1. Desenvolupament de programari

Un **programa infomàtic** és un conjunt d'esdeveniments ordenats de manera que se succeeixen de forma seqüencial en el temps, un darrere l'altre.
Un **programa informàtic** no és més que un seguit d'ordres que es porten a terme seqüencialment, aplicades sobre un conjunt de dades.
Executar un programa:
Per "executar un programa" s'entén fer que l'ordinador segueixi totes les seves ordres, des de la primera fins a la darrera.

1.2. Codi font, codi objecte i codi excutable: màquines virtuals

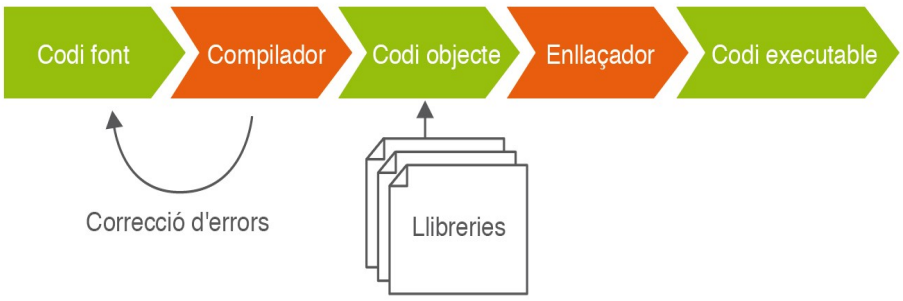
Codi Font:
Un cop s'ha acabat d'escriure el programa, el conjunt de fitxers de text resultants, on es troben les instruccions, es diu que contenen el **codi font**. Aquest codi font pot ser des d'un nivell molt alt, molt a prop del llenguatge humà, fins a un de nivell més baix, més proper al codi de les màquines, com ara el codi assemblador.

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

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

NOTA: **L' enllaçador** és l'encarregat d'inserir al codi objecte les funcions de les biblioteques que són necessàries per al programa i de dur a terme el procés de muntatge generant un arxiu executable.
Una **biblioteca** (library en anglès) és un col·lecció de codi predefinit que facilita la tasca del programador a l'hora de codificar un programa.

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

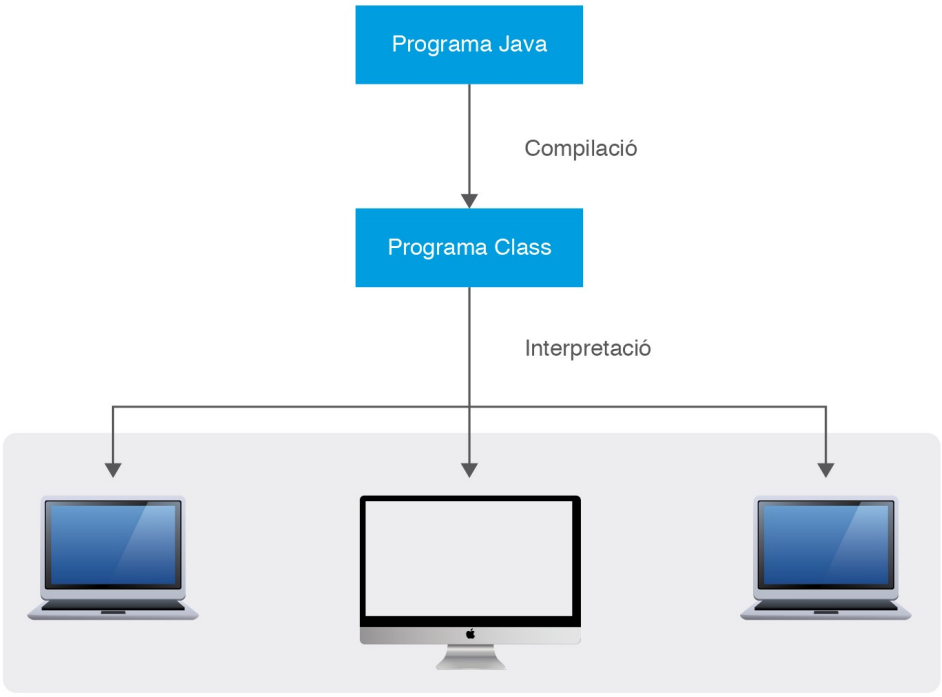


1.2.1.Màquina virtual

El concepte de màquina virtual sorgeix amb l'objectiu de facilitar el desenvolupament de compiladors que generen codi per a diferents processadors.

La **compilació** consta de dues fases:

- La primera parteix del 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.
- La segona fase tradueix el llenguatge intermedi a un llenguatge comprensible per la màquina.



Màquines amb diversos SO

La **màquina virtual** Java (JVM) és l'entorn en què s'executen els programes Java. És un programa natiu, és a dir, executable en una plataforma específica, que és capaç d'interpretar i executar instruccions expressades en un codi de bytes o (el bytecode de Java) que és generat pel compilador del llenguatge Java. 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.

NOTA: **Codi de bytes** no és un llenguatge d'alt nivell, sinó un veritable codi màquina de baix nivell, viable fins i tot com a llenguatge d'entrada per a un microprocessador físic.

