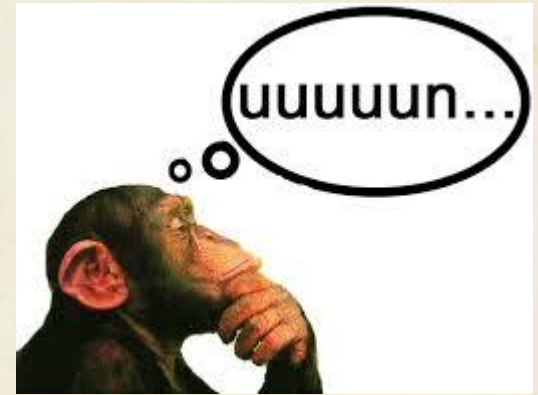


M05 - Entorns de desenvolupament

UF1. DESENVOLUPAMENT DEL PROGRAMARI

Programa informàtic



Un **programa informàtic** és un conjunt d'esdeveniments (instruccions) ordenats de manera que se succeeixen de forma seqüencial en el temps, un darrere l'altre, dins d'un sistema informàtic.

Programa informàtic

Per exemple, el programa **d'un robot de cuina** per fer una crema de pastanaga seria:



1. Espera que introduïu les pastanagues ben netejades, una patata i espècies al gust.
2. Gira durant 1 minut, avançant progressivament fins a la velocitat 5.
3. Espera que introduïu llet i sal.
4. Gira durant 30 segons a velocitat 7.
5. Gira durant 10 minuts a velocitat 3 mentre cou a una temperatura de 90 graus.
6. S'atura. La crema de pastanaga està llesta!

Programa informàtic



- **El conjunt d'ordres de l'exemple anterior no és arbitrari**, sinó que serveix per dur a terme una tasca de certa complexitat que no es pot fer d'un sol cop.
- **S'ha de fer pas per pas. Totes les ordres estan vinculades entre sí** per arribar a assolir aquest objectiu i, sobretot, **és molt important la posició en què es duen a terme.**

Programa informàtic



Per tant, refinant més la definició:

- Un **programa informàtic** no és més que un seguit d'ordres (instruccions) que es porten a terme seqüencialment, aplicades sobre un conjunt de dades.
- Quines dades processa un programa informàtic? Això dependrà del tipus de programa: processador de textos, navegadors web, ...

Aplicació Informàtica

Una **aplicació informàtica** és un o més programes que permeten als usuaris dur a terme tasques específiques

Exemples:

- portar la gestió acadèmica de l'institut
- gestió d'aeroport
- nòmines en una empresa
- venda d'entrades per a espectacles
- gestió d'un magatzem
- gestió d'una lliga de futbol
- portar la gestió d'un concessionari d'automòbils

l'objectiu del cicle formatiu és que obtingueu les habilitats necessàries per desenvolupar aplicacions en diferents plataformes !!!)



Aplicació Informàtica

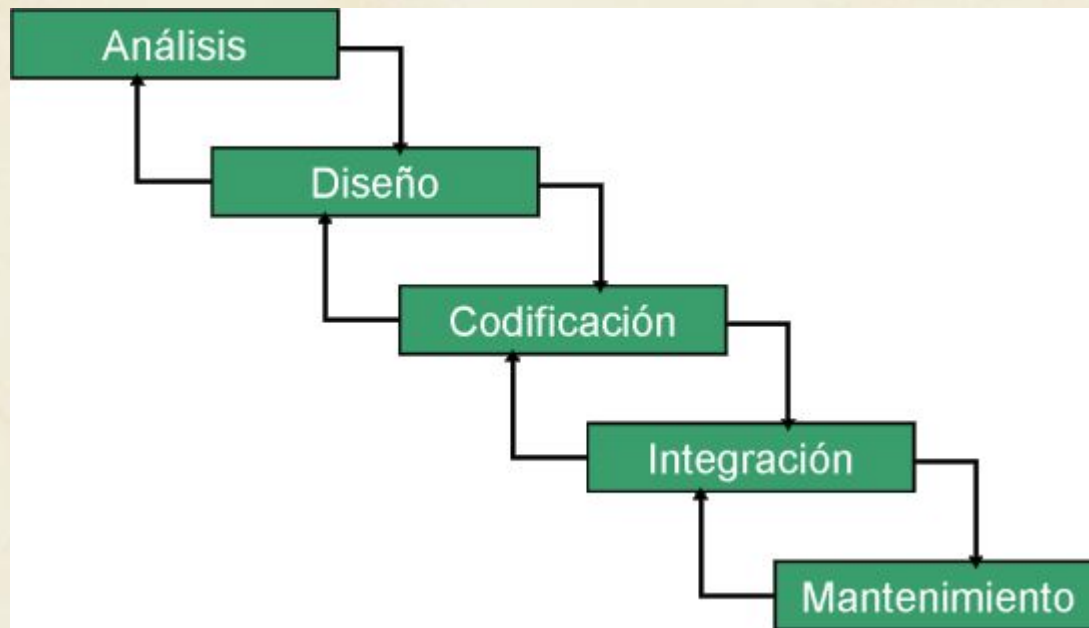
- El procés de creació de qualsevol aplicació informàtica segueix un procediment perfectament planificat i desenvolupat.



