

Tema 5

Introducció a Java

1. La plataforma Java
2. Introducció a la programació en Java
3. Dades
4. Operadors
5. Sortida de dades
6. Entrada de dades
7. Estructures de control
8. Tractament d'excepcions
9. Comentaris
10. Exercicis



1 La plataforma Java

1.1. Introducció

Java és un llenguatge de programació de propòsit general i, per tant, és adequat per a fer qualsevol tipus d'aplicació professional.

Però no és un llenguatge més, sinó que té una sèrie de característiques que el fan únic i és usat per molts fabricants per a desenvolupar aplicacions comercials de gran repercussió.

En este tema vorem les principals característiques de Java, com s'instal·la l'aplicació per a començar a treballar i quina és l'estructura bàsica d'un programa Java.



Què es pot programar amb Java?

No és cert que Java només servisca per a programar *applets* per a pàgines web ja que Java és un llenguatge de propòsit general:

- Aplicacions independents. Igual que qualsevol altre llenguatge de propòsit general.
- Applets. Són xicotetes aplicacions en Java incrustades en un document HTML i que, per tant, s'executen en l'entorn d'un navegador, en l'ordinador local.

Característiques de Java

Una de les característiques més importants és que els programes executables (creats pel compilador de Java) són **independents de l'arquitectura**. És a dir: s'executen en una gran varietat d'equips amb diferents microprocessadors i sistemes operatius. Per exemple, un programa en Java compilat pot ser executat directament en un PC amb Windows o bé en un Mac amb Linux, etc, cosa que no ocorre per exemple en un programa en C, que ha de ser compilat en cada arquitectura distinta.

Altres característiques:

- És gratuït
- Permet escriure *applets* (aplicacions incrustades en HTML).
- És fàcil d'aprendre i està ben estructurat.
- És "Orientat a Objectes".
- Permet executar tasques concurrents dins d'un mateix programa.

Què necessitem per a programar en Java?

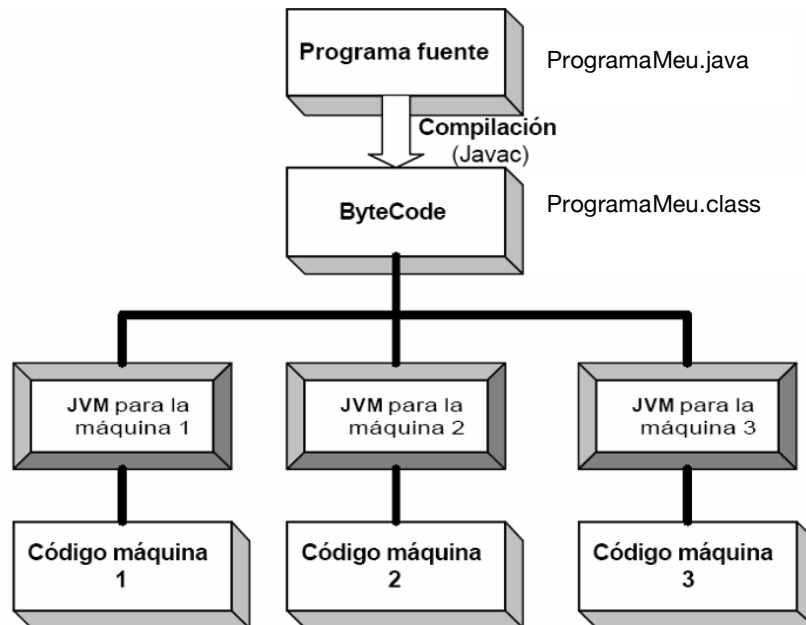
Per a treballar amb Java utilitzarem 3 elements:

- La **MVJ** (Màquina Virtual de Java) → Per a executar programes Java
- El **JDK** (Java Development Kit) → Per a programar en Java
- El **IDE** (Netbeans) → Facilita la tasca de programar

Anem a vore cadascun d'eixos elements.

1.2 La MVJ (Màquina Virtual Java)

Java és un llenguatge interpretat però necessita una compilació prèvia.



Una volta **compilat** el programa, es crea un fitxer que guarda el que s'anomena *bytecodes* o *j_code* (pseudocodi pràcticament al nivell de codi màquina). Per a executar-lo cal un **intèrpret**, la JVM (Java Virtual Machine) o **Màquina Virtual Java**.

D'esta forma, és possible compilar el programa en Linux i executar-lo en altra amb Windows usant la màquina virtual Java per a eixa versió de Windows. Esta JVM s'encarrega de llegir els *bytecodes* i traduir-los a instruccions executables directament en un determinat microprocessador, de forma eficient. Esta idea de màquina virtual fa que els programes siguin **independents de la plataforma** (Hw + S.O.) en la qual vaja a executar-se.

Encara que s'haja d'interpretar, **la velocitat d'execució no és lenta** ja que la interpretació es fa pràcticament al nivell de codi màquina. Per exemple, és molt més ràpid que un llenguatge interpretat com Visual Basic, encara que és més lent que si està escrit en un llenguatge compilat com C++.

Les JVM no ocupen molt d'espai en memòria (van ser dissenyades per a executar-se sobre xicotets electrodomèstics, com ara telèfons, televisors, etc).

Les **JVM** també s'anomenen **MVJ** (en anglés) o bé **JRE** (Java Runtime Environment: Entorn en temps d'execució de Java).

1.3 El JDK (Java Development Kit)

L'eina bàsica per a començar a desenvolupar aplicacions en Java és el JDK (Java Development Kit) o Kit de Desenvolupament de Java (també anomenat “plataforma”).

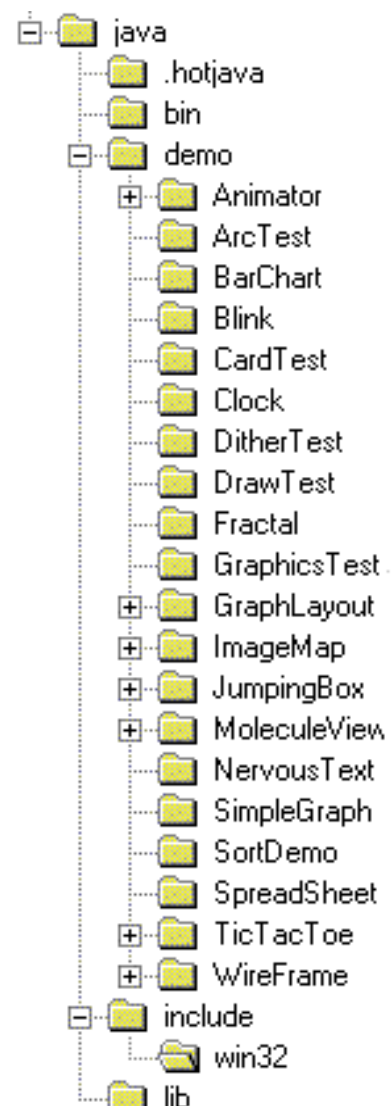
Entre altres coses, el JDK conté:

- El compilador: `javac.exe`
- L'interpret (la màquina virtual): `java.exe`
- El generador de documentació: `javadoc.exe`
- El depurador: `jdb.exe`
- El visualitzador d'applets: `appletviewer.exe`

El JDK és un fitxer executable que fa la instal·lació i crea tota l'estructura de directoris.

Alguns dels directoris són:

- **Bin:** fitxers executables: `javac`, `jdb`, `java`, `appletviewer...`
- **Demo:** directoris amb exemples de programes i applets en Java.
- **Include:** capçaleres per a utilitzar funcions escrites en C.
- **Lib:** llibreries de classes proporcionades pel llenguatge. Estan comprimides en el fitxer `classes.zip` (però no s'ha de descomprimir!).



1.4 L'IDE (Netbeans)

El JDK funciona a base de comandaments: hem de fer el programa en un editor de textos i després compilar-lo i executar-lo des de consola:

Editem el fitxer java, l'escrivim i el guardem:

```
$vi Hola.java
```

El compilem:

```
$javac Hola
```

L'executem:

```
$java Hola
```

```
Hola, món!
```

És a dir, el JDK no disposa d'un IDE. Un **IDE** o **Entorn de Desenvolupament Integrat** és una eina per a fer-nos més fàcil i agradable la tasca de desenvolupar programes mitjançant una interfície gràfica. És com, per exemple, el Visual Studio Code per a Python. Els IDE de Java més importants són Netbeans i Eclipse. Nosaltres utilitzarem el Netbeans.

Instal·lació de Netbeans

Cal instal·lar el JDK (que ja inclou la MVJ) i el propi IDE Netbeans. Ho podríem fer per separat (primer el JDK i després Netbeans) però ho farem tot alhora des d'ací:

<https://www.oracle.com/technetwork/es/java/javase/downloads/index.html>

O bé, en Ubuntu podem fer simplement: `sudo apt-get install netbeans`

Instal·lació de la documentació de les classes de Java en Netbeans

1r) Descarreguem la documentació:

<https://www.oracle.com/java/technologies/javase-downloads.html>

Documentation Download --> Abaixem un fitxer com: `jdk-..._doc-all.zip`

2n) Instal·laem la documentació en Netbeans:

Netbeans → Tools → Java Platforms → Javadoc → Add Zip Folder

Iniciem Netbeans

En C només treballàvem amb un **fitxer**. Ací crearem un **projecte** (serà un directori amb distints subdirectoris i fitxers). Per a crear un projecte en Netbeans:

File → New project → Java → Java application

The diagram shows the 'New Project' dialog in NetBeans with the following fields and callouts:

- Project name:** `nomAplicació`
 - Nom de l'aplicació**
 - ✓ Es crearà un directori on estarà tot allò relacionat amb ell.
- Create Main Class:** `nomaplicació . NomAplicació`
 - Nom del paquet**
 - ✓ Per defecte és el mateix que el nom de l'aplicació però en minúscules.
 - ✓ És opcional posar-lo. Si no el posem, tampoc posem el punt.
 - Nom del fitxer font**
 - ✓ Nom del fitxer que tindrà el programa font (amb el `.java`), que serà el mateix nom de la classe d'eixe fitxer.
 - ✓ En algunes versions de Netbeans (per baix de la 7) per defecte posa "Main", però sempre es pot canviar.
 - ✓ S'aconsella posar-lo en majúscula inicial perquè els noms de les classes van en majúscula.

