TEMA 01 – DESENVOLUPAMENT DE PROGRAMARI

Què es un programa?

* Llista d’instruccions
* Realitzades de manera seqüencial
* Programar: desenvolupar el programa

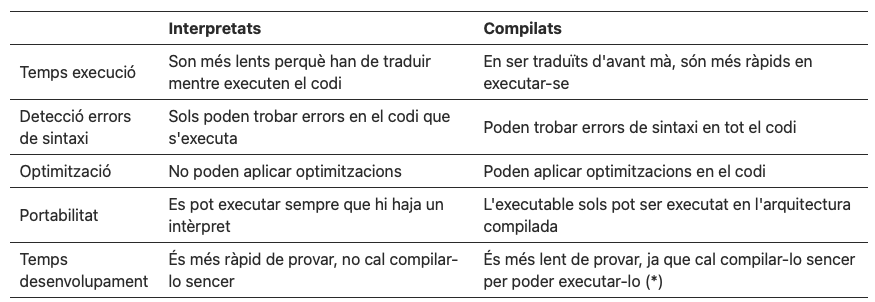
Tipus de llenguatges

* Natural
* Llenguatge de programació
  + Artificial, dissenyat per a controlar el comportament d’una màquina
  + Regles clares
  + Només té una interpretació
  + Classificació
    - Llenguatge màquina: codis directament interpretables pel processador
      * Diferent per a cada processador
    - Baix nivell o assemblador: conjunt de codis que representen instruccions bàsiques del processador
    - Alt nivell: alta abstracció dels detalles
      * Més fàcils d’utilitzar (més pareguts al llenguatge natural)
  + Traducció
    - Interpretats (JS, Python, Ruby, Perl, Bash, PHP)
      * El codi font va traduint-se a mesura que executem el codi
      * El intèrpret traduix una instrucció i el processador l’executa
      * Avantatges
        + Codi font independent de la plataforma
        + Portable
      * Desavantatges
        + Més lent d’execució (traduir i executar a la vegada)
    - Compilats (C, C++, Rust, Pascal, Swift)
      * El compilador traduix tot el codi font a màquina en un executable
      * Després, tot el codi màquina es executat
      * Avantatges
        + Més ràpids en temps d’execució
        + Capaços de trobar errors en tot el codi mentre es compilen
        + Es poden aplicar optimitzacions en el codi
      * Desavantatges
        + Procés de compilació lent
        + Més T per a provar els programes
        + El codi generat sols es pot executar en l’arquitectura del nostre entorn
    - Bytecode (Java, Kotlin, C#)
      * Codi intermedi més abstracte que el codi màquina
      * Generat mitjançant el procés de compilació
      * No traduït a codi màquina
      * S’executa a la màquina mitjançant un intèrpret
        + Aprofita els avantatges d’intertretats i compilats
      * Avantatges
        + Codi optimitzar i es poden trobar errors
        + Ràpid d’executar
        + Portable
        + No depèn de cap arquitectura

Java

* Codi font (.java): conté el programa a executar en llenuatge Java (alt nivell)
  + Consola: java nom.java
* Compilador: compila el codi font a bytecode amb el compilador de Java (JDK)
* Bytecode (.class): programa en llenguatge intermig en binari
  + Consola: javac nom.java
* JVM (Java Virtual Machine): interpreta el bytecode i el traduix a màquina
  + La JVM s’instal·la mitjançant el Java Runtime Environment (JRE) (ve inclòs el JDK)
  + Consola: java nom

