MÓDULO PROFESIONAL PROGRAMACIÓN

UD 1 – Elementos de un programa informático

Tema 1.2. Manipulación básica de datos

Estructura de un programa en java

```
// El programa Hola Mundo en Java
public class HolaMundo {
        public static void main (String[] args){
            System.out.println("¡Hola Mundo!");
        }
}
```

Estructura de un programa en java

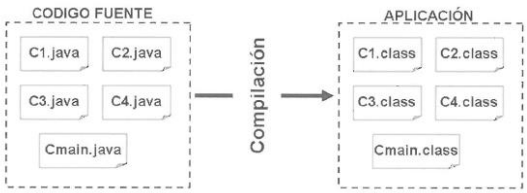
Java es un lenguaje de programación O.O. Con lo cual nuestros programas en realidad son **clases**, una clase se indica mediante la palabra reservada **class**.

El **código fuente** se guarda en archivos con el mismo nombre que la clase que contienen y con extensión **.java**

El compilador genera un archivo intermedio de clase **.class** por cada una de las clases definidas en el archivo fuente.

Estructura de un programa en java

Las aplicaciones **"ejecutables"** deben contener forzosamente el método **main**. Una clase no tiene por qué tener un método main, pero para que empiece a ejecutarse la aplicación debe existir uno. Dicho método se conoce como punto de entrada de la aplicación java.



EJECUCIÓN

java Cmain.class

Estructura de un programa en Java

Vamos a analizar cada uno de los elementos que componen el siguiente código antes de seguir avanzando:

- Librerías y/o programas importados.
- Comentarios
- Bloques de código
- Identificadores
- Sentencias
- Metacaracteres

Estructura de un programa en Java

Librerías y/o programas importados

Una librería es un conjunto de recursos. Si queremos utilizar estos recursos en nuestras aplicaciones, es necesario importar la librería de java en la que se encuentran. Para esto se utiliza la palabra reservada import seguido del nombre de la librería a importar.

Cuando importamos una librería podemos importar sólo una parte de ella indicando que recurso concreto queremos importar, o podemos importarla entera indicándolo con un *

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

Además de importar librerías propias de java, también podemos importar nuestros propios programas para poder reutilizarlos

Estructura de un programa en Java

Comentarios

Son líneas de texto insertadas en el programa para documentarlo y facilitar su lectura y comprensión por parte del programador. Los comentarios en Java se indican de 2 formas:

Comentarios de una sola línea:

```
// primer número a sumar
```

De varias líneas:

```
/* Éste es un comentario
   Tradicional. Puede
   dividirse en muchas líneas */
```

Estructura de un programa en Java

Bloques de código:

- Son el principal mecanismo de encapsulamiento, y se forman con un grupo de sentencias y otros bloques de código delimitados por una llave de apertura y una de cierre: {}
- Para mejorar la legibilidad de un código son imprescindibles el uso de comentarios y de las tabulaciones. Los comentarios aclaran la utilizad de cada método o programa, incluso instrucciones complejas. Mientras que las tabulaciones ayudan a entender que partes del código pertenecen a otras.
- Por tabulaciones entendemos a sangrar las líneas de código que están dentro de un mismo juego de {}

Estructura de un programa en Java

Identificadores:

Son los nombres que se asigna a las clases, métodos, atributos, objetos, variables etc. para poder diferenciarlos y referirse a ellos dentro del programa. Deben respetar las siguientes normas:

- No pueden contener ni espacios ni caracteres especiales salvo "_".
- No pueden empezar por un número.
- Java distingue mayúsculas de minúsculas, por lo que "Nombre" y "nombre" son identificadores diferentes.
- Se evita el uso de acentos y la letra ñ.

Estructura de un programa en Java

Identificadores:

Además de las normas anteriores, suelen respetarse las siguientes convenciones para asignar identificadores:

- El nombre de una clase siempre empieza por mayúscula: **Empleado**.
- El nombre de un método, variable, atributo etc. siempre empieza por minúscula: **salario**.
- Si el identificador está formado por más de una palabra, a partir de la segunda las iniciales deberán ser mayúsculas: **salarioEmpleado**
- Los nombres de las clases deberán ser sustantivos, los de los métodos verbos. Nombre de método: **calculaSalario**
- Los nombres de las variables y atributos deben expresar con claridad su contenido.

Estructura de un programa en Java

Sentencias:

Son las distintas ordenes que debe ejecutar el programa y terminan siempre con un punto y coma ";" podemos distinguir las siguientes:

- E/S: Piden o muestran algún dato por un dispositivo de entrada/salida.
- Asignaciones y Expresiones: Las expresiones y sentencias de asignación guardan un valor o el resultado de una operación en un atributo o variable.
- Condicionales: Expresan una condición para definir el flujo del programa. If, ifelse y switch.
- Bucles: Sentencias que se encargan de repetir una o varias sentencias un número determinado de veces. For, while, do-while.
- Saltos: Llevan al compilador a un punto específico del programa. Break, continue, return

Estructura de un programa en java

Metacaracteres:

Son una serie de caracteres especiales, que sirven para el control y la significación puntual en las sentencias y los bloques de código:

- ( ) [ ] { } \^ $?

Introducción a la programación en java

Variables

Una variable, es una zona de la memoria del computador que se reserva a lo largo de un programa informático donde se puede almacenar un valor.

Si quieres usar el resultado de una expresión más adelante en tu programa, lo puedes almacenar en una variable.

Introducción a la programación en java

Sentencia de asignación

Almacenas valores en variables con una sentencia de asignación.

Una sentencia de asignación consiste del nombre de la variable, el signo = y el valor a ser almacenado:

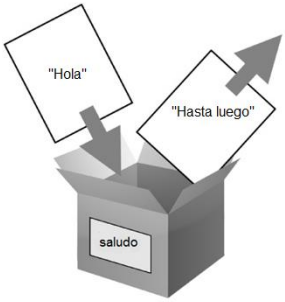
```
edad=32;
anio=2018;
id = 1234;
horaMilisegundos = 77722349;

estatura = 1.80F;
resultado = 932.45690872;
```

Introducción a la programación en java

Sentencia de asignación

Cuando a una variable se le asigna un nuevo valor el valor anterior se pierde.



