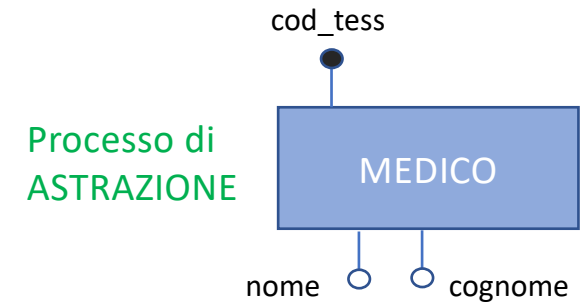
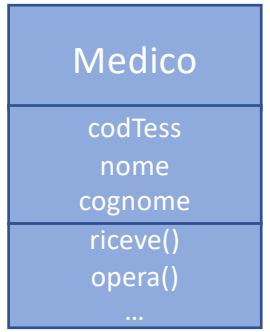
BASI DI DATI Schema/Diagramma ENTITA'-RELAZIONE

OBJECT ORIENTATION / OBJECT ORIENTED PROGR.



Processo di ASTRAZIONE

Entità — *Classe*



Class diagram UML

Attributi (dati)

Metodi (operazioni sui dati)

STRUTTURA DEI DATI

```
int x;
Medico m;
```

Processo di REIFICAZIONE/ ISTANZIAZIONE (inverso ASTRAZIONE)

```
x=5;
m=new Medico();
```

Attributi

codTess	nome	cognome	…		
12345	Alessio	Neri	…		
…	…	…	…		

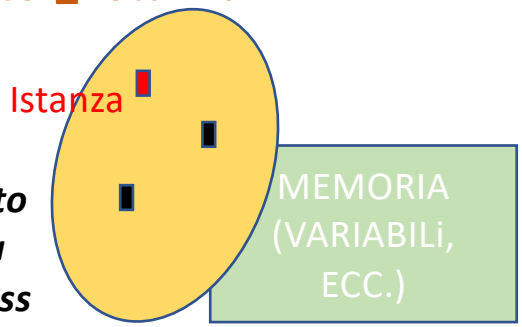
Istanza

Istanza

Una riga è una istanza, un dato "concreto"

DATI ("VERI E PROPRI")

Oggetto ≡ Istanza



Un oggetto è l'istanza di una class

MODELLO RELAZIONALE (TABELLE)

I DATI SARANNO POI REGISTRATI NELLA MEMORIA DEL SISTEMA/CALCOLATORE

PARADIGMA DI PROGRAMMAZIONE PROCEDURALE (PROCEDURE)

E CONCETTI DI **BLOCCO**, **TIPIZZAZIONE** E **PROGRAMMAZIONE STRUTTURATA**

BLOCCO

```
int y; // VARIABILE "GLOBALE"

{
    int x;  // SCOPE o AMBITO DI VISIBILITA'
    x = 5; // VAR. VISIBILE SOLO NEL BLOCCO
           //  IN CUI E' STATA DEFINITA

}
```

```
y=5; OK
x=5; ERRORE, VAR. NON VISIBILE FUORI DAL BLOCCO
```

```
int z;
z = 4.5; // ERRORE (DI TIPO), PERDITA PARTE FRAZIONARIA
```

PROGRAM. STRUTTURATA

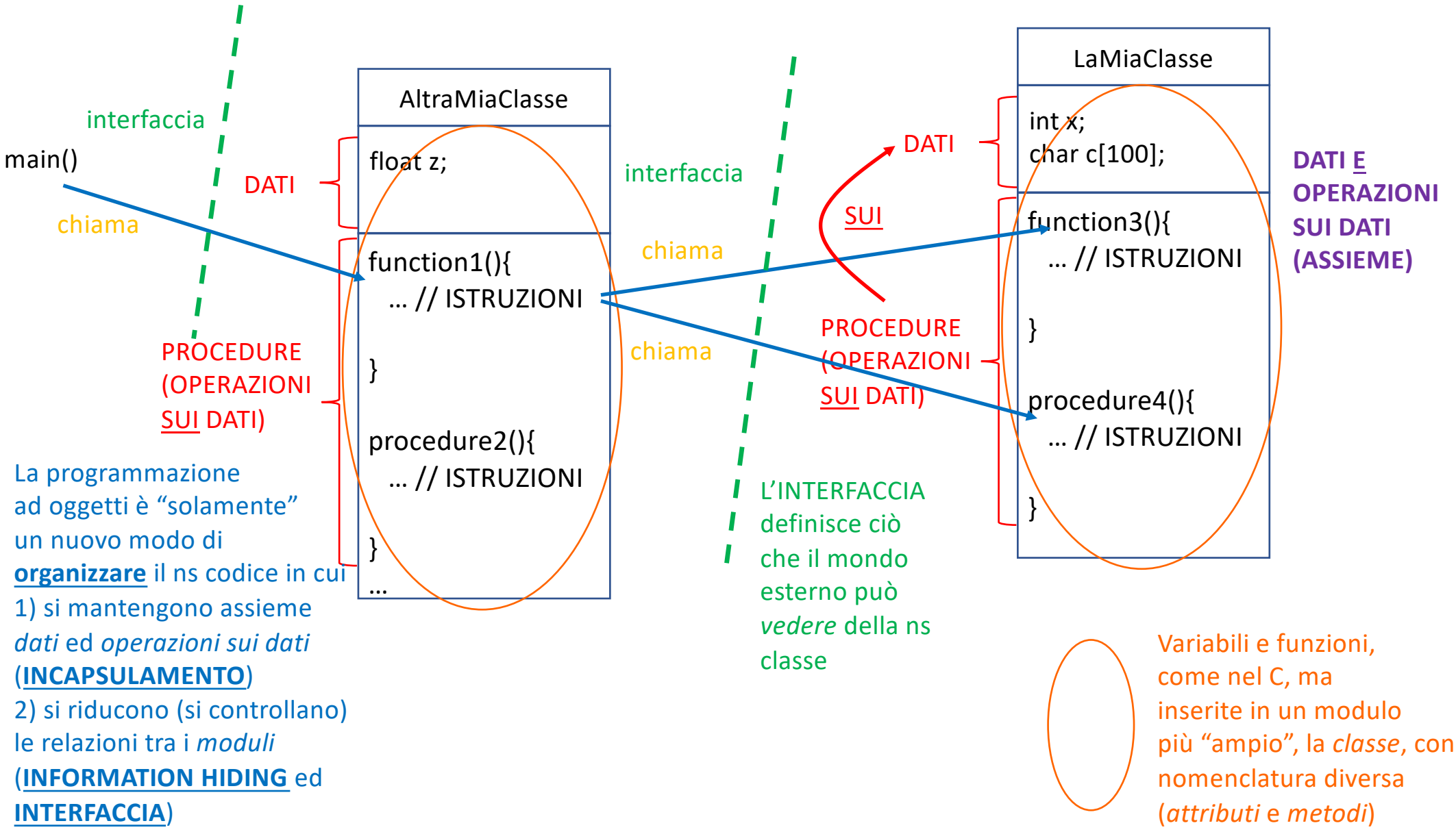
Istruzioni in sequenza
Costrutto di selezione o scelta if() else
Costrutto/i di iterazione for(), while(), …
Singolo punto di ingresso-uscita dal codice (no salti incond.)

PROCEDURE

```
function1(){
  … // ISTRUZIONI

}

procedure2(){
  … // ISTRUZIONI

}
```

ITERAZ.

```
for( …)
{
  … // FATTORIZZARE ISTR.
    // DA RIPETERE
}
```



La programmazione
ad oggetti è "solamente"
un nuovo modo di
organizzare il ns codice in cui
1) si mantengono assieme
dati ed *operazioni sui dati*
(**INCAPSULAMENTO**)
2) si riducono (si controllano)
le relazioni tra i *moduli*
(**INFORMATION HIDING** ed
INTERFACCIA)

TIPI PRIMITIVI

DICHIARAZ. DI UNA VAR.

int x;

TIPO DI DATO

in memoria …

§é

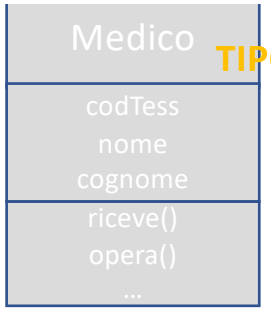
x

5

x=5;

Operazione
di assegnamento

CLASSI/OGG.



TIPO DI DATO (STRUTTURA DEI DATI)

Medico m;

DICHIARAZ. DI UNA VAR.

m è il NOME dell'OGGETTO
(ma, l'oggetto NON ESISTE ANCORA!)

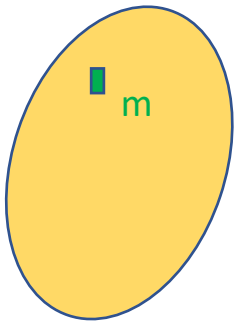
REIFICAZIONE

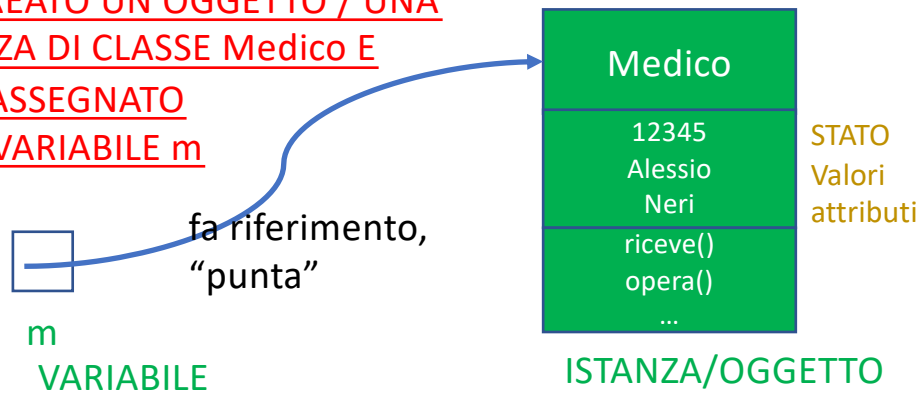
PERCHE' ESISTA, DEVO CREARLO
(ISTANZIAZIONE)

METODO "COSTRUTTORE"
Costruisce/**Crea** ed **Inizializza** l'oggetto

m = new Medico();

HO CREATO UN OGGETTO / UNA ISTANZA DI CLASSE Medico E L'HO ASSEGNATO ALLA VARIABILE m



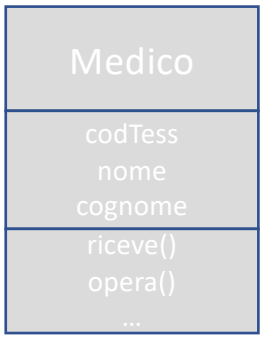


In memoria avremo gli oggetti creati, anche più di uno, ognuno con un **proprio stato** ed una **propria identità**, MA tutti con **medesima struttura** e **funzionalità** (operazioni)

