Desenvolupament de programari

Marcel García Vacas

Entorns de desenvolupament

Índex

Introducció **5**

Resultats d'aprenentatge **7**

1 Desenvolupament de programari **9**
1.1 Concepte de programa informàtic . . . 9
1.2 Codi font, codi objecte i codi excutable: màquines virtuals . . . 11
1.2.1 Màquina virtual . . . 12
1.3 Tipus de llenguatges de programació . . . 14
1.3.1 Característiques dels llenguatges de primera i segona generació . . . 16
1.3.2 Característiques dels llenguatges de tercera, quarta i cinquena generació . . . 18
1.4 Paradigmes de programació . . . 20
1.5 Característiques dels llenguatges més difosos . . . 24
1.5.1 Característiques de la programació estructurada . . . 25
1.5.2 Característiques de la programació orientada a objectes . . . 28
1.6 Fases del desenvolupament dels sistemes d'informació . . . 32
1.6.1 Estudi de viabilitat del sistema . . . 32
1.6.2 Anàlisi del sistema d'informació . . . 33
1.6.3 Disseny del sistema d'informació . . . 33
1.6.4 Construcció del sistema d'informació . . . 34
1.6.5 Implantació i acceptació del sistema . . . 34

2 Instal·lació i ús d'entorns de desenvolupament **37**
2.1 Funcions d'un entorn de desenvolupament . . . 37
2.1.1 Interfícies gràfiques d'usuari . . . 37
2.1.2 Editor de text . . . 38
2.1.3 Compilador . . . 38
2.1.4 Intèrpret . . . 38
2.1.5 Depurador . . . 39
2.1.6 Accés a bases de dades i gestió d'arxius . . . 39
2.1.7 Control de versions . . . 39
2.1.8 Refactorització . . . 39
2.1.9 Documentació i ajuda . . . 40
2.1.10 Exemples d'entorn integrat de desenvolupament . . . 40
2.2 Instal·lació d'un entorn de desenvolupament. Eclipse . . . 41
2.2.1 Instal·lació del 'Java Development Kit' . . . 42
2.2.2 Configurar les variable d'entorn "JAVA_HOME" i "PATH" . . . 43
2.2.3 Instal·lació del servidor web . . . 44
2.2.4 Instal·lació d'Eclipse . . . 44
2.2.5 Configuració de JDK amb l'IDE Eclipse . . . 50
2.2.6 Configuració del servidor web amb l'IDE Eclipse . . . 52
2.2.7 Instal·lació de connectors . . . 55
2.2.8 Instal·lació de components per al desenvolupament d'aplicacions GUI . . . 55

2.3 Ús bàsic d'un entorn de desenvolupament. Eclipse 57
2.3.1 Editors 58
2.3.2 Vistes 58
2.3.3 Barres d'eines 59
2.4 Edició de programes 60
2.4.1 Exemple d'utilització d'Eclipse: "projecte Calculadora" 61
2.5 Executables 72

Introducció

Cada vegada es pot trobar més i més electrònica, comunicacions i informàtica al nostre voltant, a la nostra vida quotidiana. Anem a un caixer automàtic d'una caixa o banc i caldrà interactuar amb una interfície tàctil per arribar a executar l'operativa que ens interessi. Circulem per determinades ciutats on ens indiquen en cartells lluminosos l'estat de la circulació en temps real o, fins i tot, si hi ha espais lliures per aparcar al propi carrer. I no cal parlar, si ens referim a aplicacions informàtiques, de tots els aparells electrònics que utilitzem diàriament:

- Ordinadors de sobretaula
- Ordinadors portàtils
- Telèfons mòbils
- Agendes electròniques
- Llibres electrònics
- Terminals de punt de venda
- Impressores i fotocopiadores
- Videoconsoles
- Televisors

Però aquests són només un petit exemple de totes les possibilitats que tenim avui dia d'utilitzar dispositius que porten un codi de programació incorporat, que disposen d'interfícies que permeten la interactuació amb els seus usuaris.

Abans que qualsevol d'aquests productes arribi al mercat, s'ha d'haver produït un procés, moltes vegades molt llarg, que comporta fer molts tipus de petites feines entre diverses persones. Una d'aquestes feines, part important de la creació dels productes, serà la programació, el desenvolupament del codi que es trobarà integrat en el producte i que permetrà mostrar aquestes interfícies, interactuar amb els usuaris i processar les informacions i dades vinculades.

Però per a això caldrà que prèviament aquests processos que es veuen automatitzats s'hagin ideat per algú i s'hagin desenvolupat. És en aquest punt que prenen importància els entorns de desenvolupament del programari.

Els entorns de desenvolupament són eines que ofereixen als programadors moltíssimes facilitats a l'hora de crear una aplicació informàtica. El mòdul "Entorns de Desenvolupament" mostra quin és el procés de desenvolupament d'una aplicació informàtica, tot indicant les característiques més importants de les eines que ajuden a aquest procediment.

Al llarg d'aquesta unitat formativa, "Desenvolupament de programari", es treballarà tot allò relacionat amb el desenvolupament del programari. La unitat formativa es troba dividida en dos apartats.

En l'apartat "Desenvolupament de programari" es defineixen tots aquells conceptes relacionats amb aquest àmbit, entrant en detall en cadascuna de les fases de desenvolupament de sistemes informàtics.

En l'apartat "Instal·lació i ús d'entorns de desenvolupament" s'agafa com a exemple un entorn integrat de desenvolupament, l'Eclipse, i s'explica tot allò relacionat amb la seva instal·lació i utilització. Eclipse és un entorn integrat de desenvolupament de codi obert que serveix per poder conèixer les característiques d'aquest tipus d'eines.

Resultats d'aprenentatge

En finalitzar aquesta unitat l'alumne/a:

1. Reconeix els elements i les eines que intervenen en el desenvolupament d'un programa informàtic, analitzant les seves característiques i les fases en què actuen fins arribar a la seva posada en funcionament.

- Identifica la relació dels programes amb els components del sistema informàtic: memòria, processador, perifèrics, entre d'altres.
- Identifica les fases de desenvolupament d'una aplicació informàtica.
- Diferencia els conceptes de codi font, objecte i executable.
- Reconeix les característiques de la generació de codi intermedi per a la seva execució en màquines virtuals.
- Classifica els llenguatges de programació.
- Avalua la funcionalitat oferta per les eines utilitzades en programació.

2. Avalua entorns de desenvolupament integrat analitzant les seves característiques per editar codi font i generar executable.

- Instal·la entorns de desenvolupament, propietaris i lliures.
- Afegeix i elimina mòduls en l'entorn de desenvolupament.
- Personalitza i automatitza l'entorn de desenvolupament.
- Configura el sistema d'actualització de l'entorn de desenvolupament.
- Genera executables a partir de codi font de diferents llenguatges en un mateix entorn de desenvolupament.
- Genera executables a partir d'un mateix codi font amb diversos entorns de desenvolupament.
- Identifica les característiques comunes i específiques de diversos entorns de desenvolupament.

1. Desenvolupament de programari

Tota aplicació informàtica, ja sigui utilitzada en un suport convencional (com un ordinador de sobretaula o un ordinador portàtil) o sigui utilitzada en un suport de nova generació (per exemple, dispositius mòbils com ara un telèfon mòbil de darrera generació o una tauleta tàctil PC), ha seguit un procediment planificat i desenvolupat detall per detall per a la seva creació. Aquest anirà des de la concepció de la idea o de la funcionalitat que haurà de satisfer aquesta aplicació fins a la generació d'un o diversos fitxers que permetin la seva execució exitosa.

Per convertir aquesta concepció d'una idea abstracta en un producte acabat que sigui eficaç i eficient hi haurà molts més passos, moltes tasques a fer. Aquestes tasques caldrà que estiguin ben planificades i que segueixin un guió que pot tenir en compte aspectes com:

- Analitzar les necessitats que tenen les persones que faran servir aquest programari, escoltar com el voldran, atendre a les seves indicacions...
- Dissenyar una solució que tingui en compte totes les necessitats abans analitzades: què haurà de fer el programari, quines interfícies gràfiques tindrà i com seran aquestes, quines dades s'hauran d'emmagatzemar i com es farà...
- Desenvolupar el programari que implementi tot allò analitzat i dissenyat anteriorment, fent-lo d'una forma al més modular possible per facilitar el posterior manteniment o manipulació per part d'altres programadors.
- Dur a terme les proves pertinents, tant de forma individualitzada per a cada mòdul com de forma complerta, per tal de validar que el codi desenvolupat és correcte i que fa el que ha de fer segons l'establert en els requeriments.
- Implantar el programari en l'entorn on els usuaris finals el faran servir.

Aquest apartat se centrarà en el tercer punt, el desenvolupament de programari.

1.1 Concepte de programa informàtic

Un primer pas per poder començar a analitzar com cal fer un programa informàtic és tenir clar què és un programa i què significa aquest concepte. En contrast amb altres termes usats en informàtica, és possible referir-se a un "programa" en el llenguatge col·loquial sense haver d'estar parlant necessàriament d'ordinadors. Es podria estar referint al programa d'un cicle de conferències o de cinema. Però, tot i que no es tracta d'un context informàtic, aquest ús ja aporta una idea general del seu significat.

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 altre ús habitual, ara sí vinculat al context de les màquines i els autòmats, podria ser referir-se al programa d'una rentadora o d'un robot de cuina. En aquest cas, però, el que se succeeix són un conjunt, no tant d'esdeveniments, sinó d'ordres que l'electrodomèstic segueix ordenadament. Un cop seleccionat el programa que volem, l'electrodomèstic fa totes les tasques corresponents de manera autònoma.

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!

Aquest conjunt d'ordres 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 si per arribar a assolir aquest objectiu i, sobretot, és molt important la disposició en què es duen a terme.

Entrant ja, ara sí, en el món dels ordinadors, la manera com s'estructuren les tasques que han de ser executades és similar als programes d'electrodomèstics anteriorment citats. En aquest cas, però, en lloc de transformar ingredients (o rentar roba bruta, si es tractés d'una rentadora), el que l'ordinador transforma és informació o dades.

Un **programa informàtic** no és més que un seguit d'ordres que es porten a terme seqüencialment, aplicades sobre un conjunt de dades.

Quines dades processa un programa informàtic? Bé, això dependrà del tipus de programa:

- Un editor processa les dades d'un document de text.
- Un full de càlcul processa dades numèriques ubicades en un fitxer.
- Un videojoc processa les dades que fan referència a la forma i ubicació d'enemics i jugadors, les interfícies gràfiques on es trobarà el jugador, els punts aconseguits...

- Un navegador web processa les ordres de l'usuari i les dades que rep des d'un servidor ubicat a internet.
- Un reproductor de vídeo processa els fotogrames emmagatzemats en un arxiu i l'àudio relacionat.

Per tant, la tasca d'un programador informàtic és escollir quines ordres constituiran un programa d'ordinador, en quin ordre s'han de dur a terme i sobre quines dades cal aplicar-les perquè el programa porti a terme la tasca que ha de resoldre.

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.

La dificultat de tot plegat serà més o menys gran depenent de la complexitat mateixa d'allò que cal que el programa faci. No és el mateix establir què ha de fer l'ordinador per resoldre una multiplicació de tres nombres que per processar textos o visualitzar pàgines a Internet.

D'altra banda, un cop fet el programa, cada cop que s'executi, l'ordinador complirà totes les ordres del programa.

De fet, un ordinador és incapaç de fer absolutament res per si mateix, sempre cal dir-li què ha de fer. I això se li diu mitjançant l'execució de programes. Tot i que des del punt de vista de l'usuari pot semblar que quan es posa en marxa un ordinador aquest funciona sense executar cap programa concret, cal tenir en compte que el seu sistema operatiu és un programa que està sempre en execució.

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

Editors de text simples

Un editor de text simple és aquell que permet escriure-hi només text sense format. En són exemples el Bloc de Notes (Windows), el Gedit o l'Emacs (Unix).

Per crear un programa el que es farà serà crear un arxiu i escriure a un fitxer el seguit d'instruccions que es vol que l'ordinador executi. Aquestes instruccions hauran de seguir unes pautes determinades en funció del llenguatge de programació escollit. A més, haurien de seguir un ordre determinat que donarà sentit al programa escrit. Per començar n'hi haurà prou amb un editor de text simple.

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.

En l'apartat "Tipus de llenguatges de programació" es descriuen les característiques dels llenguatges màquina, assemblador, d'alt nivell i de propòsit específic.

La tendència actual és fer ús de llenguatges d'alt nivell, és a dir, propers al llenguatge humà. Però això fa aparèixer un problema, i és que els fitxers de codi font no contenen el llenguatge màquina que entendrà l'ordinador. Per tant, resulten incomprensibles per al processador. Per poder generar codi màquina cal fer un procés de traducció des dels mnemotècnics que conté cada fitxer a les seqüències binàries que entén el processador.

El codi objecte de les instruccions té aquest aspecte:
10101001000001101100110
10100101011100001110111

El procés anomenat **compilació** és la traducció del codi font dels fitxers del programa en fitxers en format binari que contenen les instruccions en un format

que el processador pot entendre. El contingut d'aquests fitxers s'anomena **codi objecte**. El programa que fa aquest procés s'anomena **compilador**.

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

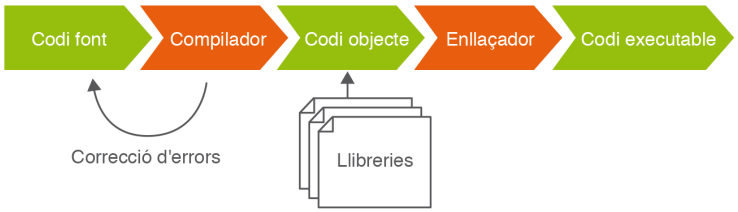
El **codi executable** és la traducció completa a codi màquina, duta a terme per l'enllaçador (en anglès, *linker*). El codi executable és interpretat directament per l'ordinador.

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.

A la figura 1.1 es mostra un resum ordenat de tots els conceptes definits. El codi font desenvolupat pels programadors es convertirà en codi objecte amb l'ajuda del compilador. Aquest ajudarà a localitzar els errors de sintaxi o de compilació que es trobin al codi font. Amb l'enllaçador, que recollirà el codi objecte i les llibreries, es generarà el codi executable.

FIGURA 1.1. Procés de transformació d'un codi font a un codi executable



1.2.1 Màquina virtual

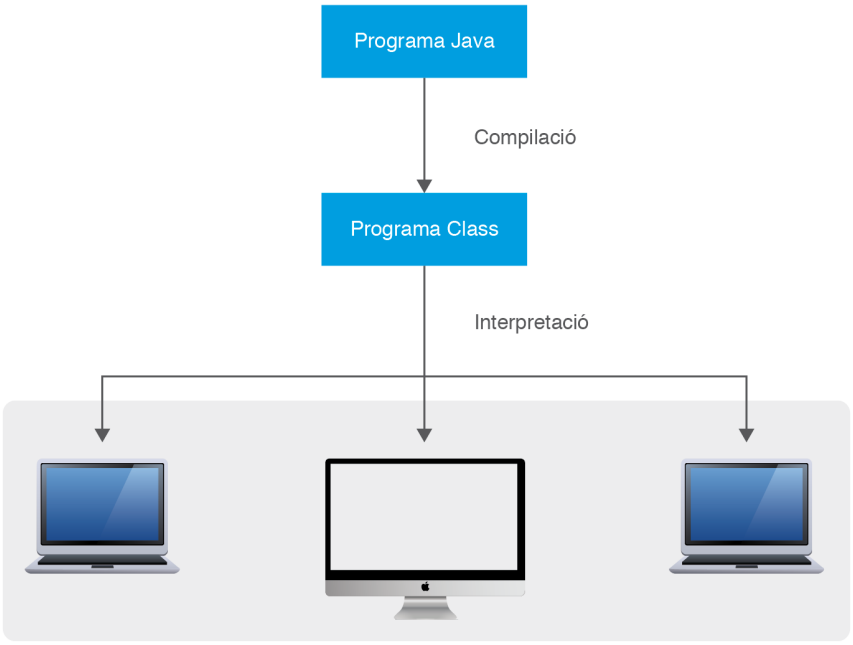
El concepte de màquina virtual sorgeix amb l'objectiu de facilitar el desenvolupament de compiladors que generen codi per a diferents processadors.

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.

Arribat aquest punt es podria plantejar la pregunta: per què dividir la compilació en dues fases? L'objectiu és que el codi de la primera fase, el codi intermedi, sigui comú per a qualsevol processador, i que el codi generat en la segona fase sigui l'específic per a cada processador. De fet, la traducció del llenguatge intermedi al llenguatge màquina no se sol fer mitjançant compilació sinó mitjançant un intèrpret, tal com es mostra en la figura 1.2.

FIGURA 1.2. Màquina virtual



Màquines amb diversos SO

La màquina virtual Java

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.

Codi de bytes

El 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.

La màquina virtual Java és una peça fonamental de la tecnologia Java. Se situa en un nivell superior al maquinari sobre el qual es vol executar l'aplicació i actua com un pont entre el codi de bytes a executar i el sistema. Així, quan un programador escriu una aplicació Java, ho fa pensant en la JVM encarregada d'executar l'aplicació i no hi ha cap motiu per pensar en les característiques de la plataforma física sobre la qual s'ha d'executar l'aplicació. La JVM serà l'encarregada, en executar l'aplicació, de convertir el codi de bytes a codi natiu de la plataforma física.

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

El concepte de màquina virtual Java s'usa en dos àmbits: d'una banda, per fer referència al conjunt d'especificacions que ha de complir qualsevol implementació de la JVM; d'altra banda, per fer referència a les diverses implementacions de la màquina virtual Java existents i de les quals cal utilitzar-ne alguna per executar les aplicacions Java.

Als materials web es pot trobar una explicació, pas a pas, de com descarregar i instal·lar la màquina virtual Java.

L'empresa Sun Microsystems és la propietària de la marca registrada Java, i aquesta s'utilitza per certificar les implementacions de la JVM que s'ajusten i són totalment compatibles amb les especificacions de la JVM, en el prefaci de les quals es diu: "Esperem que aquesta especificació documenti suficientment la màquina virtual de Java per fer possibles implementacions des de zero. Sun proporciona tests que verifiquen que les implementacions de la màquina virtual Java operen correctament."

1.3 Tipus de llenguatges de programació

Establert el concepte de programa informàtic i els conceptes de codi font, codi objecte i codi executable (així com el de màquina virtual), cal ara establir les diferències entre els diversos tipus de codi font existents, a través dels quals s'arriba a obtenir un programa informàtic.

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.

Al llarg dels darrers anys ha existit una evolució constant en els llenguatges de programació. S'ha establert una creixent evolució en la qual es van incorporant elements que permeten crear programes cada vegada més sòlids i eficients. Això facilita molt la tasca del programador per al desenvolupament del programari, el seu manteniment i l'adaptació. Avui en dia, existeixen, fins i tot, llenguatges de programació que permeten la creació d'aplicacions informàtiques a persones sense coneixements tècnics d'informàtica, pel fet d'existir una creació pràcticament automàtica del codi a partir d'unes preguntes.

Els diferents tipus de llenguatges són:

- Llenguatge de primera generació o llenguatge màquina.
- Llenguatges de segona generació o llenguatges d'assemblador.
- Llenguatges de tercera generació o llenguatges d'alt nivell.
- Llenguatges de quarta generació o llenguatges de propòsit específic.
- Llenguatges de cinquena generació.

El primer tipus de llenguatge que es va desenvolupar és l'anomenat **llenguatge de primera generació o llenguatge màquina**. És l'únic llenguatge que entén l'ordinador directament.

