Estructura d'un programa informàtic. Tipus de dades simples

Joan Arnedo Moreno

Programació bàsica (ASX)
Programació (DAM)
Programació (DAW)

Índex

Introducció

Aquesta unitat formativa representa el punt de partida per entendre com podeu desenvolupar un programa d'ordinador que permeti dur a terme una tasca d'una certa complexitat. Se sol dir que programar té una part d'art, ja que aquest procés requereix un grau no menyspreable de creativitat. La veritat és que, com en moltes altres disciplines, incloses les artístiques, ser un bon programador implica practicar i practicar moltes hores. Només llegint un manual no hi ha prou ni de bon tros. Ara bé, tot i la necessitat d'aquest grau de creativitat, sí que hi ha un seguit de metodologies i fonaments teòrics bàsics que heu de tenir clars per poder plantejar-vos com començar. Aquests fonaments seran de gran utilitat posteriorment per facilitar la feina d'entendre i millorar la tècnica adquirida.

Per aprendre a programar primer cal començar per eines més senzilles disponibles per crear un programa, i a poc a poc anar assimilant-ne de més complexes. És important que aquesta progressió es faci de manera graonada per garantir que s'han entès els conceptes. Per tant, en aquesta unitat usareu l'ordinador pràcticament com si fos una calculadora. Tot i així, això serà suficient perquè veieu alguns dels aspectes bàsics que us acompanyaran sempre, com és saber gestionar dades bàsiques mitjançant un programa.

El nucli formatiu "Iniciació a la programació" proporciona una visió partint des de zero de què és un programa d'ordinador, com funciona i quin és el procés de creació a grans trets. Com a fil conductor per veure tots aquests conceptes s'usen les diferents maneres com es poden classificar els mecanismes per crear un programa. Per dur a la pràctica tot això, ja abans d'acabar el nucli seureu davant de l'ordinador i creareu un primer programa molt senzill.

Dins del nucli "Manipulació bàsica de dades" ja començareu a veure com s'estructuren les dades dins de l'ordinador i com és possible transformar-les per fer certs càlculs senzills i mostrar-ne el resultat per la pantalla. Atès que, en darrera instància, tot el que un ordinador fa és manipular dades, aquest nucli és especialment important amb vista a disposar dels conceptes fonamentals per poder entendre parts més complexes de la programació.

Tenint en compte la necessitat de practicar davant d'un ordinador per entendre els diferents conceptes exposats en aquesta unitat, al llarg del text trobareu un seguit de reptes en els quals us plantegem petits exercicis per complementar cada explicació concreta. És important que dediqueu una estona a resoldre'ls abans de seguir endavant. En qualsevol cas, també és molt recomanable que proveu de fer els exemples que anireu trobant en el vostre entorn de treball.

Resultats d'aprenentatge

En finalitzar aquesta unitat, l'alumne/a:

1. Reconeix l'estructura d'un programa informàtic, identificant i relacionant els elements propis del llenguatge de programació utilitzat.
2. Identifica els blocs que componen l'estructura d'un programa informàtic.
3. Crea projectes de desenvolupament d'aplicacions i utilitza entorns integrats de desenvolupament.
4. Identifica els diferents tipus de variables i la utilitat específica de cadascun.
5. Modifica el codi d'un programa per crear i utilitzar variables.
6. Crea i utilitza constants i literals.
7. Classifica, reconeix i utilitza en expressions els operadors del llenguatge.
8. Comprova el funcionament de les conversions de tipus explícit i implícit.
9. Introdueix comentaris en el codi.

1. Iniciació a la programació

Com en qualsevol procés d'aprenentatge, cal començar pel principi. És important tenir clars un conjunt de conceptes bàsics que ajudin a comprendre els conceptes més avançats que vindran posteriorment. En aquest cas, es tracta d'establir què és un programa, com funciona i quin és el model general per crear-ne un. Només un cop ho tingueu clar us podeu plantejar seure davant de l'ordinador i començar a programar.

1.1 Què és un programa?

Un primer pas per poder començar a estudiar com cal fer un programa informàtic és tenir clar què és un programa. 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. Us podríeu estar referint al programa d'un cicle de conferències o de cinema. Però, tot i no tractar-se d'un context informàtic, aquest ús ja us aporta una idea general del seu significat: un conjunt d'esdeveniments ordenats de manera que succeeixen de forma seqüencial en el temps, un darrere l'altre.

Un altre ús habitual, ara ja sí que vinculat al context de les màquines i els autòmats, podria ser per referir-se al programa d'una rentadora o d'un robot de cuina. En aquest cas, però, el que succeeix és 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 blat de moro seria:

1. Espera que introduïu blat de moro i mantega.
2. Gira durant un minut, avançant progressivament de la velocitat 1 a la 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 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'estructura la mena de tasques que aquests poden fer té molt en comú amb els programes d'electrodomèstics. 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.
- Un videojoc processa les dades que diuen la forma i ubicació d'enemics i jugadors, etc.
- Un navegador web processa les ordres de l'usuari i les dades que rep des d'un servidor a Internet.

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. La dificultat de tot plegat serà més gran o petita 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 l'executeu, l'ordinador complirà totes les ordres del programa.

Executar un programa

Per "executar un programa" s'entén fer que l'ordinador segueixi totes les seves ordres, des de la primera fins la darrera.

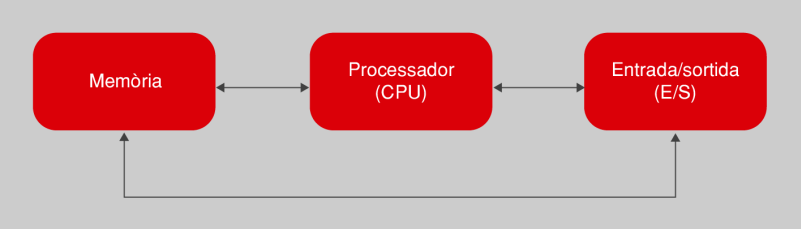
De fet, un ordinador és incapaç de fer absolutament res per si mateix, sempre cal dir-li què ha de fer. I això se li diu mitjançant l'execució de programes. Tot i que des del punt de vista de l'usuari pot semblar que quan es posa en marxa un ordinador aquest funciona sense executar cap programa concret,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

1.2 Tipus d'ordres que accepta un ordinador

Per dur a terme la tasca encomanada, un ordinador pot acceptar diferents tipus d'ordres. Aquestes es troben limitades a les capacitats dels components que el conformen, de la mateixa manera que el programa d'una rentadora no pot incloure l'ordre de gratinar, ja que no té gratinador. Per tant, és important tenir present aquest fet per saber què es pot demanar a l'ordinador quan creeu un programa.

L'estructura interna de l'ordinador es divideix en un seguit de components, tots comunicats entre si, tal com mostra la figura 1.1 de manera molt simplista, però suficient per començar. Cada ordre d'un programa està vinculada d'una manera o altra a algun d'aquests components.

FIGURA 1.1. Components bàsics d'un ordinador



El **processador** és el centre neuràlgic de l'ordinador i l'element que és capaç de dur a terme les ordres de manipulació i transformació de les dades. Un conjunt de dades es pot transformar de moltes maneres, segons les capacitats que ofereixi cada processador. Tot i així, hi ha moltes transformacions que tots poden fer. Un exemple és la realització d'operacions aritmètiques (suma, resta, multiplicació, divisió), tal com fan les calculadores.

Processador

El processador també és conegut popularment per les seves sigles en anglès: CPU (*central processing unit*, unitat central de processament).

La **memòria** permet emmagatzemar dades mentre aquestes no estan essent directament manipulades pel processador. Qualsevol dada que ha de ser tractada per un programa estarà a la memòria. Mitjançant el programa es pot ordenar al processador que desi certes dades o que les recuperi en qualsevol moment. Normalment, quan es parla de memòria a aquest nivell ens referim a memòria dinàmica o RAM (*random access memory*, memòria d'accés aleatori). Aquesta memòria no és persistent i un cop acaba l'execució del programa totes les dades amb les quals tractava s'esvaeixen. Per tant, la informació no es desarà entre successives execucions diferents d'un mateix programa.

En certs contextos és possible que ens trobem també amb memòria ROM (*read-only memory*, memòria només de lectura). En aquesta, les dades estan predefinides de fàbrica i no s'hi pot emmagatzemar res, només podem llegir el que conté. Cal dir que no és el cas més habitual.

El sistema d'**entrada/sortida** (abreujat com a E/S) permet l'intercanvi de dades amb l'exterior de l'ordinador, més enllà del processador i la memòria. Això permet traduir la informació processada en accions de control sobre qualsevol perifèric connectat a l'ordinador. Un exemple típic és establir una via de diàleg amb l'usuari, ja sigui per mitjà del teclat o del ratolí per demanar-li informació, com per la pantalla, per mostrar els resultats del programa. Aquest sistema és clau per convertir un ordinador en una eina de propòsit general, ja que el capacita per controlar tota mena d'aparells dissenyats per connectar-s'hi.

Una altra possibilitat important de l'ordinador, ateses les limitacions del sistema de memòria, és poder interactuar amb el maquinari d'emmagatzemament persistent de dades, com un disc dur.

Un ordinador és com una pizzeria

Si es vol fer un símil amb el nostre món de cada dia, un ordinador és com la cuina d'una pizzeria que accepta comandes telefòniques. Fer una comanda equival a demanar l'inici de l'execució d'un programa. Per dur a terme aquesta comanda, caldrà manipular un seguit d'ingredients, que representarien les dades. El cuiner amb els seus estris (forn, pastador, etc.) serien el processador, ja que manipulen i transformen els ingredients. La nevera, els armaris o els contenidors, d'on el cuiner pot treure ingredients o on els pot desar mentre no els està manipulant, representarien la memòria. El sistema d'entrada/sortida serien els elements de comunicació amb l'exterior de la pizzeria, com el motorista que porta la pizza o el telèfon que el cuiner pot utilitzar per demanar que li portin nous ingredients quan se li acaben, demanar informació addicional a l'usuari ("S'ha acabat el pebrot, va bé si hi posem ceba?"), o avisar-lo d'algun esdeveniment ("Em sap greu, trigarà una mica més del previst"). De fet, continuant amb el símil, el cuiner prepara una pizza seguint un conjunt de passes. En aquest cas la recepta són les ordres que ha de seguir el programa. I si el cuiner no té la recepta no pot dur a terme la comanda.

Llenguatge natural

El llenguatge natural és aquell que emprem els humans per comunicar-nos habitualment.

Partint d'aquesta descripció de les tasques que pot dur a terme un ordinador segons els elements que el componen, un exemple de programa per multiplicar dos nombres és el mostrat a la taula 1.1. El teniu expressat en llenguatge natural. Noteu com les dades han d'estar sempre emmagatzemades a la memòria per poder-hi operar.

TAULA 1.1. Un programa que multiplica dos nombres usant llenguatge natural

Ordre per donar	Element que l'efectua
1. Llegeix un nombre del teclat.	E/S (teclat)
2. Desa el nombre a la memòria.	Memòria
3. Llegeix un altre nombre del teclat.	E/S (teclat)
4. Desa el nombre a la memòria.	Memòria
5. Recupera els nombres de la memòria i fes-ne la multiplicació.	Processador
6. Desa el resultat a la memòria.	Memòria
7. Mostra el resultat a la pantalla.	E/S (pantalla)

1.3 Crear un programa executable...

Per crear un programa cal establir quines ordres s'han de donar a l'ordinador i en quina seqüència. Ara bé, avui dia els ordinadors encara no entenen el llenguatge natural (com s'utilitza a la taula 1.1), ja que està ple d'ambigüitats i aspectes semàntics que poden dependre del context.

Artificial

Per *artificial* entenem allò que no ha evolucionat a partir de l'ús entre humans, sinó que ha estat creat expressament, en aquest cas per ser usat amb els ordinadors.

Per especificar les ordres que ha de seguir un ordinador el que s'usa és un **llenguatge de programació**. Es tracta d'un llenguatge artificial dissenyat expressament per crear algorismes que puguin ser duts a terme per l'ordinador.

Igual com hi ha moltes llengües diferents, també hi ha molts llenguatges de programació, cadascun amb les seves característiques pròpies, que els fan més

o menys indicats per resoldre uns tipus de tasques o altres. Tots, però, tenen una sintaxi molt definida, que cal seguir perquè l'ordinador interpreti correctament cada ordre que se li dóna. És exactament el mateix que passa amb les llengües del món: per expressar els mateixos conceptes, el català i el japonès usen paraules i normes de construcció gramatical totalment diferents entre si.

En un llenguatge de programació determinat, el seguit d'ordres concretes que es demana a l'ordinador que faci s'anomena conjunt d'**instruccions**.

Normalment, el conjunt d'instruccions d'un programa s'emmagatzema dins d'un conjunt de fitxers. Aquests arxius els edita el programador (vosaltres) per crear o modificar el programa. Per als programes més senzills n'hi ha prou amb un únic fitxer, però per als més complexos en pot caldre més d'un.

Els llenguatges de programació es poden classificar en diferents categories segons les seves característiques. De fet, algunes de les propietats del llenguatge de programació tenen conseqüències importants sobre el procés que cal seguir per poder crear un programa i per executar-lo. Hi ha dues maneres de classificar els llenguatges de programació:

Aquestes dues categories no són mútuament excloents.

- Segons si es tracta d'un llenguatge compilat o interpretat. Aquesta propietat afecta les passes que cal seguir per arribar a obtenir un fitxer executable. O sigui, un fitxer amb el mateix format que el de les aplicacions que podeu tenir instal·lades al vostre ordinador.
- Segons si es tracta d'un llenguatge de nivell alt o baix. Aquesta propietat indica el grau d'abstracció del programa i si les seves instruccions estan més o menys estretament vinculades al funcionament del maquinari d'un ordinador.

1.3.1 ...en llenguatge de màquina

El llenguatge de màquina o codi de màquina és el llenguatge que triaríem si volguéssim fer un programa que treballés directament sobre el processador. És interessant de conèixer perquè ajuda a entendre el procés de generació d'un programa.

En aquest llenguatge, cadascuna de les instruccions es representa amb una seqüència binària, en zeros (0) i uns (1), i tot el conjunt d'instruccions del programa queda emmagatzemat de manera consecutiva dins d'un fitxer de dades en binari. Si l'intenteu obrir amb un editor de text, el que veureu en pantalla són símbols totalment incomprensibles.

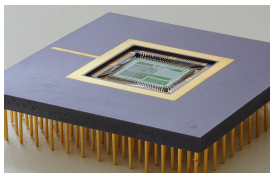
Transistor

El transistor és el component bàsic d'un sistema digital. Es pot considerar com un interruptor, en què 1 indica que hi passa corrent i 0 que no en passa.

Quan es demana l'execució d'un programa en llenguatge de màquina, aquest es carrega a la memòria de l'ordinador. Tot seguit, el processador va llegint una per una cadascuna de les instruccions, les descodifica i les converteix en

senyals elèctrics de control sobre els elements de l'ordinador per tal que facin la tasca demanada. A molt baix nivell, gairebé es pot arribar a establir una correspondència entre els 0 i 1 de cada instrucció i l'estat resultant dels transistors dins dels xips interns del processador.

El conjunt d'instruccions que és capaç de descodificar correctament un processador i convertir en senyals de control és específic per a cada model i està definit pel seu fabricant. El dissenyador de cada processador se'n va inventar la sintaxi i les instruccions del codi màquina d'acord amb les seves necessitats quan va dissenyar el maquinari. Per tant, les instruccions en format binari que pot descodificar un tipus de processador poden ser totalment incompatibles amb les que pot descodificar un altre. Això és lògic, ja que els seus circuits són diferents i, per tant, els senyals elèctrics de control que ha de generar són diferents per a cada cas. Dues seqüències de 0 i 1 iguals poden tenir efectes totalment diferents en dos processadors de models diferents, o resultar incomprensibles per a algun.



Com es pot apreciar, el processador realment és el centre neuràlgic i el cervell de l'ordinador, ja que una altra de les seves tasques és controlar l'execució ordenada de cada instrucció. Imatge de mark.sze

Un programa escrit en llenguatge de màquina és específic per a un tipus de processador concret. No es pot executar sobre cap altre processador, tret que siguin compatibles. Un processador concret només entén directament el llenguatge de màquina especificat pel seu fabricant.

Tot i que, com es pot veure, en realitat hi ha molts llenguatges de màquina diferents, s'usa aquesta terminologia per englobar-los a tots. Si es vol concretar més es pot dir "llenguatge de màquina del processador X".

Ara bé, estrictament parlant, si optéssiu per fer un programa en llenguatge de màquina, mai no ho faríeu generant directament fitxers amb tot de seqüències binàries. Només us heu d'imaginar l'aspecte d'un programa d'aquesta mena en format imprès, consistent en una enorme tirallonga de 0 i 1. Seria totalment incomprensible i pràcticament impossible d'analitzar. En realitat el que s'usa és un sistema auxiliar de mnemotècnics en el qual s'assigna a cada instrucció en binari un identificador en format de text llegible, més fàcilment comprensible per als humans. D'aquesta manera, és possible generar un programa a partir de fitxers en format text.

Aquest recull de mnemotècnics és el que es coneix com el **llenguatge assemblador**.

A títol il·lustratiu, la taula 1.2 mostra les diferències d'aspecte entre un llenguatge de màquina i assemblador equivalents per a un processador de model 6502. Sense entrar en més detalls, és important esmentar que tant en llenguatge de màquina com en assemblador cadascuna de les instruccions es correspon a una tasca molt simple sobre un dels seus components. Fer que l'ordinador faci tasques complexes implica haver de generar moltes instruccions en aquests llenguatges.

Taula 1.2. Equivalència entre un llenguatge assemblador i el seu llenguatge de màquina associat

Instrucció assemblador	Llenguatge de màquina equivalent
LDA #6	1010100100000110
CMP &3500	11001101000000000110101
LDA &70	1010010101110000
INX	11101111

1.3.2 ...mitjançant un llenguatge compilat

Per crear un programa el que farem és crear un arxiu i escriure-hi el seguit d'instruccions que volem que l'ordinador executi. Per començar n'hi haurà prou amb un editor de text simple.

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).

Un cop s'ha acabat d'escriure el programa, el conjunt de fitxers de text resultant on es troben les instruccions es diu que conté el **codi font**.

