

# M03. PROGRAMACIÓ

## UF4. Object Oriented Programming

# 1. INTRODUCCIÓ

# LET'S PLAY!

Feu grups de 3 persones i  
anoteu les  
característiques i què es  
pot fer amb els elements  
que es mostren a  
continuació

---

# QUINES CARACTERÍSTIQUES TÉ I QUÈ ES POT FER AMB UN JOC DE CARTES?

Característiques



Accions



5 minuts!

# QUINES CARACTERÍSTIQUES TÉ I QUÈ ES POT FER AMB UN LLUM?



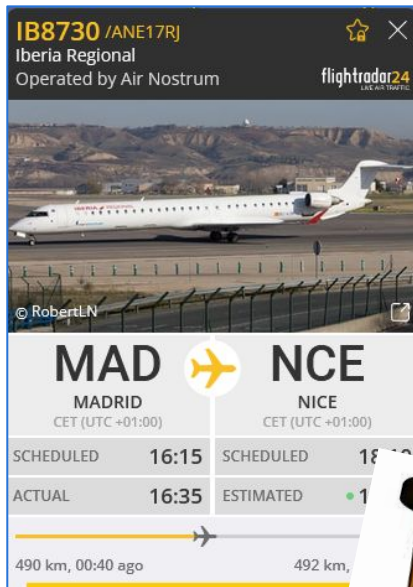
**Característiques**

**Accions**



**5 minuts!**

# QUINES CARACTERÍSTIQUES TÉ I QUÈ POT FER UN VOL?



Característiques

Accions



5 minuts!

# PARADIGMES DE LA PROGRAMACIÓ. ORIENTACIÓ A OBJECTES

- **imperatiu**: el programador instrueix a la màquina com canviar el seu estat
  - **procedimental**: agrupa les instruccions en procediments
  - **orientat a objectes**: agrupa les operacions amb la part de l'estat en el qual operen
- **declaratiu**: es declaren les propietats del resultat desitjat, però no com calcular-lo
  - **funcional**: el resultat desitjat es declara com el valor d'una sèrie d'aplicacions de funció
  - **lògic**: el resultat desitjat es declara com la resposta a una pregunta sobre un sistema de fets i regles
  - **matemàtic**: el resultat desitjat es declara com la solució d'un problema d'optimització
  - **reactiu**: es declara el resultat desitjat amb fluxos de dades i la propagació del canvi

# ORIENTACIÓ A OBJECTES

En la OOP, tot es treballa dins el concepte d'*objecte*. És a dir, tant les entitats materials (porta, banc, pedra) com els immaterials (compte corrent, hipoteca, factura,...) es defineixen com a objectes. D'aquests, ens interessa definir les seves característiques (edat, material, mida, estat) com els seus comportaments (accions que poden realitzar aquests objectes o que es poden realitzar sobre ells).





# ORIENTACIÓ A OBJECTES. JAVA

*Java* és un llenguatge d'alt nivell i orientat a objectes que es va llançar l'any 1995 per Sun Microsystems. És un llenguatge de propòsit general, la filosofia del qual resideix en el lema “Write once, run anywhere”, ja que gràcies a la seva màquina virtual (JVM) permet executar un codi implementat en Java a qualsevol plataforma, sense haver de tornar-lo a “compilar”. Les aplicacions Java es compilen i donen com a resultat un fitxer bytecode. Aquest és el fitxer que la JVM específica de la plataforma s'encarrega d'executar. Aquesta característica, que permet la portabilitat dels programes, juntament amb la gestió dinàmica de la memòria (Garbage Collector) han fet el llenguatge, i tota la plataforma que el suporta, molt popular.

Actualment s'utilitza pel desenvolupament d'aplicacions web (applets, servlets, jsp), aplicacions d'escriptori (SWING i AWT), SGI (CRM, ERPs, ...) i d'aplicacions mòbils (android), a més de sectors emergents com ara IA, Big Data (Cloudera, Spark i Scala) i Machine Learning.

# ORIENTACIÓ A OBJECTES. OBJECTES I CLASSES

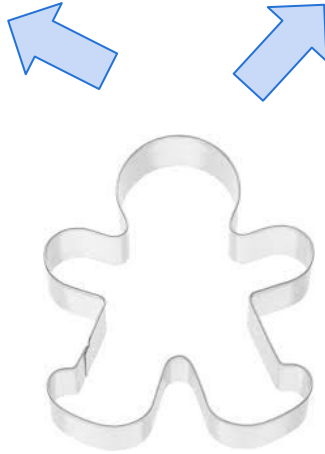
Un **objecte** és una representació (informàtica) d'una entitat del món real (ex: animal, estudiant, cercle, botó,...). Una **classe** és una plantilla (*blueprint*) que defineix les característiques (*atributs*) i els comportaments (*mètodes*) d'un objecte.

