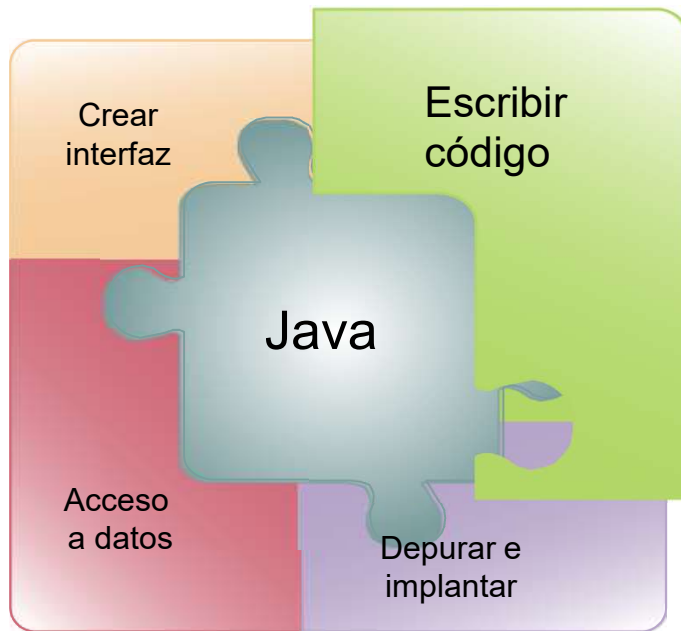


Unidad 2: Elementos del lenguaje I: datos

Descripción



1. Estructura básica de un programa
2. Variables y tipos de datos
3. Literales
4. Convertir tipos de datos
5. Operadores
6. Entrada /Salida
7. Tipos enumerados
8. Ejercicios
9. Para ampliar...

estructura básica
comentarios y separadores



ESTRUCTURA BÁSICA DE UN PROGRAMA

Estructura básica de un programa

```
/*
 * Estructura de una clase de Java
 * Si no es una clase principal el método main no aparece
 */

public class NombreDeLaClase {
    //Declaración de los atributos de la clase

    //Declaración de los métodos de la clase

    //El método main que indica dónde empieza la ejecución
    public static void main (String[] args) {
        //Declaración de las variables del método

        //Sentencias de ejecución del método
    }
}
//Si no es una clase principal el método main no aparece
```

Comentarios y separadores

- **Comentarios:**

- `/* ... diversas líneas ... */`
- `//` Una sola línea

- **Separadores:**

- `(...)` Listas de parámetros en la definición y llamada a un método
- `{...}` Engloba bloques de código y en valores iniciales de arrays
- `[...]` En la declaración de arrays y en referencias a elementos de éstos
- `;` Separador de instrucciones
- `,` Separador de identificadores y argumentos
- `.` Separador de elementos de un objeto