Introducción a la programación en java

Variables

- Poseen un **identificador** para poder **referirse y acceder** a ellas.
- Su **valor puede cambiarse** o modificarse a lo largo de un programa cuantas veces sea necesario.
- Son de un **tipo** determinado que debe indicarse al comenzar a trabajar con ella
- Una variable se crea **una sola** vez por programa, pero se puede usar muchas veces.

Introducción a la programación en java

Variables

- El nombre de una variable es su identificador, y como tal cumple con las **reglas de los identificadores.**
- No pueden utilizarse como nombres de variables las **palabras reservadas** de Java

Introducción a la programación en java

Palabras reservadas

abstract	boolean	break	byte	case
catch	char	class	continue	default
do	double	else	extends	false
final	finally	float	for	if
implements	import	instanceof	int	interface
long	native	new	null	package
private	protected	public	return	short
static	super	switch	syncroniced	this
throw	throws	transient	true	try
void	volatile	while	var	rest
byvalue	cast	const	future	generic
goto	inner	operator	outer	

Introducción a la programación en java

Variables

Según el **tipo de dato** que pueden almacenar, podemos distinguir dos tipos principales de variables:

a) Variables de tipos **primitivos**: Poseen un valor único y pueden ser entero, decimal, carácter, lógico…

b) Variables de **referencia**: Las variables de referencia son objetos de una determinada clase, pudiendo hacer referencia a más de un valor.

```
//Variables de tipo primitivo
int variableEntera;
char letra;
boolean respuesta;
double numeroDecimal;

//Variables de referencia a objetos
Scanner objetoLectura;
String palabra;
File fichero;
```

Un **tipo de dato** es una categoría de valores, y cada valor es exactamente de un tipo de dato.

Introducción a la programación en java

Declaración de una variable

Declarar una variable es el acto de crear una variable. En Java, tan solo tenemos que poner el tipo de datos de la variable seguido del identificador, separados por un espacio

```
int numero;
```

Introducción a la programación en java

Inicialización de una variable

El concepto de inicializar una variable significa asignarle un valor por primera vez en el programa.

```
int numero = -3456;
```

Una variable puede inicializarse en el momento de su declaración (creación) o en un momento posterior.

Introducción a la programación en java

Inicializar variables en java, significa asignarlas algún valor, ya sea de tipo número, lógico o de otro tipo.

Hay que tener en cuenta donde están posicionadas dichas variables en el código del programa, es decir si son **miembros de la clase** (Están situadas dentro de una clase, pero fuera de cualquier método que tenga la clase) ó son **variables locales** a un método (Están dentro de un método declaradas).

Introducción a la programación en java

Podemos inicializar cualquier variable con independencia de donde este situada, pero para las **variables miembros de una clase,** nos podemos ahorrar su inicialización ya que el compilador de java las inicializa de la siguiente manera:

- Las variables numéricas las inicializa a **0** (cero)
- Las variables de caracteres, las inicializa con **'\0'** (carácter cero)
- Las variables tipo boolean java las inicializa como **false**.
- Las referencias a cadenas de caracteres y a objetos las inicializa con **null**.

Esto lo revisaremos más adelante.

Tipos de datos primitivos

Tipos de datos. Tipos primitivos

Los tipos de datos primitivos son similares a los tipos de datos que se pueden encontrar en cualquier otro lenguaje de programación, aunque sea no orientado a objeto.

Las variables de tipos primitivos almacenan un dato simple y deben ser declaradas antes de ser utilizadas. En la declaración, se determina el tipo de que se trata y solo puede realizarse una vez.

```
byte edad;
short anio;
int id;
long horaMilisegundos;

float estatura;
double resultado;
```

Tipos de datos primitivos

Tipos de datos primitivos. Enteros

El tipo de dato **entero** representa un valor numérico, positivo o negativo, sin decimal.

- Ejemplos de enteros: 3, 0, -345, 138764, -345002, etc.
- Ejemplos de datos que se suelen representar con un entero: edad, día del mes, año, número de hijos, etc.

Tipo	Bytes para almacenar datos	Nombre	Rango de valores
byte	1	byte	-128 a 127
short	2	entero corto	-32768 a 32767
int	4	entero	-2^{31} a $2^{31} - 1$
long	8	entero largo	-2^{63} a $2^{63} - 1$

Tipos de datos primitivos

Tipos de datos primitivos. Reales

- El tipo de dato **real** representa un valor numérico, positivo o negativo, con decimales.
- Ejemplos de reales: 2.25, 4.0, -9653.3333, 100.0003, etc.
- Ejemplos de datos que se suelen representar con un real: un precio en euros, la distancia entre dos ciudades, el peso de una persona, etc.
- Las variables reales se denominan también variables de coma flotante.

Tipo	Bytes para almacenar datos	Nombre	Rango de valores
float	4	real simple precisión	1.4×10^{-45} a 3.4×10^{38}
double	8	real doble precisión	4.9×10^{-324} a 1.8×10^{308}

Actividad

Practicaremos sobre tipos de datos enteros y reales

```
public class Primitivos {

    public static void main(String[] args) {

            byte edad=32;
            short anio=2018;
            int id = 1234;
            long horaMilisegundos = 77722349;

            float estatura = 1.80F;
            double resultado = 932.45690872;

            System.out.println(edad);
            System.out.println(anio);
            System.out.println(id);
            System.out.println(horaMilisegundos);
            System.out.println(estatura);
            System.out.println(resultado);

    }
}
```