És a dir, per a crear (*instanciar*) un objecte, hem de tenir definida en primer lloc la seva classe.

Podem trobar classes ja definides a la API de Java i es poden dissenyar noves classes, específiques per a cada projecte.

Dins la classe, es defineixen els **atributs** (característiques) i els **mètodes** (comportaments) de l'objecte.

# ORIENTACIÓ A OBJECTES. OBJECTES I CLASSES



# ORIENTACIÓ A OBJECTES. OBJECTES I CLASSES

Els **atributs** (*instance fields*) són les característiques d'un objecte que defineixen el seu estat, i el seus valors s'emmagatzemen en variables declarades dins la classe, que mantindran valors independents per a cada instància (objecte creat). Els **mètodes** (*comportaments*) són similars a les funcions pròpies dels llenguatges procedimentals i defineixen accions que poden realitzar els objectes o que es poden realitzar sobre ells.

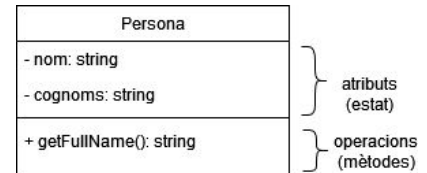


## Pseudocodi

```
Classe Persona
  Atributs
    nom, cognoms: string;
  Operacions
    getFullName(): string;
End Persona
```



## Diagrama UML



# ORIENTACIÓ A OBJECTES. OBJECTES I CLASSES



**Company name:** Maserati  
**Model name:** GranTurismo Sport  
**Fuel type:** Premium Gasoline  
**Mileage:** 12 l/100km  
**Price:** 403.000 USD

## Procedimental Programming

```
public class Main {  
    public static void main(String[] args) {  
        String companyName, modelName, fuelType;  
        float mileage = 15f;  
        double price;  
  
        if(mileage>50){  
            }  
        }  
    }
```

# ORIENTACIÓ A OBJECTES. OBJECTES I CLASSES

## Object Oriented Programming

```
import cat.institutmvm.utils.Car;

public class Main {
    public static void main(String[] args) {
        Car cotxe1 = new Car("Maserati");
        System.out.println(cotxe1.getCompanyName());

        Car cotxe2 = new Car("Renault");
        System.out.println(cotxe2.getCompanyName());
    }
}
```

```
public class Car {
    private String companyName, modelName, fuelType;
    private float mileage = 15f;
    private double price;

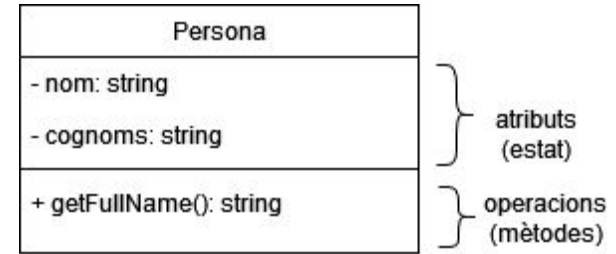
    public Car(String companyName) {
        this.setCompanyName(companyName);
    }

    public String getCompanyName() {
        return this.companyName;
    }

    public void setCompanyName(String companyName) {
        this.companyName = companyName;
    }
}
```

# ORIENTACIÓ A OBJECTES. CARACTERÍSTIQUES BÀSIQUES

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.



# ORIENTACIÓ A OBJECTES. CARACTERÍSTIQUES BÀSIQUES. ATRIBUTS

Els atributs d'un objecte es poden definir mitjançant variables de tipus primitius (comunes amb els llenguatges procedimentals) i/o instàncies d'altres objectes. A més a més, segons definim aquestes variables (modificadors), aquestes poden ser de dos tipus:

- **variables d'instància**: pertanyen a la instància de l'objecte i els seus valors són únics per a aquesta instància. Si no existeix la instància, no existeixen ni es pot accedir a aquestes variables
- **variables de classe**: aquests atributs existeixen sempre i es pot accedir als seus valors sense haver d'instanciar un objecte de la classe. Els seus valors són comuns per a totes les instàncies i, si una el modifica, aquest atribut serà modificat per a totes.

La paraula reservada **"this"** permet fer referència a l'objecte actual sobre el qual s'està executant el mètode.