DEFINIZIONE/DICHIARAZIONE
DELLA CLASSE Medico

POI CREAZIONE DEGLI
OGGETTI A PARTIRE
DALLA CLASSE

```
class Medico {
  int codTess;
  String nome;
  String cognome;

  void riceve(){ … };
  int opera(){ … };
}
```




```
Medico m;
m = new Medico();
```

INIZIALIZZAZIONE
GARANTITA
DAL COSTRUTTORE
(MA NON E' DETTO CHE
IO SAPPIA QUALE SIA)

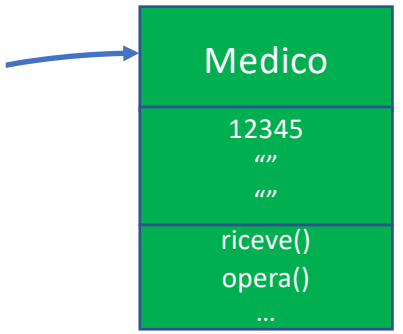
Istanza di (classe) Medico,
creata a partire della
struttura definita dalla classe

Tutte le istanze avranno la
stessa **struttura**

Si dice anche
oggetto di classe/di tipo Medico

```
m.codTess = 12345;
m.nome="Alessio";
m.cognome="Neri";
m.opera();
```

UTILIZZO DELLA
NOTAZIONE PUNTATA,
CON CUI POSSO
ACCEDERE AD
ATTRIBUTI E METODI
DELL'OGGETTO, ES. PER
INIZIALIZZARE ATTRIBUTI,
INVOCARE METODI, ECC.



Stato dell'oggetto
dopo l'inizializzazione o
dopo che sono stati
assegnati valori ai suoi
attributi

Automobile a1;

a1

Inizialmente non c’è dentro **null**a di utile (es., potrebbe essere inizializzata a **null**)

Si può quindi andare a creare un oggetto

Variabile RIFERIMENTO, con cui si fa riferimento all'area di memoria dell'oggetto (qualcosa di simile ad un puntatore del linguaggio C)

Automobile a1;

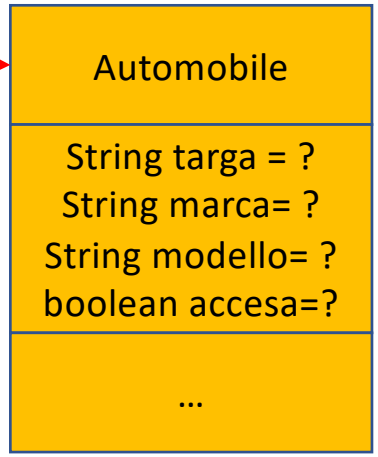
basic.Automobile@2f0e140b

UID a1

Unique IDentifier dell'oggetto creato

La "scatolina" appena create non è abbastanza grande per ospitare l'oggetto automobile con I suoi dati, ecc.

a1 = new Automobile();



System.out.println(a1);

// viene stampato basic.Automobile@2f0e140b

Variabile RIFERIMENTO, con cui si fa riferimento all'area di memoria dell'oggetto
(qualcosa di simile ad un puntatore del linguaggio C)

Automobile a1;

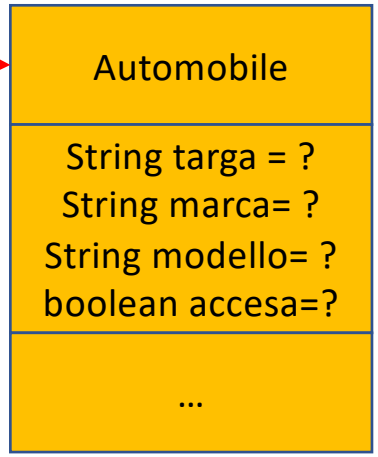
basic.Automobile@2f0e140b

UID a1

Unique IDentifier dell'oggetto creato

La "scatolina" appena create non è abbastanza grande
per ospitare l'oggetto automobile con I suoi dati, ecc.

a1 = new Automobile();



System.out.println(a1);
// viene stampato basic.Automobile@2f0e140b

// per stampare targa, marca, ecc.
// occorre accedere ai singoli attributi
System.out.println(a1.targa+" "+a1.marca+" "+…);

ACCESSO IN LETTURA
AGLI ATTRIBUTI DELL'OGGETTO a1

Variabile RIFERIMENTO, con cui si fa riferimento all'area di memoria dell'oggetto
(qualcosa di simile ad un puntatore del linguaggio C)

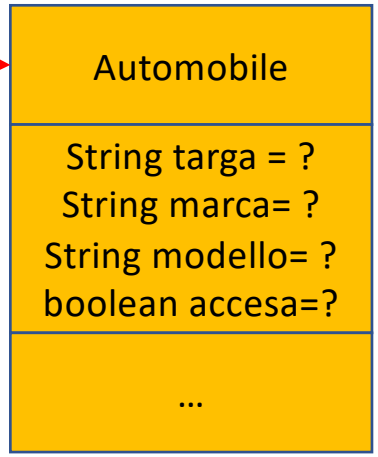
Automobile a1;

basic.Automobile@2f0e140b

UID a1

Unique IDentifier dell'oggetto creato

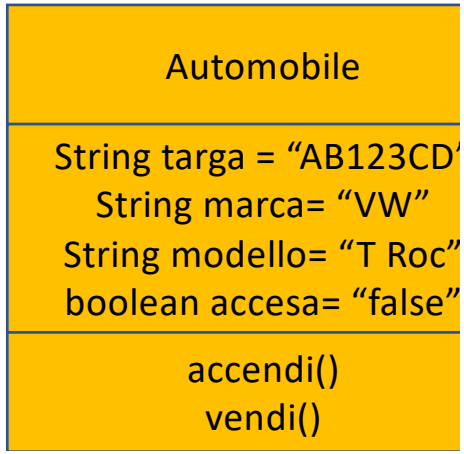
a1 = new Automobile();



System.out.println(a1);
// viene stampato basic.Automobile@2f0e140b

a1.targa="AB123CD";
a1.marca="VW";
…

ACCESSO IN SCRITTURA
AGLI ATTRIBUTI DELL'OGGETTO a1



Variabile RIFERIMENTO, con cui si fa riferimento all'area di memoria dell'oggetto
(qualcosa di simile ad un puntatore del linguaggio C)

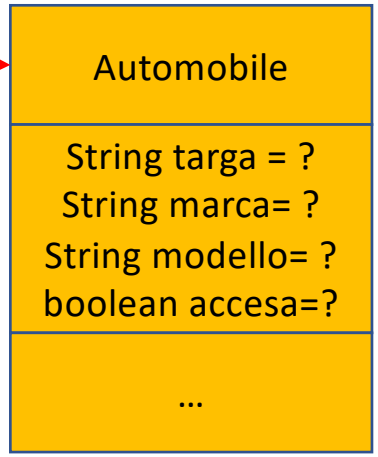
```
Automobile a1;
```

basic.Automobile@2f0e140b

UID a1

Unique IDentifier dell'oggetto creato

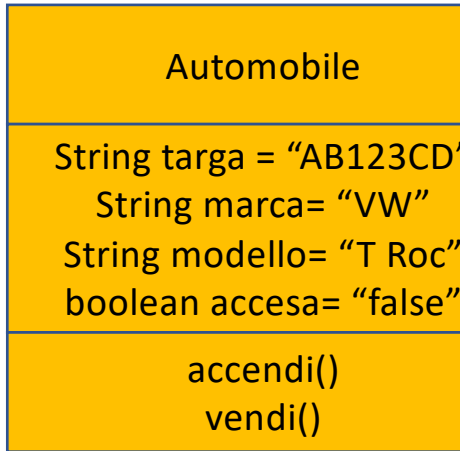
```
a1 = new Automobile();
```

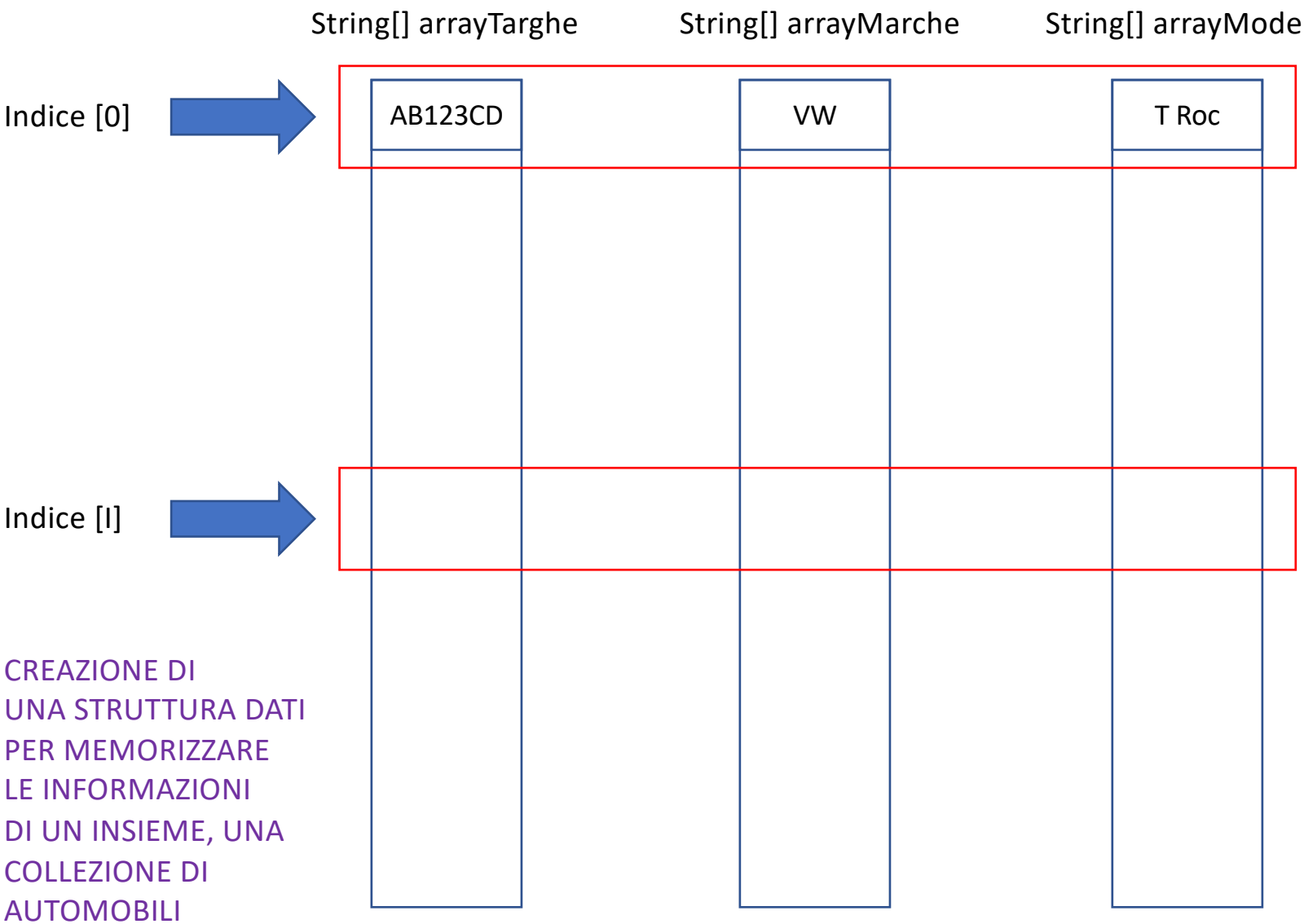



```
System.out.println(a1);
// viene stampato basic.Automobile@2f0e140b

a1.targa="AB123CD";
a1.marca="VW";
…
a1.accendi();
```

