double stipendio;
    protected Date dataNascita;

    public String getDettagli(){
        return "Nome: " + nome + "\n" +"Stipendio: " + stipendio;
    } //func
} //class
```

Supponiamo di voler costruire una classe DIRIGENTE che estende la classe IMPIEGATO

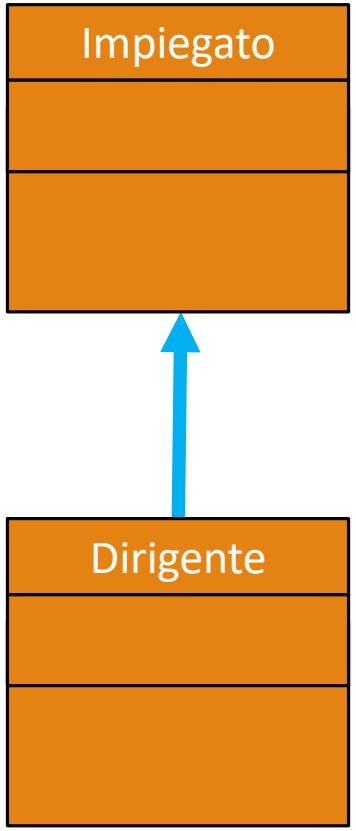


L' OVERRRIDING di un metodo

```
public class Dirigente{
    proteced String dipartimento;

    public String getDettagli(){
        return "Nome: " + nome + "\n"
               +"Stipendio: " + stipendio + "\n"
               + "Dirigente del Dipartimento: " + dipartimento;
    } //func
} //class
```

Quando viene istanziato un oggetto di classe **Dirigente** ed invocato il metodo ***getDettagli(),*** verrà automaticamente richiamato il metodo ridefinito (*overridden*) nella stessa classe Dirigente. *In assenza di overriding il metodo chiamato sarebbe stato quello ereditato dalla* ***superclasse***



Regola per L’ OVERRRIDING di un metodo

Nell’override è necessario rispettare le seguenti regole:

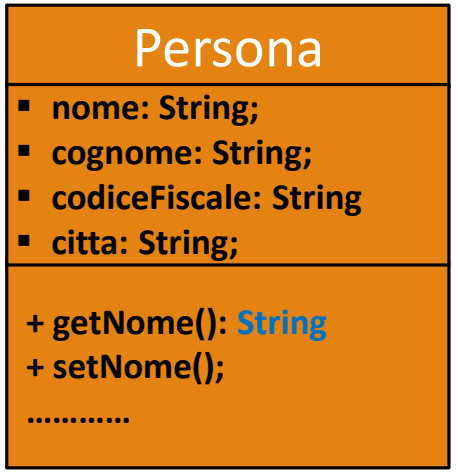
1. Il metodo ridefinito nella sottoclasse deve possedere la medesima ***firma*** di quello da riscrivere (altrimenti si opererebbe un ***overload***);
2. lo stesso tipo di ritorno;
3. lo stesso specificatore d’accesso.

ESERCIZIO:

Creare una classe di nome **Persona** con le variabili di istanza: ***cognome***, ***nome, codice fiscale e città*** di tipo **stringa**.

- La classe deve contenere un ***costruttore di default*** che inizializzi le variabili di istanza con NULL;
- ***un costruttore parametrico***;
- i ***metodi set e get*** ed un
- metodo chiamato **AnnoNascita** che a partire dal codice fiscale ricavi e restituisca l'anno di nascita di una persona.

soluzione



```
public class Persona {
    protected String cognome, nome, codiceFiscale, citta;

    Persona(){
            this.cognome = this.nome = this.codiceFiscale = this.citta = null;
    } //costr

    Persona(String n, String c, String cod, String ct){
            this.nome = n;
            this.cognome = c;
            this.codiceFiscale = cod;
            this.citta = ct;
    } //costr

    public String getNome() {
            return nome;
    } //func
    public void setNome(String n) {
            this.nome = n;
    } //func
    public String getCognome() {
            return cognome;
    } //func
```

```java
    public void setCognome(String c) {
        this.cognome = c;
    } //func

    public String getCodiceFiscale() {
        return codiceFiscale;
    }//func
    public void setCodiceFiscale(String cod) {
        this.codiceFiscale = cod;
    } //func

    public String getCitta() {
        return citta;
    }//func
    public void setCitta(String ct) {
        this.citta = ct;
    } //func