- Aquest procediment **comença amb la concepció de la idea o la funcionalitat** que haurà de satisfer aquesta aplicació **fins a la generació d'un o diversos fitxers que permetin la seva execució exitosa.**

Aplicació Informàtica

- Les tasques a desenvolupar en la creació de l'aplicació són:



- A tota aquesta seqüència d'activitats es diu **Cicle de vida d'una aplicació**.

Procés de creació d'un programa

- Primer analitzar el problema i dissenyar els algorismes.
- Segon, escriure en un fitxer el seguit d'instruccions que es vol que l'ordinador executi (codificar i provar).
- Cal tenir present que les instruccions:
 - A) Hauran de seguir unes pautes determinades en funció del llenguatge de programació escollit.
 - B) Hauran de seguir un ordre determinat que donarà sentit al programa escrit.



first, solve the problem.
then, write the code.

Codi font

- El conjunt de fitxers de text resultants on es troben les instruccions es diu que contenen el **codi font**.
- Aquest codi font pot ser des **d'un nivell molt alt**, molt a prop del llenguatge humà, fins a un de **nivell més baix**, més proper al codi de les màquines, com ara **el codi ensamblador**.
- Exemple codi font en COBOL
- Exemple codi font en Python
- Exemple codi font en Java
- Exemple codi font ensamblador





Codi font

Els fitxers de codi font
**no contenen el llenguatge màquina que
entendrà l'ordinador.**

Per poder generar codi màquina **cal fer un
procés de traducció** des dels nemotècnics que
conté cada fitxer a les seqüències binàries que
entén el processador.

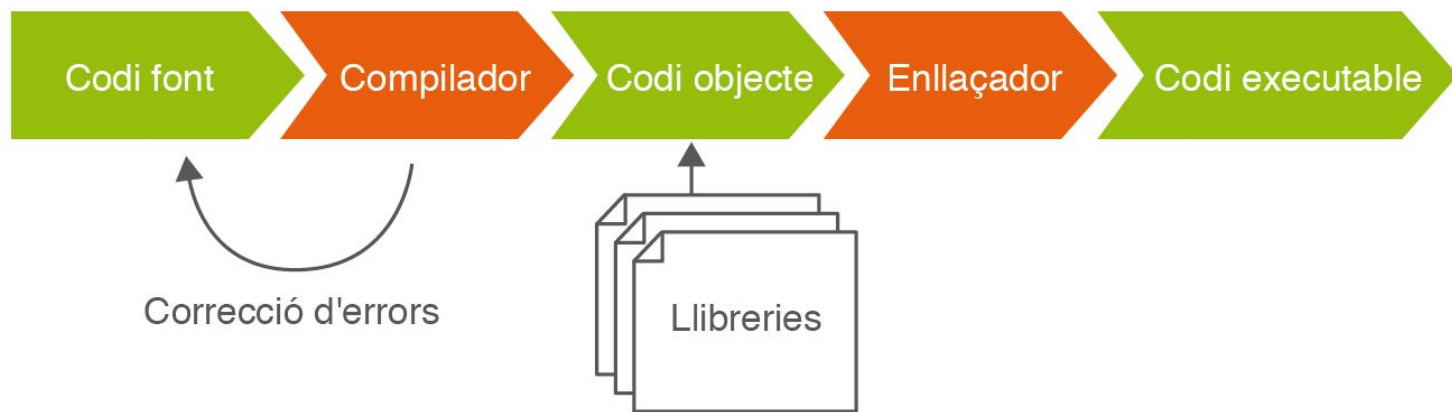
Codi executable.



- El llenguatge de màquina o codi de màquina és el llenguatge que entén el processador.
- En aquest, cadascuna de les instruccions es representa amb una seqüència binària en zeros (0) i uns (1).
- A més, tot el conjunt d'instruccions del programa queda emmagatzemat de manera consecutiva dins d'un fitxer de dades en binari.

Programes compilats

Procés de transformació d'un codi font a un codi executable:



```

    graph LR
      A[Código fuente] --> B[COMPILADOR]
      B --> C[Código máquina]
  
```

The diagram illustrates the compilation process. On the left, a box labeled 'Código fuente' (Source Code) contains a snippet of BASIC code for calculating the area of a rectangle. In the center, a large arrow labeled 'COMPILADOR' (Compiler) points from the source code to the right. On the right, a box labeled 'Código máquina' (Machine Code) contains the corresponding assembly-like instructions generated by the compiler.

