1.3 Variables

-Què és una variable?

Una variable és com se li'n diu a una unitat per a guardar dades en un programa, el valor d'aquestes variables es guarda sobre un espai en la memòria RAM el qual es reserva per a guardar aquestes dades. Les variables s'utilitzen a la majoria de llenguatges de programació on a un nom se li asocia un contingut.

-Tipus de variables:

Hi han tres tipus de variables a Java:

- Variables locals.
- Variables de instancia.
- Variables estàtiques.

Variable Local

Una variable definida dintre d'un bloc se li'n diu variable local.

Aquestes variables es creen quan el bloc ingressat o mètode es crida i destrueix després de sortir del bloc o quan la crida torna del mètode.
L'abast de aquestes variables només existeix dins del bloc en el que es declara la variable, és a dir, podem accedir a aquestes variables només dins d'aquest bloc.

```
public class StudentDetails
{
    public void StudentAge()
    {   //variable local age
        int age = 0;
        age = age + 5;
        System.out.println("La edad del estudiante es : " + age);
    }

    public static void main(String args[])
    {
        StudentDetails obj = new StudentDetails();
        obj.StudentAge();
    }
}
```

Salida:

```
La edad del estudiante es : 5
```

En el programa anterior, la variable age és una variable local. Si intentem usar la variable age fora del mètode es mostrarà un error.

Variables de instancia

Las variables de instancia són variables no estàtiques i es declaren en una clase fora de qualsevol mètode, constructor o bloc

Com les variables d'instància es declaren en una clase, aquestes variables es creen quan un objecte de la clase es crea i es destrueix quan es destrueix l'objecte.

A diferencia de les variables locals, podem usar especificadors d'accés per a variables de instancia. Si no especifiquem cap especificador d'accés, s'utilitzarà l'especificador d'accés predeterminat.

```
import java.io.*;
class Points
{
    //Estas variables son variables de instancia.
    //Estas variables están en una clase y no están dentro de ninguna función/método
    int engPoints;
    int mathsPoints;
    int phyPoints;
}

class PointsDemo
{
    public static void main(String args[])
    {   //primer objeto
        Points obj1 = new Points();
        obj1.engPoints = 50;
        obj1.mathsPoints = 80;
        obj1.phyPoints = 90;

        //segundo objeto
        Points obj2 = new Points();
        obj2.engPoints = 80;
        obj2.mathsPoints = 60;
        obj2.phyPoints = 85;

        //mostrando puntos para el primer objeto
        System.out.println("Puntos para el primer objeto:");
        System.out.println(obj1.engPoints);
        System.out.println(obj1.mathsPoints);
        System.out.println(obj1.phyPoints);

        //mostrando puntos para el segundo objeto
        System.out.println("Puntos para el segundo objeto:");
        System.out.println(obj2.engPoints);
        System.out.println(obj2.mathsPoints);
        System.out.println(obj2.phyPoints);
    }
}
```

Salida:

```
Puntos para el primer objeto:
50
80
90
Puntos para el segundo objeto:
80
60
85
```

.

Com podem veure al programa anterior, les variables engPoints, mathsPoints, phyPoints; són variables de instancia. En cas de que tinguem varios objectes com al programa anterior, cada objecte tindrà les seves pròpies copies de variables de instancia.

Variables estàtiques

Les variables estàtiques també es coneixen com a variables de clase.

Aquestes variables es declaren de forma similar a las variables de instancia, la diferencia es que les variables estàtiques se declaren utilitzant la paraula clau dins de una clase fora de qualsevol constructor o bloc de mètodes.

A diferència de las variables de instancia, **només podem tenir una copia de una variable estàtica per classe**, independentment de quants objectes creem.

Les variables estàtiques es creen a l'inici de la execució del programa y se destrueixen automàticament quan finalitza la execució.

```
import java.io.*;
class Emp {

   // salario como variable estatica
   public static double salary;
   public static String name = "Alex";
}

public class EmpDemo
{
     public static void main(String args[]) {

      //acceder a la variable estatica sin objeto
      Emp.salary = 1000;
      System.out.println(Emp.name + " tiene un salario promedio de: " + Emp.salary);
   }

}
```

Salida:

```
Alex tiene un salario promedio de: 1000.0
```

1.4 Conversions de tipus de dades.

En Java es posible transformar el tipo de una variable u objeto en otro diferente al original con el que fue declarado. Este proceso se denomina "conversión", "moldeado" o "tipado" y es algo que debemos manejar con cuidado pues un mal uso de la conversión de tipos es frecuente que dé lugar a errores.