LOS DATOS



Introducción

Dentro del programa

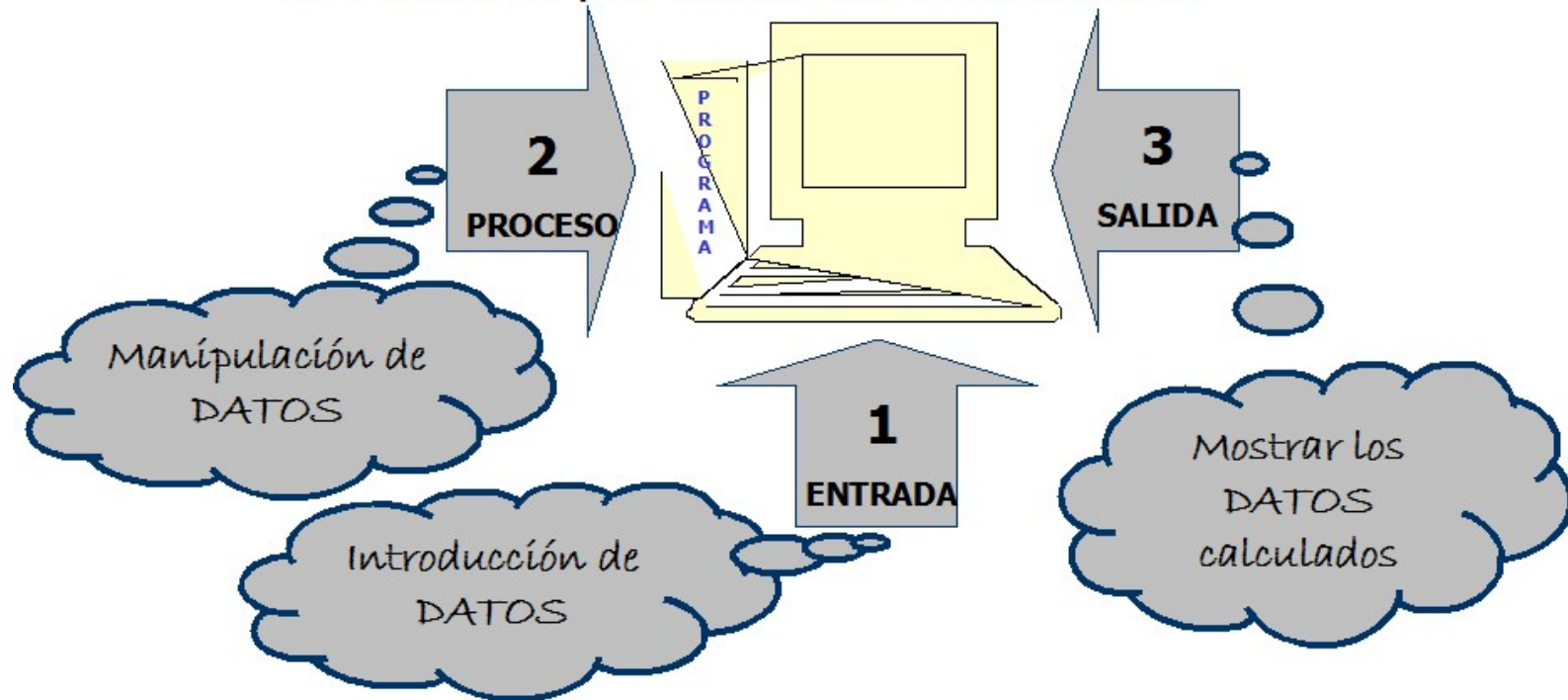
Datos y estructuras de datos

Clasificación según su permanencia

Características de los datos

Introducción

INFORMÁTICA: Tratamiento automático de la información por medio de ordenadores



DATOS

- Tipos de datos empleados
- Operaciones permitidas sobre los datos

Instrucciones

- La forma en que se especifica el orden en que se ejecutan las operaciones

- **Tipos de datos**
- **Operadores**



- **Estructuras de control**

LENGUAJE DE PROGRAMACIÓN

Dato- Estructura de datos

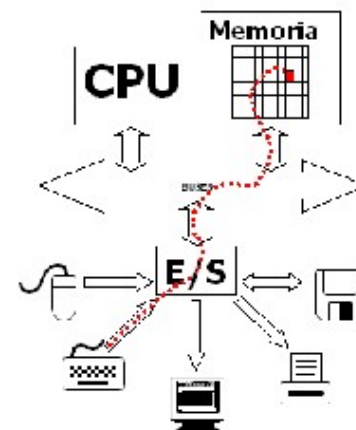
DATO

- Es toda aquella información que caracteriza a una entidad y que es susceptible de tratamiento en un programa informático.

Para que puedan ser utilizados por un programa el primer paso será reservar un espacio en la memoria del ordenador para almacenarlos durante la ejecución del mismo.

ESTRUCTURA DE DATOS

- Espacio reservado en memoria para almacenar un dato durante la ejecución de un programa.



Clasificación según su permanencia

- **Constantes:** Su información es fija durante la ejecución del programa. Se pueden expresar:
 - Mediante su **valor**. Ej.: 1325, 'a', 12.6
 - Mediante un **identificador** que habremos definido antes. Ej: $\text{PI}=3.14159$, en vez de utilizar el valor utilizamos el nombre de la constante.
- **Variables:** su información puede ser variar durante la ejecución del programa. Se definen mediante un identificador y un tipo de dato.



Características de los datos

Identificador: Nombre que sirve para referenciarlo. Existen ciertas normas generales para su empleo

- El nombre debe tener relación con la información que contiene.

Tipo: Rango de valores en función de una clasificación y que determina el espacio de memoria que hay que reservar para el dato.

Valor: Elemento perteneciente al rango de valores según su tipo y que estará contenido en el espacio de memoria reservado.

DATOS.

° CARACTERÍSTICAS

- variables
- constantes
- Identificadores en JAVA
- tipos de datos
- tipos de datos primitivos
- declaración de variables
- inicialización de variables
- asignación
- ámbito

Variables y Tipos de datos

- Una variable permite almacenar los datos, resultados y resultados intermedios de un problema en un programa
- Tiene asociado un tipo de datos
- El tipo de datos al que pertenece una variable:
 - define el conjunto de valores que son susceptibles de ser almacenados en dicha variable
 - y las operaciones que se pueden realizar con ella
- Declaración de variables en Java:

tipo identificador ;

Constantes

- Para declarar una constante usamos el modificador *final*

final tipo identificador = valor ;

Ej.:

final double PI = 3.1415926536;

- El valor de una constante no se puede modificar durante el programa
- Debemos darle un valor a la vez que se declara

Identificadores en JAVA

- Pueden referenciar variables, constantes, nombres de métodos o de clases.
- Distinguen mayúsculas y minúsculas.
edadMinima es distinto de edadMINIMA.
- Debe comenzar por una letra, _ o \$
- Debe ser una sucesión de letras (mayúsculas, minúsculas), dígitos ('0' a '9'), y los caracteres especiales '_' y '\$'
- No debe coincidir con ninguna palabra reservada del lenguaje JAVA

Recomendaciones

- Utilizar **nombres significativos** de la información que contienen
- Los nombres de **variables y métodos empiezan con minúscula**. Si es compuesto, las siguientes palabras empezarán por mayúscula, evitando uso de '_'. Ej. numeroComensales
- Nombres de **clases, comienzan con mayúsculas**, si son compuestos las siguientes palabras empiezan por mayúscula. Por ejemplo: MiClase
- Las **constantes** se escriben **en mayúsculas**. Por ejemplo: PI

Palabras reservadas en JAVA

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[s]
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
cast	future	generic	inner	
operator	outer	rest	var	



Tipos de datos

- En Java, los tipos de datos pueden clasificarse en dos grupos:
 - Tipo de datos primitivos
 - Tipo de datos referencia:
 - Strings, Arrays, Clases e Interfaces
 - Los estudiaremos más adelante...

