PROGRAMACIÓN

UNIDAD 2: Identificación de los elementos de un programa informático

Contenidos

1. Estructura y bloques fundamentales
2. Identificadores
3. Palabras reservadas
4. Tipos de datos
5. Literales
6. Variables. Declaración, inicialización y utilización
7. Constantes
8. Operadores y expresiones. Precedencia de operadores
9. Operaciones de entrada/salida de datos en Java
10. Cadenas de caracteres en Java
11. Conversiones de tipo. Implícitas y explícitas (casting)
12. Clase Math

1. Estructura y bloques fundamentales

Estructura de un programa Java

El código fuente de un programa en Java debe estar escrito en **ficheros de texto con extensión ".java"**.

Estos ficheros pueden editarse con cualquier **editor de texto sin formato** o con alguno de los programas y entornos de desarrollo específicos para la creación de aplicaciones.

Los programas o aplicaciones en Java se componen de una serie de ficheros **.class** que son ficheros en bytecode que contienen las clases del programa. Estos ficheros son generados por el compilador a partir de los ficheros **.java**. No tienen por qué estar situados en un directorio concreto, sino que pueden estar distribuidos en varios discos o incluso en varias máquinas.

Veamos a continuación el **programa más sencillo** que se puede hacer en Java. El resultado que se obtiene al ejecutar este programa es que se **muestra en pantalla el mensaje "Hola Mundo!"**.

1. Estructura y bloques fundamentales

Estructura de un programa Java

```java
package holamundo;

public class HolaMundo {
// programa HolaMundo
    public static void main(String[] args) {
        /* lo único que hace este programa es mostrar
        la cadena "Hola Mundo" por pantalla*/
        System.out.println("Hola Mundo!");
    }
}
```

Los comentarios

Existen comentarios de una línea solamente (//) y comentarios multilínea (/* */).

- // Estos comentarios comienzan en la doble barra y terminan hasta el final de la línea.
- /* */ Estos comentarios comienzan con los caracteres /* y terminan con los caracteres */ y se pueden extender múltiples líneas

1. Estructura y bloques fundamentales

Estructura de un programa Java

```java
package holamundo;

public class HolaMundo {
// programa HolaMundo
    public static void main(String[] args) {
        /* lo único que hace este programa es mostrar
        la cadena "Hola Mundo" por pantalla*/
        System.out.println("Hola Mundo!");
    }
}
```

```
1    package holamundo;
```

En la primera línea encontramos la palabra reservada package seguida del nombre del paquete al que pertenece la clase que estamos desarrollando.

1. Estructura y bloques fundamentales

Estructura de un programa Java

```
1   package holamundo;
2
3
4   public class HolaMundo {
5       public static void main(String[] args) {
6           System.out.println("Hola Mundo!");
7       }
8
9   }
```

```
4   public class HolaMundo
```

A continuación, se indica el nombre de la clase Java que estamos desarrollando. Todos los programas Java están formados por una o más clases. Es importante tener en cuenta que **el nombre de la clase debe corresponder exactamente con el nombre del fichero de texto** que contiene el código fuente. En el caso de este ejemplo, el código debe almacenarse en un fichero de texto denominado "HolaMundo.java".

Las llaves { } que puedes observar en el código indican el inicio y fin de cada bloque. Siempre deben ir en pareja, es decir, que por cada llave de apertura debe existir siempre una llave de cierre.

1. Estructura y bloques fundamentales

Estructura de un programa Java

```
1   package holamundo;
2
3
4   public class HolaMundo {
5       public static void main(String[] args) {
6           System.out.println("Hola Mundo!");
7       }
8
9   }
```

```
5       public static void main(String[] args)
```

El código Java en las clases se agrupa en métodos o funciones. Cuando Java va a ejecutar el código de una clase, lo primero que hace es buscar el método main de dicha clase para ejecutarlo. El método main tiene las siguientes particularidades:

- Es público (***public***). Esto es así para poder llamarlo desde cualquier lado.
- Es estático (***static***). Al ser static se le puede llamar sin tener que instanciar la clase.
- No devuelve ningún valor (modificador ***void***).
- Admite una serie de parámetros (***String [ ] args***) que en este ejemplo concreto no son utilizados.

