JAVA

EL LLENGUATGE

SINTAXI BÀSICA

La sintaxis del java és molt semblant a la del C. Algunes de les característiques bàsiques:

- Totes les sentències acaben amb ";"
- Sensibles a majúscules i minúscules. No és el mateix *curs* que *Curs*.
- Els blocs de codi es delimiten amb claus {}
- Identificadors: Noms de variables, mètodes, classes, ...
  - El primer caràcter ha de ser una lletra Unicode, encara que també es pot utilitzar $ i _ Es poden utilitzar accents, dièresis, ñ, ç, ...
  - La resta poden ser lletres unicode, números, $, o _
  - Convencions:
    - **Variables, mètodes i paquets**: nom en minúscules, sense abreviatures ni símbols $ i _. Si està format per més d'una paraula, la primera lletra de la segona i següents en majúscula: *nomPadríPatern.*
    - **Constants**: Tot en majúscules i les paraules separades per _ *NOMBRE_RODES.*
    - **Classes**: Com les variables, excepte la primera amb majúscula.

VARIABLES

S'utilitzen per a emmagatzemar valors durant l'execució del programa.

Declaració

Abans de poder utilitzar una variable dins el nostre programa l'haurem de declarar, dir de quin tipus és. Segons la seva declaració, una variable podrà guardar unes dades o unes altres.

```
int a; //un número sencer
char c; //un caràcter
```

Inicialització

Hem d'assignar un valor a la variable abans d'utilitzar-la. Podem fer-ho en declarar-la o més tard. El compilador ***no*** assigna valors per defecte. Si troba que utilitzam una variable abans d'inicialitzar-la dona error de compilació.

```
a=2; //assignam valors a variables ja declarades
c='r';
int i=24; //declaram i inicialitzam una variable.
```

Àmbit de referència

El punt del codi on declarem una variable determina on la podem utilitzar. Podem utilitzar una variable dins el bloc on està definida, és a dir des del punt on la declarem fins a la clau que tanca el bloc on l'hem declarada.

```
public class Desbarat{
int w=-1;

public boolean doi() {
        int a=0;
        for (int i = 1; i < 10; i++) {
                int b = 0;
                b = b + i;
        }
        System.out.println(a);
        w=b;
        return false;
}

public void doi2() {
        System.out.println(w);
}
}
```

Exemples

w és accessible per a tots els mètodes de la classe.

a és accessible dins tot el codi del mètode *doi*.

b, *i* són accessibles només dins el bloc de codi de la iteració.

La instrucció *w=b* donaria error de compilació, perquè per el compilador *b* no existeix fora de la iteració. Per arreglar-ho hauríem de declarar la variable *b* fora del bucle, o posar la instrucció *w=b* dins les claus del *for*.

Tipus de dades primitius

Són tipus de dades predefinits per el llenguatge, en contraposició a les classes que podem definir noltros. N'hi ha vuit:

byte: 8 bits, amb signe => -128 a 127

short: 16 bits, amb signe => -32,768 a 32,767

* **int**: 32 bits => -2,147,483,648 a 2,147,483,647

long: 64 bits => -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807

* **float**: 32 bits, simple precisió.

* **double**: 64 bits, doble precisió.

* **boolean**: valors *true* o *false*

char: caràcter unicode, de '\u0000' a '\uffff'

* **String**: encara que no és un tipus primitiu, sinó una classe, és el mecanisme que ofereix Java per a emmagatzemar cadenes de caràcters.

* Els més utilitzats.

Constants

No permeten modificar el seu valor una vegada assignat. Es declaren amb *final*.

```
final double PI=3.14159;
```

```
final double PI;
PI=3.14159;
```

Literals

Són valors introduïts directament en el codi.

byte, short, int i long. Es poden introduir en decimal, octal i hexadecimal.

```
int dec = 12;      // 12

int oct = 012;     // 10 en decimal. Si posam un zero inicial ho interpreta en octal.

int hex = 0x12;   // 18 en decimal. 0x inicial ho interpreta en hexadecimal.
```

float i double. Es poden expressar en notació científica (E o e) , o posant darrera del valor F o f per a floats i D o d per a doubles. Per defecte són double.

```
double d1 = 123.4;

double d2 = 1.234e2;

float f1 = 123.4f;
```

char i string. Els char van entre cometes simples, 'c', i les cadenes entre cometes dobles, "hola". Es poden utilitzar caràcters unicode, '\u00f1' =>ñ, en qualsevol lloc.