Això crea esta estructura de directoris (i el fitxer corresponent):

The diagram shows the directory structure created by NetBeans:

```
...\NetBeansProjects\nomAplicació\src\nomaplicació\NomAplicació.java
```

Callouts explain the parts of the path:

- Nom de l'aplicació**: Points to `nomAplicació`.
- Nom del paquet. Si no hem indicat cap paquet, este directori no es crea.**: Points to `nomaplicació`.
- Nom del fitxer que té el programa font.**: Points to `NomAplicació.java`.

Els paquets s'utilitzen per a dividir el programa en parts. Cada paquet (o subpaquet) s'ubica en un directori (o subdirectori) diferent. Si quan creem el programa posem punts en el nom del paquet on està, es crearà una estructura de subdirectoris.

Alerta! No poseu accents (ni caràcters tipus ñ, ç...) en noms de projectes ni fitxers, ni els poseu en llocs on la ruta absoluta tinga alguna carpeta amb eixos caràcters, ja que, possiblement, donarà problemes de compilació.

Després, altres aplicacions podran importar paquets (posant el nom dels paquets amb els punts `'.'` o amb asterisc `'*'`).

1.5. Estructura d'un programa en Java

Anem a fer el nostre primer programa en Java. Només ha de fer una cosa: traure per pantalla el text “Hola, món”.

Creem un projecte nou en Netbeans: File → New project → Java application →

| |
|--------------------------------|
| Project name: hola |
| Create Main Class: Hola |

El fitxer que tindrà el nostre programa s'anomenarà **Hola.java** i estarà ací:

| |
|-----------------------------------------|
| ...\NetBeansProjects\hola\src\Hola.java |
|-----------------------------------------|

En eixe fitxer (Hola.java) escrivim el nostre programa:

La classe principal

Java és un llenguatge de programació d'objectes. Això implica que el programa estiga inclòs en el que s'anomena classe. Tots els programes han de

```
public class Hola {  
    /**  
     * El meu primer programa Java  
     */  
    public static void main (String[] args) {  
        // mostra "Hola, món!" per pantalla  
        System.out.println("Hola Món!");  
    }  
}
```

El mètode principal

Eixa classe ha de tindre almenys una funció o mètode, que sempre

El cos del programa

Dins d'este mètode és on posarem el propi codi del programa, és a dir, les instruccions necessàries per a

Estos conceptes (objectes, classes) els estudiarem més endavant. De moment, és suficient dir que si el programa que estic fent es diu *Hola.java*, s'ha de crear una classe pública amb el mateix nom *Hola*. Compte amb les majúscules, ja que Java també és *case-sensitive*.

A la primera línia podem veure la classe que es crea (i es tanca a l'última línia):

```
public class Hola {  
    ...  
}
```

Després podem observar una sèrie de comentaris que hi ha al programa. Igual que en C, poden ser d'una línia, amb *//*, o de diverses línies, amb */** i **/*. El cas del comentari que comença amb */*** també l'estudiarem més endavant.

Per a començar a executar-se qualsevol programa necessita una porta d'entrada. En C era *main()*. Si en C un programa pot estar format per diversos fitxers **.c*, només un d'ells podia contindre el mètode *main()*. En Java passa igual: podem tindre diversos fitxers **.java*, però només un d'ells contindrà la classe pública i, a més, contindrà el mètode *main()*. Este mètode sempre l'escriurem així:

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

A la línia 7 tenim l'equivalent al *printf()* de C:

```
System.out.println("Hola Món!");
```

I per què hem de posar *System.out.println(...)* i no només *println(...)*?

Perquè en Java, la invocació de funcions depèn de les llibreries on estan. En l'exemple es crida a la llibreria *System*. Dins d'ella està la llibreria *out* (que conté les funcions de sortida) i dins d'ella està la funció *println()*, que és qui definitivament imprimix el text per pantalla. Per a accedir a eixa funció hem de posar l'estructura de llibreries separades per un punt (.) .

Esta funció també fa un salt de línia. Si no el volem, usarem: `print()`.

Fora del mètode `main()` podem declarar variables globals i funcions (com en C).

Una volta hem escrit el nostre programa, necessitem compilar-lo (amb F9) i executar-lo (amb F6).

1.6. Algunes dreceres i coses pràctiques de Netbeans

| DRECERA DE TECLAT | ACCIÓ |
|---------------------------------------------|-----------------------------------------------------------------|
| sout + TAB | System.out.println(""); |
| soutv + TAB | System.out.println de la variable anterior |
| for + TAB | Estructura for |
| fori + TAB | Estructura for per a recórrer vector |
| forv + TAB | Estructura for per a recórrer ArrayList |
| sw + TAB | Estructura switch |
| wh + TAB | Estructura while |
| do + TAB | Estructura do...while |
| if + TAB | Estructura if |
| psvm + TAB | Estructura "public static void main" |
| Ctrl + Shift + f | Tabula bé tot (o text seleccionat prèviament) |
| Ctrl + r | Reanomena identificador sobre el que estem |
| Ctrl + ESPAI | Autocompletat de codi mentre escrivim |
| Ctrl + b (O bé: Ctrl + clic) | Ens porta a la definició de funció o classe sobre la que estem. |
| Alt + INTRO (o clic en bombeta esquerra) | Encerclar text seleccionat amb un for, while, try-catch... |

Altres coses pràctiques:

- Modificar la plantilla d'un fitxer Java (llevar comentaris per defecte, etc):

Tools → templates → Java → Java Class → Open in editor

- Comentar (o descomentar) un text seleccionat:

Icones



- Errors d'execució:

Potser apareixen molts errors però a vegades són causats per un únic error. Generalment haurem de veure el que apareix en primer lloc. Si fem clic damunt l'error (en blau) ens portarà a la línia de codi que ha provocat l'error.

2. Introducció a la programació en Java

En este tema introduïrem el llenguatge de programació Java i vorem com s'implementen en este llenguatge tots els aspectes que hem vist en Python.



Tot en Java són classes

Per a començar, quan programem en Java (i en C), ens apareix el concepte de *programació modular*. Això ens obliga encapsular tot el codi que fem dins de funcions, especialment dins d'una que s'anomena *main*. A més, com Java és un llenguatge *orientat a objectes*, tot el codi ha d'estar estructurat en classes: totes les funcions (també el *main*, clar) han d'estar dins d'una estructura anomenada **class**. Això ho treballarem al final de curs, però de moment hem de:

1. Crear el nostre programa en un fitxer que s'ha d'anomenar exactament igual que la classe on encapsulem el nostre codi, i extensió *.java*.
2. Crear el nostre codi dins d'un mètode *main*, que és la funció que s'executarà quan comença el programa.

El nom del fitxer haurà de dir-se *Primer.java*

```
public class Primer {  
    public static void main(String[] args) {  
        int num = 29; // per exemple  
        boolean esPrimer = true;  
        for (int i = 2; i <= num / 2; i++) {  
            if (num % i == 0) {  
                esPrimer = false;  
                break;  
            }  
        }  
        if (esPrimer) {  
            System.out.println(num + " és un número primer.");  
        } else {  
            System.out.println(num + " no és un número primer.");  
        }  
    }  
}
```

El *main* sempre es defineix així

Compilar i executar

Una volta hem escrit el nostre programa, en el fitxer que hem anomenat *Primer.java*, cal compilar-lo per a crear el fitxer de bytes, *Primer.class*, per a que puga executar-lo l'interpret de Java.

```
javac Primer.java
```

Això comprovarà si el programa té errors sintàctics. Si està bé, generarà el fitxer de bytes *Primer.class*, i ja el podrem executar amb:

```
java Primer
```

```
# ls
Primer.java
# javac Primer.java
# ls
Primer.java
Primer.class
# java Primer
29 és un número primer.
#
```

Ara bé: nosaltres no ho farem així, "a mà", sinó que utilitzarem un **IDE** (Entorn de Desenvolupament Integrat) per a editar, compilar, executar i depurar el nostre programa. Hi ha molts IDEs per a desenvolupar programes amb Java. Entre els més populars i gratuïts estan *Netbeans* i *Eclipse*. Comentarem algun aspecte de *Netbeans* però podeu utilitzar el que vullgau.

En **Netbeans**, conforme escrius el codi, ja va indicant-te alguns errors que troba (fins i tot no espera a que acabes d'escriure la instrucció). Per a provar el codi, simplement haurem de fer clic en la icona del *Play* (de color verd). Això ja s'encarrega de guardar els canvis del fitxer, compilar-lo i, si ha anat bé, executar-lo.

3. Dades

Java, a diferència de Python, és un llenguatge de tipificació forta. Això vol dir que haurem de:

- Declarar les variables: indicar el tipus que ha de ser una variable (i que ja no podrà canviar) abans d'usar-la.
- Indicar el tipus de dades que retorna una funció (ja ho vorem).
- Comprovar que els valors que s'assignen a una variable són de tipus compatibles. *Netbeans* ho fa automàticament.