Actividad

```
public class Primitivos {

    public static void main(String[] args) {

            byte edad=32;
            short anio=2018;
            int id = 1234;
            long horaMilisegundos = 77722349;

            float estatura = 1.80F;
            double resultado = 932.45690872;

            System.out.println("Edad: " + edad);
            System.out.println("Año: " + anio);
            System.out.println("Identificador: " + id);
            System.out.println("Hora en milisegundos: " + horaMilisegundos);
            System.out.println("Estatura: " + estatura);
            System.out.println("Resultado: " + resultado);

    }
}
```

Actividad

```java
public class Primitivos {

    public static void main(String[] args) {

            byte edad=32;
            short anio=2018;
            int id = 1234;
            long horaMilisegundos = 77722349;

            //por defecto un número con decimales es double. Si queremos que se tome como un tipo float
            //hay que agregar a la derecha del mismo la letra F;
            float estatura = 1.80F;

            //No es necesario poner la letra D a la derecha del número porque por defecto es double
            double resultado = 932.45690872;

            System.out.println("Edad: " + edad);
            System.out.println("Año: " + anio);
            System.out.println("Identificador: " + id);
            System.out.println("Hora en milisegundos: " + horaMilisegundos);
            System.out.println("Estatura: " + estatura);
            System.out.println("Resultado: " + resultado);

    }
}
```

Actividad

Si bien todavía no hemos hablado del tipo de dato cadena. Observa que en la siguiente sentencia

```java
System.out.println("Edad: " + edad);
```

- El datos de la izquierda del operador "+" es una cadena de caracteres "Edad: " y el de la derecha un entero, el que almacena la variable edad.
- Evidentemente no se pueden "sumar". Sin embargo en Java se produce en ese momento una conversión automática de tipos:
- El operador "+" une a la cadena "Edad: " el resultado de convertir el valor numérico almacenado en la variable edad en una cadena de caracteres, "32".

Tipos de datos primitivos

Tipos de datos primitivos. Boolean y Char

Los tipo boolean (lógicos) y char (caracteres) completan la colección de tipos primitivos.

Las variables de tipo **boolean** toman el valor: true o false y ocupan un byte de memoria.

Las variables de tipo **char** se utiliza para almacenar caracteres individuales y ocupan dos bytes de memoria.

Ejemplos: `'a','?' o '4'`

En realidad está considerado también un **tipo numérico,** si bien su representación habitual es la del carácter cuyo código almacena.

```
public class Primitivos {

    public static void main(String[] args) {

            //Enteros
            byte edad=32;
            short anio=2018;
            int id = 1234;
            long horaMilisegundos = 77722349;

            //por defecto un número con decimales es double. Si queremos que se tome como un tipo float
            //hay que agregar a la derecha del mismo la letra F;
            float estatura = 1.80F;

            //No es necesario poner la letra D a la derecha del número porque por defecto es double
            double resultado = 932.45690872;

            System.out.println("Edad: " + edad);
            System.out.println("Año: " + anio);
            System.out.println("Identificador: " + id);
            System.out.println("Hora en milisegundos: " + horaMilisegundos);
            System.out.println("Estatura: " + estatura);
            System.out.println("Resultado: " + resultado);

            boolean esCierto;
            char letra;

            esCierto=false;
            letra ='a';

            System.out.println("Es cierto: " + esCierto);
            System.out.println("Letra: " + letra);
    }
}
```

Tipos de datos primitivos

Los datos tipo carácter se escriben entre comillas simples, así, de esta forma, el carácter a se tiene que escribir como 'a'.

Si se escribe "a" es tomado como una cadena de caracteres, aunque contenga una sola letra.

Tipos de datos primitivos

Para entender completamente el tipo de dato **char** hay que conocer el esquema de codificación **Unicode**

Unicode se inventó para superar las limitaciones de los esquemas tradicionales de codificación de caracteres.

Antes de Unicode, había muchos estándares diferentes: ASCII en los Estados Unidos, ISO 8859-1 para las lenguas idiomas europeos, KOI-8 para el ruso, GB18030 y BIG-5 para el chino, y así sucesivamente.

Tipos de datos primitivos

Esto causaba dos problemas:

- En primer lugar, un valor de código concreto corresponde a diferentes letras en los distintos esquemas de codificación.
- En segundo lugar, las codificaciones de las lenguas con grandes conjuntos de caracteres tienen una longitud variable: algunos caracteres comunes se codifican en un solo byte, otros requieren dos o más bytes.

Tipos de datos primitivos

Unicode fue diseñado para resolver estos problemas.

Unicode es un estándar de codificación de caracteres diseñado para facilitar el tratamiento informático, transmisión y visualización de textos de numerosos idiomas.

En Java, las variables de tipo **char utilizan 16 bits y codifican caracteres según el formato Unicode** (la codificación ASCII que utilizaba 8 bits)

Tipos de datos primitivos

Caracteres de escape

Dentro de los literales de tipo caracter tenemos una serie de caracteres que no se pueden representar de forma escrita: las tabulaciones, los saltos de línea, las comillas dobles o simples, etc.

Para poder representar esos caracteres, se usa lo que se conoce como caracteres de escape

Tipos de datos primitivos

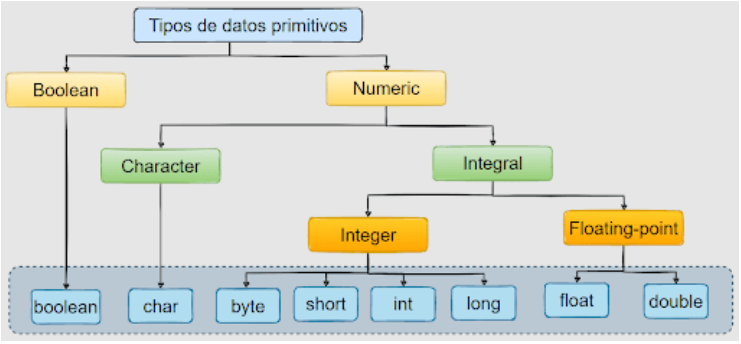
Caracteres de escape

Caracter de escape	Descripción
\n	Salto de linea
\t	Una tabulación
\b	Retroceder un caracter
\r	Retorno de carro
\'	Comilla simple
\"	Comilla doble
\\	Contrabarra

```
char saltoLinea = '\n';
char tabulacion = '\t';
```

Introducción a la programación en java

Resumen tipos primitivos



Introducción a la programación en java

Constantes

Las constantes son un tipo especial de variable, poseen un valor fijo que no se puede cambiar en el transcurso de la ejecución de un programa. Estas se declaran mediante el modificador final.

```
final int IVA = 21;
```

Como regla adicional, las constantes suelen declararse en mayúsculas para indicar que lo son.

