UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

Fondamenti di Informatica (117IN)

A.A. 2022 / 2023

Lezione 11 - Classi e orientamento agli oggetti

Ereditarietà

Sylvio Barbon Junior
sylvio.barbonjunior@units.it

Sommario:

1) Lezione scorsa - Incapsulamento
2) Ereditarietà
3) Polimorfismo
4) Esempio

Sylvio Barbon Junior - sylvio.barbonjunior@units.it

1) L’esecuzione di un **programma a oggetti:**

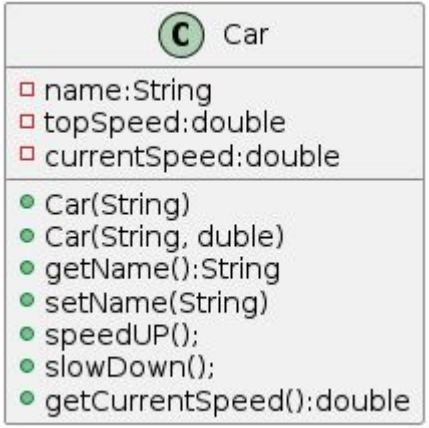
- L'**incapsulamento** consiste nella protezione dell’implementazione;
- L'**ereditarietà** permette essenzialmente di definire delle classi a partire da altre già definite;
- Il **polimorfismo** permette di scrivere **un client** che può servirsi di oggetti di classi diverse;

Sylvio Barbon Junior - sylvio.barbonjunior@units.it

Fondamenti di Informatica (117IN)

1) Incapsulamento

Esempi




```java
public class Car{
    private String name;
    private double topSpeed;
    private double currentSpeed;

    public Car(String name){
        this.name = name;
        this.currentSpeed = 0;
        this.topSpeed = 200.00;
    }

    public Car(String name, double topSpeed){
        this.name = name;
        this.topSpeed = topSpeed;
    }

    public String getName(){
        return(this.name);
    }

    public void setName(String name){
        this.name = name;
    }

    public void speedUp(){
        this.currentSpeed++;
    }

    public void slowDown(){
        this.currentSpeed--;
    }

    public double getCurrentSpeed(){
        return this.currentSpeed;
    }

}
```