3.1. Tipus de dades

Hi ha 8 tipus de dades bàsics en Java:

| Categoria | Nom | Longitud | Rang |
|-----------|----------------|----------|--------------------------|
| LÒGIC | boolean | 1 bit | false, true |
| CARÀCTER | char | 2 bytes | 0 a $2^{16}-1$ |
| ENTER | byte | 1 byte | -2^7 a 2^7-1 |
| | short | 2 bytes | -2^{15} a $2^{15}-1$ |
| | int | 4 bytes | -2^{31} a $2^{31}-1$ |
| | long | 8 bytes | -2^{63} a $2^{63}-1$ |
| REAL | float | 4 bytes | -2^{32} a $2^{32}-1$ |
| | double | 8 bytes | -2^{300} a $2^{300}-1$ |

Notes:

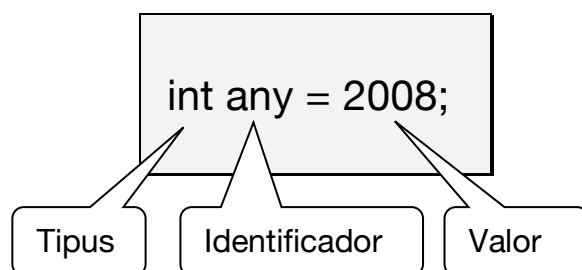
- Els valors lògics són *true* i *false* (minúscula). En Python eren: *True* i *False*.
- Les variables de tipus *char* només poden contindre 1 caràcter. A diferència de C, requereix 2 bytes (16 bits), ja que Java no emmagatzema els caràcters en format ASCII, sinó UNICODE. Per tant, en Java podem usar caràcters llatins, grecs, àrabs, ciríl·lics, hebreus i molts més.
- En Python el tipus *str* podia admetre qualsevol cadena de caràcters però en Java no existeix un tipus per a això. En compte d'això, vorem que s'utilitza la classe *String*.

3.2. Variables

Com hem dit, les variables deuen ser declarades abans de ser utilitzades (assignar-li un tipus). Amb el tipus aconseguim acotar el conjunt de valors que admet la variable (domini) i el conjunt d'operacions que podem fer sobre la variable.

Per tant, la informació bàsica que hem de saber d'una variable és:

| | |
|---------------|-------------------------------------------------------------------------------------|
| Tipus | El tipus bàsic de la variable (char, float, ...) |
| Identificador | El nom amb el qual accedim a la variable. |
| Valor | La informació que guarda la variable en un moment donat de l'execució del programa. |



El valor de la variable és opcional: quan es declara una variable no cal indicar-lo.

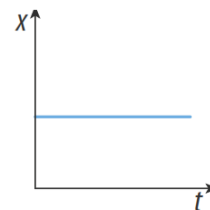
Exemples de declaracions de variables:

```
int edat;  
int sado = 1000;  
char lletraDNI = 'S';  
int comptador=0, x, y;  
char coord1='x', coord2='y', coord3='z';  
float radi, longitud;  
boolean pagat = false;
```

Com veiem, al final de cada assignació cal posar un punt i coma. També caldrà posar-lo després de cada instrucció.

3.3. Constants

Una constant és com una variable (té un tipus, un nom i un valor) però amb la condició que el seu contingut (el valor de la variable) no va a canviar al llarg del programa. Per això diem que és una **constant** i no una **variable** (no pot variar).



La forma de declarar una constant en Java és indicant-ho amb la paraula **final**. És a dir, el valor que li posem en eixe moment, és el valor "final" que tindrà:

```
final double PI = 3.141592;  
final int IVA = 21;  
final char BOMBA = 'Q';
```

Per convenció, posarem els noms de les constants tot en majúscules.

En Python no hi havia possibilitat de definir constants.

3.4. Paraules reservades

Estes paraules no podran usar-se per a noms de variables, ja que estan reservades:

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

4. Operadors



Els operadors de Java són els habituals , però recordem que no són exactament els mateixos que en Python. Ací estan tots, ordenats per prioritat d'execució:

| CATEGORIA DE L'OPERADOR | OPERADORS PYTHON | OPERADORS JAVA | ASSOCIATIVITAT |
|---------------------------------|-----------------------------|---------------------|----------------|
| Parèntesis, vectors | () [] | () [] | ESQUERRA |
| Operadors unaris | + - | ++ -- + - | DRETA |
| Potència | ** | NO | DRETA |
| Multiplicació, divisió i residu | * / // % | * / % | ESQUERRA |
| Suma i resta | + - | + - | ESQUERRA |
| Operadors relacionals | < <= > >= == != <> | < <= > >= == != | ESQUERRA |
| 'No' lògic | not | ! | ESQUERRA |
| 'I' lògic | and | && | ESQUERRA |
| 'O' lògic | or | ^ | ESQUERRA |
| Operador condicional | NO | ?: | DRETA |
| Assignacions | = += -= *= **= /= //= %= | = += -= *= /= %= | DRETA |

Vegem els operadors que sí que té C i Java però no té Python:

4.1. Operador lògic ^

En Java apareix un nou operador lògic (que no té Python) anomenat **or-Exclusiu**, que es representa per ^ . És el mateix que l'operador *or* però si els 2 operands són *true*, el resultat és *false*. És a dir, només val *true* quan un i **només un** dels 2 operadors és *true*. És a dir, segueix la següent taula de veritat:

| x | y | x ^ y |
|---|---|-------|
| F | F | F |
| F | V | V |
| V | F | V |
| V | V | F |

Els diferents de Python-Java estan **resaltats**.



4.2. Operadors aritmètics incrementals ++ i --

Els operadors unaris ++ i -- incrementen o decrem enten en 1 unitat la variable a qui acompanyen. S'utilitzen molt en Java i C (però no existeixen en Python).

Exemple:

```
x = 10;  
x++; // Ara la x val 11
```

En compte de x++ també haguérem pogut posar ++x, i el resultat seria el mateix. Ara bé, si eixe auto increment està dins d'una expressió, la cosa canvia:

Exemples d'auto increment dins d'una expressió:

| Nom | Instrucció | Funcionament | Resultat final |
|----------------|---------------------|-------------------------------------------------------|----------------------|
| Pre-increment | x = 10; y = ++x; | 1r) s'incrementa x 2n) S'assigna el valor de x a y | x val 11 y val 11 |
| Post-increment | x = 10; y = x++ | 1r) S'assigna el valor de x a y 2n) S'incrementa x | y val 10 x val 11 |

| Nom | Instrucció | Funcionament | Resultat final |
|----------------|------------------------|------------------------------------------|--------------------------|
| Pre-increment | x = 10; print(++x); | 1r) s'incrementa x 2n) Es mostra la x | x val 11 es mostra 11 |
| Post-increment | x = 10; print(x++) | 1r) Es mostra la x 2n) S'incrementa x | es mostra 10 x val 11 |

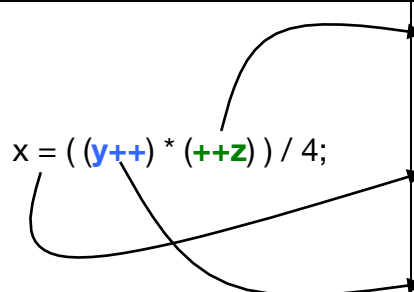
És a dir:

- Si l'operador va davant (++x), primer s'incrementa la variable i després s'utilitza el valor de la variable en l'expressió on apareix.
- Si l'operador va després (x++), primer s'utilitza el valor de la variable en l'expressió i després s'incrementa la variable.

Resumint:

| | Expressió amb auto incr/decr | Expressió equivalent | |
|------------------|------------------------------|----------------------|-------------|
| | | 1a operació | 2a operació |
| Pre - incr/decr | ++x | x = x + 1 ; | Ús d' x |
| | --x | x = x - 1 ; | Ús d' x |
| Post - incr/decr | x++ | Ús d' x | x = x + 1 ; |
| | x-- | Ús d' x | x = x - 1 ; |

Altre exemple, més enrevessat. Suposem *float* x, y = 2, z = 4;

| Instrucció amb operadors incrementals | Instruccions equivalents (s'executaran en eixe ordre) | | |
|----------------------------------------------------------------------------------------------------------------|-------------------------------------------------------|------------------|-----------|
|  x = ((y++) * (++z)) / 4; | 1r) Operadors "pre" | z = z + 1; | z val 5 |
| | 2n) Expressió pròpiament dita | x = (y * z) / 4; | x val 2.5 |
| | 3r) Operadors "post" | y = y + 1; | y val 3 |

Exercici sobre operadors aritmètics incrementals

- Donades 3 variables enteres a=4, b=5 i c=6, quin valor té cada variable després de cada seqüència d'assignacions?