La seva estructura està totalment adaptada als circuits impresos dels ordinadors o processadors electrònics i molt allunyada de la forma d'expressió i anàlisi dels problemes propis dels humans (les instruccions s'expressen en codi binari). Això fa que la programació en aquest llenguatge resulti tediosa i complicada, ja que es requereix un coneixement profund de l'arquitectura física de l'ordinador. A més, s'ha de valorar que el codi màquina fa possible que el programador utilitzi la totalitat de recursos del maquinari, amb la qual cosa es poden obtenir programes molt eficients.

```
10110000 01100001
Aquesta línia conté una instrucció que mou un valor al registre del processador
    .
```

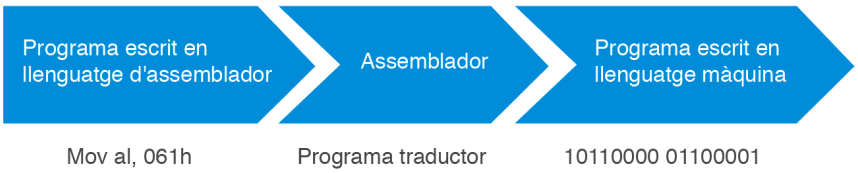
Actualment, a causa de la complexitat del desenvolupament d'aquest tipus de llenguatge, està pràcticament en desús. Només es farà servir en processadors molts concrets o per a funcionalitats molt específiques.

El segon tipus de llenguatge de programació són els **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.

Cada instrucció en llenguatge d'assemblador correspon a una instrucció en llenguatge màquina. Aquests tipus de llenguatges depenen totalment del processador que utilitzi la màquina, per això es diu que estan orientats a les màquines.

A la figura 1.3 es mostra un esquema del funcionament dels llenguatges de segona generació. 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.

Figura 1.3. Llenguatge de segona generació



En general s'utilitza aquest tipus de llenguatges per programar controladors (*drivers*) o aplicacions de temps real, ja que requereix un ús molt eficient de la velocitat i de la memòria.

1.3.1 Característiques dels llenguatges de primera i segona generació

Com a avantatges dels llenguatges de primera i segona generació es poden establir:

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

Els inconvenients són els següents:

- 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.

El següent grup de llenguatges es coneix com a **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 o d'alt nivell s'utilitzen quan es vol desenvolupar aplicacions grans i complexes, on es prioritza el fet de facilitar i comprendre com fer les coses (llenguatge humà) per sobre del rendiment del programari o del seu ús de la memòria.

Els esforços encaminats a fer la tasca de programació independent de la màquina on s'executaran van donar com a resultat l'aparició dels llenguatges de programació d'alt nivell.

Els llenguatges d'alt nivell són normalment fàcils d'aprendre perquè estan formats per elements de llenguatges naturals, com ara l'anglès. A continuació es mostra un exemple d'algorisme implementat en un llenguatge d'alt nivell, concretament en Basic. Aquest algorisme calcula el factorial d'un nombre donat a la funció com a paràmetre. Es pot observar com és fàcilment comprensible amb un mínim coneixement de l'anglès.

En Basic, el llenguatge d'alt nivell més conegut, les ordres com `IF comptador = 10 THEN STOP` poden utilitzar-se per demanar a l'ordinador que pari si la variable comptador és igual a deu.

```
' _____________________________________
' Funció Factorial
' _____________________________________
Public Function Factorial(num As Integer)As String
Dim i As Integer
    For i = 1 To num - 1
      num = num * i
      Factorial = num
    Next
End Function
```

Com a conseqüència d'aquest allunyament de la màquina i acostament a les persones, 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.

Alguns exemples de codis de programació que hauran de passar per un compilador són: Pascal, C, C++, .NET, ...

A la figura 1.4 es pot veure, en un esquema, la funció del compilador entre els dos llenguatges.

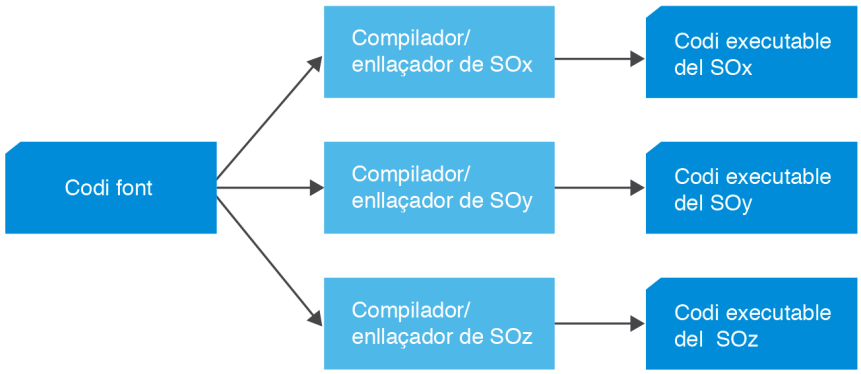
FIGURA 1.4. Codi compilat

El procediment que haurà de seguir un programador és el següent:

- 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.

A la figura 1.5 es pot observar un esquema que representa la dependència del sistema operatiu a l'hora d'escollir i utilitzar compilador.

FIGURA 1.5. Codi compilat per SO



Els intèrprets

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.

Alguns exemples de codis de programació que hauran de passar per un intèrpret són: JavaScript, PHP, ASP...

Algunes característiques dels llenguatges interpretats són:

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

Compiladors davant intèrprets

L'intèrpret és notablement més lent que el compilador, ja que du a terme la traducció alhora que l'execució. A més, aquesta traducció es fa sempre que s'executa el programa, mentre que el compilador només la du a terme una vegada. L'avantatge dels intèrprets és que fan que els programes siguin més portables. Així, un programa compilat en un ordinador amb sistema operatiu Windows no funcionarà en un Macintosh, o en un ordinador amb sistema operatiu Linux, a menys que es torni a compilar el programa font en el nou sistema.

1.3.2 Característiques dels llenguatges de tercera, quarta i cinquena generació

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.

Els avantatges dels llenguatges de tercera generació són:

- El 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.

Els inconvenients dels llenguatges de tercera generació són:

- 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.

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

Els **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.

IDE són les sigles en anglès d'Integrated Development Environment, és a dir, Entorn Integrat de Desenvolupament.

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.

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.

Alguns dels aspectes positius que mostren aquest tipus de llenguatges de programació són:

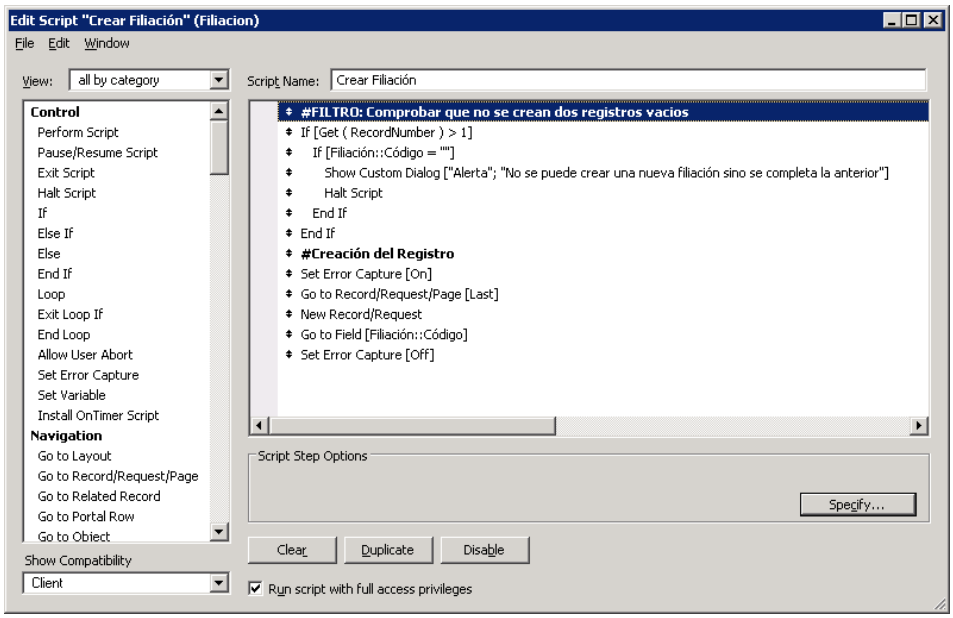
- Major abstracció.
- Menor esforç de programació.
- Menor cost de desenvolupament del programari.
- Basats en generació de codi a partir d'especificacions de nivell molt alt.
- Es poden dur a terme aplicacions sense ser un expert en el llenguatge.
- Solen tenir un conjunt d'instruccions limitat.
- Són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

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, PHP, ASP, 4D...

A la figura 1.6 es pot veure l'entorn de treball i un exemple de codi font de FileMaker.

FIGURA 1.6. FileMaker: Llenguatge de quarta generació



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

En lloc d'executar només un conjunt d'ordres, 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.

1.4 Paradigmes de programació

És difícil establir una classificació general dels llenguatges de programació, ja que existeix un gran nombre de llenguatges i, de vegades, diferents versions d'un mateix llenguatge. Això provocarà que en qualsevol classificació que es faci un mateix llenguatge pertanyi a més d'un dels grups establerts. Una classificació molt estesa, atenent a la forma de treballar dels programes i a la filosofia amb què van ser concebuts, és la següent:

- Paradigma imperatiu/estructurat.
- Paradigma d'objectes.
- Paradigma funcional.
- Paradigma lògic.

El **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.

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.
- Selecció.
- Iteració.

D'altra banda, també es proposa desenvolupar el programa amb la 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. La idea és que aquests subprogrames típicament anomenats funcions o procediments han de resoldre un únic objectiu o tasca.

Imaginem que hem de fer una aplicació que registri les dades bàsiques del personal d'una escola, dades com poden ser el nom, el DNI, i que calculi el salari dels professors així com el dels administratius, on el salari dels administratius és el sou base (`SOU_BASE`) * 10 mentre que el salari dels professors és el sou base (`SOU_BASE`) + nombre d'hores impartides (`numHores`) * 12.

```
const float SOU_BASE = 1.000;

Struct Administratiu
{
    string nom;
    string DNI;
    float Salari;
}

Struct Professor
{
    string nom;
    string DNI;
    int numHores;
    float salari;
}

void AssignarSalariAdministratiu (Administratiu administratiu1)
```

```
{
    administratiu1. salari = SOU_BASE * 10;
}

void AssignarSalariProfessor (Professor professor1)
{
    professor1. salari = SOU_BASE + (numHores * 12);
}
```

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...

Alguns dels llenguatges de programació orientada a objectes són C++, Java, C#...

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, per tant, els programes són conjunts d'objectes que interactuen entre ells a través de missatges (crides a mètodes).

La programació orientada a objectes es basa en la integració de 5 conceptes: abstracció, encapsulació, modularitat, jerarquia i polimorfisme, que és necessari comprendre i seguir de manera absolutament rigorosa. No seguir-los sistemàticament, ometre'ls puntualment per pressa o altres raons fa perdre tot el valor i els beneficis que ens aporta l'orientació a objectes.

```
class Treballador {
   private:
      string nom;
      string DNI;
   protected:
      static const float SOU_BASE = 1.000;
   public:
      string GetNom() {return this.nom;}
      void SetNom (string n) {this.nom = n;}
      string GetDNI() {return this.DNI;}
      void SetDNI (string dni) {this.DNI = dni;}
      virtual float salari() = 0;
}

class Administratiu: public Treballador {
   public:
      float Salari() {return SOU_BASE * 10};
}

class Professor: public Treballador {
   private:
      int numHores;
   public:
      float Salari() {return SOU_BASE + (numHores * 15);}
}
```

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.

Els creadors dels primers llenguatges funcionals pretenien convertir-los en llenguatges d'ús universal per al processament de dades en tot tipus d'aplicacions, però, amb el pas del temps, s'ha utilitzat principalment en àmbits d'investigació científica i aplicacions matemàtiques.

Un dels llenguatges més típics del paradigma funcional és el Lisp. Vegeu un exemple de programació del factorial amb aquest llenguatge:

```
> (defun factorial (n)
(if (= n 0)
1
(* n (factorial (- n 1)))))
FACTORIAL
> (factorial 3)
6
```

El **paradigma lògic** té com a característica principal 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.

Exemple de desplegament pràctic del paradigma lògic

Determinarem si hem de prescriure al pacient estar a casa reposant al saber que es compleixen els següents fets: malestar i 39º de temperatura corporal.

Regles de la base de coneixement:

- R1: Si febre, llavors estar a casa en repòs.
- R2: Si malestar, llavors posar-se termòmetre.
- R3: Si termòmetre marca una temperatura > 37º, llavors febre.
- R4: Si diarrea, llavors dieta.

Si seguim un raonament d'encadenament cap endavant, el procediment seria:

```
Indicar el motor d'inferència, els fets: malestar i termòmetre
    marca 39.

<code>Base de fets = { malestar, termòmetre marca º39 }
```

El sistema identifica les regles aplicables: R2 i R3. L'algorisme s'inicia aplicant la regla R2, incorporant en la base de fets "posar-se el termòmetre".

```
Base de fets = { malestar, termòmetre marca º39, posar–se termò
    metre }
```

Com que no s'ha solucionat el problema, continua amb la següent regla R3, afegint a la base de fets "febre".

```
Base de fets = { malestar, termòmetre marca º39, posar–se termò
    metre, febre }
```

Com que no s'ha solucionat el problema, torna a identificar un subconjunt de regles aplicables, excepte les ja utilitzades. El sistema identifica les regles aplicables: R1, tot incorporant a la base de fets "estar a casa en repòs".

```
Base de fets = { malestar, termòmetre marca º39, posar–se termò
    metre, febre, estar a casa en repòs}
```

Com que repòs està a la base de fets, s'ha arribat a una resposta positiva a la pregunta formulada.

El paradigma és àmpliament utilitzat en les aplicacions que tenen a veure amb la Intel·ligència Artificial, particularment en el camp de sistemes experts i processament del llenguatge humà. Un sistema expert és un programa que imita el comportament d'un expert humà. Per tant conté informació (és a dir una base de coneixements) i una eina per comprendre les preguntes i trobar la resposta correcta examinant la base de dades (un motor d'inferència).

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*).

1.5 Característiques dels llenguatges més difosos

Existeixen molts llenguatges de programació diferents, fins al punt que moltes tecnologies tenen el seu llenguatge propi. Cada un d'aquests llenguatges té un seguit de particularitats que el fan diferent de la resta.

Els llenguatges de programació més difosos són aquells que més es fan servir en cadascun dels diferents àmbits de la informàtica. En l'àmbit educatiu, per exemple, es considera un llenguatge de programació molt difós aquell que es fa servir a moltes universitats o centres educatius per a la docència de la iniciació a la programació.

Els llenguatges de programació més difosos corresponents a diferents àmbits, a diferents tecnologies o a diferents tipus de programació tenen una sèrie de característiques en comú que són les que marquen les similituds entre tots ells.

1.5.1 Característiques de la programació estructurada

La programació estructurada va ser desenvolupada pel neerlandès Edsger W. Dijkstra i es basa en el denominat teorema de l'estructura. Per això utilitza únicament tres estructures: seqüència, selecció i iteració, essent innecessari l'ús de la instrucció o instruccions de transferència incondicional (`GOTO`, `EXIT FUNCTION`, `EXIT SUB` o múltiples `RETURN`).

D'aquesta forma les característiques de la programació estructurada són la claredat, el teorema de l'estructura i el disseny descendent.

Claredat

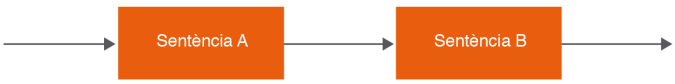
Hi haurà d'haver prou informació al codi per tal que el programa pugui ser entès i verificat: comentaris, noms de variables comprensibles i procediments entenedors... Tot programa estructurat pot ser llegit des del principi a la fi sense interrupcions en la seqüència normal de lectura.

Teorema de l'estructura

Demostra que tot programa es pot escriure utilitzant únicament les tres estructures bàsiques de control:

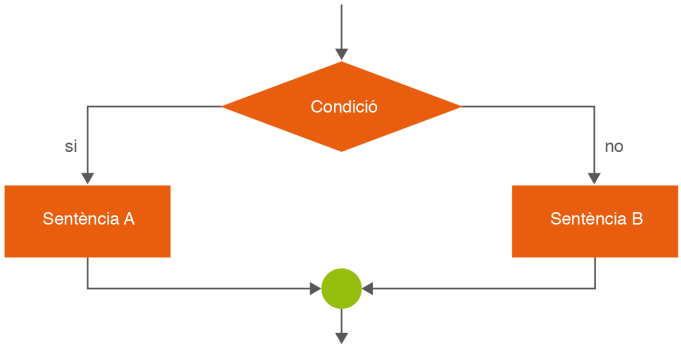
- Seqüència: instruccions executades successivament, una darrere l'altra. A la figura 1.7 es pot observar un exemple de l'estructura bàsica de seqüència, on primer s'executarà la sentència A i, posteriorment, la B.

FIGURA 1.7. Exemple de seqüència



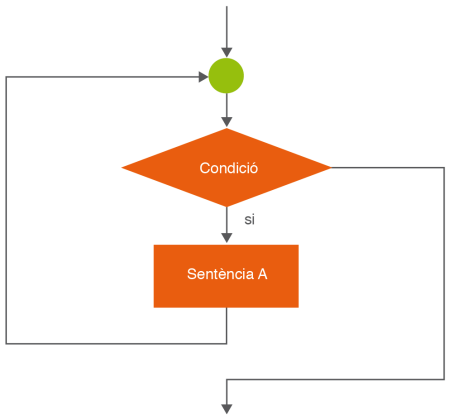
- Selecció: la instrucció condicional amb doble alternativa, de la forma "si condició, llavors SentènciaA, sinó SentènciaB". A la figura 1.8 es pot observar un esquema que exemplifica l'estructura bàsica de selecció.

FIGURA 1.8. Exemple de selecció



- Iteració: el bucle condicional "mentre condició, fes SentènciaA", que executa les instruccions repetidament mentre la condició es compleixi. A la figura 1.9 es pot observar un esquema que exemplifica l'estructura bàsica d'iteració.

FIGURA 1.9. Exemple d'iteració

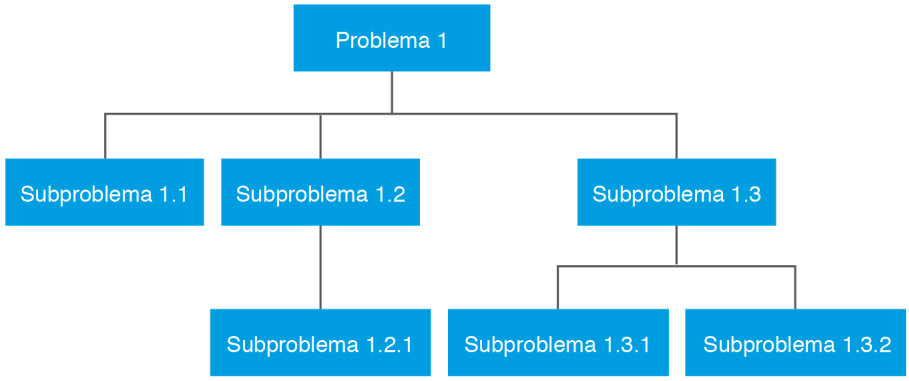


Disseny descendent

El disseny descendent és una tècnica que es basa en el concepte de "divideix i venceràs" per tal de resoldre un problema en l'àmbit de la programació. Es tracta de la resolució del problema al llarg de diferents nivells d'abstracció partint d'un nivell més abstracte i finalitzant en un nivell de detall.

