Práctica 1. FUNDAMENTOS Y SINTAXIS DEL LENGUAJE

Objetivo: Crear programas que implementen variables y constantes de diferentes tipos de datos, expresiones y estructuras de control de flujo.

Introducción:

Los lenguajes de programación tienen elementos básicos que se utilizan como bloques constructivos, así como reglas para que esos elementos se combinen. Estas reglas se denominan sintaxis del lenguaje. Solamente las instrucciones sintácticamente correctas pueden ser interpretadas por la computadora y los programas que contengan errores de sintaxis son rechazados por la máquina.

La sintaxis de un lenguaje de programación se define como el conjunto de reglas que deben seguirse al escribir el código fuente de los programas para considerarse como correctos para ese lenguaje de programación.

Los elementos básicos constructivos de un programa son:

- Palabras reservadas (propias de cada lenguaje).
- Identificadores (nombres de variables, nombres de funciones, nombre del programa, etc.)
- Caracteres especiales (alfabeto, símbolos de operadores, delimitadores, comentarios, etc.)
- Expresiones.
- Instrucciones.

Tipos de datos

Los **tipos de datos** hacen referencia al tipo de información que se trabaja, donde la unidad mínima de almacenamiento es el dato, también se puede considerar como el rango de valores que puede tomar una variable durante la ejecución del programa. Un tipo de dato define un conjunto de valores y las operaciones sobre estos valores.

En Java existen dos grupos de tipos de datos, **tipos primitivos y tipos referencia**.

Se llaman **tipos primitivos** a aquellos datos sencillos que contienen los tipos de información más habituales: valores booleanos, caracteres y valores numéricos enteros o de punto flotante.

Los t**ipos de dato referencia** representan datos compuestos o estructuras, es decir, referencias a objetos. Estos tipos de dato almacenan las direcciones de memoria y no el valor en sí (similares a los apuntadores en C). Una referencia a un objeto es la dirección de un área en memoria destinada a representar ese objeto.

Tipos de datos primitivos en Java

Tipo primirtivo	Descripción	Tamaño	Mínimo	Máximo	Envoltorio
boolean	Valos binario	-	-	-	Boolean
char	Carácter unicode	16 bits	unicode 0	unicode 2^{16-1}	Character
byte	Entero con signo	8 bits	-128	+127	Byte
short	Entero con signo	16 bits	-2^{15}	$+2^{15-1}$	Short
int	Entero con signo	32 bits	-2^{31}	$+2^{31-1}$	Integer
long	Entero con signo	64 bits	-2^{63}	$+2^{63-1}$	Long
float	Real simple precisión	32 bits	$\pm 3.40282347\ e^{+38}$ $\pm 1.40239486\ e^{-45}$		Float
double	Real doble precisión	64 bits	$\pm 1.79769313486231570\ e^{+308}$ $\pm 4.94065645841246544\ e^{-324}$		Double

Variables

Una **variable** es un nombre que contiene un valor que puede cambiar a lo largo del programa. De acuerdo con el tipo de información que contienen, en Java hay dos tipos principales de variables:

1. Variables de tipos primitivos.
2. Variables referencia.

Las variables pueden ser:

Variables miembro de una clase: Se definen en una clase, fuera de cualquier método; pueden ser tipos primitivos o referencias.

Variables locales: Se definen dentro de un método o más en general dentro de cualquier bloque entre llaves { }. Se crean en el interior del bloque y se destruyen al finalizar dicho bloque.

Una variable se define especificando el tipo y el nombre de dicha variable. Estas variables pueden ser tanto de tipos primitivos como referencias a objetos de alguna clase perteneciente al API de Java o generada por el usuario.

```
tipoDeDato nombreVariable;
```

Ejemplo:

```
int miVariable; float area; char letra;
String cadena; MiClase prueba;
```

Si no se especifica un valor en su declaración, las variables primitivas se inicializan en cero (salvo boolean y char, que se inicializan a false y '\0'). Análogamente las variables de tipo referencia son inicializadas por defecto a un valor especial: null.

Es importante distinguir entre la referencia a un objeto y el objeto mismo. Una referencia es una variable que indica dónde está guardado un objeto en la memoria (a diferencia de C/C++, Java no permite acceder al valor de la dirección, pues en este lenguaje se han eliminado los apuntadores). Al declarar una referencia todavía no se encuentra "apuntando" a ningún objeto en particular (salvo que se cree explícitamente un nuevo objeto en la declaración), y por eso se le asigna el valor null.