El contingut d'aquests fitxers s'anomena **codi objecte**.

El programa que fa aquest procés s'anomena **compilador**.



Codi font, codi objecte i codi executable.

- El **codi objecte** és el codi font traduït (pel compilador) a codi màquina, però aquest codi encara no pot ser executat per l'ordinador.



- El **codi executable** és la traducció completa a codi màquina, dut a terme per l'**enllaçador** (en anglès, *linker*). El codi executable és interpretat directament per l'ordinador.

Codi font, codi objecte i codi executable: màquines virtuals

- L' **enllaçador** és l'encarregat d'inserir al codi objecte les funcions de les llibreries que són necessàries per al programa i de dur a terme el procés de muntatge generant un arxiu executable.
- Una **llibreria** és un col·lecció de codi predefinit que facilita la tasca del programador a l'hora de codificar un programa.



Programes interpretats

Un intérprete es un traductor que recibe como entrada un programa fuente, lo traduce y lo ejecuta. Un intérprete traduce y ejecuta una instrucción en código fuente, a la vez. Los programas interpretados generalmente son más lentos en ejecución que los programas compilados.



Un llenguatge interpretat s'executa indirectament, mitjançant l'ajut d'un programa anomenat **intèrpret**.



Màquines virtuals.

En aquest cas , la **traducció a llenguatge màquina** consta de dues fases:

1. Fase de compilació: Tradueix el codi font a un llenguatge intermedi obtenint un programa equivalent amb un menor nivell d'abstracció que l'original i que no pot ser directament executat.
2. Fase d'interpretació: Tradueix el llenguatge intermedi a un llenguatge comprensible per la màquina.

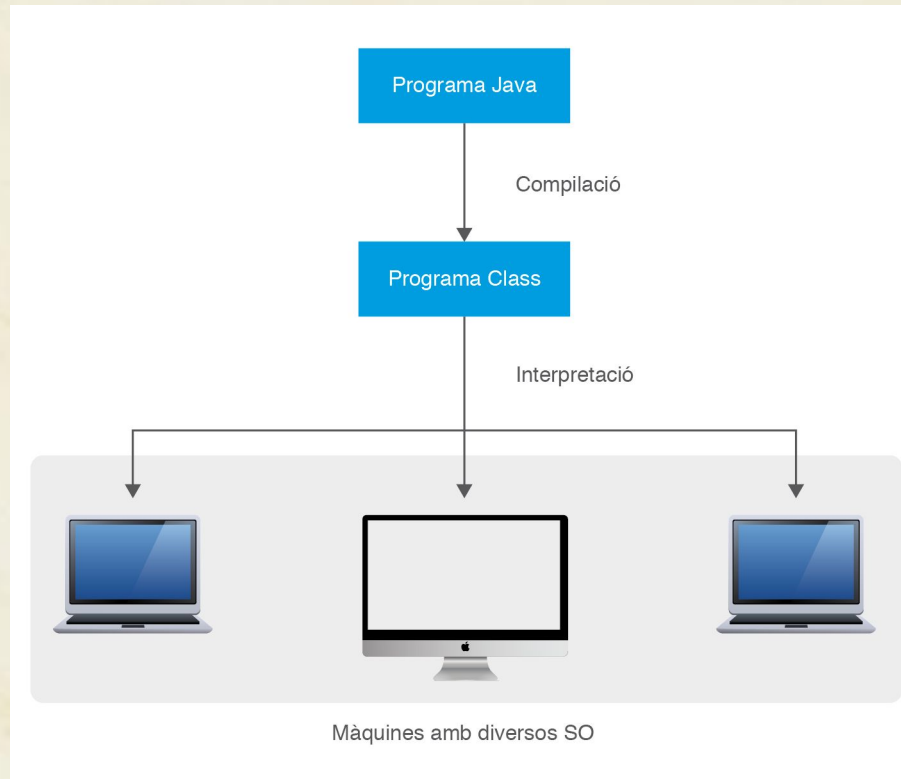
Màquines virtuals.



Per què dividir la traducció en dues fases?