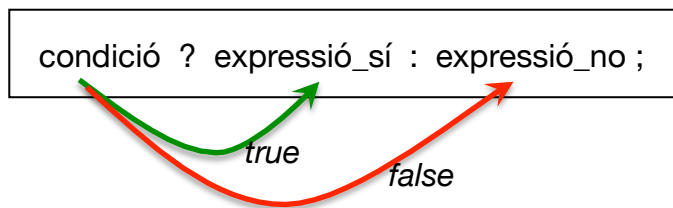
```
a = b++ - c--;
a += --b;
c *= a-- + b;
```

- Donades 3 variables enteres a=4, b=5 i c=6, què es mostrarà per pantalla?

```
System.out.println(a++ + --b);
System.out.println(a++ + " " + --b);
System.out.println(-b);
System.out.println(a + " " + b);
```

4.3. Operador condicional ternari "?"

Sintaxi:



Exemples d'ús:

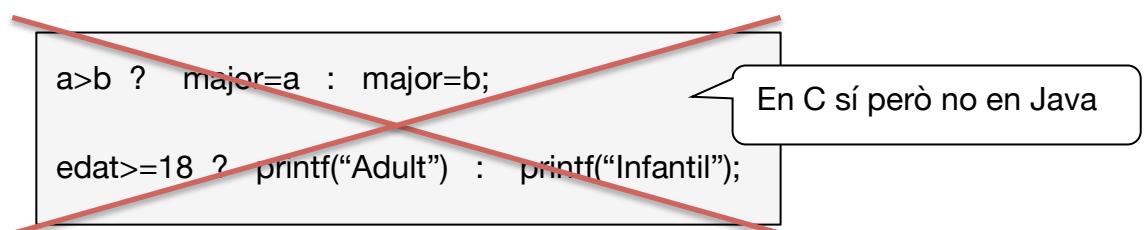
- Assignar un valor diferent a una variable segons una condició:

```
major = (a>b ? a : b);  
distancia = (a>b ? a-b : b-a);  
num_positiu = (num > 0 ? num : -num);
```

- Mostrar un valor diferent segons una condició:

```
printf( edat>18 ? "Adult" : "Infantil" );
```

Este operador també s'utilitza en el llenguatge C. Ara bé, en Java, després de l'interrogant sempre es posen **dades**, mentre que en C es poden posar **dades o instruccions**. És a dir: estos exemples estan permesos en C, però no en Java:



Nota: també es poden posar operadors condicionals dins d'altres.

Exercici sobre l'operador condicional " ? : "

- Fes un programa que, donades 2 variables enteres, mostre quin és el número més gran i quin el més menut. Millora el programa per a que mostre el número més gran de 3 variables. Usa l'operador condicional ternari.

5. Sortida de dades



5.1. Funcions *println* i *print*

En Python la sortida la fèiem amb *print*. En Java podem fer-ho de 3 maneres:

- ✓ *System.out.println(argument)*. Mostra per pantalla l'argument que se li passa entre parèntesis, i fa un salt de línia. Se li pot passar tant cadenes (String) com números (i objectes, com vorem més endavant).
- ✓ *System.out.print(argument)* és idèntica a *println* però sense fer el bot de línia al final. És a dir: *println("cadena")* és el mateix que *print("cadena\n")*.
- ✓ *System.out.printf("cadena de control", arg1, arg2, ...argN)* és el *printf* de C. Serveix per a especificar un format (quantitat de decimals, quantitat d'espais per a representar un número...). No el vorem.

Exemple:

```
public class OperadorsAritmetics {  
  
    public static void main(String[] args) {  
        int i = 12, j = 120;  
        double x = 12.345, y = 8.27;  
        System.out.print("Valors de les variables:\n");  
        System.out.println(" i = " + i);  
        System.out.println(" j = " + j);  
        System.out.println(" x = " + x);  
        System.out.println(" y = " + y);  
        System.out.print("Resultats de sumes:\n");  
        System.out.println(" i + j = " + (i + j));  
        System.out.printf(" x + y = %f", (x + y),  
        }  
    }  
}
```

Si un dels operands de + és una cadena, en compte de sumar, concatena.

Si no haguérem posat els parèntesis, concatenaria el valor de la variable *i* a la cadena de l'esquerra, i després li concatenaria el valor de la variable *j*. Però el que volem fer és una suma (*i + j*) i després concatenar eixe resultat a la cadena de l'esquerra.

5.2. Funcions sobrecarregades

La funció *println* (i *print*) admet només 1 paràmetre, però pot ser un *String* o un *int*, o un *float*... Això és perquè hi ha moltes funcions *println* (i *print*) ja definides. És a dir: moltes funcions amb el mateix nom, però distints tipus de paràmetres. Quan passa això es diu que la funció està "**sobrecarregada**".

Si en *Netbeans* escrivim *System.out.println*, automàticament ens mostra les diferents funcions que podem usar:

System.out.println

| | |
|---------------------|------|
| •println() | void |
| •println(Object x) | void |
| •println(String x) | void |
| •println(boolean x) | void |
| •println(char x) | void |
| •println(char[] x) | void |
| •println(double x) | void |
| •println(float x) | void |
| •println(int x) | void |
| •println(long x) | void |

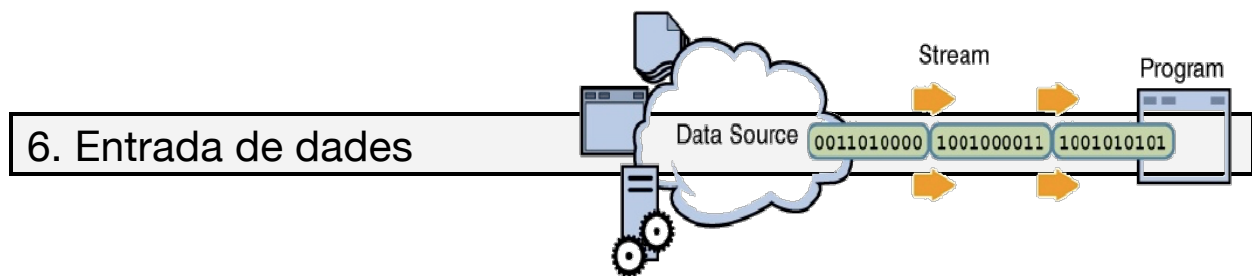


5.3. Travesses de Netbeans: *sout*, etc

I cada volta que vull mostrar una cosa he d'escriure *System.out.println()*? No, necessàriament: si en *Netbeans* escrius **sout** i la tecla del tabulador, automàticament s'escriu el *System.out.println("");* i es posa el cursor entre les cometes.

Hi ha moltes altres travesses en *Netbeans* (algunes ja les hem comentades en la introducció del tema). Les podeu consultar aquí:

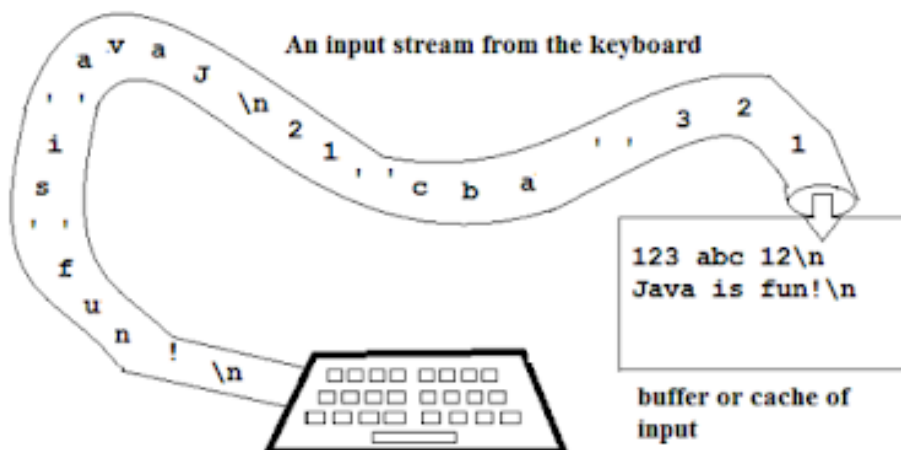
https://netbeans.org/project_downloads/usersguide/shortcuts-80.pdf



6. Entrada de dades

En Java, per a arreplegar dades de teclat és un poc més complicat que en Python, però no patiu si vos pareix complicat, ja que en la pràctica usarem una funcioneta nostra que ens servirà per a llegir en tots els programes.

Per a llegir de teclat cal usar una classe que ens "unisca" el teclat amb el programa.



Vorem 2 formes de er-ho: amb la classe **Scanner** i amb la classe **BufferedReader**.

Les principals diferències són:

| | Classe <i>Scanner</i> | Classe <i>BufferedReader</i> |
|---------------------------------|--------------------------------------------------------------------------------------------|-----------------------------------------|
| Llegir número | Directament | - Llegir cadena - Convertir a número |
| Obliga a tractar possible error | NO | SÍ |
| Cal buidar buffer d'entrada | SÍ | NO |
| Introduir decimal per teclat | Amb coma (10,3) però pot ser amb punt si abans posem: teclat.useLocale(Locale.ENGLISH); | Amb punt (10.3) |

6.1 Entrada de dades amb la classe *Scanner*

6.1.1. Passos a fer

1. Importar la classe. Cal afegir açò al principi del programa (fora de la classe):

```
import java.util.Scanner;
```

Si no ens acordem de posar-ho, no passa res, ja que Netbeans ens avisarà i ho posarà ell sol.

2. Indicar d'on volem llegir les dades (teclat, fitxer...). Cal posar en el *main*:

```
Scanner teclat = new Scanner(System.in);
```

Si no volguérem agafar dades de teclat sinó d'un fitxer, en compte del `System.in` caldria posar:

```
new File("ruta/nomFitxer");
```

3. Llegir usant els mètodes:

```
teclat.next();           //retorna String (només 1 paraula)
teclat.nextLine();       //retorna String (potser n paraules)
teclat.nextInt();        //retorna int
teclat.nextFloat();      //retorna float
teclat.next().charAt(0); //retorna un char (el primer que s'introdísca)
...
```

Per d'ús del mètode *nextInt*:

```
int edat;
edat = teclat.nextInt();
```


Exemple complet de lectura de dades amb la classe *Scanner*

```
import java.util.Scanner;

class Exemple {

    public static void main(String arg[]){
        Scanner teclat = new Scanner(System.in);
        String nom;
        int edat;
        float altura; // en metres

        System.out.print("Com et diuen? ");
        nom = teclat.nextLine();

        System.out.print("Quants anys tens?: ");
        edat = teclat.nextInt();

        System.out.print("Què medixes (en metres): ");
        altura = teclat.nextFloat();

        System.out.println("Hola, " + nom, "!");
        System.out.println("Tens " + edat + " anys i " + (altura *100) + " cm.");
    }
}
```

Exercicis sobre entrada de dades amb la classe *Scanner*

4. Fes un programa que demane per teclat les dades d'una venda: nom de l'article, quantes unitats (sense decimals) i el preu (amb decimals). Després ha de mostrar un text com: "4 cerveses a 1.25 euros la unitat, són 5.00 euros".
5. Modifica el programa anterior, però fent que primer pregunte les unitats i després el nom de l'article. Quan ho executes voràs que no obtens el resultat esperat. Mira l'apartat següent (*buffer d'entrada*) i intenta resoldre-ho.