Seqüències d'escape: \b (backspace), \t (tab), \n (line feed), \f (form feed), \r (carriage return), \" (cometa doble), \' (cometa simple), \\ (contrabarra).

OPERADORS

Assignació: = *a=2; b=a; resultat=3;*

Aritmètics: + - * / % *(suma, resta, multiplicació, divisió, mòdul)*

Aritmètics unaris: + - ++ -- *(positiu, negatiu, increment, decrement)*

```
resultat++; //resultat=4

b=resultat++ // b=4, resultat=5

b=++resultat //b=6, resultat=6
```

Relacionals: == != > >= < <= *(igualtat, diferència, major, major o igual, menor, menor o igual)*

Lògics: && || ! *(and, or, negació. Curtcircuitats: en trobar una condició que converteix l'expressió en false (and) o true (or) ja no avaluen les condicions restants.)*

Concatenació de cadenes: + *(String c="hola, "+"mon";)*

CONVERSIONS DE TIPUS IMPLÍCITES

```
variable_destí = variable_origen;
```

El tipus de la variable de destí ha de ser de mida igual o superior al tipus d'origen.

Exemples:

```
int i;

   byte b = 30;

   i = b; // correcte, conversió byte a int

int n;

   long l = 20;

   n = l; // error, no es pot convertir
```

CONVERSIONS DE TIPUS EXPLÍCITES

```
variable_destí = (tipus_destí)dada_origen;
```

Indicam al compilador que converteixi dada_origen a tipus_destí per que pugui ser emmagatzemada a variable_destí.

Aquesta operació es diu *casting*.

En alguns casos pot provocar pèrdua de dades.

Exemples:

```
char c;
     double d = 34.6;
     c = (char) d; // s'elimina la part decimal (truncat)
byte k;
     int p = 400;
     k = (byte) p; // k==112 es perden dades, però és pot fer
```

COMENTARIS

Els comentaris al codi s'utilitzen per a fer-lo més entenedor a la gent que l'hagi de llegir. No tenen cap repercussió sobre el codi final, ja que el compilador els ignora.

Una altra utilitat que tenen és permetre al programador especificar que una part del codi no s'executi, normalment quan es depura el codi cercant una errada.

En java tenim tres tipus de comentaris:

Comentaris d'una línea

```
//radi de la circumferència

float r=23.56;
```

Comentaris de diverses línies

```
/*radi de la circumferència.

Utilitzant en tots els càlculs.*/

float r=23.56;
```

Comentaris Javadoc. Aquesta és una eina que, a partir d'aquests comentaris que troba al codi de les classes de l'aplicació, genera la documentació en format html com la que podeu trobar a l'API de java.

```
/**Radi de la circumferència.

Utilitzant en tots els càlculs.*/

float r=23.56;
```

INSTRUCCIONS DE CONTROL DE FLUX

Les instruccions d'un programa s'executen de forma seqüencial, a no ser que utilitzem iteracions o condicionals.

Condicionals

Permeten decidir el codi que s'executarà segons el valor d'una certa condició.

If: permet avaluar qualsevol expressió.

```
if(condició){
    instruccions;
}
```

```
if(condició){
    instruccions1;
}else{
    instruccions2;
}
```

```
if(condició1){
    instruccions1;
}else if(condició2){
    instruccions2;
}else{
    instruccions3;
}
```

On posa condició hi podem posar qualsevol expressió que torni un valor booleà.