attributi (variabile) privati

metodi costruttori (overload) - public

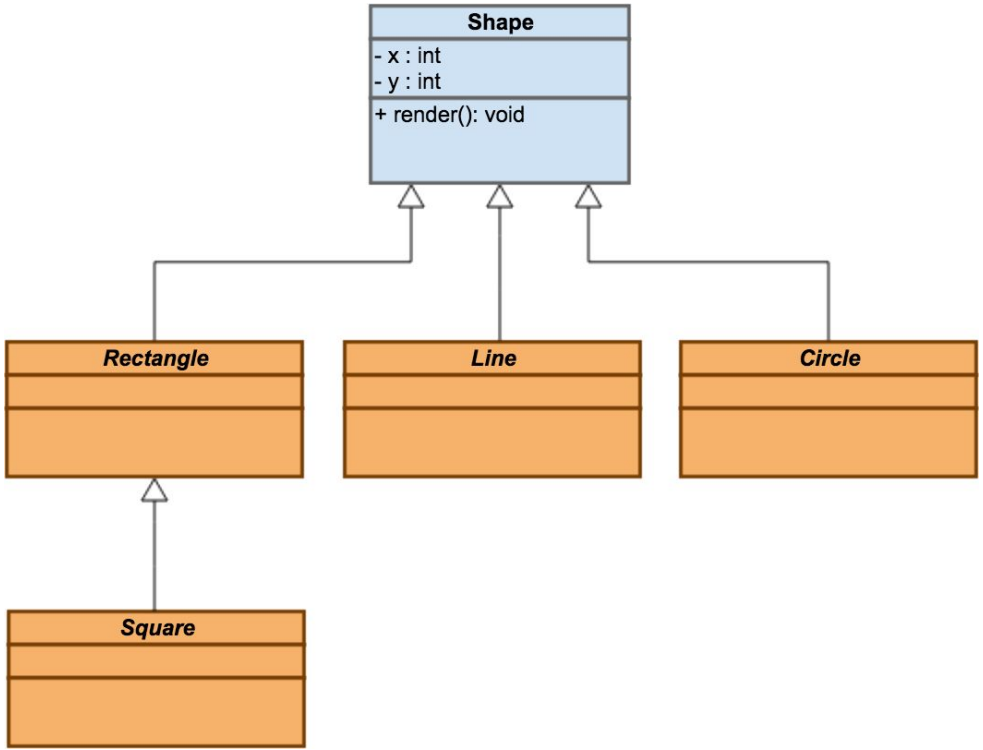
return con parenthesis

metodi "get" per recuperare i valori incapsulati - public

metodi "set" per fare modifica nelle variabile - public

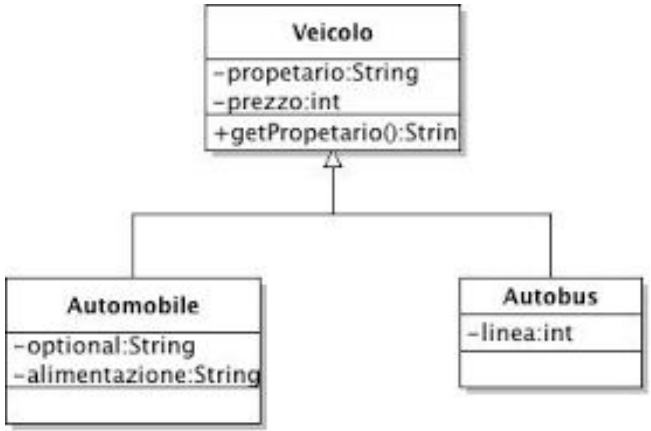
return senza parenthesis

2) **Ereditarietà**

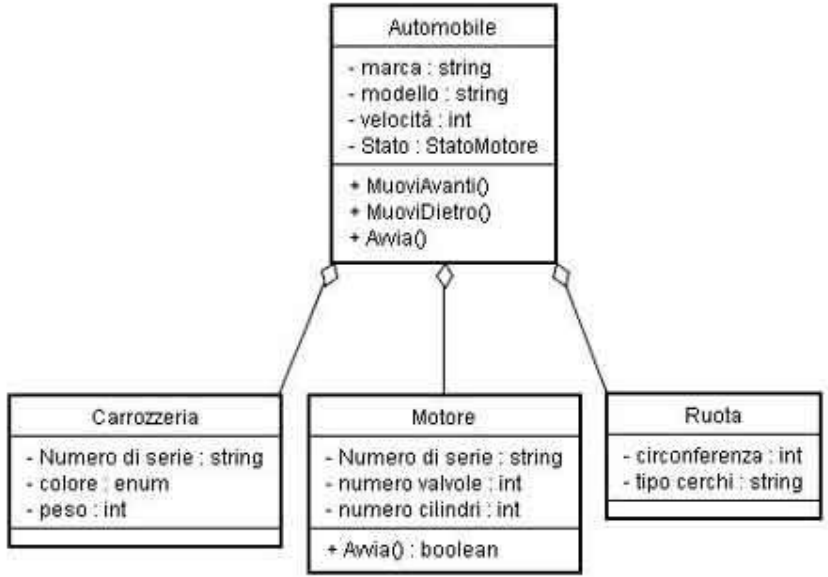


Sylvio Barbon Junior - sylvio.barbonjunior@units.it

2) **Ereditarietà**



Ereditarietá

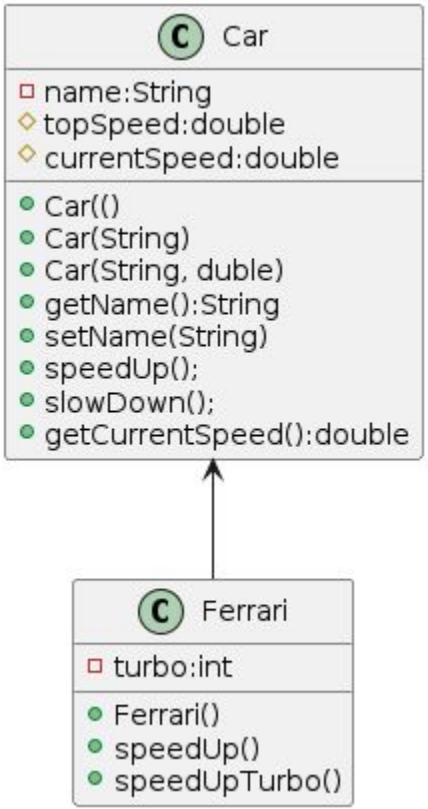


Aggregazione

Sylvio Barbon Junior - sylvio.barbonjunior@units.it

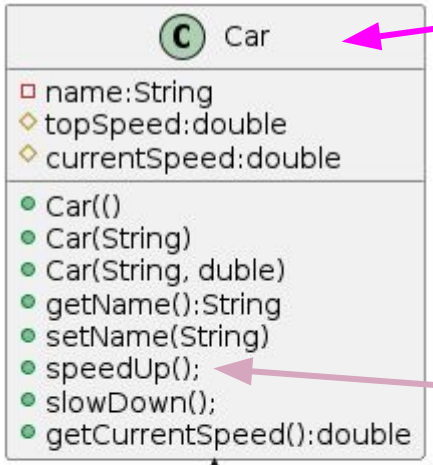
2) **Ereditarietà**

- Permette di estendere classi già definite;
- L'ereditarietà' è un meccanismo fondamentale sia per il riutilizzo del codice che per lo sviluppo incrementale di programmi;
- Questo meccanismo permette di estendere e potenziare classi già esistenti;
- **Ferrari** è una estensione di **Car**, **Ferrari** eredita la struttura di **Car**.



Sylvio Barbon Junior - sylvio.barbonjunior@units.it

2) Ereditarietà




```java
public class Ferrari extends Car{

