Elementos básicos del lenguaje

Elementos básicos del lenguaje

VISIÓN GENERAL Y ELEMENTOS BÁSICOS DEL LENGUAJE.

ÍNDICE.

Hola Mundo!

PRIMEROS PASOS CON JAVA: HOLA MUNDO.

Una mínima aplicación en Java.

La aplicación más pequeña posible es la que simplemente imprime un mensaje en la pantalla. Tradicionalmente, el mensaje suele ser "Hola Mundo!". Esto es justamente lo que hace el siguiente fragmento de código:

```
// Aplicación HolaMundo de ejemplo
//
class HolaMundo {
public static void main( String args[] ) {
System.out.println( "Hola Mundo!" );
}
}
```

HolaMundo.

Hay que ver en detalle la aplicación anterior, línea a línea. Esas líneas de código contienen los componentes mínimos para imprimir Hola Mundo! en la pantalla. Es un ejemplo muy simple, que no instancia objetos de ninguna otra clase; sin embargo, accede a otra clase incluida en el JDK.

// Aplicación HolaMundo de ejemplo
//

Estas dos primeras líneas son comentarios. Hay tres tipos de comentarios en Java, // es un comentario orientado a línea.

class HolaMundo {

Esta línea declara la clase HolaMundo. El nombre de la clase especificado en el fichero fuente se utiliza para crear un fichero nombredeclase.class en el directorio en el que se compila la aplicación. En este caso, el compilador creará un fichero llamado HolaMundo.class.

public static void main(String args[]) {

Esta línea especifica un método que el intérprete Java busca para ejecutar en primer lugar. Igual que en otros lenguajes, Java utiliza una palabra clave main para especificar la primera función a ejecutar. En este ejemplo tan simple no se pasan argumentos.

public significa que el método main() puede ser llamado por cualquiera, incluyendo el intérprete Java.

static es una palabra clave que le dice al compilador que main se refiere a la propia clase HolaMundo y no a ninguna instancia de la clase. De esta forma, si alguien intenta hacer otra instancia de la clase, el método main() no se instanciaría.

void indica que main() no devuelve nada. Esto es importante ya que Java realiza una estricta

comprobación de tipos, incluyendo los tipos que se ha declarado que devuelven los métodos.

args[] es la declaración de un array de Strings. Estos son los argumentos escritos tras el nombre de la clase en la línea de comandos:

java HolaMundo arg1 arg2 ...

System.out.println("Hola Mundo!");

Esta es la funcionalidad de la aplicación. Esta línea muestra el uso de un nombre de clase y método. Se usa el método println() de la clase out que está en el paquete System.

El método println() toma una cadena como argumento y la escribe en el stream de salida estándar; en este caso, la ventana donde se lanza la aplicación. La clase PrintStream tiene un método instanciable llamado println(), que lo que hace es presentar en la salida estándar del Sistema el argumento que se le pase. En este caso, se utiliza la variable o instancia de out para acceder al método.

Compilación y Ejecución de HolaMundo

A continuación se puede ver el resultado de esta primera y sencilla aplicación Java en pantalla. Se genera un fichero con el código fuente de la aplicación, se compilará y se utilizará el intérprete Java para ejecutarlo.

Ficheros Fuente Java

Los ficheros fuente en Java terminan con la extensión ".java". Crear un fichero utilizando cualquier editor de texto ascii que tenga como contenido el código de las ocho líneas de nuestra mínima aplicación, y salvarlo en un fichero con el nombre de HolaMundo.java. Para crear los ficheros con código fuente Java no es necesario un procesador de textos, aunque puede utilizarse siempre que tenga salida a fichero de texto plano o ascii, sino que es suficiente con cualquier otro editor.

Compilación

El compilador javac se encuentra en el directorio bin por debajo del directorio java, donde se haya instalado el JDK. Este directorio bin, si se han seguido las instrucciones de instalación, debería formar parte de la variable de entorno PATH del sistema. Si no es así, tendría que revisar la Instalación del JDK. El compilador de Java traslada el código fuente Java a byte-codes, que son los componentes que entiende la Máquina Virtual Java que está incluida en los navegadores con soporte Java y en appletviewer.

Una vez creado el fichero fuente HolaMundo.java, se puede compilar con la línea siguiente:

javac HolaMundo.java

Si no se han cometido errores al teclear ni se han tenido problemas con el path al fichero fuente ni al compilador, no debería aparecer mensaje alguno en la pantalla, y cuando vuelva a aparecer el prompt del sistema, se debería ver un fichero HolaMundo.class nuevo en el directorio donde se encuentra el fichero fuente.

Ejecución.

Para ejecutar la aplicación HolaMundo, hemos de recurrir al intérprete java, que también se encuentra

en el directorio bin, bajo el directorio en donde se haya instalado el JDK. Se ejecutará la aplicación con la línea:

java HolaMundo

y debería aparecer en pantalla la respuesta de Java:

Hola Mundo!

Cuando se ejecuta una aplicación Java, el intérprete Java busca e invoca al método main() de la clase cuyo nombre coincida con el nombre del fichero .class que se indique en la línea de comandos. En el ejemplo, se indica al Sistema Operativo que arranque el intérprete Java y luego se indica al intérprete Java que busque y ejecute el método main() de la aplicación Java almacenada en el fichero HolaMundo.class.

Problemas de compilación.

A continuación se encuentra una lista de los errores más frecuentes que se presentan a la hora de compilar un fichero con código fuente Java, tomando como base los errores provocados sobre la mínima aplicación Java que se está utilizando como ejemplo, pero también podría generalizarse sin demasiados problemas.

javac: Command not found

No se ha establecido correctamente la variable PATH del sistema para el compilador javac. El compilador javac se encuentra en el directorio bin, que cuelga del directorio donde se haya instalado el JDK (Java Development Kit).

HolaMundo.java:3: Method printl(java.lang.String) not found in class java.io.PrintStream.
System.out.printl("HolaMundo!);

Error tipográfico, el método es println no printl.

In class HolaMundo: main must be public and static

Error de ejecución, se olvidó colocar la palabra static en la declaración del método main de la aplicación.

%Can´t find class HolaMundo

Este es un error muy sutil. Generalmente significa que el nombre de la clase es distinto al del fichero que contiene el código fuente, con lo cual el fichero nombre_fichero.class que se genera es diferente del que cabría esperar. Por ejemplo, si en el fichero de código fuente de la aplicación HolaMundo.java se coloca en vez de la declaración actual de la clase HolaMundo, la línea:

class Holamundo {

se creará un fichero Holamundo.class, que es diferente del HolaMundo.class, que es el nombre esperado de la clase; la diferencia se encuentra en la a minúscula y mayúscula.

Comentarios

En Java hay tres tipos de comentarios:

// comentarios para una sola línea

**/*

comentarios de una o más líneas**

***/**

/ comentario de documentación, de una o más líneas**

***/**

Los dos primeros tipos de comentarios son los que todo programador conoce y se utilizan del mismo modo. Los comentarios de documentación, colocados inmediatamente antes de una declaración (de variable o función), indican que ese comentario ha de ser colocado en la documentación que se genera automáticamente cuando se utiliza la herramienta de Java, javadoc, no disponible en otros lenguajes de programación. Dichos comentarios sirven como descripción del elemento declarado permitiendo generar una documentación de las clases que se va construyendo al mismo tiempo que se genera el código de la aplicación.

En este tipo de comentario para documentación, se permite la introducción de algunos tokens o palabras clave, que harán que la información que les sigue aparezca de forma diferente al resto, permitiendo la incorporación de información útil, que luego se podrá ver en formato html sobre cualquier navegador.

javadoc permite también colocar códigos o tags de html embebidos en los comentarios. Estos códigos son procesados también a la hora de generar el documento html.

También se puede utilizar html como en cualquier otro documento Web para formatear el texto en descripciones, como por ejemplo:

```
/**
* Aquí <em>debes</em> introducir la lista:
* <ol>
* <li> Primer Elemento
* <li> Segundo Elemento
* <li> Tercer Elemento
* </ol>
*/
```

Dentro de los comentarios de documentación, los asteriscos al comienzo de las líneas y los espacios hasta la primera letra no se tienen en cuenta. La salida del ejemplo anterior sería:

Aquí debes introducir la lista:
1. Primer Elemento
2. Segundo Elemento

3. Tercer Elemento

Identificadores

Los **identificadores** nombran variables, funciones, clases y objetos; cualquier cosa que el programador necesite identificar o usar.

