CORSO JAVA DEVELOPER

OOP = Object Oriented Programming

Java è un linguaggio di programmazione
orientato agli oggetti

La programmazione orientata agli oggetti si basa sul
concetto di **Classe** e di **Oggetto**

E' un modo di organizzare il codice che imita il mondo reale

OOP - Tutto è un oggetto

Tutti gli oggetti hanno:

- delle caratteristiche
- uno stato
- delle azioni con cui interagiscono tra loro e col mondo esterno

OOP - Tutto è un oggetto

Noi sappiamo dare un nome a un oggetto perché ha delle caratteristiche in comune con altri oggetti della stessa **classe**

un monitor

un telefono

una pianta

una persona

OOP - Tutto è un oggetto

Un monitor ha:

- dimensioni, marca, modello, ...
- stato acceso o spento, ...
- possibilità di accendere e spegnere, regolare luminosità, ...

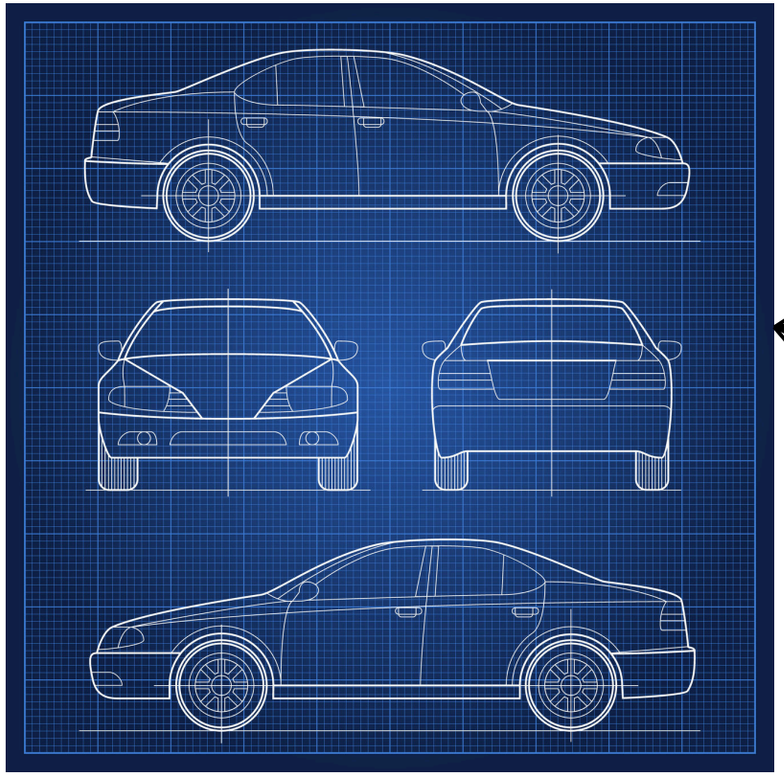
OOP - Tutto è un oggetto

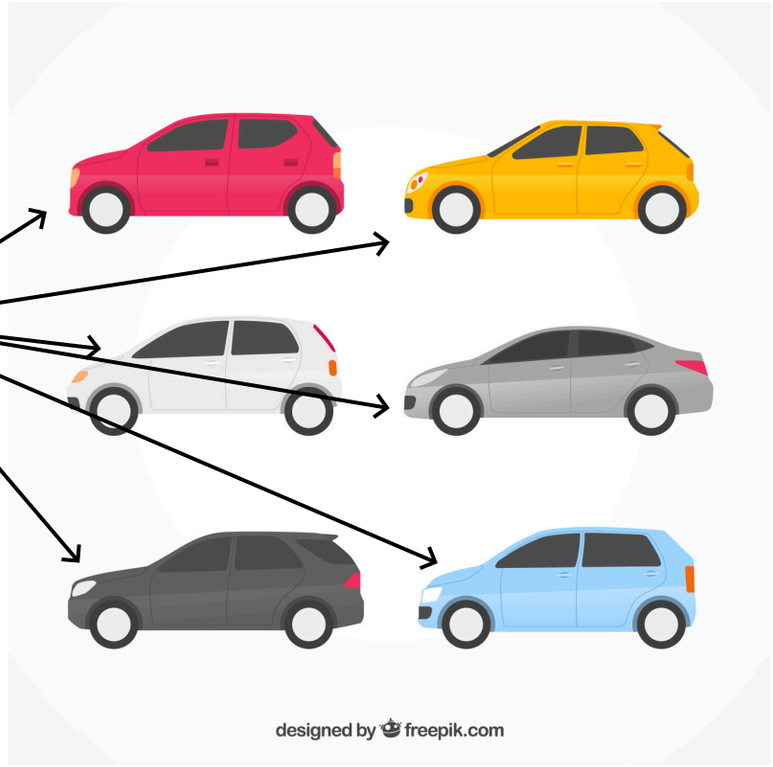
Tutti abbiamo davanti un **oggetto** che ha delle caratteristiche che lo identificano con il nostro modello astratto di monitor, cioè sono istanze di una **classe**