Tipos primitivos

	NOMBRE	TIPO	OCUPA	RANGO APROXIMADO
TIPOS PRIMITIVOS (sin métodos; no son objetos; no necesitan una invocación para ser creados)	byte	Entero	1 byte	-128 a 127
	short	Entero	2 bytes	-32768 a 32767
	int	Entero	4 bytes	$2 \cdot 10^9$
	long	Entero	8 bytes	Muy grande
	float	Decimal simple	4 bytes	Muy grande
	double	Decimal doble	8 bytes	Muy grande
	char	Carácter simple	2 bytes	---
	boolean	Valor true o false	1 byte	---

Tipos de datos primitivos

- Tipos Numéricos

NOMBRE	TAMAÑO EN BITS	VALOR MÁXIMO
byte	8	127
short	16	32767
int	32	2147483647
long	64	9223372036854775807
float	32	3.4E38
double	64	1.7E+308

Podemos utilizar el tipo byte, short, long cuando valga la pena el ahorro de memoria, ej. En arrays muy grandes

int y double suelen ser los valores por defecto para los literales enteros y reales respectivamente

Tipos de datos primitivos

- Tipo carácter

Nombre	Tamaño	Representa
char	2 Bytes	Representa caracteres como letras, números y caracteres especiales Java utiliza la codificación Unicode de 2 bytes

- Tipo booleano o lógico

Nombre	Tamaño	Representa
boolean	1 Byte	Puede tomar los valores true/false

Declaración de variables

- Para declarar una variable:

```
tipo identificador;    //método preferible, por claridad
tipo identificador = valor;
tipo ident1, ident2, ident3, etc...;
tipo ident1=valor1, ident2=valor2, etc...;
```

- **Por ejemplo:**

```
int edadPedro=34;
```

```
float precioPatata=1.2F, precioChoco=2.3F;
```

```
char car='A', car2='\u0041';
```

- // 0041 es el código Unicode en hexadecimal de la A mayúscula



Inicialización de variables

- Si una variable no ha sido inicializada tiene un valor asignado por defecto:
 - 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
 - Para las variables de tipo referencial (objetos), el valor null

Asignación

- La instrucción de asignación permite asignar valores a las variables o modificar los que ya tienen
- Su sintaxis es:
identificador = expresión;
- Se puede utilizar en el bloque de declaración de variables para definir valores iniciales:
 - `int suma=64;`
 - `char ch1, ch2='u';`
 - `float f1=2.0F, f2;`
 - `.....`
 - `f2=34.67F;`
 - `suma=suma+2;`

Ámbito de una variable

- Se llama ámbito de una variable a la parte del programa en la que es conocida y se puede utilizar
- 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 { ... }
 - Sólo serán “visibles” dentro de dicho bloque
 - Las variables definidas en un bloque deben tener nombres diferentes



LITERALES

Literales I

- Un literal es un valor que se expresa a sí mismo
- Literal entero puede expresarse :
 - en decimal (base 10)
 - Ejemplo: 21
 - octal (base 8)
 - Ejemplo: 025
 - hexadecimal (base 16)
 - Ejemplo: 0x3A
 - Por defecto el literal es int. Puede añadirse al final del mismo la letra L ó l para indicar que el entero es considerado como long
- Literal real pueden expresarse:
 - parte entera, el punto decimal (.) y la parte fraccionaria (Ej: 345.678 - 0.00056)
 - notación exponencial o científica (Ej: 3.45678e2 - 5.6e-4)
 - Se puede poner una letra como sufijo:
 - F ó f Trata el literal como de tipo float
 - D ó d Trata el literal como de tipo double
 - Por defecto el literal es double. Si deseamos que se interprete como float debemos añadir el sufijo F

Literales II

- Literal carácter

Puede escribirse como:

- **Un carácter entre comillas simples como 'a', 'ñ', 'Z', 'p', etc.**
- **El código Unicode del carácter:**
 - 97 es el código Unicode del carácter a
 - 65 es el código Unicode del carácter A
- **Entre comillas, anteponiendo la secuencia de escape '\'** si el código Unicode lo expresamos en octal
 - '\141' código Unicode en **octal** equivalente a 'a'
 - '\101' código Unicode en **octal** equivalente a 'A'
- **Entre comillas, anteponiendo la secuencia de escape '\u'** si código Unicode lo expresamos en hexadecimal
 - '\u0061' código Unicode en **hexadecimal** equivalente a 'a'
 - '\u0041' código Unicode en **hexadecimal** equivalente a 'A'

