GENERALITAT VALENCIANA
CONSELLERIA D'EDUCACIÓ, INVESTIGACIÓ, CULTURA I ESPORT

Fondo Social Europeo

Programación

UD 2: Introducción a JAVA

Ciclo Formativo de Grado Superior
Desarrollo de Aplicaciones Web

Francisco Pérez y Ferran Vilanova

El lenguaje de programación Java

¿Por qué Java en 1ºDAM?

Hay que escoger un Lenguaje Orientado a Objetos…

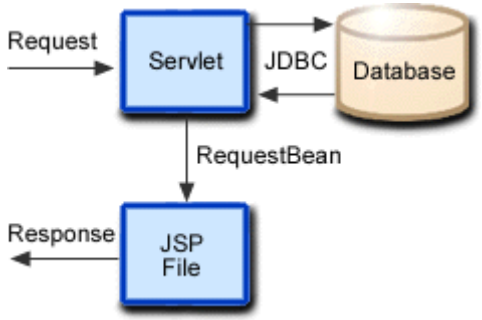
Concurso nacional ciclos formativos ProgramaMe: Java o C++

Entonces… ¿Java o C++?

El módulo Programación es común para 1º DAW y 1º DAM

2º DAW

Servlets en Java (MVC):



2º DAM

Programar para Android:

1.- Variables e identificadores

Una variable es una zona en la memoria del ordenador con un valor que puede ser almacenado para ser usado más tarde en el programa.

Las variables vienen determinadas por:

- un nombre, que permite al programa acceder al valor que contiene en memoria. Debe ser un identificador válido.

-un tipo de dato, que especifica qué clase de información guarda la variable en esa zona de memoria

- un rango de valores que puede admitir dicha variable.

1.- Variables e identificadores

Sirven para referirse tanto a objetos como a tipos primitivos.

Tienen que declararse antes de usarse:

```
tipo identificador;
```

```
int posicion;
```

Se puede inicializar mediante una asignación:

```
tipo identificador = valor;
```

```
int posicion = 0;
```

Definición de constantes:

```
final float PI = 3.14159f;
```

1.- Variables e identificadores

Se llama identificador al nombre que le damos a la variable.

Los identificadores:

- Nombran variables, funciones, clases y objetos.

- Comienza con una letra. Los siguientes caracteres pueden ser letras o dígitos.

- Se distinguen las mayúsculas de las minúsculas.

- No hay una longitud máxima establecida para el identificador.

2.- Palabras reservadas

En el lenguaje de programación Java se puede hacer uso de las palabras clave (*keywords*), también llamadas palabras reservadas, mostradas en la siguiente tabla. Dichas palabras, no pueden ser utilizadas como identificadores por los programadores para definir variables, constantes, etc.

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

true, false y null no son considerados palabras clave de Java, sino literales. Ahora bien, tampoco se pueden utilizar como indentificadores.

La descripción de la funcionalidad de todas las palabras reservadas se puede encontrar en:
https://www.abrirllave.com/java/palabras-clave.php

3.- Tipos de datos primitivos

Tipo	Descripción	Bytes	Rango	Valor por defecto
byte	Entero muy corto	1	-128 a 127	0
short	Entero corto	2	-32.768 a 32.767	0
int	Entero	4	-2.147.486.648 a 2.147.486.647 (-2^{31} a $2^{31}-1$)	0
long	Entero largo	8	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	0L
float	Número con punto flotante de precisión individual con hasta 7 dígitos significativos	4	+/-1.4E-45 (+/-1.4 times 10^{-45}) a +/-3.4E38 (+/-3.4 times 10^{38})	0.0f
double	Número con punto flotante de precisión doble con hasta 16 dígitos significativos	8	+/-4.9E-324 (+/-4.9 times 10^{-324}) a +/-1.7E308 (+/-1.7 times 10^{308})	0.0d
char	Carácter Unicode	\u0000 a \uFFFF	'\u0000'	
boolean	Valor verdadero o false	1	true o false	false

4.- Declaración e inicialización

Las declaraciones de variables pueden ir en cualquier parte del programa pero siempre antes de que la variable sea usada. Hay que tener cuidado con el rango de validez (scope) de la declaración.