- i nostri monitor hanno tutti degli attributi (ad es. marca, dimensioni) il cui valore varia da oggetto a oggetto
- i nostri monitor possono essere accesi o spenti, ma lo stato di un oggetto non interferisce con quello degli altri della stessa classe
- i monitor hanno dei metodi con cui possiamo eseguire delle azioni e modificarne lo stato (ad es. accendere, spegnere)

Classe = modello

Oggetti = istanze



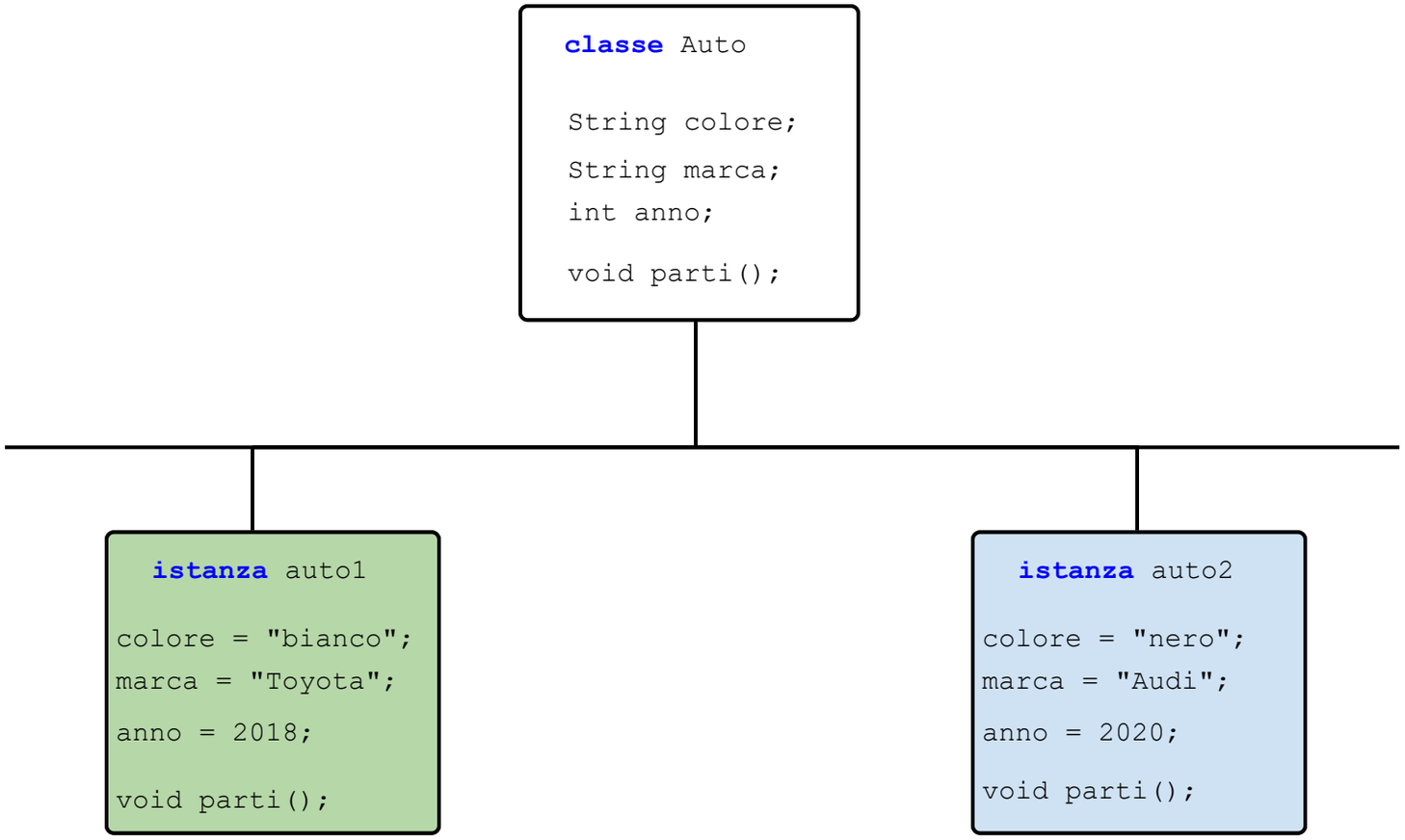


Oggetti = istanze

Una volta creato, un oggetto si comporta come un mondo a parte, indipendente dagli altri oggetti

Un oggetto ha le sue caratteristiche (attributi) e può comunicare con il mondo esterno attraverso i propri metodi

Classi e Istanze



Ogni istanza è un ambiente diverso

Classe

Per creare una classe uso la parola chiave **class**

Una classe può avere **attributi** di tipo primitivo o non primitivo (altre classi, array)

```
1  1
   2  class Auto{
   3
   4      String colore;
   5      String marca;
   6      int annoImmatricolazione;
   7
   8  }
```

Oggetto

Per creare un oggetto uso la parola chiave **new** seguita dal nome della classe e ()

Accedo agli attributi dell'oggetto, in scrittura e in lettura, con la notazione **.**

```
1  // creo una istanza di Auto
2  // la assegno a una variabile di tipo Auto
3  Auto miaAuto = new Auto();
4
5  // posso accedere agli attributi
6  // del mio oggetto con il .
7
8  // in scrittura
9  miaAuto.colore = "bianco";
10 miaAuto.marca = "Toyota";
11
12 // in lettura
13 System.out.println(miaAuto.colore);
14 System.out.println(miaAuto.marca);
15
```

Posso definire variabili di tipo Auto: creando una classe ho definito un nuovo tipo di dato non primitivo