switch: permet avaluar expressions de tipus byte, short, int, char, String i tipus enumerats. S'executa tot el codi a partir de la primera condició certa fins a trobar un break o arribar al final.

```
int month = 8;
switch (month) {
    case 1:  {
             System.out.println("Gener");
             break;
    }
    case 2:  {
             System.out.println("Febrer");
             break;
    }
   …
    case 11: {
             System.out.println("Novembre");
             break;
    }
    case 12: {
             System.out.println("Desembre");
             break;
    }
    default: {
             System.out.println("Error!.");
             break;
    }
}
```

```
int month = 8;
 switch (month) {
          case 1:
          case 3:
          case 5:
          case 7:
          case 8:
          case 10:
          case 12:{
             numDays = 31;
             break;
                }
          case 4:
          case 6:
          case 9:
          case 11: {
             numDays = 30;
             break;
                }
          case 2: {
                          numDays=28;
                          break;
                }
          default: {
             System.out.println("Mes no vàlid.");
             break;
                }
    }
```

Iteracions:

Permeten repetir codi segons una certa condició, o un nombre determinat de vegades. Dins la iteració s'ha de modificar de qualque manera el valor de la condició si volem que el fluxe d'execució surti de la iteració.

while: avalua la condició. Si es certa executa les instruccions del bloc. Repeteix aquest cicle fins que la condició sigui falsa. Pot ser que les instruccions de dins el while no s'executin mai.

```
while(condició){
        instruccions;
.}
```

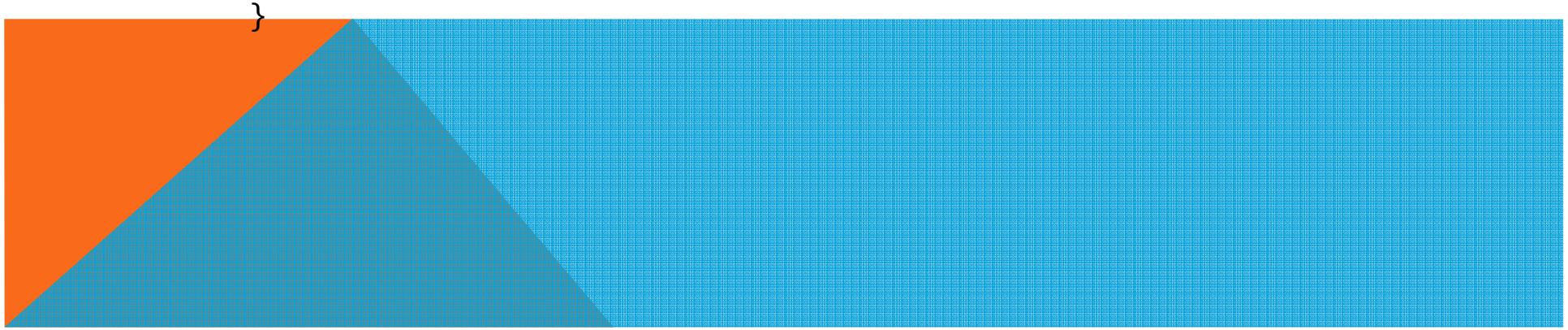
do - while: Executa les instruccions del bloc. Avalua la condició. Repeteix aquest cicle mentre la condició sigui certa. Les instruccions internes s'executen al manco una vegada.

```
do{
        instruccions;
} while(condició);
```

for: Inicialitza la iteració. Si la condició es certa executa les instruccions. Realitza l'increment, torna a comprovar la condició...

```
for(inicialització; condició; increment){
        instruccions;
}
```

```
      for(int i=0; i<10; i++){
  System.out.println(i);
        }
```



for estés (for each): Aquesta forma d'ús del for, que ja existia en altres llenguatges, facilita el recorregut d'objectes existents en una col·lecció sense necessitat de definir el nombre d'elements a recórrer.

```
  for ( TipoARecorrer nombreVariableTemporal :
nombreDeLaColección ) { Instrucciones}
```

El for estès té alguns avantatges i alguns inconvenients. No s'ha de fer servir sempre. El seu ús no és obligatori, de fet, com hem indicat, en versions anteriors ni tan sols existia en el llenguatge.