Cadenas de caracteres

Cadenas de caracteres

Más adelante veremos en profundidad las cadenas de caracteres, pero nos sirve introducirlo ahora.

En java no existen primitivas que permitan almacenar una cadena de caracteres, por lo que hay que recurrir a objetos que desempeñen esta labor.

El concepto de **objeto y clase será introducido más adelante**, si bien avanzaremos aquí algunas ideas básicas que nos permiten utilizar cadenas de caracteres.

En este apartado se utilizarán objetos de la clase **String**

Cadenas de caracteres

String:

- Para diferenciarlas de los caracteres van entre comillas dobles "a".
- "a" es una cadena de texto, 'a' es un carácter.
- Ejemplos: "hola", "Esto es una cadena", "123", " ", ""
- Su valor por defecto es "" o null

Cadenas de caracteres

Habíamos dicho que las variables que almacenan direcciones de objetos se denominan variables de referencia.

Para definir una variable que utilice como referencia a un objeto, se pone a su izquierda el nombre de la clase a la que pertenecerá el objeto, tal como:

```
String nombre;
```

Con esta sentencia, declaramos una variable que permite referenciar un objeto de la clase String.

Cadenas de caracteres

Se verán algunos métodos de la clase String para realizar las operaciones sobre cadenas de caracteres, tales como:

- Calcular su longitud
- Extraer determinados caracteres
- Investigar que carácter ocupa la posición inicial o final de la cadena
- etc

Actividad

```java
public class Cadenas {

    public static void main(String[] args) {

        String nombre = "Francisca";

        int longitud = nombre.length();
        System.out.println("Nombre: " + nombre + " tiene " + longitud + " caracteres");

        int posicion = nombre.indexOf("a");
        System.out.println("Nombre: " + nombre + " tiene la letra 'a' en la posicion " + posicion);

        System.out.println("Empieza por: " + nombre.substring(0,4));
        System.out.println("Termina por: " +  nombre.substring(4));
    }
}
```

Cadenas de caracteres

El programa Cadenas crea el objeto nombre de la clase String al que le asigna la cadena de caracteres "Francisca".

El método length calcula la longitud de la cadena. El método indexOf("a") devuelve en qué posición está la primera letra "n" empezando desde la izquierda.

El método substring(0,4) Devuelve la subcadena que empieza en la posición 0 y termina en la 4 (sin incluirla). En cambio substring(4) devuelve la subcadena desde la posición 4 al fin de la cadena.

https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

Operadores

Siempre que se utilice un método perteneciente a un objeto, se pondrá el nombre del objeto seguido del nombre del método separados por un punto

nombreObjeto.método()

El método siempre terminará con dos paréntesis tenga argumentos o no. Los argumentos son datos que se envían al método para que sean utilizados en su ejecución.

Operadores

Operadores y las expresiones

Una expresión es una combinación de operadores y operandos que se evalúa generándose un único resultado de un tipo determinado.

precio*cantidad

Los operadores son instrucciones que se usan junto a las operandos (variables) para construir expresiones.

El resultado de una expresión es un dato numérico, lógico o una cadena de caracteres.

La sentencia de asignación permite evaluar una expresión y asignar el resultado a una variable

Operadores

Tipos de operadores

Tipo	Descripción
Aritméticos	Permiten realizar una operación aritmética entre datos numéricos.
Relacionales	Permiten comparar números o caracteres.
Asignación	Permiten transferir datos desde una variable a otra, con o sin aplicación de alguna transformación.
Lógicos	Permiten realizar expresiones lógicas (de tipo boolean), es decir, permiten unir varias expresiones lógicas en otra expresión lógica más compleja.
Operadores a nivel de bits (*bitwise*)	Permiten modificar o comparar valores numéricos a nivel de bits.
Operador `instanceof`	Permite investigar si un objeto es de una determinada clase.
Operador `this`	Permite identificar la clase propia.
Operador `new`	Permite crear un objeto.
Operador `?:`	Permite una asignación condicionada.
Operador `->`	Permite crear una expresión lambda
Operador `::`	Permite referenciar un método estático
Operador `<>`	Permite la gestión de objetos genéricos

Operadores

Operadores aritméticos

Denominación	Operador	Ejemplos
Multiplicación	*	precio*cantidad
Suma	+	peras + manzanas
Diferencia	-	ingresos - gastos
División	/	resultado / 10
Resto de la división entera (módulo)	%	numero%2
Incremento	++	tareasHechas++
Decremento	--	tareasPorHacer--
Cambio de signo	-x	-dinero

Operadores

Ejemplo:

```
x = 2;
x = x +1;
System.out.println("x = " + x);
```

¿Resultado?

La expresión x+1 es evaluada y el resultado es asignado a x.

Nota que x=x+1 no es una ecuación, sino una sentencia que significa que el valor antiguo de x se le añade 1 y se almacena de nuevo en x

Operadores

```
x = x+1;
```

Esta misma sentencia podría realizarse con el operador += de la siguiente forma

```
x+=1;
```

Es decir que al valor de x se le añade 1 y se guarda otra vez en x.