1. Estructura y bloques fundamentales

Estructura de un programa Java

```
1   package holamundo;
2
3
4   public class HolaMundo {
5       public static void main(String[] args) {
6           System.out.println("Hola Mundo!");
7       }
8
9   }
```

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

Para sacar información por pantalla en Java se utiliza la clase System que puede ser llamada desde cualquier punto de un programa, la cual tiene un atributo out que a su vez tiene dos métodos muy utilizados: print() y println(). La diferencia entre estos dos últimos métodos es que en el segundo se añade un retorno de línea al texto introducido. Como se puede ver la instrucción termina en ;

(todas las instrucciones en Java terminan en ; salvo los cierres de llaves a los cuales no hace falta ponérselo pues se sobreentiende que se finaliza la instrucción).

2. Identificadores

Un identificador es un componente del lenguaje de programación que se encarga de identificar un elemento concreto, da nombre a un elemento del programa.

Usamos identificadores cuando creamos una variable, una constante, declaramos una nueva clase, creamos una función,…

2. Identificadores

Un identificador en Java se caracteriza por:

- Debe comenzar por una **letra**, el símbolo **$** o **subrayado** (_).
- Está constituido por la combinación de letras y números
- Existe distinción entre mayúsculas y minúsculas
- No se pueden elegir como identificadores palabras reservadas del lenguaje

Algunas recomendaciones:

- Aunque se pueden usar palabras acentuadas y el carácter ñ, no se recomienda.
- Cuando un identificador esté formado por dos o más palabras, escribiremos en mayúscula el primer carácter de la segunda palabra. Ej. **resultadoSuma**
- Si el identificador se encarga de dar nombre a una clase o a una interfaz lo denominaremos comenzando por una letra mayúscula. Ej. **Class Cuadrado**
- Al definir constantes usaremos palabras con todas sus letras en mayúsculas. Ej. **PI**

3. Palabras reservadas

El término **Palabra reservada** refiere a un conjunto de palabras que realizan una función concreta en el lenguaje. Estas no pueden ser usadas para desempeñar otra función que no sea para la que fueron diseñadas.

3. Palabras reservadas

Palabras reservadas de Java						
abstract	continue	for	new	switch	boolean	default
goto	null	synchronized	break	do	if	package
this	byte	doublé	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						

4. Tipos de datos

Java es un **lenguaje de programación tipado**, es decir, cada variable, atributo, constante o valor devuelto por una función debe encontrarse en un rango de elementos previamente establecido, debe ser de un tipo de datos concreto.

Un tipo de datos incluye un rango de valores que puede adoptar además de una serie de restricciones a aplicar sobre ellos, como operaciones que se pueden realizar.

4. Tipos de datos

Tipo de datos	Información representada	Rango	Descripción
byte	Datos enteros	-128 ⟷ +127	Se utilizan 8 bits (1 byte) para almacenar el dato.
short	Datos enteros	-32768 ⟷ +32767	Dato de 16 bits de longitud (independientemente de la plataforma).
int	Datos enteros	-2147483648 ⟷ +2147483647	Dato de 32 bits de longitud (independientemente de la plataforma).
long	Datos enteros	-9223372036854775808 ⟷ +9223372036854775807	Dato de 64 bits de longitud (independientemente de la plataforma).
char	Datos enteros y caracteres	0 ⟷ 65535	Este rango es para representar números en unicode, los ASCII se representan con los valores del 0 al 127. ASCII es un subconjunto del juego de caracteres Unicode.
float	Datos en coma flotante de 32 bits	Precisión aproximada de 7 dígitos	Dato en coma flotante de 32 bits en formato IEEE 754 (1 bit de signo, 8 para el exponente y 24 para la mantisa).
double	Datos en coma flotante de 64 bits	Precisión aproximada de 16 dígitos	Dato en coma flotante de 64 bits en formato IEEE 754 (1 bit de signo, 11 para el exponente y 52 para la mantisa).
boolean	Valores booleanos	true/false	Utilizado para evaluar si el resultado de una expresión booleanas es verdadero (true) o falso(false).