Reglas para la creación de identificadores:

- Java hace **distinción** entre **mayúsculas** y **minúsculas**, por lo tanto, nombres o identificadores como var1, Var1 y VAR1 son distintos.
- Pueden estar formados por cualquiera de los **caracteres** del código **Unicode**, por lo tanto, se pueden declarar variables con el nombre: añoDeCreación, raïm, etc. (se acabó la época de los nombres de variable como ano_de_creacion), aunque eso sí, el **primer carácter no puede ser un dígito numérico** y **no pueden utilizarse espacios en blanco ni símbolos coincidentes con operadores**.
- La **longitud máxima** de los identificadores es prácticamente **ilimitada**.
- **No** puede ser una **palabra reservada** del lenguaje ni los valores lógicos true o false.
- **No** pueden ser **iguales** a **otro identificador** declarado en el mismo ámbito.
- **IMPORTANTE**: Por convenio:
  - **Los nombres de las variables y los métodos deberían empezar por una letra minúscula y los de las clases por mayúscula**.
  - **Si el identificador está formado por varias palabras, la primera se escribe en minúsculas (excepto para las clases) y el resto de palabras se hace empezar por mayúscula** (por ejemplo: añoDeCreación).
  - Estas reglas no son obligatorias, pero son convenientes ya que ayudan al proceso de codificación de un programa, así como a su legibilidad. Es más sencillo distinguir entre clases y métodos o variables.

Serían identificadores válidos:

```
identificador
nombre_Usuario
Nombre_Usuario
_variable_del_sistema
$transaccion
```

y su uso sería, por ejemplo:

```
int contador_principal;
char _lista_de_ficheros;
float $cantidad_en_Ptas;
```

Tipos de datos

En Java existen dos tipos principales de datos:

1) Tipos de datos **simples**.
2) Referencias a **objetos**.

Los tipos de datos simples son aquellos que pueden utilizarse directamente en un programa, sin necesidad del uso de clases (POO). Estos tipos son:

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

El segundo tipo está formado por todos los demás. Se les llama **referencias** porque en realidad lo que se **almacena** en los mismos son **punteros** a **zonas de memoria** donde se encuentran almacenadas las estructuras de datos que los soportan. Dentro de este grupo se encuentran las clases (objetos) y también se incluyen las interfaces, los vectores y los Strings.

Pueden realizarse conversiones entre los distintos tipos de datos (incluso entre simples y referenciales), bien de forma implícita o de forma explícita.

Simples

Los tipos de datos simples soportados por Java son los siguientes:

TIPO	Descripción	Formato	longitud	Rango
byte	byte	C-2*	1 byte	- 128 … 127
short	entero corto	C-2	2 bytes	- 32.768 … 32.767
int	entero	C-2	4 bytes	- 2.147.483.648 …2.147.483.647
long	entero largo	C-2	8 bytes	-9.223.372.036.854.775.808… 9.223.372.036.854.775.807
float	real en coma flotante de s.p.**	IEEE 754	32 bits	±3,4*10-38 … ±3,4*1038
double	real en coma flotante de d.p.	IEEE 754	64 bits	±1,7*10-308 … ±1,7*10308
char	carácter	Unicode	2 bytes	0 … 65.535
boolean	lógico		1 bit	true / false

* C-2 = Complemento a dos. ** s.p. = Simple Precisión / d.p. = Doble Precisión.

No existen más datos simples en Java. Incluso éstos que se enumeran son envueltos por clases equivalentes (java.lang.Integer, java.lang.Double, java.lang.Byte, etc.) , que pueden tratarlos como si fueran objetos en lugar de datos simples.

A diferencia de otros lenguajes de programación como el C, en Java los tipos de datos simples no dependen de la plataforma ni del sistema operativo. Un entero de tipo int siempre tendrá 4 bytes, por lo que no tendremos sorpresas al migrar un programa de un sistema operativo a otro. Es más, ni siquiera hay que volverlo a compilar.

Importante

Eso sí, **Java no realiza una comprobación de los rangos.** Por ejemplo: si a una variable de tipo short con el valor 32.767 se le suma 1, el resultado será -32.768 y no se producirá ningún error de ejecución.

Importante

A diferencia de otros lenguajes de programación, **los Strings en Java** no son un tipo simple de datos sino un **objeto**. Los valores de tipo String van entre comillas dobles (“Hola”), mientras que los de tipo char van entre comillas simples (‘K’)

Referenciales

El resto de tipos de datos que no son simples, son considerados referenciales. Estos tipos son básicamente las clases, en las que se basa la programación orientada a objetos. Al declarar un objeto que perteneciente a una determinada clase, se está reservando una zona de memoria donde se almacenarán los atributos y otros datos pertenecientes a dicho objeto. Lo que se almacena en el objeto en sí, es un puntero (referencia) a dicha zona de memoria.

Dentro de estos tipos pueden considerarse las interfaces, los Strings y los vectores, que son unas clases un tanto especiales, y que se verán en detalle posteriormente.

En realidad, el momento en el que se realiza la reserva física del espacio de memoria es cuando se instancia el objeto realizando la llamada a su constructor, y no en el momento de la declaración.

Existe un tipo referencial especial nominado por la palabra reservada null que puede ser asignado a cualquier variable de cualquier clase y que indica que el puntero no tiene referencia a ninguna zona de memoria (el objeto no está inicializado).

Además, todos los tipos de datos simples vistos en el punto anterior pueden ser declarados como referenciales (objetos), ya que existen clases que los engloban.

Estas clases son:

Tipo de datos simple	**Clase equivalente**
byte	java.lang.Byte
short	java.lang.Short
int	java.lang.Integer
long	java.lang.Long
float	java.lang.Float
double	java.lang.Double
char	java.lang.Character
boolean	java.lang.Boolean

Declaración de variables

La forma básica de una declaración de variable, por ejemplo, sería:

tipo identificador [= valor][,identificador [= valor] ...];

La declaración de una variable se realiza de la misma forma que en C. Siempre contiene:

1. el nombre (identificador de la variable)
2. el tipo de dato al que pertenece.

El ámbito de la variable depende de la localización en el programa donde es declarada.

Ejemplo: int x;

Las variables pueden ser inicializadas en el momento de su declaración, siempre que el valor que se les asigne coincida con el tipo de dato de la variable.

Ejemplo: int x = 0;

Palabras Clave

Las siguientes son las palabras clave que están definidas en Java y que no se pueden utilizar como identificadores:

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

Palabras Reservadas

Además, el lenguaje se reserva unas cuantas palabras más, pero que hasta ahora no tienen un cometido específico. Son:

cast	uture	generic	inner
operator	outer	rest	var

Ámbito de una variable

El ámbito de una variable es la porción de programa **donde** dicha variable **es visible** para el código del programa y, por tanto, referenciable. El ámbito de una variable **depende del lugar** del programa **donde es declarada**, pudiendo pertenecer a cuatro categorías distintas.

- Variable local.
- Atributo.
- Parámetro de un método.
- Parámetro de un tratador de excepciones.

Importante

Como puede observarse, **NO existen las variables globales.** Esto no es un “defecto” del lenguaje sino todo lo contrario. La utilización de variables globales resulta **peligrosa**, ya que puede ser modificada en cualquier parte del programa y por cualquier procedimiento. Además, a la hora de utilizarlas hay que buscar dónde están declaradas para conocerlas y dónde son modificadas para evitar sorpresas en los valores que pueden contener.

Los ámbitos de las variables u objetos en Java siguen los criterios “**clásicos**”, al igual que en la mayoría de los lenguajes de programación como Pascal, C++, etc. No existen sorpresas.
Si una variable no ha sido inicializada, tiene un valor asignado por defecto. Este valor es:

- Para las variables de **tipo referencial** (objetos), el valor **null**.
- Para las variables de **tipo numérico**, el valor por defecto es **cero** (0),.
- Las variables de **tipo char**, el valor ‘**\u0000**’.
- Las variables de **tipo boolean**, el valor **false**.

Pero es una buena práctica inicializarlas siempre.

Variables locales

Una variable local se declara dentro del cuerpo de un método de una clase y **es visible únicamente dentro de dicho método**. Se puede declarar en cualquier lugar del cuerpo, incluso después de instrucciones ejecutables, aunque es una buena costumbre declararlas justo al principio.

También pueden declararse variables dentro de un bloque parentizado por llaves {…}. En ese caso, sólo serán “visibles” dentro de dicho bloque.