A la figura 1.10 es pot observar un exemple del disseny descendent. A partir del problema1 s'obtenen diversos subproblemes (subproblema 1.1, subproblema 1.2 i subproblema 1.3). La resolució d'aquests subproblemes serà molt més senzilla que la del problema original per tal com se n'ha reduït considerablement l'abast i la mida. De forma iterativa es pot observar com aquests subproblemes es tornen a dividir, a la vegada, en altres subproblemes.

FIGURA 1.10. Disseny descendent



La visió moderna de la programació estructurada introdueix les característiques de programació modular i tipus abstractes de dades (TAD).

Programació modular

La realització d'un programa sense seguir una tècnica de programació modular produeix sovint un conjunt enorme de sentències l'execució de les quals és complexa de seguir, i d'entendre, amb la qual cosa es fa gairebé impossible la depuració d'errors i la introducció de millores. Fins i tot, es pot donar el cas d'haver d'abandonar el codi preexistent perquè resulta més fàcil començar de nou.

Quan es parla de programació modular, ens referim a la divisió d'un programa en parts més manejables i independents. Una regla pràctica per aconseguir aquest propòsit és establir que cada segment del programa no excedeixi, en longitud, d'un pam de codificació.

En la majoria de llenguatges, els mòduls es tradueixen a:

- Procediments: són subprogrames que duen a terme una tasca determinada i retornen 0 o més d'un valor. S'utilitzen per estructurar un programa i millorar la seva claredat.
- Funcions: són subprogrames que duen a terme una determinada tasca i retornen un únic resultat o valor. S'utilitzen per crear operacions noves que no ofereix el llenguatge.

Tipus abstractes de dades (TAD)

En programació, el *tipus de dades* d'una variable és el conjunt de valors que la variable pot assumir. Per exemple, una variable de tipus booleà pot adoptar només dos valors possibles: `vertader` o `fals`. A més, hi ha un conjunt limitat però ben definit d'operacions que tenen sentit sobre els valors d'un tipus de dades; així, operacions típiques sobre el tipus booleà són `AND` o `OR`.

Els llenguatges de programació assumeixen un nombre determinat de tipus de dades, que pot variar d'un llenguatge a un altre; així, en Pascal tenim els *enters*, els *reals*, els *booleans*, els *caràcters*... Aquests tipus de dades són anomenats *tipus de dades bàsics* en el context dels llenguatges de programació.

Fins fa uns anys, tota la programació es basava en aquest concepte de tipus i no eren pocs els problemes que apareixien, lligats molt especialment a la complexitat de les dades que s'havien de definir. Va aparèixer la possibilitat de poder definir *tipus abstractes de dades*, on el programador pot definir un nou tipus de dades i les seves possibles operacions.

Exemple d'implementació d'un tipus abstracte de dades implementat en el llenguatge C

```
1  struct TADpila
2  {
3          int top;
4          int elements[MAX_PILA];
5  }
6
7  void crear(struct TADpila *pila)
8  {
9          Pila.top = -1;
10 }
11
12 void apilar(struct TADpila *pila, int elem)
13 {
14         Pila.elements[pila.top++] = elem;
15 }
16
17 void desapilar(struct TADpila *pila)
18 {
19         Pila.top--;
20 }
```

1.5.2 Característiques de la programació orientada a objectes

Un dels conceptes importants introduïts per la programació estructurada és l'abstracció de funcionalitats a través de funcions i procediments. Aquesta abstracció permet a un programador utilitzar una funció o procediment coneixent només què fa, però desconeixent el detall de com ho fa.

Aquest fet, però, té diversos inconvenients:

- Les funcions i procediments comparteixen dades del programa, cosa que provoca que canvis en un d'ells afectin a la resta.
- Al moment de dissenyar una aplicació és molt difícil preveure detalladament quines funcions i procediments necessitarem.
- La reutilització del codi és difícil i acaba consistint a copiar i enganxar determinats trossos de codi, i retocar-los. Això és especialment habitual quan el codi no és modular.

L'orientació a objectes, concebut als anys setanta i vuitanta però estesa a partir dels noranta, va permetre superar aquestes limitacions.

L'**orientació a objectes** (en endavant, OO) és un paradigma de construcció de programes basat en una abstracció del món real.

En un programa orientat a objectes, l'abstracció no són els procediments ni les funcions, són els objectes. Aquests objectes són una representació directa d'alguna cosa del món real, com ara un llibre, una persona, una organització, una comanda, un empleat...

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

Els llenguatges de POO (programació orientada a objectes) són aquells que implementen més o menys fidelment el paradigma OO. La programació orientada a objectes es basa en la integració de 5 conceptes: abstracció, encapsulació, modularitat, jerarquia i polimorfisme, que és necessari comprendre i seguir de manera absolutament rigorosa. No seguir-los sistemàticament o ometre'ls puntualment, per pressa o altres raons, fa perdre tot el valor i els beneficis que aporta l'orientació a objectes.

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. En el procés d'abstracció no ha de ser preocupant la implementació de cada mètode o atribut, només cal definir-los.

En la tecnologia orientada a objectes l'eina principal per suportar l'abstracció és la **classe**. Es pot definir una classe com una descripció genèrica d'un grup d'objectes que comparteixen característiques comunes, les quals són especificades en els seus atributs i comportaments.

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. 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.

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. En altres paraules, 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.

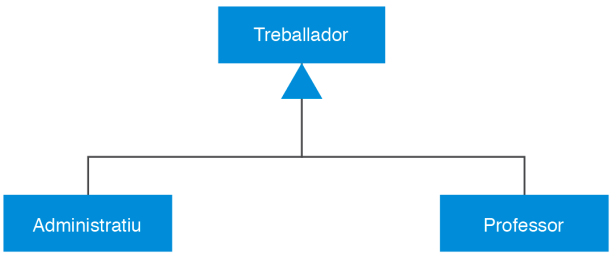
Jerarquia

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

L'herència també es pot veure com una forma de compartir codi, de manera que quan s'utilitza l'herència per definir una nova classe només s'ha d'afegir allò que sigui diferent, és a dir, reaprofita els mètodes i variables, i especialitza el comportament.

Per exemple, es pot identificar una classe *pare* anomenada `treballador` i dues classes *filles*, és a dir dos subtipus de treballadors, `administratiu` i `professor`.

FIGURA 1.11. Exemple d'herència

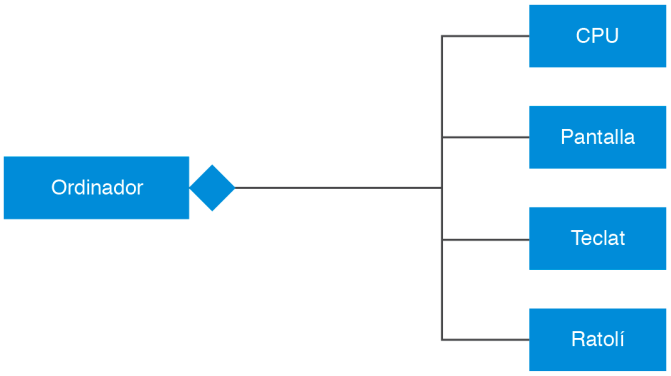


A la figura 1.11 es pot observar la representació en forma de diagrama de l'exemple explicat anteriorment: les classes `administratiu` i `professor` que hereten de la classe `treballador`.

L'agregació és un objecte que està format de la combinació d'altres objectes o components. Així, un ordinador es compon d'una CPU, una pantalla, un teclat i un ratolí, i aquests components no tenen sentit sense l'ordinador. A la figura 1.12

es pot observar un exemple d'agregació en què la classe `ordinador` està composta per les altres quatre classes.

FIGURA 1.12. Exemple d'agregació



El 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.

Un exemple de mètode sobrecarregat és aquell que calcula el salari d'un treballador en una empresa. En funció de la posició que ocupa el treballador tindrà més o menys conceptes a la seva nòmina (més o menys incentius, per exemple).

El mateix mètode, que podríem anomenar *CàlculSalari* quedarà implementat de forma diferent en funció de si es calcula el salari d'un operari (amb menys conceptes en la seva nòmina, la qual cosa provoca que el mètode rebi menys variables) o si es calcula el salari d'un directiu.

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).

Un exemple de sobrecàrrega de mètodes podria ser el del mètode `Area()`. A partir d'una classe `Figura` que conté el mètode `Area()`, existeix una classe derivada per a alguns tipus de figures (per exemple, `Rectangle` o `Quadrat`).

La implementació del mètode `Area()` serà diferent a cada una de les classes derivades; aquestes poden implementar-se de forma diferent (en funció de com es calculi en cada cas l'àrea de la figura) o definir-se de forma diferent.

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.

Dintre d'aquestes metodologies hi ha Mètrica v3.0. Ha estat desenvolupada pel Ministeri d'Administracions Públiques. Es tracta d'una metodologia per a la planificació, desenvolupament i manteniment dels sistemes d'informació d'una organització. Per al desenvolupament de programari cal fixar-se en la part que fa referència al desenvolupament dels sistemes d'informació (SI), dins la metodologia Mètrica. Divideix el desenvolupament en 5 fases, que se segueixen de forma seqüencial.

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

A la figura 1.13 hi podeu observar les cinc fases principals de la metodologia Mètrica v3.0.

FIGURA 1.13. Fases de desenvolupament d'una aplicació

1.6.1 Estudi de viabilitat del sistema

El propòsit d'aquest procés és analitzar un conjunt concret de necessitats, amb la idea de proposar una solució a curt termini. Els criteris amb què es fa aquesta proposta no seran estratègics sinó tàctics i relacionats amb aspectes econòmics, tècnics, legals i operatius.

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.

En primer lloc, es descriu el sistema d'informació, a partir de la informació obtinguda en l'estudi de viabilitat. Es delimita el seu abast, es genera un catàleg de requisits generals i es descriu el sistema mitjançant uns models inicials d'alt nivell.

Es recullen de forma detallada els requisits funcionals que el sistema d'informació ha de cobrir. A més, s'identifiquen els requisits no funcionals del sistema, és a dir, les facilitats que ha de proporcionar el sistema, i les restriccions a què estarà sotmès, quant a rendiment, freqüència de tractament, seguretat...

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. D'altra banda, s'aconsella dur a terme una definició d'interfícies d'usuari, ja que facilitarà la comunicació amb els usuaris clau.

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.

Per aconseguir aquest objectiu, es recull la informació elaborada durant la fase de disseny, es prepara l'entorn de construcció, es genera el codi de cada un dels components del sistema d'informació i es van duent a terme, a mesura que es vagi finalitzant la construcció, les proves unitàries de cada un d'ells i les d'integració entre subsistemes. Si fos necessari efectuar una migració de dades, és en aquest procés on es porta a terme la construcció dels components de migració i dels procediments de migració i càrrega inicial de dades.

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.

Un cop revisada l'estratègia d'implantació, s'estableix el pla d'implantació i es detalla l'equip que el portarà a terme.

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.

2. Instal·lació i ús d'entorns de desenvolupament

Per poder elaborar una aplicació és necessari utilitzar una sèrie d'eines que ens permetin escriure-la, depurar-la, traduir-la i executar-la. Aquest conjunt d'eines es coneix com a entorn de desenvolupament integrat i la seva funció és proporcionar un marc de treball per al llenguatge de programació.

La selecció i una utilització òptima dels entorns de desenvolupament serà una decisió molt important en el procediment de creació de programari. L'entorn de desenvolupament és l'eina amb la qual el programador haurà de treballar durant la major part de temps que dediqui a la creació de noves aplicacions.

Si l'entorn de desenvolupament és el més adient per a un determinat llenguatge de programació i per al desenvolupament d'una aplicació determinada, i si el programador que la faci servir és coneixedor de la majoria de les funcionalitats i sap aprofitar totes les facilitats que ofereixi l'entorn, es podrà optimitzar el temps de desenvolupament de programari i facilitar l'obtenció d'un producte de qualitat.

2.1 Funcions d'un entorn de desenvolupament

IDE vol dir, en anglès, Integrated Development Environment.

Un **entorn de desenvolupament integrat**, o IDE, és un programa compost per una sèrie d'eines que utilitzen els programadors per desenvolupar codi. Aquest programa pot estar pensat per a la seva utilització amb un únic llenguatge de programació o bé pot donar cabuda a diversos.

GUI, en anglès, vol dir Graphical User Interface.

Les eines que normalment componen un entorn de desenvolupament integrat són un sistema d'ajuda per a la construcció d'interfícies gràfiques d'usuari (GUI), un editor de text, un compilador/intèrpret i un depurador.

2.1.1 Interfícies gràfiques d'usuari

Les interfícies gràfiques d'usuari són un conjunt de funcionalitats que permeten incorporar, editar i eliminar components gràfics de forma senzilla en l'aplicació que s'està desenvolupant. Aquests components facilitaran la interacció de l'usuari amb l'ordinador.

2.1.2 Editor de text

Un editor de text és un programa que permet crear i modificar arxius digitals compostos únicament per text sense format, coneguts comunament com arxius de text o text pla.

L'editor de text és l'eina més utilitzada perquè ofereix la possibilitat de crear i modificar els continguts desenvolupats, el codi de programació que farà funcionar adequadament l'aplicació.

Sempre que l'IDE reconegui el llenguatge de programació (o disposi del component adequat) l'editor oferirà:

- Ressaltat de sintaxi (*syntax highlighting*): les paraules clau seran reconegudes amb colors, cosa que facilitarà molt la feina del programador.
- Compleció de codi (*code completion*): es reconeixerà el codi que s'està escrivint i, per exemple en un objecte o classe, oferirà els seus atributs, propietats o mètodes per tal que el programador seleccioni quin vol referenciar.
- Corrector d'errors, normalment des d'un punt de vista de sintaxi.
- Regions plegables: ocultació de certes parts del codi, per tal de facilitar la visualització d'aquells fragments més rellevants.

2.1.3 Compilador

En funció del llenguatge de programació utilitzat, l'IDE podrà oferir la funcionalitat de compilar-lo. Un compilador tradueix un codi de programació en un llenguatge màquina capaç de ser interpretat pels processadors i de ser executat. A l'hora de compilar un codi de programació, els entorns integrats de desenvolupament disposaran de diferents fases d'anàlisi del codi, com són la fase semàntica i la fase lexicogràfica. La compilació mostrarà els errors trobats o generarà codi executable, en el cas de trobar-ne cap.

2.1.4 Intèrpret

L'intèrpret tradueix el codi d'alt nivell a codi de bytes, però, a diferència del compilador, ho fa en temps d'execució.

2.1.5 Depurador

El depurador és un programa que permet provar i depurar el codi font d'un programa, facilitant la detecció d'errors.

Algunes de les funcionalitats típiques dels depuradors són:

- Permetre l'execució línia a línia del codi validant els valors que van adquirint les variables.
- Pausar el programa en una determinada línia de codi, fent ús d'un o diversos punts de ruptura (*breakpoints*).
- Alguns depuradors ofereixen la possibilitat de poder modificar el contingut d'alguna variable mentre s'està executant.

Opcionalment, poden presentar un sistema d'accés a bases de dades i gestió d'arxius, un sistema de control de versions, un sistema de proves, un sistema de refactorització i un generador automàtic de documentació.

2.1.6 Accés a bases de dades i gestió d'arxius

Els llenguatges de quarta generació ofereixen la possibilitat d'integrar codi d'accés a bases de dades (o codi de sentències estructurades). Per facilitar aquesta funcionalitat és molt recomanable disposar de la possibilitat d'accedir directament a la base de dades des del mateix entorn de desenvolupament i no haver-ne d'utilitzar un altre. El mateix succeirà amb la gestió d'arxius.

2.1.7 Control de versions

A mesura que un programador va desenvolupant noves línies de codi, és important tenir un històric de la feina feta o de les versions que s'han donat per bones o s'han modificat. El control de versions permet tornar a una versió anterior que sigui estable o que compleixi unes determinades característiques que els canvis fets no compleixen.

2.1.8 Refactorització

La refactorització (*refactoring*) és una tècnica de l'enginyeria de programari dirigida a reestructurar un codi font, alterant la seva estructura interna sense canviar el seu comportament extern.

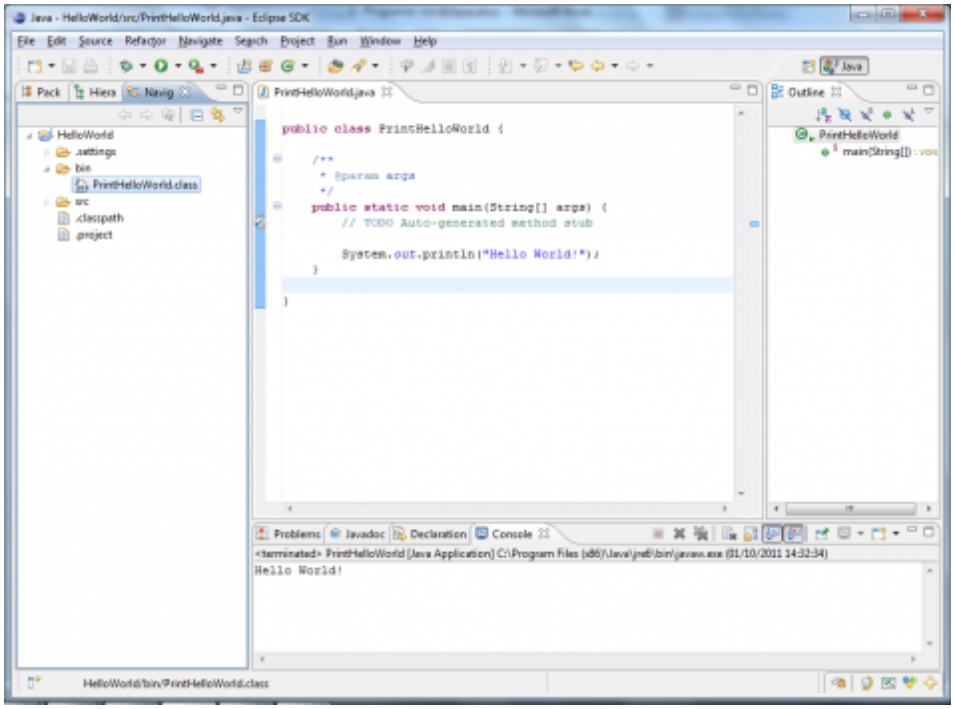
2.1.9 Documentació i ajuda

La documentació i ajuda proveeix accés a documentació, manuals i ajuda contextual sobre les instruccions i la sintaxi dels llenguatges suportats.

2.1.10 Exemples d'entorn integrat de desenvolupament

A la figura 2.1 es mostra un exemple d'un entorn integrat de desenvolupament. Concretament, es tracta de l'eina Eclipse.

FIGURA 2.1. Exemple d'entorn integrat de desenvolupament: Eclipse



Avui dia els entorns de desenvolupament proporcionen un marc de treball per a la majoria dels llenguatges de programació existents en el mercat (per exemple C, C++, C#, Java o Visual Basic, entre d'altres). A més, és possible que un mateix entorn de desenvolupament permeti utilitzar diversos llenguatges de programació, com és el cas d'Eclipse (al que es pot afegir suport de llenguatges addicionals mitjançant connectors *-plugins-*) o Visual Studio (que està pensat per treballar amb els llenguatges VB.Net, C#, C++...).

Com a exemples d'IDE multiplataforma es poden trobar, entre molts d'altres:

- Eclipse, projecte multiplataforma (Windows, Linux, Mac) de codi obert, fundat per IBM el novembre de 2001, desenvolupat en Java.
- Netbeans, projecte multiplataforma (Windows, Linux, Mac, Solaris) de codi obert, fundat per Sun Microsystems el juny de 2000, desenvolupat en Java.