Aquest sistema de programar més còmode per als humans fa sorgir un problema, i és que els fitxers de codi font no contenen llenguatge de màquina i, per tant, resulten incomprensibles per al processador. No se li pot demanar que l'executi directament; això només és possible usant llenguatge de màquina. 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 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**.

Per al cas de l'assemblador el procés de compilació és molt senzill, ja que és una mera traducció immediata de cada mnemotècnic a la seqüència binària que li correspon. En principi, amb això ja hi hauria prou per poder fer qualsevol programa, però cenyir-se només a l'ús de llenguatge assemblador comporta certs avantatges i inconvenients que fan que en realitat no sigui usat normalment, només en casos molt concrets.

El codi objecte de les instruccions a la taula 1.2 té aquest aspecte:

```
1010100100000110 11
0011010000000000 11
0101101001010111 00
0011101111
```

Per la banda positiva, amb assemblador el programador té control absolut del maquinari de l'ordinador a nivell molt baix. Pràcticament es pot dir que controla cada senyal elèctric i els valors dels transistors dins dels xips. Això permet arribar a fer programes molt eficients en què l'ordinador fa exactament allò que li dieu sense cap mena d'ambigüitat. En contraposició, els programes en assemblador només funcionen per a un tipus de processador concret, no són portables. Si s'han de fer per a una altra arquitectura, normalment cal començar de zero. A més a més —i és el motiu de més pes per pensar-s'ho dues vegades si es vol usar aquest

llenguatge— crear un programa complex requereix un grau enorme d'expertesa sobre com funciona el maquinari del processador, i la llargària seria considerable. Això fa que siguin programes complicats d'entendre i que calgui dedicar molt de temps a fer-los.

Llenguatges compilats de nivell alt

Programes de nivell baix

Es considera que el codi de màquina i l'assemblador són els llenguatges de nivell més baix existents, ja que les seves instruccions depenen directament del tipus de processador.

Actualment, per generar la immensa majoria de programes s'utilitzen els anomenats llenguatges de nivell alt. Aquests ofereixen un conjunt d'instruccions que són fàcils d'entendre per a l'ésser humà i, per tant, posseeixen un grau d'abstracció més alt que el llenguatge assemblador (ja que no estan vinculats a un model de processador concret). Cadascuna de les instruccions es correspon a una ordre genèrica en què el més important és el seu aspecte funcional (què es vol fer), sense que importi com es materialitza això en el maquinari de l'ordinador ni molt menys en senyals elèctrics. En qualsevol cas, cal advertir que aquesta classificació no sempre és absoluta. Es pot dir que un llenguatge és de "nivell més alt o baix que un altre", segons el grau relatiu d'abstracció de les seves instruccions i la seva proximitat al funcionament intern del maquinari d'un ordinador.

El procés per generar un programa a partir d'un llenguatge de nivell alt és molt semblant al que cal seguir per fer-ho usant el llenguatge assemblador, ja que les instruccions també s'escriuen en format text dins de fitxers de codi font. L'avantatge addicional és que el format i la sintaxi ja no estan lligats al processador, i per tant, poden tenir el format que vulgui l'inventor del llenguatge sense que hagi de tenir en compte el maquinari dels ordinadors on s'executarà. Normalment, les instruccions i la sintaxi han estat triades per tal de facilitar la tasca de creació i comprensió del codi font del programa.

De fet, en els llenguatges de nivell alt més freqüents, entre els quals hi ha el que aprendreu a usar en aquest mòdul, les instruccions dins d'un programa s'escriuen com una seqüència de sentències.

> Una **sentència** és l'element mínim d'un llenguatge de programació, sovint identificat per una cadena de text especial, que serveix per descriure exactament una acció que el programa ha de fer.

Per tant, a partir d'ara s'usarà el terme *sentència* en lloc d'*instrucció* quan el text es refereixi a un llenguatge d'aquest tipus.

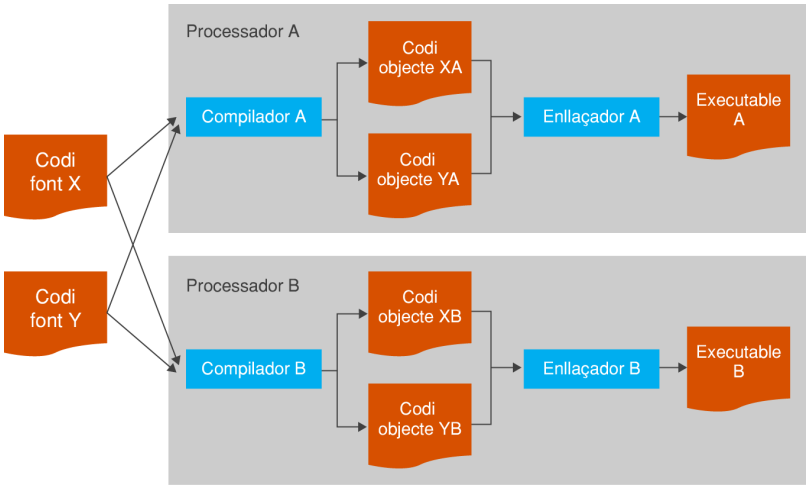
Un cop s'han acabat de generar els fitxers amb el seu codi font, aquests també s'han de compilar per tal de traduir-los a codi objecte. Ara bé, en aquest cas, el procés de traducció a codi objecte serà força més complicat que des d'assemblador. El compilador d'un llenguatge de nivell alt és un programa molt més complex. Pel que fa al procés de compilació, una conseqüència addicional del fet que el llenguatge no depengui directament del tipus de processador és que des d'un mateix codi font es pot generar codi objecte per a diferents processadors. Només cal disposar d'un compilador diferent per a cada tipus de processador que es vulgui suportar. Per tant, un mateix codi font original pot servir per generar programes

que funcionin amb diferents tipus de processador sense haver-lo de modificar cada vegada.

Ja que per a cada fitxer de codi font es genera un fitxer de codi objecte, després del procés de compilació hi ha un pas addicional anomenat *enllaçament* (*link*), en el qual aquests dos codis es combinen per generar un únic fitxer executable. Col·loquialment, quan us demanem que compileu un programa ja se sol donar per fet que també s'enllaçarà, si s'escau. Tot i així, formalment es consideren dues passes diferenciades.

La figura 1.2 mostra un esquema que serveix de resum del procés de generació del fitxer executable usant un llenguatge compilat.

Figura 1.2. Procés de compilació (i enllaçament) del codi font



Alguns exemples de llenguatges de nivell alt compilats molt populars són C o Pascal. Com s'ha vist, l'assemblador també és un llenguatge compilat, però de nivell baix.

Errors de compilació

El compilador és fonamental per generar un programa en un llenguatge compilat, ja sigui de nivell alt o baix. Per poder fer la seva feina de manera satisfactòria i generar codi objecte a partir del codi font cal que les instruccions segueixin perfectament la sintaxi del llenguatge triat. Per exemple, cal usar només les instruccions especificades en el llenguatge i fer-ho en el format adient. Si no és així, el compilador és incapaç d'entendre l'ordre que es vol donar a l'ordinador i no sap com traduir-la a llenguatge màquina.

> Quan el compilador detecta que una part del codi font no segueix les normes del llenguatge, el procés de compilació s'interromp i anuncia que hi ha un **error de compilació**.

Quan passa això, caldrà repassar el codi font i intentar esbrinar on hi ha l'errada. Normalment, el compilador dóna algun missatge sobre què considera que està malament.

Cal ser conscients que un programador pot arribar a dedicar una bona part del temps de la generació del programa a la resolució d'errors de compilació. Ara bé, que un programa compili correctament només vol dir que s'ha escrit d'acord amb les normes del llenguatge de programació, però no aporta cap garantia que sigui correcte, és a dir, que faci correctament la tasca per a la qual s'ha ideat.

Els llenguatges de programació i el llenguatge natural

Intentant fer un símil entre un llenguatge de programació i el llenguatge natural, si una persona que parla català és com un compilador, que és capaç d'entendre o traduir una frase sempre que se segueixin les normes d'aquesta llengua, sense una mica d'imaginació se li pot fer difícil entendre la frase: "*gelat kérem un comra*". Hi ha paraules en un ordre estrany, i a més a més n'hi ha d'altres que no pertanyen al català o que no simplement no existeixen...

D'altra banda, la frase "El gelat condueix un full de paper" pot ser gramaticalment correcta i no tenir cap error de sintaxi. Algú que parli català la pot entendre. Ara bé, és clar que alguna cosa no encaixa. En la comprensió del significat d'un llenguatge hi ha aspectes que van més enllà de la sintaxi, i els llenguatges de programació no en són excepció.

1.3.3 ...mitjançant un llenguatge interpretat

En contraposició del llenguatges compilats, tenim els llenguatges interpretats. En aquest cas, no es fa una distinció interna entre nivell alt i baix, ja que la immensa majoria de llenguatges interpretats són de nivell alt. El que interessa és entendre'n la idea general del funcionament i les diferències amb els compilats. Com en el cas dels llenguatges compilats, els programes també s'escriuen en fitxers de text que contenen codi font. La divergència sorgeix immediatament després d'acabar d'escriure'ls, en la manera com es genera un fitxer executable. El *quid* de la qüestió és que, precisament, ni es genera cap codi objecte ni cap fitxer executable. Es treballa directament amb el fitxer de codi font. Un cop aquest està escrit, el procés de creació del programa executable ha finalitzat.

Intèrpret

Alerta, un intèrpret no tradueix el codi font del programa a codi objecte i llavors l'executa. El que fa és executar diferents instruccions del seu propi codi segons cada instrucció llegida del codi font.

Imagineu un programa que accepta un seguit de dades que codifiquen unes instruccions, les va llegint una per una i les va processant de manera que actua d'una manera o d'una altra, és a dir, executa una part o una altra del seu propi codi objecte segons el tipus d'instrucció llegida. A fi de comptes, seria un programa que imita el comportament d'un processador, però a escala de programari. Doncs això és exactament un intèrpret.

Un llenguatge interpretat s'executa indirectament, mitjançant l'ajut d'un programa auxiliar anomenat **intèrpret**, que processa el codi font i en gestiona l'execució.

Com que un intèrpret és un programa executable està vinculat a un processador i a un sistema operatiu concrets.

Igual que en els llenguatges compilats, pot succeir que el programador hagi inclòs sense adonar-se'n alguna errada de sintaxi en les instruccions. En aquest cas, serà l'intèrpret qui mostrarà l'error i es negarà a executar el programa fins que hagi estat solucionat.

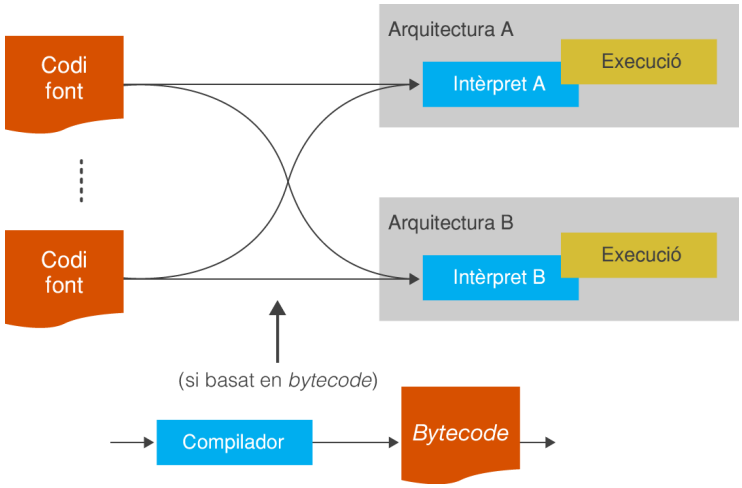
Col·loquialment, la generació de *bytecode* a partir del codi font s'anomena igualment *compilar*.

Alguns llenguatges interpretats usen una aproximació híbrida. El codi font es compila i com a resultat es genera un fitxer de dades binàries anomenades *bytecode*. Aquest *bytecode*, però, no és formalment codi objecte, ja que no és capaç d'entendre'l el maquinari de cap processador. Només un intèrpret el pot processar i executar. Simplement és una manera d'emmagatzemar més eficient i en menys espai, en format binari i no en text, les instruccions incloses al codi font. Aquest és el motiu pel qual, tot i necessitar un procés de compilació, aquests llenguatges no es consideren realment compilats i es continuen classificant com a interpretats.

Per les seves característiques, els llenguatges interpretats no requereixen un procés posterior d'enllaçament.

La figura 1.3 mostra un esquema del procés d'execució d'un programa en llenguatge interpretat. Noteu que en el cas d'un llenguatge amb *bytecode*, el que es proporciona a l'intèrpret són fitxers amb la versió del codi font prèviament compilat en *bytecode*, i no el codi font directament.

FIGURA 1.3. Procés d'interpretació del codi font



El terme "arquitectura" és refereix a la combinació de processador i sistema operatiu

Entre els llenguatges interpretats més coneguts trobem JavaScript, PHP o Perl. Molts són llenguatges de *script*, que permeten el control d'aplicacions dins un sistema operatiu, dur a terme processos per lots (*batch*) o generar dinàmicament contingut web. Entre els llenguatges interpretats basats en *bytecode*, Java és un dels més populars.

1.3.4 Entorns integrats d'edició

Un cop s'ha descrit el procés general per desenvolupar i arribar a executar un programa, es fa evident que cal tenir instal·lats i correctament configurats dos programes completament diferents i independents al vostre ordinador per desenvolupar-los: editor, per una banda, i compilador (incloent l'enllaçador) o intèrpret per l'altra, segons el tipus de llenguatge. Cada cop que vulgueu

modificar i provar el vostre programa haureu d'anar alternant execucions entre els dos. Realment, seria molt més còmode si tot plegat es pogués fer des d'un únic programa, que integrés els tres. Un editor avançat des del qual es pugui compilar, enllaçar si s'escau, i iniciar l'execució de codi font per comprovar si funciona.

Exemples d'IDE

Alguns exemples d'IDE són el Visual Studio, per als llenguatges C#, C++ i Visual Basic; el Netbeans, per als llenguatges Java i Ruby; el Dev-Pascal, per al llenguatge Pascal, o el Dev-C, per al llenguatge C.

Un **IDE** (*integrated development environment* o entorn integrat de desenvolupament) és una eina que integra tot el que cal per generar programes d'ordinador, de manera que la feina sigui molt més còmoda.

La utilització d'aquestes eines agilitza increïblement la feina del programador. A més a més, els IDE més moderns van més enllà d'integrar editor, compilador i enllaçador o intèrpret, i aporten altres característiques que fan encara més eficient la tasca de programar. Per exemple:

- Possibilitat de fer ressaltar amb codis de colors els diferents tipus d'instruccions o aspectes rellevants de la sintaxi del llenguatge suportat, per facilitar la comprensió del codi font.
- Accés a documentació i ajuda contextual sobre les instruccions i sintaxi dels llenguatges suportats.
- Detecció, i en alguns casos fins i tot correcció, automàtica d'errors de sintaxi en el codi, de manera similar a un processador de text. Així, no cal compilar per saber que el programa està malament.
- Suport simultani del desenvolupament de llenguatges de programació diferents.
- Un depurador, una eina molt útil que permet pausar l'execució del programa en qualsevol moment o fer-la instrucció per instrucció, de manera que permet analitzar com funciona el programa i detectar errades.
- En els més avançats, sistemes d'ajut per a la creació d'interfícies gràfiques.

En definitiva, usar un IDE per desenvolupar programes és una opció molt recomanable. Tot i així, cal tenir present que són programes més complexos que un simple editor de text i, com passaria amb qualsevol altre programa, cal dedicar un cert temps a familiaritzar-se amb aquests i amb les opcions de què disposen.

1.4 El vostre primer programa

Hi ha diferents llenguatges de programació, alguns realment molt diferents entre si. Abans de seguir endavant cal triar-ne un que serà l'usat per practicar tots els conceptes de programació bàsica que veureu d'ara en endavant. Un cop aprengueu a programar en un llenguatge, dominar altres llenguatges us serà molt fàcil, ja que molts dels conceptes bàsics, i fins i tot alguns aspectes de la sintaxi, es mantenen entre diferents llenguatges de nivell alt.

En aquest mòdul, el llenguatge amb el qual aprendrem a programar serà el **Java**.

1.4.1 Característiques rellevants del Java

Malauradament, no hi ha el llenguatge perfecte, sense cap inconvenient i que sigui ideal per crear qualsevol tipus de programa. Sempre cal arribar a un compromís entre avantatges i inconvenients. De fet, la tria mateixa de quin llenguatge cal usar pot arribar a condicionar enormement el procés de creació d'un programa, i no és assenyat usar sempre el mateix per resoldre qualsevol problema. En qualsevol cas, val la pena comentar els motius pels quals es considera interessant usar Java.

- **Popular**: Java va ser creat l'any 1995 per la firma Sun Microsystems, que el 2009 va ser comprada per l'empresa de bases de dades Oracle. El seu propòsit era oferir un llenguatge al menys lligat possible a l'arquitectura sobre la qual s'executa. Això el va convertir en els seus inicis en el mecanisme més versàtil existent per executar aplicacions sobre navegadors web (actualment, la tecnologia Flash ja li ha pres el relleu). Des de llavors, la seva popularitat ha anat en augment també com a a llenguatge per crear aplicacions d'escriptori, i actualment és un dels llenguatges més utilitzats en aquest camp. Això fa que la demanda de professionals que el dominin sigui molt alta i que tingui una gran acceptació i quantitat de documentació disponible.

Aplicació d'escriptori

Es considera una aplicació d'escriptori (*desktop application*) la que és totalment autocontinguda i que es pot executar en ordinadors de sobretaula o portàtils. Aquest terme s'usa en contraposició amb les aplicacions basades en Web.

