PAU CASESNOVES
Centre Integrat de Formació Professional

Desenvolupament d'aplicacions web

Programació

UT 2.3: Java. El llenguatge.

CC BY NC SA

Joan Pons Tugores

Índex de continguts

Sintaxi bàsica

La sintaxis del java és molt semblant a la del C i altres llenguatges derivats d'ell. Algunes de les característiques bàsiques:

- Totes les sentències acaben amb “;”
- Sensibles a majúscules i minúscules. No és el mateix la variable *curs* que la variable *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 de nomenclatura

- **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.

```
AlumneGrauSuperior
```

Variables

S'utilitzen per a emmagatzemar valors durant l'execució del programa. El seu valor pot canviar 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 i quin nom té. 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 en una expressió, com a paràmetre, ... 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 declaram 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 declaram fins a la clau que tanca el bloc on l'hem declarada.

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.

```
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);
  }
}
```

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;
```

o bé

```
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).

```
String s="Hola";

char c='h';

String s="\"Hola\"";
```

Operadors

- Assignació: =

```
a=2;   b=a; resultat=3;
```

- Aritmètics: + - * / % (suma, resta, multiplicació, divisió, mòdul)
- Aritmètics unaris: + - ++ -- (positiu, negatiu, increment, decrement)

```
int resultat=3;

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.)

- Bit: ~ & ^ | << >>

```
int a = 5  (0101)

int b = 7  (0111)


a&b=5 (101)

a|b=7 (111)

a^b=2 (10)

~a=-6 (11111111111111111111111111111010)

a<<2=20 (10100)

a>>2=1 (1)
```

- Concatenació de cadenes: +

```
String c="hola, "+"mon";
```

Conversions de tipus

De vegades tenim una dada en una variable d'un determinat tipus i la volem traspassar a una variable d'un altre tipus, o volem operar amb una variable d'un altre tipus. En aquests casos hem de fer una conversió de tipus.

Les conversions de tipus poden ser implícites o explícites.

Implícites

Vol dir que es fan automàticament, sense haver d'afegir res especial:

```
variable_destí = variable_origen;
```

El tipus de la variable de destí ha de ser semblant (per exemple totes dues numèriques) i de mida igual o superior al tipus d'origen. Si no, no és capaç de representar el valor.

Exemples:

- En aquest exemple posam un valor de tipus *byte* dins un *int*. Tots dos són numèrics i el valor màxim que pot emmagatzemar *byte* és més petit que el de l'*int*.

```
int i;

byte b = 30;

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

- En aquest cas intentam guardar un long dins un int. Tots dos són numèrics,però el valor màxim que pot emmagatzemar *long* és més gros que el de l'*int*. Es podria perdre informació i no ho permet.

```
int n;

long l = 20;

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

Explícites (Càsting)

Quan no es pot fer una conversió implícita hem d'afegir qualque cosa de manera que li diguem al compilador que assumim el risc de fer la conversió, que podem perdre dades en el procés.

```
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. Per això és necessari explicitar-ho, per forçar el canvi en casos en els que sabem que no es perdran dades perquè el valor es sufucientment petit o perquè podem assumir la pèrdua.

Exemples:

- En aquest cas sabem que el valor guardat dins el long es pot guardar dins un int, 20 és un valor acceptable, per tant podem forçar la conversió fent el càsting.

```
int n;

long l = 20;

n = (int) l;
```

- En aquest cas el valor de l'*int* no es pot emmagatzemar dins un *byte*. Ho podem forçar fent el càsting, però llavors el valor guardat dins el byte no tendrà gaire a veure amb l'original. Ja és responsabilitat del programador decidir si el resultat és acceptable o no.

```
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. A la versió final del codi no hauria d'aparèixer codi comentat.

En java tenim tres tipus de comentaris:

- Comentaris d'una línia

```
//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 expressió. Tenim dues instruccions, *if* i *switch*.

if

Permet avaluar qualsevol expressió booleana. Defineix un bloc de codi que s'executarà només si la condició, l'expressió booleana, és certa.

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

El cas general ens permet definir també un bloc de codi que s'executa només quan la condició s'avalua com a falsa

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

Si *condició* s'avalua a cert s'executaran *instruucions1* i si s'avalua a fals *instruccions2*.

switch

Aquesta instrucció ens permet avaluar una expressió i decidir que volem fer per a cada resultat possible de l'expressió.

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.

Per a cada resultat possible o que volguem tractar posam un bloc case amb el valor i el codi a executar. Darrera el case hi ha d'haver un valor concret, no hi pot haver una expressió.