- Anjuta DevStudio, per al GNU/Linux, creat per Naba Kumar el 1999.
- JBuilder, eina multiplataforma (Windows, Linux, Mac), propietat de l'empresa Borland, apareguda el 1995. La versió 2008 incorpora tres edicions (Enterprise –de pagament–, Professional –de pagament– i Turbo –gratuïta-).
- JDeveloper, eina multiplataforma (Windows, Linux, Mac) gratuïta, propietat de l'empresa Oracle, apareguda el 1998, inicialment basada en JBuilder però desenvolupada des de 2001 en Java.

A part dels que hem esmentat, existeixen molts altres IDE, molt coneguts, per a determinades plataformes, com per exemple:

- Visual Studio
- Dev-Pascal
- Dev-C++
- MonoDevelop

2.2 Instal·lació d'un entorn de desenvolupament. Eclipse

Cada programari i cada entorn de desenvolupament té unes característiques i unes funcionalitats específiques. Això quedarà reflectit també en la instal·lació i en la configuració del programari. En funció de la plataforma, entorn o sistema operatiu en què es vulgui instal·lar el programari, es farà servir un paquet d'instal·lació o un altre, i s'hauran de tenir en compte unes opcions o unes altres en la seva configuració.

Tot seguit es mostra com es du a terme la instal·lació d'una eina integrada de desenvolupament de programari, com és l'Eclipse. Però també podreu observar els procediments per instal·lar altres eines necessàries o recomanades per treballar amb el llenguatge de programació JAVA, com l'Apache Tomcat o la Màquina Virtual de Java.

Podeu trobar més documentació referent a l'entorn Eclipse als "Annexos" i "Activitats" dels materials Web d'aquest apartat.

Cal que tingueu presents els següents conceptes:

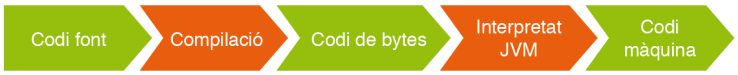
Per ampliar el concepte de Màquina Virtual de Java podeu consultar l'apartat "Codi font, codi objecte i codi executable: màquines virtuals" de la unitat "Desenvolupament de programari".

- **JVM** (*Java Virtual Machine*, màquina virtual de Java) s'encarrega d'interpretar el codi de bytes i generar el codi màquina de l'ordinador (o dispositiu) en el qual s'executa l'aplicació. Això significa que ens cal una JVM diferent per a cada entorn.
- **JRE** (*Java Runtime Environment*) és un conjunt d'utilitats de Java que inclou la JVM, llibreries i el conjunt de programari necessari per executar les aplicacions client de Java, així com el connector per tal que els navegadors d'internet puguin executar les *applets*.

- **JDK** (*Java Development Kit*, *kit* de desenvolupament de Java) és el conjunt d'eines per a desenvolupadors; conté, entre altres coses, el JRE i el conjunt d'eines necessàries per compilar el codi, empaquetar-lo, generar documentació...

A la figura 2.2 es pot observar de forma esquemàtica el procés de conversió del codi Java, des de la creació del seu codi font fins a l'obtenció del codi màquina.

FIGURA 2.2. Procés de conversió del codi



El procés d'instal·lació consisteix en els següents passos:

1. Descarregar, instal·lar i configurar el JDK.
2. Descarregar i instal·lar un servidor web o d'aplicacions.
3. Descarregar, instal·lar i configurar Eclipse.
4. Configurar JDK amb l'IDE d'Eclipse.
5. Configurar el servidor Apache Tomcat en Eclipse.
6. En cas de ser necessari, instal·lació de connectors.
7. En cas de ser necessari, instal·lació de nou programari, com per exemple WindowBuilder.

2.2.1 Instal·lació del 'Java Development Kit'

Per poder executar Eclipse, cal tenir instal·lat el JDK prèviament a l'ordinador. El podeu descarregar a la pàgina bit.ly/1iOZIrD.

Descarregar i instal·lar el JDK

Podem diferenciar entre:

- Java SE (Java Standard Edition): és la versió estàndard de la plataforma, essent aquesta plataforma la base per a tot entorn de desenvolupament en Java pel que fa a aplicacions client, d'escriptori o web.
- Java EE (Java Enterprise Edition): és la versió més gran de Java i s'utilitza en general per crear aplicacions grans de client/servidor i per a desenvolupament de WebServices.

En aquest curs s'utilitzaran les funcionalitats de Java SE.

El fitxer és diferent en funció del sistema operatiu on s'hàgi d'instal·lar. Així:

- Per als sistemes operatius Windows i Mac OS hi ha un fitxer instal·lable.
- Per als sistemes operatius GNU Linux que admeten paquets *.rpm* o *.deb* també hi ha disponibles paquets d'aquests tipus.
- Per a la resta de sistemes operatius GNU Linux hi ha un fitxer comprimit (acabat en *.tar.gz*).

En els dos primers casos, només cal seguir el procediment d'instal·lació habitual al sistema operatiu amb el qual es treballa.

Al darrer cas, però, cal descomprimir el fitxer i copiar-lo a la carpeta on el volem instal·lar. Normalment, tots els usuaris tindran permís de lectura i execució sobre aquesta carpeta.

A partir de la versió 11 del JDK Oracle distribueix el programari amb una **llicència** significativament més restrictiva que la de les versions anteriors. En concret, només pot utilitzar-se per a "desenvolupar, provar, fer prototipus i demostrar les vostres aplicacions". S'exclou explícitament tot ús "per a finalitats comercials, de producció o de negoci interns" diferent de l'esmentat abans.
En cas de necessitar-lo per algun d'aquests usos no permesos a la nova llicència, a més de les versions anteriors del JDK, existeixen versions de referència d'aquestes versions amb llicència "GNU General Public License version 2, with the Classpath Exception", que permet la majoria dels usos habituals. Aquestes versions estan enllaçades a la mateixa pàgina de descàrregues i, també, en l'adreça jdk.java.net.

2.2.2 Configurar les variable d'entorn "JAVA_HOME" i "PATH"

Consulteu a l'annex "Configuració de Java i exemple inicial" per veure pas a pas la configuració de la variable d'entorn PATH.

Una vegada descarregat i instal·lat el JDK, cal configurar algunes variables d'entorn:

- La variable `JAVA_HOME`: indica la carpeta on s'ha instal·lat el JDK. No és obligatori definir-la, però és molt convenient fer-ho, ja que molts programes cerquen en ella la ubicació del JDK. A més, facilita definir les dues varibles següents.
- La variable `PATH`. Ha d'apuntar al directori que conté l'executable de la màquina virtual. Sol ser la subcarpeta *bin* del directori on hem instal·lat el JDK.

Variable CLASSPATH

Una altra variable que té en compte el JDK és la variable `CLASSPATH`. Apunta a les carpetes on són les biblioteques pròpies de l'aplicació que es vol executar amb l'ordre *java*. És preferible, però, indicar la ubicació d'aquestes carpetes amb l'opció *-cp* de la mateixa ordre *java*, ja que cada aplicació pot tenir biblioteques diferents i les variables d'entorn afecten a tot el sistema.

Configurar la variable `PATH` és imprescindible perquè el sistema operatiu trobi les ordres del JDK i pugui executar-les.

2.2.3 Instal·lació del servidor web

Un servidor web és un programa que serveix per atendre i respondre a les diferents peticions dels navegadors, proporcionant els recursos que sol·licitin per mitjà del protocol HTTP o el protocol HTTPS (la versió xifrada i autenticada).

Podeu consultar l'annex "Descàrrega i instal·lació de l'Apache Tomcat" en la secció "Annexos".

Bàsicament, un servidor web consta d'un intèrpret HTTP que es manté a l'espera de peticions de clients i respon amb el contingut sol·licitat. El client, un cop rebut el codi, l'interpreta i el mostra en el navegador.

De fet, un servidor web sol executar de forma infinita el següent bucle:

- Espera peticions al port TCP (Protocol de Control de Transmissió) indicat (l'estàndard per defecte per a HTTP és el 80).
- Rep una petició.
- Cerca el recurs.
- Envia el recurs utilitzant la mateixa connexió mitjançant la qual va rebre petició.

El servidor web en què es basen els exemples d'aquesta unitat és Apache Tomcat.

2.2.4 Instal·lació d'Eclipse

Eclipse és una aplicació de codi obert desenvolupada actualment per la Fundació Eclipse, una organització independent, sense ànim de lucre, que fomenta una comunitat de codi obert i la utilització d'un conjunt de productes, serveis, capacitats i complements per a la divulgació de l'ús de codi obert en el desenvolupament d'aplicacions informàtiques. Eclipse va ser desenvolupat originalment per IBM com el successor de VisualAge.

Aquesta plataforma de programari és independent d'altres plataformes, entorns o sistemes operatius. Aquesta plataforma és utilitzada per desenvolupar entorns integrats de desenvolupament, com l'IDE de Java (Java Development Toolkit JDT).

Com que Eclipse està desenvolupat en Java, és necessari, per a la seva execució, que hi hagi un JRE (Java Runtime Environment) instal·lat prèviament en el sistema. Per saber si es disposa d'aquest JRE instal·lat, es pot fer la prova a la web oficial de Java, a l'apartat ¿Tengo Java?: bit.ly/2P2YOv1. A la figura 2.3 es pot veure un exemple d'aquesta prova. En el cas que no es tingui instal·lat, allà mateix se'n podrà descarregar la darrera versió i instal·lar-la sense cap dificultat.

FIGURA 2.3. Validació de la correcta instal·lació del JRE a l'ordinador

Si desenvoluparem amb Java, com és el nostre cas, cal tenir instal·lat el JDK (recordeu que és un superconjunt del JRE).

Les versions actuals de l'entorn Eclipse s'instal·len amb un instal·lador. Aquest, bàsicament s'encarrega de descomprimir, resoldre algunes dependències i crear els accessos directes.

Aquest instal·lador es pot obtenir baixant-lo directament del lloc web oficial del Projecte Eclipse www.eclipse.org. Podeu trobar les versions per als diferents sistemes operatius en aquest enllaç: bit.ly/2Ppj1OP. En aquesta pàgina, a més, trobareu les instruccions per utilitzar-lo. No són gens complexes.

Al cas de GNU Linux i MAC OS, l'arxiu és un fitxer comprimit. Caldrà, doncs, descomprimir-lo i, a continuació executar l'instal·lador. Aquest és al fitxer *eclipse-inst*, dins la carpeta *eclipse*, que és una subcarpeta del resultat de descomprimir el fitxer anterior.

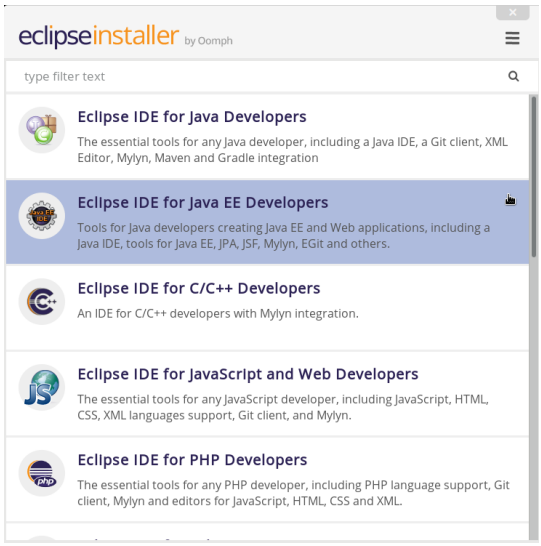
Si només l'usuari actual utilitzarà l'IDE, pot realitzar-se la instal·lació sense utilitzar privilegis d'administrador o *root* i seleccionar, per la instal·lació, una carpeta pròpia d'aquest usuari. Si es dessitgés compartir la instal·lació entre diferents usuaris, caldria indicar a l'instal·lador una carpeta sobre la qual tots aquests usuaris tinguessin permís de lectura i execució.

Un cop executem l'instal·lador, ens apareixerà una pantalla semblant a la de la figura 2.4.

Figura 2.4. Pantalla inicial de l'instal·lador d'Eclipse

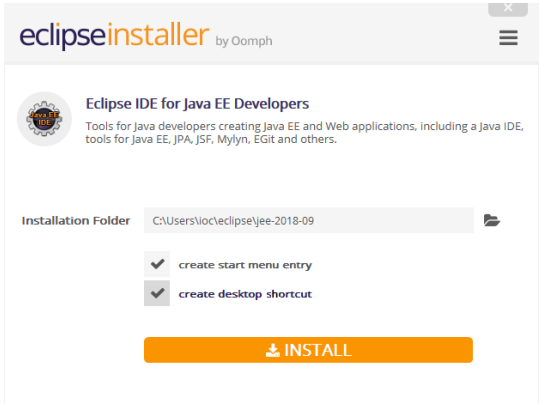
Com es veu a la figura 2.5, l'instal·lador demanarà quina versió volem instal·lar. La versió que utilitzarem és "Eclipse IDE for Java EE Developers".

Figura 2.5. Selecció de la versió d'Eclipse



A continuació, com es veu a la figura figura 2.6, demanarà la carpeta on farem la instal·lació.

Figura 2.6. Selecció de la carpeta on fer la instal·lació

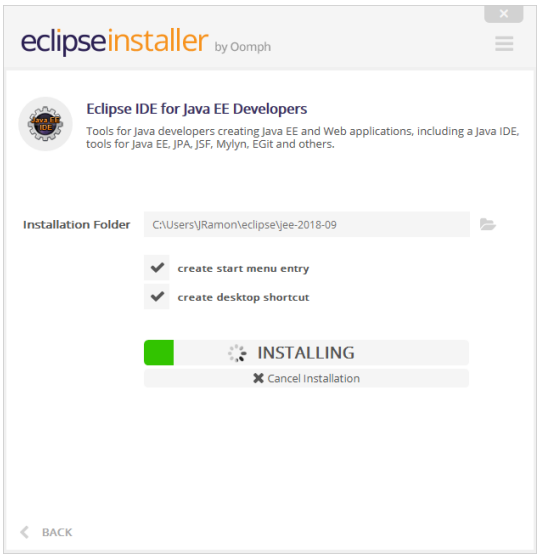


Per seleccionar la carpeta correcta, cal tenir en compte quins usuaris utilitzaran l'entorn. Tots ells han de tenir permís de lectura i execució sobre la carpeta en

qüestió. Un cop introduïda la carpeta, podem seleccionar les opcions dels menús i accessos directes o llençadores que desitgem i clicar el botó *INSTALL* perquè comenci la instal·lació.

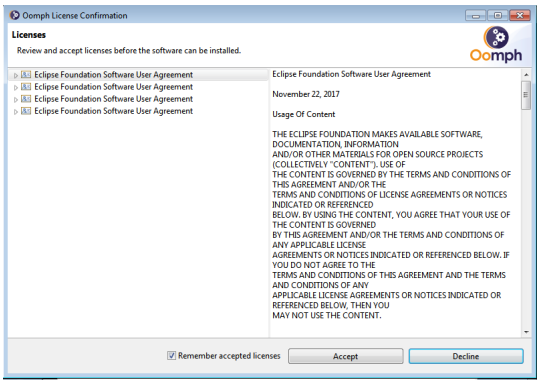
Durant la instal·lació ens apareixerà una pantalla de progrés com la que es veu a la figura 2.7.

FIGURA 2.7. Pantalla de progrés de la instal·lació



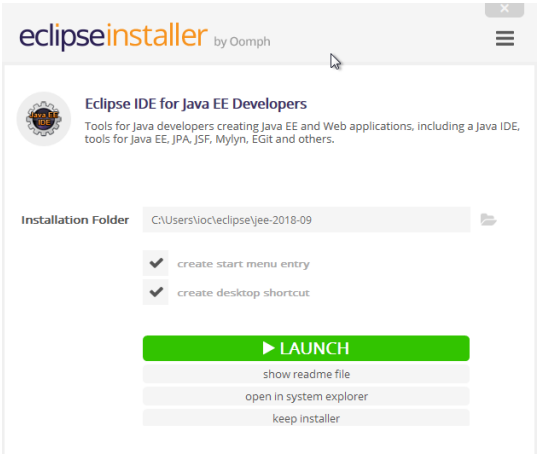
També se'ns demanarà que acceptem les llicències del programari que s'instal·larà, com mostra la figura 2.8.

FIGURA 2.8. Pantalla amb les llicències que cal acceptar



Un cop acabada la instal·lació se'ns mostra una pantalla com la de la figura 2.9 que ens convida a executar directament l'entorn.

FIGURA 2.9. Instal·lació finalitzada



Aquest primer cop podrem executar l'entorn Eclipse fent clic al botó *LAUNCH*. La resta de vegades, caldrà invocar-lo des dels accessos directes o llençadores, si s'han creat o, en cas contrari, invocant directament l'executable. Aquest s'anomena *eclipse* i el trobareu a en una subcarpeta de la carpeta d'instal·lació que s'anomena també *eclipse*. La ruta exacta pot variar d'una versió a una altra.

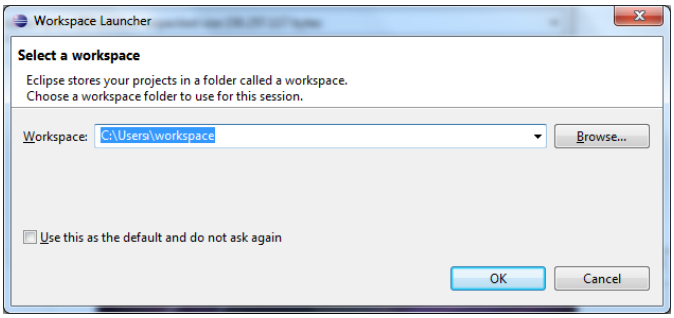
Si en un futur calgués desinstal·lar-lo, només s'hauria d'esborrar la carpeta on s'ha instal·lat ja que la instal·lació d'Eclipse no apareix al respositori de Linux ni al panell de control a Windows.

Quan executem l'entorn, ens apareixerà una pantalla com la que mostra la figura figura 2.10.

FIGURA 2.10. Pantalla d'inici d'Eclipse

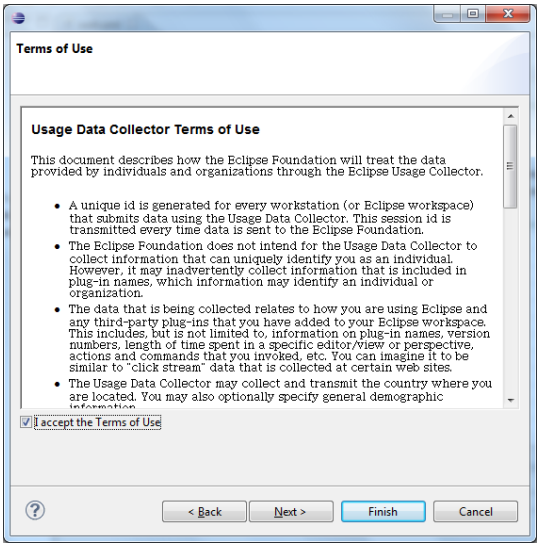
De seguida se'ns demanarà demanarà a quina carpeta caldrà ubicar l'espai de treball, com es veu a la figura 2.11. Podem demanar-li que el recordi per a la resta d'execucions activant l'opció "Use this as the default and do not ask again".