# ORIENTACIÓ A OBJECTES. CARACTERÍSTIQUES BÀSIQUES. MÈTODES

Cada classe pot tenir un conjunt de mètodes associats amb ella, diversos d'ells amb el mateix nom (*overloading*). El *method signature* (signatura de mètode), diferencia cada mètode, ja que està definit pel nom del mètode i els arguments que rep.

La **signatura del mètode** està formada només per dos dels components de la declaració del mètode: el **nom del mètode** i els tipus i nombre d'arguments.

El tipus de retorn d'un mètode no forma part de la seva signatura, i per tant no serveix per a diferenciar mètodes (en cas d'*overloading* i/o *overriding*).

# ORIENTACIÓ A OBJECTES. CARACTERÍSTIQUES BÀSIQUES. MÈTODES

Classificació dels mètodes:

- **Accessor methods**

- anomenats habitualment "getter" methods
- retornen el valor d'una instància específica

- **Mutador methods**

- anomenats habitualment "setter" methods
- modifiquen o estableix el valor d'un atribut específic

- **Functional methods**

- retornen o realitzen algun tipus de funcionalitat (comportament) per l'objecte

# ORIENTACIÓ A OBJECTES. CARACTERÍSTIQUES BÀSIQUES. CONSTRUCTORS

Cas particular dels mètodes. D'accés públic i declarats amb el mateix nom que la seva classe, permeten crear una instància d'un objecte.

- els constructors s'invoquen amb la paraula clau *new*
- es pot declarar més d'un constructor en una declaració de classe, sempre i quan tinguin signatures diferents (overloading)
- la crida al constructor ha d'incloure arguments que coincideixin amb la llista d'arguments del constructor
- si no es declara un constructor, Java proporcionarà un constructor per defecte (blank)
- quan es declara un constructor propi, el constructor per defecte proporcionat per Java ja no estarà disponible
- els constructors assignen valors inicials a les variables d'instància d'una classe
- els constructors dins d'una classe es declaren com a mètodes però no retornen cap valor

# ORIENTACIÓ A OBJECTES. ENCAPSULAMENT

Ocultació de les decisions de disseny, de manera que canvis en una classe afectin el mínim possible el programari ja desenvolupat. Els **access modifiers** (modificadors d'accés) especifiquen l'accessibilitat per a modificar variables, mètodes i classe.

Hi ha quatre tipus de modificadors d'accés:

- **public**: l'element és accessible per a tothom. El menys restrictiu.
- **protected**: l'element és accessible només si la classe accessora (accessing class) està al mateix package o si la classe accessora és una subclasse dins de qualsevol altre package.
- **private**: només és accessible des de dins de la mateixa classe. És el més restrictiu.
- **default** (blank): també conegut com "package private", s'aplica quan no s'indica el modificador d'accés. Només és accessible per classes i subclasses del mateix package (default).

# ORIENTACIÓ A OBJECTES. ENCAPSULAMENT

Els **access modifiers** especifiquen l'accessibilitat per canviar variables, mètodes i classes. Els camps són normalment privats i els mètodes públics. En cas de declarar una variable o mètode sense cap modificador de control d'accés, estant disponibles per a altra classe dins del mateix paquet.

- Qualsevol variable declarada sense un modificador pot ser llegida o modificada per una altra classe del mateix paquet.
- Qualsevol mètode declarat sense un modificador pot ser invocat per qualsevol classe en el mateix paquet.

Un **objecte** ha de protegir sempre el seu **estat** (atributs), per tal de no permetre accessos directes. Els **atributs** han de ser declarats de manera **privada** i **accessibles a través dels mètodes**.

# ORIENTACIÓ A OBJECTES. ENCAPSULAMENT

Els **non-access modifiers** (modificadors de no accés), no canvien l'accessibilitat de variables i mètodes, sinó que poden alterar el comportament dels mètodes i els hi proporcionen propietats especials:

- **static**: en el cas dels atributs i els mètodes, fa que siguin dependents de la classe (no de la instància). En el cas de les classes, encapsulen funcionalitats típiques a utilitzar en les nostres aplicacions
- **final**: aplicable a les classes, mètodes i variables per a indicar que no es canviaran
- **abstract**: aplicable a classes i mètodes, els defineix com a abstractes (declarats però no definits)
- **synchronized**: utilitzat en la sincronització de threads (fils)