5. Literales

Un literal en un lenguaje de programación es la representación de los valores exactos que puede adoptar un tipo de datos concreto.

Un literal puede ser una expresión:

- De tipo de dato simple.
- El valor *null*.
- Un *string* o cadena de caracteres (por ejemplo “Hola Mundo”).

Ejemplos de literales en Java pueden ser ‘a’, 322, 3.1416, “pi” o “programación estructurada”.

5. Literales

Entre los literales de carácter encontramos una serie de valores especiales (secuencias de escape):

Secuencia de escape	carácter
‘\n’	Salto de línea
‘\t’	Tabulación
‘\r’	Retorno de carro
‘\’’	Comilla simple
‘\\’	Contrabarra
‘\”’	Comillas dobles
‘\f’	Salto de página
‘\0’	Carácter nulo

6. Variables. Declaración, inicialización y utilización. Almacenamiento en memoria

Una variable no es más ni menos que una zona de memoria donde se puede almacenar información del tipo que desee el programador.

Para declarar una variable necesitaremos:

- Concretar el tipo de datos que va a almacenar la variable
- El nombre o identificador que la definirá durante todo el programa
- Los valores o literales que podrán asignárseles, además de qué valor será su valor inicial.

6. Variables. Declaración, inicialización y utilización. Almacenamiento en memoria

En Java, la declaración de la variable se hará de las siguientes maneras:

```
tipo_dato nombre_variable;
tipo_dato nombre_variable=literal;
tipo_dato var1, var2, var3;
```

6. Variables. Declaración, inicialización y utilización

Al declarar una variable, si esta no está inicializada, adquiere un valor por defecto:

- Variables de tipo numérico toman el valor **cero**
- Variables booleanas toman en valor **false**
- Los tipos String (cadenas de caracteres) toman el valor nulo (**null**)
- Los caracteres toman el valor \u0000

6. Variables. Declaración, inicialización y utilización

Ejemplos de declaración y utilización de variables:

- **int** num=3;
- **float** reales=2.4f;
- **char** letra;
- **boolean** encontrado;

  encontrado = **true;**

6. Variables. Declaración, inicialización y utilización

Ámbito de utilización de las variables

```
public class Suma {

    static int n1=50;

    public static void main(String[] args) {
        int  n2=30, suma=0;
        suma=n1+n2;
        System.out.print("LA SUMA ES: ");
        System.out.println(suma);
    }

}
```

- las variables se declaran dentro de un bloque (por bloque se entiende el contenido entre las llaves { }) y son accesibles solo dentro de ese bloque.
- Las variables declaradas en el bloque de la clase como n1 se consideran miembros de la clase, mientras que las variables n2 y suma pertenecen al método main y solo pueden ser utilizados en el mismo.
- Las variables declaradas en el bloque de código de un método son variables que se crean cuando el bloque se declara, y se destruyen cuando finaliza la ejecución de dicho bloque.

7. Constantes

Una constante se encarga de almacenar un valor que no se modificará en ningún momento durante la ejecución del programa.

Las constantes se declaran siguiendo el siguiente formato:

final [static] <tipo de datos> <nombre de la constante> = <valor>;

- El calificador **final** identificará que es una constante
- La palabra **static** si se declara implicará que solo existirá una copia de dicha constante en el programa aunque se declare varias veces

Ejemplo:

final static double PI=3.141592;

8. Operadores y expresiones.

Operadores aritméticos

Se utilizan para realizar operaciones matemáticas

Operador	Uso	Operación
+	A + B	Suma
-	A - B	Resta
*	A * B	Multiplicación
/	A / B	División
%	A % B	Módulo o resto de una división entera

8. Operadores y expresiones.

Operadores relacionales

Con los operadores relacionales se puede evaluar la igualdad y la magnitud.