O también se podría poner de la siguiente forma

```
x++;
```

En la que el valor x se ve incrementado en una unidad a través del operador ++

Operadores

Operadores aritméticos incrementales

Incrementan o decrementan en una unidad el valor de la variable:

- ++ : incrementa el valor en 1
- -- : decrementa el valor en 1

```
int numero1 = 1;
numero1 ++; //Ahora numero1 vale 2: numero1 = numero1 + 1

numero1 = 7;
numero1 --; //Ahora numero1 vale 6: numero1 = numero1 -1
```

Operadores

Operadores aritméticos incrementales (ejemplos)

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo

++	Incremento
	i++ primero se utiliza la variable y luego se incrementa su valor
	++i primero se incrementa el valor de la variable y luego se utiliza
		4++
		a=5;
		b=a++;
		a=5;
		b=++a;
			5
			
			a vale 6 y b vale 5
			
			a vale 6 y b vale 6

--	decremento	4--	3

Actividad

```
num2= 5 num3= 5
num1= 5 num2= 4
num1= 4 num3= 4
```

Muestra como funciona el operador de decremento

```
public class Decremento {
    public static void main(String[] args) {
        int num1, num2 = 5 , num3 = 5;
        System.out.println("num2= " + num2 + " num3= " + num3);

        num1=num2--;
        System.out.println("num1= " + num1 + " num2= " + num2);

        num1=--num3;
        System.out.println("num1= " + num1 + " num3= " + num3);
    }
}
```

Operadores

Operadores aritméticos

Si en una sentencia se incluye una expresión, esta es evaluada antes de ejecutar la sentencia. Por ejemplo:

```
precio = 23.4;
cantidad = 2;
System.out.println("Cantidad total =  " + precio * cantidad);
```

Resultado:

Cantidad total = 46.8

Operadores

Sin embargo la sentencia

```
System.out.println("Cantidad total =  " + precio + cantidad);
```

Da como resultado

Cantidad total = 23.42.0

Ya que el operador "+" que hay entre las comillas y la variable precio se convierte en el operador de unión de cadenas al ser forzado por el dato que hay a la izquierda, que es una cadena de caracteres.

Lo mismo ocurre al otro operador por lo que los valores no se suman, sino que se unen luego de ser convertidos a cadenas de caracteres.

Para forzar la operación de suma debe encerrarse la operación entre paréntesis.

```
System.out.println("Cantidad total =  " + (precio + cantidad));
```

Operadores

Operador de concatenación de cadenas

El operador concatenación +, es un operador binario que devuelve una cadena resultado de concatenar las dos cadenas que actúan como operandos. Si sólo uno de los operandos es de tipo cadena, el otro operando se convierte implícitamente en tipo cadena.

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
+	Operador concatenación	"Hola" + "Juan"	"HolaJuan"

Operadores

Operadores de relación (comparativos)

Permiten comparar dos datos numéricos y el resultado es verdadero o falso. El lenguaje Java representa como verdadero el valor true y falso el valor false.

Denominación	Operador	Ejemplos
Mayor que	>	estatura1 > estatura2
Mayor o igual que	>=	estatura1 >= estatura2
Menor que	<	nota < 4
Menor o igual que	<=	nota <= 4
Igual que	==	resultado == 0
Distinto de	!=	contador != 100

Operadores

Operadores de lógicos (boobleanos)

Los operadores lógicos permiten concatenar expresiones lógicas con objeto de evaluar si es cierto o falso el conjunto de expresiones

Por ejemplo:

x=60;

x > 5 && x<=50

Denominación	Operador	Ejemplos
Conjunción (y) AND	&&	x > 5 && x <=50
Disyunción (o) OR	||	x < 20 || x >=100
Negación lógica (no) NOT	!	!a

Operadores

x=60;

x > 5 && x<=50

TABLA DE VERDAD DE LA OPERACIÓN LÓGICA AND

Condición 1	Condición 2	Resultado
FALSO	FALSO	FALSO
FALSO	VERDADERO	FALSO
VERDADERO	FALSO	FALSO
VERDADERO	VERDADERO	VERDADERO

TABLA DE VERDAD DE LA OPERACIÓN LÓGICA OR

Condición 1	Condición 2	Resultado
FALSO	FALSO	FALSO
FALSO	VERDADERO	VERDADERO
VERDADERO	FALSO	VERDADERO
VERDADERO	VERDADERO	VERDADERO

TABLA DE VERDAD DE LA OPERACIÓN LÓGICA NOT

Condición 1	Resultado
FALSO	VERDADERO
VERDADERO	FALSO

Operadores

Operadores de asignación

Permiten asignar un valor a una variable

variable= expresión

x=2;

x=x+1;

Operadores

Operador	Denominación	Ejemplos		Expresión equivalente
=	Asignación	x=2;	Asigna el valor	
+=	Incremento	x+=5;	Incrementa x en 5	x=x+5
-=	Decremento	x-=5;	Disminuye x en 5	x=x-5
*=	Multiplicación	x*=5;	Multiplica x en 5	x=x*5
/=	División	x /= 5;	Divide x por 5	x=x/5
%=	Resto de división	x %=5;	Resto de la división x/5	x= x%5

Por tanto existen operadores de asignación que además realizan operaciones antes de la asignación.

Operadores

En el lenguaje Java está permitido realizar múyiples asignaciones en una sola sentencia, por ejemplo:

```
a=b=c=d= z +24;
```

Actividad

```
public class OperadoresAsignacion {

    public static void main(String[] args) {

        double capital = 500.0;
        double interes = 6.25;

        double rentaSimple;

        rentaSimple=capital*interes/100;

        capital +=rentaSimple;

        System.out.println("Intereses = " + rentaSimple);
        System.out.println("Acumulado = " + capital);

    }
}
```

Operadores

El operador condicional

Este operador ternario tomado de C/C++ permite devolver valores en función de una expresión lógica. Sintaxis:

```
expresionLogica ? expresion_1 : expresion_2
```

Si el resultado de evaluar la expresión lógica es verdadero, devuelve el valor de la primera expresión, y en caso contrario, devuelve el valor de la segunda expresión.