Ejemplos:

```
int  i;
int j = 1;
double pi = 3.14159;
char c = 'a';
boolean estamosBien = true;
```

5.- Literales

Literal carácter

Un **literal carácter** puede escribirse como un carácter entre comillas simples como 'a', 'ñ', 'Z', 'p', etc. o por su código de la tabla Unicode, anteponiendo la secuencia de escape '****' si el valor lo ponemos en octal o '**\u**' si ponemos el valor en hexadecimal.

Por ejemplo, si sabemos que tanto en ASCII como en Unicode, la letra A (mayúscula) es el símbolo número 65, y que 65 en octal es 101 y 41 en hexadecimal, podemos representar esta letra como '**\101**' en octal y '**\u0041**' en hexadecimal.

Existen unos caracteres especiales que se representan utilizando secuencias de escape:

Secuencia de escape	Significado	Secuencia de escape	Significado
\b	Retroceso	\r	Retorno de carro
\t	Tabulador	**\"**	Carácter comillas dobles
\n	Salto de línea	**\'**	Carácter comillas simples
\f	Salto de página	****	Barra diagonal

5.- Literales

Literales de cadenas de caracteres

Los **literales de cadenas de caracteres** se indican entre comillas dobles.

Al construir una cadena de caracteres se puede incluir cualquier carácter Unicode excepto un carácter de retorno de carro, por ejemplo en la siguiente instrucción utilizamos la secuencia de escape **\”** para escribir dobles comillas dentro del mensaje:

String texto = “Pedro dijo: \"Hoy hace un día fantástico…\"";

En el ejemplo anterior de tipos enumerados ya estábamos utilizando secuencias de escape, para introducir un salto de línea en una cadena de caracteres, utilizando el carácter especial **\n**.

*Normalmente, los objetos en Java deben ser creados con la orden new. Sin embargo, los literales **String** no lo necesitan ya que son objetos que se crean implícitamente por Java.*

6.- Constantes

Constantes o variables finales

Son aquellas variables cuyo valor no cambia a lo largo de todo el programa.

Declaración de constantes en Java

```
final double PI = 3.1415926536;
```

En nombre de las constantes se deben ser en mayúsculas.

7.- Operadores y expresiones

Operadores Aritméticos:

Suma +

Resta -

Multiplicación *

División /

Resto de la División %

7.- Operadores y expresiones

Operadores de Asignación:

El principal es '=' pero hay más operadores de asignación con distintas funciones.

'+=': op1 += op2	op1 = op1 + op2
'-=': op1 -= op2	op1 = op1 - op2
'*=': op1 *= op2	op1 = op1 * op2
'/=': op1 /= op2	op1 = op1 / op2
'%=': op1 %= op2	op1 = op1 % op2

7.- Operadores y expresiones

Operadores Relacionales:

Permiten comparar variables según relación de igualdad/desigualdad o relación mayor/menor. Devuelven siempre un valor boolean.

```
'>':  Mayor que
'<':  Menor que
'==': Iguales
'!=': Distintos
'>=': Mayor o igual que
'<=': Menor o igual que
```

7.- Operadores y expresiones

Operadores Lógicos:

Nos permiten construir expresiones lógicas.

'&&' : devuelve true si ambos operandos son true.

'||' : devuelve true si alguno de los operandos son true.

'!' : Niega el operando que se le pasa.

A	B	A && B	A || B	! A
true	**true**	true	true	false
true	**false**	false	true	false
false	**true**	false	true	true
false	**false**	false	false	true

7.- Operadores y expresiones

Operador de Concatenación:

Operador de concatenación con cadena de caracteres '+':

Ejemplo:

```
System.out.println("El total es " + result + " unidades.");
```

Operadores Incrementales:

Son los operadores que nos permiten incrementar las variables en una unidad. Prefija ó sufija.

'++'
'--'

8.- Conversiones de tipo

El casting es un procedimiento para transformar una variable primitiva de un tipo a otro.

También se utiliza para transformar un objeto de una clase a otra clase siempre y cuando haya una relación de herencia entre ambas.

*En este tema nos centraremos en el primer tipo de casting.

Dentro de este casting de variables primitivas se distinguen dos clases:

Casting implícito

Casting explícito

Las conversiones de tipo se realizan para hacer que el resultado de una expresión sea del tipo que nosotros deseamos.