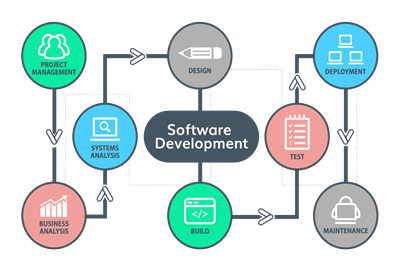




TEMA 02 – LLENGUATGES DE PROGRAMACIÓ

Conceptes bàsics

* Informàtica, binari, digital i software
  + Informàtica:
    - Informació: conjunt dades
    - Automàtica: funciona per si mateix
    - Ciència que estudia el tractament d la informació mitjançant un sistema automàtic
    - Permet automatitzar el processament de la informació
  + Codi binari
    - Unitat bàsica (0, 1): bit
    - 8 bits: byte
    - 1 byte: 256 possibilitats diferents (28)
  + Digital: nformació en forma de bits (≠ analògic)
  + Software: programari
    - Equipament lògic
    - Aplicacions: tasca en concret
    - Sistema operatiu: proporcionar una plataforma única i encarregat de comunicar-se amb el maquinari
* Programar vs Desenvolupar programari
  + Programar: especificar les ordres )instruccions) i dades en un dispositiu que executarà per rebre una sèrie de resultats o provocar un cert comportament
  + Desenvolupar software: tot el procés d’analitzar, dissenyar, provar, documentar i mantindré el programari



Llenguatges de programació (TEMA 01)

Segons el paradigma (manera de treballar i com estan estructurats)

* Paradigma imperatiu
  + Conèixer l’estat de la màquina i modificar-lo mitjançant instruccions
  + Fem servir ordres per a dir al dispositiu què ha de fer i com fer-ho (imperar)
  + La majoria d’arquitectures d’ordinadors seguixen una filosofia imperativa
  + La traducció d’un llenguatge imperatiu a codi màquina es més senzilla
  + Quasi tots els llenguatges són imperatius
  + Tipus:
    - Llenguatges estructurats
      * Sols permeten 3 estructures: seqüència, selecció i iteració
      * Fa innecessari l’ús de salts (go to) i es consideren mala pràctica
      * Es seguixen utilitzant salts com el break en alguns casos
      * Fortan, Algol, Cobol, Basic, Pascal, C, Ada
    - Llenguatges orientats a objectes
      * Estratègia de construcció de programes basada en una abstracció del món real
      * Són combinació de dades (atributs) i mètodes que ens permeten interactuar amb ell
      * Es basa en: abstracció, encapsulació, modularitat, jerarquia i polimorfisme
      * Java, C++, Python, C#
* Paradigma declaratiu
  + Fixen un objectiu, però no el camí per a arribar-hi
  + S’indica quin és el valor que es desitja obtindré
  + El dispositiu troba la solució seguint una lògica interna
  + Llenguatges més propers per a les matemàtiques
  + SQL, Prolog, LISP

Com triar el llenguatge de programació

* Sector productiu al qual va adreçada l’aplicació (medicina, acadèmic, data mining...)
  + Existixen llibreries que faciliten la tarea?
* Tipus de dispositiu on s’executarà
* SO sobre la qual s’executarà

TEMA 03 – ENTORNS DE DESENVOLUPAMENT

Debugger

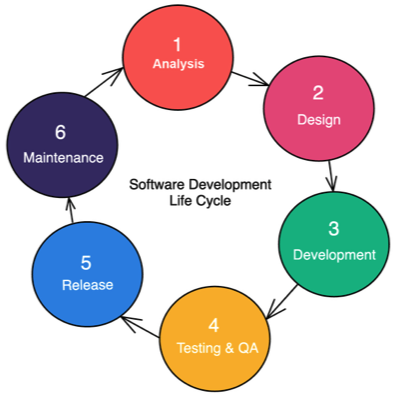
(<https://joapuiib.github.io/itb/DAM-ED/UD1/materials/02_entorns_desenvolupament.html>)

TEMA 04 – ENGINYERIA DE PROGRAMARI

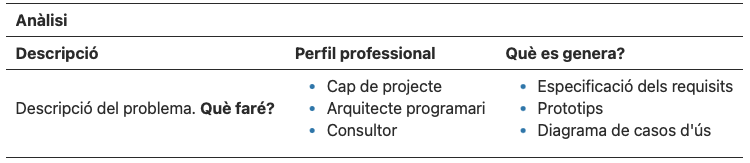
* Què es?
  + Disciplina que engloba ferramentes, recomanacions i mètodes que s’utilitzen en la creació d’un projecte de programari
  + Té la programació com a pilar fonamental, però inclou altres abans i després
* Importància
  + 80% errors es produïen en l’anàlisi prèvia
  + Aspectes importants
    - Latència de decisió (quant més ràpid, millor)
    - Abast mínim (més gran, més índex de fracàs)
    - Responsables del projecte (1 persona millor que junta directiva)