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo

?:	operador condicional	`a = 4;`
		`b = a == 4 ? a+5 : 6-a;`
		`b = a > 4 ? a*7 : a+8;`
			`b vale 9`
			`b vale 12`

Operadores

El operador condicional

La sentencia de asignación:

```
valor = (expresionLogica ? expresion_1 : expresion_2);
```

como se verá en la próxima unidad didáctica es equivalente a:

```
if (expresionLogica)
    valor = expresion_1;
else
    valor = expresion_2
```

Actividad

Escribe una clase **ParImpar** que dado que asigne a la variable numero el valor 33 y determine si es par o impar mostrando en pantalla el resultado. Debe usar el **operador condicional** mencionado en la diapositiva anterior.

Precedencia de operadores

Precedencia de operadores

En una expresión pueden aparecer varios operadores y el valor final de la expresión puede variar en función del orden de ejecución de los operadores

Por ejemplo,

```
System.out.println(5-1*2);
```

Resultado?

```
System.out.println((5-1)*2);
```

Resultado?

Precedencia de operadores

Precedencia de operadores

El orden de ejecución viene dado por la prioridad de ejecución de unos operadores frente a otros. Esta prioridad se recoge en la siguiente tabla. Está ordenada de arriba hacia abajo por prioridad

Precedencia de operadores

Descripción	Operadores
operadores posfijos	op++ op--
operadores unarios	++op --op +op -op ~ !
multiplicación y división	* / %
suma y resta	+ -
desplazamiento	<< >> >>>
operadores relacionales	< > <= =>
equivalencia	== !=
operador AND	&
operador XOR	^
operador OR	|
AND booleano	&&
OR booleano	||
condicional	?:
operadores de asignación	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Precedencia de operadores

Dentro de una expresión que contenga dos o más operadores de la misma prioridad, estos se ejecutan de izquierda a derecha. Las expresiones entre paréntesis se evalúan primero y, dentro de estas, se evalúan las expresiones entre paréntesis más interiores y, a continuación, las externas al paréntesis.

Por ejemplo:

```
x = (((3 + 2) * 4) – 3);
```

Resultado?

Precedencia de operadores

Cuando se tengan dudas de la prioridad de ejecución de los operadores en una expresión, encerramos entre paréntesis las expresiones que se desea que se ejecuten antes:

Por ejemplo:

```
tasa = precio / (superficie + 100);
```

Precedencia de operadores

La ejecución de las sentencias:

```
int a = 3;
int b = 2;
System.out.println("La suma de: " + a + "+" + b + " es: " + a + b);
```

Tienen como resultado:

```
La suma de: 3+2 es: 32
```

¿Por qué?

Precedencia de operadores

Este "falso" resultado se debe a que el operador **"+"** situado entre **" es: "** y **a**

```
" es: " + a + b
```

Actúa como operador de concatenación de cadenas al tener a la izquierda una cadena de caracteres, por lo que obliga al valor de la derecha a convertirse en una cadena de caracteres.

De esta forma el valor de la variable **a** convertida en cadena de caracteres obliga al operador de su derecha **"+"** a ser también un operador de concatenación por lo que obliga a el valor de la variable **b** a convertirse también en cadena de caracteres. Por lo tanto no se realiza la suma.

Precedencia de operadores

Así, de esta forma las sentencias:

```
int a = 3;
int b = 2;
System.out.println("La suma de: " + a + "+" + b + " es: " + (a + b));
```

Tienen como resultado

```
La suma de: 3+2 es: 5
```

Precedencia de operadores

Sin embargo si la operación fuese el producto, tal como la sentencia:

```
System.out.println("La suma de: " + a + "+" + b + " es: " + a*b);
```

No tendría ese problema puesto que la * tiene precedencia sobre la suma.

E/S de información

Entrada y salida de datos

Salida de datos

Hemos visto que para poder realizar una salida de datos por la pantalla es necesario usar la clase System. Esta clase tiene el atributo out que hacer referencia a la salida estándar (pantalla o monitor).

Cuando se desee mostrar un mensaje por pantalla bastará con escribir la siguiente instrucción

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

Donde el mensaje a mostrar puede ser una variable o una cadena de caracteres. Si se trata de una cadena debe ir entre comillas dobles " "

E/S de información

Salida de datos

Caracteres de Escape

Los caracteres de escape son pequeñas constantes de gran utilizad para formatear las salidas de datos:

"\"" escribir comillas dobles: "hola \"mundo\" ": hola "mundo"

"\\" permite escribir una \ : "hola mundo \\": hola mundo \

"\b" borra el último carácter: " hola mundo\b": hola mund

"\r" retorno de carro: "hola \r mundo": mundo

"\t" introduce un tabulador: "hola \t mundo" : hola mundo

"\n" introduce un salto de línea: "hola \n mundo": hola
mundo

E/S de información

Formateando la salida

System.out.print () y println () no proporcionan formato de salida, como controlar el número de espacios para imprimir un int y el número de lugares decimales para un doble.

Java 1.5 introdujo un nuevo método llamado printf() para la salida formateada y toma la siguiente forma:

E/S de información

Formateando la salida

printf(formatting-string, arg1, arg2, arg3, ...);

La cadena de formato (formatting-string) contiene textos normales y los denominados especificadores de formato.

Los textos normales (incluidos los espacios en blanco) se imprimirán tal como están. Los especificadores de formato, en la forma de

% [flags] [width] conversion-code

serán sustituidos por los argumentos que siguen a la cadena de formato.

Un especificador de formato comienza con un '%' y termina con el código de conversión, por ejemplo, %d para entero, %f para número decimal , %c para carácter y %s para cadena.

E/S de información

Formateando la salida

Opcional [ancho] se puede insertar en medio para especificar el ancho de campo. De manera similar, se pueden usar [banderas] opcionales para controlar la alineación, el relleno y otros.