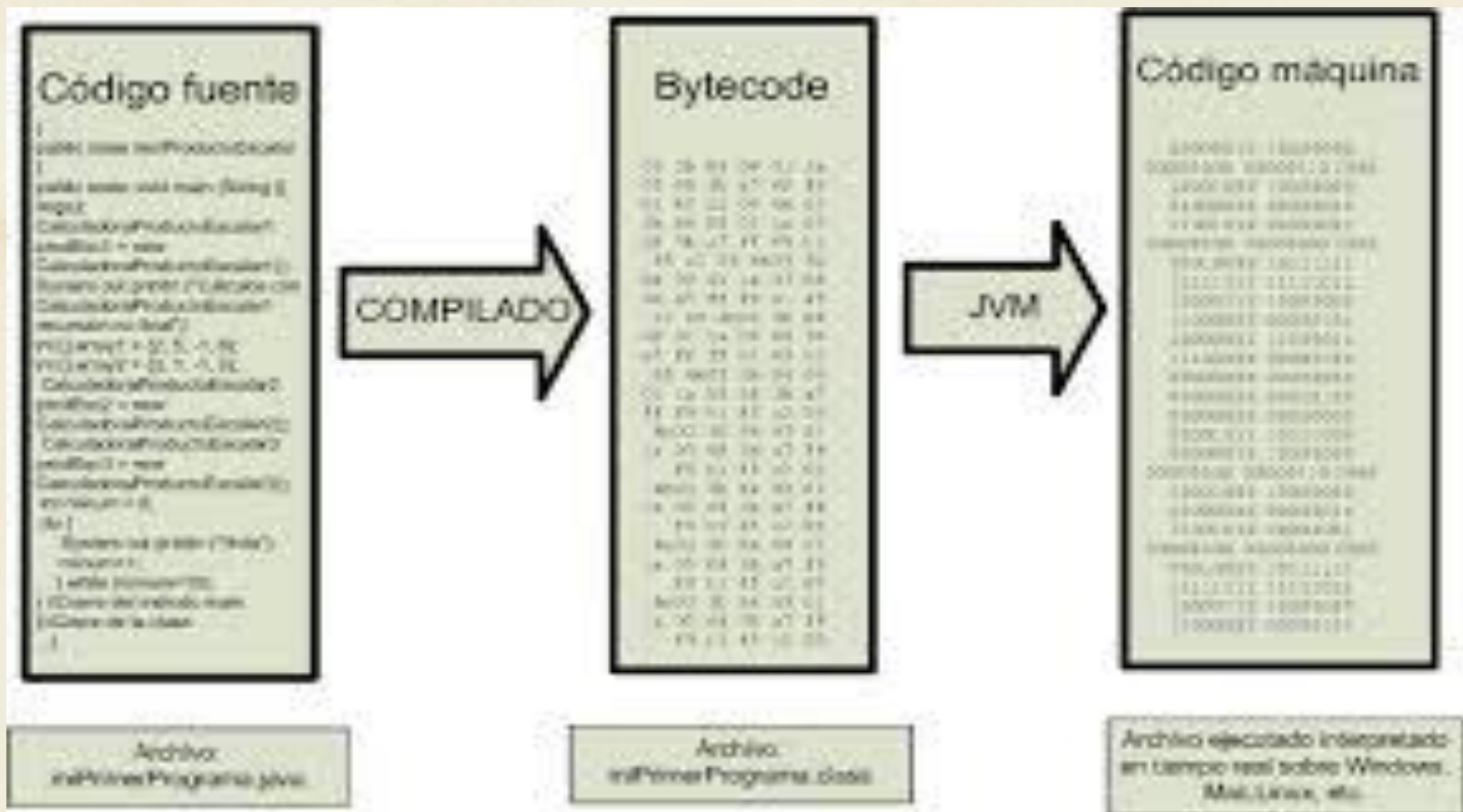
- El codi de la primera fase, el codi intermedi, serà comú per a qualsevol processador.
- El codi generat en la segona fase serà l'específic per a cada processador.

Màquines virtuals.



La màquina virtual Java

- La màquina virtual Java (JVM) és l'entorn en què s'executen els programes Java.



La màquina virtual Java (JVM)

- És un programa natiu, és a dir, executable en una plataforma específica, que és capaç d'interpretar i executar instruccions.
- Aquestes instruccions estan expressades en un codi de bytes o (el *bytecode* de **Java**) que és generat pel compilador del llenguatge Java.