1.3. 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.

Tipus:

- Llenguatge de primera generació o llenguatge màquina (binari).

- Llenguatges de **segona generació o llenguatges d'assemblador** → 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. depenen totalment del processador que utilitzi la màquina, per això es diu que estan orientats a les màquines.

Característiques dels llenguatges de primera i segona generació:
Ventajas:

- Permeten escriure programes molt optimitzats que aprofiten al màxim el maquinari (hardware) disponible.
- Permeten al programador especificar exactament quines instruccions vol que s'executin.

Desventajas:

- Els programes escrits en llenguatges de baix nivell estan completament lligats al maquinari on s'executaran i no es poden traslladar fàcilment a altres sistemes amb un maquinari diferent.
- Cal conèixer a fons l'arquitectura del sistema i del processador per escriure bons programes.
- No permeten expressar de forma directa conceptes habituals a nivell d'algorisme.
- Son difícils de codificar, documentar i mantenir.

- Llenguatges de tercera generació o llenguatges d'alt nivell. Aquests llenguatges, més evolucionats, 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. Els llenguatges de tercera generació són aquells

que són capaços de contenir i executar, en una sola instrucció, l'equivalent a diverses instruccions d'un llenguatge de segona generació.

Ventajas:

- l codi dels programes és molt més senzill i comprensible.
- Són independents del maquinari (no hi fan cap referència). Per aquest motiu és possible "portar" el programa entre diferents ordinadors / arquitectures / sistemes operatius (sempre que en el sistema de destinació existeixi un compilador per a aquest llenguatge d'alt nivell).
- És més fàcil i ràpid escriure els programes i més fàcil mantenir-los.

Inconvenientes:

- La seva execució en un ordinador pot resultar més lenta que el mateix programa escrit en llenguatge de baix nivell, tot i que això depèn molt de la qualitat del compilador que faci la traducció.

NOTA: Exemples de llenguatges de programació de tercera generació: C, C++, Java, Pascal…

Els programes escrits en llenguatges de programació de tercera generació no poden ser interpretats directament per l'ordinador, sinó que és necessari dur a terme prèviament la seva traducció a llenguatge màquina. Hi ha dos tipus de traductors: els **compiladors** i els **intèrprets.**

Compiladors:
Són programes que tradueixen el programa escrit amb un llenguatge d'alt nivell al llenguatge màquina. El compilador detectarà els possibles errors del programa font per aconseguir un programa executable depurat. (Algunos lenguajes que pasan por el compilardo: Pascal, C, C++, .NET,…).

Procediment:

- Crear el codi font.
- Crear el codi executable fent ús de compiladors i enllaçadors.
- El codi executable depèn de cada sistema operatiu. Per a cada sistema hi ha un compilador, és a dir, si es vol executar

el codi amb un altre sistema operatiu s'ha de recompilar el codi font.
    - El programa resultant s'executa directament des del sistema operatiu.

Interprete:

L'intèrpret també és un programa que tradueix el codi d'alt nivell al llenguatge màquina, però, a diferència del compilador, ho fa en temps d'execució. És a dir, no es fa un procés previ de traducció de tot el programa font a codi de bytes, sinó que es va traduint i executant instrucció per instrucció.

Caracteristiques:

- El codi interpretat no és executat directament pel sistema operatiu, sinó que fa ús d'un intèrpret.
- Cada sistema té el seu propi intèrpret.

NOTA: L'intèrpret és notablement més lent que el compilador, ja que du a terme la traducció alhora que l'execució. L'avantatge dels intèrprets és que fan que els programes siguin més portables.

- 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.

  Aspectos positivos:

  - 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.

  Nota: Alguns exemples de llenguatges de quarta generació són Visual Basic, Visual Basic .NET, ABAP de SAP, FileMaker, PHP, ASP, 4D…

- Llenguatges de **cinquena generació**. Els llenguatges de cinquena generació són llenguatges específics per al tractament de problemes relacionats amb la intel·ligència artificial i els sistemes experts.

  Nota: Alguns exemples de llenguatges de cinquena generació són Lisp o Prolog.

1.4. Paradigmes de programació

Clasificador:

- Paradigma **imperatiu/estructurat**. Deu el seu nom al paper dominant que exerceixen les sentències imperatives, és a dir aquelles que indiquen dur a terme una determinada operació que modifica les dades guardades en memòria.
  NOTA: Alguns dels llenguatges imperatius són C, Basic, Pascal, Cobol…
  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**: Instruccions executades successivament, una darrere l'altra.
  - **Selecció**: La instrucció condicional amb doble alternativa, de la forma "si condició, llavors SentènciaA, sinó SentènciaB".
  - **Iteració**: El bucle condicional "mentre condició, fes SentènciaA", que executa les instruccions repetidament mentre la condició es compleixi. A la figura.9 es pot observar un esquema que exemplifica l'estructura bàsica d'iteració.