Cicle de vida del programari

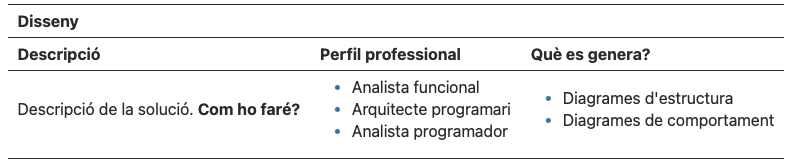
* Fases més comunes



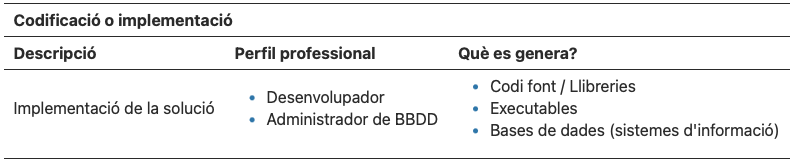
* + Anàlisi



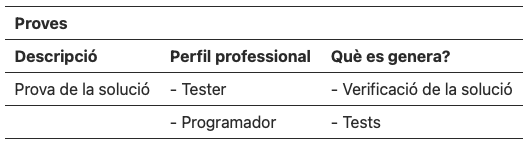
* + Disseny



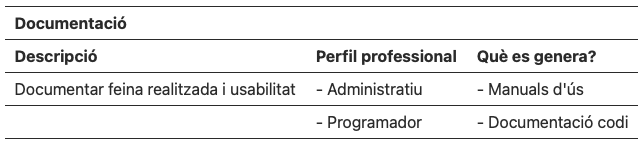
* + Codificació



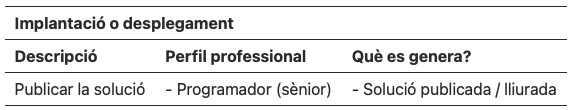
* + Proves



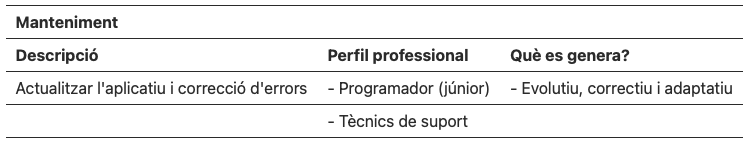
* + Documentació



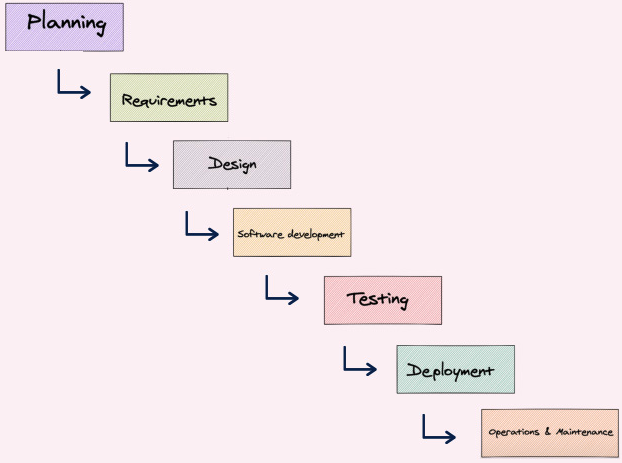
* + Implantació o desplegament



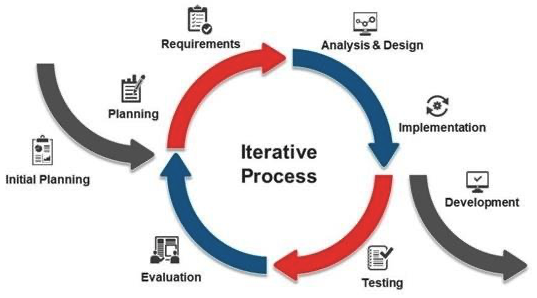
* + Manteniment



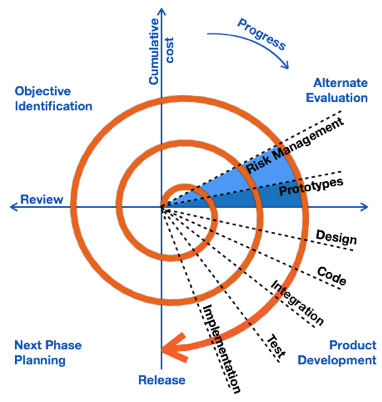
* Tipus de cicles de vida
  + Clàssic o cascada



* + Iteratiu

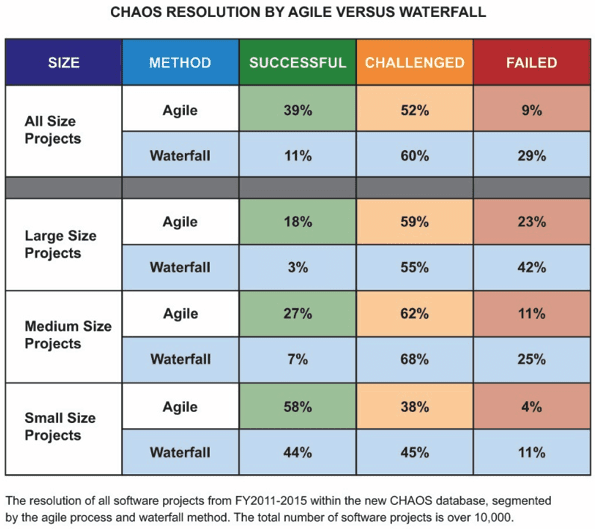


* + Espiral



Metodologies

* Tradicionals: combinacions del cicles anteriors
  + Massa llarg
  + Feedback insuficient
* Metodologies àgils
  + Reajustament continu dels objectius de desenvolupament amb les necessitats i expectatives del client
  + Proporcionen processos de desenvolupament de programari “més lleugers”, ràpids i “àgils”
  + Poden adaptar-se als inevitables canvis en els requisits del client



TEMA 05 – GIT

* Comandes Bash:

<https://joapuiib.github.io/itb/DAM-ED/UD2/materials/06_resum_bash.html>

* Comandes GIT:

<https://joapuiib.github.io/itb/DAM-ED/UD2/materials/05_resum_git.html>

* Tutorial configurar GIT:

<https://joapuiib.github.io/itb/DAM-ED/UD2/exercicis/01_tutorial_git.html>

* Introducció a GIT:

<https://joapuiib.github.io/itb/DAM-ED/UD2/materials/01_git.html>

* Repositori remot:

<https://joapuiib.github.io/itb/DAM-ED/UD2/materials/02_git_remote.html>

* Branques:

<https://joapuiib.github.io/itb/DAM-ED/UD2/materials/03_git_branches.html>

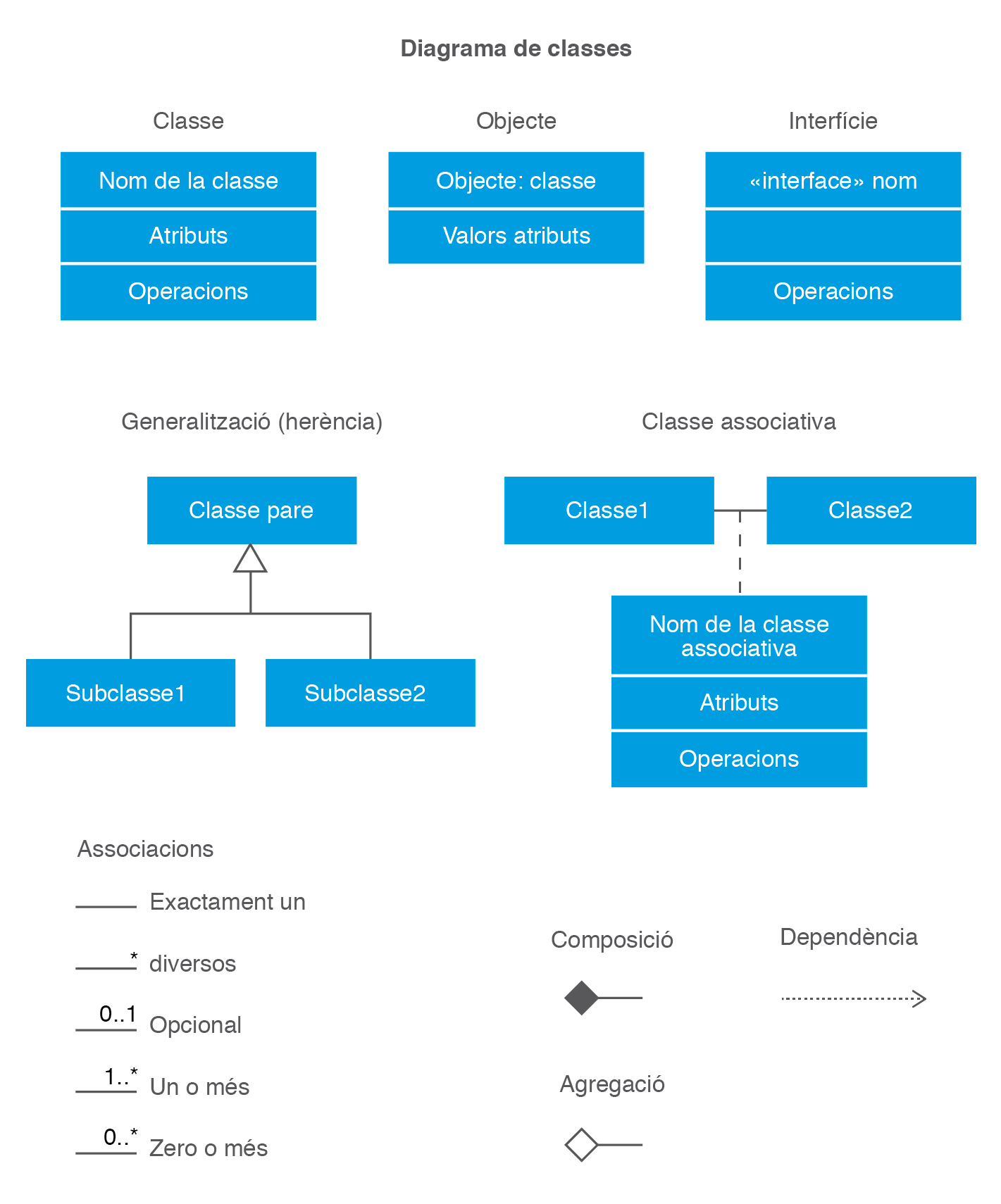
TEMA 06 – DIAGRAMA DE CLASSES

UML

* Unified Modeling Language
* Per a especificar, dissenyar, elaborar i documentar models de sistemes
  + Particularment aplicacions informàtiques
* És el llenguatge de modelització de sistemes més conegut i utilitzat
* Avantatges:
  + Notació gràfica concreta i fàcil d’interpretar
    - Completada amb explicacions escrites
  + Grau de detall que es considere oportú
  + Permet tindre una visió global de les característiques del sistema a implementar
  + Promou la reutilització
* Cal tindre en compte que:
  + Es una notació, no una metodologia
  + No és un llenguatge de programació
  + Pot resultar complex obtindré un coneixement domplet de les possibilitats del llenguatge