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

<u>Secuencia</u>	<u>Significado</u>
\'	Comilla simple
\"	Comillas dobles
\\	Contrabarra
\b	Backspace
\n	Cambio de línea
\r	Retorno de carro
\t	Tabulador
\f	Salto de página

Literales III

- Literal booleano:
 - palabras reservadas **true** y **false**
 - Ejemplo: boolean activado = false;
- Literal Strings o cadena de caracteres
 - No forman parte de los tipos de datos elementales en Java
 - Encerrado entre comillas dobles (“ ”)
 - Ejemplo:
 - `System.out.println("Primera línea\nSegunda línea del string");`
 - `System.out.println("Ho\u0061");`
 - `System.out.println("Escribe \n y no saltes de línea");`

Tipo String. Uso sencillo

- El tipo String permite representar secuencias de caracteres

- Ejemplo:

```
String frase, palabra, linea;
```

```
frase="En un lugar de la Mancha de cuyo nom...";
```

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

```
//.....solicitamos al usuario un texto
```

```
//..... y el texto de entrada es: Oh! es terrible
```

```
palabra=lector.next( ); //palabra = Oh!
```

```
linea=lector.nextLine( ); // linea = Oh! es terrible
```



4.- CONVERTIR TIPOS DE DATOS

Conversión de tipos

- Cuando se realiza una instrucción de asignación:
Identificador = expresión;
tanto la variable como la expresión deben de ser del mismo tipo o de tipos compatibles
- Una expresión puede asignarse a una variable siempre que sea de un tipo de tamaño menor que el tipo de la variable. Por lo tanto podemos asignar en este orden:

short ➡ int ➡ long ➡ float ➡ double
- Otras formas de conversión de tipos se pueden realizar explícitamente a través de lo que se llama casting
(tipo) expresión Ejemplo:
num= (int) 34.56 //trunca
- También utilizando funciones adecuadas de ciertos paquetes

5.- OPERADORES

operadores aritméticos

operadores unarios

operadores de comparación

operadores lógicos

combinar operadores lógicos y de comparación

operadores de asignación

operadores de concatenación

operadores de bit (...ya lo veremos)

prioridad de operadores

Operadores aritméticos I

- Pueden realizar operaciones aritméticas que implican el cálculo de valores numéricos representados por literales, variables, otras expresiones, llamadas de funciones y propiedades, y constantes

- **Sintaxis:**

`expresion1 operador_aritmético expresion2`

- **Ejemplo:**

```
int x;  
x = 52 * 17;  
x = 120 / 4;  
x = 67 + 34;  
x = 32 - 12;  
X = 7 % 2;
```




Operadores aritméticos II

- + Suma
- Resta
- * Multiplicación
- / División
- % Resto de la división entera

Operadores Unarios

- Signo
 - Poner un signo + o un signo - delante de una expresión
 - Ejemplo:
 - +45
 - -32
- Incremento (++) y Decremento (--)
 - Aumentar y disminuir en 1 el valor de la variable
 - Pueden ir delante(pre) o detrás(post) de la variable
 - Ejemplo:
 - `int valor, i=5;`
 - `i++;` // ahora i vale 6. Es equivalente a `i=i+1;`
 - `--i;` // ahora i vale 5. Es equivalente a `i=i-1;`
 - La diferencia entre pre y post aparece en una instrucción compuesta:
 - `valor=i++;` //ahora valor vale 5 y i vale 6
 - // Es equivalente a `{ valor=i; i=i+1; }`
 - `valor=++i;` // ahora valor vale 7 y i vale 7
 - // Es equivalente a `{ i=i+1; valor=i; }`

Operadores Unarios. Ejemplos

Ejemplos:

- Supongamos que
 - $a=3$ y $b=7$
- Qué visualizaría `System.out.println(3 + b++);` ?
 - La salida por pantalla sería 10
- Qué visualizaría `System.out.println(3 + ++b);` ?
 - La salida por pantalla sería 11

Operadores Unarios. Ejemplos

- En realidad son dos expresiones anidadas:
- $3 + b++$

$3 + b++$

Equivale a:

$3+b$

$b++$

$3+ ++b$

Equivale a

$b++$

$3+b$

Operadores de comparación I

- Símbolos que evalúan expresiones condicionales y devuelven un valor boolean

- **Sintaxis:**

expresion1 *operador_de_comparación* expresion2

Operador
< (Menor que)
<= (Menor o igual que)
> (Mayor que)
>= (Mayor o igual que)
= (Igual a)
!= (Distinto de)

Operadores de comparación II

Operador	True si	False si
< (Menor que)	expresion1 < expresion2	expresion1 >= expresion2
<= (Menor o igual que)	expresion1 <= expresion2	expresion1 > expresion2
> (Mayor que)	expresion1 > expresion2	expresion1 <= expresion2
>= (Mayor o igual que)	expresion1 >= expresion2	expresion1 < expresion2
= (Igual a)	expresion1 == expresion2	expresion1 != expresion2
!= (Distinto de)	expresion1 != expresion2	expresion1 == expresion2