Por ejemplo (**NO ES NECESARIO ENTENDER LO QUE HACE EL PROGRAMA**):

```
class Caracter {
        char ch;
        public Caracter(char c) {
                ch=c;
        }
        public void repetir(int num) {
        int i;
        for (i=0;i<num;i++)
                System.out.println(ch);
        }
}
class Ej1 {
        public static void main(String argumentos[]) {
            Caracter caracter;
            caracter = new Caracter('H');
            caracter.repetir(20);
        }
}
```

En este ejemplo existe una variable local: int i; definida en el método repetir de la clase Caracter, por lo tanto, únicamente es visible dentro del método repetir.

También existe una variable local en el método main. En este caso, la variable local es un objeto: Caracter caracter; que sólo será visible dentro del método en el que está declarada (main).

Es importante hacer notar que una declaración como la anterior le indica al compilador el tipo de la variable caracter pero no crea un objeto. El operador que crea el objeto es new, que necesita como único parámetro el nombre del constructor (que será el procedimiento que asigna valor a ese objeto recién instanciado).

Importante

Cuando se pretende declarar múltiples variables del mismo tipo pueden declararse, en forma de

lista, separadas por comas:

Ejemplo:

```
int x,y,z;
```

Declara tres variables x, y, z de tipo entero.

Podrían haberse inicializado en su declaración de la forma:

```
int x=0,y=0,z=3;
```

No es necesario que se declaren al principio del método. Puede hacerse en cualquier lugar del mismo, incluso de la siguiente forma:

```
public void repetir(int num) {
for (int i=0;i<num;i++)
        System.out.println(ch);
}
```

En el caso de las variables locales, éstas no se inicializan con un valor por defecto, como se ha comentado en el punto anterior, sino que es necesario asignarles algún valor antes de poder utilizarlas en cualquier instrucción, de lo contrario el compilador genera un error, de tal forma que **es imposible hacer uso de una variable local no inicializada sin que se percate de ello el compilador.**

Importante

Las variables locales pueden ser antecedidas por la palabra reservada **final**. En ese caso, sólo permiten que se les asigne un valor una única vez.

Ejemplo:

```
final int x=0;
```

No permitirá que a x se le asigne ningún otro valor. Siempre contendrá 0.

No es necesario que el valor se le asigne en el momento de la declaración, podría haberse inicializado en cualquier otro lugar, pero una sola vez.

```
//Ejemplo:
final int x;
…
```

```
x=y+2;
```

Después de la asignación x=y+2, no se permitirá asignar ningún otro valor a x.

Por lo tanto una variable precedida de la palabra final se convierte en una constante.

Entrada y salida estandard

SALIDA.

Ya hemos visto el uso de ***System.out*** para enviar información a la pantalla con los métodos ***print*** y ***println*** (el segundo introduce un salto de línea después de imprimir la cadena). La utilización de System.err sería totalmente análoga para enviar los mensajes producidos por errores en la ejecución (es el canal que usa también el compilador para notificar los errores encontrados).
Por ejemplo, para presentar el mensaje de saludo habitual por pantalla, y después un mensaje de error, tendríamos la siguiente clase (aunque en realidad toda la información va a la consola de comandos donde estamos ejecutando el programa):

```
public class Hola_err{
    public static void main(String [ ] arg) {
    System.out.print("HOLA ");
    System.out.println("mundo");
    System.err.println("Mensaje de error");
    }
}
```

ENTRADA

En cuanto a la entrada estándar en Java, es bastante más compleja. El procedimiento más simple y de más bajo nivel para leer de la entrada estándar es invocar directamente al método ***read()*** del flujo System.in. Este método nos devolverá un byte del flujo de entrada en forma de entero, o un -1 si no hay más entrada disponible.

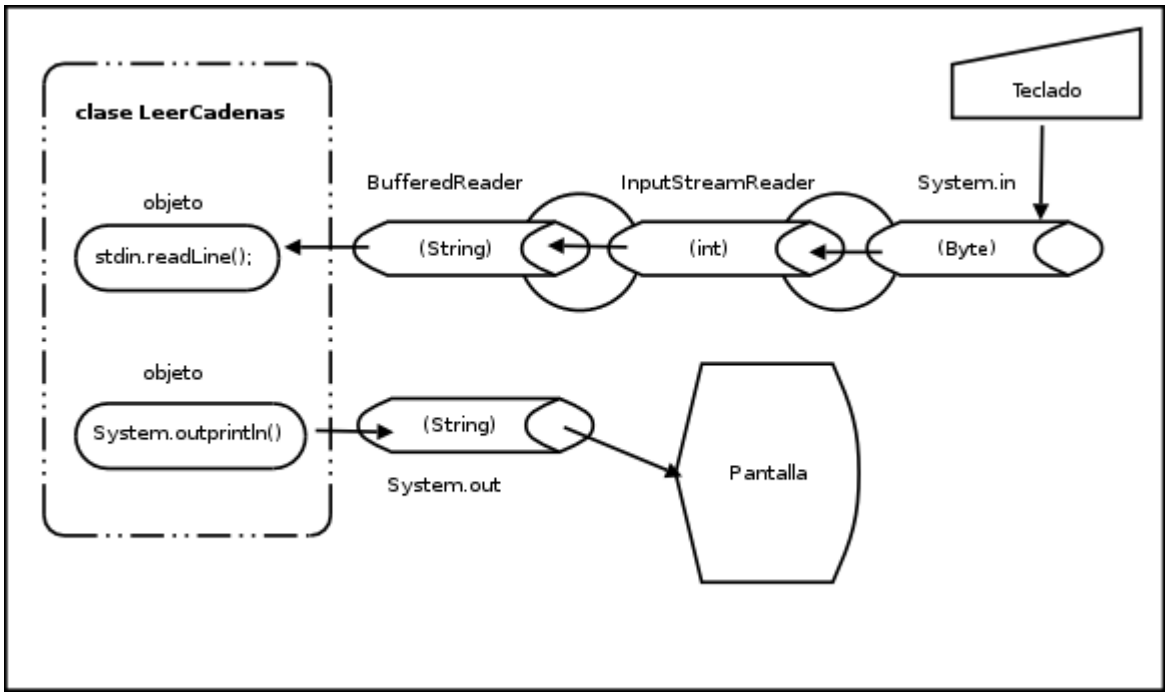
El siguiente ejemplo lo ilustra:

```
import java.io.*;

public class Read {
   public static void main(String [] arg) throws IOException {
   char c;
   System.out.println("Teclee un carcater seguido de ENTER");
       c = (char) System.in.read();
   System.out.println("Ha tecleado "+c);
    }
}
```

Como el método read almacena el resultado como un entero, es necesario convertirlo después a un carácter con una operación de cast (molde) .

El uso directo del método **read** para leer bytes es muy poco práctico y, por lo tanto poco usado. No hay más que pensar en el esfuerzo que supondría leer datos de esta manera y hacer las conversiones apropiadas para cada caso. Lo que se hace es utilizar varias clases **java.io**. El método que utilizaremos es **readLine()** que pertenece a la clase denominada **BufferedReader**. Este método permite leer una secuencia de caracteres de texto, en formato Unicode, para devolver una cadena de tipo **String**. Sin embargo, para crear el objeto de esta clase necesitamos utilizar otra clase intermedia, **InputStreanReader**, que nos permite cambiar un objeto de clase **InputStream** (que lee bytes) en un objeto de tipo **Reader**, necesario para crear un objeto de tipo BufferedReader.




```
import java.io.*;
  public class LeerCadena {
    public static void main(String [] args) throws IOException {
  BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
            String cadena = new String();
           System.out.print("Escriba una cadena  :");
   cadena = stdin.readLine();
   System.out.println("Ha tecleado :"+cadena);
   }
  }
```

Como podemos ver una vez contituido el objeto stdin, simplemente le pedimos cadenas de texto (objetos String), ya que el proceso de convertir bytes en caracteres y agruparlos en cadenas es totalmente transparente para el programador.

Importante

Resumiendo, lo única que podemos leer desde el teclado son cadenas de caracteres (tipo

String), estas cadenas son objetos.

Para poder hacer esto debemos:

- Importar la libreria que me permite utilizar estos objetos.

  import java.io.*;

- Preparar mi clase principal (programa) para posibles errores, esto es obligatorio.

  throws IOException

- Declarar un objeto que se asocia al teclado del ordenador, este objeto se llamará stdin (podéis darle el nombre que queráis).