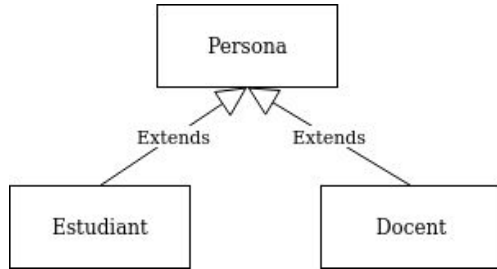
# ORIENTACIÓ A OBJECTES. HERÈNCIA

L'herència és un dels conceptes fonamentals de la programació orientació a objectes i té gran importància en el desenvolupament de software. S'utilitza per crear classes noves a partir de les ja definides. Una classe nova pot ser una extensió o bé una restricció de la classe original

L'herència es pot implementar de dues maneres diferents:

- **herència simple** (jeràrquica): una classe només pot heretar d'una classe, és a dir, només pot estendre les característiques i comportaments d'una única classe.
- **herència múltiple**: cada classe pot heretar mètodes i/o atributs de qualsevol nombre de superclasses (no hi ha limitació).

# ORIENTACIÓ A OBJECTES. HERÈNCIA



## Cas pràctic:

- estudiants i docents tenen atributs comuns (nom, cognom, dni) però d'altres específics per a cada objecte (estudiant: RALC, docent: anys de docència).
- les característiques i comportaments genèrics es defineixen dins la superclasse, mentre que les especificitats es defineixen a les subclasses.

Aquesta funcionalitat permet **simplicar codi** i **evitar redundàncies**.



# ORIENTACIÓ A OBJECTES. TIPOLOGIA DE CLASSES

En una jerarquia de classe, **com més important és aquesta classe, més abstracta és la seva definició**. Una classe per sobre de la jerarquia d'altres classes pot definir només el comportament i atributs comuns a totes les classes. Comportaments i atributs més específics s'hauran d'especificar en classes situades en la part baixa de la jerarquia.

Les classes en Java es classifiquen en:

- **Top-level** classes
- **Abstract** classes
- **Interfaces**

# ORIENTACIÓ A OBJECTES. TIPOLOGIA DE CLASSES

## TOP-LEVEL (base)

### Visibilitat

- **public**
  - única classe al codi font (.java)
  - nom de la classe = nom fitxer  
codi font
- **default/package private**
  - diverses classes dins el mateix  
fitxer
  - totes default + 1 public
  - indiferent nom del fitxer

### Tipus

- **helpers** (static)
- **blueprints** (modelen objectes)

## ABSTRACT

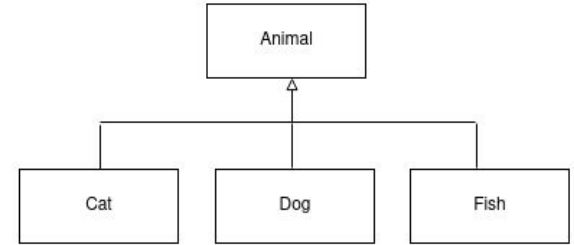
### Propòsit

**Capturar** característiques comuns d'objectes relacionats (concepte, no objecte real).

## INTERFACE

### Propòsit

**Descriure** comportaments capturats per un grup de mètodes, però que no aporten informació sobre l'objecte en si.



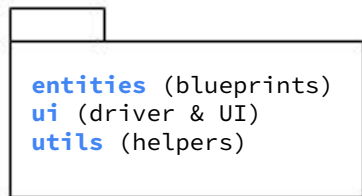
### Tipus de mètodes

- **abstract** (versions < Java 8)
- **default** (Java 8)
- **static**
- **private** } (versions > Java 8)

# ORIENTACIÓ A OBJECTES. ORGANITZACIÓ I VISIBILITAT

**Package:** organitzador de classes

cat.mvm (url invertida)

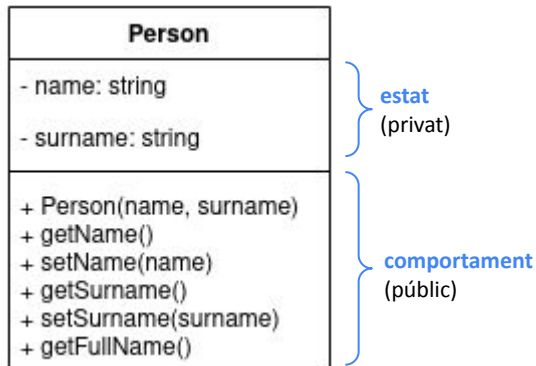


## Importacions

Permet instanciar un objecte d'una classe des d'un altre package o invocar algun dels seus mètodes, sense indicar el FQCN.

```
import cat.mvm.entities.Person; (específica)
import cat.mvm*; (genèrica)
```

cat.mvm.entities.Person ← FQCN (Fully Qualified Class Name)