Operador	Uso	Operación
<	A< B	A menor que B
>	A > B	A mayor que B
<=	A<= B	A menor o igual que B
>=	A>= B	A mayor o igual que B
!=	A!= B	A distinto que B
= =	A= = B	A igual que B

8. Operadores y expresiones.

Operadores lógicos

Con los operadores lógicos se pueden realizar operaciones lógicas.

Operador	Uso	Operación
&& o &	A&& B o A&B	A AND B. El resultado será true si ambos operandos son true y false en caso contrario.
|| o |	A || B o A | B	A OR B. El resultado será false si ambos operandos son false y true en caso contrario.
!	!A	Not A. Si el operando es true el resultado es false y si el operando es false el resultado es true.
^	A ^ B	A XOR B. El resultado será true si un operando es true y el otro false, y false en caso contrario.

8. Operadores y expresiones.

Operadores de asignación

Operador	Uso	Operación
=	A = B	Asignación. Operador ya visto.
*=	A *= B	Multiplicación y asignación. La operación A*=B equivale a A=A*B.
/=	A /= B	División y asignación. La operación A/=B equivale a A=A/B.
%=	A %= B	Módulo y asignación. La operación A%=B equivale a A=A%B.
+=	A += B	Suma y asignación. La operación A+=B equivale a A=A+B.
-=	A -= B	Resta y asignación. La operación A-=B equivale a A=A-B.

8. Operadores y expresiones.

Operador asignación condicional

Nos permite asignar a una expresión un valor en función de la condición establecida.

?:	Asignación condicional	Sintaxis: var=(condición)?valor_si_verdad:valor_si_falso;

```
public class MenorDiez {

   final static int LIMITE = 10;

    public static void main(String[] args) {
        int valor = 4;
        String texto = (valor<LIMITE)?"Es menor":"Es mayor";
    }

}
```

8. Operadores y expresiones.

Operadores unitarios

Operador	Uso	Operación
~	~A	Complemento a 1 de A
-	-A	Cambio de signo del operando
--	A--	Decremento de A
++	A++	Incremento de A
!	! A	Not A (ya visto)

Los operadores incremento y decremento no proporcionan el mismo resultado si los colocamos delante o detrás de la variable que se desea incrementar

- x++ primero se usa el valor de x y luego se realiza el incremento
- ++x primero se produce el incremento y luego se usa la variable
- x-- primero se usa el valor de x y luego se realiza el decremento
- --x primero se produce el decremento y luego se usa la variable

8. Operadores y expresiones.

Operadores de bits

Operador	Uso	Operación
&	A & B	AND lógico. A AND B.
|	A | B	OR lógico. A OR B.
^	A ^ B	XOR lógico. A XOR B.
<<	A << B	Desplazamiento a la izquierda de A B bits rellenando con ceros por la derecha.
>>	A >> B	Desplazamiento a la derecha de A B bits rellenando con el BIT de signo por la izquierda.
>>>	A >>> B	Desplazamiento a la derecha de A B bits rellenando con ceros por la izquierda.

8. Operadores y expresiones.

Precedencia de los operadores

Mayor prioridad	[] ()
	++ -- ! ~
	new (tipo_dato)expresión
	* / %
	+ -
	<< >> >>>
	< > <= >=
	== !=
	&
	^
	|
	&&
	||
	?:
Menor prioridad	= += -= *= /= %= &= ^= |= <<= >>= >>>=

9. Operaciones de entrada/salida de datos

Hasta el momento estamos realizando aplicaciones de consola, es decir, tanto la entrada como la salida de datos la realizamos a través del teclado o la consola del sistema, los dispositivos de entrada/salida estándar.

Los objetos que referencian la entrada y salida estándar son **System.in** y **System.out** respectivamente.