  BufferedReader stdin = new
  BufferedReader(new InputStreamReader(System.in));

- Declarar objeto para poder guardar lo que leamos desde el teclado.

  String cadena = new String();

- Realizar la lectura desde el teclado y guardar el valor introducido en objeto de tipo String. RECUERDA siempre será una cadena alfanumérica.

  cadena = stdin.readLine();

Entrada de datos numéricos

Finalmente, una vez que tenemos los datos como cadenas a través de readLine(), si necesitamos tener tipos de datos básicos podemos recurrir a las **clases envoltorio,** con las funcionalidades **parser** para extraer datos, como se indica a continuación:

```
import java.io.*;
 public class LeerDatos {
     public static void main(String [] args) throws IOException {
     BufferedReader stdin;
     stdin=new BufferedReader(new InputStreamReader(System.in));


int entero;
float real;

  // Lectura de un número entero, int, short, byte o long
    // Short.parseSort()  Byte.parseByte()  Long.parseLong()

     System.out.print("Escriba un número entero :");
     entero = Integer.parseInt(stdin.readLine());
   System.out.println("Ha tecleado :"+entero);

      // Lectura de un número real: float o double
   // Float.parseFloat() Double.parseDouble()

     System.out.print("Escriba un número real :");
     real = Float.parseFloat(stdin.readLine());
   System.out.println("Ha tecleado :"+real);

   System.out.println();
   System.out.println("La suma vale "+(entero+real));
```

Importante

En realidad lo que hacemos es convertir la cadena de caracteres que leemos desde el tecla a un numérico especifico, byte, short, int, long, float o double.

En Java, a excepción de los tipos básicos descritos antes, todos los demás elementos son clases y objetos, lo que hace que en algunas circustancias tengamos que convertir estos tipos básicos en objetos.

Para facilitar esta conversión utilizamos los envoltorios, que son clases especiales que recubren el tipo básico con una clase, y a partir de este momento el tipo básico envuelto se convierte en un objeto.

Existen nueve envoltorios para tipos básicos de Java. Cada uno de ellos envuelve un tipo básico (añadiendo el tipo vacío, void) y nos permiten trabajar con objetos en lugar de con tipos básicos. En concreto, son los siguientes: ***Integer, Long, Float, Double, Short, Byte, Character, Boolean y Void.***

La función principal de estas clases es la de crear objetos cuya información sea la de un tipo básico asociado.

Además, existen algunos métodos que nos permiten convertir cadenas de caracteres en tipos básicos. Asi, podemos convertir la cadena "123" en el número entero 123 utilizando el método ***parseInt()*** de la clase ***Integer.***

```
int num = Integer.parseInt("123");
```

Entrada y salida clase Scanner

Desde Java 1.5 disponemos de la clase Scanner. La cual nos va a ayudar a leer los datos de una forma más sencilla que el habitual manejo de Lectura de datos por Consola con Java.

La utilización de la clase Scanner es muy sencilla. Lo primero que tenemos que hacer es declarar un objeto Scanner instanciandolo contra la consola, es decir, contra el objeto System.in

Scanner reader = new Scanner(System.in);

Ahora, para leer lo que el usuario está introduciendo por la consola deberemos de utilizar el método .next. Este nos devolverá los caracteres que encuentre en la consola hasta encontrarse un retorno de carro y salto de línea. El valor se lo asignaremos a una variable String.

String sTexto = reader.next();

Una vez que se declara e inicializa la variable se pueden invocar métodos de la clase Scanner tales como:

- **nextByte**: obtiene un número entero tipo byte.
- **nextShort**: obtiene un número entero tipo short.
- **nextInt**: obtiene un número entero tipo int.
- **nextLong**: obtiene un número entero tipo long.
- **nextFloat**: obtiene un número real float.
- **nextDouble**: obtiene un número real double.
- **nextLine**: obtiene una cadena de caracteres.

No existen métodos para obtener directamente booleanos ni para obtener un solo carácter.

nextLine().charAt(0) puede resolver el segundo problema

Ejemplo 1

Ejemplo de lectura de double:

```
import java.util.Scanner;

public class CircleApp2 {

public static void main(String[] args) {

        double radio, area, circumferencia;

        Scanner entrada = new Scanner(System.in);

        System.out.print("Introduce el radio: ");
        radio = entrada.nextDouble();

        area = Math.PI * Math.pow(radio, 2);
        circumferencia = 2 * Math.PI * radio;

        System.out.println("El area es " + area);
        System.out.println("La circumferencia es " + circumferencia);

}
}
```

Ejemplo 2

Ejemplo de lectura de cadenas con Scanner

```
import java.util.Scanner;

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

        String nombre;
        int horas;
        double impuestos, retencion;
        Scanner entrada = new Scanner(System.in);

        System.out.print("Introduce tu nombre:  ");
        nombre = entrada.nextLine();

        System.out.print("Horas trabajadas : ");
        horas = entrada.nextInt();

        ...

}
}
```

Operadores

Los operadores son partes indispensables en la construcción de expresiones. Existen muchas definiciones técnicas para el término expresión. Puede decirse que una expresión es una combinación de operandos ligados mediante operadores.

Los operandos pueden ser variables, constantes, funciones, literales, etc. y los operadores pueden ser:

- Aritméticos.
- Relacionales.
- Lógicos.
- Bits (no son importantes).
- Operador asignación.

Precedencia de operadores en Java:

Operadores postfijos [ ] . (paréntesis)
Operadores unarios ++expr --expr -expr ~ !
Creación o conversión de tipo new (tipo)expr
Multiplicación y división * / %
Suma y resta + -
Desplazamiento de bits << >> >>>
Relacionales < > <= >=
Igualdad y desigualdad == !=
AND a nivel de bits &
XOR a nivel de bits ^
OR a nivel de bits |
AND lógico &&
OR lógico ||
Condicional al estilo C ? :
Asignación = += -= *= /= %= ^= &= |= >>= <<=

Asignación

Operadores de asignación:

El operador de asignación es el símbolo igual (=).

op1 = Expresión

Asigna el resultado de evaluar la expresión de la derecha a op1.

Además del operador de asignación existen unas abreviaturas cuando el operando que aparece a la izquierda del símbolo de asignación también aparece a la derecha del mismo:

Operador	Formato	Equivalencia
+=	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
|=	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

Aritméticos

Operadores aritméticos:

Operador	Formato	Descripción
+	**op1 + op2**	Suma aritmética de dos operandos.
-	**op1 - op2** **-op1**	Resta aritmética de dos operandos. Cambio de signo.
*****	**op1 * op2**	Multiplicación de dos operandos
/	**op1 / op2**	División entera de dos operandos
%	**op1 % op2**	Resto de la división entera (o módulo)
++	**++op1** **op1++**	Incremento unitario
--	**-- op1** **op1--**	Decremento unitario

El operador - puede utilizarse en su versión unaria (- op1) y la operación que realiza es la de invertir el signo del operando.

Como en C, los operadores unarios ++ y -- realizan un incremento y un decremento respectivamente. Estos operadores admiten notación prefija y postfija.

- **++op1**: En primer lugar realiza un incremento (en una unidad) de op1 y después ejecuta la instrucción en la cual está inmerso.
- **op1++**: En primer lugar ejecuta la instrucción en la cual está inmerso y después realiza un incremento (en una unidad) de op1.
- **--op1**: En primer lugar realiza un decremento (en una unidad) de op1 y después ejecuta la instrucción en la cuál está inmerso.
- **op1--**: En primer lugar ejecuta la instrucción en la cual está inmerso y después realiza un decremento (en una unidad) de op1.

La diferencia entre la notación prefija y la postfija no tiene importancia **en expresiones en las que únicamente existe dicha operación**:

```
++contador    es equivalente a:    contador++;
--contador;                        contador--;
```

Es importante no emplear estos operadores en expresiones que contengan más de una referencia a la

variable incrementada, puesto que esta práctica puede generar resultados erróneos fácilmente.