La màquina virtual Java

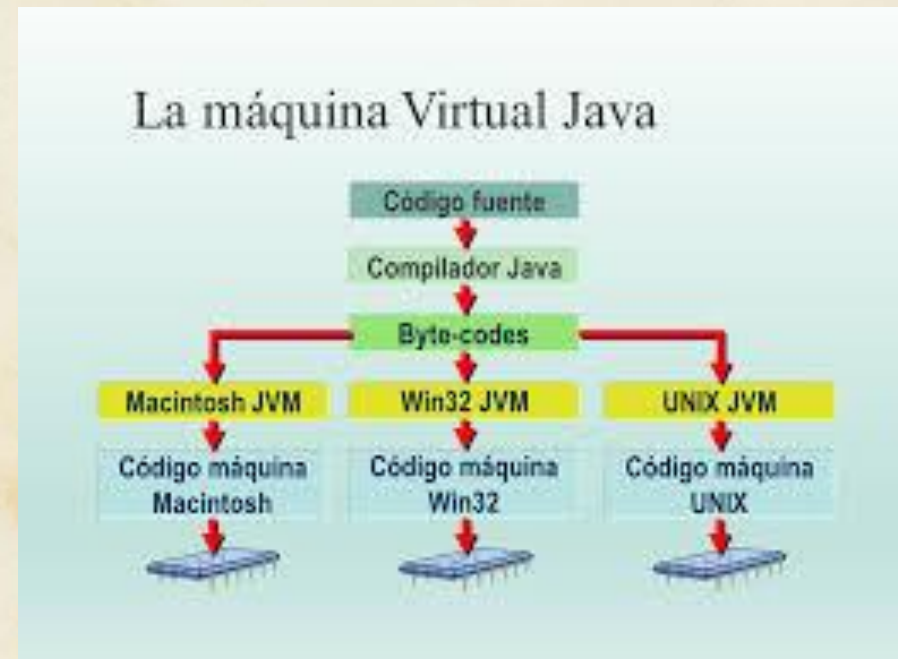


- La màquina virtual Java se situa en un nivell superior al maquinari sobre el qual es vol executar l'aplicació.
- Actua com un pont entre el codi de bytes a executar i el sistema.
- La JVM serà l'encarregada, en executar l'aplicació, de convertir el codi de bytes a codi natiu de la plataforma física.

La màquina virtual Java

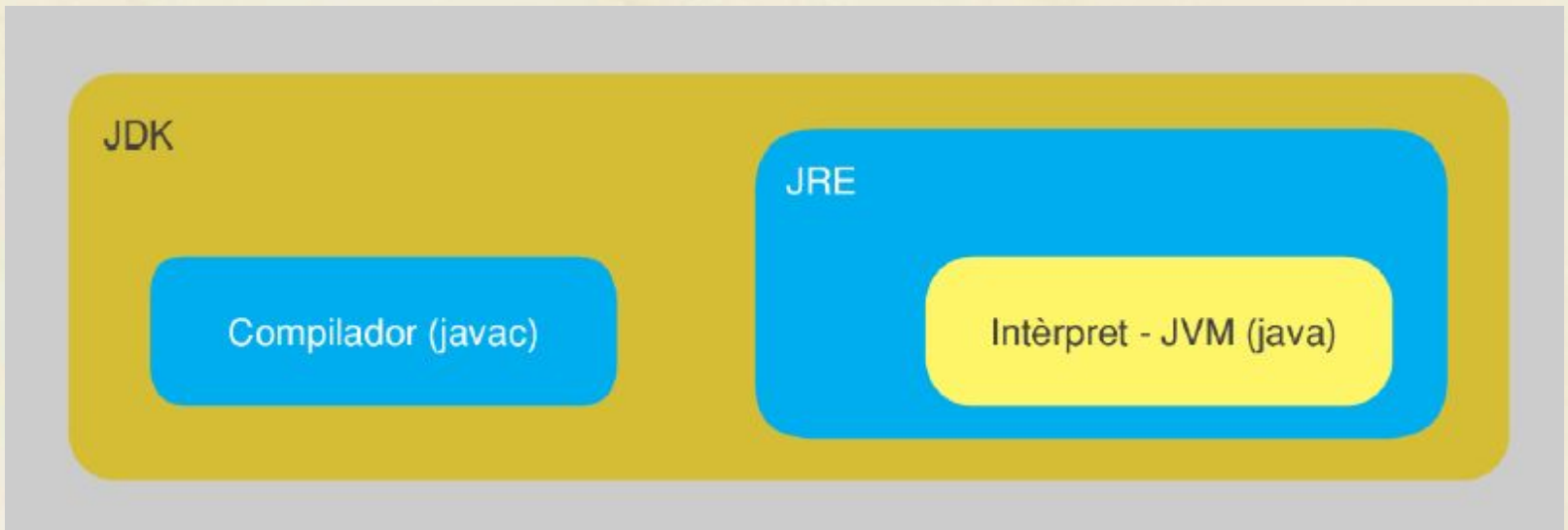
- El gran avantatge de la JVM és que possibilita la portabilitat de l'aplicació a diferents plataformes.

Un programa Java escrit en un sistema operatiu Windows es pot executar en altres sistemes operatius (Linux, Solaris i Apple OS X) amb l'únic requeriment de disposar de la JVM per al sistema corresponent.



Creació i execució de programes Java

- Un cop disposem del fitxer .class l'executarem amb l'interpret de Java, conegut popularment com a JVM (*Java Virtual Machine*).
-
- *JVM* Pertany al paquet *JRE* (*Java Runtime Environment*) present també al paquet *JDK*.





Creació i execució de programes Java

- El compilador i l'interpret de Java són dos programes que haureu d'instal·lar al vostre ordinador.
- El compilador de Java està inclòs dins de l'anomenat JDK (Java development kit), equip de desenvolupament del Java).
- El programa que posa en marxa el compilador és l'executable anomenat ***javac*** (en un sistema Windows, javac.exe).
- La instrucció "javac HolaMon.java" compila el fitxer font, donant com a resultat el fitxer en bytecode `HolaMon.class`.