6.1.2. Solució al problema del buffer d'entrada

Tot el que s'introdueix per teclat va guardant-se en un "buffer d'entrada".

Quan usem una funció de lectura de teclat, si en el buffer hi ha alguna cosa, no espera res de teclat sinó que ho agafa del buffer i després ho lleva del buffer.

A vegades convé buidar eixe buffer. Això es fa amb el *nextLine()* també.

Vegem amb un exemple diferents motius d'haver d'esborrar el buffer.

```
import java.util.Scanner;
```

```
class Exemple {
```

```
    public static void main(String arg[]) {  
        Scanner teclat = new Scanner(System.in);  
        String nom;  
        int edat;  
        float altura;
```

```
        System.out.print("Quants anys tens?: ");  
        edat = teclat.nextInt();  
        teclat.nextLine();
```

Igual que en C, si després de llegir un número volem llegir un text, com en el buffer encara estarà el caràcter de l'*intro* (\n), la lectura de text agafarà eixe caràcter i no esperarà a llegir res de teclat.

```
        System.out.print("Com et diuen? ");  
        nom = teclat.next(); // next() i no nextLine() si només volem 1 paraula  
        teclat.nextLine();
```

Si hem posat més paraules, les llevem per a que no les intente agafar una lectura de dades posterior.
Si eixa 2a lectura fora lectura de número, donaria error.

```
        altura = -1;  
        do {
```

```
            try {  
                System.out.print("Què medixes (en metres): ");  
                altura = teclat.nextFloat();
```

```
            } catch (Exception e) {  
                teclat.nextLine();  
                System.out.println("Error. Lletres no.");  
            }
```

Si volem evitar que quan demanem un número l'usuari pose una lletra, ho podem tractar amb un *try-catch* (ja ho vorem) i un bucle per a tornar a demanar el número. Però haurem d'esborrar eixa lletra del buffer. Si no, entraria en un bucle infinit sense poder posar res per teclat.

```
        } while (altura == -1);
```

```
        System.out.println("Hola, " + nom + "!");  
        System.out.println("Tens "+edat+" anys i " + (altura * 100) + " cm.");
```

```
    }  
}
```

6.2. Entrada de dades amb la classe *BufferedReader*

6.2.1. Passos a fer

1. Importar la classe. Cal afegir açò al principi del programa (fora de la classe):

```
import java.io.*;
```

Si no ens acordem de posar-ho, no passa res, ja que Netbeans ens avisarà i ho posarà ell sol.

2. Indicar d'on volem llegir les dades (teclat, fitxer...). Cal posar en el *main*:

```
BufferedReader teclat = new BufferedReader(new InputStreamReader(System.in));
```

Si no volguérem agafar dades de teclat sinó d'un fitxer, en compte del *new Input...* caldria posar:

```
new FileReader("ruta/nomFitxer");
```

3. Llegir usant els mètodes *readLine()*:

En esta classe, en compte d'haver una funció diferent per a llegir diferents tipus de dades, només hi ha una funció (*readLine()*) que llig cadenes de text:

```
try {  
    String temp = teclat.readLine();  
}  
catch( IOException e ) {  
    System.out.println("Error en l'entrada de dades");  
}
```

Per què el *try-catch*?

El *readLine* serveix per a llegir tant teclat com de fitxer. Si ho fem de fitxer pot donar error (si no existeix, no té permisos, etc.). Esta forma de llegir de teclat (amb la classe *BufferedReader*) obliga al programador a tractar eixe error: cal posar la lectura en un bloc *try-catch* (ja ho vorem en detall). Si no es fa, el compilador de Java dona error. Netbeans ens ofereix la possibilitat de tancar només eixa línia de lectura en el *try*, o bé posar tot el bloc de codi dins del *try*.

Exemple de lectura de dades amb la classe *BufferedReader*

```
import java.io.*;

class Proves {

    public static void main(String arg[]) {
        BufferedReader teclat = new BufferedReader(new InputStreamReader(System.in));
        String nom;
        int edat;
        float altura;

        try {
            System.out.print("Com et diuen? ");
            nom = teclat.readLine();

            System.out.print("Quants anys tens?: ");
            edat = Integer.parseInt( teclat.readLine() );

            System.out.print("Què medixes (en metres): ");
            altura = Float.parseFloat( teclat.readLine() );

            System.out.println("Hola, " + nom + "!");
            System.out.println("Tens " + edat + " anys i " + (altura * 100) + " cm.");

        } catch (Exception ex) {
            System.out.println("Ha hagut algun error d'entrada de dades");
        }
    }
}
```

Si volem llegir de teclat un número enter, cal llegir una cadena de text i després convertir-la a enter.

El mateix si volem llegir un número *float*, etc.

Exercici sobre entrada de dades amb la classe *BufferedReader*

6. Fes un programa que demane per teclat les dades d'un pacient: nom, edat, pes (amb decimals) i si és home o dona (h/d). A continuació mostra eixes dades en el format que vullgues. Fes la lectura de dades amb la classe *BufferedReader*. En l'apartat següent pots consultar com llegir de teclat un caràcter amb esta classe.

6.2.2. Conversions de *String* a un tipus de dades

Com hem vist a l'exemple anterior, Java permet passar un *String* a un *int*, *float*, etc. Això es fa amb el que s'anomena classes de cobertura (o *wrapper*), les quals tenen funcions (mètodes) per a passar un *String* al tipus bàsic desitjat de Java. Això ens servirà per si volem capturar de teclat un número, etc. i no un *String*.

Les classes *wrapper* existents són:

| Classe de cobertura (wrapper) | Mètode | Tipus bàsic de Java (al qual convertix la cadena) |
|-------------------------------|---------------------------|---------------------------------------------------|
| Integer | parseInt(cadena) | int |
| Float | parseFloat(cadena) | float |
| Double | parseDouble(cadena) | double |
| Boolean | parseBoolean(cadena) | boolean |
| Byte | parseByte(cadena) | byte |
| Short | parseShort(cadena) | short |
| Long | parseLong(cadena) | long |
| Char | parseChar(cadena) | char |

Compte: No existeix un *Integer.parseChar()* ja que no podem convertir tota una cadena en només 1 caràcter. Si volem llegir de teclat només 1 caràcter, farem com si llegírem una cadena però ens quedarem només en el primer caràcter d'eixa cadena. Això es fa amb el *charAt(0)*, com féiem en la classe *Scanner*:

```
char lletra;  
lletra = teclat.nextLine().charAt(0);
```

6.2.3. Conversions d'un tipus de dades a un *String*

Suposem que tenim dos variables: *cadena* de tipus *String* i *numero* de tipus *int*. Podem fer la conversió del número al *String* de formes diferents:

- `cadena = "" + numero;`
- `cadena = String.valueOf(numero);`
- `cadena = Integer.toString(numero); // Float.toString(), Double.toString()...`

6.3. Entrada de dades amb les nostres pròpies funcions

Potser ens resulte un poc farragós la lectura de teclat en Java. Per a evitar escriure tant de codi cada vegada que volem llegir alguna cosa de teclat, podem fer-nos una llibreria amb les nostres pròpies funcions (ja vorem com fer-ho en detall).

```
import java.util.Scanner;

class Funcionetes {
    static Scanner teclat = new Scanner(System.in);

    // -----
    public static int lligInt(String pregunta) {
        int numero;

        do {
            try {
                System.out.println( pregunta );
                numero = teclat.nextInt(); // Si hem posat lletres, va al catch
                // Si passa per aquí, no ha donat error. Buidem \n i tornem el número.
                teclat.nextLine();
                return numero;

            } catch (Exception e) { // Si hem posat lletres, avisa i buida el buffer:
                System.out.print("Ha de ser un número enter:");
                teclat.nextLine();
            }
        } while (true); // Bucle infinit fins que retornem el número correcte.
    }
    // -----
    public static float lligFloat(String pregunta) {
        ...
    }
    // -----
}
```

Funcionetes.java

I en el programa on tenim el *main*, simplement faríem:

```
class Proves {

    public static void main(String[] args) {
        int edat, any;
        edat = Funcionetes.lligInt("Quants anys tens?");
        any = Funcionetes.lligInt("En quin any nasqueres?");
        ...
    }
}
```

Proves.java

7. Estructures de control

Anem a veure com s'implementen en Java les estructures de control (*if*, *while*...) que ja veiérem en Python. En les estructures de control hi ha condicions i blocs d'accions. Recordem que un **bloc** és un conjunt d'instruccions que depenen d'una condició. Veiem-ho en este exemple de Python, on cada requadre és un bloc.

```
edat = int(input("Quants anys tens?"))
if edat < 0:
    print("Error")
else:
    print("Edat correcta")
    if edat < 12:
        print("No has fet ESO")
        if edat < 6:
            print("No has fet primària")
        else:
            print("Estàs en primària")
        print("Has de fer ESO")
    elif edat < 16:
        print("Estàs en ESO")
    elif edat < 65:
        print("Estàs en edat de treballar")
    else:
        print("Xe, jubila't!")
    print("T'he dit coses segons l'edat")
print("Adéu")
```