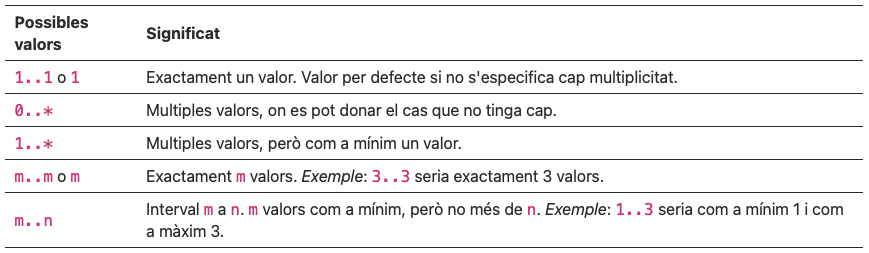
Diagrama de classes

* Diagrama estàtic
* Dels més utilitzats en les metodologies d’anàlisis i de disseny que es basen en UML
* Representa les classes que seran utilitzades dins del sistema i les relacions que existixen entre elles
* Conceptes que ajudaran a entendre’n la creació i el funcionament en la seua totalitat:
  + Classe, atribut i mètode (operacions o accions)
  + Visibilitat
  + Objecte. Instanciació
  + Relacions. Herència, composició i agregació
  + Classe associativa
  + Interfícies

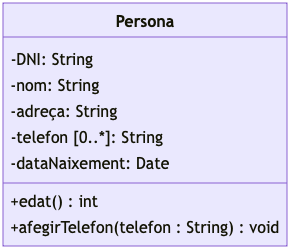


Classes. Atributs i mètodes

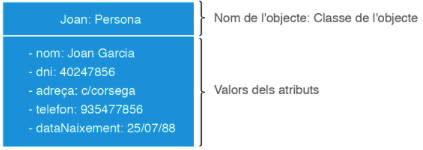
* Classe: representació d’un objecte del món real o abstracte
* Composat d’atributs (característiques de l’objecte)
* I mètodes (accions de l’objecte)
* Notació UpperCamelCase
* Atributs:
  + També propietats o característiques
  + Dades detallades que contenen els objectes
  + lowerCamelCase
  + Tipus primitius: int, char, boolean, float...
  + Tipus derivats String, Date... (altres classes)
* La multiplicitat indica quants diferents valors poden haver en un atribut:



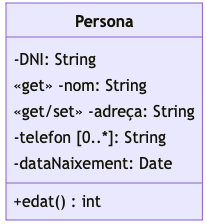
* Mètodes:
  + Canvis sobre els atributs, càlculs, obrir l’objecte, tancar-lo, carregar-lo...
  + Especificar nom i paràmetres que reb
    - Paràmetres: nom, tipus i multiplicitat
  + Poden llançar excepcions, que són errors que poden ocórrer durant l’execució de codi
    - Si el mètode pot llançar alguna excepció concreta, cal indicar-ho
* Representació:
  + Nom de la classe
  + Atributs: visibilitat i tipus
  + Mètodes: visibilitat, paràmetres i **tipus del valor de retorn**



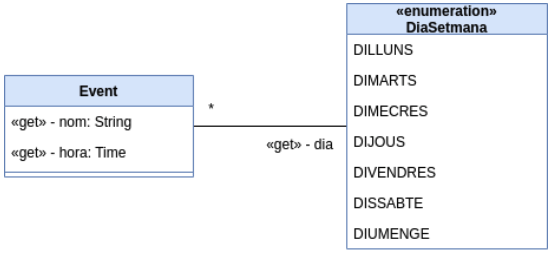
* Objectes. Instanciació
  + Instanciació d’una classe



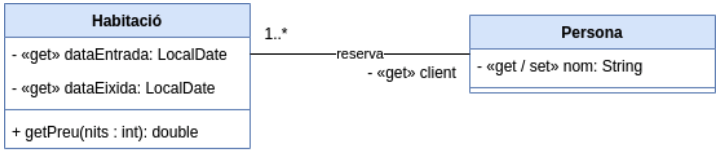
* Visibilitat
  + Públic (+): per tots els altres elements del sistema
  + Privat (-): només pels elements continguts dins el mateix objecte
  + Protected (#): només per als elements del seu mateix tipus i les seues herències
  + Get i Set es poden incloure en el programa, però realment no està definit en el model UML