Creació i execució de programes Java

- ▣ Les eines que calen són
 - ▣ Un editor de text simple.
 - ▣ Un compilador de llenguatge java.
 - ▣ Un intèrpret de Java.
- ▣ Quan s'edita codi font mitjançant un fitxer, aquest deu tenir associada una extensió.
- ▣ **.java** és l'extensió del codi font de Java.
- ▣ També se sol fer servir notació "UpperCamelCase" per anomenar els fitxers java.
 - ▣ (Ex: HolaMon.java).

Tipus de llenguatges de programació

Un llenguatge de programació és un llenguatge que permet establir una comunicació entre l'home i la màquina.



El llenguatge de programació identificarà el codi font, que el programador desenvoluparà per indicar a la màquina, una vegada aquest codi s'hagi convertit en codi executable, quins passos ha de donar.

Llenguatges de Programació



leng. Humano



ALTO

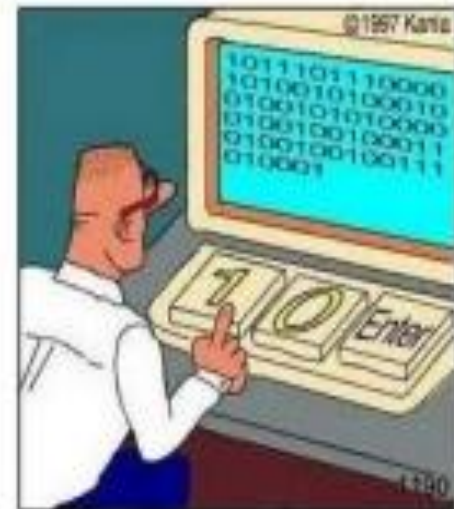


**NIVEL DEL LENGUAJE
DE PROGRAMACION**

leng. Maquina



BAJO



Generacions de llenguatges de programació





Tipus de llenguatges de programació

Els diferents tipus de llenguatges són:

1. Llenguatge de primera generació o llenguatge màquina. 1010101010101010
2. Llenguatges de segona generació o llenguatges d'assemblador.
3. Llenguatges de tercera generació o llenguatges d'alt nivell.
4. Llenguatges de quarta generació o llenguatges de propòsit específic.
5. Llenguatges de cinquena generació.

Llenguatge de primera generació o llenguatge màquina.

- És l'únic llenguatge que entén l'ordinador directament.
- Les instruccions s'expressen en codi binari.
- La programació en aquest llenguatge resulta tediosa i complicada.
- Requereix un coneixement profund de l'arquitectura física de l'ordinador.
- Fa possible que el programador utilitzi la totalitat de recursos del maquinari, amb la qual cosa es poden obtenir programes molt eficients.



Llenguatges de segona generació o llenguatges d'assemblador.


```

    .DOSSEG                ; Programa de demostración
    .MODEL SMALL
    .STACK 1024

Two EQU 2                  ; Constante

    .DATA
VarB DB ?                  ; define un Byte, cualquier valor
VarW DW 1010b              ; define un Word, en binario
VarW2 DW 257                ; define un Word, en decimal
VarD DD 0AFFFFh            ; define un DoubleWord, en hexa
S    DB "Hello I",0        ; define un String

    .CODE
main: MOV AX,DGROUP         ; resuelto por el linker
      MOV DS,AX             ; inicializa el reg. de segmento de datos
      MOV [VarB],42         ; inicializa VarB
      MOV [VarD],-7         ; setea VarD
      MOV BX,Offset[S]      ; dirección de "H" de "Hello I"
      MOV AX,[VarW]         ; poner el valor en el acumulador
```



Llenguatges de segona generació o llenguatges d'assemblador.

- Es tracta del primer llenguatge de programació que utilitza codis mnemotècnics per indicar a la màquina les operacions que ha de dur a terme.
- Aquestes operacions, molt bàsiques, han estat dissenyades a partir de la coneixença de l'estructura interna de la pròpia màquina.
- A partir del codi escrit en llenguatge d'assemblador, el programa traductor (**assemblador**) ho converteix en codi de primera generació, que serà interpretat per la màquina.

Llenguatges de segona generació o llenguatges d'assemblador.



- Es fan servir per programar controladors (drivers) o aplicacions de temps real, ja que requereix un ús molt eficient de la velocitat i de la memòria.

Llenguatges de tercera generació o llenguatges d'alt nivell.

ejemplo C: Hola Mundo!

```
#include <stdio.h>

int main()
{
    printf("Hola Mundo!\n");
    return 0;
}
```

Utilitzen paraules i frases relativament fàcils d'entendre i proporcionen també facilitats per expressar alteracions del flux de control d'una forma bastant senzilla i intuïtiva.