La diferencia es apreciable en instrucciones en las cuáles están incluidas otras operaciones.
Por ejemplo:

cont = 1; **do {** **…}** **while (cont++ < 3);**	**cont = 1;** **do {** **…}** **while (++cont < 3);**

En el primer caso, el bucle se ejecutará 3 veces, mientras que en el segundo se ejecutará 2 veces

Relacionales

Operadores relacionales:

Operador	Formato	Descripción
>	**op1 > op2**	Devuelve true (cierto) si op1 es mayor que op2
<	**op1 < op2**	Devuelve true (cierto) si op1 es menor que op2
>=	**op1 >= op2**	Devuelve true (cierto) si op1 es mayor o igual que op2
<=	**op1<= op2**	Devuelve true (cierto) si op1 es menor o igual que op2
==	**op1 == op2**	Devuelve true (cierto) si op1 es igual a op2
!=	**op1 != op2**	Devuelve true (cierto) si op1 es distinto de op2

Los operadores relacionales actúan sobre valores enteros, reales y caracteres (char); y devuelven un valor del tipo boolean (true o false).

```
class Relacional {
        public static void main(String arg[]) {
                double op1,op2;
                op1=1.34;
                op2=1.35;
                System.out.println("op1="+op1+" op2="+op2);
                System.out.println("op1>op2 = "+(op1>op2));
                System.out.println("op1<op2 = "+(op1<op2));
                System.out.println("op1==op2 = "+(op1==op2));
                System.out.println("op1!=op2 = "+(op1!=op2));
                char op3,op4;
                op3='a'; op4='b';
                System.out.println("'a'>'b' = "+(op3>op4));
        }
}
```

Salida por pantalla:

```
op1=1.34 op2=1.35
op1>op2 = false
op1<op2 = true
op1==op2 = false
op1!=op2 = true
'a'>'b' = false
```

Nota: Los operadores == y != actúan también sobre valores lógicos (boolean)

Lógicos

Operadores lógicos:

Operador	Formato	Descripción
&&	op1 && op2	Y lógico. Devuelve true (cierto) si son ciertos op1 y op2
||	op1 || op2	O lógico. Devuelve true (cierto) si son ciertos op1 o op2
!	! op1	Negación lógica. Devuelve true (cierto) si es false op1.

Estos operadores actúan sobre operadores o expresiones lógicas, es decir, aquellos que se evalúan a cierto o falso (true / false).

Por ejemplo, el siguiente programa:

```
class Bool {
	public static void main ( String argumentos[] ) {
		boolean a=true;
		boolean b=true;
		boolean c=false;
		boolean d=false;
		System.out.println("true Y true = " + (a && b) );
		System.out.println("true Y false = " + (a && c) );
		System.out.println("false Y false = " + (c && d) );
		System.out.println("true O true = " + (a || b) );
		System.out.println("true O false = " + (a || c) );
		System.out.println("false O false = " + (c || d) );
		System.out.println("NO true = " + !a);
		System.out.println("NO false = " + !c);
		System.out.println("(3 > 4) Y true = " + ((3 >4) && a) );
	}
}
```

produciría la siguiente salida por pantalla:

```
true Y true = true
true Y false = false
false Y false = false
true O true = true
true O false = true
false O false = false
NO true = false
NO false = true
(3 > 4) Y true = false
```

Bits

Operadores de bits (opcional):

Operador	Formato	Descripción
>>	**op1 >> op2**	Desplaza op1, op2 bits a la derecha
<<	**op1 << op2**	Desplaza op1, op2 bits a la izquierda
>>>	**op1 >>> op2**	Desplaza op1, op2 bits a la derecha (sin signo)
&	**op1 & op2**	Realiza un Y (AND) a nivel de bits
|	**op1 | op2**	Realiza un O (OR) a nivel de bits
^	**op1 ^ op2**	Realiza un O exclusivo (XOR) a nivel de bits
~	**~op1**	Realiza el complemento de op1 a nivel de bits.

Los operadores de bits actúan sobre valores enteros (byte, short, int y long) o caracteres (char).

```
class Bits {
	public static void main ( String argumentos[] ) {
		byte a=12;
		byte b=-12;
		byte c=6;
		System.out.println("12 >> 2 = " + (a >> 2) );
		System.out.println("-12 >> 2 = " + (b >> 2) );
		System.out.println("-12 >>> 2 = " + (b >>> 2) );
		System.out.println("12 << 2 = " + (a << 2) );
		System.out.println("-12 << 2 = " + (b << 2) );
		System.out.println("12 & 6 = " + (a & c) );
		System.out.println("12 | 6 = " + (a | c) );
		System.out.println("12 ^ 6 = " + (a ^ c) );
		System.out.println("~12 = " + ~a);
	}
}
```

Clase Math

La clase Math

Se echan de menos operadores matemáticos más potentes en Java. Por ello se ha incluido una clase especial llamada **Math** dentro del paquete java.lang. Para poder utilizar esta clase, se debe incluir esta instrucción:

```
import java.lang.Math;
```

Esta clase posee métodos muy interesantes para realizar cálculos matemáticos complejos. Por ejemplo:

```
double x= Math.pow(3,3);
```

Math posee dos constantes, que son:

```
double E  El número e (2, 7182818245...)
double PI El número Π (3,14159265...)
```

Por otro lado posee numerosos métodos que son:

operador	significado
double ceil(double x)	Redondea x al entero mayor siguiente: **Math.ceil(2.8) vale 3** **Math.ceil(2.4) vale 3** **Math.ceil(-2.8) vale -2**
double floor(double x)	Redondea x al entero menor siguiente: **Math.floor(2.8) vale 2** **Math. floor (2.4) vale 2** **Math. floor (-2.8) vale -3**
int round(double x)	Redondea x de forma clásica: **Math.round(2.8) vale 3** **Math. round (2.4) vale 2** **Math. round (-2.8) vale -3**
double rint(double x) Id	idéntico al anterior, sólo que éste método da como resultado un número double mientras que round da como resultado un entero tipo int
double random()	Número aleatorio de 0 a 1
tiponúmero abs(tiponúmero x)	Devuelve el valor absoluto de x.

tiponúmero min(tiponúmero x, tiponúmero y)	Devuelve el menor valor de x o y
tiponúmero max(tiponúmero x, tiponúmero y)	Devuelve el mayor valor de x o y
double sqrt(double x)	Calcula la raíz cuadrada de x
double pow(double x, double y)	Calcula x^y
double exp(double x)	Calcula e^x
double log(double x)	Calcula el logaritmo neperiano de x
double acos(double x)	Calcula el arco coseno de x
double asin(double x)	Calcula el arco seno de x
double atan(double x)	Calcula el arco tangente de x
double sin(double x)	Calcula el seno de x
double cos(double x)	Calcula el coseno de x
double tan(double x)	Calcula la tangente de x
double toDegrees(double anguloEnRadianes)	Convierte de radianes a grados
double toRadians(double anguloEnGrados)	Convierte de grados a radianes

Literales

Valores literales.

A la hora de tratar con **valores** de los tipos de **datos simples** (y **Strings**) se utiliza lo que se denomina "**literales**".

Los literales son elementos que sirven para **representar un valor** en el código fuente del programa.

En Java existen literales para los siguientes tipos de datos:

- Enteros (byte, short, int y long).
- Reales (double y float).
- Lógicos (boolean).
- Carácter (char).
- Cadenas de caracteres (**String**).

Literales de tipo entero.

Los literales de tipo entero: **byte**, **short**, **int** y **long** pueden expresarse en:

- Decimal (base 10) por ejemplo el número 21. En Java, el compilador identifica un entero decimal al encontrar un número cuyo primer dígito es cualquier símbolo decimal excepto el cero (del 1 al 9). A continuación pueden aparecer dígitos del 0 al 9.
- Octal (base 8) por ejempo 025. En octal un número siempre empieza por cero, seguido de dígitos octales (del 0 al 7).
- Hexadecimal (base 16) por ejemplo 0x15. En hexadecimal un número siempre empieza por 0x seguido de dígitos hexadecimales: del 0 al 9, de la 'a' a la 'f' y de la 'A' a la 'F'
- Además, puede añadirse al final del mismo la letra 'l' o 'L' para indicar que el entero es considerado como long (64bits

Ejemplos: Sistemas de numeración.

Decimal	Octal	Hexadecimal
34	042	0x22
962	01702	0x3c2
8186	017772	0x1FFA

La letra 'l' o 'L' al final de un literal de tipo entero puede aplicarse a cualquier sistema de numeración e indica que el número decimal sea tratado como un entero largo (de 64 bits). Esta letra 'l' o 'L' puede ser mayúscula o minúscula, aunque es aconsejable utilizar la mayúscula ya que de lo contrario puede confundirse con el dígito uno (1) en los listados.