ACCESSO IN SCRITTURA
AGLI ATTRIBUTI DELL'OGGETTO a1

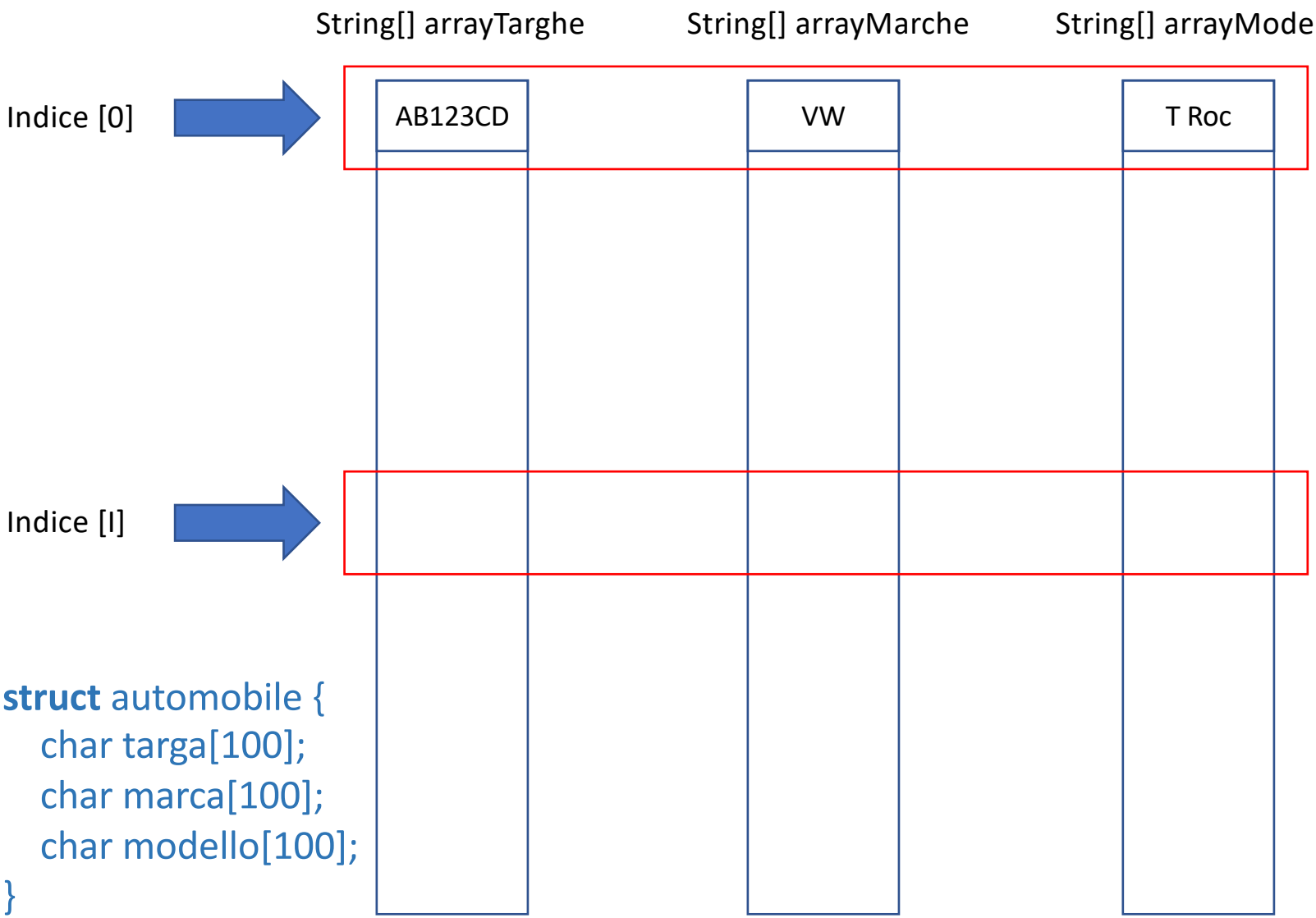
INVOCAZIONE DEI METODI DELL'OGGETTO ai





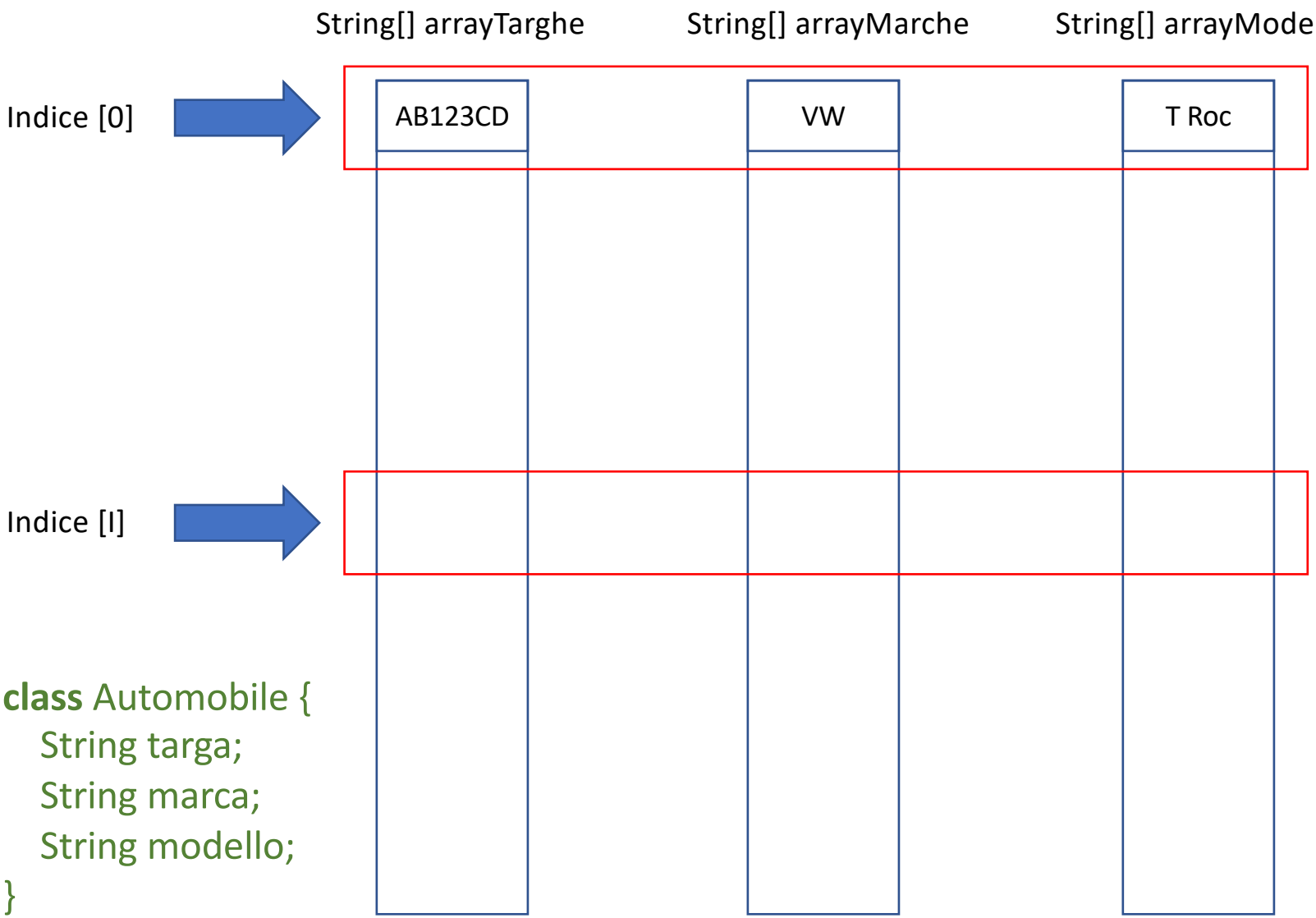
CREAZIONE DI
UNA STRUTTURA DATI
PER MEMORIZZARE
LE INFORMAZIONI
DI UN INSIEME, UNA
COLLEZIONE DI
AUTOMOBILI

Sia usando delle variabili, es. targa1, marca1, ecc., sia usando degli array “paralleli” le informazioni di ciascuna automobile sono distribuite, non si riescono a raggruppare in un unico contenitore




```
struct automobile {
    char targa[100];
    char marca[100];
    char modello[100];
}
```