9. Operaciones de entrada/salida de datos

Métodos para la salida de datos a través de la salida estándar

Hasta el momento estamos realizando aplicaciones de consola, es decir, tanto la entrada como la salida de datos la realizamos a través del teclado o la consola del sistema, los dispositivos de entrada/salida estándar.

- ***System.out.println("Texto");***

  Como parámetro del método println podemos usar cualquiera de los tipos de datos primitivos, arrays, objeto String,… Al final añadirá un salto de línea.

- ***System.out.print("Texto");***

  Hace lo mismo que println pero sin añadir el salto de línea al final

- ***System.out.printf("cadena de formato", var1,…,varn);***

  Permite escribir una cadena usando los valores de variables sin necesidad de utilizar el operador +

  Ejemplo:

  ```
  System.out.printf("Los números son: %d - %d, num1, num2);
  ```

9. Operaciones de entrada/salida de datos

Métodos para la salida de datos a través de la salida estándar

Los caracteres especiales que podemos usar con printf() son:

caracteres especiales	Significado
%d	Número entero
%c	Carácter
%s	Cadena de caracteres
%f	Número real de tipo float o doublé
%x.yf	Número float o double que debe mostrarse en formato x dígitos de parte entera e y dígitos de parte decimal
%x.ye %n	Número float o double que debe mostrarse en notación científica del tipo 1.1e+02. x establece el número de dígitos de la parte entera e y dígitos de parte decimal, incluyendo +e y –e
%o	Número entero que debe mostrarse en octal
%h	Número entero que debe mostrarse en hexadecimal

9. Operaciones de entrada/salida de datos

Métodos para la entrada de datos a través de la entrada estándar

El objeto que usamos para referirnos a la entrada estándar es System.in, que dispone de un único método read()

- ***System.in.read();***

  System.in.read() espera a que el usuario pulse una tecla y almacena el código ASCII correspondiente a la tecla pulsada.

Esta forma de recibir datos del teclado no es muy práctica, por lo que lo haremos utilizando alguna de las maneras que se indican en los siguientes apartados.

9. Operaciones de entrada/salida de datos

Métodos para la entrada de datos a través de la entrada estándar

InputStreamReader

Se encarga de tratar o manejar un flujo de entrada de datos, que sólo va a ser leído.

Sintaxis:

InputStreamReader nombre_variable = new InputStreamReader(flujo);

Ejemplo:

InputStreamReader flujoEntrada = new InputStreamReader(System.in);

Al incluir entre los paréntesis el objeto System.in le estamos indicando que vamos a recoger los datos del teclado.

InputStreamReader mantiene métodos de lectura que no son demasiado prácticos, para poder recoger información del teclado de forma más cómoda usaremos el objeto *BufferedReader*.

9. Operaciones de entrada/salida de datos

Métodos para la entrada de datos a través de la entrada estándar

BufferedReader

Usa el flujo de entrada definido por el objeto InputStreamReader.

Sintaxis:

BufferedReader nombre_variable = new BufferedReader(ObjetoInputStreamReader);

Ejemplo:

InputStreamReader flujoEntrada = new InputStreamReader(System.in);

BufferedReader teclado = new BufferedReader(flujoEntrada);

Podríamos crear el objeto InputStreamReader directamente dentro del paréntesis:

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

9. Operaciones de entrada/salida de datos

Métodos para la entrada de datos a través de la entrada estándar

Cuando se requieren utilizar otras clases, como es el caso de *BufferedReader* y *InputStreamReader*, debemos importarlas antes de la declaración de la clase. En nuestro caso, las clases *BufferedReader* y *InputStreamReader se e*ncuentran en el paquete java.io por lo que las importamos con la siguiente sintaxis:

```
import java.io.BufferedReader;

import java.io.InputStreamReader;
```

9. Operaciones de entrada/salida de datos

Métodos para la entrada de datos a través de la entrada estándar

```java
import java.io.IOException;

import java.io.BufferedReader;
import java.io.InputStreamReader;

public class EscribeNombre {

    public static void main(String[] args) throws IOException {

        InputStreamReader flujoEntrada = new InputStreamReader(System.in);
        BufferedReader teclado = new BufferedReader(flujoEntrada);

        System.out.println("Escribe tu nombre");
        String texto = teclado.readLine();
        System.out.println(texto);
    }

}
```