FIGURA 2.11. Escollir la ubicació de l'espai de treball



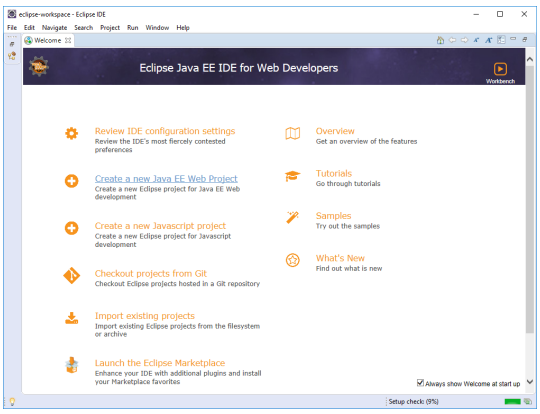
A partir d'aquest moment, ja es podrà utilitzar l'IDE Eclipse, havent d'acceptar en la primera execució els termes d'ús establerts (figura 2.12).

FIGURA 2.12. Acceptació dels termes d'ús de l'aplicació Eclipse



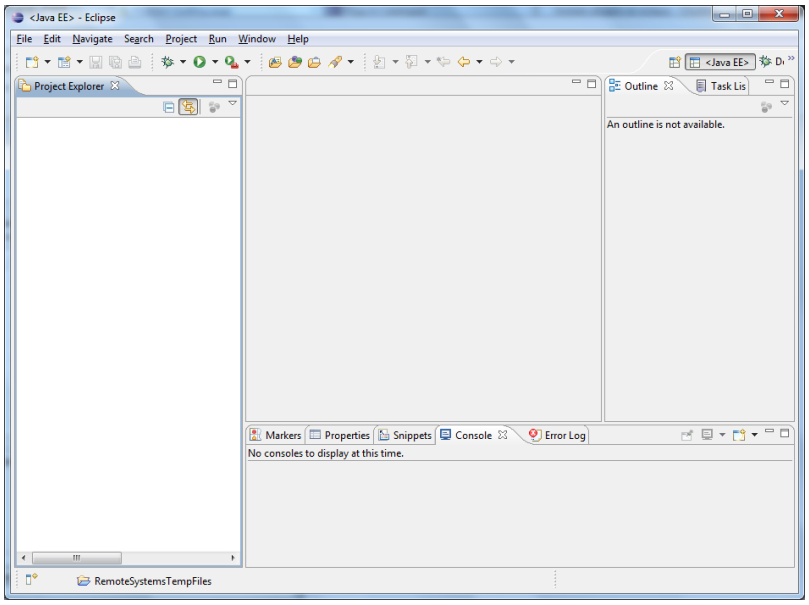
La primera vegada que l'executem, es mostrarà la pestanya de benvinguda, com es veu a la figura 2.13. Podem demanar que no ens la mostri més desactivant l'opció "Always show Welcome at start up".

FIGURA 2.13. Entorn de treball d'Eclipse



Un cop tancada aquesta pestanya, l'entorn de treball serà similar al mostrat per la figura 2.14.

FIGURA 2.14. Entorn de treball d'Eclipse

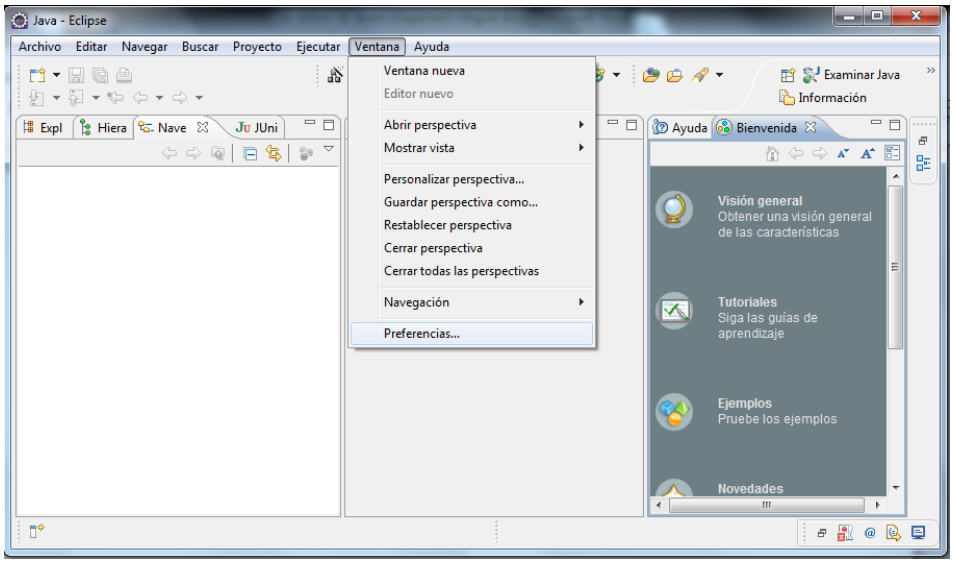


2.2.5 Configuració de JDK amb l'IDE Eclipse

Arribats a aquest punt, es parametritzarà l'entorn d'Eclipse amb el JDK instal·lat. Es pot observar aquest procediment a les figures següents.

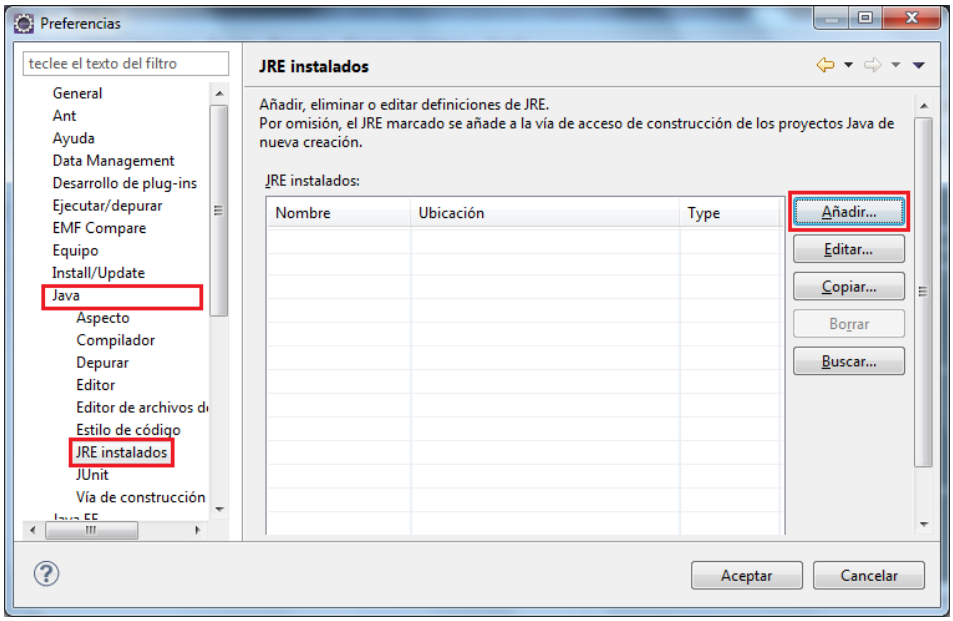
A la figura 2.15 es mostra l'opció de menú que cal seleccionar.

FIGURA 2.15. Parametrització de l'entorn d'Eclipse



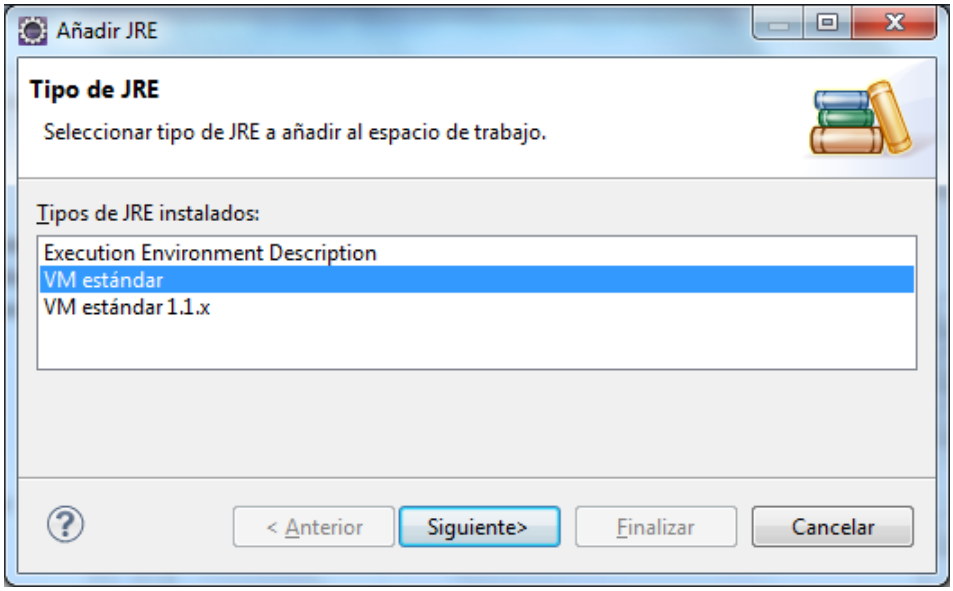
A la figura 2.16 s'arriba a l'apartat "Preferències", on es mostren els JRE instal·lats. Per defecte, no hi ha cap JRE instal·lat, amb la qual cosa s'haurà d'afegir el que s'hagi instal·lat.

Figura 2.16. Especificació del JRE o JDK instal·lat



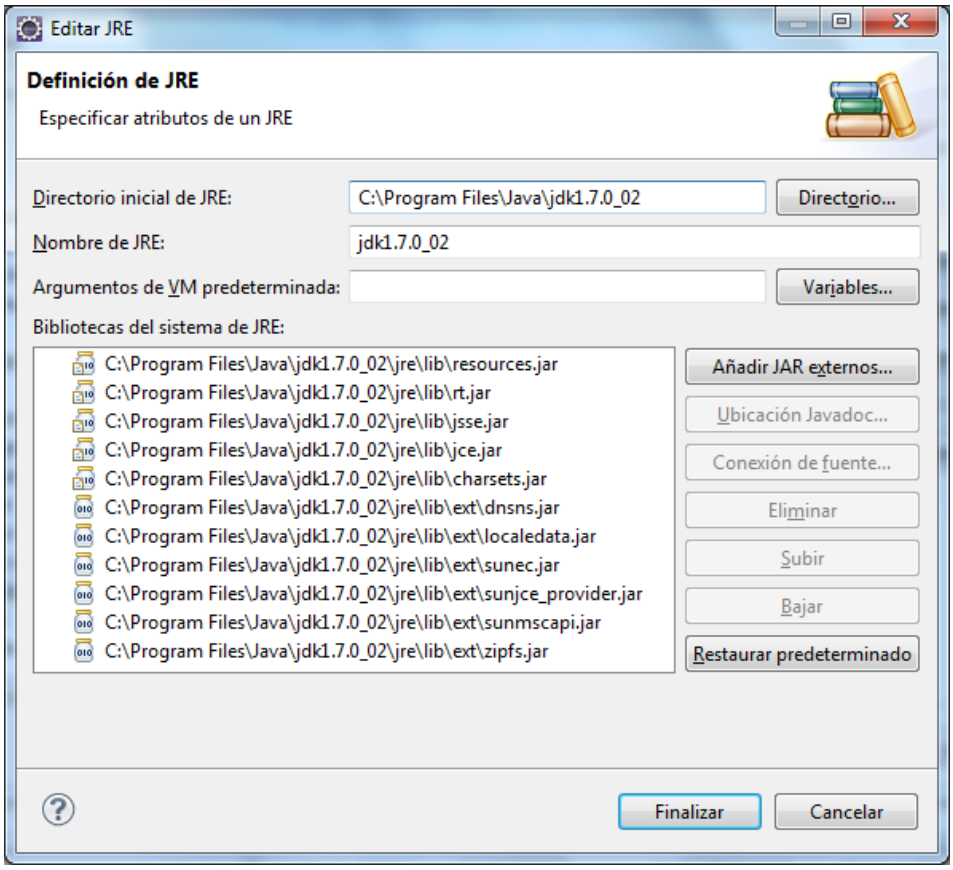
A la figura 2.17 es mostra el diàleg per afegir el JRE seleccionat.

Figura 2.17. Especificació del JRE o JDK instal·lat



Tot seguit caldrà especificar la ruta on s'ha instal·lat el JDK, abans de finalitzar la parametrització (figura 2.18).

FIGURA 2.18. Especificació del JRE o JDK instal·lat



2.2.6 Configuració del servidor web amb l'IDE Eclipse

A la figura 2.19 i figura 2.20 es mostra com seleccionar la pestanya Servers de l'IDE per fer-la visible, si no ho estava prèviament.

FIGURA 2.19. Mostrar la vista de servidors

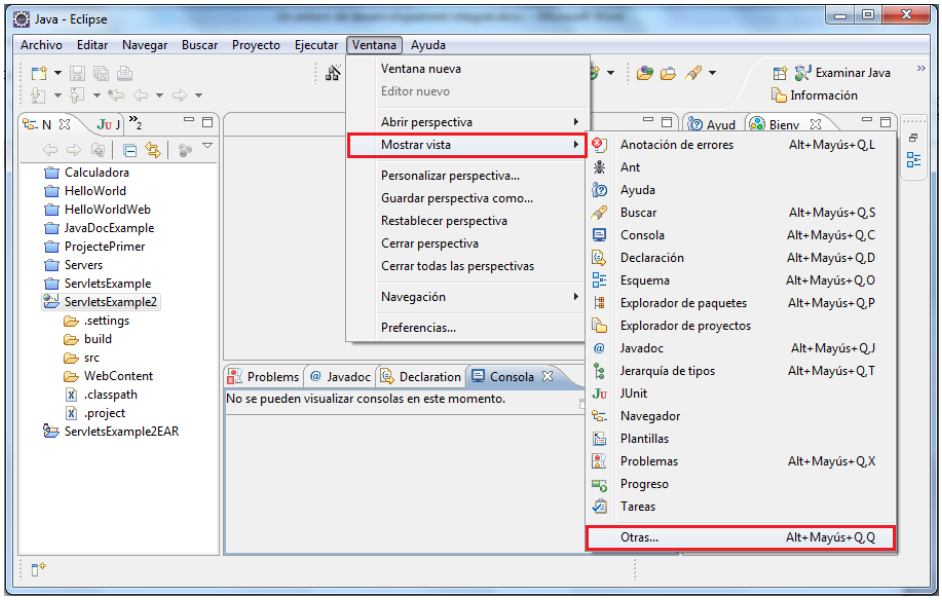
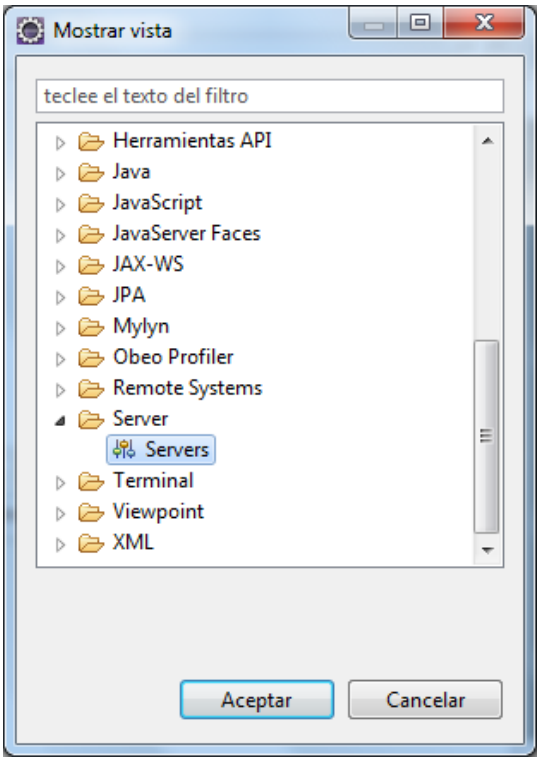
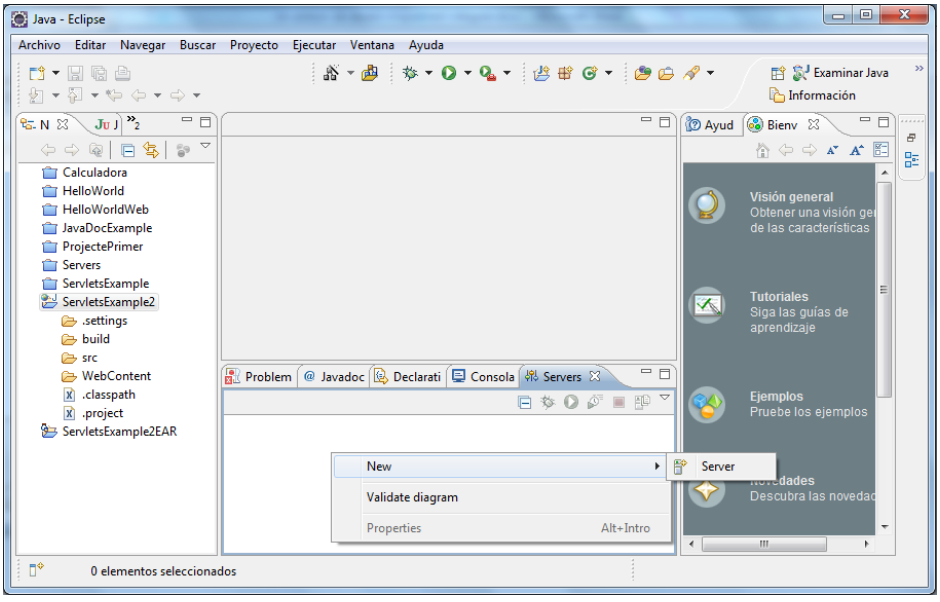


FIGURA 2.20. Selecció de la vista a visualitzar: servidors



En la part inferior de la figura 2.21 es visualitza la vista de servidors buida; s'hi haurà d'indicar el servidor d'aplicacions.

FIGURA 2.21. Afegir un nou servidor



A la figura 2.22 i figura 2.23 queda explicat com indicar quin és el servidor d'aplicacions.

Cal tenir cura de triar la versió que s'ha instal·lat (les captures s'han realitzat amb la versió 7, però la vostra pot ser una altra).