Si se desea que esta referencia apunte a un nuevo objeto es necesario crear el objeto utilizando el operador ***new***. Este operador reserva espacio en la memoria para ese objeto (variables y funciones). También es posible igualar la referencia declarada a otra referencia a un objeto existente previamente.
Ejemplo:

```
MyClass unaRef;
unaRef = new MyClass( );
MyClass segundaRef = unaRef;
```

Un tipo particular de referencias son los arrays o arreglos, sean éstos de variables primitivas (por ejemplo, de enteros) o de objetos. En la declaración de una referencia de tipo array hay que incluir los corchetes [ ]. Ejemplo:

```
int [ ] vector;
vector = new int[10];
MyClass [ ] lista = new MyClass[5];
```

En Java todas las variables deben estar incluidas en una clase. En general las variables declaradas dentro de llaves { }, es decir, dentro de un bloque, son visibles y existen dentro de estas llaves. Por ejemplo las variables declaradas al principio de un método existen mientras se ejecute el método; las variables declaradas dentro de un bloque if no serán válidas al finalizar las sentencias correspondientes a dicho if y las variables miembro de una clase (es decir declaradas entre las llaves { } de la clase pero fuera de cualquier método) son válidas mientras existe el objeto de la clase.

Constantes

Una constante es una variable cuyo valor no puede ser modificado. Para definir una constante en Java se utiliza la palabra reservada ***final***, delante de la declaración del tipo, de la siguiente manera:

```
final tipoDato nombreDeConstante = valor;
```

Ejemplo:

```
final double PI = 3.1416;
```

Entrada y salida de datos por consola

Una de las operaciones más habituales que tiene que realizar un programa es intercambiar datos con el exterior. Para ello el API de java incluye una serie de clases que permiten gestionar la

entrada y salida de datos en un programa, independientemente de los dispositivos utilizados para el envío/recepción de datos.

Para el envío de datos al exterior se utiliza un flujo de datos de impresión o print stream.
Esto se logra usando la siguiente expresión:

```
System.out.println("Mi mensaje");
```

De manera análoga, para la recepción o lectura de datos desde el exterior se utiliza un flujo de datos de entrada o input stream. Para lectura de datos se utiliza la siguiente sintaxis:

```
Scanner sc = new Scanner(System.in);
String s = sc.next( ); //Para cadenas
int x = sc.nextInt( ); //Para enteros
```

Para usar la clase Scanner se debe incluir al inicio del archivo la siguiente línea:

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

Al finalizar su uso se debe cerrar el flujo usando el método close. Para el ejemplo:

```
sc.close( );
```

Estas clases y sus métodos se verán más a detalle en la práctica de Manejo de archivos.

Operadores

Operadores aritméticos: Son operadores binarios (requieren siempre dos operandos) que realizan las operaciones aritméticas habituales: suma (+), resta (-), multiplicación (*), división (/) y resto de la división o módulo (%).

Operadores de asignación: Los operadores de asignación permiten asignar un valor a una variable. El operador de asignación por excelencia es el operador igual (=). La forma general de las sentencias de asignación con este operador es:

```
variable = expression;
```

Java dispone de otros operadores de asignación. Se trata de versiones abreviadas del operador (=) que realizan operaciones "acumulativas" sobre una variable.

Operador	Valor previo	Operación	Resultado
=	**c** vale 30	**c = 15;**	**c** vale 15
+=	**c** vale 10	**c += 12;**	**c** vale 22
-=	**c** vale 4	**c -= 6;**	**c** vale -2
***=**	**c** vale 2	**c *= 15;**	**c** vale 30
/=	**c** vale 27	**c /= 3;**	**c** vale 9
%=	**c** vale 9	**c %= 4;**	**c** vale 1

Operadores unarios: Los operadores más (+) y menos (-) unarios sirven para mantener o cambiar el signo de una variable, constante o expresión numérica. Su uso en Java es el estándar de estos operadores.

Operador instanceof: El operador instanceof permite saber si un objeto pertenece o no a una determinada clase. Es un operador binario cuya forma general es:

```
objectName instanceof ClassName
```

Este operador devuelve true o false según el objeto pertenezca o no a la clase.

Operador condicional ?: Este operador, tomado de C/C++, permite realizar bifurcaciones condicionales sencillas. Su forma general es la siguiente:

```
booleanExpression ? res1 : res2
```

Donde se evalúa booleanExpression y se devuelve res1 si el resultado es true y res2 si el resultado es false. Es el único operador ternario (tres argumentos) de Java. Como todo operador que devuelve un valor puede ser utilizado en una expresión.