9. Operaciones de entrada/salida de datos

Métodos para la entrada de datos a través de la entrada estándar

Problema de *BufferedReader*

- Únicamente posee el método *readLine()* para leer la entrada
- readLine() siempre retorna String, de modo que para leer un número entero:
  - Leer primero como String
  - Convertir el String a entero usando el método *parseInt()* de la clase *Integer*.
  - Hay que ser muy cuidadosos con *Integer.parseInt()* pues si el usuario no ingreso un número sino letras, se disparará una excepción que debemos controlar (más adelante veremos esto con detalle)

9. Operaciones de entrada/salida de datos

Métodos para la entrada de datos a través de la entrada estándar. Clase Scanner.

La clase *Scanner* de la librería *java.util* es también muy sencilla para obtener datos de entrada del usuario, a diferencia de *BufferedReader*, *Scanner* sí posee un método para la lectura de números y para la lectura de texto que nos ayudarán a facilitar un poco las cosas.

Con el uso de Scanner ya no nos tenemos que preocupar por el tipo de dato, solo debemos usar el método adecuado según sea entero, String, float, etc.

Sintaxis:

```
Scanner nombre_variable = new Scanner(System.in);

String nombre_variable_string = sc.nextLine();

int nombre_variable_int = sc.nextInt();
```

9. Operaciones de entrada/salida de datos

Métodos para la entrada de datos a través de la entrada estándar. Clase Scanner.

La clase *Scanner* de la librería *java.util* es también muy sencilla para obtener datos de entrada del usuario, a diferencia de *BufferedReader*, *Scanner* sí posee un método para la lectura de números y para la lectura de texto que nos ayudarán a facilitar un poco las cosas.

Con el uso de Scanner ya no nos tenemos que preocupar por el tipo de dato, solo debemos usar el método adecuado según sea entero, String, float, etc.

Sintaxis:

```
Scanner nombre_variable = new Scanner(System.in);

String nombre_variable_string = sc.nextLine();

int nombre_variable_int = sc.nextInt();
```

9. Operaciones de entrada/salida de datos

Métodos para la entrada de datos a través de la entrada estándar. Clase Scanner.

Métodos de la clase Scanner	Ejemplo de uso
nextByte()	byte b = teclado.nextByte();
nextDouble()	double d = teclado.nextDouble();
nextFloat()	float f = teclado.nextFloat();
nextInt()	int i = teclado.nextInt();
nextLong()	long l = teclado.nextLong();
nextShort()	short s = teclado.nextShort();
next()	String p = teclado.next(); **next()** solo lee hasta donde encuentra un espacio.
nextLine()	String o = teclado.nextLine(); **nextLine()** lee toda la línea.

9. Operaciones de entrada/salida de datos

Métodos para la entrada de datos a través de la entrada estándar. Clase Scanner.

```java
import java.util.Scanner;


public class UsoScanner {
    public static void main(String[] args)
    {
        //Se crea el lector
        Scanner sc = new Scanner(System.in);

        //Se pide un dato al usuario
        System.out.println("Por favor ingrese su nombre");

        //Se lee el nombre con nextLine() que retorna un String con el dato
        String nombre = sc.nextLine();

        //Se pide otro dato al usuario
        System.out.println("Bienvenido " + nombre + ". Por favor ingrese su edad");

        //Se guarda la edad directamente con nextInt()
        int edad = sc.nextInt();

        //Nótese que ya no hubo necesidad de usar parseInt() pues nextInt nos retorna un entero derectamente
        System.out.println("Gracias " + nombre + " en 10 años usted tendrá " + (edad + 10) + " años."); //Operacion numerica con la edad

    }

}
```

10. Cadenas de caracteres en Java

En Java hemos visto que cuando queremos almacenar un valor entero definimos una variable de tipo int, si queremos almacenar un valor con decimales definimos una variable de tipo float. Ahora si queremos almacenar una cadena de caracteres (por ejemplo un nombre de una persona) debemos definir un objeto de la **clase String**.