- **De nivell alt amb compilador estricte**: Java és un llenguatge de nivell força alt, amb tots els avantatges que això implica. Addicionalment, el seu compilador és especialment estricte a l'hora de fer comprovacions sobre la sintaxi emprada i com es manipulen les dades que s'estan tractant en el programa. Si bé això de vegades pot semblar una mica empipador quan es produeixen certs errors de compilació, en realitat és un avantatge, ja que ensenya al programador a tenir més grau de control sobre el codi font que genera, de manera que sigui correcte.
- **Multiplataforma**: un dels factors decisius en la popularitat de Java és que els seus programes es poden executar en qualsevol plataforma sense que calgui tornar a compilar. Un cop el codi font s'ha compilat una vegada, el bytecode resultant pot ser portat a altres plataformes basades en altres tipus de processador i continuarà funcionant. Només cal disposar de l'intèrpret corresponent per a la nova plataforma. Això homogeneïtza enormement l'aprenentatge del llenguatge independentment de la plataforma que useu per estudiar-lo.
- **Orientat a objectes**: aquest és el nom d'una metodologia avançada, molt popular i útil, per dissenyar programes. Si bé els seus detalls estan totalment fora de l'abast d'aquest mòdul, n'hi ha prou de dir que és un mecanisme de

everywhere" (Compila un cop, executa a tot arreu).

nivell molt alt per apropar la manera com es fan els programes al mètode de pensament humà. En aquest curs no veurem un enfocament orientat a objectes de la programació, però el llenguatge Java us permetrà, quan conegueu aquesta tecnologia, treballar-hi.

També caldrà ser conscients d'algunes particularitats:

- **Orientat a objectes**: algunes parts de la sintaxi i la nomenclatura formal de Java estan íntimament vinculades a la metodologia de l'orientació a objectes i no se'n poden separar. Per tant, en alguns moments serà inevitable haver de fer front a aspectes lligats a l'orientació a objectes, encara que es tracti d'una metodologia avançada que no sigui objecte d'estudi en aquest curs. L'avantatge és que quan passeu a programar amb orientació a objectes ja coneixereu el llenguatge.
- **Interpretat amb *bytecode***: aquesta és una característica derivada del fet que sigui multiplataforma. En tractar-se d'un llenguatge interpretat, cal disposar de l'intèrpret correctament instal·lat i configurat a cada màquina on vulgueu executar el vostre programa. Això vol dir que hi ha un programa més que heu de configurar correctament al vostre sistema. Això també fa que l'execució dels programes en Java no segueixi el procés típic de qualsevol altra aplicació (per exemple, executar-lo des de línia d'ordres o fer doble clic a la interfície gràfica). No hi ha cap fitxer que es pugui identificar clarament com a executable.

1.4.2 Creació i execució de programes Java

Per ara es tractarà el cas de la creació de programes senzills que es componguin d'un únic fitxer de codi font.

Aquest apartat se centra a mostrar detalladament com es crea i s'executa un programa en llenguatge Java.

Atès que Java és un llenguatge interpretat, les eines que us calen són:

- Un editor de text simple qualsevol.
- Un compilador del llenguatge Java, per generar *bytecode*.
- Un intèrpret de Java, per poder executar els programes.

Disposar d'un editor de text és el menys problemàtic, ja que tots els sistemes operatius de propòsit general en solen tenir instal·lat algun per defecte. Ara bé, quan s'edita un fitxer de codi font, cal assignar-li una extensió específica d'acord amb el llenguatge de programació emprat, de manera que pugui ser fàcilment identificat com a tal.

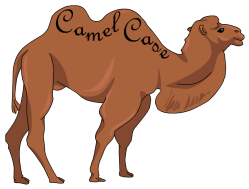
L'extensió dels fitxers de codi font en el Java és la `.java`.

Altres llenguatges tenen altres extensions. A títol d'exemple, en llenguatge C els fitxers tenen l'extensió `.c`, en assemblador `.asm`, en Perl `.pl`, etc. És important que en editar codi font des de qualsevol editor de text deseu el fitxer amb aquesta extensió i no la que us ofereixi per defecte (normalment, `.txt`).

Per què es diu notació de camell?

En el cas concret de Java, hi ha una convenció a l'hora de donar nom a un fitxer de codi font. Se sol usar *UpperCamelCase* (notació de camell amb majúscules). Aquesta correspon a usar només lletres consecutives sense accents (ni espais, subratllats o números), i en què la inicial de cada paraula usada sigui sempre en majúscula. Això no és estrictament imprescindible, però sí molt recomanable, ja que és l'estil de nomenclatura que segueixen tots els programadors de Java i la que trobareu en documentació, guies o altres programes. A més, en alguns sistemes, l'ús de caràcters especials, com els accents, pot dur a errors de compilació.

Alguns exemples de noms de fitxers de codi font acceptables són: `Prova.java`, `HolaMon.java`, `ElMeuPrograma.java`, etc.



Notació de camell

El compilador i l'intèrpret de Java són dos programes que haureu d'instal·lar al vostre ordinador, ja que normalment no estan preinstal·lats per defecte. N'hi ha diversos tipus, de diferents fabricants, però el més recomanable és instal·lar el que proporciona de manera gratuïta l'actual propietari de Java, l'empresa Oracle, a la seva pàgina de descàrregues. Hi ha diferents versions de l'entorn de treball amb Java, però el més habitual és treballar amb la Java SE (*Java standard edition*, edició estàndard de Java).

La pàgina de descàrregues d'Oracle és http://www.oracle.com/downloads. Hi ha versions del JDK per a diferents processadors i sistemes operatius.

El compilador de Java està inclòs dins de l'anomenat **JDK** (*Java development kit*, equip de desenvolupament del Java). Aquest proporciona un seguit d'executables via línia d'ordres que serveixen per fer diferents tasques amb codi font Java.

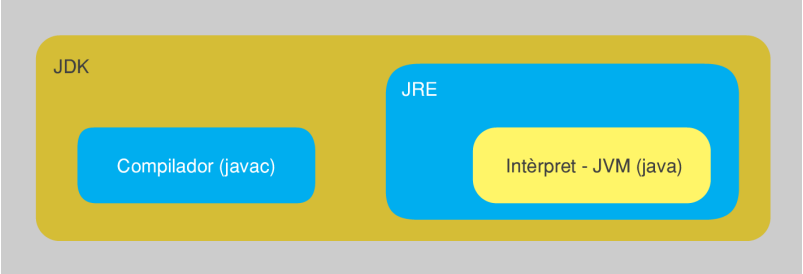
El programa que posa en marxa el compilador és l'executable anomenat `javac` (en un sistema Windows, `javac.exe`). Per tant, per al fitxer amb codi font Java anomenat `HolaMon.java` caldria obrir una línia d'instruccions i executar:

```
1 javac HolaMon.java
```

El fitxer amb *bytecode* resultant del procés de compilació s'anomena igual que el fitxer de codi font, però amb l'extensió `.class`.

Un cop disposem del fitxer amb *bytecode*, aquest només pot ser executat amb l'ajut de l'intèrpret de Java, conegut popularment com la **JVM** (*Java virtual machine*, màquina virtual de Java). Aquesta s'inclou dins el paquet anomenat **JRE** (*Java runtime environment*, entorn d'execució de Java). Alhora, el JRE ja va incorporat automàticament dins del JDK. La figura 1.4 mostra un esquema de la relació entre eines incloses als diferents paquets per al desenvolupament i execució de programes en Java.

FIGURA 1.4. Relació d'eines incloses a cada paquet de programari del Java



L'executable que posa en marxa l'intèrpret de Java és l'anomenat `java` (en un sistema Windows, `java.exe`). Un cop es disposa del fitxer de *bytecode* `HolaMon.class` es pot executar des de la línia d'ordres fent:

```
1 java HolaMon
```

Noteu que no s'especifica cap extensió. Ell sol dedueix que l'extensió ha de ser `.class`. Immediatament, el programa es posarà en marxa.

Disposeu d'un annex en què s'explica el funcionament de l'IDE que usareu al llarg del curs.

Tot i que aquest és l'entorn bàsic de desenvolupament de Java, afortunadament hi ha diversos IDE que suporten aquest llenguatge i concentren en un únic entorn aquest programari. Per treballar amb el Java hi ha diversos IDE amb diferents graus de popularitat: Netbeans, Eclipse, Jcreator, etc. Cadascun té les seves pròpies particularitats i graus de complexitat de configuració i ús. Cal dir, però, que actualment l'IDE oficial d'Oracle és el Netbeans.

Finalment, és important remarcar que en el cas que només vulgueu executar un programa que ja ha estat desenvolupat per algú altre, sigui mitjançant un IDE o no, l'única eina que us cal tenir instal·lada a l'ordinador per poder executar el fitxer `.class` resultant és la darrera versió del JRE. No cal instal·lar res més.

1.4.3 Hola, món!

Podeu trobar una llista del codi font del programa "Hello, world!" per a diferents llenguatges a http://www.scriptol.com/programming/hello-world.php

Dins de l'àmbit de la programació és tradició que el primer programa que s'escriu i s'executa quan s'inicia l'estudi d'un nou llenguatge sigui l'anomenat "Hola, món!" (originalment en anglès, "Hello, world!"). Aquesta tasca és un simple exercici de copiar el codi font del programa, per la qual cosa ni tan sols cal entendre encara la sintaxi del llenguatge. L'objectiu principal de fer aquest programa és veure que l'entorn de treball es troba correctament instal·lat i configurat, ja que per la seva senzillesa és difícil que doni problemes. A més a més, també us fa servei com a plantilla de l'estructura bàsica d'un programa en el llenguatge escollit i permet repassar-ne l'estructura i alguns dels elements bàsics.

El codi font per a la versió en Java és el següent:

```
1 //El programa "Hola, món!" en Java
```

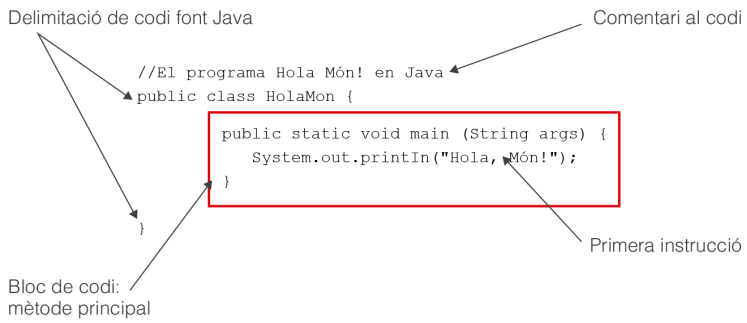
```
public class HolaMon {
  public static void main(String[] args) {
    System.out.println("Hola, món!");
  }
}
```

Repte 1: copieu el codi font d'aquest programa en un fitxer anomenat `HolaMon.java` i executeu-lo en el vostre entorn de treball. El resultat de l'execució hauria de ser que per pantalla es mostri la frase *Hola, món!*.

> El llenguatge Java és sensible a majúscules i minúscules. El significat dels termes varia segons com estan escrits. Per tant, és imprescindible que tant el codi font com el nom del fitxer s'escriguin exactament tal com es mostren.

La figura 1.5 us mostra a quina part característica d'un fitxer de codi font es correspon cada part del text del codi font.

Figura 1.5. Elements del codi font del programa "Hola, món!"



Importació d'extensions (biblioteques)

Una biblioteca és un conjunt d'extensions al conjunt d'instruccions disponibles quan genereu un programa. Per poder usar aquestes instruccions addicionals cal que, dins del codi font, com a preàmbul, en declareu la *importació*. En cas contrari, les extensions no estan disponibles per defecte dins del llenguatge. En el cas del programa "Hola, món!", en ser molt senzill, no és necessari, però en Java s'usaria la sintaxi:

```
import <nomBiblioteca>;
```

Indicador d'inici del codi font

El codi font on comença realment el programa en Java comença amb el text de declaració que es mostra a continuació, en què `<NomFitxer>` pot variar però sempre ha de correspondre exactament amb el nom del fitxer que el conté. Això és essencial. Altrament, el compilador ens donarà un error. El codi del programa s'escriurà tot seguit, sempre entre claus, `{ ... }`.

```
public class <NomFitxer> {
  ...
}
```

Aquest text declara que aquest fitxer és l'inici d'allò que en nomenclatura Java s'anomena una **classe**. Aquest és un terme estretament vinculat a l'orientació a objectes, però per ara l'usarem simplement per referir-nos a un fitxer que conté codi font de Java.

Per a desgràcia del programador principiant, moltes parts de la sintaxi de Java estan vinculades a l'orientació a objectes.

Comentaris al codi

Opcionalment, també es poden escriure comentaris dins del codi font. Es tracta de text que representa anotacions lliures al programa, però que el compilador no processa. Normalment, un comentari s'identifica per ser una línia de text lliure que comença per una combinació de caràcters especials. Els comentaris poden estar a qualsevol part del fitxer, ja que el compilador els ignora. No serveixen realment per a res amb vista a l'execució del programa, i només són d'utilitat per a qui està editant el fitxer de codi font.

> Comentar el codi font per explicar què fa cada part del programa, especialment en aquelles més complexes, és una tasca molt important que demostra si un programador és acurat o no.

En el programa `Hola, món` hi ha el comentari següent:

```
//El programa "Hola, món!" en Java
```

En llenguatge Java els comentaris s'escriuen o bé precedint-los amb dues barres, en el cas que tinguin una sola línia, o bé en el format següent si ocupen més d'una línia.

```
/**
 * Aquest és el programa "Hola, món!"
 * en el llenguatge de programació Java
**/
```

A partir d'ara, en tots els exemples de codi, els detalls sobre què fa cada part del programa es descriuran mitjançant un comentari abans de cada instrucció.

Indicador de la primera instrucció per executar

Per tal que l'ordinador sàpiga per on començar a executar instruccions, abans de res ha de saber quina és la primera de totes. Un cop localitzada, continuarà executant la resta de manera seqüencial, per ordre d'aparició al codi font. En alguns llenguatges això es fa implícitament, ja que la primera instrucció és directament la primera línia de text que apareix al codi font. En el cas del Java, hi ha un text que la indica clarament.

En el Java, el bloc d'instruccions en què s'engloba la primera instrucció del programa en la majoria de llenguatges de programació s'anomena el **mètode principal**.

Aquest mètode principal engloba totes les instruccions del programa dins d'un bloc d'instruccions, entre claus, `{...}`. Abans de les claus hi ha un seguit de text que s'ha d'escriure exactament tal com es mostra. Si no, l'intèrpret de Java serà incapaç de trobar-lo i d'iniciar correctament l'execució del programa. Concretament, dirà que "No troba el mètode pricipal" (*main method not found*).

```
1 public static void main (String[] args) {
2   ...
3 }
```

La primera instrucció és la primera que hi ha escrita tot just després de la clau oberta, `{`. La darrera instrucció és l'escrita immediatament abans de la clau tancada, `}`.

Blocs de codi o d'instruccions

Les instruccions o sentències del programa estan escrites una darrere de l'altra, normalment en línies separades per fer el codi més fàcil d'entendre. En alguns llenguatges, al final de cada línia cal un delimitador especial, que serveixi per indicar quan acaba una sentència i en comença una altra. Amb el salt de línia no n'hi ha prou. En el cas del Java, es tracta del punt i coma, `;`.

Els programes més simples, com els que veurem per ara, només disposen d'un únic bloc d'instruccions.

Les diferents instruccions se solen agrupar en **blocs d'instruccions**. L'inici i la fi de cada bloc diferent queden identificats en Java perquè les instruccions estan envoltades per claus, `{...}`.

Per tant, en aquest programa només hi ha una única, alhora primera i darrera, instrucció.

```
1 System.out.println("Hola, món!");
```

Tot i que encara no coneixeu cap de les instruccions del Java ni la seva sintaxi, possiblement podeu deduir de tota manera que aquesta serveix per ordenar a l'ordinador que mostri per pantalla el text escrit entre cometes. És a dir, que és una ordre sobre el component d'entrada/sortida (pantalla). Aquesta és una instrucció molt útil. En aquest codi també es pot apreciar que en el llenguatge Java les instruccions acaben amb un punt i coma, `;`.

Repte 2: modifiqueu el codi font perquè el programa escrigui per pantalla *Adéu, món!*.

2. Manipulació bàsica de dades

El propòsit principal de tot programa d'ordinador, en darrera instància, és processar dades de tota mena. Per assolir aquesta tasca, l'ordinador ha d'emmagatzemar les dades a la memòria, de manera que posteriorment el processador les pugui llegir i transformar d'acord amb els propòsits del programa. Per tant, un conjunt d'instruccions molt important dins de tot llenguatge de programació, i un bon lloc per on començar l'aprenentatge, és el que dóna a la memòria les ordres de desar-les o consultar-les. Un cop ja se saben gestionar dades a la memòria, és el moment de veure algunes de les transformacions bàsiques que permet fer el processador.

> El terme **dada** indica tota informació que fa servir l'ordinador en les execucions dels programes.

Tot i que de vegades s'usa el terme *dades*, en plural, com si fos quelcom en general, cal tenir en compte que dins d'un programa cada dada que es vol tractar és un element individual i independent. Per exemple, en un programa que suma dos nombres qualssevol hi ha tres dades amb les quals treballa: els dos operands inicials i el resultat final.

D'altra banda, fins al moment s'ha dit que l'ordinador s'encarrega de processar i transformar cadascuna d'aquestes dades individuals, però tampoc no s'ha especificat exactament com es representen o classifiquen els diferents tipus de dades dins del programa, o de quina manera es poden transformar. Ha arribat el moment de concretar una mica més tots aquests detalls.

2.1 Tipus de dades

Estrictament parlant, qualsevol informació es pot transformar en dades que pugui entendre i manipular un ordinador. Una dada individual dins del vostre programa pot ser un document de text, una imatge, un plànol d'un edifici, una cançó, etc. Només cal veure la immensa varietat de programes que hi ha avui dia per fer-se'n una idea. Ara bé, un dels punts importants és que dins d'un programa les dades es poden classificar dins de diferents categories, els *tipus de dades*.

> Un **tipus de dada** és la definició del conjunt de valors vàlids que poden prendre unes dades i el conjunt de transformacions que s'hi pot fer.

Per exemple, en el vostre dia a dia sovint tracteu amb dades que tenen com a característica comuna el fet que es poden representar mitjançant nombres: una

distància, una edat, un període de temps, etc. Es pot dir que una ciutat està a 8 km d'una altra, que algú té 30 anys o que han passat 15 dies des d'un esdeveniment. Per tant, els valors 8, 30 o 15 formarien part d'un mateix tipus de dada.