In linguaggio C avrei potuto pensare di utilizzare una **struct**

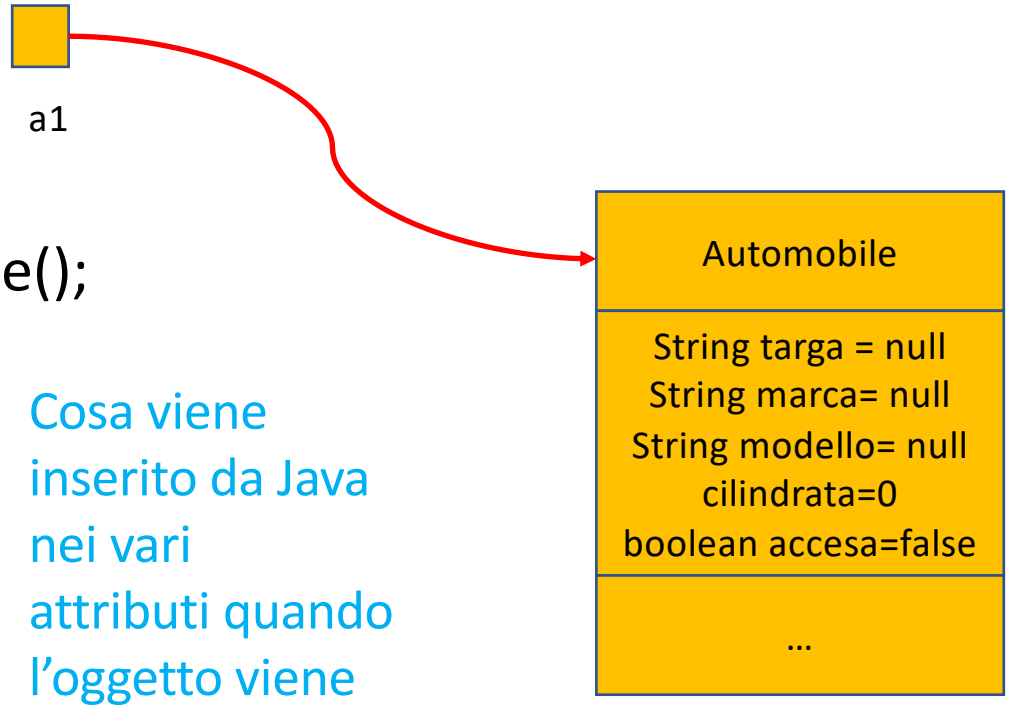



```
class Automobile {
    String targa;
    String marca;
    String modello;
}
```

In linguaggio C avrei potuto pensare di utilizzare una **struct**

In Java, non la struct (non esiste), ma la **classe**

Automobile a3;



a3 = new Automobile();

// Stampando a3
Targa: null
Marca: null
Modello: null
Cilindrata: 0
Accesa: false

Cosa viene inserito da Java nei vari attributi quando l'oggetto viene creato?
Java li **inizializza** con valori **predefiniti**

Numerici a 0 / 0.0
Boolean a false
String a null

Automobile a3;

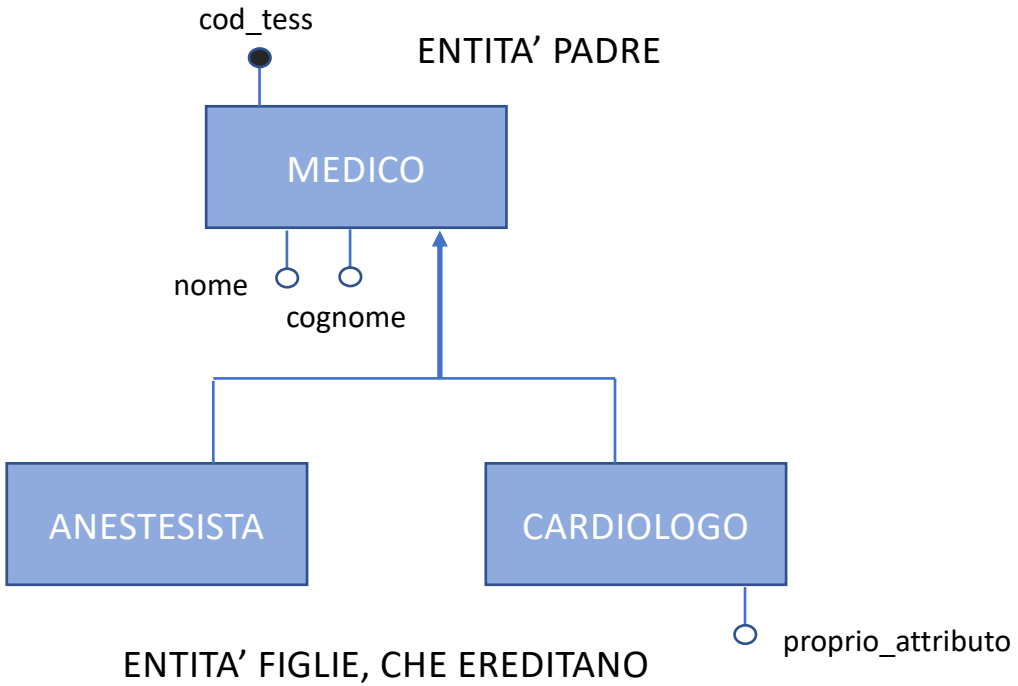
a3 = new Automobile();

CREAZIONE ED **INIZIALIZZAZIONE** DELL'OGGETTO AVVIENE TRAMITE **L'INVOCAZIONE** DEL (METODO) COSTRUTTORE **Automobile()**

IN JAVA OGNI CLASSE DEVE ESSERE DOTATA **DI ALMENO UN** COSTRUTTORE

COSTRUTTORE **DI DEFAULT** (ANCHE DETTO "VUOTO" PERCHE' SENZA PARAMETRI

SE IL PROGRAMMATORE NON LO HA SCRITTO, ESISTE IMPLICITAMENTE, ED **INIZIALIZZA GLI ATTRIBUTI AI VALORI PREDEFINITI**

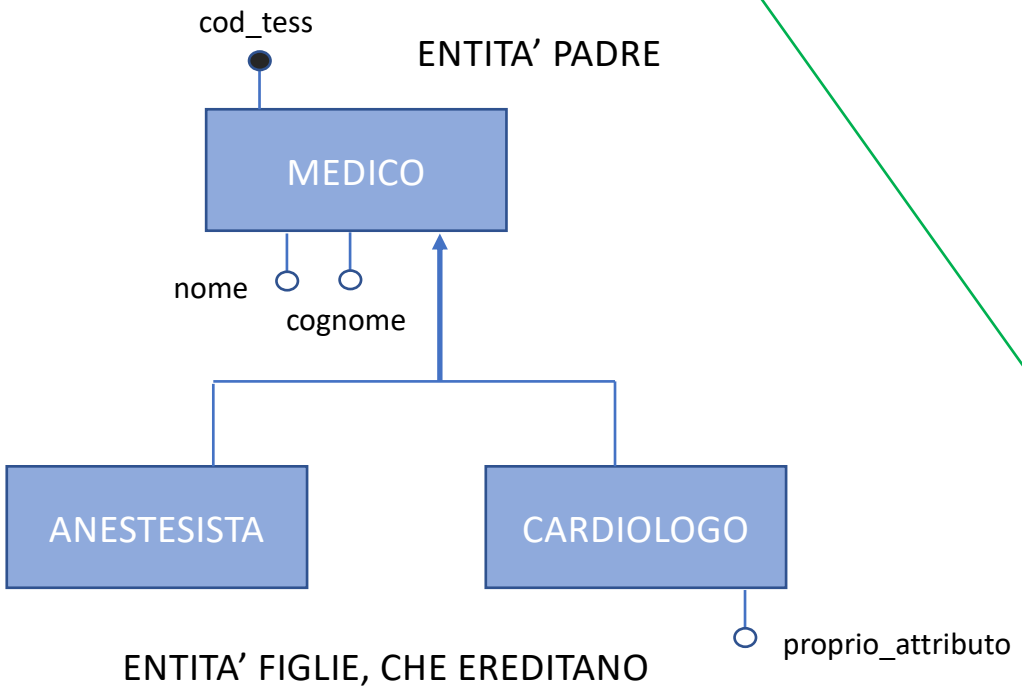


EREDITARIETA' NELLE BASI DI DATI

EREDITARIETA' IN OOP

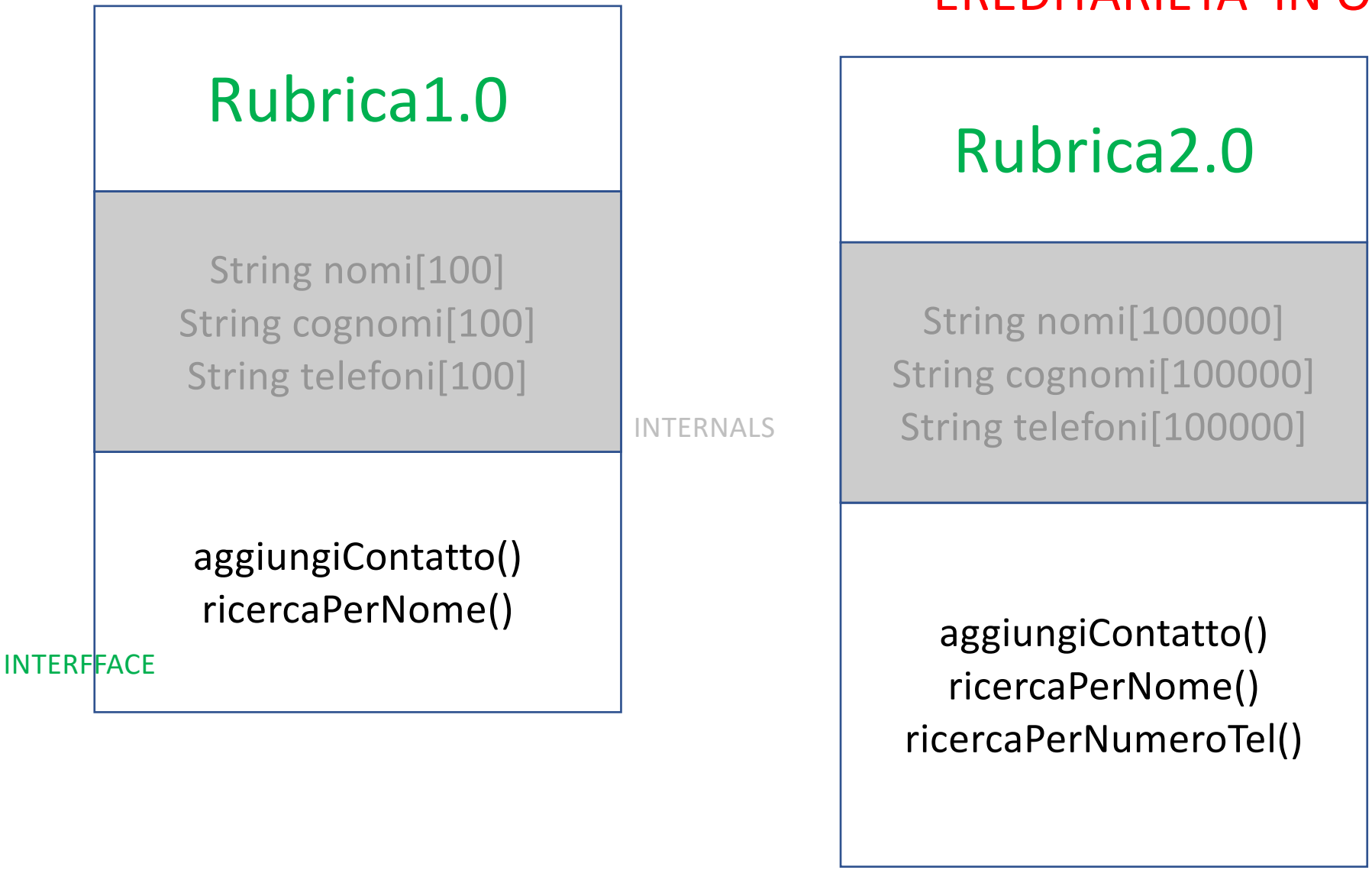
NELL'ORIENTAMENTO AD OGGETTI (INVECE)
Una classe che eredita da un'altra