    //dal Codice fiscale si ricava la sesta e la settima (Da index a index-1) cifra/lettera
    //queste lettere rappresentano l'anno di nascita nel codice fiscale
    public String AnnoDiNascita(){
        String anno;
        anno= codiceFiscale.substring(6,8);
        return anno;
    } //func
} //class
```

ESERCIZIO:

Creare un'applicazione Java che istanzi un oggetto della classe Persona e ne visualizzi in seguito nome, cognome ed anno di nascita.

soluzione

```
public class Main {

    public static void main(String[] args) {
        Persona p = new Persona("Ciccio", "Sciascia", "CCISCI98A08A741S", "Roma");

        System.out.println("\n\nECCO I DATI INSERITI:");
        System.out.println("====================");
        System.out.println("NOME: "+p.getNome());
        System.out.println("CONOME: "+p.getCognome());
        System.out.println("CITTA: "+p.getCitta());
        System.out.println("ANNO DI NASCITA: "+p.AnnoDiNascita());
    } //main
} //class
```

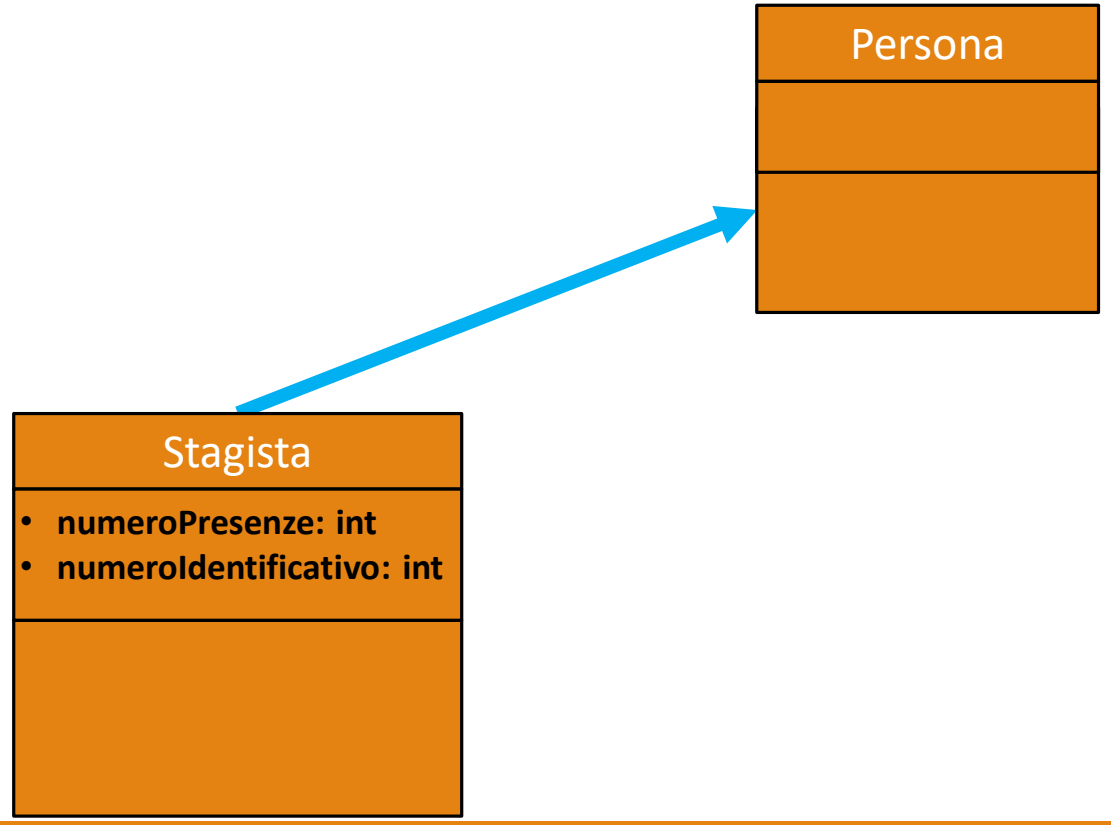
```
ECCO I DATI INSERITI:
====================
NOME: Ciccio
CONOME: Sciascia
CITTA: Roma
ANNO DI NASCITA: 98
```

ESERCIZIO: Costruire una sottoclasse di **Persona**, chiamata **Stagista**, che contiene 2 variabili di istanza di tipo intero:

1. ***numeroPresenze***, che registra il numero di ore di presenza, e
2. ***numeroIdentificativo*** che registra il numero identificativo.

La sottoclasse deve contenere un costruttore parametrico ed i metodi *set() e get()*..

soluzione