    private int turbo = 2;

    public Ferrari(){
        super.setName("Ferrari");
        super.topSpeed = 340.00;
    }

    public void speedUp(){
        super.currentSpeed+=7;
    }

    public void speedUpTurbo(){
        super.currentSpeed+= (7*turbo);
    }
}
```

extends - parola riservata

attributo *privato*

metodo costruttore - *public*

metodo overriding: Una sottoclasse può' riscrivere un metodo della superclasse (stesso nome, stessi parametri, stesso tipo)

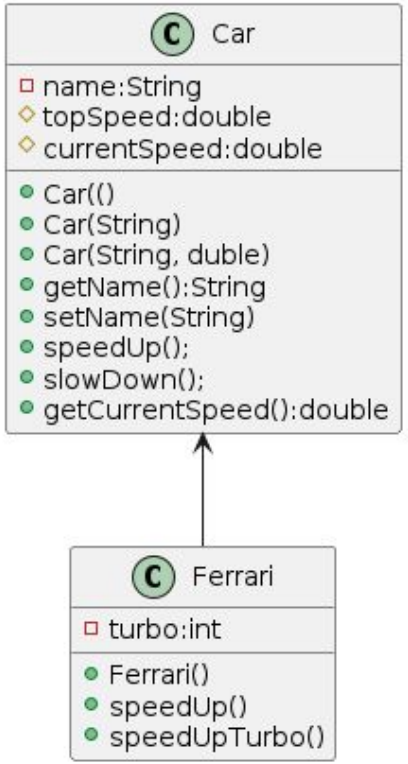
metodi particolari di Ferrari- public

super, accesa la *superclasse*

Sylvio Barbon Junior - sylvio.barbonjunior@units.it

2) Ereditarietà

- La parola chiave extends significa che Ferrari è una **sottoclasse** o classe derivata di Car
- Car è una superclasse o classe **genitrice** di Ferrari
- Analogamente Ferrari è un sottotipo di Car
- Un'istanza della classe Ferrari è utilizzabile in ogni parte del codice in cui sia possibile utilizzare un'istanza della classe Car.



Sylvio Barbon Junior - sylvio.barbonjunior@units.it

2) Ereditarietà

protected -> figlio (sottoclasse) ha accesso.