Ejemplo:

```
long max1 = 9223372036854775807L; //valor máximo para un entero largo (64 bits)
```

Literales de tipo real.

Los literales de tipo real sirven para indicar valores **float** o **double**. A diferencia de los literales de tipo entero, no pueden expresarse en octal o hexadecimal.

Existen dos formatos de representación:

- mediante su parte entera, el punto decimal (.) y la parte fraccionaria.
- mediante notación exponencial o científica.

Ejemplos:

3.1415
0.31415e1
.31415e1
0.031415E+2
.031415e2
314.15e-2
31415E-4

Si no se indica nada se supone que pertenecen al tipo ***double***. Si se desea que se intepreten como del tipo **float** se debe añadir un carácter 'f' o 'F' detrás del valor. Se puede añadir el carácter 'd' o 'D' para indicar explicitamente que el valor es del tipo **double**.

- 'f' o 'F' --> Trata el literal como de tipo float.
- 'd' o 'D' --> Trata el literal como de tipo double.

Ejemplos:

3.1415F
.031415d

Literales lógicos o booleanos.

Son únicamente dos: las palabras reservadas **true** y **false** y se escriben siempre en minúsculas.

Ejemplo: **boolean activado = false;**

Literales de tipo carácter.

Los literales de tipo carácter se representan siempre **entre comillas simples**. Entre las comillas simples puede aparecer:

- Un símbolo (letra) siempre que el carácter esté asociado a un código Unicode.

Ejemplos: ‘a’ , ‘B’ , ‘{‘ , ‘ñ’ , ‘á’ .

- Un octal. POr ejemplo '\102'
- Un Hexadecimal. Por ejemplo '\u00A3'
- Una “secuencia de escape”. Las secuencias de escape son combinaciones del símbolo contrabarra \ seguido de una letra, y sirven para representar caracteres que no tienen una equivalencia en forma de símbolo.

Las posibles secuencias de escape son:

Secuencia de escape	Significado
’\’’	Comilla simple.
’\”’	Comillas dobles.
’\\’	Contrabarra
’\b’	Backspace (Borrar hacia atrás).
’\n’	Cambio de línea.
’\f’	Form feed.
’\r’	Retorno de carro.
’\t’	Tabulador

Literales de tipo String.

Los **Strings** o **cadenas de caracteres** no forman parte de los tipos de datos elementales en Java, sino que son instanciados a partir de la clase java.lang.String, pero aceptan su inicialización a partir de literales de este tipo, por lo que se tratan en este punto.
Un literal de tipo string va encerrado entre comillas dobles (“) y debe estar incluido completamente en una sola línea del programa fuente (no puede dividirse en varias líneas). Entre las comillas dobles puede incluirse cualquier carácter del código Unicode (o su código precedido del carácter \ ) además de las secuencias de escape vistas anteriormente en los literales de tipo carácter. Así, por ejemplo, para incluir un cambio de línea dentro de un literal de tipo string deberá hacerse mediante la secuencia de escape \n :
Ejemplo:

```
System.out.println(”Primera línea\nSegunda línea del string\n”);
System.out.println(”Hola”);
```

La visualización del string anterior mediante println() produciría la siguiente salida por pantalla:

Primera línea
Segunda línea del string
Hola

La forma de incluir los caracteres: comillas dobles (“) y contrabarra (\ ) es mediante las secuencias de escape \” y \\ respectivamente (o mediante su código Unicode precedido de \ ).

Atención

Si el string es demasiado largo y debe dividirse en varias líneas en el fichero fuente, puede utilizarse el operador de concatenación de strings **+** .de la siguiente forma:

```
"Este String es demasiado largo para estar en una línea del " +
"fichero fuente y se ha dividido en dos."
```

Conversión de tipo

En muchas ocasiones resulta necesario realizar algunas conversiones de tipos, de forma que el resultado sea del tipo esperado. Para convertir valores entre tipos existen dos formas:

- **Conversión automática de tipos**. Cuando el tipo al que se asigna un valor es "mayor", la conversión se raliza automáticamente. Así un valor de tipo double se puede multiplicar por otro de tipo entero (int). Antes de hacer la multiplicación se convierte el valor int a double y luego se hace la multiplicación entre reales.
- **Conversión explícita**. Se pone delante del valor a convertir, entre paréntesis, el tipo al que se desea convertir. Como esta conversión suele implicar una pérdida de valor, hay que tener mucho cuidado. Por ejemplo:

  (int) 34,45 hace una conversión a tipo int, trunca el valor y se pierden los decimales.

  area = (float) 3,1416 * radio; 3,1416 se considerará como double, por lo que tendremos que convertir el resultado.

En una conversión de tipos existe un tipo especial de expresiones que involucra a los valores de tipo char. Un tipo char siempre se puede uyilizar en una expresión junto con números eneteros como si fuee un número entero más:

```
char c = 'A';

int n;

n = c + 2;
```

Enumerados

Los enumerados son conjuntos de valores constantes para los que no existe tipo predefinido. Por ejemplo, no existe ningún tipo predefinido para representar los días de la semana, las estaciones del año, los meses del año, los turnos de clases, etc.

Para definir un tipo enumerado con sus valores se haría de la siguiente forma:

- enum DiaSemana {LUNES, MARTES, MIÉRCOLES, JUEVES, VIERNES, SABADO, DOMINGO}
- enum TurnoDeClase {MAÑANA, TARDE}
- enum TipoDeClase {TEORIA, LABORATORIO, SEMINARIO, CHARLA, EXPERIMENTO}

En el siguiente ejemplo se manejan los valores de los días de la semana.

```
DiaSemana hoy = DiaSemana.JUEVES;

DiaSemana ultimo = DiasSemana.DOMINGO;

System.out.println("Hoy es "+hoy);

System.out.println("El último día es "+ultimo);
```

Ejercicios básicos

Entrada y salida. Familiarizarse con la lectura desde el teclado y la escritura en la pantalla.

Saludo.java

Programa que muestra en varias líneas el siguiente mensaje:

> Buenos dias.
> Sr. José Antonio Díaz-Alejo Gómez.
> Que tenga un buen día

Saludo2.java

Programa que muestra, en varias líneas el siguiente mensaje (esta vez el nombre de la persona debe solicitarse primero).

> Buenos dias.
> Sr/a. **nombre_de_la_persona.**
> Que tenga un buen día.

Operaciones. Familiarizarse con la sintaxis para realizar operaciones y recoger los resultados.

Opera.java

Programa que lee dos números enteros de tipo int y muestra la suma, resta, multiplicación, división, resto de la división, incremento y decremento unario

Opera2.java

Programa que lee dos números reales de tipo float y muestra la suma, resta, multiplicación, división, resto de la división, incremento y decremento unario.

Relacional.java

Programa que muestra el resultado de comparar dos números (enteros o reales), si son iguales, distintos, mayor, menor,...

Millas.java

Programa que dado un valor en millas nos lo traduce a metros

Rectángulo.java

Escribe un programa que defina dos variables enteras para describir las longitudes de los lados de un rectángulo. El programa debe pedir, calcular y escribir en la pantalla las longitudes de los lados, el perímetro y el área del rectángulo.

Circunferencia.java

Programa que solicita el radio de una circunferencia y muestra la longitud de la circunferencia, el área del círculo que se crea y el volumen de la esfera que forma.