Una forma de realizar conversiones consiste en colocar el tipo destino entre paréntesis, a la izquierda del valor que queremos convertir de la forma siguiente: Tipo VariableNueva = (NuevoTipo) VariableAntigua;

Por ejemplo: int miNumero = (int) ObjetoInteger; char c = (char)System.in.read();

En el primer ejemplo, extraemos como tipo primitivo int el valor entero contenido en un campo del objeto Integer. En el segundo caso, la función read devuelve un valor int, que se convierte en un char debido a la conversión (char), y el valor resultante se almacena en la variable de tipo carácter c.

El tamaño de los tipos que queremos convertir es muy importante. No todos los tipos se convertirán de forma segura. Por ejemplo, al convertir un long en un int, el compilador corta los 32 bits superiores del long (de 64 bits), de forma que encajen en los 32 bits del int, con lo que si contienen información útil, ésta se perderá. Este tipo de conversiones que suponen pérdida de información se denominan “conversiones no seguras” y en general se tratan de evitar, aunque de forma controlada pueden usarse puntualmente.

De forma general trataremos de atenernos a la norma de que "en las conversiones debe evitarse la pérdida de información". En la siguiente tabla vemos conversiones que son seguras por no suponer pérdida de información.

TIPO ORIGEN	TIPO DESTINO
byte	double, float, long, int, char, short
short	double, float, long, int
char	double, float, long, int
int	double, float, long
long	double, float
float	Double

MÉTODO VALUEOF PARA CONVERSIÓN DE TIPOS

El método valueOf es un método sobrecargado aplicable a numerosas clases de Java y que permite realizar conversiones de tipos. Veamos algunos ejemplos de uso.

EXPRESIÓN	INTERPRETACIÓN aprenderaprogramar.com
miInteger = miInteger.valueOf (i)	Con i entero primitivo que se transforma en Integer
miInteger = miInteger.valueOf (miString)	El valor del String se transforma en Integer
miString = miString.valueOf (miBooleano)	El booleano se transforma en String “true” o “false”
miString = miString.valueOf (miChar)	El carácter (char) se transforma en String
miString = miString.valueOf (miDouble)	El double se transforma en String. Igualmente aplicable a float, int, long.

No todas las conversiones son posibles. Muchas veces por despiste los programadores escriben instrucciones de conversión incoherentes como miInteger = (int) miString;. El resultado en este caso es un error de tipo “Inconvertible types”. Un uso típico de valueOf es para convertir tipos primitivos en objetos.

1.5 Constants. Tipus i utilitat.

Una constant es declara igual que una variable, posant la paraula final abans del tipo (enlloc del tipo podem usar var). La declaració té lloc al principi del mètode main(), igual com fem en les variables.
El nom de les constants s'escriu en majúscules. Si és paraula composta les separarem en '_'.
El valor de les constants no pot canviar.

Se puede hacer una división de las constantes en tres clases:

- constantes literales (sin nombre)
- constantes declaradas (con nombre)
- constantes expresión

Constantes literales

Son valores de cualquier tipo que se utilizan directamente, no se declaran ya que no tienen nombre. En el siguiente ejemplo tienes un par de constantes literales (el 3, el 4, y el 3.1416):
VolumenEsfera := 4/3 * 3.1416 * Radio * Radio * Radio;
Constantes declaradas
También llamadas constantes con nombre, son las que se declaran en la sección const asignándoles un valor directamente. Por ejemplo:

```
const
Pi = 3.141592; (* valor real *)
Min = 0; (* entero *)
Max = 99; (* entero *)
Saludo = 'Hola'; (* cadena caract. *)
```

Constantes expresión

También se declaran en la sección constante, pero a estas no se les asigna un valor directamente, sino que se les asigna una expresión. Esta expresión se evalúa en tiempo de compilación y el resultado se le asigna a la constante. Ejemplo:

```
const
Min = 0;
Max = 100;
Intervalo = 10;
N = (Max - Min) div Intervalo;
Centro = (Max - Min) div 2;
```

1.6 Operadors del llenguatge de programació.

Un operador realiza una función, toma uno o más argumentos y devuelve un resultado. Cuando vimos las variables, definimos un tipo de dato con un conjunto de operaciones asociadas. Es de esperar que podamos realizar operaciones aritméticas con números y lógicas con los booleanos. Estas operaciones se representan mediante operadores.
Los operadores, al igual que los métodos, se pueden sobrecargar, es decir se puede redefinir su funcionalidad dependiendo de los tipos de datos de los operandos que reciba. Así, podemos indicar que el operador (+) realice una suma aritmética si los operandos que recibe son dos enteros y una concatenación si recibe una cadena y otro objeto.

Tipos de operadors

Operadores aritméticos

Realizan las operaciones aritméticas básicas: suma (+), resta (-), multiplicación (*) ,división (/) y módulo (%) para datos de tipo numérico, tanto enteros como reales. Estas son operaciones binarias porque admiten dos operandos.