```java
public class Stagista extends Persona {
        private int numeroPresenze;
        private int numeroIdentificativo;

        Stagista (String n, String c, String cd, String ct, int np,int id){
                super(c,n,cd,ct);
                this.numeroPresenze = np;
                this.numeroIdentificativo = id;
        } //costr

        public void setNumeroPresenze(int npr){
                this.numeroPresenze = npr;
        } //func

        public int getNumeroPresenze(){
                return numeroPresenze;
        } //func

        public void setNumeroIdentificativo(int id){
                this.numeroIdentificativo = id;
        } //func

        public int getNumeroIdentificativo(){
                return numeroIdentificativo;
        } //func
}//class
```

ESERCIZIO:

Creare un'applicazione Java che istanzi tre oggetti di tipo Stagista li memorizzi in un array e visualizzare lo stagista più giovane (sulla base dell’anno di nascita maggiore).

soluzione

```java
public class Main {
	public static void main(String[] args) {
		Stagista st1, st2, st3;
		int massimo = 0;
		int posizione = 0;

		st1 = new Stagista ("Ciccio", "Sciascia", "SCICCI98A08A741S", "Roma", 34, 9742);
		st2 = new Stagista ("Pippo", "Rossi", "RSOPPI04F45A541H", "Roma", 132, 82345);
		st3 = new Stagista ("Salvo", "Petralia", "PTRSLA16T66P832R", "Milano", 7, 10345);

		Stagista stagisti[] = {st1,st2,st3};

		massimo = Integer.parseInt(stagisti[0].AnnoDiNascita());

		for(int i=0; i<stagisti.length; i++){
			if (massimo<Integer.parseInt(stagisti[i].AnnoDiNascita()))
				massimo = Integer.parseInt(stagisti[i].AnnoDiNascita());
			posizione = i;
		} //for

		System.out.println("\n\nECCO I DATI delLO STAGISTA PIU' GIOVANE: ");
		System.out.println("=======================================");
		System.out.println("\nNOME: "+stagisti[posizione].getNome());
		System.out.println("CONOME: "+stagisti[posizione].getCognome());
		System.out.println("CITTA: "+stagisti[posizione].getCitta());
		System.out.println("ID: "+stagisti[posizione].getNumeroIdentificativo());
		System.out.println("NUMERO DI PRESENZE: "+stagisti[posizione].getNumeroPresenze());
		System.out.println("ANNO DI NASCITA: "+stagisti[posizione].AnnoDiNascita());
	} //main
} //class
```

```
ECCO I DATI delLO STAGISTA PIU' GIOVANE:
=======================================

NOME: Petralia
CONOME: Salvo
CITTA: Milano
ID: 10345
NUMERO DI PRESENZE: 7
ANNO DI NASCITA: 16
```

```
class Triangolo{
    int l1,l2,l3;
    //costruttore per un trinagolo equilatero
    public Triangolo(int a){
        l1=l2=l3=a;
    }//costr
    //costruttore per un trinagolo isoscele
    public Triangolo(int a, int b){
        l1=a;
        l2=l3=b;
    }//costr
    // costruttore per un trinagolo scaleno
    public Triangolo(int a, int b, int c){
        l1=a;
        l2=b;
        l3=c;
    }//costr
    //si calcola il perimetro
    public int perimetro(){
        return (l1+l2+l3);
    }
}//class Triangolo
```

ESERCIZIO: *Definire una classe Triangolo che preveda la possibilità di calcolare il perimetro di qualunque triangolo*

ESERCIZIO: *Definire un programma che, facendo uso della classe* ***Triangolo*** *istanzi un oggetto opportuno della classe e ne visualizzi il perimetro*

```
public class TestTriangolo{
    public static void main(String[] args){
        Triangolo t1 = new Triangolo (a,b);
        int scelta, a,b,c;
        Scanner tastiera = new Scanner (System.in);

        System.out.println("Che tipo di Triangolo vuoi usare?");
        System.out.println("1.ISOSCELE");
        System.out.println("2.SCALENO");
        System.out.println("3.EQUILATERO");

        scelta = tastiera.nextInt();

        if (scelta ==1){
               System.out.println("Si vuole creare un triangolo
               isoscele, inserire le misure della base e del lato?\n");
               a = tastiera.nextInt();
               b = tastiera.nextInt();
               System.out.println("\nPerimetro = \n"+t1.perimetro());
        }
        else if (scelta ==2){ ……
```