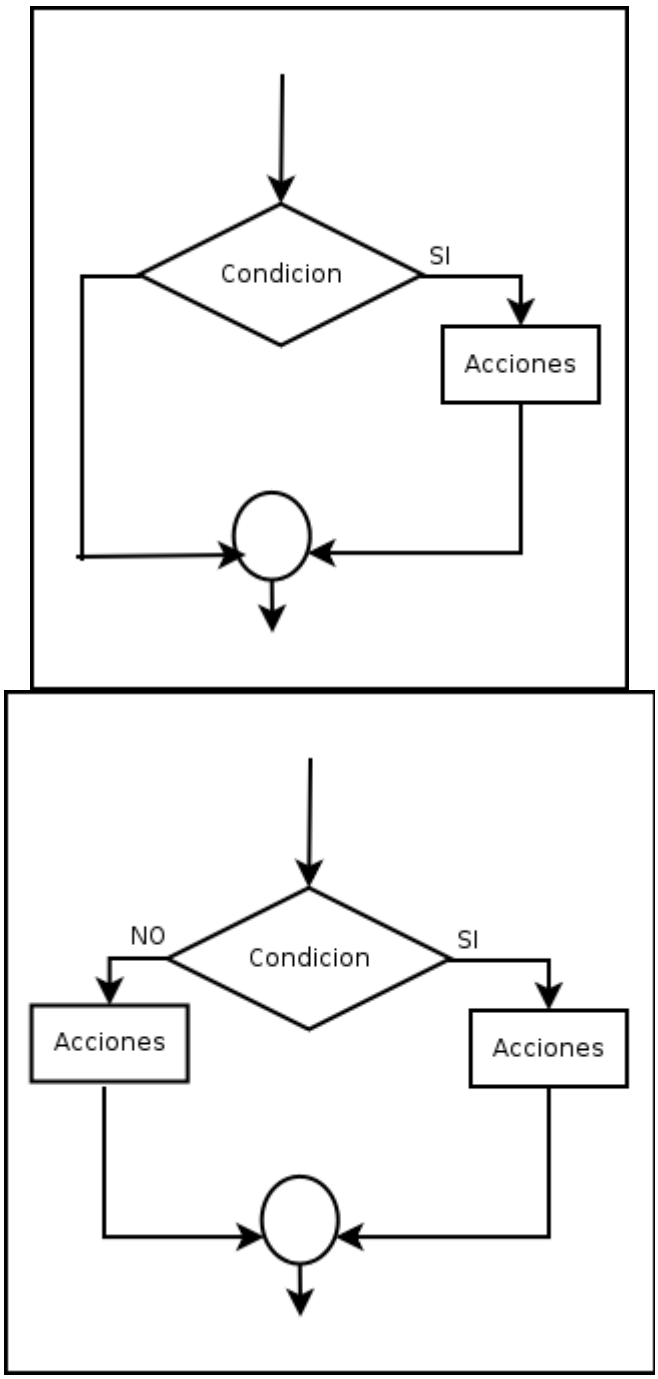
Tiempo.java

Escribe un programa que solicite al usuario una cantidad en segundos y la convierta en a días, horas, minutos y segundos.

Alternativas

Estructuras alternativas.

Las estructuras alternativas son construcciones que permiten alterar el flujo secuencial de un programa, de forma que en función de una condición o el valor de una expresión, el mismo pueda ser desviado en una u otra alternativa de código.



Alternativa Simple Alternativa Doble

Las estructuras alternativas disponibles en Java son:

- Operador condicional.
- Alternativa if-else.
- Alternativa switch

Operador condicional.

El operador ternario es otro de los operadores condicionales. Es una forma reducida de escribir un if-then-else. El operador ternario es representado mediante el símbolo ?:

La estructura del operador ternario es:

variable = (condición) ? (valor si la condición es cierta) : (valor si la condición es falsa);

- En el caso de que la expresión tenga un valor de true se retorna el valor indicado después del cierre de interrogación (?)
- Y si la expresión tiene un valor de false se retorna el valor que esté después de los dos puntos (:).

El operador ternario se suele utilizar para decidir que valor asignar. Un ejemplo de código del operador ternario sería:

```
int vble1 = 5;
int vble2 = 4;
int mayor;

mayor = (vble1 > vble2)?vble1:vble2;

System.out.println("El mayor de los dos números es " + mayor);
```

Vemos que si la variable1 es mayor que la variable2 guardaremos el valor de la variable1 en la variable mayor. En caso contrario se guardaría el valor de la variable2, ya que en ese caso sería la mayor.

Alternativa if - else

if-else.

Forma simple:

```
if (expresión) {
                Bloque instrucciones;
               {
```

El bloque de instrucciones se ejecuta si, y sólo si, la expresión (que debe ser lógica) se evalúa a true, es decir, se cumple una determinada condición.

```
if (cont == 0)
            System.out.println(“he llegado a cero”);
```

La instrucción System.out.println(“he llegado a cero”); sólo se ejecuta en el caso de que cont contenga el valor cero.

Con cláusula else:

```
if (expresión){
                Bloque instrucciones 1;
              }
    else
      {
        Bloque instrucciones 2;
      }
```

El bloque de instrucciones 1 se ejecuta si, y sólo si, la expresión se evalúa a true. Y en caso contrario, si la expresión se evalúa a false, se ejecuta el bloque de instrucciones 2.

```
if (cont == 0) {
            System.out.println(“he llegado a cero”);
          }
        else
          {
            System.out.println(“no he llegado a cero”);
          }
```

Si cont vale cero, se mostrará en el mensaje “he llegado a cero”. Si cont contiene cualquier otro valor distinto de cero, se mostrará el mensaje “no he llegado a cero”.

Atención

Recordad que el operador relacional para comprobar si son iguales es ==, no un solo = que corresponde con el operador de asignación.

Este error no lo detecta el compilador y es dificil de averiguar.

if-else anidados.

En muchas ocasiones, se anidan estructuras alternativas if-else, de forma que se pregunte por una condición si anteriormente no se ha cumplido otra sucesivamente.

Por ejemplo: supongamos que realizamos un programa que muestra la nota de un alumno en la forma (insuficiente, suficiente, bien, notable o sobresaliente) en función de su nota numérica. Podría codificarse de la siguiente forma:

```
class Nota {
public static void main (String arg[]) {
      BufferedReader stdin;
      stdin=new BufferedReader(new InputStreamReader(System.in));
      int nota;

      System.out.println("Dame un número entre 0 y 10 :");
      nota =Integer.parseInt(stdin.readLine())
      if (nota<0 || nota>10) {
            System.out.println("Error el número debe estar entre 0 y 10 :");
            }
       else
            {
            if (nota<5)
                  System.out.println("Insuficiente");
                  else
                  if (nota<6)
                        System.out.println("Suficiente");
                        else
                        if (nota<7)
                              System.out.println("Bien");
                              else
                              if (nota<9)
                                    System.out.println("Notable");
                                    else
```

```
                System.out.println("Sobresaliente");
            }
        }
        }
```

Tanto Java, como en C y a diferencia de otros lenguajes de programación, en el caso de que el bloque de instrucciones conste de una sola instrucción (no necesita ser encerrado entre llaves), se pone un punto y coma (;) justo antes de la cláusula else.

Alternativa multiple

switch.

Forma simple:

```
switch (expresión) {
   case valor1: instrucciones1;
   case valor2: instrucciones2;
   …
   case valorN: instruccionesN;
}
```

En este caso, si instrucciones1 ó instrucciones2 ó instruccionesN están formados por un bloque de instrucciones sencillas, no es necesario parentizarlas mediante las llaves ({ … }).

En primer lugar se evalúa la expresión cuyo resultado **debe ser un valor**. El programa comprueba el primer valor (valor1). En el caso de que el valor resultado de la expresión coincida con valor1, se ejecutaran las instrucciones1.

Pero ¡ojo! También se ejecutarían las instrucciones instrucciones2 .. instruccionesN hasta encontrarse con la palabra reservada **break**. Si el resultado de la expresión no coincide con valor1, evidentemente no se ejecutarían instrucciones1, se comprobaría la coincidencia con valor2 y así sucesivamente hasta encontrar un valor que coincida o llegar al final de la construcción switch.
En caso de que no exista ningún valor que coincida con el de la expresión, no se ejecuta ninguna acción.

Si lo que se desea es que únicamente se ejecuten las instrucciones asociadas a cada valor y no todas las de los demás case que quedan por debajo, la construcción switch sería la siguiente:

```
switch (expresión) {
        case valor1: instrucciones1;
        break;
        case valor2: instrucciones2;
        break;
        …
        case valorN: instruccionesN;
        }
class DiaSemana {
      public static void main(String arg[ ]) {
            BufferedReader stdin;
            stdin=new BufferedReader(new InputStreamReader(System.in));
            int dia;
            System.out.println("Dame un número entre 1 y 7 :");
            dia =Integer.parseInt(stdin.readLine());
            if (dia<1 || dia>7) {
                        System.out.println("Error el número debe estar entre 0 y 7 :");
                        }
```

```
            else
                {
                  switch (dia) {
                        case 1: System.out.println("Lunes");
                              break;
                        case 2: System.out.println("Martes");
                              break;
                        case 3: System.out.println("Miércoles");
                              break;
                        case 4: System.out.println("Jueves");
                              break;
                        case 5: System.out.println("Viernes");
                              break;
                        case 6: System.out.println("Sábado");
                              break;
                        case 7: System.out.println("Domingo");
                        }
                  }
            }
}
```

Nótese que en el caso de que se introduzca un valor no comprendido entre 1 y 7, nos dará un mensaje de error.