- Ha gli stessi attributi e metodi della classe da cui eredita
- Può **AGGIUNGERE** altri attribute e metodi
- E **PUO' CAMBIARE** quanto ereditato



EREDITARIETA' NELLE BASI DI DATI

EREDITARIETA' IN OOP



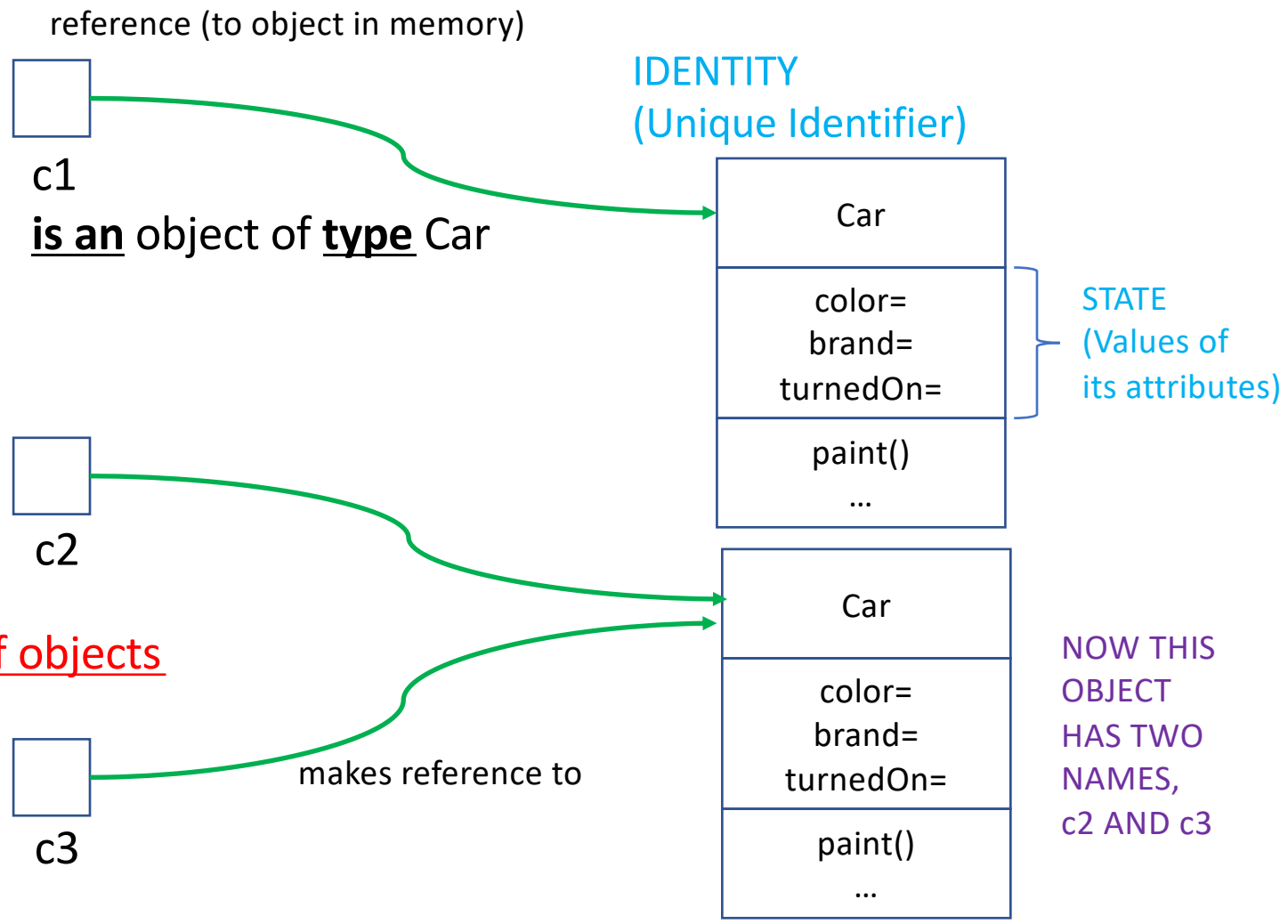
Car c1;

c1 = new Car();

Car c2 = new Car();

c1, c2, etc. are the names of objects

c3 = c2;



Car c1;

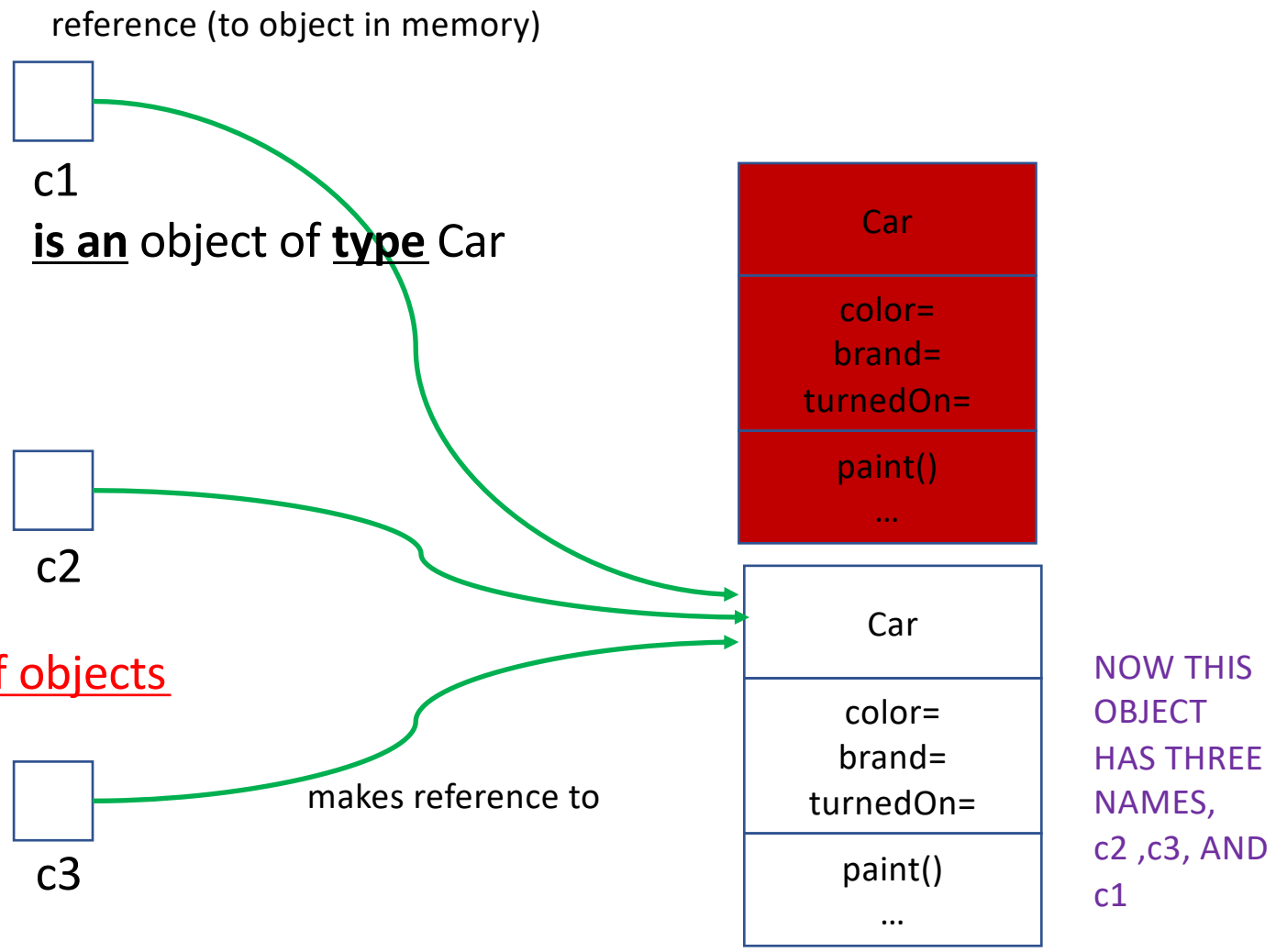
c1 = new Car();

Car c2 = new Car();

c1, c2, etc. are the names of objects

Car c3 = c2;

c1 = c2;



WHEN AN OBJECT HAS MORE NAMES: ALIASING

Car c1;