Podem definir un cas com a *default* que reculli tots els valors de l'expressió que no hem recollit en cap *case*.

Al següent exemple posam un break a cada *case* perquè ens interessa que només mostri per pantalla el nom correcte del mes

```
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;
        }
    }
```

En canvi, en aquest exemple volem mostrar els dies que té cada mes. No fa falta repetir set vegades el codi d'assignar 31 dies a la variable, basta juntar els casos, i només afegir el codi al darrer. SI més és igual a 8, acabarà posant 31 dies a la variable.

```
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

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

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.

do-while

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

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.

for

L'utilitzam per generar valors que hem d'utilitzar al codi, o quan sabem el nombre de vegades que hem de repetir-lo.

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

Inicialitza la variable. Avalua la condició. SI es falsa acaba. Si es certa, executa les instruccions i realitza l'increment. Torna a avaluar la condició ...

El següent exemple mostra per pantalla els senceres del 0 al 9.

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

A la part d'inicialització podem inicialitzar més d'una variables separades per , . A la part d'increment també podem modificar el valor de més d'una variable. Per exemple, el següent codi mostra els valors de i del 0 al 9 i els de j del 10 a l'1.

La condició, en canvi, sempre ha de tornar un únic valor booleà.

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

Bifurcacions

Permeten rompre la seqüència lògica del programa. Normalment s'utilitzen per interrompre les iteracions.

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.

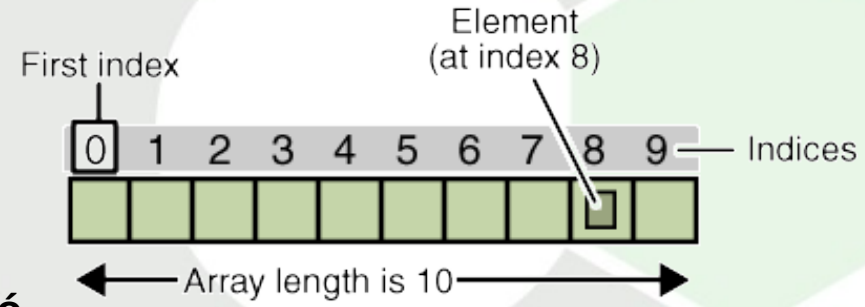
```
while(condició){
        instruccions1;
        if(condició) {
                continue;
        }
        instruccions2;
}
```

Arrays

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

Cada posició està indexada per un valor sencer. La primera casella té l'índex zero, per tant l'última tendrà l'índex igual a la longitud de l'array -1.

Aquest índex s'utilitza per accedir o modificar el valor guardat dins la casella corresponent.



Declaració

No crea la variable, únicament la declara. Darrera el tipus es posa un parell de claudàtors, [ ]

```
int[] notes;
```

Si volem un array multidimensional, de més d'una dimensió com una matriu, posam un parell de claudàtors per a cada dimensió.

```
Int[ ] [ ] caselles;
```

Creació

Reserva l'espai per l'array. Especificam la quantitat d'elements que contindrà. **No inicialitza l'array**. Podem especificar la longitud de cada dimensió amb un literal o una variable. Han de ser de tipus sencer.

```
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. Si imprimim la variable apareixerà una expressió estranya, no el contingut.

Inicialització

Abans de poder recuperar els valors guardats per l'array s'han d'assignar. Hem d'assignar els valors posició a posició, indicant el seu índex entre els claudàtors.

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

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

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

Propietats

- **length**: és una propietat de l'array. Torna la longitud de l'array.

```
System.out.println(noms.length)
```

Còpia d'arrays

La variable array en realitat conté una referència a l'array. Si executam el següent codi:

```
String[] alumnes = noms;
```

El que hem fet realment és posar un àlias a l'array de manera que si modificam la posició 1 d'*alumnes*

```
alumnes[1]=”Josep”;
```

llavors també haurem modificat la posició 1 de *noms*

```
System.out.println(noms[1]);  //Mostra Josep
```

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

```
String[alumnes]=new String[noms.length];
for(int index=0; index<alumnes.length; index++){
        alumnes[index]=noms[index];
}
```

o utilitzar

```
System.arraycopy(origen, posInicial, destí, posInicial, nombreElements);
System.arraycopy(noms,0, alumnes, 0, noms.length);
```

o, més senzill

```
String[ ] desti = Arrays.copyOf(origen, origen.length);

String[ ] alumnes = Arrays.copy(noms, noms.length);
```