- S'utilitzen quan es vol desenvolupar aplicacions grans i complexes, on es prioritza comprendre com fer les coses (llenguatge humà) per sobre del rendiment del programari o del seu ús de la memòria.



Llenguatges de tercera generació o llenguatges d'alt nivell.

- Aquests llenguatges no poden ser interpretats directament per l'ordinador.
- És necessari dur a terme prèviament la seva traducció a llenguatge màquina.
- Hi ha dos tipus de traductors:
 - els compiladors
 - els intèrprets.

Exemples de llenguatges de 3a Gen

- Compilats:
 - Pascal,
 - C,
 - C++,
 - .NET,...
- Interpretats:
 - Python,
 - PHP,
 - Javascript, ...






Compiladors vs intèrprets

- L'intèrpret és notablement més lent que el compilador. Tradueix el codi font cada vegada que executem el programa.
- Per contra, els intèrprets fan que els programes siguin més portables. Els programes compilats cal tornar-los a compilar si canvien de sistema operatiu.

Llenguatges de quarta generació o llenguatges de propòsit específic




- Aporten un nivell molt alt d'abstracció en la programació, permetent desenvolupar aplicacions sofisticades en un espai curt de temps, molt inferior al necessari per als llenguatges de 3a generació.




Llenguatges de quarta generació o llenguatges de propòsit específic

- S'automatitzen certs aspectes que abans calia fer a mà.
- Inclouen eines orientades al desenvolupament d'aplicacions (IDE) que permeten definir i gestionar bases de dades, dur a terme informes (p.ex.: Oracle reports), consultes (p.ex.: informix 4GL), mòduls... , escrivint molt poques línies de codi o cap.



Llenguatges de quarta generació o llenguatges de propòsit específic


- Permeten la creació de prototipus d'una aplicació ràpidament.
- Els prototipus permeten tenir una idea de l'aspecte i del funcionament de l'aplicació abans que el codi estigui acabat. Això facilita l'obtenció d'un programa que reuneixi les necessitats i expectatives del client.



Llenguatges de quarta generació o llenguatges de propòsit específic

Avantatges

- Major abstracció.
- Menor esforç de programació.
- Menor cost de desenvolupament del programari.
- Basats en generació de codi a partir d'especificacions de nivell molt alt.
- Es poden dur a terme aplicacions sense ser un expert en el llenguatge.
- Solen tenir un conjunt d'instruccions limitat.
- Són específics del producte que els ofereix.



Llenguatges de quarta generació o llenguatges de propòsit específic

- Aquests llenguatges de programació de quarta generació estan orientats, bàsicament, a les aplicacions de negoci i al maneig de bases de dades.
- Alguns exemples de llenguatges de quarta generació són Visual Basic, Visual Basic .NET, ABAP de SAP, FileMaker, ASP, 4D...

Llenguatges de cinquena generació

Llenguatges específics per al tractament de problemes relacionats amb la intel·ligència artificial i els sistemes experts.



- L'objectiu d'aquests sistemes és “pensar” i anticipar les necessitats dels usuaris. Aquests sistemes es troben encara en desenvolupament. Es tractaria del paradigma lògic.
- Alguns exemples de llenguatges de cinquena generació són Lisp o Prolog.
- Exemples: <https://es.slideshare.net/JanselNut/prolog-ejercicios-resueltos>

Paradigmes dels llenguatges






Paradigmes de programació

Atenent a la forma de treballar dels programes i a la filosofia amb què van ser concebuts, els programes es poden classificar en:

1. Paradigma imperatiu/estructurat.
2. Paradigma d'objectes.
3. Paradigma funcional.
4. Paradigma lògic.



Paradigma imperatiu/estructurat

- La tècnica seguida en la programació imperativa és la **programació estructurada**.
- La idea és que qualsevol programa, per complex i gran que sigui, pot ser representat mitjançant tres tipus d'estructures de control:
 - Seqüència.
 - Selecció.
 - Iteració.



Paradigma imperatiu/estructurat

- **Disseny descendent (*top-down*).** És a dir, modular el programa creant porcions més petites de programes amb tasques específiques, que se subdivideixen en altres subprogrames, cada vegada més petits.
- La idea és que aquests subprogrames típicament anomenats funcions o procediments han de resoldre un únic objectiu o tasca.



El paradigma d'objectes

- Programació Orientada a Objectes (POO, o OOP en anglès).
- És un paradigma de construcció de programes basat en una abstracció del món real.
- En un POO, l'abstracció no són procediments ni funcions sinó els objectes.
- Aquests objectes són una representació directa d'alguna cosa del món real, com ara un llibre, una persona, una comanda, ...



El paradigma d'objectes

- Un objecte és **una combinació de dades** (anomenades atributs) i **mètodes** (funcions i procediments) que ens permeten interactuar amb ell.
- En aquest tipus de programació els programes són conjunts d'objectes que interactuen entre ells a través de missatges (crides a mètodes).