Más adelante veremos en profundidad y detenimiento los conceptos de CLASE y OBJETO, por ahora solo nos interesa la mecánica para trabajar con cadenas de caracteres.

String nombre_variable;

10. Cadenas de caracteres en Java

Su lectura por teclado se hace llamando al método next() del objeto de la clase Scanner:

```
Scanner nombre_sc = new Scanner(System.in);

String nombre_variable = nombre_sc.next();
```

La primera salvedad que tenemos que hacer cuando utilizamos el método next() es que solo nos permite ingresar una cadena de caracteres con la excepción del espacio en blanco

Veamos que existe otro método llamado nextLine() que nos permite cargar espacios en blanco pero para su uso se complica cuando cargamos otras valores de tipo distinto a String (por ejemplo int, float etc.)

10. Cadenas de caracteres en Java

Lectura por teclado llamando al método nextLine() del objeto de la clase Scanner:

```
Scanner nombre_sc = new Scanner(System.in);
String nombre_variable = nombre_sc.nextLine();
```

Problema: si llamamos al método nextLine() y previamente hemos llamado al método nextInt()

Después de ejecutar el método nextInt() queda almacenado en el objeto de la clase Scanner el carácter "Enter" y si llamamos inmediatamente al método nextLine() este almacena dicho valor de tecla y continúa con el flujo del programa. Para solucionar este problema debemos generar un código similar a:

```
System.out.print("Ingrese edad:");
edad1=teclado.nextInt();
System.out.print("Ingrese el apellido y el nombre:");
teclado.nextLine();
apenom2=teclado.nextLine();
```

Como vemos llamamos al método nextLine() dos veces, la primera retorna la tecla "Enter" y la segunda se queda esperando que ingresemos el apellido y nombre (tener en cuenta que esto es necesario solo si previamente se llamó al método nextInt() o nextFloat().

10. Cadenas de caracteres en Java

Algunas operaciones básicas con cadenas

Comparación de cadenas

Para comparar si el contenido de dos String son iguales no podemos utilizar el operador ==

Debemos utilizar un método de la clase String llamado **equals** y pasar como parámetro el String con el que queremos compararlo:

boolean variable = cadena1.equals(cadena2);

El método equals retorna verdadero si los contenidos de los dos String son exactamente iguales, y falso en caso contrario.

En el caso que necesitemos considerar igual caracteres mayúsculas y minúsculas podemos utilizar el método **equalsIgnoreCase**:

boolean variable = cadena1.equalsIgnoreCase(cadena2);

10. Cadenas de caracteres en Java

Algunas operaciones básicas con cadenas

Concatenación de cadenas

El símbolo + nos va a permitir concatenar dos o más cadenas

También vamos a poder concatenar cadenas con otros tipos de datos: int, float,…

Esto nos facilita la edición de textos y mensajes:

```
System.out.print(apenom1);
System.out.print(" tiene ");
System.out.print(edad1);
System.out.println(" años.");
System.out.print(apenom2);
System.out.print(" tiene ");
System.out.print(edad2);
System.out.println(" años.");
```

=

```
System.out.print(apenom1 + " tiene " + edad1 + " años.");
System.out.print(apenom2 + " tiene " + edad2 + " años.");
```

10. Cadenas de caracteres en Java

Extraer una parte substring

String substring(): este método devuelve una nueva cadena que es una parte de esta cadena.

- Si solo se indica un parámetro, la “subcadena” comienza con el carácter índice especificado y se extiende hasta el final de esta string.
- Si se indican dos parámetros, la “subcadena” inicia en el primer valor y finaliza en el segundo – 1

Ejemplo:

```
String cadena = “Barcelona”;

String dos_primeros = fecha.substring(0,2); → Ba
```

11. Conversiones de tipo. Implícitas y explícitas (casting)

Cuando creamos una variable o usamos literales, podemos convertir estos para adaptarlos a variable de otros tipos.