Operadores de comparación III

Ejemplo:

```
int cantidad=3000;  
boolean pedidoGrande;  
pedidoGrande = cantidad > 1000 T
```

```
boolean testResult ;  
testResult = ( 45 < 35 ); F  
testResult = ( 45 == 45 ); T  
testResult = ( 4 != 3 ); T  
testResult = ( 'a' > 'b' ); F
```

Operadores lógicos I

- Los operadores lógicos realizan una evaluación lógica de expresiones y devuelven un valor boolean

- **Sintaxis:**

expresion1	<i>operador_lógico</i>	expresion2
------------	------------------------	------------

Operador

&&

--

!

- **Ejemplo:**

edad>18 && sexo=='H'

edad>18 sexo=='H'

Operadores lógicos II

Operador	Función
&&	Combina dos expresiones. Cada expresión debe ser True para que toda la expresión sea True
	Combina dos expresiones. Si una expresión es True, toda la expresión es True.
!	Proporciona el negativo lógico de la entrada

Operadores lógicos III .Tablas de verdad

Operador lógico **And**

<u>Expres1</u>	<u>Expres2</u>	<u>Resultado</u>
True	True	True
True	False	False
False	True	False
False	False	False

Operador lógico **Not**

<u>Expresión1</u>	<u>Resultado</u>
True	False
False	True

Operador lógico **Or**

<u>Expres1</u>	<u>Expres2</u>	<u>Resultado</u>
True	True	True
True	False	True
False	True	True
False	False	False

Combinar operadores lógicos y de comparación

- Podemos combinar operadores de comparación y operadores lógicos con instrucciones condicionales
- Ejemplo:

**Operador de
comparación**

**Operador
lógico**

```
edad <= 25 && edad >=14
```

Operadores de asignación

Operador	ejemplo
=	edad = 34
+=	edad += 1 edad = edad + 1
- =	edad - = 3 edad = edad - 3
* =	edad *= 2 edad = edad * 2
/=	edad /= 2 edad = edad / 2
% =	edad %= 2 edad = edad % 2

Podemos combinar el operador de ASIGNACIÓN con los operadores aritméticos

Operador de concatenación

- Permite generar una cadena de caracteres a partir de otras dos

`expresion1 + expresión2`

Ejemplo: **“Hola” + “, “ + ”buenos días”**

Ejemplo: **“Debe pagar :” + total + ” euros”**

Prioridad de operadores I

- Cuando aparecen varias operaciones en una expresión se evalúa y se resuelve en un orden predeterminado
- Orden por niveles:
 1. Paréntesis, de dentro a fuera
 2. Unarios (prefijos se evalúan antes) (postfijos se evalúan al final)
 3. Aritméticos
 - A. Multiplicativos: * / %
 - B. Sumativos: + -
 4. Comparativos o Relacionales
 5. Lógicos o booleanos
 - A. Not !
 - B. And &&
 - C. Or ||
 6. Asignación
 7. Unarios postfijos
- Cuando aparecen operadores de la misma prioridad juntos en una expresión el compilador evalúa cada operación de izquierda a derecha

Prioridad de operadores II

- Ejemplo

$a = -3 + 5 + 2 * 4 - 6 / 4 * 3 - 5 \% 2;$

ORDEN: 1 6 7 2 8 3 4 9 5

Valor final a igual a 6

- Ejemplo

`System.out.println ("Cuidado: " + 4*5 + 6);`



6.-TIPOS ENUMERADOS

- ```
public class Semana {
 public enum DiaSemana{LUNES, MARTES, MIERCOLES, JUEVES,
 VIERNES, SABADO, DOMINGO}

 public static void main (String[] args){

 DiaSemana hoy = DiaSemana.JUEVES;
 DiaSemana ultimo=DiaSemana.DOMINGO;

 System.out.println("Hoy es " +hoy +"\n Y el ultimo dia es
 "+ultimo);
 }
}
```

Salida de datos por pantalla  
Entrada de datos del teclado



## **7.- ENTRADA / SALIDA**

# Salida de datos por pantalla

- Utilizaremos los métodos **print( )** o **println( )**
  - **println( )** incluye el retorno de carro al final de la salida

```
System.out.print("Se imprime este mensaje sin el
retorno de carro");
```

```
System.out.println("Se imprime este mensaje con un
retorno de carro");
```

# Entrada de datos del teclado I

- El método `read( )` lee un solo carácter  
`char c = (char) System.in.read();`
  - Esta manera de leer del teclado es muy poco práctica y sería tedioso programar la lectura de un número, de una cadena de caracteres...
- Declarar un objeto de la clase `Scanner` y usar sus métodos
  - Entenderemos los detalles en próximos temas
  - En el ejemplo siguiente vemos como hacerlo