També com a exemple, en contraposició a les dades de tipus numèric, una altra informació que tracteu habitualment és text en paraules: el nom d'una mascota, d'un carrer, d'una institució, etc. En aquest cas, podeu estar parlant d'en "Shiro", del "Passeig de Gràcia", de l'"Institut Obert de Catalunya", etc. En aquest cas, les dades es representen amb símbols diferents i expressen unes altres idees. Per tant, formarien part d'un altre tipus de dada.

Una propietat important per veure si dues dades són de tipus diferents és si el conjunt de valors amb el qual es poden expressar també és diferent. En el dos exemples anteriors, la consideració que es tracta de tipus de dades diferents també es troba suportada per aquest fet. Per exemple, el conjunt de valors que pot prendre el nom d'un carrer o una persona no és qualsevol, ja que no es pot expressar exclusivament mitjançant valors numèrics.

Com que cada dada dins del vostre programa sempre ha de pertànyer a algun tipus, precisament part de la vostra tasca com a programadors és identificar quin és el tipus que s'apropa més a la informació que voleu representar i processar. Establir quin és el tipus d'una dada implica que s'estableixen un conjunt de condicions sobre aquella dada al llarg de tot el programa. Una de les més importants és l'efecte sobre la manera com aquesta dada es representarà, tant internament dins del maquinari de l'ordinador com a l'hora de representar-la en el codi font dels vostres programes. Recordeu que, per la seva naturalesa de sistema digital, totes les dades que hi ha dins d'un ordinador es codifiquen com a seqüències binàries. Quina seqüència de 0 i 1 cal usar per representar el valor de cada dada dependrà del tipus escollit.

Cada llenguatge de programació incorpora els seus tipus de dades propis i, a part, gairebé sempre ofereix mecanismes per definir-ne de nous partint de tipus de dades ja existents. Per tant, com a programadors, només heu de triar entre tots els que ofereix el llenguatge quin s'apropa més a la mena d'informació que voleu tractar. Ara bé, malauradament no es pot garantir que les instruccions bàsiques definides en la sintaxi d'un llenguatge de programació siguin capaces de tractar de manera directa qualsevol dada. Això només succeeix amb un conjunt limitat de tipus de dades molt simples, anomenats *tipus primitius*.

Els **tipus primitius de dades** són els que ja estan incorporats directament dins d'un llenguatge de programació, i són usats com a peces bàsiques per construir-ne de més complexos.

Els llenguatges de programació identifiquen cada tipus amb una paraula clau pròpia.

El suport a diferents tipus primitius pot variar també entre llenguatges. De totes maneres, n'hi ha quatre que es pot considerar que, d'una manera o d'una altra, tots els llenguatges els suporten. El motiu és que aquests tipus estan estretament lligats als tipus de seqüències binàries que normalment un ordinador pot processar i representar directament dins del seu maquinari. Es tracta dels nombres enters, els reals, els caràcters i els booleans.

Per començar, n'hi ha prou de veure quina mena de dades representen. Ara bé, per il·lustrar com podem usar dins del codi font d'un programa dades que pertanyen a tipus primitius, començareu treballant amb *literals*.

Un **literal** és un text usat per representar un valor fix dins del codi font d'un programa.

2.1.1 El tipus de dada booleà

El tipus de dada **booleà** representa un valor de tipus lògic per tal d'establir la certesa o falsedat d'un estat o afirmació.

La paraula clau per identificar aquest tipus de dada en Java és *boolean*.

- Exemples de literals d'una dada booleana: `true` (cert) o `false` (fals). No n'hi ha cap altre.
- Exemples de dades que se solen representar amb un booleà: interruptor encès o apagat, estar casat, tenir dret de vot, disposar de carnet de conduir B1, la contrasenya és correcta, etc.

Un literal de tipus booleà es representa simplement escrivint el text tal com s'ha descrit a dalt. En un IDE, normalment aquest text queda ressaltat en un color especial perquè quedi clar que s'ha escrit un literal de tipus booleà. Per exemple, el programa següent en Java mostra un seguit de literals d'aquest tipus per pantalla. Proveu-lo al vostre entorn de treball. Recordeu que ha d'estar dins d'un fitxer anomenat `LiteralsBoolea`, i que el nom no pot tenir accents.

```
1 public class LiteralsBoolea {
2   public static void main (String[] args) {
3     System.out.println(true);
4     System.out.println(false);
5   }
6 }
```

2.1.2 El tipus de dada enter

El tipus de dada **enter** representa un valor numèric, positiu o negatiu, sense cap decimal.

La paraula clau per identificar aquest tipus de dada en Java és *int*.

- Exemples de literals enters: 3, 0, -345, 138764, -345002, etc.
- Exemples de dades que se solen representar amb un enter: edat, dia del mes, any, nombre de fills, etc.

Un literal de tipus enter es representa simplement escrivint un nombre sense decimals. Per exemple, el programa següent en Java mostra un seguit de literals d'aquest tipus per pantalla. Proveu-lo al vostre entorn de treball. Recordeu que ha d'estar dins un fitxer anomenat `LiteralsEnter`.

```
1 public class LiteralsEnter {
2   public static void main (String[] args) {
3     System.out.println(3);
4     System.out.println(0);
5     System.out.println(-345);
6     System.out.println(138764);
7     System.out.println(-345002);
8   }
9 }
```

2.1.3 El tipus de dada real

La paraula clau per identificar aquest tipus de dada a Java és *double*.

El tipus de dada **real** representa un valor numèric, positiu o negatiu, amb decimals.

- Exemples de literals reals: 2.25, 4.0, -9653.3333, 100.0003, etc.
- Exemples de dades que se solen representar amb un real: un preu en euros, el rècord mundial dels 100 m llisos, la distància entre dues ciutats, etc.

Per referir-se a una dada de tipus real, aquesta inclou sempre els seus decimals amb un punt (.). En els exemples noteu el detall del valor 4.0. Els nombres reals representen valors numèrics amb decimals, però en la seva definició res no impedeix que el decimal sigui 0. Per tant, estrictament, el valor 4 i el valor 4.0 corresponen a tipus de dades diferents. El primer és un valor per a un tipus de dada enter i el segon per a un de real.

Per exemple, el programa següent en Java mostra un seguit de literals d'aquest tipus per pantalla. Proveu-lo al vostre entorn de treball. Recordeu que ha d'estar dins un fitxer anomenat `LiteralsReal`.

```
1 public class LiteralsReal {
2   public static void main (String[] args) {
3     System.out.println(2.25);
4     System.out.println(4.0);
5     System.out.println(-9653.3333);
6     System.out.println(100.0003);
7   }
8 }
```

Si voleu saber més sobre les diferències entre la representació d'enters i reals, podeu cercar informació sobre els formats "Complement a Dos" (*Two's Complement*) i "IEEE 754".

Una pregunta que potser us podeu plantejar ara és quin sentit té el tipus enter si amb el tipus real ja podem representar qualsevol nombre, amb decimals i tot. No és una mica redundant? Bé, d'entrada pot semblar que sí, però hi ha un motiu per diferenciar-los. Sense entrar en detalls molt tècnics, representar i fer operacions amb enters internament dins l'ordinador (en binari) és molt més senzill i ràpid que

no pas amb reals. A més a més, un enter requereix menys memòria. Evidentment, un programa no serà perceptiblement més lent o més ràpid pel simple fet de fer una operació entre dues dades de tipus real en lloc d'enter, però és un bon costum usar sempre el tipus de dada que s'adapti exactament a les vostres necessitats. Si una dada no té sentit que tingui decimals, com un número d'any o de mes, és millor usar el tipus enter.

2.1.4 El tipus de dada caràcter

El tipus de dada **caràcter** representa una unitat fonamental de text usada en qualsevol alfabet, un nombre o un signe de puntuació o exclamació.

La paraula clau per identificar aquest tipus de dada en Java és *char*.

- Exemples de literals caràcter: 'a', 'A', '4', '>', '?', 'Γ' (lletra grega gamma majúscula), etc.
- Exemples de dades que se solen representar amb un caràcter: cadascun dels símbols individuals d'un alfabet.

Per referir-se a una dada de tipus caràcter, aquesta es rodeja de cometes simples ('). Per tant, no és el mateix el caràcter '4' i que el valor enter 4, ja que pertanyen a tipus diferents. El primer l'usareu per referir-vos a una representació textual, mentre que el segon és el concepte pròpiament matemàtic. Aneu amb compte, ja que entre les cometes simples només hi pot haver un sol caràcter, o Java dirà que la sintaxi no és correcta.

Per exemple, el programa següent en Java mostra un seguit de literals d'aquest tipus per pantalla. Proveu-lo al vostre entorn de treball. Recordeu que ha d'estar dins un fitxer anomenat `LiteralsCaracter`, sense usar cap accent en el nom.

```
1  public class LiteralsCaracter {
2    public static void main (String[] args) {
3      System.out.println('a');
4      System.out.println('A');
5      System.out.println('4');
6      System.out.println('>');
7      System.out.println('?');
8      System.out.println('Γ');
9    }
10 }
```

El Java representa els seus caràcters internament usant la taula UNICODE.

Sistemes de representació dels caràcters

A l'hora de decidir com es representa internament un caràcter, s'utilitzen diferents taules de codis establerts, entre els quals destaquen els codis ASCII i UNICODE. El codi ASCII va ser un dels més estesos, sobretot per la gran proliferació d'ordinadors personals (anomenats en anglès *personal computer*, PC). El codi ASCII inicial representava 128 caràcters en lloc de 256. Quan es van necessitar caràcters especials per als diferents idiomes, es va ampliar amb 128 caràcters més, i es va constituir el codi ASCII estès.

Malauradament, el codi ASCII té una pega, i és que només permet representar alfabets occidentals, per la qual cosa els programes que l'usen per representar les seves dades

de tipus caràcter són incompatibles amb sistemes amb altres alfabets, com tots els asiàtics, el ciríl·lic, etc. Per aquest motiu, posteriorment es va crear la taula de codificació UNICODE, que permet codificar fins a 65.536 caràcters, però mantenint la compatibilitat amb la codificació ASCII. Això permet donar suport a qualsevol llengua actual, i fins i tot d'antigues, com els jeroglífics egipcis. Aquest sistema és el que actualment usen la majoria d'aplicacions modernes.

2.2 Transformació de les dades

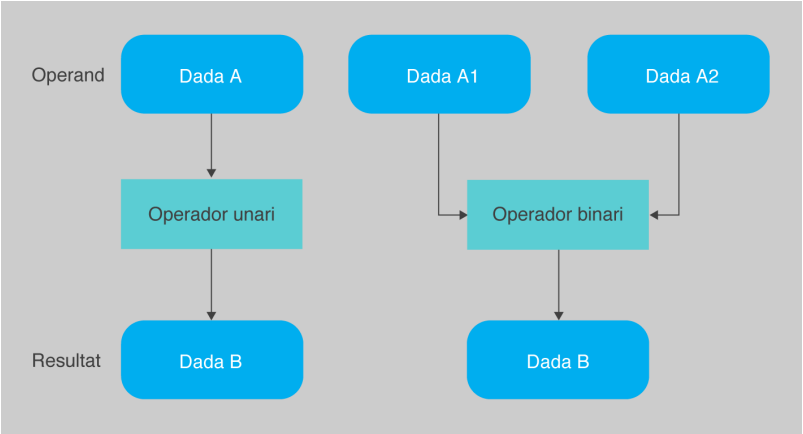
Dins d'un programa les dades s'acabaran representant usant algun dels quatre tipus primitius. Un cop heu escollit quin tipus usareu per representar una informació concreta, ja heu limitat el conjunt de valors que pot prendre. Ara bé, un altre punt molt important d'aquesta decisió és que també haureu fixat de quina manera és possible transformar-les. Per a cada tipus de dada, els seus valor només es pot transformar d'unes maneres molt concretes: mitjançant el que formalment es coneix com a *operacions*. Cada tipus de dada estableix el conjunt d'operacions admissibles sobre els valors que comprèn.

Un llenguatge de programació que només permet operacions entre dades del mateix tipus s'anomena **fortament tipat**. El Java ho és.

Normalment, només es poden fer operacions entre dades que pertanyin al mateix tipus. Com se sol dir, no es poden sumar pomes i taronges. Ara bé, sí que es pot donar el cas que una operació entre dos tipus de dades iguals doni com a resultat una dada d'un tipus diferent. Per exemple, no podreu sumar un nombre enter (el nombre 3, per exemple) amb un booleà (el valor `true`, per exemple). D'entrada, no tindria cap sentit.

Aplicar operacions sobre diverses dades us permet crear nous valors. Les operacions es divideixen en dos tipus: unàries o binàries, segons el nombre d'operands que usen (un o dos). La figura 2.1 presenta un esquema de com les operacions ens permeten crear noves dades.

FIGURA 2.1. Esquema d'aplicació d'operadors i operands per generar noves dades



El símbol com s'identifica cadascuna de les operacions dins d'un tipus de dada és el seu **operador**. Les dades sobre les quals s'aplica una operació són els seus **operands**.

2.2.1 Operacions entre booleans

Aquest tipus de dada deu el seu nom a l'àlgebra de Boole, que és el conjunt de normes i operacions que regeixen de quina manera es poden combinar els valors `true` i `false`.

> Les **taules de veritat** són una representació que ens permet esbrinar, amb claredat i fiabilitat, tots els resultats possibles d'una operació a partir de totes les combinacions dels valors possibles dels operands.

Les operacions bàsiques que es poden fer sobre dades d'aquest tipus són de dos tipus: lògiques i relacionals. Els operadors lògics que es poden aplicar són la negació (`!`), la conjunció (`&&`) i la disjunció (`||`). Per a aquest tipus d'operacions, una manera senzilla de plasmar el seu resultat en ser aplicats sobre dades de tipus booleà és usar una taula de veritat.

La taula 2.1 resumeix el resultat d'aplicar les diferents operacions lògiques. La negació té la particularitat que només s'aplica sobre una única dada.

Taula 2.1. Taula de veritat de les operacions booleanes

Operands		**Resultats de l'operació**		
A	B	A && B	A || B	! A
false	false	false	false	true
true	false	false	true	false
false	true	false	true	true
true	true	true	true	false

Si es vol fer un resum de la taula 2.1, es veu que:

- La negació dóna com a resultat el valor contrari.
- La conjunció és certa si totes dos operands són certs.
- La disjunció és certa si algun dels operands ho és.
- La disjunció exclusiva és certa només si un dels operands és cert, però no si ho són tots dos.

Vist amb un exemple més concret, les operacions lògiques segueixen el raonament següent. Suposeu que A i B són interruptors que poden estar encesos (`true`) o no (`false`). La negació és com commutar un interruptor. Si estava encès passa a estar apagat i viceversa. La conjunció és com preguntar "És cert que els dos interruptores estan encesos?". Només cal que un dels dos estigui apagat perquè la resposta sigui que no (`false`). La disjunció és com preguntar "És cert que algun dels dos interruptors està encès?". Si un dels dos, el que sigui, ho està, la resposta serà si (`true`).

Respecte als operadors relacionals, aquests es refereixen a les operacions igual (==) i diferent (!=) , aplicats tal com indica la taula 2.2.

TAULA 2.2. Taula de veritat de les operacions relacionals entre booleans

Operands		Resultats de l'operació	
A	B	A == B	A != B
false	false	true	false
true	false	false	true
false	true	false	true
true	true	true	false

En aquest cas, si es torna a l'exemple dels interruptors A i B, l'igual és equivalent a preguntar "És cert que els dos interruptors estan en el mateix estat?" (els dos apagats o els dos encesos). L'operació diferent seria el cas contrari: "És cert que un està un apagat i l'altre encès?".

El programa següent mostra, a mode d'exemple, el resultat de fer diverses operacions entre dades de tipus booleà. Us pot servir per veure més clarament els resultats expressats a les taules de veritat anteriors.

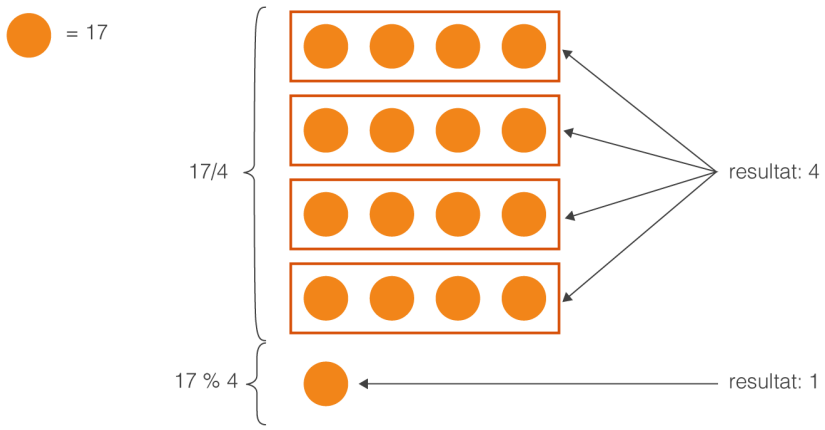
```
1  public class OperacionsBoolea {
2    public static void main (String[] args) {
3      //Operacions lògiques: el resultat és un booleà
4      System.out.println(!true);
5      System.out.println(true && true);
6      System.out.println(true && false);
7      System.out.println(true || false);
8      System.out.println(false || false);
9      //Operacions relacionals: el resultat també és un booleà
10     System.out.println(false == false);
11     System.out.println(false != true);
12   }
13 }
```

2.2.2 Operacions entre enters

Les operacions bàsiques que es poden fer sobre dades d'aquest tipus són de dos tipus: aritmètiques i relacionals.

Els operadors aritmètics generalment suportats són el canvi de signe (-, operador unari), la suma (+), resta (-, operador binari), multiplicació (*) i divisió (/). Qualsevol operació entre dues dades de tipus enter sempre dóna com a resultat una nova dada també de tipus enter. Ara bé, com que els nombres enters no disposen de decimals, cal tenir present que, en el cas de la divisió, el resultat es trunca i es perden tots els decimals. En alguns llenguatges de programació també hi ha una cinquena operació anomenada *mòdul* (%). El resultat d'aplicar-la sobre dos enters retorna la resta de l'operació de divisió, tal com esquematitza la figura 2.2.

Figura 2.2. Esquema del funcionament de les operacions de divisió i mòdul d'enters



La taula 2.3 mostra alguns exemples de resultats de l'operació divisió i mòdul en dades de tipus enter.