En C i Java estes són les diferències respecte a Python per a representar els blocs:

| BLOCS DE CODI | | |
|---------------|-------------------|---------------------------------------|
| | Python | C i Java |
| Condicció | Amb ":" al final | Entre parèntesis (...) |
| Abast | Sagnat obligatori | Entre claus { } Sagnat recomanable |

7.1. Bifurcacions



7.1.1. Bifurcació simple i doble: *if* / *if-else*

Sintaxi de l'*if* de Java:

La part de l'*else* és opcional,
com en Python.
Però no existeixen els *elif*.

```
if (condició) {  
    // Accions quan la condició és true  
}  
else {  
    // Accions quan la condició és false  
}
```

Amb este exemple d'*if* de Java podem veure també com es representen els blocs:

```
if (a < b) {  
    System.out.println("El menor és: " + a);  
}  
else {  
    if (a > b) {  
        System.out.println("El menor és: " + b);  
    }  
    else {  
        System.out.println("Els dos són iguals");  
    }  
}
```

En este exemple cadascun dels blocs tenen una única instrucció. Per tant, cap de les claus seria obligatòria. Este codi seria equivalent:

```
if (a < b) System.out.println("El menor és: " + a);  
else  
    if (a > b) System.out.println("El menor és: " + b);  
    else System.out.println("Els dos són iguals");
```

Exercici sobre bifurcacions *if-else*

7. Fes un programa que demane per teclat 3 números i que mostre el major.
8. Demana una nota amb decimals i mostra el text corresponent: "ins", "suf", "bé", "not" o "exc". O bé "error" si la nota no està entre 0 i 10.

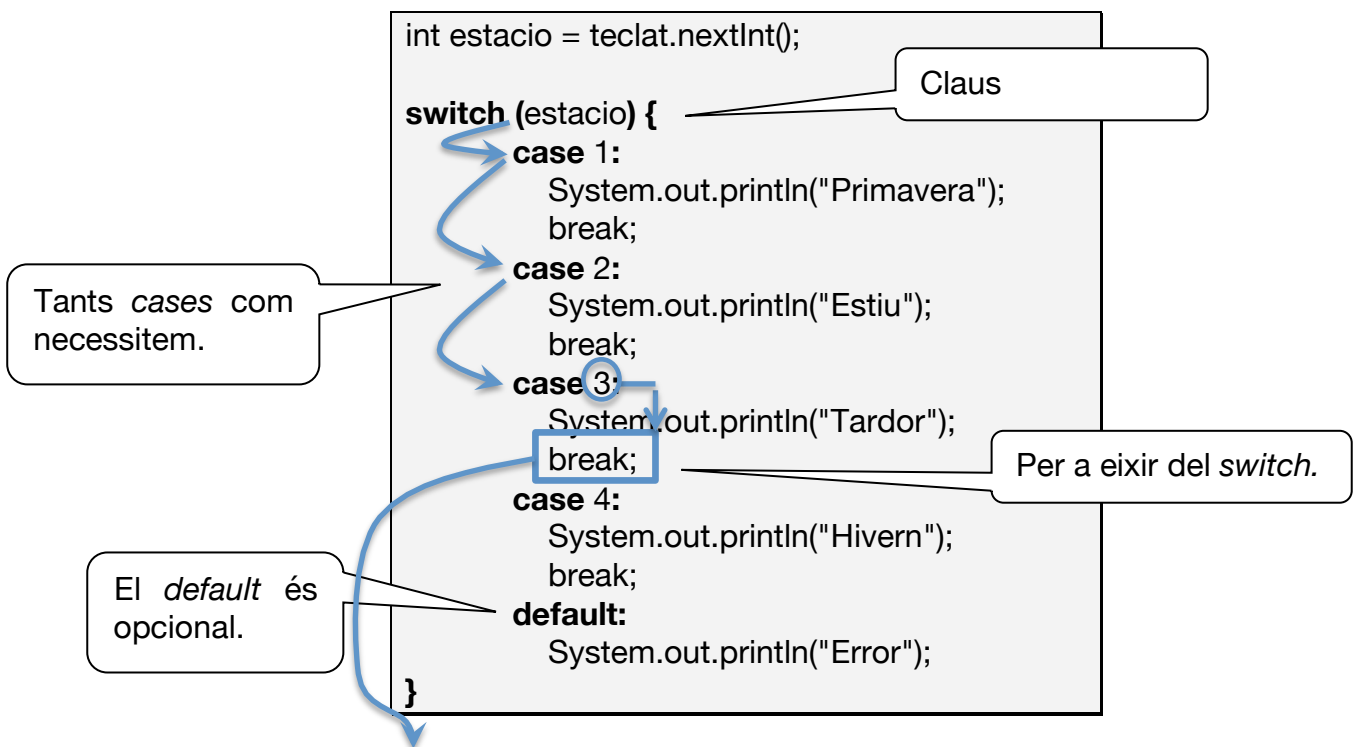
7.1.2. Bifurcació composta: **switch**

Depenent dels diferents valors d'una variable (o expressió), es podran executar diferents blocs de codi.

Eixa variable o expressió ha de ser de tipus *int*, *char* o *String*.



Exemple:



La idea sembla clara: depenent del valor de la variable *estacio*, s'executarà un tros de codi o un altre.

Per a què serveix el **break** en un **switch**?

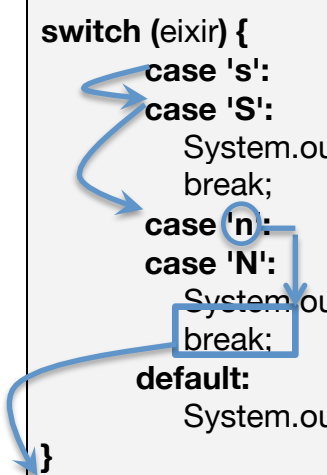
El funcionament del **switch** no és exactament "com voldríem". Suposem que **no** hem posat cap *break* i que *estacio* val 3. Per pantalla es mostraria: *Tardor*, *Hivern* i *Error*. És a dir: després d'avaluar-se el que hem posat en el **switch** (*estacio*), l'interpret comprova si el valor correspon al primer **case**. Si no, amb el segon. Si no, amb el tercer (i així successivament). Si sí que coincideix amb el valor d'un **case**, s'executarà tot el codi no només d'eixe **case**, sinó de tots els **cases** següents, fins trobar un *break*.

Per tant, si volem que s'execute només el codi corresponent al valor trobat, posarem un *break* en l'última instrucció de cada *case*.

Si l'expressió que s'avalua és un caràcter, els valors que es posen en els *case* han d'anar entre cometes simples:

```
System.out.println("Vols abandonar la partida?");
char eixir = teclat.nextLine().charAt(0);

switch (eixir) {
    case 's':
    case 'S':
        System.out.println("Adéu");
        break;
    case 'n':
    case 'N':
        System.out.println("Continuem jugant");
        break;
    default:
        System.out.println("Havies de contestar s/S/n/N");
}
```



En este exemple veiem que, amb el truquet de no posar *break* en la 's', fa que si la variable *eixir* és una 's', s'execute tot fins trobar un *break*. És a dir, tant si és 's' com si és 'S', s'executarà el mateix codi. Igual per a la 'n' i 'N'.

I si l'expressió és un *String*, els valors dels *case* han d'anar en cometes dobles:

```
System.out.println("De quin poble eres?");

switch (teclat.nextLine()) {
    case "Sueca":    System.out.println("La merda seca");    break;
    case "Algemesí": System.out.println("Ni dona ni rossí"); break;
    case "Canals":   System.out.println("Borts i criminals"); break;
    case "Museros":  System.out.println("Pocs i punyeteros"); break;
}
```

Com veiem, no és necessari el *default*.

Python no té equivalent exacte al *switch*. Però es podria implementar una cosa semblant usant els *elif*:

```
poble = input("De quin poble eres?")
```

```
if poble == "Sueca":  
    print("La merda seca")  
elif poble == "Algemesí":  
    print("Ni dona ni rossí")  
elif poble == "Canals":  
    print("Borts i criminals")  
elif poble == "Museros":  
    print("Pocs i punyeteros")
```

En el *switch* de Java s'entrarà en cada cas si es compleix una **condició d'igualtat** respecte la **mateixa expressió**.

Mentre que en els *elif* de Python es pot entrar en cada cas si es compleix la **condició d'igualtat, de menor, major...**, respecte **diferents expressions**.

Exercici sobre bifurcacions *switch*

Fes els següents exercicis usant el *switch* (encara que es podrien fer amb estructures *if-else*).

9. Demana una nota **sense decimals** i mostra el text corresponent: "ins", "suf", "bé", "not" o "exc". O bé "error" si la nota no està entre 0 i 10. Fes-ho amb un *switch*. Observa que si fora amb decimals, no es podria fer amb un *switch*.
10. Calculadora. Fes un programa que llija de teclat 2 números i una operació aritmètica. El programa farà el càlcul i imprimirà el resultat. Fes-ho amb *switch*.

Nota: s'admetran els següents caràcters per a cada operació aritmètica:

- Suma: s, S, +
- Resta: r, R, -
- Multiplicació: m, M, *, x
- Divisió: d, D, /

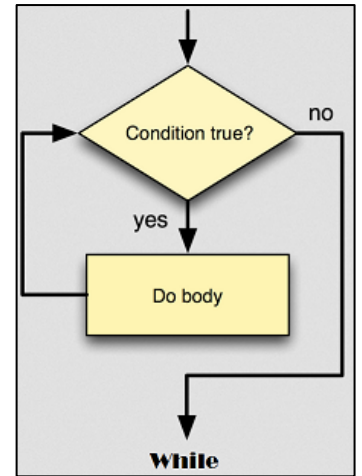
7.2. Bucles

7.2.1. Bucle condicional *while*

Sintaxi del *while* de Java:

```
while (condició) {  
    // Accions que es repetixen mentre la condició és true  
}
```

Com en Python, també podem posar *continue* i *break* dins del bucle, però no existeix la part de l'*else* de fora del bucle.