FIGURA 2.22. Selecció del servidor

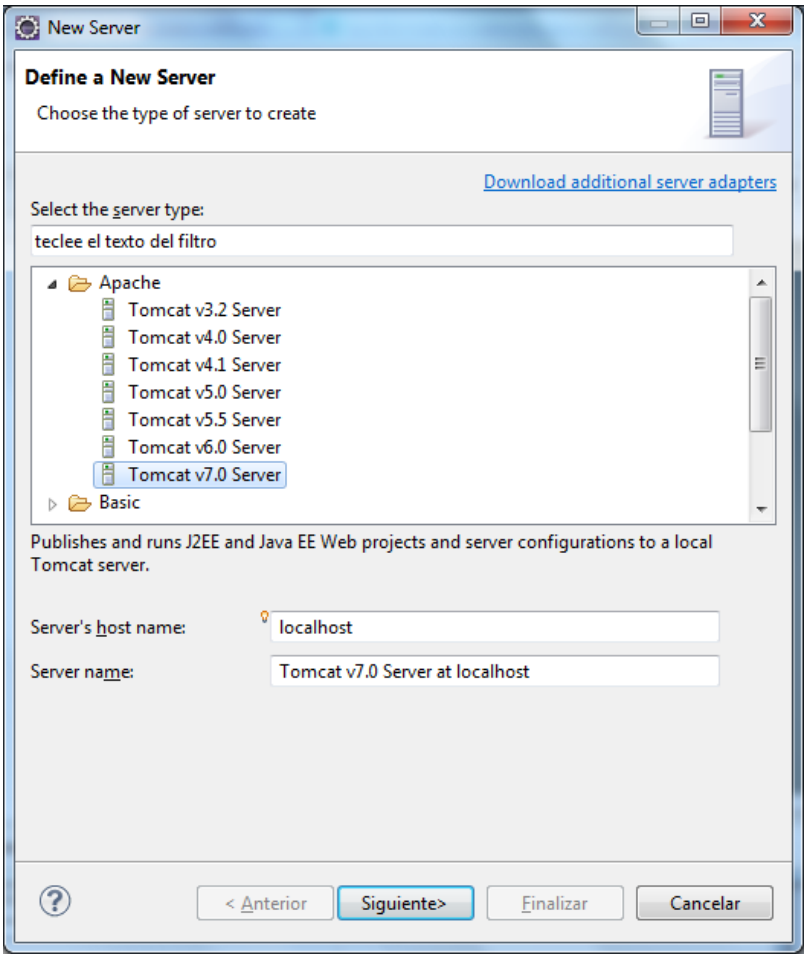
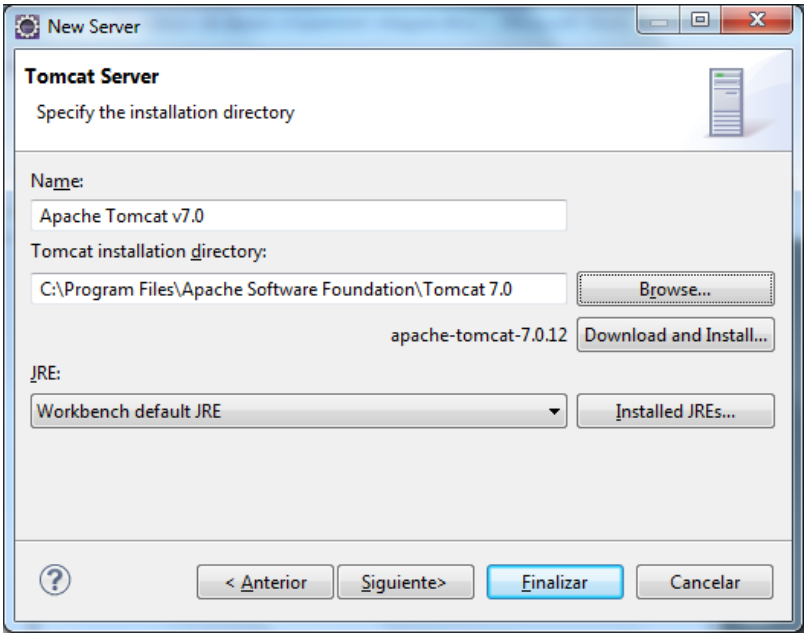


FIGURA 2.23. Especificació de la ruta del servidor



Podeu consultar l'annex "Instal·lació de connectors" en els materials Web.

2.2.7 Instal·lació de connectors

Una vegada instal·lat l'Eclipse, caldrà afegir aquells complements que siguin necessaris per a una correcta execució. La descàrrega bàsica de l'entorn Eclipse inclou algunes de les funcionalitats més bàsiques, però sempre és desitjable obtenir alguna funcionalitat extra. Aquestes funcionalitats es troben en els connectors, que caldrà localitzar, descarregar i instal·lar.

Connectors són un conjunt de components de programari que afegiran funcionalitats noves a les aplicacions instal·lades.

En l'apartat "Community" del lloc web oficial d'Eclipse es poden trobar enllaços a centenars de connectors. Aquests poden haver estat desenvolupats per programadors de la mateixa Fundació o haver-ho estat per usuaris de l'aplicació que volen compartir amb altres usuaris de forma altruista complements que ells mateixos han desenvolupat per solucionar alguna mancança que han trobat.

És molt important una **selecció acurada dels connectors**. Existeixen molts connectors que es poden instal·lar, però a mesura que es vagin afegint, això influirà en el rendiment de l'IDE Eclipse, especialment, en el temps d'arrencada inicial de l'aplicació.

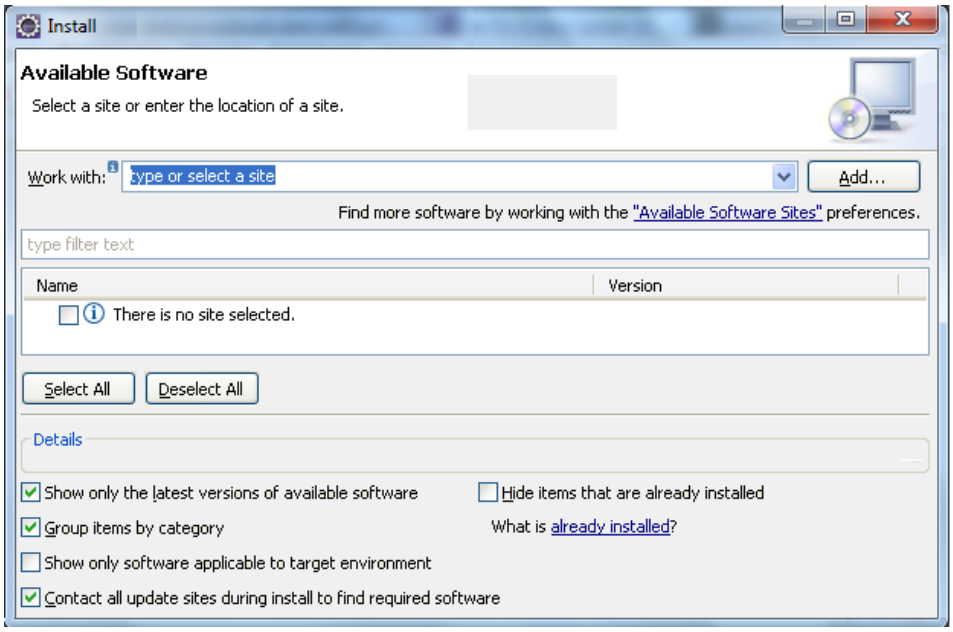
2.2.8 Instal·lació de components per al desenvolupament d'aplicacions GUI

El component que emprarem en aquesta unitat per desenvolupar aplicacions GUI és **WindowsBuilder Pro**.

Instal·lació del component

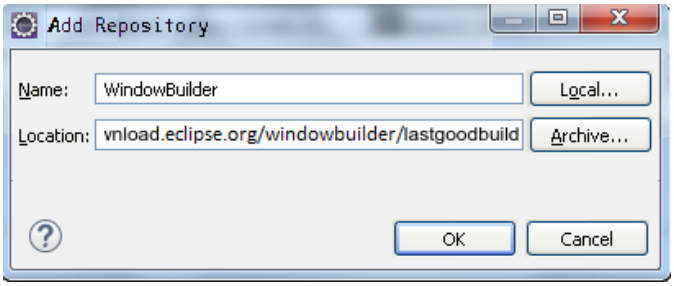
Per tal d'instal·lar-lo s'haurà d'accedir a l'opció del menú *Help* i seleccionar l'opció *Install new software*. Ens apareixerà la finestra de la figura 2.24.

Figura 2.24. Instal·lació de components



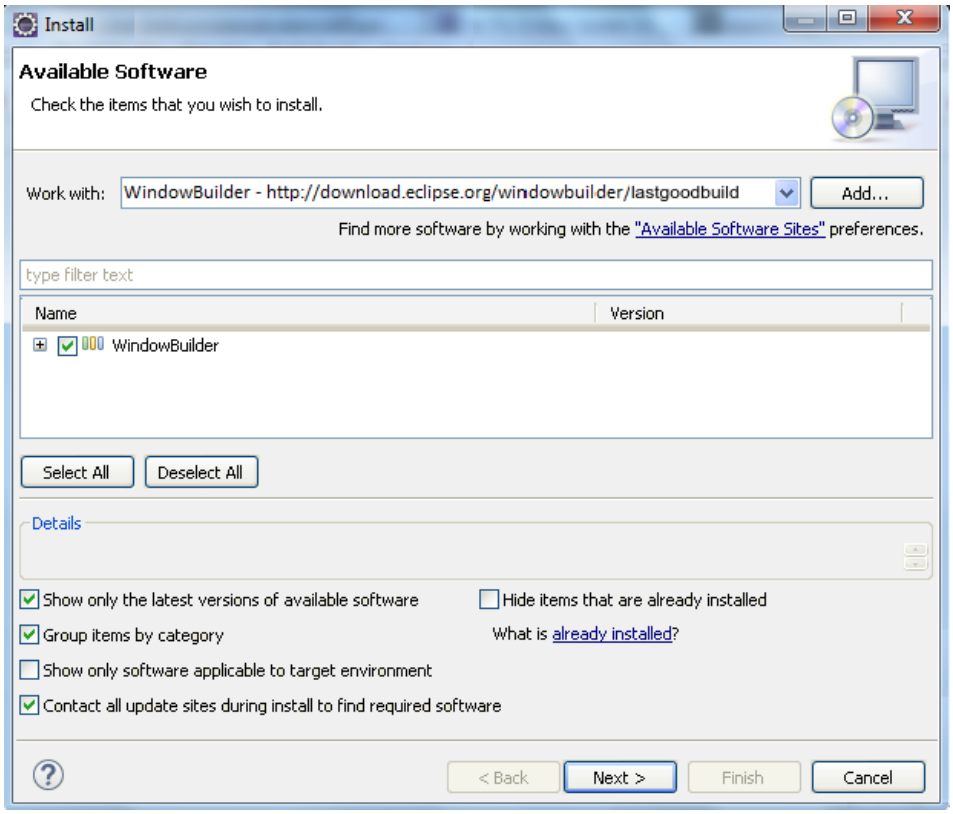
En ella caldrà fer clic al botó *Add...*. Se'ns obre una finestra com la de la figura 2.25. En ella posarem *WindowBuilder* a l'apartat *Name* i, a l'apartat *Location*, l'adreça bit.ly/2X4c31e, que és l'adreça des d'on s'instal·la el connector.

Figura 2.25. Afegir un repositori



Ens apareix una finestra com la de la figura 2.26. En ella cal fer clic al botó *Select All* i, a continuació, novament clic al botó *Next>*. Després, només caldrà seguir les indicacions de l'assistent i, quan aquest ens ho demani, reiniciar l'entorn Eclipse.

Figura 2.26. Instal·lació de WindowsBuilder



Obtenció del Run-time de SWT

Una vegada descarregat el Run-time de SWT (Standard Widget Toolkit), es descomprimeix i s'ubica en un determinat directori.

Podeu descarregar el Run-time de SWT a la pàgina www.eclipse.org/swt.

S'especifica la seva URL com a variable d'entorn.

```
1 CLASSPATH = URL/swt.jar
```

2.3 Ús bàsic d'un entorn de desenvolupament. Eclipse

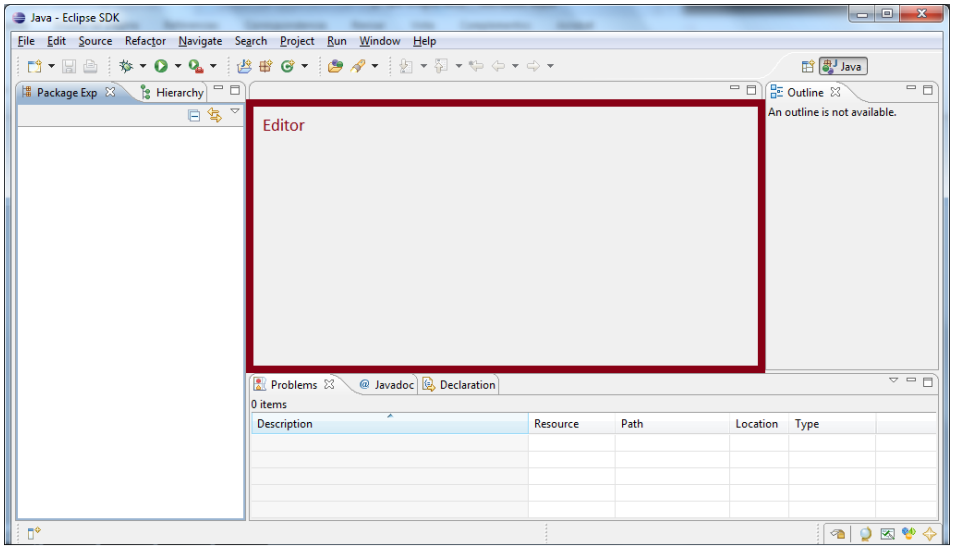
Una vegada instal·lat el programari, cal fer una revisió del seu entorn de treball i revisar els espais que ofereix i les funcions de cadascun d'ells. Dintre dels apartats es poden identificar, entre d'altres, els següents:

- Editor.
- Vistes.
- Barra d'eines principal.
- Barres de perspectives.

2.3.1 Editors

La finestra principal (la més gran a la figura 2.27) es diu Editor. Els editors són el lloc on es podrà desenvolupar el codi de programació, podent-lo afegir, modificar o esborrar. És possible tenir diversos editors oberts al mateix temps, apilats un sobre l'altre. A la part superior de la finestra d'editors, es mostraran solapes que permeten accedir a cada un dels editors oberts (o bé tancar-los).

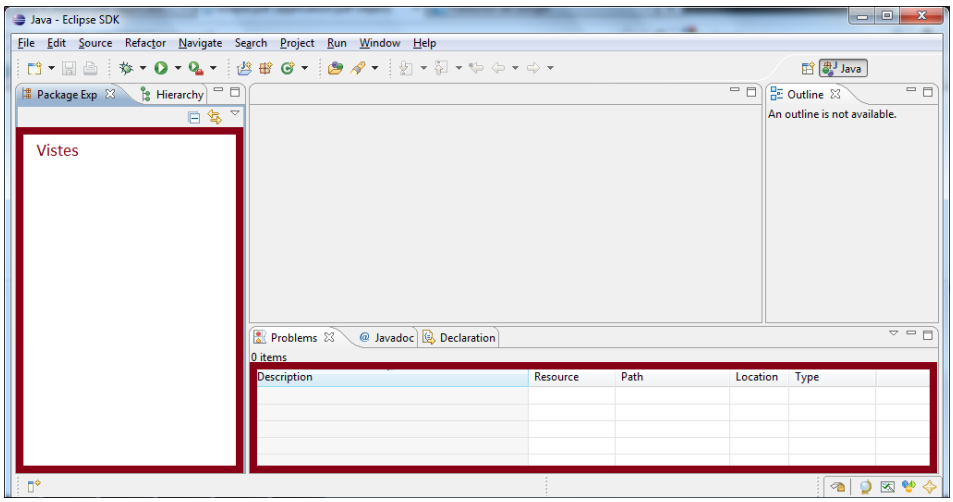
FIGURA 2.27. Identificació de l'espai Editor



2.3.2 Vistes

Les Vistes són un altre tipus de finestres, anomenades finestres secundàries. Serveixen per a qualsevol cosa, des de navegar per un arbre de directoris fins a mostrar el contingut d'una consulta SQL. Són finestres auxiliars destinades a mostrar informació i resultats de compilacions o logs, requerir dades...

FIGURA 2.28. Identificació dels espais Vistes

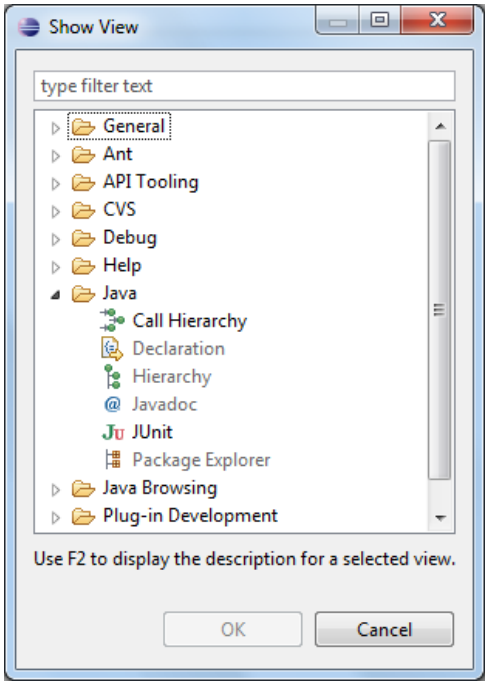


A la figura 2.28 es mostren dos tipus de vistes:

- La vista vertical de l'esquerra mostra la jerarquia dels projectes (quan n'hi hagi).
- La vista horitzontal inferior mostra un petit històric de tasques pendents que pot introduir l'usuari, de forma directa, o Eclipse, en funció de determinats esdeveniments.

Per seleccionar quines són les Vistes que s'han de mostrar, s'utilitza l'opció *Show View* al menú *Window* (vegeu la figura 2.29).

Figura 2.29. 'Show View'



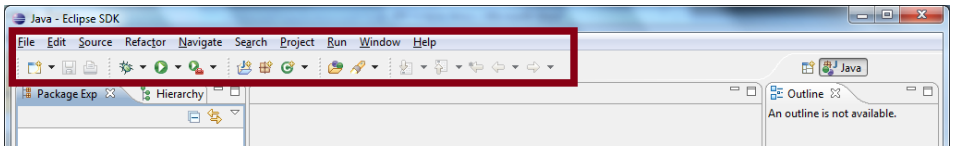
2.3.3 Barres d'eines

Un altre dels components de l'entorn són les Barres d'eines. N'hi ha dues: la Barra d'eines principal i la Barra de perspectives.

Barra d'eines principal

La Barra d'eines principal conté accessos directes a les operacions més usuals. És una barra de menú que tindrà l'accés a totes les opcions del programari (vegeu la figura 2.30).

FIGURA 2.30. Barra d'eines principal



Barra de perspectives

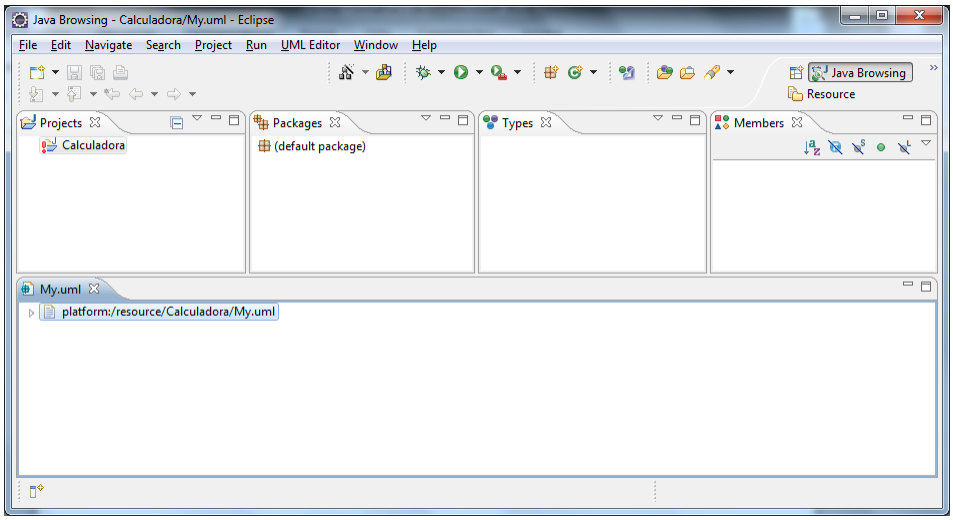
> Una **perspectiva** és un conjunt de finestres (editors i vistes) relacionades entre si.

La Barra de perspectives conté accessos directes a les perspectives que s'estan utilitzant en el projecte. Per exemple, hi ha una perspectiva Java que facilita el desenvolupament d'aplicacions Java i que inclou, a més de l'Editor, Vistes per navegar per les classes, els paquets...

Es pot seleccionar les perspectives actives -les que es mostren a la Barra de perspectives- utilitzant l'opció *Open Perspective* del menú *Window*. Des d'aquest mateix menú també és possible definir perspectives personalitzades. Es pot observar a la figura 2.31.