L'utilitzarem quan sapiguem per endavant que hem de recórrer completament la col·lecció sempre.

```
public void listar_TotsElsNoms () {

        for (String i: listaDeNoms) {

        System.out.println (i); }}
```

Bifurcacions

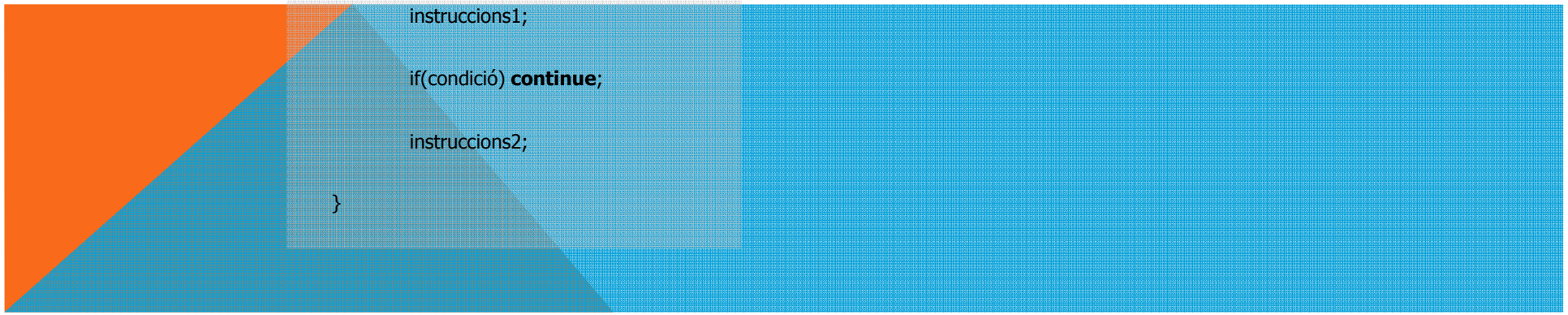
Permeten rompre la seqüència lògica del programa.

break: acaba l'execució de la iteració (o switch) dins la qual es troba. S'executarà la instrucció posterior a la iteració (o al switch).

```
while(condició){
        instruccions1;
        if(condició) break;
        instruccions2;
.}
.Instruccions3;
```

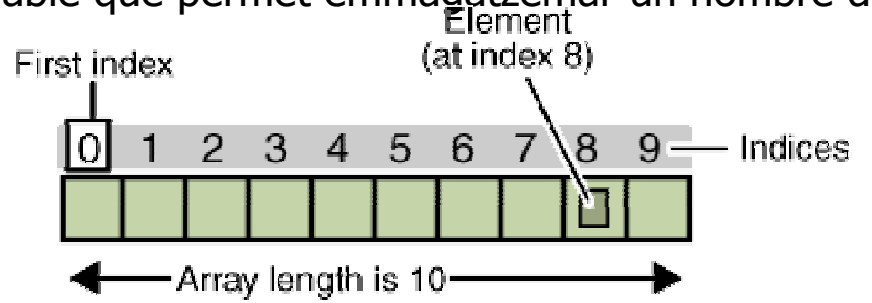
continue: Omet les instruccions posteriors i continua amb el següent cicle de la iteració.

```
while(condició){
        instruccions1;
        if(condició) continue;
        instruccions2;
}
```



Arrays

Un array és una variable que permet emmagatzemar un nombre determinat de valors del mateix tipus.



- Declaració: No crea la variable, únicament la declara

```
int[] notes;
int[][] caselles;
```

- Creació: reserva l'espai per l'array. Especifiquem la quantitat d'elements que contindrà

```
notes = new int[10];
caselles = new int[lonX][lonY];
```

- La variable no conté l'array sinó la referència, és a dir "l'adreça de la memòria" on realment està guardat l'array.

- Inicialització: Abans de poder recuperar els valors guardats per l'array s'han d'assignar.

```
notes[3]=6;
```

Arrays

Hi ha una forma de declarar l'array i inicialitzar-lo tot a l'hora

```
String[] noms = {"Miquel", "Toni", "Margalida", "Magdalena"}
```

- length: noms.length torna la longitud de l'array.

- Còpia d'arrays: La variable array en realitat és una referència a l'array. Si executam el següent codi::

```
String[] alumnes = noms;
alumnes[1]="Josep";
```

llavors

```
noms[1]=="Josep";
```

Per fer-ne una còpia hem de crear una altre array i passar-li els elements un a un, o utilitzar

```
System.arraycopy(origen, posInicial, destí, posInicial, nombreElements);
```