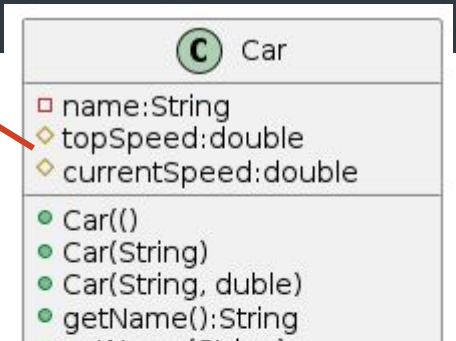
```java
public class Car{
    private String name;
    protected double topSpeed;
    protected double currentSpeed;

    public Car(){}

    public Car(String name){
        this.name = name;
        this.currentSpeed = 0;
        this.topSpeed = 200.00;
    }

    public Car(String name, double topSpeed){
        this.name = name;
        this.topSpeed = topSpeed;
    }

    public String getName(){
        return(this.name);
    }

    public void setName(String name){
        this.name = name;
    }

    public void speedUp(){
        this.currentSpeed++;
    }
```

stessa segnatura

```java
public class Ferrari extends Car{

    private int turbo = 2;

    public Ferrari(){
        super.setName("Ferrari");
        super.topSpeed = 340.00;
    }

    public void speedUp(){
        super.currentSpeed+=7;
    }

    public void speedUpTurbo(){
        super.currentSpeed+= (7*turbo);
    }
}
```

stessa segnatura

>>"sovrascrivere (override)"



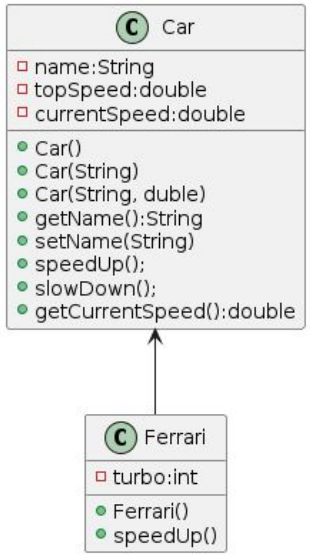
2) Ereditarietà

private -> *Migliore opzione, via costruttore!!!*

super() >> *costruttore della superclasse*

stessa segnatura

```java
public class Car{
    private String name;
    private double topSpeed;
    private double currentSpeed = 0;

    public Car(){}

    public Car(String name){
        this.name = name;
        this.topSpeed = 200.00;
    }

    public Car(String name, double topSpeed){
        this.name = name;
        this.topSpeed = topSpeed;
    }

    public String getName(){
        return(this.name);
    }

    public void setName(String name){
        this.name = name;
    }

    public void speedUp(){
        this.currentSpeed++;
    }

    public void slowDown(){
        this.currentSpeed--;
    }

    public double getCurrentSpeed(){
        return this.currentSpeed;
    }
}
```




```java
public class Ferrari extends Car{

    private int turbo = 7;

    public Ferrari(){
        super("Ferrari", 340.00);
    }

    public void speedUp(){
        for(int i=0; i<turbo; i++)
            super.speedUp();
    }
}
```