Attributi

La classe è una sola ma posso creare tanti oggetti

Ogni oggetto ha gli stessi attributi, ma ognuno col suo valore

Gli attributi sono delle variabili e possono cambiare valore

```

Auto miaAuto1 = new Auto();
Auto miaAuto2 = new Auto();


miaAuto1.colore = "bianco";

miaAuto2.colore = "rosso";
```

Il valore che gli attributi di un oggetto assumono in ogni momento ne definiscono lo **stato**

Metodi

La classe può avere dei metodi = dei sottoprogrammi che servono a compiere delle azioni

i metodi hanno:

- un tipo di dato restituito
- un nome
- dei parametri in ingresso (opzionale)

```
1  1
   2  class Auto{
   3
   4      String colore;
   5
   6      void parti(){
   7          System.out.println("VROOM!");
   8      }
   9
  10      void cambiaColore(String nuovoColore){
  11          colore = nuovoColore;
  12      }
  13
  14      String saluta(){
  15        String saluto = "Ciao sono un'auto";
  16        saluto += " di colore " + colore;
  17        return saluto;
  18      }
  19  }
```

Se un metodo ritorna un tipo di dato non vuoto (void) deve avere una istruzione di **return**

Metodi

Posso eseguire un metodo di un oggetto con la notazione **.**

la chiamata a un metodo è il nome del metodo seguito da **()**

Se il metodo richiede parametri, vanno inseriti nelle **()** dei valori o delle variabili del **tipo** di parametro, nell'**ordine** in cui sono stati definiti

```
1  1
   2  Auto miaAuto1 = new Auto();
   3
   4  // assegno valore a un attributo
   5  miaAuto1.colore = "bianco";
   6
   7  // chiamo un metodo senza parametri
   8  miaAuto1.parti();
   9
  10  // chiamo un metodo con parametri
  11  miaAuto1.cambiaColore("nero");
```