Operadores relacionales

Revisando algunas definiciones matemáticas, nos enteramos que los números conforman un conjunto ordenado. Cada uno tiene una posición relativa. Sabemos que el 2 "es menor que" el 4 y que el 6 "es más grande que" el 1. Al comparar dos números, realizamos una función de relación.
En java disponemos de los operadores relacionales para verificar si se cumple una relación. Por ejemplo el operador de equivalencia (==) nos devuelve un valor de verdadero si los operandos son iguales. Estas operaciones comparan dos valores numéricos y devuelven un valor booleano.

Operador	Utilización	Resultado
>	A > B	verdadero si A es mayor que B
>=	A >= B	verdadero si A es mayor o igual que B
<	A < B	verdadero si A es menor que B
<=	A <= B	verdadero si A es menor o igual que B
==	A == B	verdadero si A es igual a B
!=	A != B	verdadero si A es distinto de B

Operadores booleanos

Como deben suponer, trabajan con operandos booleanos. Realizan las operaciones lógicas de conjunción (AND), disyunción (OR), negación (NOT) y la disyunción exclusiva (XOR).

Nombre	Operador	Utilización	Resultado
AND	&&	A && B	verdadero cuando A y B son verdaderos. Evaluación condicional.
OR	||	A || B	verdadero cuando A o B son verdaderos. Evaluación condicional.
NOT	!	!A	verdadero si A es falso.
AND	&	A & B	verdadero cuando A y B son verdaderos. Siempre evalúa ambos operandos.
OR	|	A | B	verdadero cuando A o B son verdaderos. Siempre evalúa ambos operandos.
XOR	^	A ^ B	verdadero cuando A y B son diferentes

Cada una de estas operaciones tiene asociada una tabla de verdad. Esto nos permite ver el resultado de un operador aplicado a las distintas combinaciones de valores que pueden tener los operandos.

Operadores de bits

Como sabrán, los datos en una computadora internamente se representan en código binario. El microprocesador solo entiende de ceros y unos. Luego, mediante una serie de procesos, nosotros vemos a este código ya transformado en números, caracteres, imágenes y sonido. Pero en realidad en la trastienda todo sigue siendo binario.

Operadores de asignación

Es el operador de asignación. Este aparece con un signo igual (=). Cambia el valor de la variable que está a la izquierda por un literal o el resultado de la expresión que se encuentra a la derecha.

Operadores de asignación

Operación	Operador	Utilización	Operación equivalente
Suma	+=	A += B	A = A + B
Resta	-=	A -= B	A = A - B
Multiplicación	*=	A *= B	A = A * B
División	/=	A /= B	A = A / B
Resto de división	%=	A %= B	A = A % B
Desplazamiento a la izquierda	<<=	A <<= B	A = A << B
Desplazamiento a la derecha	>>=	A >>= B	A = A >> B
Desplazamiento a la derecha sin signo	>>>=	A >>>= B	A = A >>> B
AND de bits	&=	A &= B	A = A & B
OR de bits	|=	A |= B	A = A | B
XOR de bits	^=	A ^= B	A = A ^ B

Operadores cast

En java se puede forzar un dato, variable o una expresión a convertirse o cambiarse a un nuevo tipo de dato.
El operador cast realiza este proceso, es decir convierte datos, variables o expresiones a un nuevo tipo de dato.

Tipus de dades simples

Java es un lenguaje de tipado estático, es decir, se define el tipo de dato de la variable a la hora de definir esta. Es por ello que todas las variables tendrán un tipo de dato asignado.
El lenguaje Java da de base una serie de tipos de datos primitivos.

- byte
- short
- int
- long
- float
- double
- boolean
- char

Es importante saber que estos son tipos de datos del lenguaje y que no representan objetos. Cosa que sí sucede con el resto de elementos del lenguaje Java.

byte

Representa un tipo de dato de 8 bits con signo. De tal manera que puede almacenar los valores numéricos de -128 a 127 (ambos inclusive).

short

Representa un tipo de dato de 16 bits con signo. De esta manera almacena valores numéricos de -32.768 a 32.767.

int

Es un tipo de dato de 32 bits con signo para almacenar valores numéricos. Cuyo valor mínimo es -231 y el valor máximo 231-1.

long

Es un tipo de dato de 64 bits con signo que almacena valores numéricos entre -263 a 263-1

float

Es un tipo dato para almacenar números en coma flotante con precisión simple de 32 bits.

double

Es un tipo de dato para almacenar números en coma flotante con doble precisión de 64 bits.

boolean

Sirve para definir tipos de datos booleanos. Es decir, aquellos que tienen un valor de true o false. Ocupa 1 bit de información.

char

Es un tipo de datos que representa a un carácter Unicode sencillo de 16 bits.

Dato Primitivo	Valor por Defecto
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'u0000'
String (o cualquier objeto)	null
boolean	false