Podemos hacerlo directamente con la cláusula por defecto:

```
switch (expresión) {
      case valor1: instrucciones1;
      case valor2: instrucciones2;
      …
      case valorN: instruccionesN;
      default: instruccionesPorDefecto;
      }
```

En este caso, ***instruccionesPorDefecto*** se ejecutarán en el caso de que ningún valor case coincida con el valor de expresión.

```
class DiaSemana {
      public static void main(String arg[ ]) {
      BufferedReader stdin;
      stdin=new BufferedReader(new InputStreamReader(System.in));
      int dia;
      System.out.println("Dame un número entre 1 y 7 :");
      dia =Integer.parseInt(stdin.readLine());

      switch (dia) {
                  case 1: System.out.println("Lunes");
                  break;
                  case 2: System.out.println("Martes");
                        break;
                  case 3: System.out.println("Miércoles");
```

```
                    break;
                case 4: System.out.println("Jueves");
                    break;
                case 5: System.out.println("Viernes");
                    break;
                case 6: System.out.println("Sábado");
                    break;
                case 7: System.out.println("Domingo");
                    break;
                default: System.out.println("Error el número debe estar entre 0 y 7 :");
                }
        }
    }
```

Ejecutar algunas instrucciones en alguno de los case, que no haya ninguna instrucción break desde ese punto hasta la cláusula default.

```
    class DiasMes {
        public static void main(String argumentos []) {
        BufferedReader stdin;
        stdin=new BufferedReader(new InputStreamReader(System.in));
        int mes;

        System.out.println("Dame un número entre 1 y 12 :");
        mes =Integer.parseInt(stdin.readLine());

        switch (mes) {

                case 1:
                case 3:
                case 5:
                case 7:
                case 8:
                case 10:
                case 12: System.out.println("El mes "+mes +" Tiene 31 días");
                    break;
                case 4:
                case 6:
                case 9:
                case 11: System.out.println("El mes "+mes + " Tiene 30 días");
                    break;
                case 2: System.out.println("El mes "+mes + " Tiene 28 ó 29 días");
                    break;
                default: System.out.println("El mes "+mes +" no existe");
                }

        }
    }
```

En este ejemplo, únicamente se ejecutará la cláusula default en el caso en que el valor de mes sea

distinto a un número comprendido entre 1 y 12, ya que todas estas posibilidades se encuentran contempladas en las respectivas cláusulas case y, además, existe un break justo antes de la cláusula default, por lo que en ningún otro caso se ejecutará la misma.

IMPORTANTE !!!

Nunca utilizaremos la sentencia break para otra cosa que no sea marcar el final del bloque de la estructura switch.

Si se utiliza en otro lugar estaremos incurriendo en programación NO ESTRUCTURADA, saltos incondicionales.

Desde la versión JDK 7, se puede usar un objeto **String** en la expresión de una instrucción switch :

```
String dia;
switch (diaDeSemana) {
        case "lunes":
                dia = "Inicio de la semana de trabajo";
                break;
        case "martes":
        case "miércoles":
        case "jueves":
                dia= "Entre semana";
                break;
        case "viernes":
                dia= "Fin de la semana de trabajo";
                break;
        case "sábado":
        case"Domingo":
                dia= "Fin de semana";
                break;
       default:
                System.out.println( ("Día inválido de la semana:" + diaDeSemana);
}
```

La instrucción switch compara el objeto String en su expresión (diaDeSemana) con las expresiones asociadas a cada etiqueta de case como si estuviera usando el método String.***equals***; por lo tanto, la comparación de objetos String en sentencias switch es sensible a mayúsculas y minúsculas.

El compilador de Java genera bytecode generalmente más eficiente a partir de sentencias switch que usan objetos String que de sentencias if-then-else encadenadas.

Ejercicios condicionales

Estructuras alternativas. Familiarizarse con la utilización de las estructuras alternativas (if, if-else, if anidados y switch).

Realiza los siguientes programas en JAVA

Ejercicio común nº 10: Programa de un programa que pide la edad y nos muestra el mensaje “ERES MAYOR DE EDAD” solo si lo somos.

Ejercicio común nº 11: Programa de un programa que pide la edad y nos muestra el mensaje “ERES MAYOR DE EDAD” o el mensaje “ERES MENOR DE EDAD”.

Ejercicio común nº 12: Programa que lee dos números, calcula y muestra el valor de su suma, resta, producto y división. *** tener en cuenta la división por cero ***

Ejercicio común nº 13: Programa de un programa que lee 2 números y muestra el mayor.

Ejercicio común nº 14: Programa de un programa que lee un número y me dice si es positivo o negativo, consideraremos el cero como positivo.

Ejercicio común nº 15: Programa de un programa que lee dos números y los visualiza en orden ascendente.

Ejercicio común nº 16: Programa que lee dos números y nos dice cuál es el mayor o si son iguales.

Ejercicio común nº 17: Programa que lee tres números distintos y nos dice cuál es el mayor.

Ejercicio común nº 18: Programa que lee una calificación numérica entre 0 y 10 y la transforma en calificación alfabética, escribiendo el resultado.

de 0 a <3 Muy Deficiente.
de 3 a <5 Insuficiente.
de 5 a <6 Suficiente

de 6 a<7 Bien
de 6 a <9 Notable
de 9 a 10 Sobresaliente

Ejercicio comun nº 19: Programa que lee tres números cualesquiera y nos indica todas sus relaciones de igualdad.

Ejercicio común nº 20: Programa que recibe como datos de entrada una hora expresada en horas, minutos y segundos que nos calcula y escribe la hora, minutos y segundos que serán, transcurrido un segundo.

Ejercicio común nº 21: Programa que calcula el salario neto semanal de un trabajador en función del número de horas trabajadas y la tasa de impuestos de acuerdo a las siguientes hipótesis:

Las primeras 35 horas se pagan a tarifa normal.
Las horas que pasen de 35 se pagan a 1,5 veces la tarifa normal.

Las tasas de impuestos son:

Los primeros 500 euros son libres de impuestos.
Los siguientes 400 tienen un 25% de impuestos.
Los restantes un 45% de impuestos.

Escribir nombre, salario bruto, tasas y salario neto.

Ejercicio común nº 22: En un determinado comercio se realiza un descuento dependiendo del precio de cada producto. Si el precio es inferior a 6 euros, no se hace descuento; si es mayor o igual a 6 euros y menor que 60 euros, se hace un 5 % de descuento; y si es mayor o igual a 60 euros, se hace un 10 % de descuento.

Programa que lee el precio de un producto y nos calcula y escribe su precio final.

Ejercicio común nº 23: Programa que lee como dato de entrada un año y nos dice se si trata de un año bisiesto o no. Se sabe que son bisiestos todos los años múltiplos de 4, excepto los que sean múltiplos de 100 sin ser de 400.

Ejercicio común nº 24: Programa que lee como datos de entrada una fecha expresada en día (del 1 al 31) mes(del 1 al 12) y año (xxxx) y nos dice la fecha que será al día siguiente. Se supondrá que febrero tiene siempre 28 días.

Ejercicio común nº 25: Programa que lee dos números positivos y distintos y nos dice si el menor es divisor del mayor.

Ejercicio Común nº 26: Programa que calcula la edad de una persona recibiendo como datos de entrada la fecha de nacimiento y la fecha actual, ambas en tres variables que recibirán el día (del 1 al 31), el mes (del 1 al 12) y el año en número entero.

Ejercicio común nº 27: Programa que lee tres números y los visualiza en orden ascendente.

Ejercicios con alternativas (voluntarios)

Estructuras alternativas. Familiarizarse con la utilización de las estructuras alternativas (if, if-else, if anidados y switch).

Mes.java

Programa que muestra el nombre del mes introduciendo el número.

ParImpar.java

Diremos al usuario que introduzca un número, ahora le diremos si el número introducido es par o impar.

Numeros1al5.java

switch: Pediremos un numero del 1 al 5. Si de verdad han pulsado un numero del 1 al 5, diremos el numero en letra (uno, dos,..). Si no, diremos que han hecho trampa.

Producto.java

Programa que nos pide dos números y su producto. Una vez tecleado comprueba y nos dice si es correcto el producto.

Fecha.java

Programa que solicita una fecha (primero el día dd, luego el mes mm y finalmente el año aaaa), y nos dice si es correcta o no. Contemplar años bisiestos.