## Declaració

visibilitat	àmbit	tipus	nom
private		int	age;
private	static final	int	MIN_AGE = 18;

## Modificació/accés

A través de gettes i setters (públics)

# ORIENTACIÓ A OBJECTES. STATIC METHODS AND FIELDS

## Static Fields

- pertanyen a la classe i no a l'objecte (instància)
- s'inicialitzen només una vegada, a l'inici de l'execució. S'inicialitzaran abans de la inicialització de qualsevol variable d'instància
- una sola còpia pot ser compartida per totes les instàncies de la classe
- es pot accedir directament a un atribut estàtic pel nom de classe i no necessita cap objecte

# ORIENTACIÓ A OBJECTES. STATIC METHODS AND FIELDS

## Static Methods

- són mètodes que pertanyen a la classe i no a l'objecte.
- només poden accedir a dades estàtiques (no poden accedir a dades no estàtiques, variables d'instància)
- poden cridar només a altres mètodes estàtics i no es pot invocar un mètode no estàtic a partir d'ell.
- es pot accedir a un mètode estàtic directament pel nom de la classe i no necessita cap objecte
- no poden fer referència a “this” o “super”

# ORIENTACIÓ A OBJECTES. INTERFACES

Una **interface** (interfície) és una col·lecció de comportaments abstractes que poden ser adoptats per qualsevol classe, sense haver de ser heretats des d'una superclasse. Només conté definicions de mètodes abstractes i constants; no hi ha variables d'instància o implementacions de mètode.

Per accedir als mètodes d'interfície, la interfície ha de ser "implementada" per una altra classe amb la paraula reservada *implements*. El cos del mètode d'interfície és proporcionat per la classe implementadora.

Java no és compatible amb "l'herència múltiple", però es pot aconseguir amb interfícies, perquè una classe pot implementar múltiples interfícies (separant-les amb una coma).

# ORIENTACIÓ A OBJECTES. POLIMORFISME

És la característica dels objectes que permet implementar un mètode amb el mateix nom en diverses classes, cadascun amb una funcionalitat diferent. Aquesta particularitat es dona amb l'herència i l'abstracció.

Dos tipus de polimorfisme:

- *Overloading*: polimorfisme en temps de compilació. En la sobrecàrrega del mètode, més d'un mètode comparteix el mateix nom del mètode però amb una method signature diferent definida dins la classe.
- *Overriding*: polimorfisme en temps d'execució. En la sobreescritura de mètodes, la classe derivada proporciona la implementació específica del mètode que ja proporciona la classe base o la classe pare. El tipus de retorn ha de ser el mateix o covariant (el tipus de retorn pot variar en la mateixa direcció que la classe derivada).

# ORIENTACIÓ A OBJECTES. POLIMORFISME

Per diferenciar:

- quan dos o més mètodes de la mateixa classe tenen el mateix nom però paràmetres diferents, s'anomena **overloading**.
- quan la signatura del mètode (nom i arguments) és la mateixa a la superclasse i a la subclasse, s'anomena **overriding**.



# ORIENTACIÓ A OBJECTES. POLIMORFISME VS OVERLOADING

Els termes polimorfisme i sobrecàrrega es poden veure com a complementaris. El polimorfisme és un concepte unificador mentre que la sobrecàrrega és un concepte seleccionador.

- Overloading (sobrecàrrega)
  - Java selecciona el mètode al qual s'invoca segons la seva signatura (method signature). La sobrecàrrega ens permet seleccionar el mètode adequat
  - No confondre la sobrecàrrega amb la sobrescritura (@Override)
    - Sobrecarregar significa definir nous mètodes
    - Sobreescriure significa ocultar un mètode amb una nova definició d'aquest mateix mètode
    - La sobrecàrrega no implica herència, la sobrescritura si
- Polimorfisme
  - directament relacionat amb l'herència
  - permet que allà on utilitzem un objecte de tipus superclasse, també podem utilitzar qualsevol instància d'alguna de les seves subclasses
  - també funciona amb interfaces

# ORIENTACIÓ A OBJECTES. POLIMORFISME VS OVERLOADING

Overloading	Overriding	Polimorfisme
No implica herència	Implica herència	Directament relacionat amb l'herència
Es defineixen nous mètodes	Permet ocultar un mètode amb una nova definició d'aquest mateix mètode	Permet que allà on utilitzem un objecte de tipus superclasse, també podem utilitzar qualsevol instància d'alguna de les seves subclasses