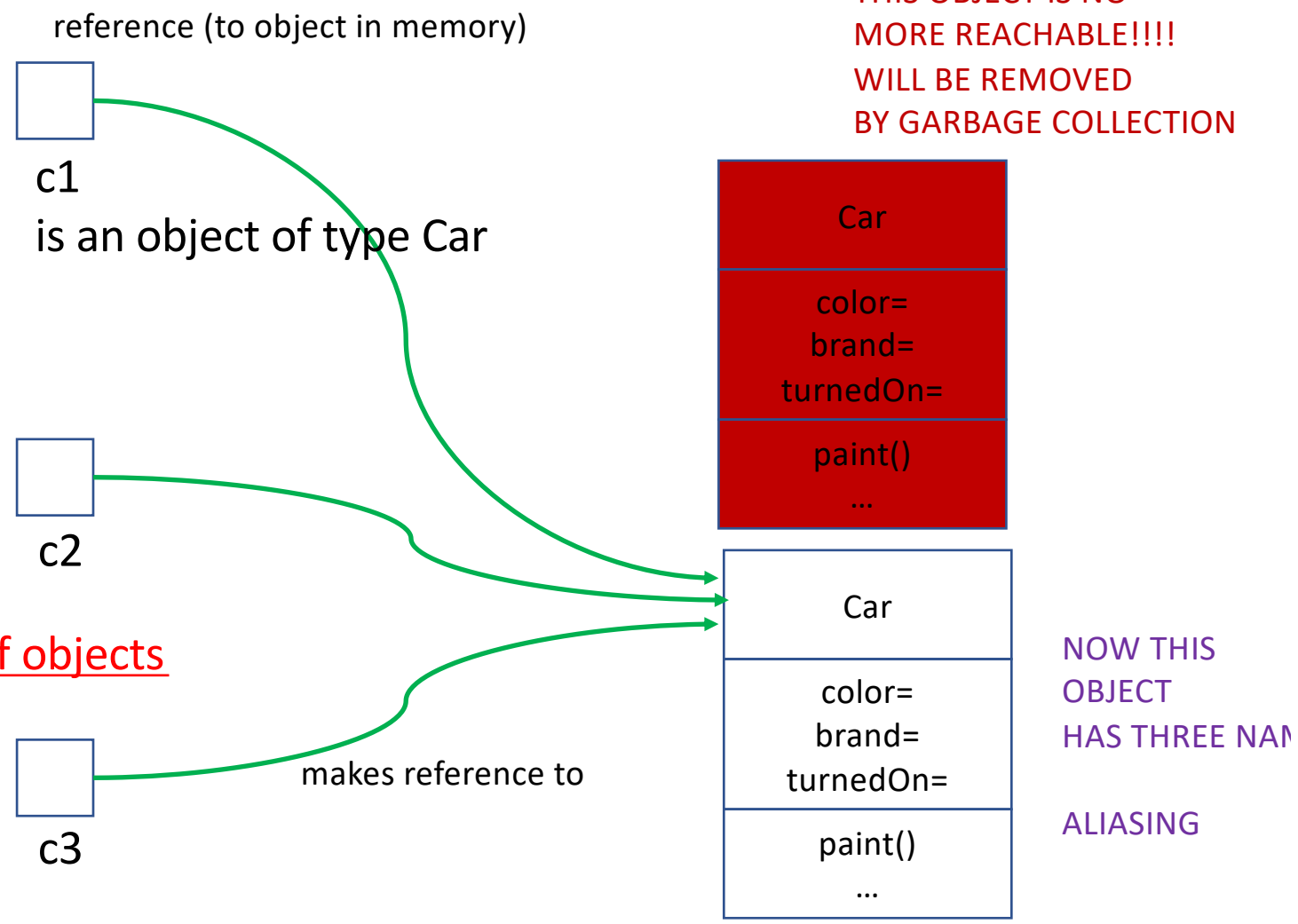
c1 = new Car();

Car c2 = new Car();

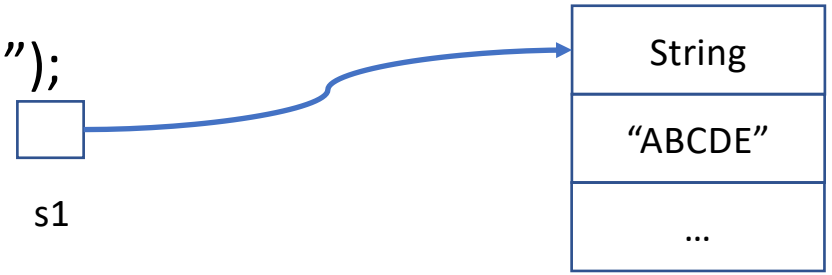
c1, c2, etc. are the names of objects

Car c3 = c2;

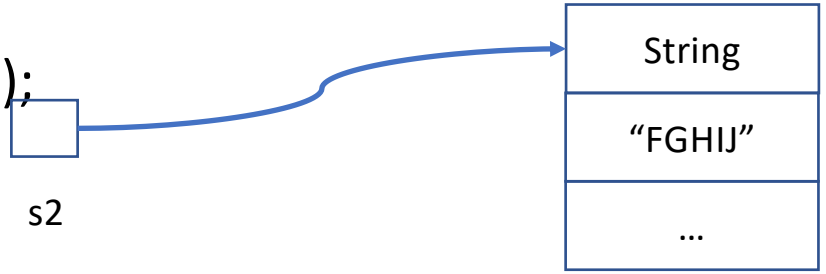
c1 = c2;



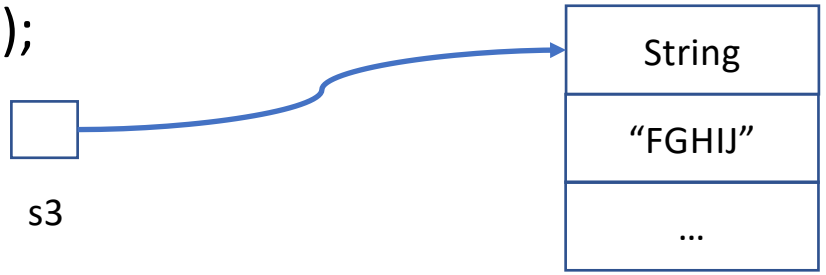
String s1 = new String(”ABCDE”);



String s2 = new String(“FGHIJ”);



String s3 = new String(“FGHIJ”);



ARE THEY **EQUAL**?

THEY HAVE THE SAME CONTENT, BUT ARE TWO DIFFERENT OBJECTS (s2, s3)

if(s2==s3){…} // HERE WE ARE **COMPARING REFERENCES**

String s2 = new String(”FGHIJ”);

String s3 = new String(”FGHIJ”);

if(s2==s2){ // DOES **NOT COMPARE THE STRING CONTENT**, JUST COMPARES THE UIDs

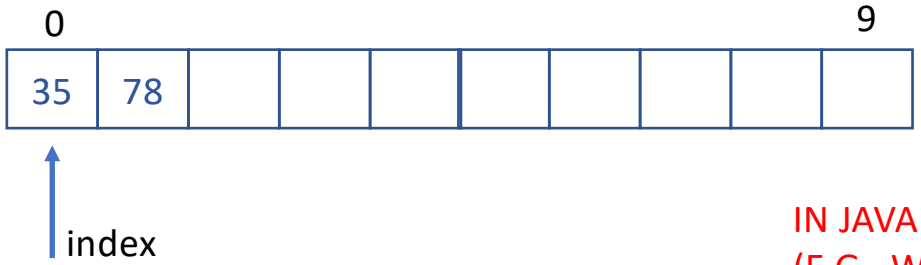
} // IN PRINCIPLE SHOULD RETURN false
  // IN JAVA, WITH STRINGS, THESE COMPARISONS MAY RETURN true
  // IN GENERAL, AVOID COMPARING OBJECTS USING == (REFERENCES)

HOW TO COMPARE THE CONTENT OF TWO STRINGS?

USE String CLASS METHOD NAMED compareTo(String otherString)

```
if(s2.compareTo(s2)==0){
   // THEY ARE EQUAL
}
```

int array[] = new int[10]; // IN C HAD TO BE A CONSTANT, IN JAVA EVEN A VARIABLE



IN JAVA AN ARRAY IS “TREATED LIKE” AN OBJECT
(E.G., WE CAN USE DOT NOTATION ON IT)

FOR INSTANCE **array.length** GIVES ME THE NUMBER OF CELLS (10 HERE)
“DELEGATION”, WHO BETTER THAN THE ARRAY IS ABLE TO …?

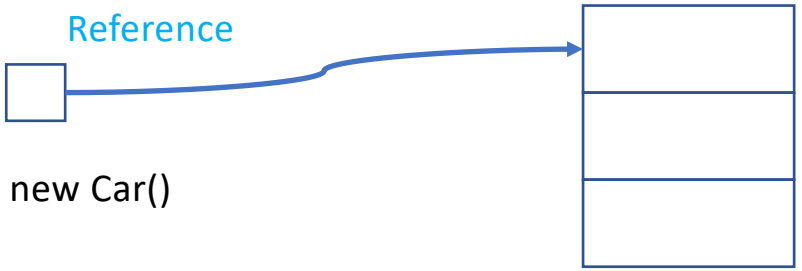
PRIMITIVE TYPES

5

int x=5;

CLASS TYPES

Reference



Car c= new Car()

ARRAY

int arrayInt[] = new int[10];

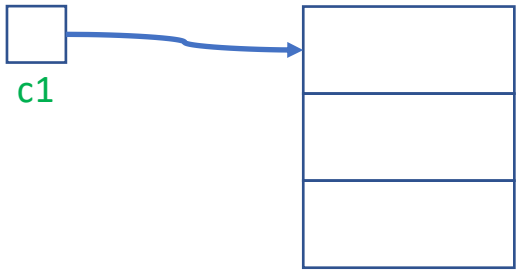
arrayInt[0] = 5

10 ReferenceS

READY TO HOST THE REFERENCES TO OBJECTS (STILL TO BE CREATED)

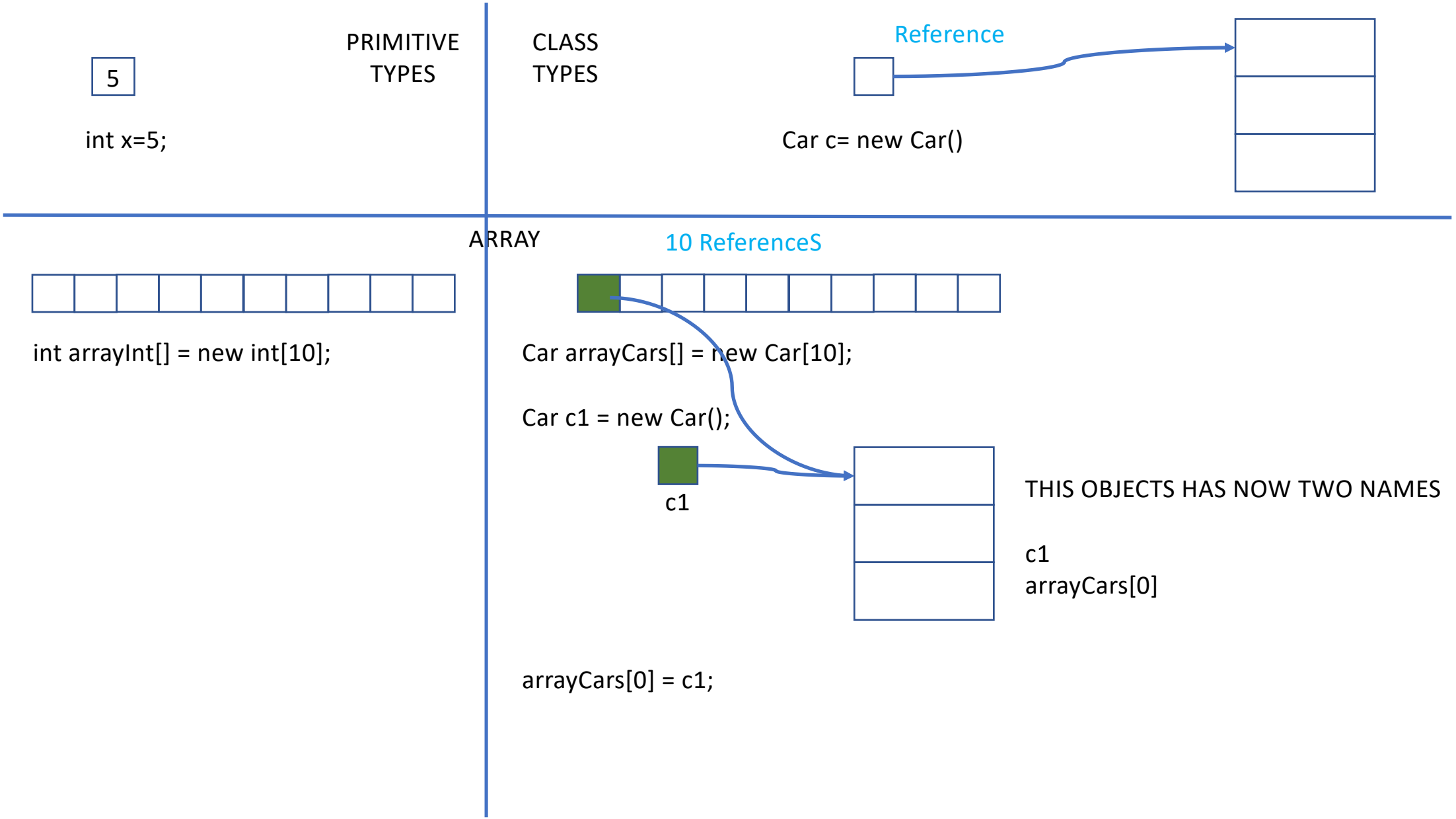
Car arrayCars[] = new Car[10];