A més de la Barra d'eines principal, cada vista pot tenir la seva pròpia barra d'eines.

FIGURA 2.31. Barres de perspectives



2.4 Edició de programes

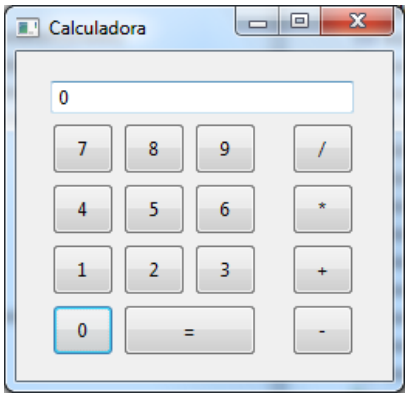
L'edició de programes és la part més important de les que es duran a terme amb Eclipse. Tant la creació com la modificació i el manteniment d'aplicacions necessitaran d'un domini de l'eina en les seves funcionalitats. Una vegada vist com instal·lar l'Eclipse i el seu entorn de treball, el millor per mostrar com editar, crear i mantenir programes és mostrar-ho amb un exemple.

Amb la versió estàndard de l'entorn integrat de desenvolupament Eclipse es distribueix el connector necessari per programar en llenguatge Java, el nom del qual és JDT. És important tenir això en compte, ja que Eclipse és un IDE que no està orientat específicament a cap llenguatge de programació en concret. Si es volgués aprofitar l'eina per desenvolupar en un altre llenguatge de programació, caldria descarregar el connector corresponent per tal que li donés suport.

2.4.1 Exemple d'utilització d'Eclipse: "projecte Calculadora"

Es tracta d'un exemple senzill d'implementar i, per això precisament, és el més adient per poder entendre com utilitzar de forma bàsica aquesta eina integrada de desenvolupament. A la figura 2.32 es pot observar un exemple de quina seria la interfície gràfica que es vol implementar.

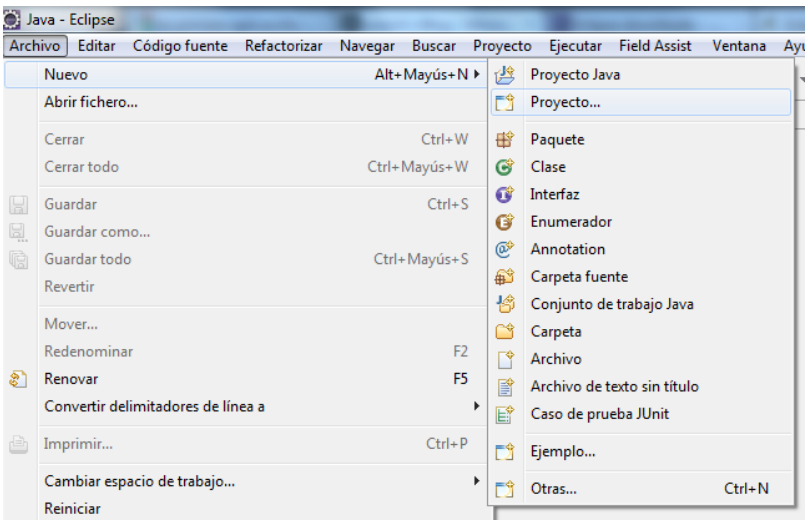
Figura 2.32. Calculadora



Per poder dur a terme un programa nou en Eclipse serà necessari crear un projecte. Per crear un projecte, com es pot veure a la figura 2.33, caldrà escollir l'opció del menú *Arxiu* / *Nou* / *Projecte*, des de la Barra d'eines principal o des de la vista Navigator (obrint el menú contextual amb el botó dret del ratolí i la opció *Nou* / *Project*).

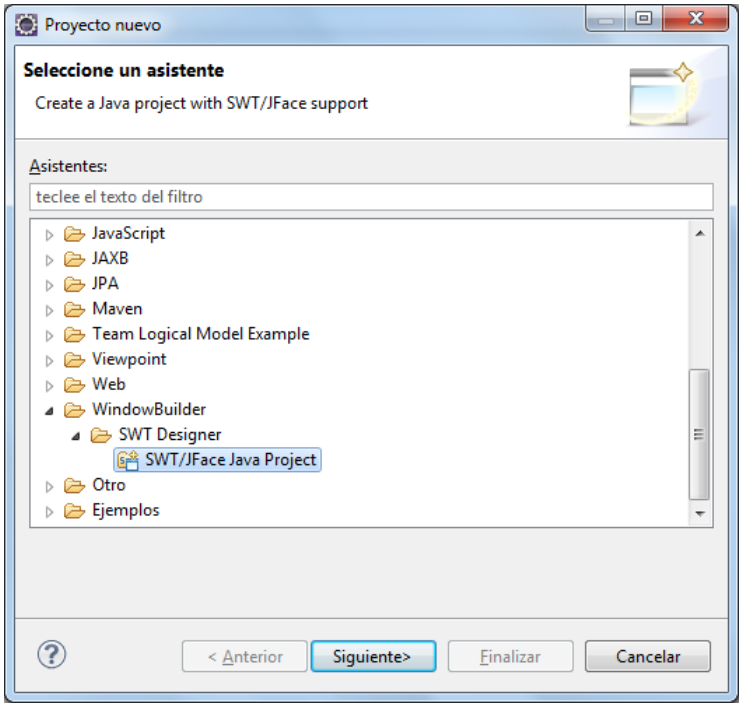
Un projecte agrupa un conjunt de recursos relacionats entre si (codi font, diagrames de classes, documentació...).

Figura 2.33. Nou projecte



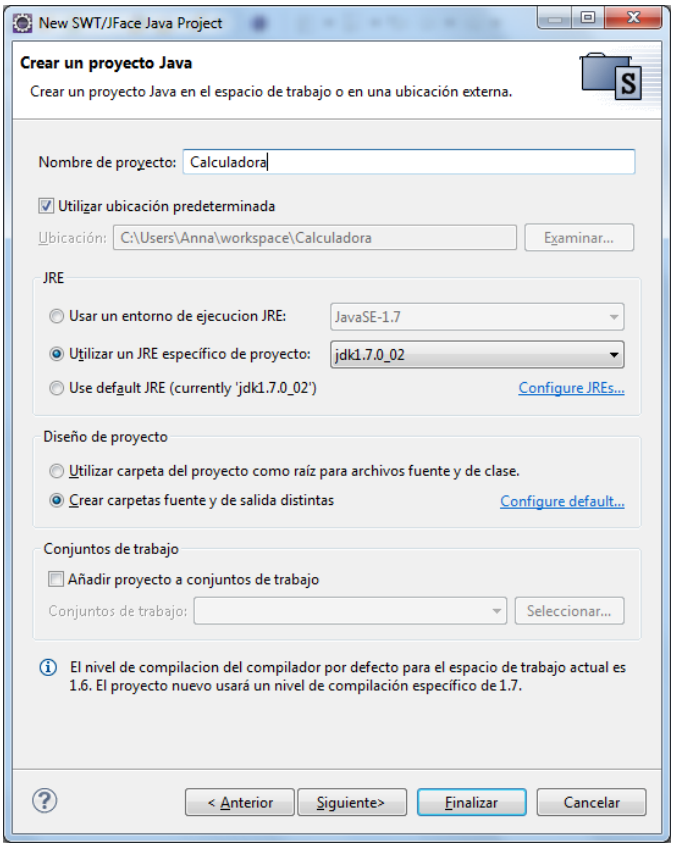
A la figura 2.34 es pot observar la finestra que es mostra al crear un nou projecte.

FIGURA 2.34. Projecte de tipus WindowBuilder



Qualsevol de les opcions mostrarà l'assistent de creació de projectes que es pot veure a la figura 2.35 i a la figura 2.36.

FIGURA 2.35. Assistent de creació del projecte

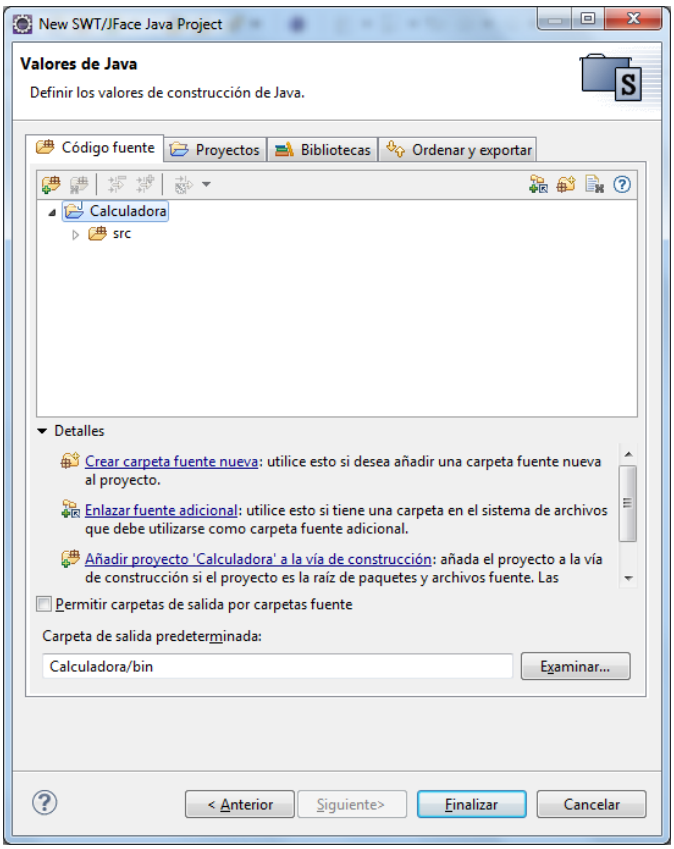


Per iniciar un nou projecte de tipus Window s'ha de seleccionar l'opció *Window-Builder / SWT Designer / SWT / JFace Java Project*.

Aquesta opció demanarà que s'indiqui un nom i una ubicació per al nou projecte. Com es pot observar a la figura 2.36, es podran parametritzar algunes característiques del nou projecte, com poden ser, entre altres:

- Seleccionar la carpeta on s'emmagatzemaran les llibreries (JAR) que necessita el projecte.
- Definir variables d'entorn.
- Indicar si hi haurà dependències del projecte que s'està creant en relació amb projectes ja duts a terme (existents en el mateix espai de treball).
- Crear una carpeta per emmagatzemar el codi de programació i una de diferent per emmagatzemar les classes una vegada ja compilades.

FIGURA 2.36. Assistent de creació del projecte

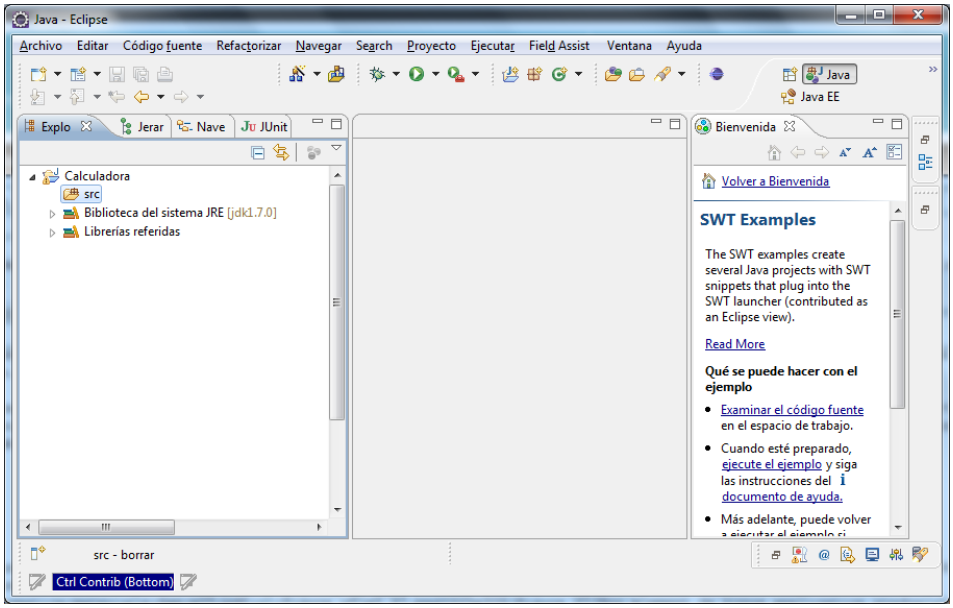


Dins d'aquestes opcions cal fer alguns comentaris. Cal recomanar sempre definir una carpeta per contenir el codi font, amb un nom clar i indicatiu del fet que hi haurà l'origen del projecte, i una altra amb un nom per indicar que contindrà els arxius binaris, on es deixaran els .class generats. També cal indicar que a l'opció de Llibreries es poden afegir tots els JAR que siguin necessaris.

Totes aquestes configuracions poden modificar-se en qualsevol moment a través del menú contextual de la vista Navegador, en l'opció *Propietats*.

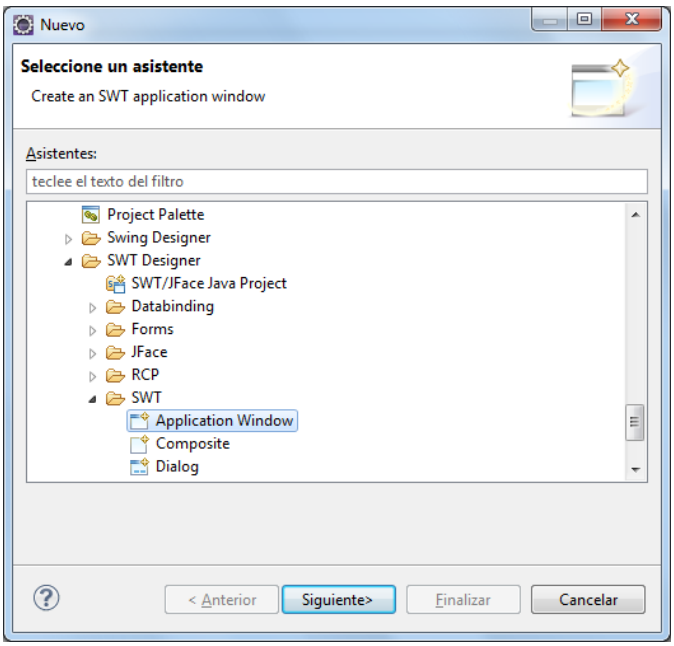
A la figura 2.37 es pot observar, en crear el projecte, com s'obrirà de forma automàtica la perspectiva Java, que és la col·lecció de Vistes que defineix el connector JDT. Si la perspectiva Java està activa, s'afegeixen a la Barra d'eines principal alguns botons extra que permeten accedir amb rapidesa a les funcions més usuals (execució del codi, depuració del codi, creació de noves classes...)

FIGURA 2.37. IDE una vegada creat el projecte



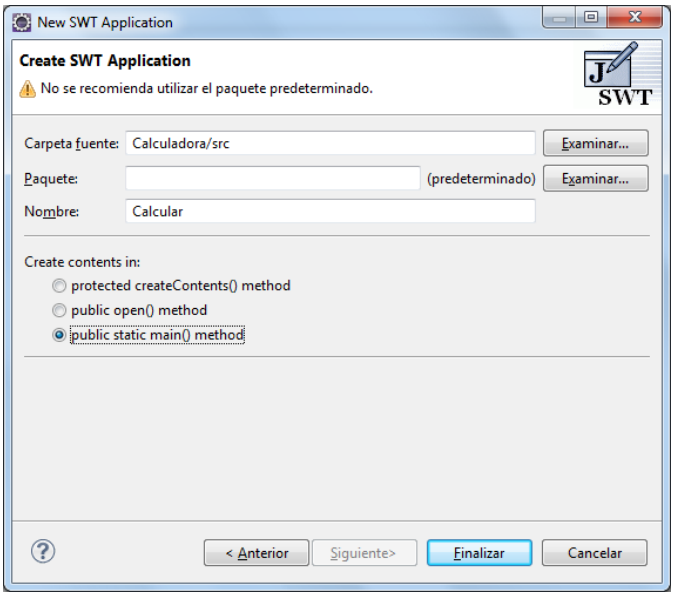
Una vegada creat el projecte, s'haurà de crear una nova classe on es dissenyarà la calculadora. La creació de la classe es du a terme executant, en la Barra d'eines, la ruta *Arxiu* / *Nou* / *Altres*, on s'inicia un assistent que permetrà especificar el tipus d'element a crear. En el cas de la Calculadora s'haurà de seleccionar *WindowBuilder* / *SWT Designer* / *SWT* / *Application Window* (vegeu la figura 2.38).

FIGURA 2.38. Nou element de tipus Application Window



Tal com es pot observar en la figura 2.39, l'assistent permet especificar la ruta on s'emmagatzemarà la classe, el paquet, el nom de la classe i el tipus de mètode que es crearà. En el cas de l'exemple se seleccionarà l'opció *public static main()method*.

FIGURA 2.39. Assistent del nou element

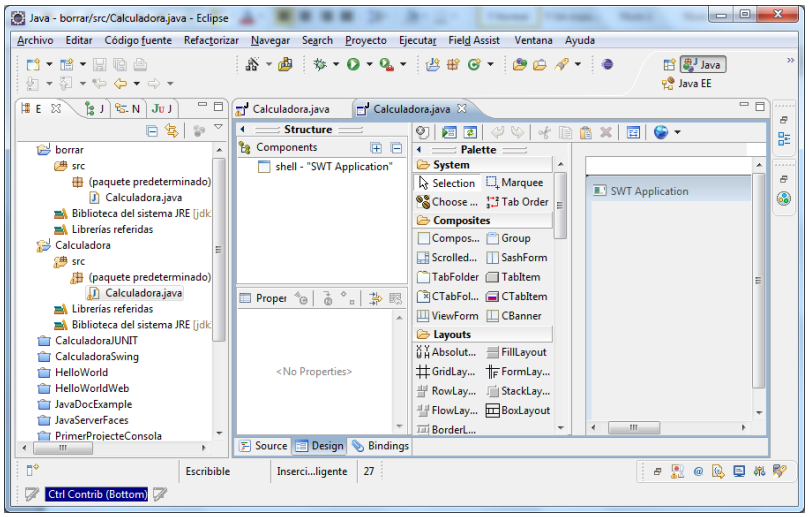


Automàticament, l'assistent ha creat el següent codi:

```
1  import org.eclipse.swt.widgets.Display;
2  import org.eclipse.swt.widgets.Shell;
3
4  public class Calculadora {
5      /**
6      * Launch the application.
7      * @param args
8      */
9      public static void main(String[] args) {
10         Display display = Display.getDefault();
11         Shell shell = new Shell();
12         shell.setSize(450, 300);
13         shell.setText("SWT Application");
14
15         shell.open();
16         shell.layout();
17         while (!shell.isDisposed()) {
18             if (!display.readAndDispatch()) {
19                 display.sleep();
20             }
21         }
22     }
23 }
```

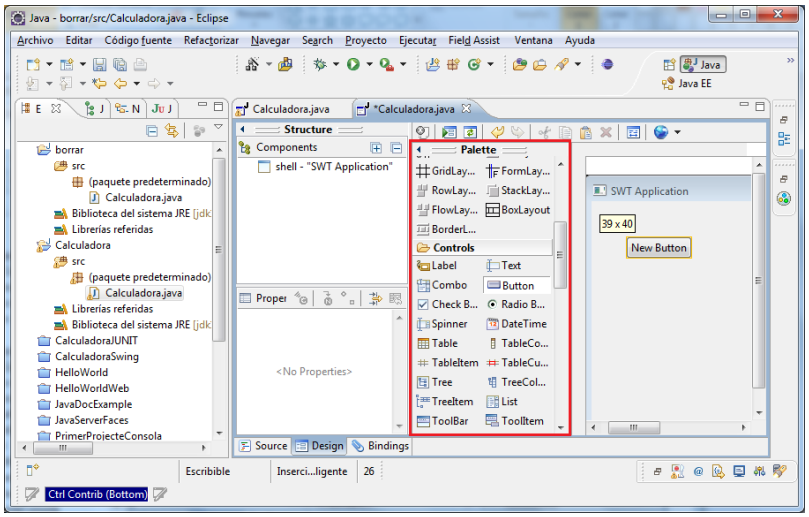
Arribats a aquest punt és quan es podrà començar a dissenyar gràficament el formulari de la calculadora fent ús de la vista Design (vegeu la figura 2.40).