8.- Conversiones de tipo

Casting implícito o automático

Cuando a una variable de un tipo se le asigna un valor de otro tipo numérico con menos bits para su representación, se realiza una conversión automática.

En ese caso, el valor se dice que es promocionado al tipo más grande (el de la variable), para poder hacer la asignación.

También se realizan conversiones automáticas en las operaciones aritméticas, cuando estamos utilizando valores de distinto tipo, el valor más pequeño se promociona al valor más grande, ya que el tipo mayor siempre podrá representar cualquier valor del tipo menor (por ejemplo, de int a long o de float a double).

En este caso no se necesita escribir código para que la conversión se lleve a cabo. Ocurre cuando se realiza lo que se llama una conversión ancha (*widening casting*), es decir, cuando se coloca un valor pequeño en un contenedor grande.

Ejemplo:

```
int  num1 = 100;
long num2 = num1; //Un int "cabe" en un long
```

8.- Conversiones de tipo

Casting explícito

Cuando hacemos una conversión de un tipo con más bits a un tipo con menos bits.

En estos casos debemos indicar que queremos hacer la conversión de manera expresa, ya que se puede producir una pérdida de datos y hemos de ser conscientes de ello. Este tipo de conversiones se realiza con el **operador cast**.

El operador cast es un operador unario que se forma colocando delante del valor a convertir el tipo de dato entre paréntesis. Tiene la misma precedencia que el resto de operadores unarios y se asocia de izquierda a derecha. El formato general para indicar que queremos realizar la conversión es:

(tipo) valor_a_convertir

En el casting explícito sí es necesario escribir código. Ocurre cuando se realiza una conversión estrecha (*narrowing casting*), es decir, cuando se coloca un valor grande en un contenedor pequeño.

```
int num1   = 100;
short num2 = (short) num1;  //Casting explícito:
                            //short tiene menor rango que int
```

8.- Conversiones de tipo

Debemos tener en cuenta que un valor numérico nunca puede ser asignado a una variable de un tipo menor en rango, si no es con una conversión explícita.

Ejemplo:

```
int a;
byte b;

a = 12; // no se realiza conversión alguna
b = 12; // se permite porque 12 está dentro del rango permitido de valores para b
b = a;  // error, no permitido (incluso aunque 12 podría almacenarse en un byte)

byte b = (byte) a; // Correcto, forzamos conversión explícita
```

En el ejemplo anterior vemos un caso típico de error de tipos, ya que estamos intentando asignarle a **b** el valor de **a**, siendo **b** de un tipo más pequeño. Lo correcto es promocionar **a** al tipo de datos **byte**, y entonces asignarle su valor a la variable **b**.

8.- Conversiones de tipo

Conversión de números en Coma flotante (float, double) a enteros (int)

Cuando convertimos números en coma flotante a números enteros, la parte decimal se trunca (redondeo a cero). Si queremos hacer otro tipo de redondeo, podemos utilizar, entre otras, las siguientes funciones:

Math.round(num): Redondeo al siguiente número entero.
Math.floor(num): Redondeo al numero actual.

Ejemplo:

```
double num = 3.5;

x = Math.round(num);   // x = 4
z = Math.floor(num);   // z = 3
```

8.- Conversiones de tipo

Conversiones de tipo con cadenas de caracteres (String)

Para convertir cadenas de texto a otros tipos de datos se utilizan las siguientes funciones:

```
num = Byte.parseByte(cad);
num = Short.parseShort(cad);
num = Integer.parseInt(cad);
num = Long.parseLong(cad);
num = Float.parseFloat(cad);
num = Double.parseDouble(cad);
```

Por ejemplo, si hemos leído de teclado un número que está almacenado en una variable de tipo String llamada cadena, y lo queremos convertir al tipo de datos byte, haríamos lo siguiente:

```
byte n = Byte.parseByte(cadena);
```

9.- Comentarios

Comentarios de una sola línea:

Utilizaremos el delimitador // para introducir comentarios de sólo una línea.

Ejemplo:

```
// comentario de una sola línea
```

Comentarios de múltiples líneas:

Para introducir este tipo de comentarios, utilizaremos una barra inclinada y un asterisco (/*), al principio del párrafo y un asterisco seguido de una barra inclinada (*/) al final del mismo.

Ejemplo:

```
/* Esto es un comentario
 * de varias líneas.
 * En concreto, de 3 líneas. */
```