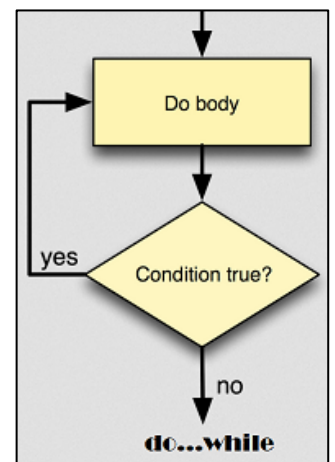


El funcionament és igual que en Python: mentre es complisca la condició es repetiran les accions. Quan el resultat de la condició siga *false*, acabarà el bucle i continuarà en la següent instrucció.

7.2.2. Bucle condicional *do-while*

Este tipus de bucle no el té Python. També repeteix un tros de codi mentre es compleix una condició però ara primer es fan les accions i després es comprova si torna a entrar:

Sintaxi del *do-while* de Java:



Primer s'executen les accions.

```
do {  
    // Accions que es repetixen mentre la condició és true  
} while (condició);
```

Després es comprova si es tornen a repetir les accions.

Per tant, en el *while* "normal" potser no s'executen mai les accions de dins del bucle (si des d'un principi ja no es complira la condició), mentre que en el *do-while*, segur que sempre es van a executar les accions, almenys 1 vegada.

Exercicis sobre bucles condicionals

Tots els problemes de bucles poden fer-se amb *while* o amb *do-while*, encara que potser una de les dos és més idònia. Fes els següents exercicis de 2 formes distintes (amb *while* i amb *do-while*) i pensa quina de les 2 és la més idònia en cada exercici.

11. Programa que demane un número *i*, a continuació, que demane contínuament quin és el quadrat d'eixe número fins que siga encertat.

12. Programa que vaja demanant les notes que ha tret l'alumnat fins que s'introduïska una nota -1. En acabant, que mostre la nota mitja, quantes notes estan aprovades i quantes suspeses (el número -1 introduït no comptarà per a les estadístiques, clar).

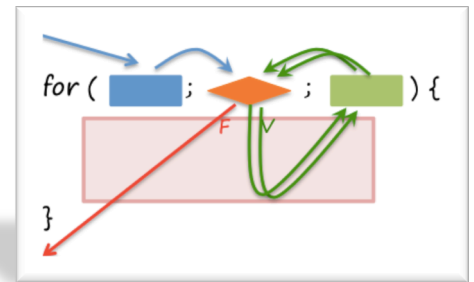
Estos exercicis següents es faran millor amb un bucle incondicional (*for*) però intenta fer-los amb bucles condicionals.

13. Programa que mostre els números del 10 al 20.

14. Programa que mostres els números del 20 al 10, de 3 en 3.

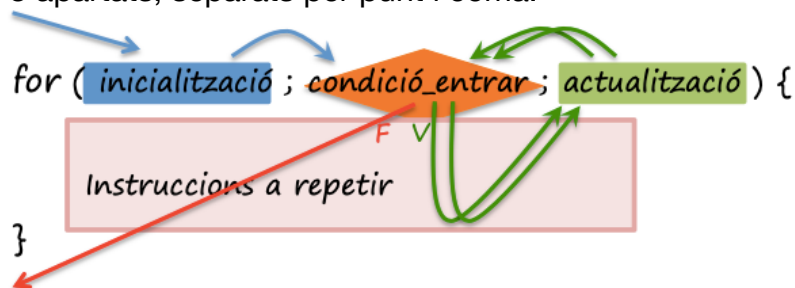
7.2.3. Bucle incondicional *for*

En l'explicació del *for* de Python del tema anterior veiérem l'exemple de recórrer els números de l'1 al 9. Veiem ara com seria en C i Java:



```
int i;  
for (i=1 ; i<10 ; i++) {  
    System.out.println( i);  
}
```

Veiem que són 3 apartats, separats per punt i coma:



On tenim que:

- **Inicialització**. Instrucció (o instruccions separades per coma) que s'executaran **només** abans de la primera iteració del *for* (i abans d'avaluar la condició). Sol usar-se per a inicialitzar la variable que fa de comptador.
- **Condició**. Si es compleix la condició (que pot ser composta), s'executaran les sentències de dins del *for*. Si no es compleix, eixirem del *for*.
- **Actualització**. Instrucció (o instruccions separades per coma) que s'executaran després d'executar les sentències de dins del *for*. A continuació, torna a avaluar-se la condició.

És a dir, un possible flux seria:

Inicialització
Condició *True* --> sentències --> actualització
Condició *True* --> sentències --> actualització
Condició *True* --> sentències --> actualització
...
Condició ***False*** --> El bucle acaba

Altres exemples:

```
int i;
for (i=10 ; i<=30 ; i+=5) {
    printf("%d\n", i);
}
```

Això mostrarà els números: 10, 15, 20, 25, 30 (del 10 al 30, de 5 en 5)

```
int i;
for (i=10 ; i <=30 ; i++) {
    if (i %5 == 0) {
        printf("%d\n", i);
    }
}
```

Això també mostra els números: 10, 15, 20, 25, 30 (múltiples de 5 en eixe rang). Esta última solució és menys eficient ja que fa més iteracions del bucle *i*, a més, en cada iteració fa una comprovació (si és múltiple de 5).

```
int i;
for (i=10 ; i>4 ; i--) {
    printf("%d\n", i);
}
```

Això mostrarà els números: 10, 9, 8, 7, 6, 5 (del 10 al 4, en ordre descendent).

Fixa't que, com ara el comptador "va cap arrere", la *i* no s'incrementa (*i++*) sinó que es decrementa (*i--*). I la condició no és menor (<), sinó major (>).

Exemple de bucle infinit (que cal evitar):

```
int i;
for (i=1 ; i != 10 ; i+=3) {
    printf("%d\n", i);
}
```

Compte! Això mostrarà els números: 1, 4, 7, 10, 13, 16, 19, 22... fins a l'infinit!

Cal anar en compte en la condició. En este cas, per molt que incrementem la variable *i*, sempre es complirà que *i* és distint de 10. Per tant es genera el que s'anomena *un bucle infinit*.

Exercicis sobre bucles incondicionals

Com ja hem vist, els bucles incondicionals (*for*) són més adequats quan sabem quantes vegades s'ha de repetir unes sentències, mentre que els condicionals (*while*, *do-while*) són més adequats quan depenen d'una condició que no sabem quan es complirà. No obstant, qualsevol bucle pot implementar-se de les 3 maneres.

Fes en Java els exercicis següents amb bucles *for* (i, si vols, intenta també fer-los amb bucles condicionals). Són els mateixos exercicis que ja férem en Python.

15. Programa que demane una taula de multiplicar i la mostre.

16. Programa que calcule el màxim de 10 números introduïts per teclat.

17. Programa que calcule el màxim, mínim i mitjana de 10 números entrats per teclat.

18. Programa que mostre les taules de multiplicar del 2 al 9.

19. Programa que calcule el factorial d'un número introduït per teclat ($n!$) tenint en compte que: $0! = 1$

$$n! = n * (n-1) * (n-2) * \dots * 2 * 1 \quad (\text{sent } n > 1)$$

Feu-ho amb diferents solucions:

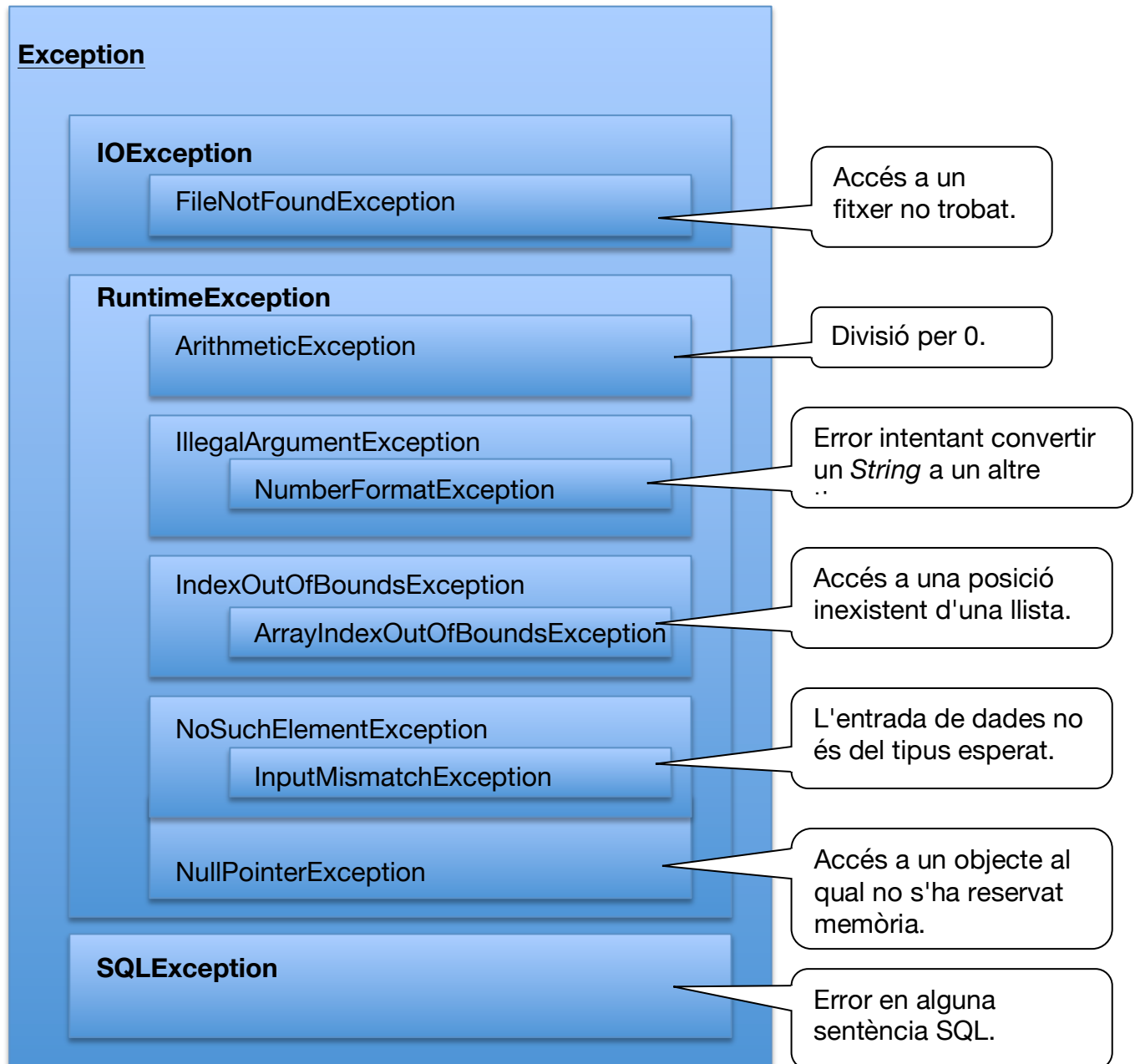
- a) Amb un *for* recorrent els números des de l'1 fins n
- b) Amb un *for* recorrent els números des d' n fins a 1
- c) Amb un *while* recorrent els números des de l'1 fins n
- d) Amb un *while* recorrent els números des d' n fins a 1

8. Tractament d'excepcions



Una **excepció** és un error que pot fer avortar un programa. Cada error que es produeix és un objecte de la classe *Exception* (ja vorem això de classes i objectes).

Com hi ha molts tipus d'errors, hi ha moltes classes d'*Exceptions*. De fet, hi ha una **jerarquia d'excepcions**. Ací hi ha una xicoteta mostra de les més freqüents:



Si volem “capturar” un error i tractar-lo sense que avorte el programa, ho podem fer amb blocs *try-catch-finally*:

Sintaxi:

```
// Instruccions que no deuen donar error
```

```
try {  
    // Instruccions que podrien donar error  
}  
catch ( TipusExcep1 e ) {  
    // Instruccions per si ocorre en error d'eixe tipus  
}  
catch ( TipusExcep2 e ) {  
    // Instruccions per si ocorre en error d'eixe tipus  
}  
finally {  
    // Instruccions que SEMPRE s'executaran  
    // encara que haja hagut algun error no capturat.  
}
```

Si dona error en alguna instrucció, no s'executaran les següents d'este bloc.

Tants catch com tipus d'error volem capturar. S'executarà el bloc corresponent a l'error produït.

```
// Instruccions que no deuen donar error
```

Si ha hagut un error no capturat, després d'executar el *finally* el programa avortarà i, per tant, estes instruccions no s'executaran.

Exemple de tractament d'excepcions:

```
public static void main(String[] args) {  
    Scanner teclat = new Scanner(System.in);  
    int alumnes, ordinadors;  
    float mitja;  
    System.out.println("Calculem la mitja d'ordinadors per alumne en classe");  
  
    try {  
        System.out.println("Quants alumnes?");  
        alumnes = teclat.nextInt();  
        System.out.println("Quants ordinadors?");  
        ordinadors = teclat.nextInt();  
        mitja = alumnes/ordinadors;  
        System.out.println("1 ordinador cada " + mitja + " alumnes");  
    }  
  
    catch (ArithmeticException e){ // Divisió per 0  
        System.out.println("Sense alumnes no es pot calcular la mitja");  
    }  
  
    catch (InputMismatchException e){ // Error intentant llegir enter de teclat  
        System.out.println("Cal introduir números enters");  
    }  
  
    System.out.println("Adéu");  
}
```

Més endavant vorem com propagar excepcions o crear les nostres pròpies excepcions i llençar-les.

20. Fixa't en el següent programa i contesta les preguntes:

```
import java.util.InputMismatchException;
import java.util.Scanner;

class Proves {

    public static void main(String[] args) {
        Scanner teclat = new Scanner(System.in);
        int valor=0;
        System.out.println("Dis-me un número enter:");
        try {
            System.out.println("Entrem al try");
            valor = teclat.nextInt();
            valor = 10 / valor;
            System.out.println("Tot ok i valor val: " + valor);
        } catch (InputMismatchException e) {
            System.out.println("Error: no has posat números enters");
            System.out.println("Ara valor val: " + valor);
        } catch (ArithmeticException e) {
            System.out.println("Error de divisió per 0");
            valor = 10 / valor;
            System.out.println("Ara valor val: " + valor);
        } finally {
            System.out.println("Entrem al finally");
            valor++;
            System.out.println("Valor al final de finally : " + valor);
        }
        System.out.println("Hem eixit de zona de control d'excepcions");
        valor++;
        System.out.println("Valor final: " + valor);
    }
}
```

- Què mostrarà per pantalla si introduïm números enters diferents de 0?
- Què mostrarà per pantalla si introduïm el número 0?
- Què mostrarà per pantalla si introduïm lletres?
- Hi ha un catch per a controlar que s'introduïsquien lletres, però el programa avorta. Per què?

9. Comentaris

Els comentaris s'utilitzen per a explicar (al mateix programador, no a l'usuari final) alguna part del programa. Per exemple, servixen per a explicar el significat d'alguna variable o indicar a principi del programa l'autor del programa, data de creació, etc.

El compilador no analitza les línies comentades. Simplement, passa d'elles.

En Java podem incloure comentaris als nostres programes de 3 formes diferents:

- Comentari d'una línia: A partir de la combinació de caràcters //

```
int import;    // guardarà els diners, en euros
char torn;     // guardarà una 'N' si és Nocturn o una 'D' si és Diürn
```

- Comentari de diverses línies: Per a comentar grans blocs. Es considera comentari tot allò que queda entre /* i */

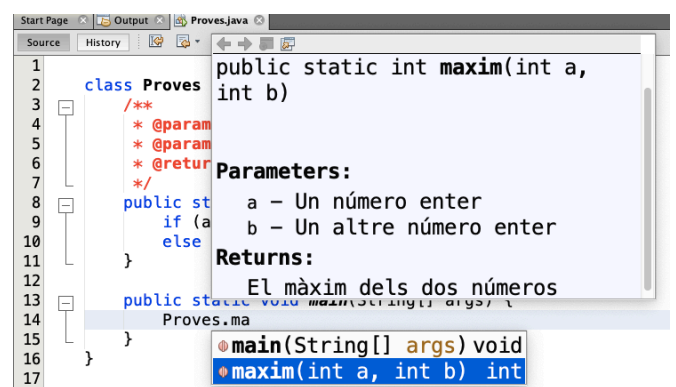
```
/*
  Autor: Abdó Garcia
  Data: 25-08-2020
  Utilitat: Este programa només és per a explicar els comentaris
*/
public static void main(String[] args) {
    ...
}
```

- Javadoc: comentaris per a documentació de Java. Sol servir per a documentar funcions (ja les vorem):

En Netbeans, si escrivim /** i intro, ja ens posa els @param de la funció i el @return.

```
/**
 * @param a Un número enter
 * @param b Un altre número enter
 * @return El màxim dels dos números
 */
public static int maxim(int a, int b){
    if (a > b) return a;
    else return b;
}
```

Una volta documentat amb el Javadoc, si el programador escriu el nom de la funció, apareixerà l'ajuda de la informació que li hem posat, amb una estètica estàndard:



10. Exercicis

21. Què valdran les següents expressions?

1. $(x \geq 0) \parallel (x < 0)$
2. $(x == y) \parallel (x != y)$
3. $(x == y) \wedge (x != y)$
4. $(4 > 3) \wedge (3 < 4)$

22. Programa que demane per teclat una quantitat d'euros múltiple de 5 i el desglosse en bitllets (els mínims possibles). Per exemple, si tenim 1395 euros:

2 bitllets de 500 euros
1 bitllet de 200 euros
1 bitllet de 100 euros
1 bitllet de 50 euros
2 bitllets de 20 euros
1 bitllet de 5 euros

Nota: com ja sabem, hi ha bitllets de 500, 200, 100, 50, 20, 10 i 5 euros.

23. Programa que demane 10 números per teclat. En acabant, caldrà mostrar un missatge dient si s'ha introduït algun número negatiu o si no. A més, es mostrarà la quantitat de números parells i la quantitat d'imparells.

24. Demana 2 números i calcula la potència (el primer elevat al segon) amb un bucle. En acabant, mostra-la.

25. Programa d'endevinar un número. L'usuari pensa un número de l'1 al 100. L'ordinador ha d'endevinar-lo. L'usuari anirà indicant si el número a encertar és major o menor que el que ha dit, fins que siga encertat.

26. Altre programa d'endevinar un número. L'ordinador tria un número de l'1 al 100 (busca a Internet una funció que calcule un número aleatori en Java). L'usuari ha d'intentar endevinar-lo amb els missatges que vaja dient l'ordinador de si el número a encertar és major o menor del que ha dit l'usuari.