Car c1 = new Car();



arrayCars[0] = c1;

GOAL: CREATING AN ARRAY OF OBJECTS KNOWING HOW TO CREATE AN ARRAY OF PRIMITIVE TYPES


PRIMITIVE
TYPES
CLASS
TYPES
5
int x=5;
Reference
Car c= new Car()
ARRAY
10 ReferenceS
int arrayInt[] = new int[10];
Car arrayCars[] = new Car[10];
Car c1 = new Car();
c1
THIS OBJECTS HAS NOW TWO NAMES
c1
arrayCars[0]
arrayCars[0] = c1;

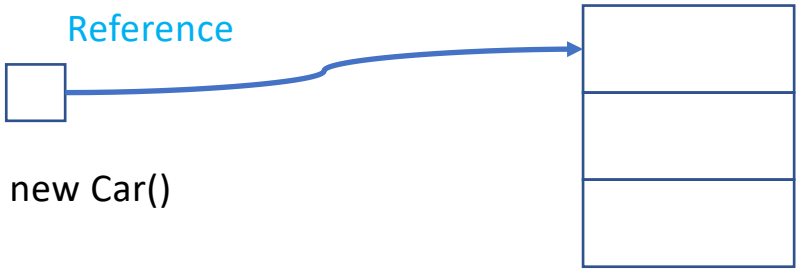
PRIMITIVE TYPES

5

int x=5;

CLASS TYPES

Reference



Car c= new Car()

ARRAY

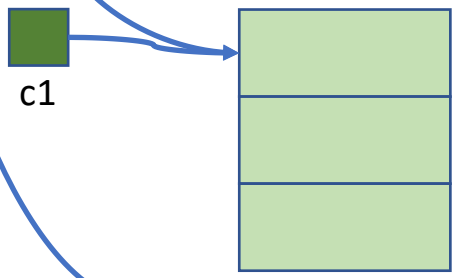
5

int arrayInt[] = new int[10];

10 ReferenceS

Car arrayCars[] = new Car[10];

Car c1 = new Car();



c1

THIS OBJECTS HAS NOW TWO NAMES (ALIASING)

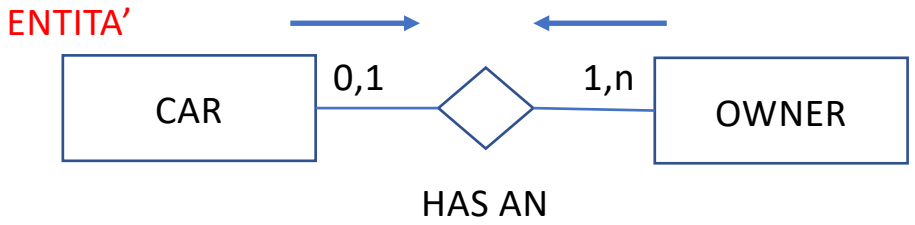
c1
arrayCars[0]

arrayCars[0] = c1;
Car c2 = new Car();
arrayCars[1] = c2;

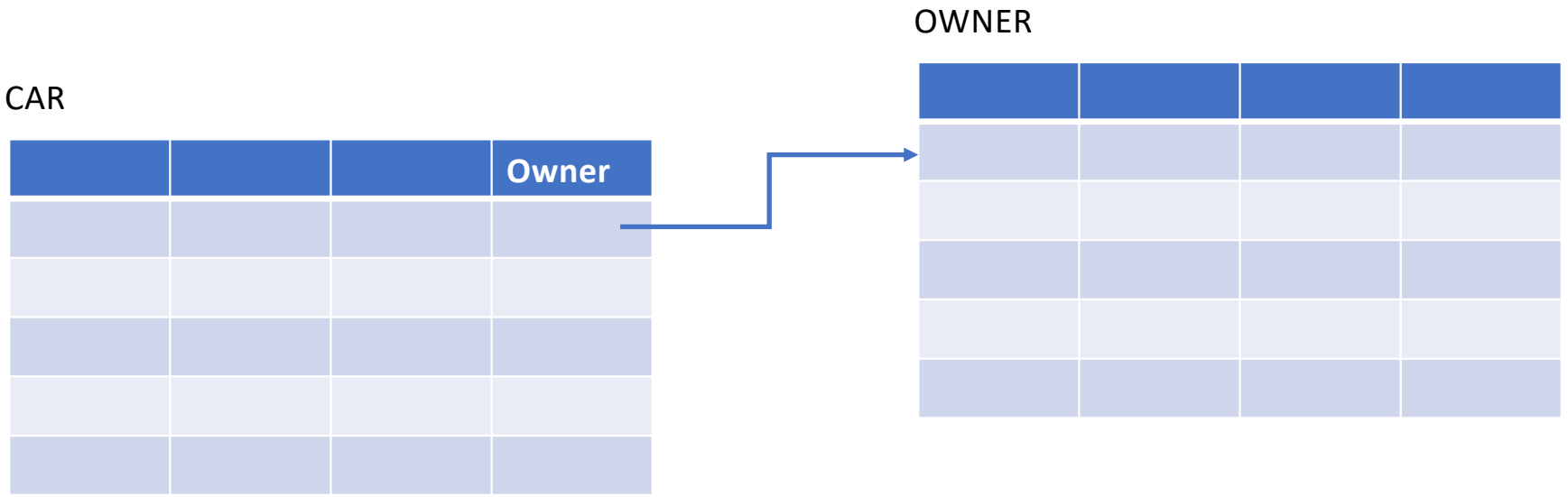
c2

ARRAY CELLS ARE AT **null** UNTIL WE FILL THEM UP WITH REF. TO OBJECTS

NEL MONDO DELLE BASI DI DATI …

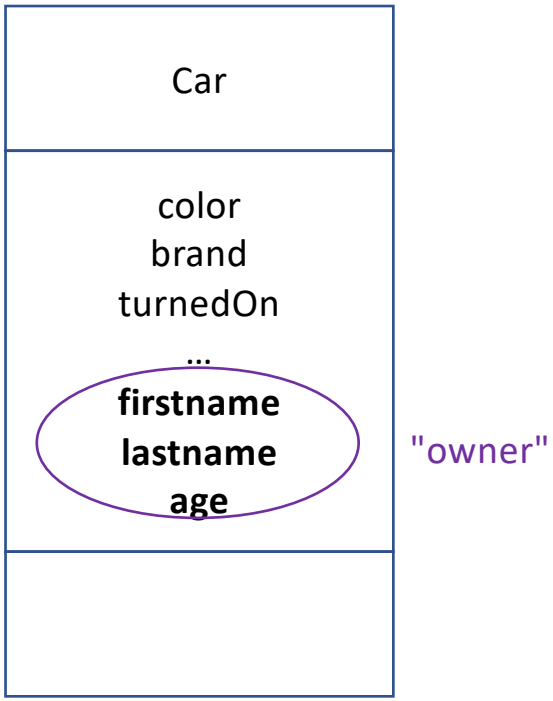


SCHEMA ENTITA’-RELAZIONI



COME MEMORIZZARE LE INFORMAZIONI
RELATIVE ALL'OWNER DELLA CAR?

POTREI AGGIUNGERLE COME
ALTRI ATTRIBUTI …



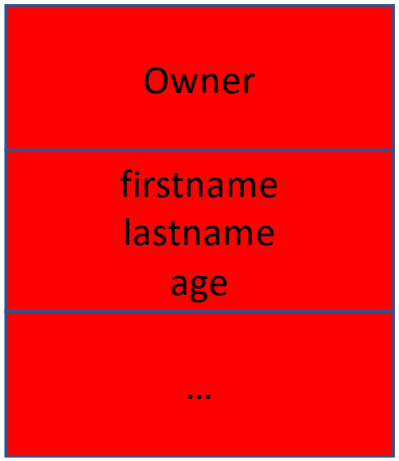
COME MEMORIZZARE LE INFORMAZIONI
RELATIVE ALL'OWNER DELLA CAR?

POTREI AGGIUNGERLE COME
ALTRI ATTRIBUTI …
… MA NON STAREI UTILIZZANDO
I PRINCIPI DELL'OOP

COME MEMORIZZARE LE INFORMAZIONI RELATIVE ALL'OWNER DELLA CAR?