Operadores incrementales: Java dispone del operador incremento (++) y decremento (--). El operador (++) incrementa en una unidad la variable a la que se aplica, mientras que (--) la reduce en una unidad. Estos operadores se pueden utilizar de dos formas:

- Precediendo a la variable (por ejemplo: ++i). En este caso primero se incrementa la variable y luego se utiliza (ya incrementada) en la expresión en la que aparece.
- Siguiendo a la variable (por ejemplo: i++). En este caso primero se utiliza la variable en la expresión (con el valor anterior) y luego se incrementa.

La actualización de contadores en ciclos for es una de las aplicaciones más frecuentes de estos operadores.

Operadores relacionales: Los operadores relacionales sirven para realizar comparaciones de igualdad, desigualdad y relación de menor o mayor. El resultado de estos operadores es siempre un valor boolean (true o false) según se cumpla o no la relación considerada.

Operador	Utilización	El resultado es true
>	op1 > op2	si op1 es mayor que op2
>=	op1 >= op2	si op1 es mayor o igual que op2
<	op1 < op2	si op1 es menor que op2
<=	op1 <= op2	si op1 es menor o igual que op2
==	op1 == op2	si op1 y op2 son iguales
!=	op1 != op2	si op1 y op2 son diferentes

Estos operadores se utilizan con mucha frecuencia en las estructuras de control.

Operadores lógicos: Los operadores lógicos se utilizan para construir expresiones lógicas, combinando valores lógicos (true y/o false) o los resultados de los operadores relacionales.

Operador	Nombre	Utilización	Resultado
&&	AND	op1 && op2	true si op1 y op2 son true. Si op1 es false ya no se evalúa op2
||	OR	op1 || op2	true si op1 u op2 son true. Si op1 es true ya no se evalúa op2
!	negación	!op	true si op es false y false si op es true
&	AND	op1 & op2	true si op1 y op2 son true. Siempre se evalúa op2
|	OR	op1 | op2	true si op1 u op2 son true. Siempre se evalúa op2

Debe notarse que en ciertos casos el segundo operando no se evalúa porque ya no es necesario (si ambos tienen que ser true y el primero es false, ya se sabe que la condición de que ambos sean true no se va a cumplir). Esto puede traer resultados no deseados y por eso se han añadido los operadores (&) y (|) que garantizan que los dos operandos se evalúan siempre.

Operador de concatenación de cadenas de caracteres (+): El operador más (+) se utiliza también para concatenar cadenas de caracteres. Por ejemplo, para escribir una cantidad con un rótulo y valores puede utilizarse la sentencia:

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

Donde el operador de concatenación se utiliza dos veces para construir la cadena de caracteres que se desea imprimir por medio del método ***println( )***. La variable numérica result es convertida automáticamente por Java en cadena de caracteres para poderla concatenar. En otras ocasiones se deberá llamar explícitamente a un método para que realice esta conversión.

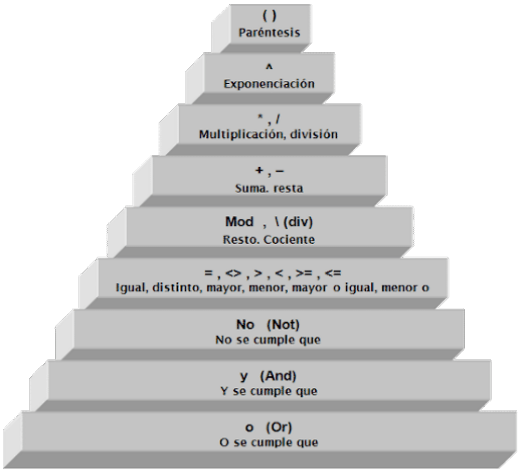
Operadores a nivel de bits: Java dispone también de un conjunto de operadores que actúan a nivel de bits. Las operaciones de bits se utilizan con frecuencia para definir señales o flags, esto es, variables de tipo entero en las que cada uno de sus bits indica si una opción está activada o no.