# Entrada de datos del teclado II

```
//1.-Importamos el fichero donde están las clases
import java.util.Scanner;

public class Exemple {
 public static void main (String[] args) {
 int primerNum;
 //2.-Se declara el objeto lector de la clase Scanner
 Scanner lector = new Scanner(System.in);
 ...
 System.out.print("Escribe un número y pulsa retorn: ");
 //se lee un valor entero
 primerNum = lector.nextInt();
 System.out.print("El numero introducido es: "+primerNum);
 ...
 }
}
```

# Entrada de datos del teclado III

| <u>Método</u>                     | <u>Tipo de dato leído</u> |
|-----------------------------------|---------------------------|
| <code>lector.nextByte()</code>    | <code>byte</code>         |
| <code>lector.nextShort()</code>   | <code>short</code>        |
| <code>lector.nextInt()</code>     | <code>int</code>          |
| <code>lector.nextLong()</code>    | <code>long</code>         |
| <code>lector.nextFloat()</code>   | <code>float</code>        |
| <code>lector.nextDouble()</code>  | <code>double</code>       |
| <code>lector.nextBoolean()</code> | <code>boolean</code>      |
| <code>lector.next()</code>        | <code>String</code>       |
| <code>Lector.nextLine()</code>    | <code>String</code>       |



- 
- Buscar la diferencia entre `next()` y `nextline()`

# Ejemplo I

```
import java.util.Scanner;

/*
 *ejemplo Scanner
 */
public class EjemploScanner {

 public static void main(String[] args) {
 int entero;
 double real;
 boolean siOno;
 char character;
 String texto; //no es un tipo primitivo

 Scanner lector = new Scanner(System.in);

 System.out.print("Introduce un entero: ");
 entero = lector.nextInt();
 System.out.println("El entero introducido es " + entero);

 System.out.print("Introduce un real: ");
 real = lector.nextDouble();
 //ATENCIÓN! el programa fallara si el usuario entra el numero con punto decimal
 System.out.println("El real introducido es " + real);

 System.out.print("Introduce si o no: ");
 siOno = lector.nextBoolean();
 //ATENCIÓN! el programa fallara si el usuario no entra true o false
 System.out.println("El boolean introducido es " + siOno);
 }
}
```

# Ejemplo I

```
System.out.print("Introduce si o no: ");
siOno = lector.nextBoolean();
//ATENCIÓN! el programa fallara si el usuario no entra true o false
System.out.println("El boolean introducido es " + siOno);

lector.nextLine();
//ATENCIÓN! con esta instrucción limpio el buffer de entrada
//si no la pongo, no me deja entrar el carácter posterior
//encuentra el salto de línea y considera que esa es la tira introducida
//pruébalo!

System.out.print("Introduce un caracter: ");
caracter = lector.nextLine().charAt(0);
//ATENCIÓN! leo un String y me quedo con el primer carácter
System.out.println("El caracter introducido es " + caracter);

//lector.nextLine();
//Hace falta aquí limpiar el buffer de entrada?

System.out.print("Introduce tu nombre: ");
texto = lector.nextLine();
System.out.println("Tu nombre es " + texto);

}
```

# Ejemplo II

```
import java.util.*;
Public class Adivinanza {
 public static void main (String[] args) {
 • Scanner tcl = new Scanner(System.in);
 • int valor;
 • System.out.println("Piensa un numero"); tcl.nextLine();
 • System.out.println("Multiplicalo por 5");tcl.nextLine();
 • System.out.println("Sumale 6"); tcl.nextLine();
 • System.out.println("Multiplicalo por 4");tcl.nextLine();
 • System.out.println("Sumale 9"); tcl.nextLine();
 • System.out.println("Multiplicalo por 5");tcl.nextLine();
 • System.out.println("Escribe el resultado");
 • valor=tcl.nextInt();
 • System.out.println("El numero que has pensado es: ");
 • System.out.println((valor-165)/100);
 }
}
```



## **8.- EJERCICIOS**



# Ejercicio:

- Escribe un programa que permita introducir dos números y calcular la suma de los mismos
- Provoca errores en el programa:
  - Errores de compilación:
    - No declarar variables
    - Asignar tipos
    - ....
  - Errores de ejecución:
    - Introducir un string
    - Introducir un entero demasiado grande
    - ...





## 9.- PARA AMPLIAR...

Conocimientos puntuales que se ampliarán más adelante

- Clase STRING
- Clases envolventes de los tipos primitivos
- Entrada y Salida de datos:
  - Entrada de 1 carácter
  - Entrada de reales
  - Salida formateada



# Clase String

- El tipo cadena “**cadena de caracteres**” es una clase especial que permite instanciar objetos sin un constructor específico.
- Como objeto que es tiene métodos que nos facilitan su manejo y en ocasiones su tratamiento difiere del que podemos dar a un dato de tipo primitivo (como la comparación de cadenas)

## Algunos métodos de la clase String

- **charAt(**n**)**

Nos permite obtener el carácter que ocupa la posición **n** de una cadena (empezando por la izquierda la primera posición es la 0)