Taula 2.3. Exemples de divisió i mòdul entre enters

Operands		Resultat de les operacions	
A	B	A / B	A % B
17	4	4 , en lloc de 4.25	1
9	5	1 , en lloc de 1.8	4
23	10	2 , en lloc de 2.3	3

Els operadors relacionals inclouen l'igual (==), diferent (!=), major que (>), menor que (<), major o igual que (>=) i menor o igual que (<=). En aquest cas, el resultat de dur a terme aquesta operació és una dada de tipus booleà (cert/fals). La taula 2.4 en mostra alguns exemples.

Taula 2.4. Exemples d'operacions relacionals amb enters

Operands		Resultat de les operacions				
A	B	A == B	A > B	A < B	A >= B	A $\leq$ B
4	3	false	true	false	true	false
14	-2	false	true	false	true	false
-78	34	false	false	true	false	true
12	12	true	false	false	true	true

El programa següent mostra, a mode d'exemple, el resultat de fer diverses operacions entre dades de tipus enter.

```
1  public class OperacionsEnter {
2    public static void main (String[] args) {
3      //Operacions aritmètiques: el resultat és un enter
4      System.out.println(3 + 2);
5      System.out.println(4 - 10);
6      System.out.println(3 * 8);
7      System.out.println(10 / 3);
8      System.out.println(10 % 3);
9      //Operacions relacionals: el resultat és un booleà
```

```
    System.out.println(4 == 4);
    System.out.println(5 > 6);
    System.out.println(7 < 10);
  }
}
```

2.2.3 Operacions entre reals

Les operacions que es poden fer entre dades de tipus real són exactament les mateixes que entre enters, tant aritmètiques com relacionals. L'única excepció és l'operació mòdul, que deixa de tenir sentit, ja que ara la divisió sí que es fa amb càlcul de decimals. La taula 2.5 mostra com és la divisió entre dades de tipus real.

TAULA 2.5. Exemples de divisió entre reals

Operands		Resultat de l'operació
A	B	A / B
17.0	4.0	4.25
9.0	5.0	1.8
23.0	10.0	2.3

El programa següent mostra, a mode d'exemple, el resultat de fer diverses operacions entre dades de tipus real.

```
public class OperacionsReal {
  public static void main (String[] args) {
    //Operacions aritmètiques: el resultat és un enter
    System.out.println(8.5 + 3.2);
    System.out.println(5.66 - 3.1);
    System.out.println(3.1 * 8.4);
    System.out.println(17.0 / 4.0);
    //Operacions relacionals: el resultat és un booleà
    System.out.println(1.2 == 1.2 );
    System.out.println(-4.3 > -12.45);
    System.out.println(3.14 < 5.126);
  }
}
```

2.2.4 Operacions entre caràcters

Les dades d'aquest tipus només accepten operacions relacionals que els permetin comparar-los entre ells, de manera similar als enters o els reals: igual (==), diferent (!=), més gran que (>), menor que (<), més gran o igual que (>=) i menor o igual que (<=). En els casos de les operacions que inclouen "més gran que" o "menor que", se'n considera l'ordre en la taula emprada per representar-los a l'ordinador.

El programa següent mostra, a mode d'exemple, el resultat de fer diverses operacions entre dades de tipus caràcter.

```java
public class OperacionsCaracter {
  public static void main (String[] args) {
    //Operacions relacionals: el resultat és un booleà
    System.out.println('a' == 'a');
    System.out.println('a' != 'a');
    System.out.println('z' > 'w');
    System.out.println('k' < 'b');
  }
}
```

2.2.5 Construcció d'expressions

Com s'ha vist, donat un conjunt de dades d'un tipus concret, el mecanisme principal per transformar-les i obtenir-ne de noves és l'aplicació d'operacions entre diversos operands. Fins al moment, però, els exemples d'aplicació d'operacions s'han limitat a una única operació, amb el nombre d'operands corresponents segons si aquesta era unària o binària (un o dos). Ara bé, en molts casos, és útil poder aplicar d'una sola vegada un conjunt d'operacions diferents sobre un seguit de dades.

Una **expressió** és una combinació qualsevol d'operadors i operands.

Per construir una expressió, cal que aquesta sigui correcta en dos nivells, sintàcticament i semànticament, de manera que es respecti el significat de les dades usades com a operands i els seus operadors. En qualsevol cas, les expressions sempre s'escriuen en una sola línia, com qualsevol text llegible en català, ordenant els elements d'esquerra a dreta i de dalt a baix.

Des del punt de vista **sintàctic**, les normes bàsiques de construcció d'expressions usant literals són les següents:

1. Es considera que un literal sol és en si mateix una expressió.
2. Donada una expressió correcta E, també ho és escriure-la entre parèntesis: (E).
3. Donada una expressió correcta E i un operador unari qualsevol *op*, *op*E és una expressió correcta.
4. Donades dues expressions correctes E1 i E2 i un operador binari qualsevol *op*, E1 *op* E2 és una expressió correcta.

Aquestes normes són un cas general vàlid per representar qualsevol combinació de literals i operacions, sigui quina sigui la llargària. Exemples d'expressions que les segueixen, mostrades de manera que s'apliquen de manera acumulativa, són:

- `6`, per a la primera regla.

- `(6 + 5)`, per a la segona regla, en què E és `6 + 5`.
- `-4`, per a la tercera regla, en què *op* és `-` i E és `4`.
- `6 + 5`, per a la quarta regla, en què E1 és `6`, E2 és `5` i *op* és `+`.
- `(6 + 5) * -4`, per a la quarta regla, en què E1 és `(6 + 5)`, E2 és `-4` i *op* és `*`.

Des del vessant **semàntic**, les normes que sempre ha de complir una expressió són les següents:

1. Qualsevol operació sempre ha de ser entre dades del mateix tipus.
2. L'operació usada ha d'existir per al tipus de dada.

El primer punt és molt important i cal tenir cura en aplicar-lo. Per exemple, si partim de literals, escriure un `6` correspon al literal que representa el valor numèric "sis" dins del tipus de dada enter. El text `6.0` és el mateix però per als reals, i `'6'` (noteu les cometes simples) per als caràcters. Ara bé, tot i que representen el mateix concepte, no es poden fer operacions entre aquests tres literals, ja que pertanyen a tipus diferents. Tanmateix, noteu com amb petits detalls d'escriptura es pot establir a quin tipus de dada pertany cada literal dins del vostre codi font directament. De fet, tots els exemples de valors possibles donat un tipus de dada, mostrats a la secció anterior, són literals.

Les expressions següents són correctes sintàcticament, però no semànticament. Per tant, no es consideren expressions vàlides.

- `5.3 == '4'` : tot i que reals i caràcters disposen de l'operació igualtat, els dos literals pertanyen a tipus diferents (primera regla).
- `true + false`, l'operació suma no existeix per als booleans (segona regla).
- `-'g'`, l'operació canvi de signe no existeix per als caràcters (segona regla).
- `5 == false`, tot i que en booleans i enters existeix la igualtat, l'operació és entre tipus diferents (primera regla).
- `5 || 4.0`, es fa una operació entre tipus diferents i, a part, la disjunció no existeix en els enters ni en els reals (regla primera i segona).

2.2.6 Avaluació d'expressions

Un literal individual entre parèntesis és equivalent al mateix literal sense parèntesis.

Entenem com a *avaluar* una expressió anar aplicant tots els seus operadors sobre les diferents dades que la conformen fins arribar a un resultat final, que és la dada resultant. L'avaluació sempre es comença calculant les expressions que es troben entre parèntesis, en ordre de més intern a més extern. Un cop han desaparegut tots els parèntesis, llavors s'apliquen les operacions segons el seu ordre de precedència.

L'**ordre de precedència** d'un conjunt d'operadors és la regla usada per establir de manera no ambigua l'ordre com s'han de resoldre les operacions dins d'una expressió.

Les operacions amb ordre de precedència major s'avaluen abans que les d'ordre menor. Per a les operacions que s'han descrit fins ara, aquest ordre és el següent. En cas d'empat, es resol l'expressió ordenadament d'esquerra a dreta. La taula taula 2.6 mostra aquest ordre, de més prioritari (1) a menys (7).

Taula 2.6. Ordre de precedència dels operadors

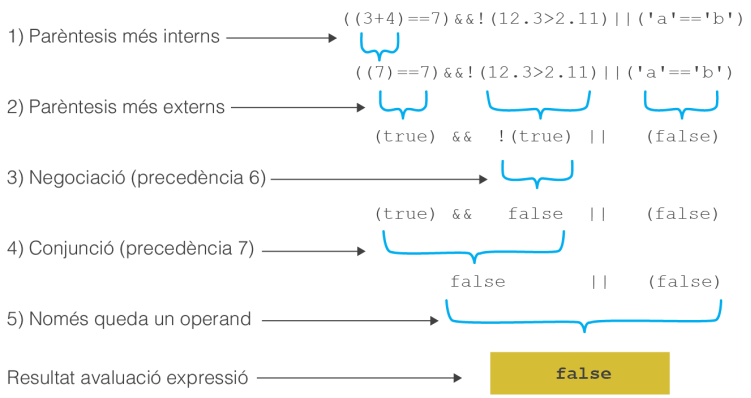
Ordre	Operació	Operador
1	Canvi de signe	-(unari)
2	Producte, divisió i mòdul	* / %
3	Suma i resta	+ -
4	Relacionals de comparació	> < <= >=
5	Relacionals d'igualtat	== !=
6	Negació	! (unari)
7	Conjunció	&&
8	Disjunció	||

Tot i haver-hi aquest ordre, és molt recomanable que totes les expressions basades en un operador binari sempre s'incloguin entre parèntesis quan s'han de combinar amb nous operadors per generar expressions complexes, amb vista a millorar la llegibilitat de l'expressió general.

A la figura 2.3 es mostra el procés d'avaluació d'una expressió de certa complexitat, que és la que teniu al programa següent:

```
1 //Comprova una expressió complexa
2 public class ProvaExpressio {
3   public static void main(String[] args) {
4       System.out.println(((3 + 4) == 7 )&&!(12.3 > 2.11)||('a' == 'b'));
5   }
6 }
```

Figura 2.3. Esquema d'avaluació d'una expressió d'exemple



Fixeu-vos atentament com en aquesta expressió apareixen literals de tipus de dades totalment diferents. Tot i així, sempre que s'aplica una operació binària, sempre és entre dades del mateix tipus. De fet, el resultat d'avaluar-la és una dada de tipus booleà, però en lloc de l'expressió original hi havia dades booleanes.

Repte 1: feu un programa que avaluï una expressió que contingui literals dels quatre tipus de dades (booleà, enter, real i caràcter) i la mostri per pantalla.

2.2.7 Desbordaments i errors de precisió

Un fet molt important que heu de tenir en compte quan avalueu expressions aritmètiques entre dades de tipus numèric (enters i reals), és que aquests tipus tenen una limitació en els valors que poden representar. En haver-hi infinits nombres, això vol dir que per representar-los tots dins de l'ordinador caldrien seqüències binàries d'infinita llargària, cosa impossible. Per tant, els llenguatges de programació només poden representar un rang concret en tots dos casos. Si mai se supera aquest rang, es diu que heu patit un desbordament (*overflow*) i el resultat de qualsevol expressió serà sempre incorrecte.

En el cas dels reals, a més d'haver-n'hi en nombre infinit, hi ha la particularitat que, donat un nombre real, aquest pot tenir infinits decimals. Això significa que tampoc no és possible representar qualsevol seqüència de decimals amb un nombre real. Per a certs valors en realitat es fan arrodoniments. La conseqüència directa d'aquest fet és que en fer càlculs amb dades reals poden aparèixer errors de precisió. Ho haureu de tenir en compte en programes de càlcul complexos, en què cada decimal és important.

Tenint en compte la circumstància que el rang de dades que es pot representar usant tipus numèrics està fitat, molts llenguatges de programació en realitat divideixen els tipus enters i reals en diferents categories, cadascuna definida amb una paraula clau diferent. Cadascuna d'aquestes categories es considera un nou tipus primitiu diferent. El tret distintiu per a cada cas és el rang de valors que pot assolir i la llargària de la seva representació interna en binari (nombre de bits).

Per al llenguatge Java, la taula 2.7 us mostra quins tipus primitius hi ha, preveient aquesta circumstància. Hi podeu apreciar quin és el rang acceptable per a cada cas i la seva mida. Més enllà d'aquests rangs, ja no és possible operar amb tipus primitius; calen solucions més complexes.

Els literals que usa Java per defecte, i els que s'han usat fins ara per descriure dades de tipus enter i real es corresponen als tipus enter simple (`int`) i real de doble precisió (`double`). Per tant, a partir d'ara, sempre que es parli de tipus de dades enters i reals en general, el text es referirà a aquests dos tipus primitius en concret.

TAULA 2.7. Rangs i paraules clau dels tipus primitius numèrics en Java

Tipus	Paraula clau Java	Mida (bits)	Rang
byte	`byte`	8	-128 a 127
enter curt	`short`	16	-32768 a 32767
enter simple	`int`	32	-2147483648 a 2147483648
enter llarg	`long`	64	-9,223,372,036,854,775,808 a 9,223,372,036,854,775,808
real de simple precisió	`float`	32	-3.40292347*10^38 a 3.40292347*10^38
real de doble precisió	`double`	64	-4.94065645841246544*10^24 a 1.79769313486231570*10^308

Si es vol indicar que un literal és d'un altre tipus diferent d'aquests dos, cal indicar-ho explícitament afegint al final el caràcter 'L', per escriure un `long`, o 'F', per un `float`. En Java no es pot explicitar que un literal és un enter curt. Per exemple:

- `5400000000L`
- `3.141592F`

En Java, un `float` pot representar fidelment fins a 6 o 7 decimals. Un `double` fins a uns 15.

Sempre que s'avalua una operació binària en què apareixen literals numèrics de tipus diferent, el tipus del resultat serà el mateix que l'operand amb més rang.

Per exemple, el resultat d'avaluar l'expressió `(2 + 5) * 40L` és un enter de tipus `long` (concretament, `280L`).

Repte 2: en el vostre entorn de treball, creeu el programa següent. Observeu què passa exactament. Llavors, intenteu arreglar el problema.

```
1 //Un programa que usa un enter mooooolt gran
2 public class TresMilMilions {
3   public static void main (String[] args) {
4     System.out.println(3000000000);
5   }
6 }
```

2.3 Gestió de dades a la memòria

Fins al moment s'ha treballat amb literals dins del codi font dels programes. Ara bé, aquesta només és la punta de l'iceberg a l'hora d'operar amb dades dins d'un programa. Si us limiteu a això, la funció de l'ordinador no va més enllà de ser una simple calculadora més complicada d'usar. De fet, els literals només són suficients en els casos que us cal representar dades amb un valor conegut en el moment d'escriure el programa i que mai variarà al llarg de l'execució, per a cap de les execucions.

Ara bé, en molts altres casos, el valor de les dades que es volen tractar dins d'un programa o bé és desconegut (per exemple, es tracta precisament del resultat que s'està intentant calcular o dependrà d'una resposta de l'usuari) o bé el valor anirà variant al llarg del programa (per exemple, el preu total per pagar en una botiga virtual quan encara no s'ha finalitzat la compra). Potser fins i tot en coneixeu el valor inicial (en una botiga virtual, el preu quan no s'ha comprat res encara segur que és zero), però no se'n pot predir l'evolució. Un altre cas que us podeu trobar és que vulgueu desar les dades resultants d'avaluar una expressió per usar-la més endavant, en un o diversos llocs dins del programa. En cap d'aquests casos no es poden usar literals i el que cal usar és una *variable* emmagatzemada dins de la memòria de l'ordinador.

> Una **variable** és una dada emmagatzemada a la memòria que pot veure modificat el seu valor en qualsevol moment durant l'execució del programa.

2.3.1 Com funciona la memòria de l'ordinador

Abans de veure com podem usar variables, val la pena fer un repàs dels aspectes més importants del funcionament de la memòria de l'ordinador des del punt de vista de com la usareu a l'hora de fer un programa. Això us oferirà una certa perspectiva sobre alguns dels aspectes més importants en l'ús de les variables.

Suposeu que esteu a l'escola primària i que un bon dia el professor us fa sortir a la pissarra i us diu que haureu de resoldre una suma de 4 nombres enters de 8 xifres. Primer de tot, el que fareu és cercar un tros de pissarra que estigui lliure per poder escriure-hi vosaltres, que no impliqui haver d'esborrar aquelles coses que el professor ha escrit abans. Concretament, us cal prou espai per poder escriure els 4 nombres que us dictarà el professor i per poder escriure la solució final. Si no us veiéssiu capaços de calcular-ho tot de memòria, també podríeu cercar espais per poder-hi fer càlculs auxiliars.

Després, a mesura que el professor us va dictant els nombres, vosaltres els aneu escrivint a la pissarra per no oblidar-los. Un cop els teniu tots apuntats, ja podeu procedir a fer les operacions pertinents i veure si us en sortiu o no. Mentre feu els càlculs, podeu anar escrivint, si us va bé, nous nombres que representen càlculs parcials, sempre que tingueu espai a la pissarra. També podeu esborrar o modificar en tot moment la informació que heu escrit a la pissarra. Finalment, amb l'ajuda de totes aquestes dades, obteniu el resultat final. Un cop dieu el resultat final, i el professor us diu si és correcte o no, esborreu la part de la pissarra on heu escrit per deixar espai als vostres companys i torneu al vostre lloc.

Conceptualment, la memòria de l'ordinador tal com la faria servir un programa no és gaire diferent de vosaltres i d'aquesta pissarra. És un espai disponible on els diferents programes poden desar lliurement dades per poder dur a terme la seva tasca. Ara bé, els dos punts més importants que aquest símil té en comú amb la memòria d'un ordinador, i que heu de tenir en compte, són:

- **La memòria és un espai compartit** entre tots els programes que es troben en execució, com el sistema operatiu. Per tant, abans de poder emmagatzemar cap dada, cal poder cercar un espai buit que no s'estigui usant.
- **Les dades emmagatzemades no són persistents**, només existeixen mentre el programa està resolent la tasca en curs. En acabar, s'esborren. En el cas d'un programa, les seves dades només estan presents a la memòria mentre aquest està en execució. Quan acaba, les dades desapareixen totalment. L'única manera d'evitar-ho és que, abans d'acabar l'execució, siguin traspassades a un medi persistent (per exemple, en el cas de la pissarra, a un full de paper, i en el cas de l'ordinador, a un fitxer).