- Paradigma **d'objectes**: El paradigma d'objectes, típicament conegut com a 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 programa orientat a objectes, 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, un empleat…

  Tècnica de 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

  NOTA: Alguns dels llenguatges de programació orientada a objectes són C++, Java, C#…

  La programació orientada a objectes es basa en la integració de 5 conceptes:

  - **Abstracció**: És el procés en el qual se separen les propietats més importants d'un objecte de les que no ho són. És a dir, per mitjà de l'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.

- **Encapsulació**: Permet als objectes triar quina informació és publicada i quina informació és amagada a la resta dels objectes. Per això 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.

    - 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.

- **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.

- **Jerarquía**: Permet l'ordenació de les abstraccions. Les dues jerarquies més importants d'un sistema complex són l'herència i l'agregació.

- **Polimorfisme**: És una característica que 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.
    - **La sobreescriptura (*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).

- 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. Son utilitzat principalment en àmbits d'investigació científica i aplicacions matemàtiques.
  NOTA: Un dels llenguatges més típics del paradigma funcional és el Lisp

- Paradigma **lògic**: té com a característica principal l'aplicació de les regles de la lògica per inferir conclusions a partir de dades.
  NOTA: Un dels llenguatges més típics del paradigma lògic és el Prolog.

1.6. Fases del desenvolupament dels sistemes d'informació

Una aplicació informàtica necessitarà moltes petites accions (i no tan petites) per ser creada. S'han desenvolupat moltes metodologies que ofereixen un acompanyament al llarg d'aquest desenvolupament, proporcionant pautes, indicacions, mètodes i documents per ajudar, sobretot, els caps de projecte més inexperts.

També és important tenir clarament identificats els rols dels components de l'equip de projecte que participaran en el desenvolupament de l'aplicació informàtica. A Mètrica aquests perfils són:

- Parts interessades (stakeholders)
- Cap de Projecte
- Consultors
- Analistes
- Programadors

1.61.1.Estudi de viabilitat del sistema

Els resultats de **l'estudi de viabilitat del sistema** constituiran la base per prendre la decisió de seguir endavant o abandonar el projecte.

1.6.2. Anàlisi del sistema d'informació

El propòsit d'aquest procés és aconseguir l'especificació detallada del sistema d'informació, per mitjà d'un catàleg de requisits i d'una sèrie de models que cobreixin les necessitats d'informació dels usuaris per als quals es desenvoluparà el sistema d'informació i que seran l'entrada per al procés de Disseny del sistema d'informació.

NOTA: Normalment, per tal d'efectuar l'anàlisi se sol elaborar els models de casos d'ús i de classes, en desenvolupaments orientats a objectes, i de dades i processos en desenvolupaments estructurats.

1.6.3. Disseny del sistema d'informació

El propòsit del disseny és obtenir la definició de l'arquitectura del sistema i de l'entorn tecnològic que li donarà suport, juntament amb l'especificació detallada dels components del sistema d'informació. A partir d'aquesta informació, es generen totes les especificacions de construcció relatives al propi sistema, així com l'especificació tècnica del pla de proves, la definició dels requisits d'implantació i el disseny dels procediments de migració i càrrega inicial.

En el **disseny** es generen les especificacions necessàries per a la construcció del sistema d'informació, com per exemple:

- Els components del sistema (mòduls o classes, segons el cas) i de les estructures de dades.
- Els procediments de migració i els seus components associats.
- La definició i revisió del pla de proves, i el disseny de les verificacions dels nivells de prova establerts.
- El catàleg d'excepcions, que permet establir un conjunt de verificacions relacionades amb el propi disseny o amb l'arquitectura del sistema.
- L'especificació dels requisits d'implantació.

1.6.4. Construcció del sistema d'informació

La construcció del sistema d'informació té com a objectiu final la construcció i la prova dels diferents components del sistema d'informació, a partir del seu conjunt d'especificacions lògiques i físiques, obtingut en la fase de disseny. Es desenvolupen els procediments d'operació i de seguretat, i s'elaboren els manuals d'usuari final i d'explotació, aquests últims quan sigui procedent.

NOTA: Per aconseguir aquest objectiu, es recull la informació elaborada durant la fase de disseny.

1.6.5. Implantació i acceptació del sistema

Aquest procés té com a objectiu principal el lliurament i l'acceptació del sistema en la seva totalitat, que pot comprendre diversos sistemes d'informació desenvolupats de manera independent, i un segon objectiu, que és dur a terme les activitats oportunes per al pas a producció del sistema.

Per a l'inici d'aquest procés es prenen com a punt de partida els components del sistema provats de forma unitària i integrats en el procés de construcció, així com la documentació associada. El sistema s'ha de sotmetre a les proves d'implantació amb la participació de l'usuari d'operació. La responsabilitat, entre altres aspectes, és comprovar el comportament del sistema sota les condicions més extremes. El sistema també serà sotmès a les proves d'acceptació, que seran dutes a terme per l'usuari final. En aquest procés s'elabora el pla de manteniment del sistema, de manera que el responsable del manteniment conegui el sistema abans que aquest passi a producció. També s'estableix l'acord de nivell de servei requerit una vegada que s'iniciï la producció. L'acord de nivell de servei fa referència a serveis de gestió d'operacions, de suport a usuaris i al nivell amb què es prestaran aquests serveis.