Ej.: String cad;

cad= teclado.next();

car=cad.charAt(0); //primer carácter de cad

## Algunos métodos de la clase String

- **equals(cadena)**

Compara la cadena con el parámetro que se le pasa

```
Ej.: String cad="Hola";
 if(cad.equals("Hola"))
 System.out.print("Son iguales");
```

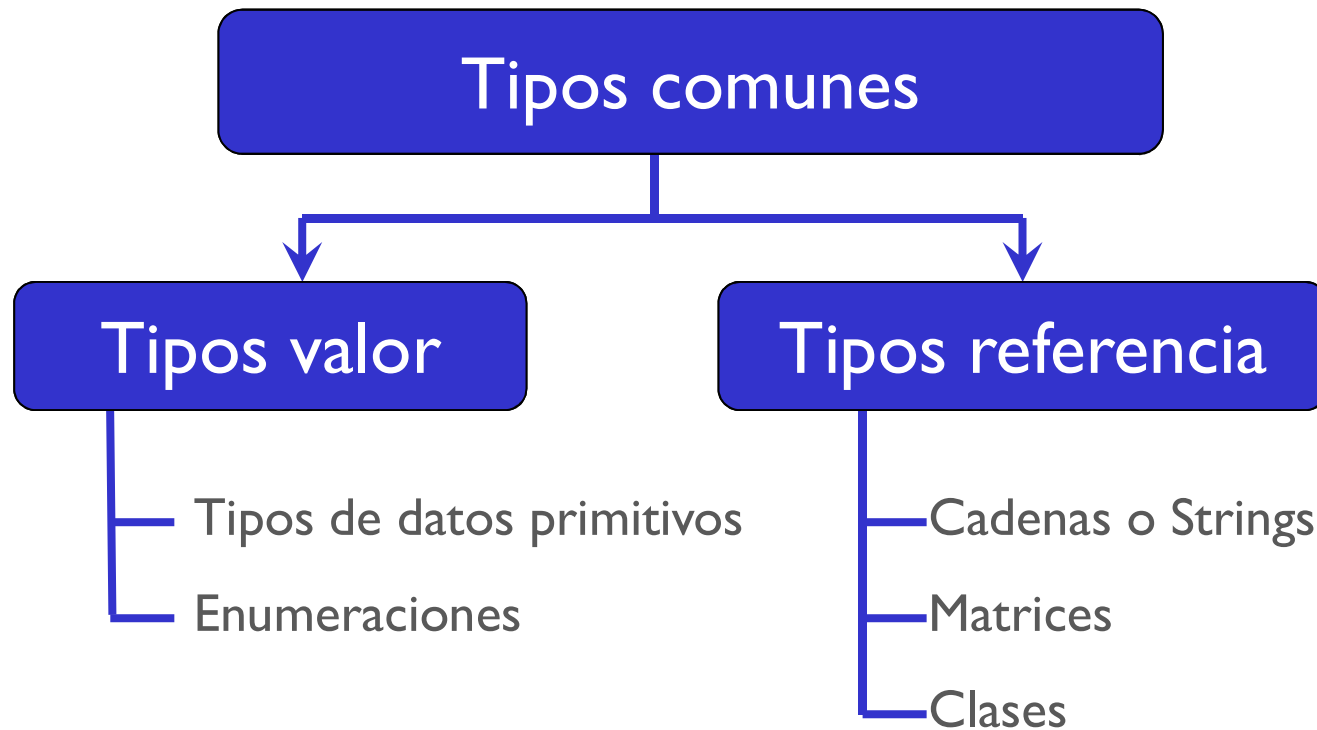
- **equalsIgnoreCase(cadena)**

No distingue entre mayúsculas y minúsculas

# equals

- ¿por qué no se utiliza == para comparar cadenas?
- Porque == compara OBJETOS y lo que nos dice es si 2 objetos son iguales, no si su “contenido” es igual.
- En ocasiones funciona porque depende de cómo se hayan instanciado los objetos cadena, el compilador decide que ya existe una instancia al declarar el segundo y simplemente la segunda apunta a la primera y se trata del mismo objeto (mismo espacio de memoria)

# Sistema de tipos



Una variable **de tipo valor** contiene directamente sus datos

Una variable de **tipo referencia** contiene una referencia o puntero al valor de un objeto



# Clases envolventes

Cada tipo primitivo en java tiene su correspondiente clase envolvente cuyos objetos pueden almacenar un valor del tipo primitivo correspondiente.

## Tipos de datos simples

**byte**

**short**

**int**

**long**

**float**

**double**

**char**

**boolean**

## Clase equivalente

**java.lang.Byte**

**java.lang.Short**

**java.lang.Integer**

**java.lang.Long**

**java.lang.Float**

**java.lang.Double**


**java.lang.Character**

**java.lang.Boolean**



## Clases envolventes de los tipos primitivos

- Estos objetos se suelen utilizar para utilizar sus métodos asociados que permiten la conversión de unos tipos primitivos a otros o desde una cadena de caracteres a un número y viceversa
- Igualmente se utilizan en las colecciones de datos que sólo pueden contener objetos.