Ara bé, per la natura de sistema digital d'un ordinador, la manera com la seva memòria organitza totes les dades contingudes no és exactament com una pissarra. També hi ha dos aspectes en què el símil divergeix i que cal tenir ben presents:

- Recordeu que **les dades es representen en format binari** quan es troben dins de l'ordinador. Un ordinador només pot representar internament la informació en zeros i uns, d'acord amb l'estat dels transistors dels seus xips.

Cada cel·la individual s'indexa amb un identificador numèric anomenat la seva *adreça*.

- De fet, **les dades s'organitzen en cel·les** dins de la memòria, d'una manera més aviat semblant a un full de càlcul. Cadascuna pot contenir 8 bits d'informació. Ara bé, una dada pot ocupar més d'una cel·la i el nombre exacte que n'ocupa depèn del seu tipus i del llenguatge usat.

2.3.2 Declaració de variables

De la descripció del funcionament de la memòria, el punt més important és que per poder disposar d'una variable dins del vostre programa, primer s'ha de cercar un espai lliure a la memòria, compost de diverses cel·les, que serà assignat a aquesta variable de manera exclusiva. Afortunadament, els llenguatges de programació de nivell alt ofereixen un sistema relativament senzill per fer-ho.

El llenguatge de programació ja s'encarrega internament de tots els aspectes de nivell baix vinculats a l'estructura real de la memòria.

Tota variable dins del codi font d'un programa ha d'haver estat **declarada** prèviament pel programador abans de poder fer-la servir.

Per declarar una variable dins el vostre programa, només cal especificar tres coses: el tipus, un identificador o etiqueta únic i un valor inicial. Tot i que la sintaxi per declarar una variable pot variar segons el llenguatge de programació, gairebé sempre cal definir aquestes tres coses.

L'identificador ens permet diferenciar les diferents variables que hi hagi dins del codi font del programa, el tipus delimita quins valors pot emmagatzemar i quines operacions són aplicables, i el valor inicial el contingut que hi haurà tan bon punt s'ha declarat. Evidentment, el valor inicial ha de ser una dada del

mateix tipus que la variable. Un cop completada correctament la declaració, immediatament després d'aquella línia de codi, és possible usar des de llavors en endavant l'identificador escollit per llegir el valor de les dades emmagatzemades o sobreescriure-les.

La importància d'inicialitzar les variables

No tots els llenguatges requereixen assignar un valor inicial a una variable. En aquests casos, simplement no es pot predir quin és el valor emmagatzemat a la memòria (la seqüència de zeros i uns en aquell espai). Pot ser qualsevol valor, per la qual cosa el resultat de qualsevol operació en què intervingui aquesta variable serà impredictible. Per aquest motiu, tot i que el compilador accepti la sintaxi com a correcta si no es fa, és un hàbit molt bo inicialitzar sempre les variables.

En cas de dubte, sempre podeu assignar el 0 com a valor inicial.

La sintaxi per declarar una variable en una línia de codi font Java és la següent. Recordeu que les paraules clau de Java per als quatre tipus primitius presentats són `boolean`, `int`, `double` i `char`:

```
paraulaClauTipus identificadorVariable = valorInicial;
```

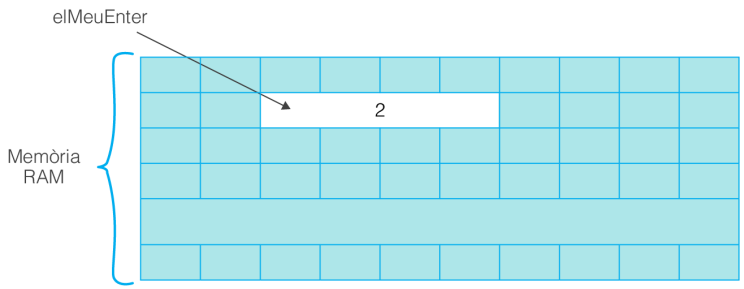
Estrictament, es considera que la part a l'esquerra del signe igual és la declaració pròpiament de la variable, mentre que la part dreta és la **inicialització**: l'especificació de quin és el valor inicial.

La manera més senzilla de veure-ho és mitjançant un exemple concret:

```
//El programa que només declara una variable de tipus enter
public class DeclararEnter {
  public static void main(String[] args) {
    int elMeuEnter = 2;
    //Ara hi ha una variable en memòria que conté el valor enter 2.
  }
}
```

Tot just després d'executar aquesta instrucció, la memòria quedaria com mostra la figura 2.4. En aquest cas, aquesta variable ocupa dues cel·les, ja que Java emmagatzema els enters a la memòria usant 32 bits. En qualsevol cas, aquest fet és totalment transparent per al programador.

FIGURA 2.4. Memòria de l'ordinador en declarar la variable "elMeuEnter"



També és possible declarar més d'una variable del mateix tipus de dada en una sola línia separant els identificadors i inicialitzacions amb comes. Per exemple:

```java
//El programa que declara de cop dues variables de tipus enter
public class DeclararEnter {
  public static void main(String[] args) {
    int elMeuEnter = 2, elMeuAltreEnter = 4;
    //Ara hi ha dues variables a la memòria
    //Contenen els valors enters 2 i 4, respectivament.
  }
}
```

2.3.3 Identificadors

Un identificador pren la forma d'un text arbitrari. Podeu triar el que més us agradi. De totes formes, és molt i molt recomanable que sempre useu algun text que us ajudi a entendre fàcilment què s'està emmagatzemant a cada variable, ja que això ajuda a entendre què fa el codi font del vostre programa. Quan un programa és llarg o fa molt que no s'ha editat, fins i tot pot ser problemàtic per al seu propi creador tornar a entendre per a què servia cada variable.

Col·loquialment, per referir-se a un identificador també se sol usar el terme *nom*. Exemple: "El nom d'una variable".

Noteu les diferències entre els dos codis següents. Tot i que el codi de declaració de variables és totalment equivalent (es declaren tres variables de tipus real), segur que en un us resulta més fàcil saber què fa el programa i quin paper té cada variable en el procés sense ni tan sols haver de conèixer la resta de les instruccions.

```java
//Programa A. Un codi poc entenedor.
public class DivideixISuma {
  public static void main(String[] args) {
    double x = 20.0;
    double y = 6.0;
    double z = 3.0;
    //Ara vindria la resta del codi
    ...
  }
}
```

```java
//Programa B. Un codi més entenedor.
public class DivideixISuma {
  public static void main(String[] args) {
    double dividend = 20.0;
    double divisor = 6.0;
    double sumarAlFinal = 3.0;
    //Ara vindria la resta del codi
    ...
  }
}
```

No tingueu mandra d'usar identificadors de variables entenedors, encara que siguin una mica més llargs. Evidentment, la clau de tot plegat sempre està en l'equilibri. Usar un identificador que ocupi cent lletres i no es pugui llegir en una sola línia de text tampoc no ajuda gaire.

Malauradament, no qualsevol identificador es considera vàlid. Cada llenguatge de programació imposa unes condicions sobre el format que considera admissible. Tot i que aquí s'explica el cas concret del llenguatge Java, no hi ha gaires variacions entre els diferents llenguatges. Un identificador:

- No pot contenir espais.
- No pot començar amb un nombre.
- Es desaconsella usar accents.
- No pot ser igual que alguna de les paraules paraules clau del llenguatge.

Una **paraula clau (o reservada)** d'un llenguatge de programació és aquella que té un significat especial dins la seva sintaxi i s'usa per compondre certes parts o instruccions en el codi font d'un programa.

El conjunt de paraules reservades en Java s'enumera tot seguit:

```
abstract, continue, for, new, switch, assert, default, goto,
package, synchronized, boolean, do, if, private, this, break,
double, implements, protected, throw, byte, else, import,
public, throws, case, enum, instanceof, return, transient,
catch, extends, int, short, try, char, final, interface,
static, void, class, finally, long, strictfp, volatile, const,
float, native, super, while
```

Fixeu-vos que en aquesta llista hi ha algunes paraules que ja heu vist, usades per declarar variables o especificar on comença el codi font o el mètode principal del programa: `boolean`, `int`, `double`, `char`, `public`, `class`, `void`, etc.

Convencions de nomenclatura

Tot i que teniu llibertat total per triar els identificadors de les variables, igual que passa amb els noms de les classes de Java, hi ha una convenció de codi per anomenar-los en Java. En aquest cas cal usar *lowerCamelCase* (notació de camell minúscula). Aquesta notació és com *UpperCamelCase*, però en aquest cas la primera paraula sempre va en minúscula i no en majúscula.

Exemples d'identificadors de variables que segueixen la convenció són: `divisor`, `resultatDivisio`, `elMeuEnter`, etc.

Finalment, recordeu que el Java és sensible a majúscules i minúscules, per la qual cosa `elMeuEnter` i `elmeuEnter` es consideren identificadors totalment diferents.

2.3.4 Ús de variables

La principal vàlua d'una variable com a peça fonamental dins d'un programa és la seva capacitat per emmagatzemar dades i recuperar-ne el valor en qualsevol moment al llarg del codi font del programa, un cop declarades. La recuperació de les dades emmagatzemades és el mecanisme més directe.

Quan una variable ha estat declarada, el seu identificador es pot usar exactament igual que un literal dins de qualsevol expressió. El valor que representarà serà el del valor emmagatzemat en memòria en aquell instant per a aquella variable.

En el cas del Java, el delimitador de fi de bloc és la clau `}`.

Ara bé, cal tenir present que en la majoria de llenguatges de programació la declaració d'una variable només té validesa des de la línia de codi on es declara fins a trobar el delimitador que marca el final del bloc de codi on s'ha declarat. Fora d'aquest rang d'instruccions del codi font, el seu *àmbit*, és com si no estigués declarada.

L'**àmbit d'una variable** és el context sota el qual es considera declarada.

Si s'intenta fer ús d'un identificador que no correspon a cap variable declarada (ja sigui perquè mai no s'ha declarat o perquè se'n fa ús fora del seu àmbit), en compilar el codi font es produirà un error. Cal dir que l'ús de variables fora del seu àmbit no és un problema que ara mateix us hagi d'amoïnar gaire, ja que els programes que s'estan tractant es componen d'un únic bloc que engloba totes les instruccions (el mètode principal).

El programa següent divideix dues dades de tipus real i després en suma una tercera usant variables en lloc de literals:

```
1  //Un programa que calcula una divisió i una suma, completat
2  public class DivideixISuma {
3    public static void main(String[] args) {
4      double dividend = 20.0;
5      double divisor = 6.0;
6      double sumarAlFinal = 3.0;
7      //Les variables es poden usar com a literals dins una expressió.
8      //Aquí és equivalent a fer (20.0/6.0) + 3.0
9      System.out.println((dividend/divisor) + sumarAlFinal);
10   }
11 }
```

En aquest exemple l'ús de les variables és correcte, ja que s'accedeix al seu identificador en una línia de codi que es troba entre la línia on s'han declarat i la clau de final de bloc del mètode principal. L'ús s'ha fet dins el seu àmbit.

Ara bé, l'ús de variables no té gaire sentit quan per fer una operació és podrien usar literals directament i el resultat seria el mateix, com en l'exemple anterior. La vertadera utilitat consisteix a poder canviar el valor que hi ha emmagatzemat en memòria. El valor d'una variable es pot modificar en qualsevol moment dins d'un programa amb l'operador d'assignació (=):

```
1  identificadorVariable = expressió;
```

Aquesta sintaxi ja l'havíeu vista, ja que és pràcticament igual a la usada per establir el valor inicial en declarar la variable. De fet, una variable també es pot inicialitzar a partir d'una expressió, no cal usar un únic literal.

Si el tipus del resultat de l'expressió i el de la variable en què s'assigna no és el mateix hi haurà un error de compilació.

En fer una assignació a una variable, el primer que succeeix és que s'avalua

l'expressió que hi ha a la banda dreta. El resultat obtingut llavors es converteix immediatament en el nou valor emmagatzemat dins de la variable. El valor anterior es perd per sempre, ja que queda sobreescrit. Ara bé, recordeu que l'operador d'assignació (=), com qualsevol altre operador binari, sempre requereix que els seus operands pertanyin al mateix tipus de dada. Per tant, només és correcte assignar expressions que avaluïn un tipus idèntic a l'utilitzat en declarar la variable.

Alerta. No és el mateix l'operador d'assignació (=) que el d'igualtat (==). Cal no confondre'ls.

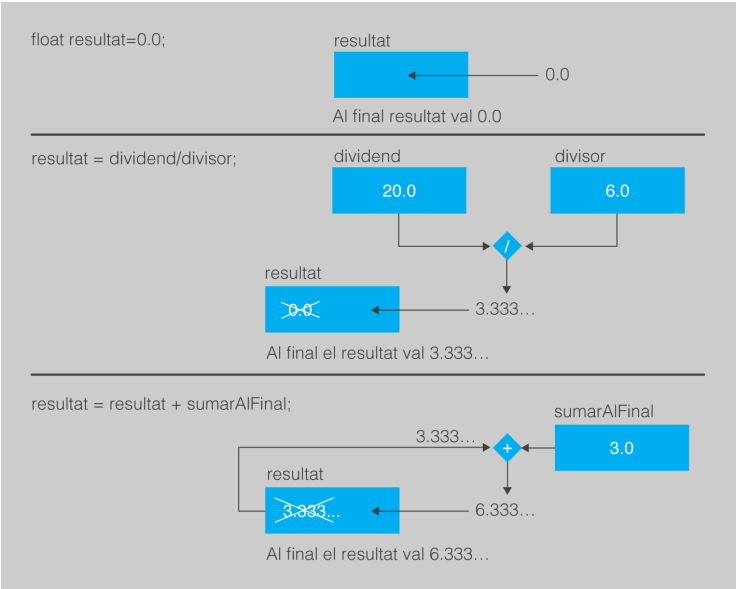
És molt important que sigueu conscients de l'ordre per resoldre una assignació. Primer cal avaluar l'expressió i després realment modificar el valor de la variable, ja que dins d'una expressió es pot usar l'identificador de la mateixa variable a la qual s'està fent l'assignació! Per tant, a la banda dreta s'usa el seu valor original, i un cop feta l'assignació, aquesta s'haurà modificat d'acord amb el resultat d'avaluar l'expressió.

La millor manera de veure el comportament d'una variable quan se'n modifica el valor és amb un exemple:

```
1  //Un programa que calcula una divisió i una suma, però a poc a poc
2  public class DivideixISumaDetallat {
3    public static void main(String[] args) {
4      double dividend = 20.0;
5      double divisor = 6.0;
6      double sumarAlFinal = 3.0;
7      double resultat = 0.0;
8      //El resultat serà una variable que s'anirà modificant.
9      resultat = dividend/divisor;
10     //Una expressió que usa la mateixa variable que es modifica!
11     resultat = resultat + sumarAlFinal;
12     System.out.println(resultat);
13   }
14 }
```

La figura 2.5 mostra un esquema de com es va modificant la variable `resultat` a cada pas.

Figura 2.5. Canvi del valor de la variable "resultat"



Per al cas de la conjunció i disjunció, els operadors d'aquest tipus són `&=` i `|=`.

Operació i assignació simultània

Modificar el valor d'una variable a partir d'aplicar una operació sobre el seu valor és quelcom que passa molt sovint dins un programa. Per aquest motiu, alguns llenguatges (com el Java) ofereixen un seguit de dreceres anomenades *operadors d'operació, aritmètica o lògica, i d'assignació*. En aquest es combina una única operació directament amb l'assignació d'acord amb el format següent:

Donat un operador binari *op*, `x op= y` equival a `x = x op y`.

Per tant, en l'exemple anterior també es podria calcular el resultat final fent:

```
resultat += sumarAlFinal;
```

Repte 3: feu un programa amb dues variables que, sense usar cap literal enlloc excepte per inicialitzar aquestes variables, vagi calculant i imprimint successivament els 5 primers valors de la taula de multiplicar del 4. Podeu usar operadors aritmètics i d'assignació, si voleu.

2.3.5 Constants

En definir variables, hi ha la possibilitat d'assignar un identificador que us permet donar una certa semàntica al valor que emmagatzema. Si trieu bé aquest identificador, pot ser molt més senzill entendre què fa el programa quan llegiu el codi font. Malauradament, en usar literals es perd una mica aquesta expressivitat, ja que només es disposa del valor tal qual, però no se'n pot saber exactament la funció sense inspeccionar el codi amb atenció. Per solucionar aquest detall, els llenguatges de programació permeten reemplaçar l'ús de literals per constants.

Un codi ben comentat també ajuda a entendre el codi font d'un programa.

> Una **constant** és un tipus especial de variable que té la particularitat que dins del codi del programa el seu valor només pot ser llegit, però mai modificat.

Hi ha diverses maneres de definir constants en el Java, però veureu la més popular. En aquest cas, les constants es defineixen fora del mètode principal, però dins del bloc que identifica un codi font Java. La sintaxi és idèntica a la definició de la variable, però abans de la paraula clau per al tipus de dada cal afegir les paraules reservades `private static final` , en aquest ordre. O sigui:

L'ús d'aquest terme és semblant al que se'n fa en matemàtiques o física, on s'usa per indicar certs valors universals i immutables: π (3,1415...), c (300.000 m/s, la velocitat de la llum), etc.

```
1 private static final paraulaClauTipus identificadorVariable = valorInicial;
```

Igual que amb les variables, el valor inicial pot ser tant un literal com una expressió. Ara bé, aquesta expressió no pot contenir variables. De totes maneres, el més normal és usar literals, per la qual cosa ens hi referirem al llarg de les explicacions.

Convencions de nomenclatura

Pel cas de les constants, les convencions de codi existents en el Java són diferents. En aquest sempre s'usen majúscules per a totes les lletres i, en cas d'anomenar-les amb una composició de diferents paraules, aquestes se separen amb un símbol de subratllat (*underscore*, _).

Exemples d'identificadors de constants que segueixen la convenció són: CONSTANT,

UNA_CONSTANT, LA_MEVA_CONSTANT, etc.