- %ad: entero on a espacios
- %as: cadena con a espacios.
- %a.bf: número decimal con a espacios con b digitos decimales
- %n: nueva línea

E/S de información

Formateando la salida. Ejemplos

```
System.out.printf("Hi,%s%4d%n", "Hello", 88);
```

Hi,Hello 88

```
System.out.printf("Hi, %d %4.2f%n", 8, 5.556);
```

Hi, 8 5.56

```
System.out.printf("Hi, Hi, %.4f%n", 5.56);
```

Hi, Hi, 5.5600

Ten en cuenta que printf () no avanza el cursor a la siguiente línea después de imprimir. Se debe imprimir explícitamente un carácter de nueva línea al final de la cadena de formato para avanzar el cursor a la siguiente línea, si lo deseas.

E/S de información

Entrada de datos

La lectura de datos de teclado en un programa en java no es una operación trivial. Para esto son necesarios ciertos conocimientos de programación dirigida a objetos, que permitan configurar la comunicación con el teclado.

Para simplificar esta tarea utilizaremos una de las utilidades propias de java. La clase Scanner. Esta clase se encuentra dentro del paquete util de java, por lo que será necesario importarlo en todos nuestros programas que necesiten usar el teclado:

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

E/S de información

Entrada de datos

Para poder trabajar con esta utilidad es necesario inicializarla antes de realizar cualquier lectura de la siguiente forma:

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

Donde System.in indica que estamos leyendo del teclado.

sc es el nombre que le queremos dar a la conexión que hemos abierto con el teclado, podemos darle cualquier nombre: sc, leer, teclado…

Esta operación sólo hay que realizarla una vez, independientemente de cuantas lecturas queramos hacer.

E/S de información

La clase Scanner dispone de los siguientes métodos para poder leer datos:

- nextBoolean() //Lee un booleano (True, False)
- nextByte() nextShort() nextInt() nextLong() //Leen enteros
- nextFloat() nextDouble() //Leen decimales
- nextLine() //Lee una línea de texto (una cadena que termina cuando //presionamos la tecla <intro>
- next() //Lee una cadena hasta que encuentra un espacio en blanco
- next().charAt(0) //Lee un carácter

actividad

```
import java.util.Scanner;

public class EntradaSalidaDatos {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.println("Dime un número");
        int num=sc.nextInt();
        System.out.println(num);    //muestra el número leído por pantalla
    }

}
```

Actividad

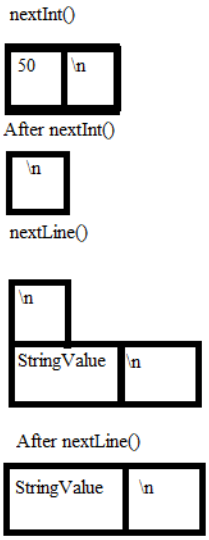
Agrega al código anterior las sentencias necesarias para que pida un nombre por teclado y luego muestre por pantalla todos los datos leídos

E/S de información

Limpiar el Buffer de entrada

Al introducir un dato numérico de cualquier tipo, se introduce dicho número más el carácter de intro ('\n'), al realizar la lectura, solamente se lee el número y el salto de línea permanece en el buffer de lectura

Esto puede ser problemático si después de leer un número, se lee una cadena de caracteres, ya que esta tomará ese ultimo salto de línea del teclado en lugar de realizar una lectura. Para evitar este problema, basta con realizar una lectura extra para eliminar dicho salto de línea.



Actividad

```java
import java.util.Scanner;

public class EntradaSalidaDatos {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.println("Dime un número");
        int num=sc.nextInt();

        sc.nextLine();

        System.out.println("Ahora dime un nombre");
        String cadena = sc.nextLine();

        System.out.println("Has introducido el número: " + num + " y el nombre " + cadena);
    }
}
```

Actividad

next() solo lee hasta donde encuentra un espacio (hasta un espacio).
nextLine() lee todo incluyendo espacios (hasta un enter).

```
import java.util.Scanner;

public class EntradaSalidaDatos {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.println("Dime un número");
        int num=sc.nextInt();

        System.out.println("Ahora dime un nombre");
        String cadena = sc.next();

        System.out.println("Has introducido el número: " + num + " y el nombre " + cadena);

    }

}
```

Conversiones de tipos de datos primitivos

Convertir un tipo de dato en otro suele ser una operación más o menos habitual en un programa y la mayoría de los lenguajes de programación facilitan algún mecanismo para llevarla a cabo. **En cualquier caso, no todas las conversiones entre los distintos tipos de dato son posibles.**

Por ejemplo, en Java no es posible convertir valores booleanos a ningún otro tipo de dato y viceversa. Además, en caso de que la conversión sea posible es importante evitar la pérdida de información en el proceso.

Conversiones de tipos de datos primitivos

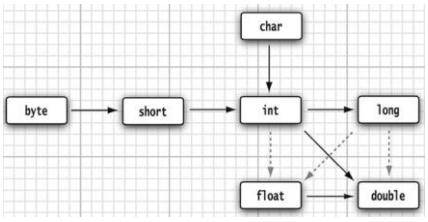
En general, existen dos categorías de conversiones:

- De ensanchamiento o promoción. Por ejemplo: pasar de un valor **entero** a un **real**.
- De estrechamiento o contracción. Por ejemplo: pasar de un valor **real** a un **entero**.

Conversiones de tipos de datos primitivos

Las conversiones de promoción transforman un dato de un tipo a otro con el mismo o mayor espacio en memoria para almacenar información.

En estos casos puede haber una cierta pérdida de precisión al convertir un valor entero a real al desechar algunos dígitos significativos. Las conversiones de promoción en Java para tipos numéricos se resumen en la siguiente diagrama.



Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Conversiones de tipos de datos primitivos

Las conversiones de contracción son más comprometidas ya que transforman un dato de un tipo a otro con menor espacio en memoria para almacenar información. En estos casos se corre el riesgo de perder o alterar sensiblemente la información. Las conversiones de contracción en Java se resumen en la siguiente tabla.