- 
- Las clases envolventes son Integer, Boolean, Character, Long, Float y Double.
  - Tienen el mismo nombre pero con la primera letra en mayúsculas.
  - Ej de método:

`Float.parseFloat(cad)`

`//Convierte un String, cad, en un float`

# ENTRADA / SALIDA

- Lectura de 1 carácter:
  - Leemos una palabra y nos quedamos con el primer carácter (el que ocupa la posición 0)
    - `teclado.next().charAt(0);`
- Lectura de un real (float o double)
  - Al leer un real en lugar de aceptar el punto decima espera una coma, o da error.
  - Para evitarlo leemos una cadena y la convertimos en el tipo que nos interese
    - Ej.: `Double.parseDouble(teclado.next())`

# Ejemplo método parseDouble

```
package Tipos; import java.util.Scanner;
/** * * @author maria */
public class ClasesEnvolventes1 {
 public static void main(String[] args) {
 Scanner tec=new Scanner(System.in);
 double realSimple;
 System.out.println("Teclea un real utilizando la , como
separador decimal");
 realSimple=tec.nextDouble();
 System.out.println("Real leído con .: "+realSimple);
 System.out.println("Teclea un real utilizando el . como
separador decimal");
 realSimple=Double.parseDouble(tec.next());
 System.out.println("\nReal leído con ,: "+realSimple);
 }
}
```

# Ejemplo de charAt()

```
import java.util.Scanner;
/** * * @author maria */
public class LeerUnCaracter {
 public static void main(String[] args) {
 Scanner tec=new Scanner(System.in);
 char car;
 String cad;
 System.out.println("Teclea un caracter");
 car=tec.next().charAt(0);
 tec.nextLine();
 System.out.println("Caracter: "+car);
 System.out.println("Teclea una cadena de
caracteres: ");
 cad=tec.nextLine();
 System.out.println("Cadena:"+cad);
 }
}
```



# Salida

La sintaxis para los especificadores de formato de printf es:

`%[posición_dato$][indicador_de_formato][ancho][.precision]carácter_de_conversión`

| INDICADORES DE FORMATO |                                                    |           |                                      |
|------------------------|----------------------------------------------------|-----------|--------------------------------------|
| Indicador              | Significado                                        | Indicador | Significado                          |
| -                      | Alineación a la izquierda                          | +         | Mostrar signo + en números positivos |
| (                      | Los números negativos se muestran entre paréntesis | 0         | Rellenar con ceros                   |
| ,                      | Muestra el separador decimal                       |           |                                      |

| CARACTERES DE CONVERSIÓN |                                                                                               |          |                                    |
|--------------------------|-----------------------------------------------------------------------------------------------|----------|------------------------------------|
| Carácter                 | Tipo                                                                                          | Carácter | Tipo                               |
| d                        | Número entero en base decimal                                                                 | X, x     | Número entero en base hexadecimal  |
| f                        | Número real con punto fijo                                                                    | s        | String                             |
| E, e                     | Número real notación científica                                                               | S        | String en mayúsculas               |
| g                        | Número real. Se representará con notación científica si el número es muy grande o muy pequeño | C, c     | Carácter Unicode. C: en mayúsculas |

# printf



`posicion$` indica el 'ordinal' del parámetro que viene a continuación.

Si no aparece, por defecto los formatos se corresponden con los parámetros en el mismo orden de aparición

# Algunas funciones predefinidas. La clase Math

- Las constantes E y PI  
Math.E=2.7182818284590452354  
Math.PI=3.14159265358979323846
- Las funciones de redondeo, con x de tipo double:
  - `ceil(x)` : devuelve el número entero más pequeño que es mayor o igual a x
  - `floor(x)` : devuelve el número entero más grande que es menor o igual a x
  - `round(x)` : convierte el real x al entero más próximo
- Funciones trigonométricas
  - `sin(x)` : calcula el seno del ángulo (en radianes) x
  - `cos(x)` : calcula el coseno del ángulo (en radianes) x
  - `asin(x)` : calcula el arco seno de x (x entre -1 y 1)
  - `acos(x)` : calcula el arco coseno de x (x entre -1 y 1)
  - `atan(x)` : calcula el arco tangente de x

# Algunas funciones predefinidas. La clase Math

- Otras funciones
  - `abs(x)` : calcula el valor absoluto de `x` (entero o real)
  - `exp(x)` : calcula `e` elevado a `x` (`x` es real)
  - `log(x)` : calcula el logaritmo natural de `x` (`x` real y no negativo)
  - `max(x,y)` : compara los números `x` e `y` (enteros o reales) y devuelve el mayor
  - `min(x,y)` : compara los números `x` e `y` (enteros o reales) y devuelve el menor
  - `pow(x,y)` : calcula `x` elevado a `y`. No está definida si `x` es negativo o 0 e `y` no es entero, ni tampoco si `x=0` e `y` es negativo o 0
  - `random( )` : genera un número pseudo-aleatorio entre 0.0 y 1.0
  - `sqrt(x)` : calcula la raíz cuadrada de `x` ( `x` no negativo)