Compareu la llegibilitat del codi dels dos programes següents. En el primer sembla que s'apliqui una multiplicació arbitrària al final de codi, però en el segon queda ben clar per què cal fer-la.

```java
//Calcula el preu de comprar diverses coses
public class CalculaPreu {
  public static void main(String[] args) {
    double preuFinal = 0.0;
    //Va sumant preus de productes
    ...
    preuFinal = preuFinal + (preuFinal * 0.18);
    System.out.println(preuFinal);
  }
}
```

```java
//Calcula el preu de comprar diverses coses, i queda més clar com ho fa
public class CalculaPreu {
  private static final double IVA = 0.18;
  public static void main(String[] args) {
    double preuFinal = 0.0;
    //Va sumant preus de productes
    ...
    preuFinal = preuFinal + (preuFinal * IVA);
    System.out.println(preuFinal);
  }
}
```

Les constants són també especialment útils quan, a part de voler etiquetar certs literals de manera que el codi sigui més fàcil d'entendre, el literal en qüestió apareix diverses vegades dins el codi font. Suposeu que en una versió posterior del programa cal fer un canvi en el valor d'aquest literal. Per exemple, l'IVA que cal aplicar dins d'un programa de botiga virtual passa de 0,18% a 0,16%. Si no s'ha declarat com a constant, caldrà cercar aquest valor línia per línia i assegurar-se que s'ha usat per fer un càlcul d'IVA (potser s'ha usat aquest mateix literal, però amb una altra finalitat). Si l'heu declarat, n'hi ha prou de modificar només la declaració de la constant. El canvi es produeix automàticament en totes les expressions en què s'usa la constant amb aquest identificador.

La declaració de constants en lloc d'usar directament un literal quan aquest apareix en molts llocs dins el codi també ofereix un altre avantatge subtil. Suposeu que us equivoqueu en escriure alguna de les aparicions del literal. En lloc de 0,18% d'IVA escriviu 0,28% (ja que les tecles '1' i '2' estan juntes). En aquest cas, el programa compilarà perfectament, però no farà el que ha de fer correctament. El pitjor és que, d'entrada, no tindreu ni la menor idea de per què passa això i haureu de repassar tot el codi fins trobar el literal incorrecte. En canvi, amb una constant, el literal l'escriviu una única vegada i ja està. En canvi, si mai us equivoqueu en escriure l'identificador de la constant, el compilador considerarà que esteu fent ús d'un identificador no declarat prèviament i us donarà un error, i indicarà exactament el lloc dins el codi font on us heu equivocat. Arreglar-ho serà molt fàcil.

La millor manera de veure-ho és amb un exemple, que per fer-lo més interessant incorpora l'ús de variables. Tot seguit hi ha dues versions diferents del codi font d'un programa que fa exactament el mateix. Si us diuen que en tots dos hi ha una errada, en quin dels dos casos us seria més senzill identificar-la?

```java
//Què deu fer aquest programa? Compila perfectament, però té una errada!
public class Conversio {
  public static void main(String[] args) {
    double valor = 12.0;
    System.out.println(valor*1.3656);
    valor = 300.0;
    System.out.println(valor*1.3756);
    valor = 189.0;
    System.out.println(valor*1.3656);
  }
}
```

```java
//Potser ja està més clar què fa el programa. Per trobar l'errada, només cal
    compilar-lo.
public class ConversioError {
  public static final double CONVERSIO_EURO_A_DOLAR = 1.3656;
  public static void main(String[] args) {
    double valor = 12.0;
    System.out.println(valor*CONVERSIO_EURO_A_DOLAR);
    valor = 300.0;
    System.out.println(valor*CONEVRSIO_EURO_A_DOLAR);
    valor = 189.0;
    System.out.println(valor*CONVERSIO_EURO_A_DOLAR);
  }
}
```

Proveu de compilar els dos programes. De fet, per trobar l'errada en el segon ni tan sols cal una inspecció detallada del codi per part vostra, ja que apareixerà un error de compilació i us dirà exactament on es troba. Ara suposeu també que el valor de conversió ha canviat i l'euro val 1,4 dòlars. En quin dels dos programes és més senzill fer el canvi? Doncs imagineu-vos el mateix en un programa amb centenars o milers de línies de codi.

Una pregunta que us podeu fer és si no es pot assolir exactament el mateix simplement usant una variable, i no modificar-ne mai el valor dins el programa. És una bona pregunta, i la resposta és que a l'efecte funcional seria el mateix. Ara bé, definir constants té dos avantatges, però per entendre-ho cal tenir una mica d'idea de com funciona un compilador. El primer avantatge és que el compilador controla explícitament que cap constant declarada és modificada en alguna instrucció del codi font programa. Si això succeeix, considera que el codi font és erroni i genera un error de compilació. Això pot ser molt útil per detectar errors i garantir que les vostres constants realment mai no veuen modificat el seu valor al llarg de l'execució del programa a causa d'alguna distracció. El segon és que, en alguns llenguatges, abans de processar el codi font, el compilador reemplaça totes les aparicions de l'identificador de cada constant pel literal que se li ha assignat com a valor. Per tant, al contrari que les variables, les constants no ocupen espai realment a la memòria mentre el programa s'executa.

Per tant, és un bon costum cenyir-se a l'ús de constants quan s'està tractant amb dades generals que mai no canvien de valor.

Repte 4: feu dos programes, un que mostri per pantalla la taula de multiplicar del 3, i un altre, la del 5. Els dos han de ser exactament iguals, lletra per lletra, excepte en un únic literal dins de tot el codi.

2.4 Conversions de tipus

Fins al moment s'ha dit que sempre que hi ha una assignació d'un valor, ja sigui en establir el valor d'una variable o el d'una constant, els tipus ha de ser exactament igual en totes dues bandes de l'assignació. Bé, en realitat això no és estrictament cert. Sota certes condicions és possible fer assignacions entre tipus de dades diferents si sobre el valor en qüestió es pot fer una conversió de tipus.

Aquest procés també es coneix com a *casting*.

> Una **conversió de tipus** és la transformació d'un tipus de dada en un altre de diferent.

Dins dels llenguatges de programació hi ha dos tipus diferents de conversions de tipus: les implícites i les explícites. El primer cas correspon a aquelles conversions fàcils de resoldre i que el llenguatge de programació és capaç de gestionar automàticament sense problemes. El segon cas és més complex i el força el programador, si és possible.

2.4.1 Conversió implícita

Aquest cas és el més simple i es fonamenta en el fet que hi ha certes compatibilitats entre tipus primitius de dades diferents. La millor manera d'il·lustrar-ho és amb el programa següent:

```
1 public class ConversioTipus {
2   public static void main (String[] args) {
3     //El literal "100" és un enter i "real" és un float. Són tipus diferents,
          no?
4     float real = 100;
5     //Però, "doble" i "real" són dos variables de tipus diferent, no?
6     double doble = real;
7     System.out.println(doble);
8   }
9 }
```

La sintaxi d'aquest programa no es correspon al que s'ha explicat fins ara. En canvi, si proveu de compilar-lo i executar-lo, funcionarà correctament. Això és perquè en aquest codi hi ha els dos casos de conversions implícites més habituals.

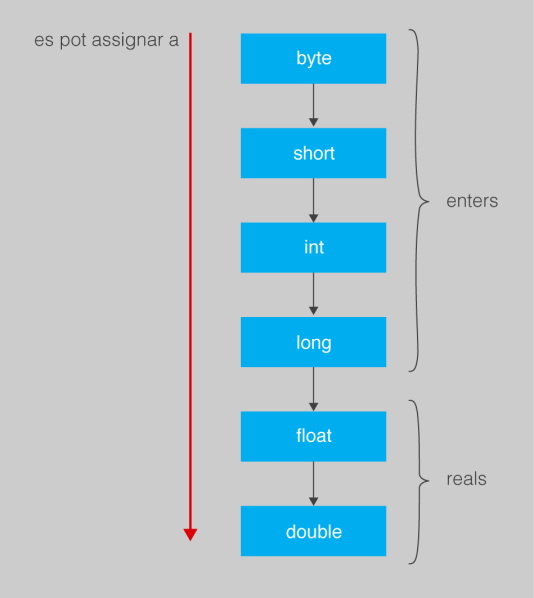
El primer cas es correspon al fet que tots els valors que pertanyen al tipus de dada enter es poden traduir molt fàcilment a real. Simplement cal considerar que els seus decimals són `.0` i ja està. Això és el que passa aquí. El literal `100` és un enter (concretament, un `int`), però s'assigna sobre una variable de tipus `float`, per la qual cosa en principi seria incorrecte. Caldria usar el literal que expressa aquest valor com un real de precisió simple, el `100.0F`. Tot i això, el compilador de Java considera que l'assignació és correcta, ja que és capaç de fer automàticament aquesta conversió des del tipus enter a real.

El segon cas està vinculat al rang del tipus de dada numèric. Si un valor és d'un

tipus primitiu numèric concret, aquest pot ser interpretat fàcilment com una dada dels tipus que accepta un rang més gran de valors. Per exemple, qualsevol valor de tipus `int` que us pugueu imaginar segur que també està dins dels valors acceptats entre els `long`. Igual passa entre els `float` i els `double`. El compilador de Java també s'adona d'això i per aquest motiu accepta una assignació d'un valor `float` (menys rang) a un `double` (més rang) sense problemes, ja que sap fer la conversió.

La figura 2.6 fa un resum de les relacions entre tipus numèrics. Les relacions són transitives, de manera que tota dada d'un tipus numèric pot ser assignada a variables o constants de qualsevol dels tipus posteriors en la relació.

FIGURA 2.6. Relacions de compatibilitat entre tipus numèrics



En qualsevol cas, al llarg del text sempre s'usaran els literals del tipus correcte en fer assignacions.

2.4.2 Conversió explícita

En qualsevol cas, on hi hagi assignacions entre dades de tipus diferent però el llenguatge triat sigui incapaç de fer una conversió implícita, el compilador sempre donarà error. Ja de per si, l'aparició d'aquest error sol significar que alguna cosa no està bé i que cal modificar el programa. Però hi ha casos en què és permissible eliminar aquest error especificant al codi que es vol forçar la conversió de les dades a un nou tipus de totes formes. Bàsicament, el que fa és obligar el compilador a transformar una dada d'un tipus en una de diferent "tan bé com pugui". Per indicar-li-ho, davant de l'expressió per convertir es posa el tipus de dada resultant que es vol obtenir entre parèntesis. La sintaxi és la següent:

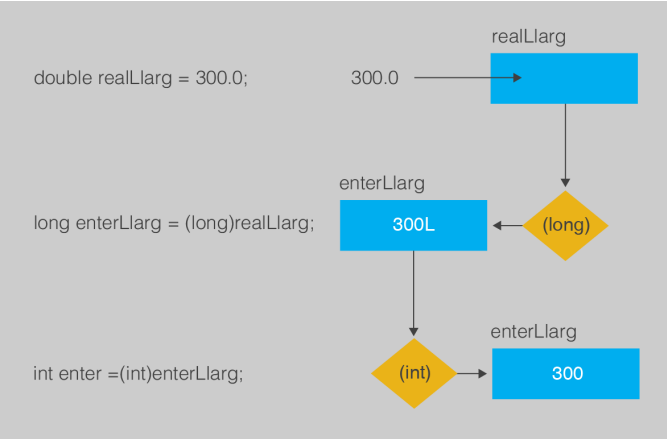
```
1 identificador = (paraulaClauTipus)expressió;
```

En fer això, el resultat de l'expressió s'intentarà convertir al tipus de dada especificat, sigui quin sigui el tipus en què avaluï originalment. El valor final obtingut dependrà molt de la situació sota la qual es fa la conversió, ja que és evident que convertir dades a tipus totalment diferents del seu origen no és una acció directa. En qualsevol cas, el tipus triat per a la conversió ha de ser compatible amb el de la variable en què s'assigna, com sempre.

> La conversió explícita s'ha d'usar amb molt de compte i tenint molt clar què s'està fent.

La situació més freqüent en què s'usa la conversió explícita és per invertir l'ordre establert en les relacions de la figura 2.7 (que mostra un esquema de què està succeint a cada línia del codi), ja que normalment un llenguatge de programació no permet fer assignacions en direcció oposada a les fletxes.

Figura 2.7. Conversió explícita entre variables



Això es deu al fet que, per exemple, un real o un valor de tipus `long` molt gran no es poden traduir normalment com un enter (`int`). En el primer cas no hi ha manera de representar els decimals i en el segon se surt fora del rang admissible. Ara bé, sí que hi ha casos en què aquesta assignació té sentit: quan els decimals del real són exactament `.0` o quan el valor del `long` està dins del rang acceptable per als enters. Llavors, usant la conversió explícita podeu evitar que el compilador doni un error i fer que intenti convertir de manera correcta els valors en enters. Si podeu garantir que el valor original compleix les condicions perquè la traducció sigui correcta, la conversió es completarà amb èxit. En cas contrari, el valor final assignat serà de ben segur erroni.

Tot seguit es mostra un exemple de conversions explícites entre tipus que se suposa que no són compatibles. Ara bé, com que el programador pot garantir que el valor real per convertir, `300.0`, és representable en diferents tipus d'enter, tot funciona correctament. Al final es veu per pantalla el valor `300` (o sigui, un literal de tipus enter). Però si esborreu les conversions explícites, `(long)` i `(int)`, el programa no compilarà.

```
1 public class ConversioExplicita {
2   public static void main (String[] args) {
3     double realLlarg = 300.0;
4     //Assignació incorrecta. Un real té decimals, no?
```

```
    long enterLlarg = (long)realLlarg;
    //Assignació incorrecta. Un enter llarg té un rang major que un enter, no?
    int enter = (int)enterLlarg;
    System.out.println(enter);
  }
}
```

Perquè aquest mecanisme funcioni és molt important que el programador estigui segur que el valor per convertir realment pertany al tipus de dada de destinació. En cas contrari, el resultat final no s'adaptarà fidelment a la dada original. Per exemple, en el cas de la conversió de qualsevol tipus real en enter es perdran tots els decimals (es truncarà el nombre), i en el cas d'assignar un valor fora del rang possible entre qualsevol tipus de dada obtindreu un desbordament.

Repte 5: experimenteu què passa si en el programa anterior s'inicialitza la variable `realLlarg` amb un valor amb diversos decimals. El programa continua compilant? Quin resultat dóna? Després proveu-ho assignant un valor superior al rang dels enters (per exemple, `3000000000.0`).

2.4.3 Conversió amb tipus no numèrics

En certs llenguatges de programació també es pot intentar fer conversions de tipus en què alguna o totes dues dades són de tipus no numèric. Llavors el que es demana és fer una operació a baix nivell amb la representació interna, en binari, de la dada. Per entendre què succeeix en aquesta situació, cal recordar que la combinació de zeros i uns com es representa una dada concreta depèn del seu tipus. Ara bé, també cal tenir present que una mateixa seqüència binària pot representar valors diferents per a tipus de dades diferents. Per exemple, el valor de tipus caràcter '`a`' es codifica internament amb la seqüència binària `0000000001100001`, però aquesta seqüència també és usada per codificar el valor de tipus enter '97'. El tipus de dada proporciona un context sobre el significat de les dades en binari quan es llegeix una dada dins de l'ordinador.

El llenguatge de l'ordinador i el llenguatge natural

Per exemplificar què vol dir que una mateixa representació d'una dada la poden compartir molts contextos diferents, es pot fer novament un símil amb el llenguatge natural. Suposem que trobeu un paper on hi ha escrita la paraula *clau*. En aquest símil, el paper representa la memòria, i *clau*, la dada emmagatzemada. Sense cap altra informació podeu ser capaços de saber-ne el significat? En realitat, amb aquestes quatre lletres un es pot referir a moltes coses: una clau per obrir una porta, un clau per penjar un quadre, la clau per descobrir un misteri (una pista), una clau d'arts marcials, etc. De fet, la paraula *clau* té més de quinze significats. El tipus de dada correspondria al context sota el qual s'ha d'interpretar aquesta paraula.

Fer una conversió amb dades de tipus no numèric indica a l'ordinador que ha d'agafar el valor binari trobat a la memòria tal qual i interpretar-lo com si fos un altre tipus de dada totalment diferent.

En la majoria de casos, aquestes s'han de fer explícitament, però en situacions

molt concretes el llenguatge fins i tot és capaç d'interpretar-les implícitament. A mode d'exemple, Java és capaç de dur a terme la conversió de caràcter a enter, i viceversa, de manera implícita. En altres llenguatges s'ha de fer explícitament o simplement no es pot fer.

```
1  public class ConversioCaracter {
2    public static void main (String[] args) {
3      int enter = 'a';
4      char caracter = 98;
5      //Escriu el valor en binari del literal 'a' interpretat com un enter: 97.
6      System.out.println(enter);
7      //Escriu el valor en binari del literal 98 interpretat com un caràcter: 'b
           '.
8      System.out.println(caracter);
9    }
10 }
```

En altres llenguatges, com C, es pot intentar amb pràcticament qualsevol tipus de dada.

Cal dir que l'ús d'aquest cas és molt marginal. De fet, tot i aquesta darrera opció d'operar a baix nivell, en cada llenguatge hi ha conversions explícites que simplement són impossibles. Si les intenteu fer, el compilador dirà que hi ha un error igualment. En Java per exemple, és el cas de convertir una dada del tipus booleà a qualsevol altre.

2.5 Visualització i entrada de les dades en Java

La veritat és que no té cap sentit que un programa dugui a terme el processament d'unes dades si aquesta tasca després no té cap repercussió en el sistema d'entrada/sortida. Sobretot tenint en compte que un cop el programa finalitza, tots els valors tractats s'esborren de la memòria. El mínim que pot fer és mostrar el resultat de la tasca per pantalla, de manera que l'usuari que l'ha posada en marxa sap quin ha estat el resultat de tot plegat. Tampoc no té sentit usar variables si en aquestes sempre s'emmagatzemen expressions fruit d'operacions entre literals. A efectes pràctics, es podrien usar ja directament literals sempre. Altres accions més complexes ja serien desar les dades en un fitxer de manera persistent, imprimir-les o que aquestes serveixin com a senyal de control sobre algun perifèric extern.

Tota dada mostrada per un perifèric està representada per dades dins de l'ordinador.