PROCESSO DI ASTRAZIONE:

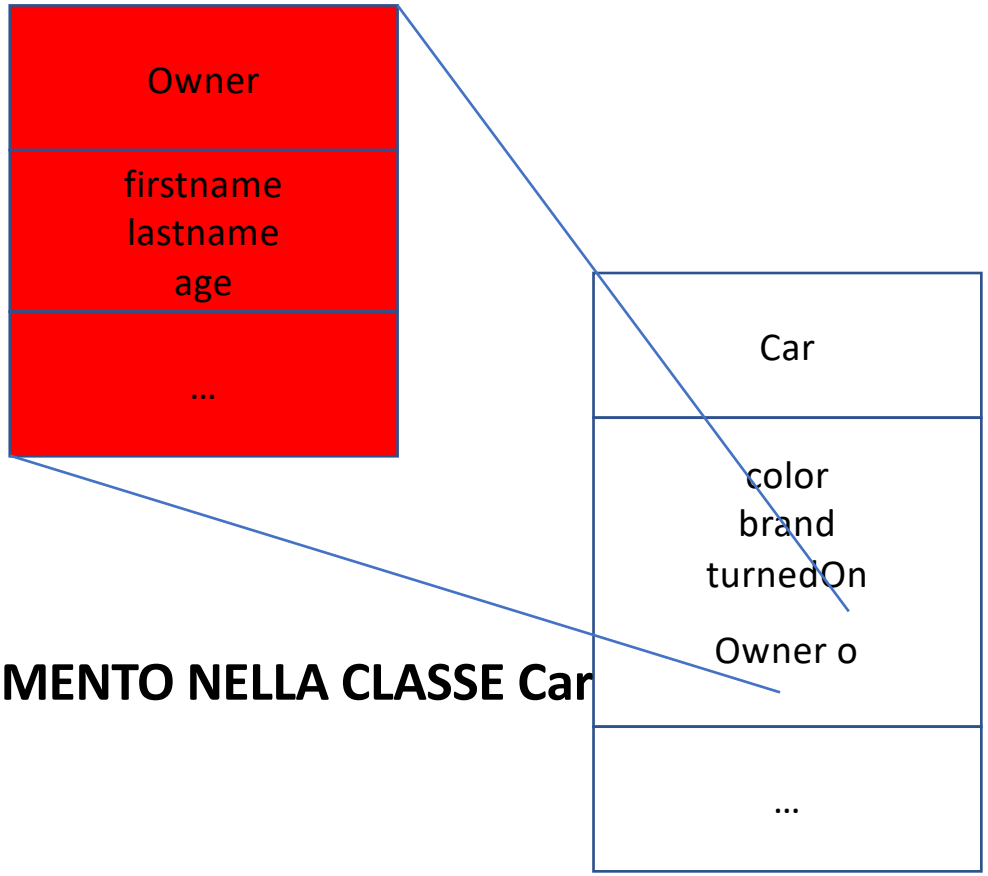
DOVREI DEFINIRE UNA CLASSE Owner



COME MEMORIZZARE LE INFORMAZIONI RELATIVE ALL'OWNER DELLA CAR?

PROCESSO DI ASTRAZIONE:

DOVREI DEFINIRE UNA CLASSE Owner

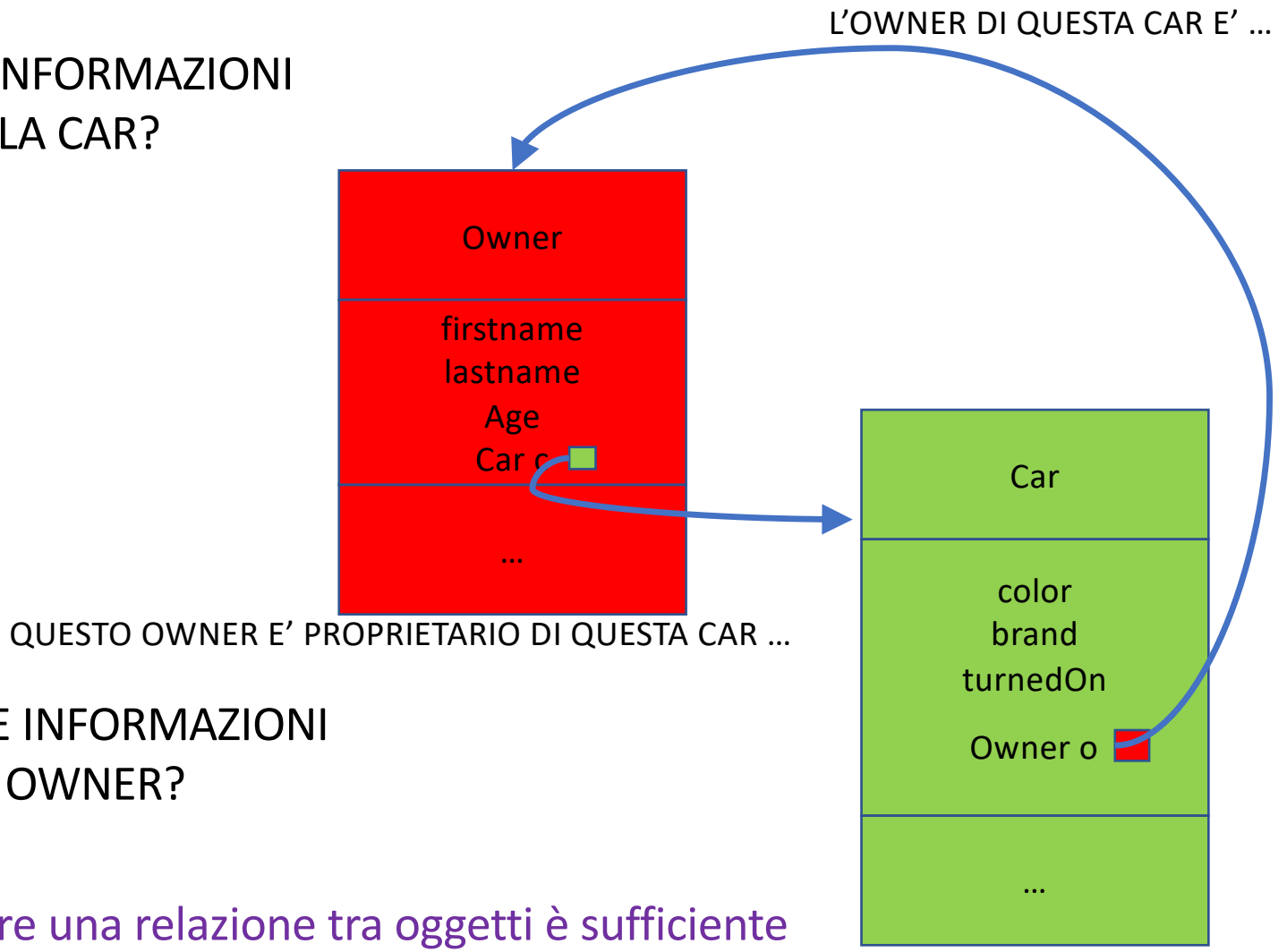


E INSERIRNE UN RIFERIMENTO NELLA CLASSE Car

COME MEMORIZZARE LE INFORMAZIONI RELATIVE ALL'OWNER DELLA CAR?

DICHIARO UNA VARIABILE DI TIPO Owner ALL'INTERNO DELLA CLASSE Car: QUESTA FUNGERA' DA RIFERIMENTO ALL'OGGETTO Owner CHE RAPPRESENTA L'OWNER DI QUELLA CAR

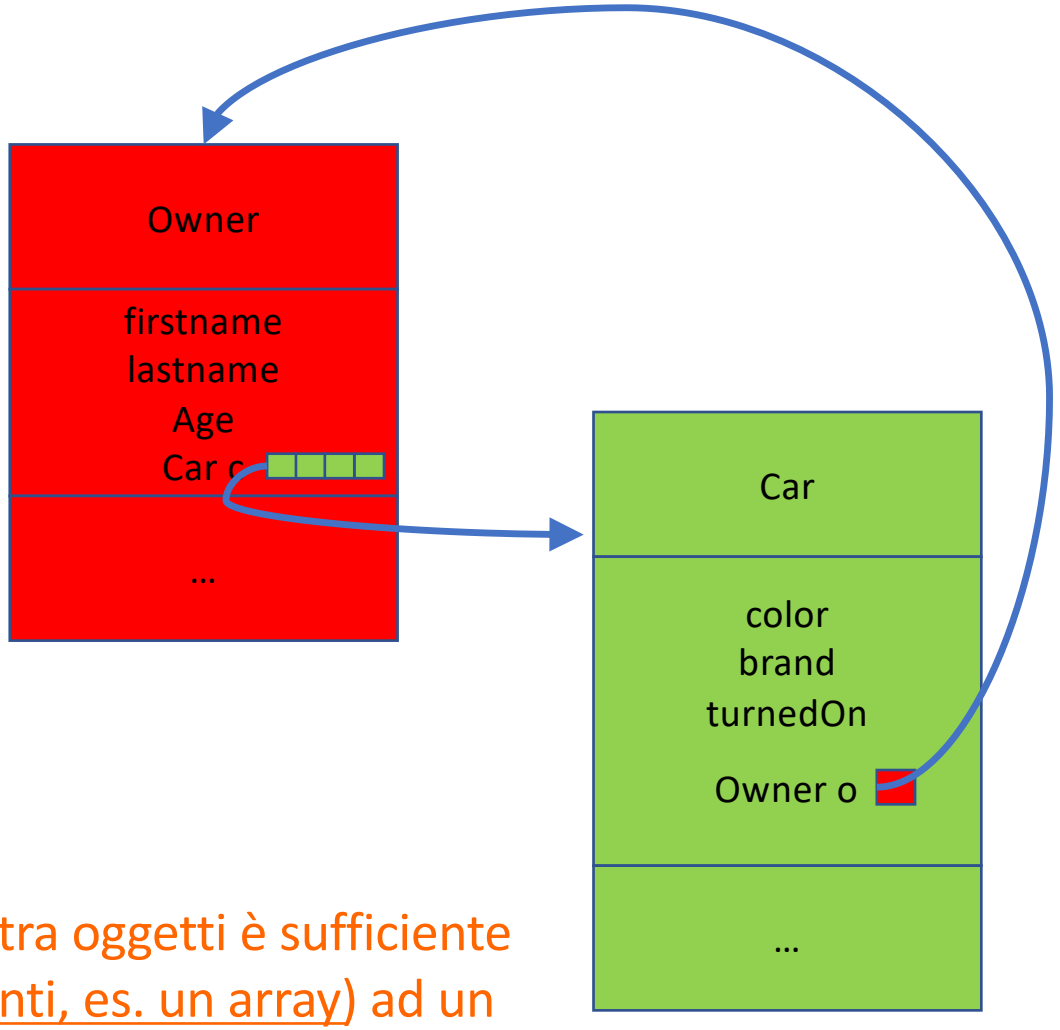
COME MEMORIZZARE LE INFORMAZIONI RELATIVE ALL'OWNER DELLA CAR?



E COME MEMORIZZARE LE INFORMAZIONI RELATIVE ALLA CAR DI UN OWNER?

Nella OOP, per definire una relazione tra oggetti è sufficiente inserire un riferimento ad un oggetto in un altro oggetto

COME MEMORIZZARE LE INFORMAZIONI RELATIVE ALL'OWNER DELLA CAR?



E COME MEMORIZZARE LE INFORMAZIONI RELATIVE ALLA CAR DI UN OWNER?

Nella OOP, per definire una relazione tra oggetti è sufficiente inserire un riferimento (o più riferimenti, es. un array) ad un oggetto (o a più oggetti) in un altro oggetto