Tipo de origen	Tipo de destino
byte	char
short	byte, char
char	byte, short
int	byte, short, char
long	byte, short, char, int
float	byte, short, char, int, long
double	byte, short, char, int, long, float

Conversiones de tipos de datos primitivos

Tanto las conversiones de promoción como las de contracción se realizan por medio de los siguientes mecanismos:

- Por asignación (casting implícito): cuando una variable de un determinado tipo se asigna a una variable de otro tipo. Sólo admite conversiones de promoción.

Por ejemplo: si n es una variable de tipo int que vale 25 y x es una variable de tipo double, entonces se produce una conversión por asignación al ejecutarse la sentencia

```
int n = 25;
double x = n;
```

la variable x toma el valor 25.0 (valor en formato real). El valor de n no se modifica.

Conversiones de tipos de datos primitivos

- Por promoción aritmética: como resultado de una operación aritmética. Como en el caso anterior, sólo admite conversiones de promoción. Por ejemplo, si producto y factor1 son variables de tipo double y factor2 es de tipo int entonces la ejecutarse la sentencia

```
double producto, factor1=1.4;
int factor2=3;

producto = factor1 * factor2;
```

el valor de factor2 se convierte internamente en un valor en formato real para realizar la operación aritmética que genera un resultado de tipo double. El valor almacenado en formato entero en la variable factor2 no se modifica.

Conversiones de tipos de datos primitivos

- Con casting explícito: Es un procedimiento para transformar una variable primitiva de un tipo a otro y se realiza con operadores que producen la conversión entre tipos. Admite las conversiones de promoción y de contracción indicadas anteriormente.

Por ejemplo: si se desea convertir un valor de tipo double a un valor de tipo int se utilizará el siguiente código

```
int n;
double x = 82.4;
n = (int)x;
```

la variable n toma el valor 82 (valor en formato entero). El valor de x no se modifica.

Conversiones de tipos de datos primitivos

Como se ve, el formato general para indicar que queremos realizar la conversión es:

(tipo) valor_a_convertir

```
int num1 = 100;
short num2 = (short)num1;
```

En este ejemplo, si se sustituye la primera línea int num1=100 por int num1=1000000, el código compilaría bien, pero habría pérdida de datos, pues el 1000000 se sale del rango de short, que comprende desde -32768 a 32767. Al mostrar por consola el valor se obtendría un resultado incongruente.

Conversiones de tipos de datos primitivos

CONVERTIR DESDE	CONVERTIR A:							
	boolean	byte	short	char	int	long	float	double
boolean	-	No	No	No	No	No	No	No
byte	No	-	Si	Casting	Si	Si	Si	Si
short	No	Casting	-	Casting	Si	Si	Si	Si
char	No	Casting	Casting	-	Si	Si	Si	Si
int	No	Casting	Casting	Casting	-	Si	Si*	Si
long	No	Casting	Casting	Casting	Casting	-	Si*	Si*
float	No	Casting	Casting	Casting	Casting	Casting	-	Si
double	No	Casting	Casting	Casting	Casting	Casting	Casting	-

Conversiones de tipos de datos primitivos

En algunos lenguajes de programación existe una relación entre los caracteres y los número enteros.

El lenguaje Java utiliza el conjunto de caracteres **Unicode,** que incluye no solamente el conjunto ASCII sino también caracteres específicos de la mayoría de los alfabetos.

Para simplificar el siguiente ejemplo, podemos tomar la tabla de codificación de caracteres ASCII.

Conversiones de tipos de datos primitivos

En dicha tabla existe una correspondencia entre el nº de orden del caracter y la representación literal del caracter.

Por lo tanto hay una serie de conversiones implícitas y explícitas que se permiten entre los tipos **int y char:**

```
char letraA=65;     // valor ASACII 'A'
System.out.println(letraA);

int valorA=letraA;
System.out.println(valorA);

int valorB='B';     //entero ASCII 66
System.out.println(valorB);

//casting explícito
char letraB = (char)valorB;     //convierto de int a char
System.out.println(letraB);
```

```
A
65
66
B
```

Actividad

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

        int a = 2;
        double b = 3.0;
        float c = (float) (20000*a/b + 5);

        System.out.println("Valor en formato float: " + c);
        System.out.println("Valor en formato double: " + (double) c);
        System.out.println("Valor en formato byte: " + (byte) c);
        System.out.println("Valor en formato short: " + (short) c);
        System.out.println("Valor en formato int: " + (int) c);
        System.out.println("Valor en formato long: " + (long) c);
    }
}
```

Documentación interna

Documentación interna de un programa

Un programa debe ser fácilmente legible, el nombre de las variables y métodos deben indicar cual es el papel que desempeña cada uno en el programa; esto de debe complementar con oportunos comentarios.

Deben evitarse los comentarios innecesarios que sean evidentes

```
System.out.println("Ventas= " + ventas); //Imprime el valor de ventas
```

Documentación interna

Documentación interna de un programa

Una documentación interna correcta a través de **oportunos comentarios** y de **adecuados nombres para variables y métodos** nos permite revisar un programa luego de 6 meses, o más, casi con la misma rapidez que cuando se implementó.

Sin embargo un programa confuso es difícil de entender y puede resultar a veces más fácil hacerlo de nuevo que modificarlo

En la escritura de un programa, debe procurarse que **cada línea tenga una sola sentencia**, si bien algunas líneas pueden tener más de una sentencia, sobre todo si ambas sentencias son complementarias en una acción concreta.

Documentación interna

Las líneas dentro de un bloque deben sangrarse. Esto nos permite observar fácilmente el inicio y el final de un bloque.

Entre dos bloques de sentencias, que hacen tareas diferenciadas, se debe dejar una o varias líneas en blanco

Ejercicios

Ejercicios disponibles en el aula virtual

Recursos

- Curso youtube : Java desde 0
- Libro Java 9. Manual imprescindible. F. Javier Moldes Teo. Editorial Anaya
- App SoloLearn: Aprende a Programar. Curso Java