Hi ha moltes maneres d'entrar o mostrar les dades a l'usuari, només cal veure les interfícies gràfiques dels sistemes moderns, però aquí veureu la més simple: cadenes de text en pantalla o per teclat.

2.5.1 Instruccions de sortida de dades per pantalla

Tot i que fins al moment ja us heu trobat en els exemples d'apartats anteriors algunes instruccions que serveixen per mostrar coses per pantalla, ha arribat el moment d'inspeccionar-les amb una mica més de deteniment. Abans de començar, però, val la pena dir que cada llenguatge de programació té les seves pròpies instruccions per visualitzar dades per pantalla. Si bé la idea general pot tenir certes

similituds, la sintaxi pot ser totalment diferent. Per tant, cal tenir en compte que les descripcions se centren exclusivament en com funciona el llenguatge Java.

Principalment, Java proporciona dues instruccions per mostrar dades a la pantalla:

```
1 //Mostra a la pantalla el resultat d'avaluar l'expressió //x//.
2 System.out.print(x);
3 //Fa el mateix, però al final afegeix un salt de línia.
4 System.out.println(x);
```

Quan escriviu amb el teclat, el text es va mostrant en el mateix ordre com aneu cridant les instruccions de sortida per pantalla. No es pot demanar un posicionament absolut a qualsevol lloc de la pantalla. Per tant, cada instrucció començarà a escriure immediatament després d'on l'ha deixada la instrucció anterior.

Un dels aspectes més importants de representar dades per pantalla és que aquest perifèric només tracta amb text. O sigui, conjunts de caràcters en Java (`char`). Per tant, en el procés de sortida a la pantalla tota dada que es vol mostrar ha de patir una transformació des de la representació original fins al conjunt de caràcters que la representen en format text.

D'enter a text

Suposeu que tenim una variable de tipus enter en Java (`int`), en què hi ha emmagatzemat el valor 1309. Això internament es representa amb el valor binari: `0000010100010100`. Ara bé, per la pantalla el que s'haurà de mostrar són els 4 caràcters (`char`) '1', '3', '0', '9'. És a dir, s'ha d'ordenar al sistema d'entrada/sortida per pantalla que mostri els quatre valors de la representació en binari:

- '1': `00000000000000000110001`
- '3': `00000000000000000110011`
- '0': `00000000000000000110000`
- '9': `00000000000000000111001`

Per tant, si es vol escriure un enter, caldria un pas de transformació previ a representar-lo en diferents caràcters individuals. Pel que fa a aquest tipus d'instruccions, un dels grans avantatges de Java, com a llenguatge de nivell alt en relació amb altres llenguatges de nivell més baix, és que les dues instruccions bàsiques de què disposa poden acceptar qualsevol expressió que en ser avaluada resulti en qualsevol tipus primitiu, sigui quin sigui (enter, real, booleà o caràcter). Automàticament s'encarreguen elles mateixes de transformar el valor emmagatzemat en la seva representació en format text, i la mostren per pantalla.

Altres llenguatges ofereixen altres mecanismes, alguns no tan simples, per fer aquesta transformació.

A mode d'exemple, en fer:

```
1 //Mostra diferents expressions simples per pantalla.
2 public class MostrarExpressions {
3   public static void main(String[] args) {
4     //Mostra el text "3" per pantalla.
5     System.out.println(3);
6     //Mostra el text "a" per pantalla.
7     System.out.println('a');
8     //Mostra el text "3.1416" per pantalla.
```

```
9       double x = 3.1416;
10      System.out.println(x);
11      //Mostra el text "true" per pantalla.
12      boolean a = true;
13      boolean  b = false;
14      System.out.println((a||b)&&(true));
15    }
16  }
```

Es mostra per pantalla el següent:

```
1  3
2  a
3  3.1416
4  true
```

2.5.2 Tipus de dada avançats: cadenes de text

En molts casos, segurament no en tindreu prou només escrivint la representació textual d'un tipus de dada simple. Us agradaria escriure una frase sencera de l'estil "El resultat obtingut és...". És a dir, escriure blocs de text. Partint d'aquesta necessitat, si reflexioneu una mica, potser arribareu a la conclusió que el tipus de dada caràcter no és gaire útil, ja que per mostrar frases llargues, amb el que heu vist fins al moment, caldria fer-ho caràcter per caràcter, de manera molt molesta.

En realitat, el tipus de dada caràcter no se sol usar directament, sinó que els llenguatges de programació l'usen com a base per generar un nou tipus de dada més avançat que serveixi per processar text.

La paraula clau per identificar aquest tipus de dada en Java és `String`.

Una **cadena de text** (*String*) és un tipus de dada que permet representar qualsevol seqüència de caràcters de longitud arbitrària.

- Exemples de valors d'una cadena de text: "Hola, món!", "El resultat final és...", etc.
- Exemples de dades que se solen representar amb una cadena de text: qualsevol missatge de text per pantalla.

Per referir-nos a una dada de tipus cadena de text, l'escrivim entre cometes dobles ("). Aneu amb molt de compte, ja que no és el mateix 'a' que "a". La primera és un literal que representa una dada de tipus caràcter i la segona és de tipus cadena de text.

Les cadenes de text **no** són un tipus primitiu.

Cada llenguatge ofereix la seva manera de gestionar cadenes de text i representar-les per pantalla. Des de l'exemple del programa "Hola, món!", ja heu vist que les

instruccions de sortida bàsica de Java per pantalla també poden mostrar cadenes de text de Java sense problemes, exactament igual que es fa amb els tipus primitius.

```
System.out.println("Hola, món!");
```

Finalment, recordeu que, com a tipus de dada, també es poden declarar variables usant la paraula clau, igual que amb qualsevol altre tipus primitiu:

```
//Mostra el text "Hola, món!" per pantalla.
public class MostrarHolaMon {
  public static void main(String[] args) {
    String holaMon = "Hola, món!";
    System.out.println(holaMon);
  }
}
```

Operadors de les cadenes de text

En contraposició dels tipus primitius, no es pot generalitzar en parlar d'operacions de cadenes de text. Cada llenguatge en pot oferir de diferents (o cap). En el cas del Java, les cadenes de text accepten l'operador suma (+), de manera que es poden generar expressions de la mateixa manera que es pot fer amb els tipus primitius. En aquest cas, el resultat d'aplicar-lo en una operació entre cadenes de text és que aquestes es concatenen. Es genera una nova cadena de text formada per la unió de cadascun dels operands de manera consecutiva.

```
//Mostra el text "Hola, món!" per pantalla usant concatenació de cadenes de
    text.
public class HolaMonConcatenat {
  public static void main(String[] args) {
    String hola = "Hola,";
    String mon = "món";
    String exclamacio = "!";
    //Mostra el text "Hola, món!" per pantalla
    System.out.println(hola + mon + exclamacio);
  }
}
```

Noteu que les cadenes de text s'enganxen directament l'una darrere de l'altra, sense afegir cap espai.

De fet, per facilitar encara més la feina del programador, Java va una mica més enllà i permet aplicar aquest operador entre una cadena de text i qualsevol tipus primitiu. El resultat sempre és una nova cadena de text en què el tipus primitiu es transforma en la seva representació textual.

```
//Mostra el resultat d'una divisió simple entre reals.
public class Dividir{
  public static void main(String[] args) {
    double dividend = 18954.74;
    double divisor = 549.12;
    double resultatDivisio = dividend/divisor;
    String missatge = "El resultat obtingut és " + resultatDivisio + ".";
    System.out.println(missatge);
  }
}
```

Les dues darreres línies també es poden convertir en una de sola si s'usa directament l'expressió sobre la instrucció de sortida:

```
System.out.println("El resultat obtingut és " + resultatDivisio + ".");
```

Caràcters de control

En alguns llenguatges de programació, com en Java, hi ha un conjunt de literals dins dels caràcters que són especialment útils dins de les cadenes de text, ja que permeten representar símbols o accions amb un significat especial. Aquests s'anomenen *seqüències d'escapament* i sempre es componen d'una contrabarra (\) i un caràcter addicional. La taula 2.8 mostra una llista dels més habituals.

Taula 2.8. Seqüències d'escapament típiques

Seqüència d'escapament	Acció o símbol representat
\t	Una tabulació.
\n	Un salt de línia i retorn de carro.
\´	El caràcter "cometa simple".
\"	El caràcter "cometes dobles" (").
\\	El caràcter "contrabarra" (\).

Com es pot apreciar, per als dos primers casos cal usar una seqüència especial, ja que són accions que trobem en un teclat, però no són representables gràficament. Per exemple, el codi:

```
//Mostra text usant caràcters de control.
public class CaractersControl{
  public static void main(String[] args) {
    System.out.println("Linia 1\n\tLínia2\nLínia 3");
  }
}
```

mostraria per pantalla:

```
Línia 1
  Línia 2
Línia 3
```

En la resta de casos, la seva existència es deu a la manera com es representen els literals de caràcters o cadenes de text. Com que el literal d'un caràcter s'envolta amb dues cometes simples i el d'una cadena de text amb cometes dobles, és l'única manera de distingir dins el text si aquest símbol s'usa per delimitar un literal o com a text que en forma part. La contrabarra es representa d'aquesta manera per distingir-la respecte de quan s'usa, precisament, per especificar una seqüència d'escapament. Així doncs, el codi:

```
//Mostra una cadena de text que inclou cometes dobles.
public class HolaMonCometes{
  public static void main(String[] args) {
    System.out.println("Hola \"món\"!");
  }
}
```

mostraria per pantalla:

```
Hola "món"!
```

Repte 6: feu un programa que mostri en pantalla de manera tabulada la taula de veritat d'una expressió de disjunció entre dues variables booleanes.

2.5.3 Entrada simple de dades per teclat

Explicar l'entrada de dades per teclat quan tot just s'ha començat a aprendre a programar usant Java té un seguit de complicacions, ja que per entendre exactament la sintaxi de cada instrucció usada cal conèixer nombrosos conceptes que encara no s'han explicat. A més a més, per acabar-ho de complicar, també hi intervé l'aspecte de llenguatge orientat a objectes de Java. Per aquest motiu, de moment n'hi ha prou que simplement aprengueu les instruccions des d'un punt de vista purament pràctic i funcional, però sense haver de saber exactament què està passant o quina sintaxi concreta s'està usant. Sempre que us calgui, podeu usar com a plantilla el codi font d'exemple en aquest apartat.

Per llegir dades de teclat cal usar una biblioteca, un conjunt d'extensions a les instruccions disponibles per defecte en el llenguatge. Per usar alguna de les extensions disponibles en una biblioteca, cadascuna identificada amb un nom, primer cal importar-la. Llavors, el primer que es fa és inicialitzar l'extensió, assignant-li un identificador, i a partir de llavors ja la podeu usar dins del vostre codi font.

Recordeu que el codi font ha d'estar en un fitxer anomenat `Sumar.java`.

Escriviu i executeu l'exemple següent per veure com podeu entrar per teclat dos nombres enters i fer una operació entre ells.

```
//Aquesta línia fa que la biblioteca estigui disponible.
import java.util.Scanner;
//Un programa que llegeix un enter i el mostra per pantalla.
public class Sumar {
  public static void main (String[] args) {
    //S'inicialitza la biblioteca.
    Scanner lector = new Scanner(System.in);
    //Es posa un missatge de benvinguda.
    System.out.println("Anem a sumar dos nombres enters");
    //Es llegeix un valor enter per teclat.
    //S'espera que es pitgi la tecla de retorn.
    System.out.print("Escriu un nombre i pitja la tecla de retorn: ");
    int primerEnter = lector.nextInt();
    lector.nextLine();
    //Es torna a fer...
    System.out.print("Torna a fer-ho: ");
    int segonEnter = lector.nextInt();
    lector.nextLine();
    //Fem l'operació.
    int resultat = primerEnter + segonEnter;
    //Imprimeix el resultat per pantalla!
    //S'usa l'operador suma entre una cadena de text i un enter
    System.out.println("La suma dels dos valors és " + resultat + ".");
  }
}
```

En realitat, `lector` és un identificador intercanviable amb qualsevol altre.

Un cop l'extensió està inicialitzada (en aquest cas, només es fa un sol cop), el programa està preparat per llegir dades de diferents tipus des del teclat. De

moment, n'hi ha prou que aprengueu com podeu llegir dades d'una en una, usant diferents línies de text per entrar cada dada individual. Per fer-ho, cal alternar les instruccions de lectura enumerades a la taula 2.9, segons el tipus de dada que es vol llegir des del teclat, i la instrucció `lector.nextLine();`, tal com es veu a l'exemple.

TAULA 2.9. Instruccions per a la lectura de dades per teclat

Instrucció	Tipus de dada llegida
`lector.nextByte()`	`byte`
`lector.nextShort()`	`short`
`lector.nextInt()`	`int`
`lector.nextLong()`	`long`
`lector.nextFloat()`	`float`
`lector.nextDouble()`	`double`
`lector.nextBoolean()`	`boolean`
`lector.next()`	`String`

Per no haver d'entrar en detalls, podeu considerar que cadascuna de les instruccions de la taula es comporta exactament igual que una expressió que com a resultat avalua la dada introduïda pel teclat. La instrucció mateixa és la que s'encarrega de transformar el text escrit, una cadena de text, en una dada del tipus corresponent. Normalment, tota dada llegida serà assignada a alguna variable prèviament declarada. Si us resulta més còmode, es pot fer en una sola línia com en el codi d'exemple, assignant el valor llegit directament com a part de la inicialització de la variable.

Usant aquest procediment, si en una mateixa línia s'introdueix text amb més d'una dada, només es llegeix la primera de totes. La resta s'ignora.

Aquest mecanisme no permet llegir caràcters individuals. Recordeu, però, que des del punt de vista del tipus de dada, una cadena de text que conté un únic caràcter **no** és el mateix que un caràcter individual. Un altre aspecte important és que la manera com reconeix els valors reals, si interpreta els decimals separats amb un punt o una coma (per exemple, `9.5` o `9,5`), depèn de la configuració local del sistema. En les configuracions catalana i espanyola se separen amb una coma.

Si feu proves us adonareu que en aquest programa, si us equivoqueu i introduïu un valor diferent d'un enter, el programa s'atura immediatament i mostra un missatge d'error per pantalla. Això és normal, ja que la dada escrita per teclat ha d'encaixar amb el tipus de dada que llegeix la instrucció usada. També podeu comprovar que, igual que en avaluar expressions, intentar assignar una dada que excedeix el rang admissible per al tipus d'una variable resultarà en un desbordament. En el cas d'entrada pel teclat, tots aquests fets són especialment complicats de controlar, ja que mai no sabeu què escriurà realment l'usuari, expressament o per equivocació. De moment, podeu considerar que tot això no és cap problema. Per ara, simplement, procureu no entrar mai dades incorrectes pel teclat.

Aquest missatge indica que ha succeït un error d'execució (error de *run-time*).

Repte 7: feu un programa que mostri per pantalla la multiplicació de tres nombres reals entrats per teclat.

2.6 Solucions dels reptes proposats

Repte 1:

```
public class AvaluarExpressionsLiterals {
  public static void main (String[] args) {
    //booleà == ((enter > (enter + enter))&&(caràcter == caràcter))||(real <
        real)
    System.out.println(false == ((6 > (5 + 3))&&('a' == 'b'))||(4.4 < 10.54) );
  }
}
```

Repte 2:

```
public class TresMilMilions {
  public static void main (String[] args) {
    //Cal dir que el literal es correspon a un "long" (afegir una "l" al final)
    System.out.println(3000000000l);
  }
}
```

Repte 3:

```
public class IniciTaulaMultiplicarQuatre {
  public static void main (String[] args) {
    int a = 4;
    int b = 4;
    System.out.println(a);
    a = a + b;
    System.out.println(a);
    a = a + b;
    System.out.println(a);
    a = a + b;
    System.out.println(a);
    a = a + b;
    System.out.println(a);
  }
}
```

Repte 4:

```
public class TaulaTresICinc {
  //Per fer la del 5, canviar el 3 per un 5.
  public static final int TAULA = 3;
  public static void main (String[] args) {
    System.out.println(1*TAULA);
    System.out.println(2*TAULA);
    System.out.println(3*TAULA);
    System.out.println(4*TAULA);
    System.out.println(5*TAULA);
    System.out.println(6*TAULA);
    System.out.println(7*TAULA);
    System.out.println(8*TAULA);
    System.out.println(9*TAULA);
    System.out.println(10*TAULA);
  }
}
```

Repte 5:

```java
public class ConversioExplicitaDiversosDecimals {
  public static void main (String[] args) {
    double realLlarg = 3000000000.0;
    //Assignació incorrecta. Un real té decimals, no?
    long enterLlarg = (long)realLlarg;
    //Assignació incorrecta. Un enter llarg té un rang més gran que un enter,
        no?
    int enter = (int)enterLlarg;
    System.out.println(enter);
  }
}
```

Repte 6:

```java
public class TaulaVeritatDisjuncio {
  public static void main(String[] args) {
    System.out.println("A    \tB    \t(A || B)");
    System.out.println("false\tfalse\tfalse");
    System.out.println("true \tfalse\ttrue");
    System.out.println("false\ttrue \ttrue");
    System.out.println("true \ttrue \ttrue");
  }
}
```

Repte 7:

```java
import java.util.Scanner;
public class MultiplicarEntradaTeclat {
  public static void main(String[] args) {
    //S'inicialitza la biblioteca.
    Scanner lector = new Scanner(System.in);
    //Es posa un missatge de benvinguda.
    System.out.println("Anem a multiplicar tres nombres reals");
    //Es llegeix un valor enter per teclat.
    //S'espera que es pitgi la tecla de retorn.
    System.out.print("Escriu un nombre i pitja la tecla de retorn: ");
    double primerReal = lector.nextDouble();
    lector.nextLine();
    //Es torna a fer...
    System.out.print("Torna a fer-ho: ");
    double segonReal = lector.nextDouble();
    lector.nextLine();
    //Es torna a fer...
    System.out.print("I un altre cop: ");
    double tercerReal = lector.nextDouble();
    lector.nextLine();
    //Fem l'operació.
    double resultat = primerReal * segonReal * tercerReal;
    //Imprimeix el resultat per pantalla!
    System.out.println("La multiplicació dels valors és " + resultat + ".");
  }
}
```