El paradigma d'objectes

- La programació orientada a objectes es basa en la integració de 5 conceptes:
 - abstracció,
 - encapsulació,
 - modularitat,
 - jerarquia
 - polimorfisme,



El paradigma funcional

- Està basat en un model matemàtic.
- La idea és que el resultat d'un càlcul és l'entrada del següent, i així successivament fins que una composició produeixi el resultat desitjat.
- S'utilitza principalment en àmbits d'investigació científica i aplicacions matemàtiques.



El paradigma lògic

- La característica principal és l'aplicació de les regles de la lògica per inferir conclusions a partir de dades.
- Un programa lògic conté una base de coneixement sobre la que es duen a terme consultes.
- La base de coneixement està formada per fets, que representen la informació del sistema expressada com a relacions entre les dades i regles lògiques que permeten deduir conseqüències a partir de combinacions entre els fets i, en general, altres regles.
- Un dels llenguatges més típics del paradigma lògic és el Prolog.




El paradigma lògic

- Es fa servir en les aplicacions d'Intel·ligència Artificial i en el camp de sistemes experts i processament del llenguatge humà.
- També és útil en problemes combinatoris o que requereixin una gran quantitat o amplitud de solucions alternatives, d'acord amb la naturalesa del mecanisme de tornada enrere (*backtracking*).

Característiques dels paradigmes





Característiques de la programació estructurada

- **la claredat**, el programa ha de ser entès i verificat: comentaris, noms de variables comprensibles i procediments entenedors...
- **el teorema de l'estructura**, demostra que tot programa es pot escriure utilitzant únicament les tres estructures bàsiques de control: **seqüencial**, **selecció** i **iteració**.
- **el disseny descendent**, es basa en el concepte de “divideix i venceràs” per tal de resoldre un problema.




Programació modular

- És la divisió d'un programa en parts més manejables i independents.
- En la majoria de llenguatges, els mòduls són subprogrames de tipus:
 - Procediments.
 - Funcions.

Tipus abstractes de dades (TAD)


- Permeten al programador definir un nou tipus de dades i les seves possibles operacions.






Característiques de la programació orientada a objectes

- Un **objecte** és una combinació de dades (anomenades atributs) i mètodes (funcions i procediments) que ens permeten interactuar amb ell.
- En POO, doncs, els programes són conjunts d'objectes que interactuen entre ells a través de missatges (crides a mètodes).



Característiques de la programació orientada a objectes

- **Abstracció:** es defineixen les característiques essencials d'un objecte del món real, els atributs i comportaments que el defineixen com a tal, per després modelar-lo en un objecte de programari.
- En la tecnologia orientada a objectes l'eina principal per suportar l'abstracció és la **classe**.



Característiques de la programació orientada a objectes


- **Encapsulació**, permet als objectes triar quina informació és publicada i quina serà amagada a la resta dels objectes.
- Els objectes solen presentar els seus mètodes com a interfícies públiques i els seus atributs com a dades privades o protegides, essent inaccessibles des d'altres objectes.



Encapsulació d'objectes


Les característiques que es poden atorgar són:

- **Públic:** qualsevol classe pot accedir a qualsevol atribut o mètode declarat com a públic i utilitzar-lo.
- **Protegit:** qualsevol classe heretada pot accedir a qualsevol atribut o mètode declarat com a protegit a la classe mare i utilitzar-lo.
- **Privat:** cap classe no pot accedir a un atribut o mètode declarat com a privat i utilitzar-lo.




Característiques de la programació orientada a objectes

- **Modularitat**, permet poder modificar les característiques de cada una de les classes que defineixen un objecte, de forma independent de la resta de classes en l'aplicació.
- Si una aplicació es pot dividir en mòduls separats, normalment classes, i aquests mòduls es poden compilar i modificar sense afectar els altres, aleshores aquesta aplicació ha estat implementada en un llenguatge de programació que suporta la modularitat.




Característiques de la programació orientada a objectes

- **Jerarquia**, permet l'ordenació de les abstraccions. El dos conceptes principals que es fan servir són:
 - l'herència
 - l'agregació



Característiques de la programació orientada a objectes

- **El polimorfisme**, permet donar diferents formes a un mètode, ja sigui en la definició com en la implementació.
- **La sobrecàrrega (*overload*)** de mètodes consisteix a implementar diverses vegades un mateix mètode però amb paràmetres diferents, de manera que, en invocar-lo, el compilador decideix quin dels mètodes s'ha d'executar, en funció dels paràmetres de la crida.



Característiques de la programació orientada a objectes

- **La sobreescritura (*override*)** de mètodes consisteix a reimplementar un mètode heretat d'una superclasse exactament amb la mateixa definició (incloent nom de mètode, paràmetres i valor de retorn).