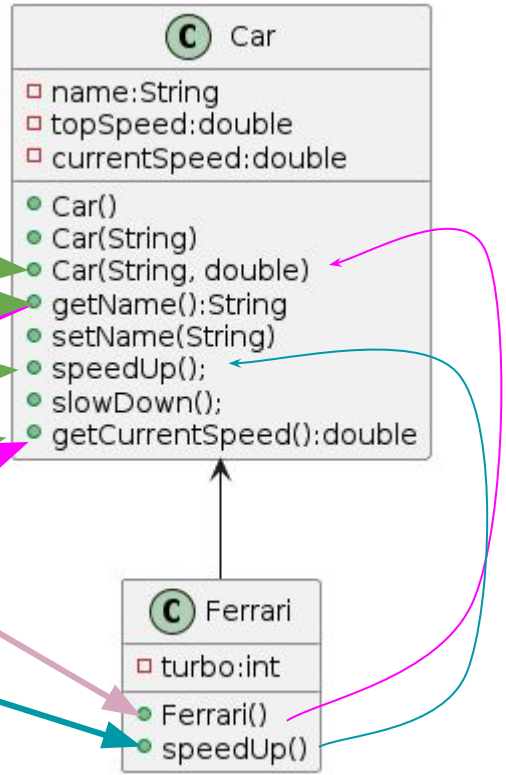
accessando superclasse via metodi

Sylvio Barbon Junior - sylvio.barbonjunior@units.it

2) Ereditarietà - “Ereditarietà di Metodi”

```
1  public class ExecuteFerrari{
2      public static void main(String[] args) {
3          new ExecuteFerrari();
4      }
5
6      ExecuteFerrari(){
7          Car car = new Car("Punto", 180);
8          Ferrari ferrari = new Ferrari();
9
10         System.out.println("Car:"+car.getName());
11         System.out.println("Ferrari:"+ferrari.getName());
12
13         for(int i = 0; i<10; i++){
14             car.speedUp();
15             ferrari.speedUp();
16         }
17
18         System.out.println("Car:"+car.getCurrentSpeed());
19         System.out.println("Ferrari:"+ferrari.getCurrentSpeed());
20     }
21
22 }
```

```
barbon@barbon-XPS13-9333: ~/Downloads/FI/uml
(base) barbon@barbon-XPS13-9333:~/Downloads/FI/uml$ java ExecuteFerrari
Car:Punto
Ferrari:Ferrari
Car:10.0
Ferrari:70.0
(base) barbon@barbon-XPS13-9333:~/Downloads/FI/uml$
```

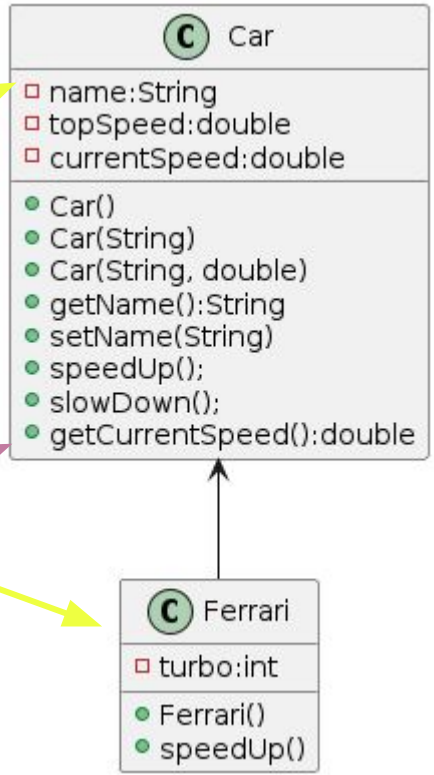


Sylvio Barbon Junior - sylvio.barbonjunior@units.it

2) Ereditarietà - Primo Esperimento “Polimorfismo”

```java
public class ExecuteFerrari2{
    public static void main(String[] args) {
        new ExecuteFerrari2(args[0]);
    }

    ExecuteFerrari2(String tipo){
        Car car;
        if(tipo.equals("Ferrari")){
            car = new Ferrari();
        }else{
            car = new Car("Tipo");
        }

        for (int i=0; i<20; i++) {
            car.speedUp();
        }

        System.out.println("La velocità finale è "+car.getCurrentSpeed());

    }
}
```

```
barbon@barbon-XPS13-9333: ~/Downloads/FI/uml
(base) barbon@barbon-XPS13-9333:~/Downloads/FI/uml$ java ExecuteFerrari2 Ferrari
La velocità finale è 140.0
(base) barbon@barbon-XPS13-9333:~/Downloads/FI/uml$ java ExecuteFerrari2 Test
La velocità finale è 20.0
(base) barbon@barbon-XPS13-9333:~/Downloads/FI/uml$
```



3) Polimorfismo

- Il polimorfismo rappresenta il principio in funzione del quale diverse classi derivate possono implementare uno stesso comportamento definito nella superclasse;
- La differenza fondamentale tra ereditarietà e polimorfismo è che l'ereditarietà consente di riutilizzare il codice già esistente in un programma, e il polimorfismo fornisce un meccanismo per decidere dinamicamente quale forma di una funzione deve essere invocata.

Sylvio Barbon Junior - sylvio.barbonjunior@units.it

4) Esempio

Sylvio Barbon Junior - sylvio.barbonjunior@units.it