Metodi

```
1  1
   2  class Auto {
   3
   4      String colore;
   5      int numeroDiAirbag;
   6      boolean ibrida;
   7
   8      void definisciOptional (String coloreOption,
   9      int numeroDiAirbagOption, boolean ibridaOption){
  10          colore = coloreOption;
  11          numeroDiAirbag = numeroDiAirbagOption;
  12          ibrida = ibridaOption;
  13      }
  14
  15
  16  }
```

```
1  1
   2  miaAuto = new Auto();
   3
   4  miaAuto.definisciOptional("verde", 4, true);
```

Un metodo l'abbiamo già visto... qualche idea?

Metodo main

Il metodo **main** ci permette di lanciare l'esecuzione della classe

Quando scriviamo un programma ci capiterà di definire molte classi e di importarne altre, ma solo una classe ha il metodo main

LIVE CODING

Costruttori

Un costruttore in Java è un metodo speciale che si usa per inizializzare gli oggetti

Il costruttore è chiamato quando un oggetto di una classe viene creato

Può essere usato per definire il valore iniziale degli attributi dell'oggetto

```
1   1
    2 class Monitor {
    3
    4     boolean acceso;
    5
    6     // costruttore
    7     Monitor(){
    8
    9         // inizializza l'attributo
   10          acceso = false;
   11     }
   12 }
```

Costruttori - Importante

Il costruttore deve avere lo stesso nome della classe e non può avere un tipo di ritorno

Il costruttore è chiamato nel momento il cui l'oggetto viene creato (parola chiave **new**)

Tutte le classi devono avere un costruttore di default: se non lo definisci tu, Java ne crea uno per te. Ma in quel caso non puoi inizializzare i valori degli attributi.

Costruttori con parametri

Un costruttore può anche ricevere parametri, che servono ad inizializzare gli attributi dell'oggetto.

All'interno del costruttore si possono eseguire anche altre operazioni se serve (ad esempio richiamare metodi).

```
1  1
   2  class Monitor {
   3     boolean acceso;
   4     String modello;
   5     int numeroDiSerie;
   6
   7     // costruttore con parametri
   8     Monitor(String nomeModello){
   9       modello = nomeModello;
  10
  11       // richiamo un metodo
  12       printStatus();
  13     }
  14
  15     void printStatus() {
  16        System.out.println("Sono acceso");
  17     }
  18
  19  }
```

La parola chiave this

La parola chiave **this** si riferisce all'oggetto corrente

L'uso più comune di **this** è quello di eliminare ambiguità tra gli attributi della classe e parametri che hanno lo stesso nome.

Un altro uso del this è richiamare altri costruttori.

```
1  1
   2  class Monitor {
   3
   4      boolean acceso;
   5      String modello;
   6
   7
   8      // costruttore con parametri
   9      Monitor(String modello, boolean acceso) {
  10
  11        // richiama l'altro costruttore
  12        this();
  13
  14        this.modello = modello;
  15        this.acceso = acceso;
  16      }
  17
  18      Monitor() {
  19
  20         printStatus();
  21      }
  22
  23      void printStatus() {
  24
  25         System.out.println("Sono acceso");
  26      }
  27
  28  }
```

LIVE CODING

Garbage collector

Facciamo pulizia

Abbiamo visto come creare nuovi oggetti con la keyword **new**

Ogni oggetto occupa spazio in memoria, e la memoria non è infinita...

Come facciamo a sbarazzarci degli oggetti che non ci servono più, liberando spazio in memoria?

In linguaggi come C o C++ è il programmatore che deve occuparsi di distruggere gli oggetti (e se si dimentica si rischia un *memory leak*)

In Java, invece, c'è un processo automatico chiamato Garbage Collector, che si occupa di gestire la memoria e di fare pulizia al posto nostro