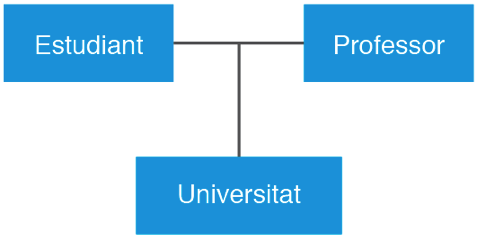
* Enumeracions
  + Classes que representen un tipus de dada que sols permet un conjunt definit de valors



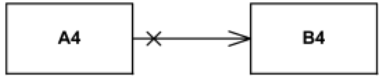
* Relacions entre classes
  + Tenen assignada una multiplicitat
  + Les relacions entre les diferents classes generalment s’anomenen associacions
  + Es representent mitjançant una línia continua
  + Binàries:



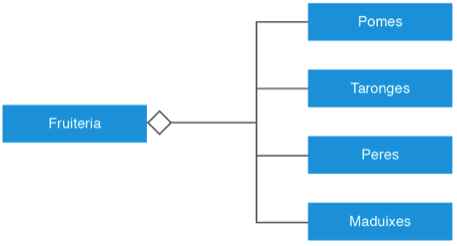
* + Ternaries:



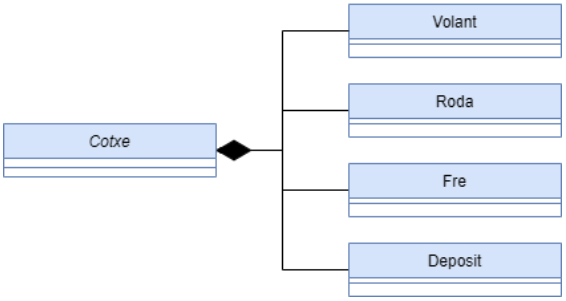
* Navegabilitat
  + No especificat: per defecte
  + Navegable: fletxa oberta al final de l’associació
  + No navegable: X al final de l’associació



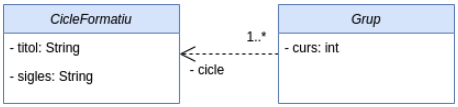
* + - A4 pot accedir a B4, però B4 no pot accedir a A4
* Agregació
  + Relació del tipus tot-part
  + Objecte base i objecte/s inclòs en l’objecte base
  + Si desapareix l’objecte base, els inclosos NO desapareixen i podran continuar existint amb les seues funcionalitats pròpies
  + Rombe buit en l’objecte base i línia continua



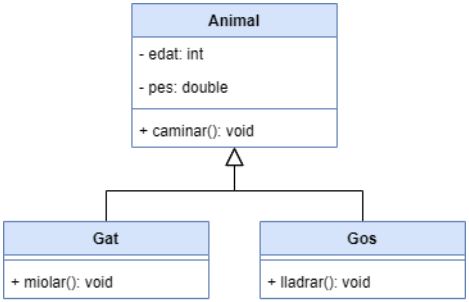
* Composició
  + Paregut a agregació
  + Si deixa d’existir l’objecte base, deixa també d’existir l’objecte inclòs
  + Rombe pintat en l’objecte base i línia continua



* Dependència
  + Tipus de relació
  + Línia discontinua i fletxa al dinal
  + Es tracta d’una relació semàntica
    - Si hi ha un canvi en l’objecte independent, l’objecte dependent es vorà afectat



* + - Grup es dependent, CicleFormatiu és independent
* Herència, especialització o generalització
  + Es dona entre dos classes on hi ha un vincle que es pot considerar d’herència
  + Superclasse 🡪 filles
  + Els fills hereden els atributs, els mètodes i el comportament del pare
  + Fletxa que ix del fill i acaba en el pare



* Classe associativa
  + Quan una associació té propietats o mètodes propis
  + Unida a la línia de l’associació mitjançant una línia discontinua