- La conversión se realiza de forma implícita entre determinados tipos de datos
- La conversión se realiza de forma explícita mediante **casting**

11. Conversiones de tipo. Implícitas y explícitas (casting)

Conversiones de tipo implícitas

Los tipos de datos que ocupan un menor tamaño son convertidos implícitamente.

Por ejemplo, podremos asignar a una variable long el número 34 (entero) o podríamos asignar a una variable de tipo double un float o un int, pero no a la inversa.

11. Conversiones de tipo. Implícitas y explícitas (casting)

Conversiones de tipo explícitas mediante casting

La conversión mediante **casting** sigue la siguiente sintaxis:

*variable = **(tipo_de_dato)**variable_tipo_mayor;*

*variable = **(tipo_de_dato)**expresión;*

Ejemplos:

***byte** num=**(byte)**12;*

***char** caracter=**(char)**199;*

***int** numero=**(int)**3.4;*

11. Conversiones de tipo. Implícitas y explícitas (casting)

Funciones parseInt, parseDouble

parseInt

Convierte una cadena de texto en un número entero.

Ejemplo:

```
int numero = Integer.parseInt("12");
```

parseDouble

Convierte una cadena de texto en un número double.

Ejemplo:

```
double numero = Double.parseDouble("12.12");
```

12. Clase Math

Las funciones matemáticas en Java vienen definidas en la clase Math. En la siguiente tabla se muestran los métodos más importantes de la clase Math, con ejemplos de uso de cada una de ellas.

Función matemática	Significado	Ejemplo de uso	Resultado
abs	Valor absoluto	int x = Math.abs(2.3);	x = 2;
atan	Arcotangente	double x = Math.atan(1);	x = 0.78539816339744;
sin	Seno	double x = Math.sin(0.5);	x = 0.4794255386042;
cos	Coseno	double x = Math.cos(0.5);	x = 0.87758256189037;
tan	Tangente	double x = Math.tan(0.5);	x = 0.54630248984379;
exp	Exponenciación neperiana	double x = Math.exp(1);	x = 2.71828182845904;
log	Logaritmo neperiano	double x = Math.log(2.7172);	x = 0.99960193833500;
pow	Potencia	double x = Math.pow(2.3);	x = 8.0;
round	Redondeo	double x = Math.round(2.5);	x = 3;
random	Número aleatorio	double x = Math.ramdom();	x = 0.20614522323378;
floor	Redondeo al entero menor	double x = Math.floor(2.5);	x = 2.0;
ceil	Redondeo al entero mayor	double x = Math.ceil(2.5);	x = 3.0;

12. Clase Math

Las funciones matemáticas en Java vienen definidas en la clase Math. En la siguiente tabla se muestran los métodos más importantes de la clase Math, con ejemplos de uso de cada una de ellas.

Función matemática	Significado	Ejemplo de uso	Resultado
abs	Valor absoluto	int x = Math.abs(2.3);	x = 2;
atan	Arcotangente	double x = Math.atan(1);	x = 0.78539816339744;
sin	Seno	double x = Math.sin(0.5);	x = 0.4794255386042;
cos	Coseno	double x = Math.cos(0.5);	x = 0.87758256189037;
tan	Tangente	double x = Math.tan(0.5);	x = 0.54630248984379;
exp	Exponenciación neperiana	double x = Math.exp(1);	x = 2.71828182845904;
log	Logaritmo neperiano	double x = Math.log(2.7172);	x = 0.99960193833500;
pow	Potencia	double x = Math.pow(2.3);	x = 8.0;
round	Redondeo	double x = Math.round(2.5);	x = 3;
random	Número aleatorio	double x = Math.ramdom();	x = 0.20614522323378;
floor	Redondeo al entero menor	double x = Math.floor(2.5);	x = 2.0;
ceil	Redondeo al entero mayor	double x = Math.ceil(2.5);	x = 3.0;