Operador	Utilización	Resultado
>>	op >> op2	Desplaza los bits de op1 a la derecha una distancia op2
<<	op1 << op2	Desplaza los bits de op1 a la izquierda una distancia op2
>>>	op1 >>> op2	Desplaza los bits de op1 a la derecha una distancia op2 (positiva)
&	op1 & op2	Operador AND a nivel de bits
|	op1 | op2	Operador OR a nivel de bits
^	op1 ^ op2	Operador XOR a nivel de bits (1 si sólo uno de los operadores es 1)
~	~op2	Operador complemento (invierte el valor de cada bit)

Operador	Utilización	Equivalente a
&=	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
>>>=	op1 >>>= op2	op1 = op1 >>> op2

Precedencia de operadores

El **orden** en que se realizan las operaciones es fundamental para determinar el resultado de una expresión. La siguiente lista muestra el orden en que se ejecutan los distintos operadores en una sentencia, de mayor a menor precedencia:



En Java, todos los operadores binarios (excepto los operadores de asignación) se evalúan de izquierda a derecha. Los operadores de asignación se evalúan de derecha a izquierda, lo que significa que el valor de la derecha se copia sobre la variable de la izquierda.

Las **estructuras de control** permiten modificar el flujo de ejecución de las instrucciones de un programa. Todas las estructuras de control tienen un único punto de entrada. Las estructuras de control se pueden clasificar en: secuenciales, transferencia de control e iterativas. Básicamente lo que varía entre las estructuras de control de los diferentes lenguajes es su sintaxis, cada lenguaje tiene una sintaxis propia para expresar la estructura.

La sintaxis de Java coincide prácticamente con la utilizada en C/C++, lo que hace que para un programador de C/C++ no represente ninguna dificultad adicional.
Sentencias o expresiones: Una expresión es un conjunto de variables unidos por operadores. Son órdenes que se le dan a la computadora para que realice una tarea determinada. Una sentencia es una expresión que acaba en punto y coma (;). Se permite incluir varias sentencias en una línea, aunque lo habitual es utilizar una línea para cada sentencia.
Ejemplo:

```
i = 0; j = 5; x = i + j; // Línea compuesta de tres sentencias
```

Estructuras de selección: Las estructuras de selección o bifurcaciones permiten ejecutar una de entre varias acciones en función del valor de una expresión lógica o relacional. Se tratan de estructuras muy importantes ya que son las encargadas de controlar el flujo de ejecución de un programa. Existen dos bifurcaciones diferentes: if y switch.

IF / IF-ELSE: Tiene la forma siguiente:

```
if (booleanExpression) {
      statements1;
} else {
      statements2;
}
```

Si se desea introducir más de una expresión de comparación se usa if / else if. Si la primera condición no se cumple, se compara la segunda y así sucesivamente. En el caso de que no se cumpla ninguna de las comparaciones se ejecutan las sentencias correspondientes al else.

```
if (booleanExpression1) {
      statements1;
} else if (booleanExpression2) {
      statements2;
} else {
      statements3;
}
```

SWITCH: Su forma general es la siguiente:

```
switch (expression) {
      case value1:
            statements1;
      break;
      case value2:
            statements2;
      break;
      case value3:
            statements3;
      break;
      [default:
            statements4;]
}
```

Estructuras de repetición: Las estructuras de repetición, lazos, ciclos o bucles se utilizan para realizar un proceso repetidas veces.

While: Las sentencias statements se ejecutan mientras booleanExpression sea true.

```
while (booleanExpression) {
      statements;
}
```

Do-While: Es similar al ciclo while pero con la particularidad de que el control está al final del ciclo.

```
do {
      statements
} while (booleanExpression);
```

For: La forma general del for es la siguiente:

```
for (initialization; booleanExpression; increment) {
      statements;
}
```

Break y Continue

La sentencia break es válida tanto para las bifurcaciones como para los ciclos. Hace que se salga inmediatamente del ciclo o bloque que se está ejecutando, sin realizar la ejecución del resto de las sentencias.

La sentencia continue se utiliza en los ciclos (no en bifurcaciones). Finaliza la iteración "*i*" que en ese momento se está ejecutando (no ejecuta el resto de sentencias que hubiera hasta el final del bucle). Vuelve al comienzo del bucle y comienza la siguiente iteración (*i+1*).

Ejercicio: Hacer un menú que contenga las siguientes las siguientes opciones.
1 Captura de nombre y calificaciones
2 Imprimir calificaciones y promedio de calificaciones
3 Imprimir calificaciones y calificación más baja
4 Imprimir calificaciones y calificación más alta
5 Salir

Actividades:

- Crear variables y constantes de diferentes tipos de datos.
- Crear diversas expresiones (operadores, declaraciones, etc.).
- Implementar estructuras de control de flujo (if/else, switch, for, while, etc.).
- Hacer un reporte con portada de hoja completa que incluya el código fuente y una captura de pantalla de la práctica funcionando.
- Si la práctica contiene menú, se agregará una captura de pantalla por cada opción implementada.
- Subir a blackboard reporte y código fuente.