FIGURA 2.40. Vista de disseny



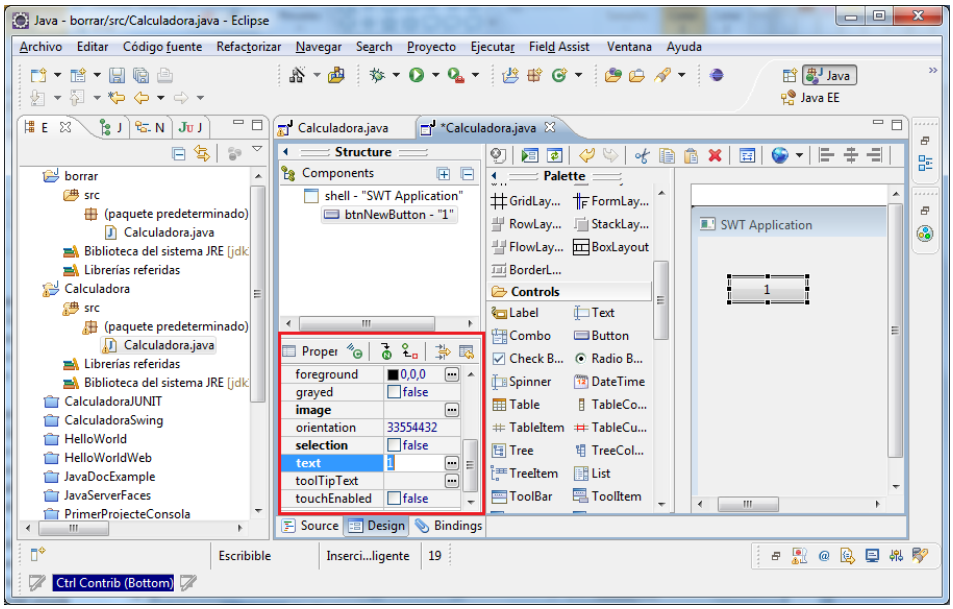
Es visualitza un conjunt de vistes que permet seleccionar un control, per exemple un botó, i desplaçar-lo cap al formulari (figura 2.41).

FIGURA 2.41. Afegir un botó



En la vista de Propietats es podrà modificar les propietats del control seleccionat (figura 2.42).

FIGURA 2.42. Especificar les propietats del botó



El codi que es genera automàticament en afegir el botó corresponent al número 1 de la calculadora és:

```
1 Button btnNewButton = new Button(shell, SWT.NONE);
2         btnNewButton.setBounds(30, 38, 47, 25);
3         btnNewButton.setText("1");
```

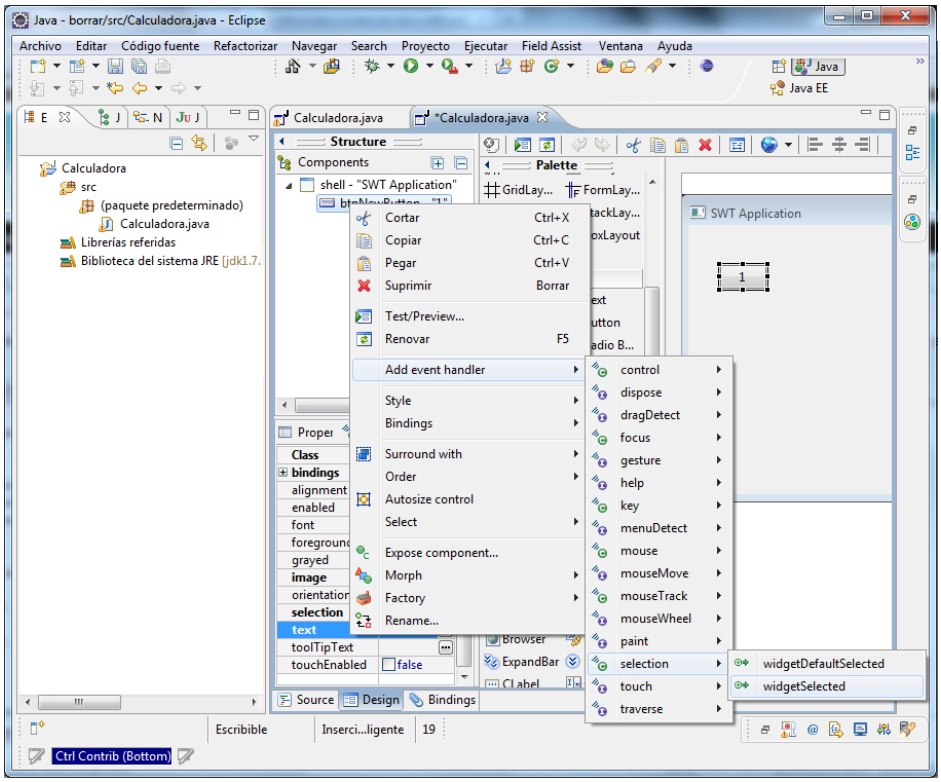
Es repeteix el procés per als 10 números (0,1,2,3,4,5,6,7,8,9), els operands (+, -, /, *) i l'igual (=).

El control utilitzat per mostrar el resultat de l'operació és un text.

El següent pas és activar l'esdeveniment per a cada un dels botons creats per tal de poder especificar l'acció a dur a terme.

En la vista de Components se selecciona el control i amb el botó dret se selecciona *Afegeix controlar d'esdeveniments* / *Seleccionar* / widget *seleccionat*. Es pot observar a la figura 2.43.

FIGURA 2.43. Esdeveniment en clicar el botó.



Automàticament, el codi corresponent al botó queda actualitzat.

```
1  Button btnNewButton = new Button(shell, SWT.NONE);
2       btnNewButton.addSelectionListener(new SelectionAdapter() {
3            @Override
4            public void widgetSelected(SelectionEvent e) {
5            }
6       });
7       btnNewButton.setBounds(30, 38, 47, 25);
8       btnNewButton.setText("1");
```

Syntax highlighting és el reconeixement sintàctic de expressions reservades del llenguatge.

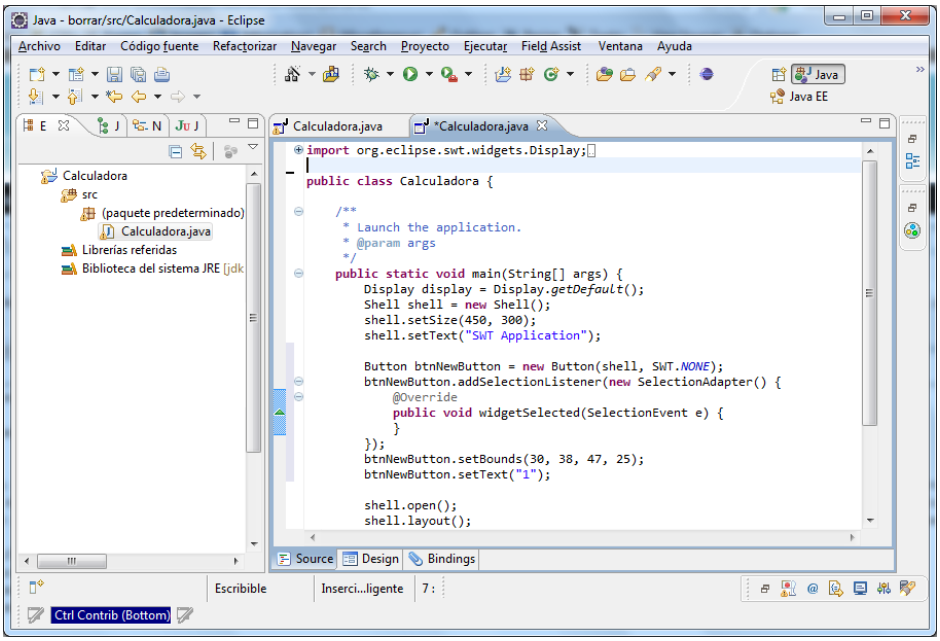
Es repeteix l'acció per a cadascun dels botons.

Una vegada s'ha generat automàticament el codi dels controls, caldrà començar a programar el codi Java que es vol desenvolupar. Tal com es pot veure en el codi generat de la classe `Calculadora`, algunes paraules mostren un color diferent a la resta. Aquesta forma de marcar amb colors les paraules és a causa que els Editors Java inclouen la capacitat per efectuar el ressaltat de la sintaxi (*syntax highlighting*).

La codificació en colors (figura 2.44) dels termes és:

- Les paraules reservades del llenguatge apareixeran escrites en color bordeus i en negreta.
- Els comentaris apareixeran en verd.
- Els comentaris de documentació (Javadoc) apareixeran en blau.

FIGURA 2.44. Codificació de colors



Es crearan els mètodes necessaris per tal de codificar la funcionalitat de la calculadora. Per exemple, es crea un mètode que executa l'operació de suma, resta, multiplicació o divisió entre dos nombres.

```
int executarOperacio() {
     int resultat = 0;
     int numeroVisualitzar = getResultatInt();

     //Operació: divisió
     if (última_operació.equals("/"))
          resultat = ultim_valor / numeroVisualitzar;

     //Operació: multiplicació
     if (ultima_operacio.equals("*"))
          resultat = ultim_valor * numeroVisualitzar;

     //Operació: resta
     if (ultima_operacio.equals("-"))
          resultat = ultim_valor - numeroVisualitzar;

     //Operació: suma
     if (ultima_operacio.equals("+"))
          resultat = ultim_valor + numeroVisualitzar;

     return resultat;
}
```

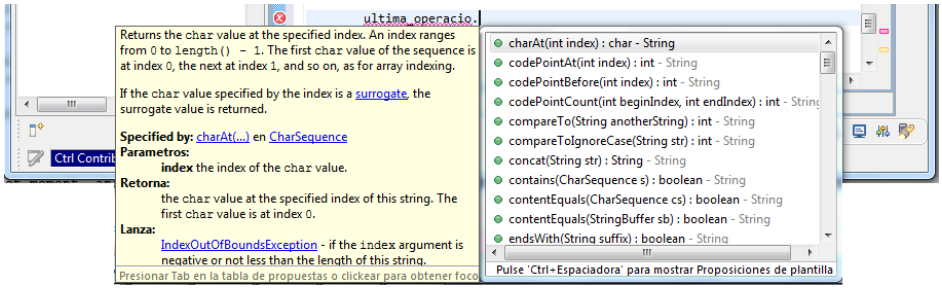
El mecanisme de *code completion* en Eclipse és molt similar al que implementen altres entorns integrats de desenvolupament: quan es deixa d'escriure durant un determinat interval de temps es mostren, si n'hi ha, tots els termes (paraules reservades, noms de funcions, de variables, de camps...) que comencen amb el o els caràcters escrits.

Code Completion vol dir compleció automàtica de les sentències inacabades.

Una altra característica és que escrivint un punt es provoca, sense esperar el temps establert, que s'ofereixin les alternatives o termes proposats. A la figura 2.45 es pot observar un exemple. També és interessant l'assistència a l'escriptura en trucades a funcions. Automàticament, quan s'estan a punt d'escriure els paràmetres que

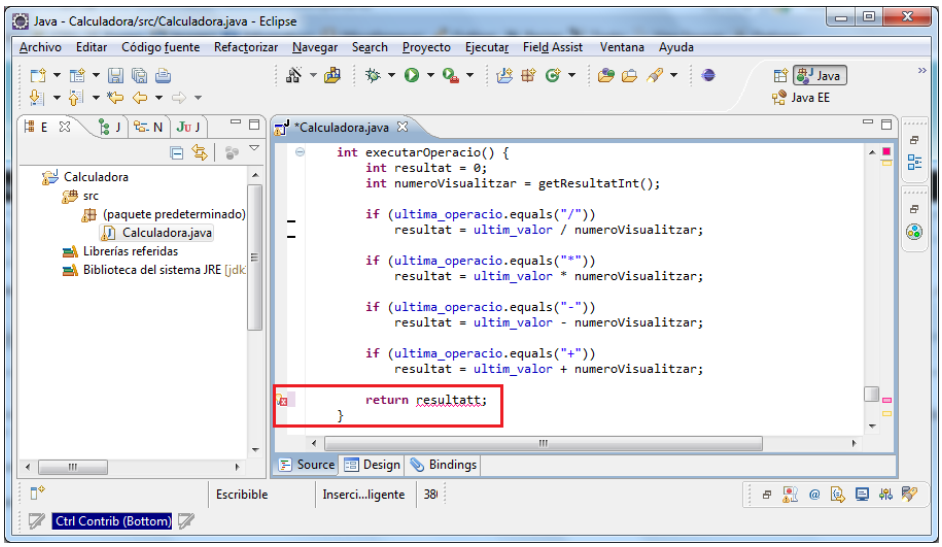
es passen a un mètode, es mostra una caixa de text indicant els tipus que aquests poden tenir.

Figura 2.45. Compleció de codi



D'altra banda, és bastant fàcil que a l'hora de codificar es cometi algun error.

Figura 2.46. Compleció de codi



El corrector d'errors serveix per a la detecció i el marcatge sobre el codi d'errors o avisos.

Com es pot observar a la figura 2.46, el connector JDT pot detectar, i marcar sobre el codi d'un programa, els llocs on es poden produir errors de compilació. Aquesta característica funciona de manera molt semblant als correctors ortogràfics que tenen els processadors de textos. Quan Eclipse detecta un error de compilació, s'ha de marcar la sentència errònia, subratllant amb una línia ondulada vermella (o groga, si en lloc d'un error es tracta d'un avís).

Si el programador posiciona el punter del ratolí sobre la instrucció que va produir la decisió, es mostrarà una breu explicació de per què aquesta instrucció s'ha marcat com a errònia.

Cada línia de codi que contingui un missatge d'error o un avís, també serà marcada amb una icona que apareixerà a la barra de desplaçament esquerra de l'Editor. Si el programador prem un cop sobre aquesta marca, es desplegarà un menú finestra emergent (*pop-up*) mitjançant el qual Eclipse mostrarà possibles solucions per als errors detectats.

Vegeu una possible codificació de la calculadora en l'annex "Calculadora: codi font" de la secció "Annexos".

L'error detectat en l'exemple de la figura 2.46 es troba en el nom de la variable. S'ha escrit erròniament *resultatt* en lloc de *resultat*.

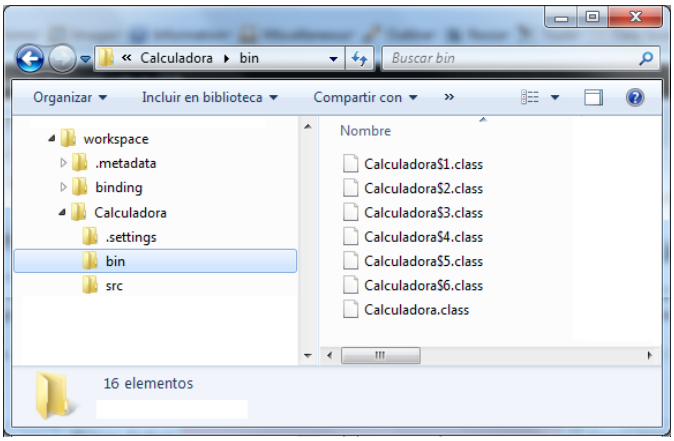
Una vegada desenvolupat el codi, caldrà **compilar-lo** per comprovar que el lèxic, la sintaxi i la semàntica siguin correctes, i generar, així, els fitxers binaris que es puguin executar.

Cal explicar com es compilen els projectes amb Eclipse, perquè ofereix un sistema una mica curiós. A Eclipse no existeix cap botó que permeti compilar individualment un fitxer concret. La compilació és una tasca que es llança automàticament en desar els canvis duts a terme en el codi. Per aquesta raó és pràcticament innecessari controlar manualment la compilació dels projectes.

Existeix una opció a l'entrada Project del menú principal, anomenada *Rebuild Project*, que permet llançar tot el procés de compilació complet, en cas que sigui necessari. Per compilar tots els projectes oberts, també es podrà utilitzar l'opció *Rebuild All*.

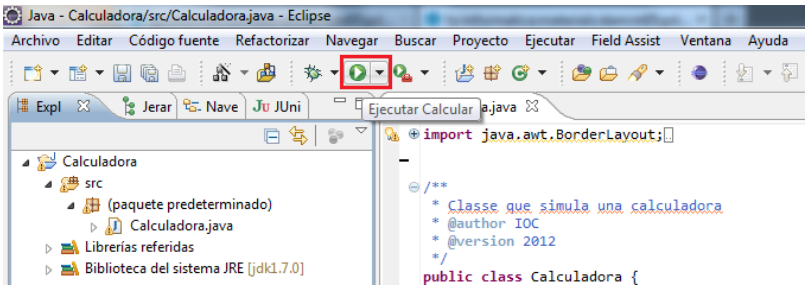
Tal com es mostra en la figura 2.47, en compilar, es genera una carpeta que sol denominar-se *bin* amb els fitxers .class, necessaris per poder executar el projecte.

Figura 2.47. Compilació



Una vegada el codi s'ha compilat, caldrà executar-lo per confirmar que fa el que s'havia planificat que hauria de fer. A la figura 2.48 es pot observar com accedir a la funcionalitat Run, que permetrà tant l'execució com la utilització d'altres opcions, entre les quals hi ha la d'accedir a la depuració per un altre camí.

Figura 2.48. Execució



2.5 Executables

En les aplicacions desenvolupades en el llenguatge C, C++, Visual Basic, Pascal... els fitxers binaris que genera el compilador únicament poden ser executats en la plataforma sobre la qual es va desenvolupar.

En les aplicacions desenvolupades en Java, els fitxers binaris que genera el compilador no és específic d'una màquina física en particular, sinó d'una màquina virtual, la qual cosa permet generar aplicacions compatibles amb diverses plataformes.

Per aquest motiu, l'IDE Eclipse no genera executables sinó codi intermig (codi de bytes) que s'emmagatzema en el directori .bin en forma de fitxers .class. La distribució de les aplicacions i posterior desplegament es pot dur a terme creant algun dels següents tipus d'encapsulaments:

- **JAR** (Java Archive): és un format d'arxiu independent de la plataforma que permet que diversos arxius puguin ser encapsulats dins d'un de sol, de manera que aquest pugui ser una aplicació completa de fàcil mobilitat i execució.
- **WAR** (Web Application archive): és un arxiu JAR (amb l'extensió WAR) usat per distribuir una col·lecció d'arxius JSP, servlets, classes Java, arxius XML i contingut web estàtic (HTML). En conjunt constitueixen una aplicació Web.
- **EAR** (Enterprise Archive File): és un format per empaquetar diversos mòduls en un sol arxiu. Permet desplegar diversos mòduls d'aquests en un servidor d'aplicacions. Conté arxius XML, anomenats descriptors de desplegament, que descriuen com efectuar aquesta operació (EAR = JAR + WAR).

En la figura 2.49, es mostra una representació gràfica dels diferents tipus d'encapsulaments.

FIGURA 2.49. Tipus de fitxers d'encapsulament