Unidad 2. El lenguaje Java

Sintaxis básica del lenguaje Java

José L. Berenguel

Tabla de Contenidos

1. Estructura y elementos básicos.
2. Tipos de datos primitivos.
3. Variables e identificadores.
4. Literales y valores por defecto.
5. Conversiones de tipo (*casting*).
6. Operadores.
7. Construcciones de tipo objeto.
8. Cadenas de caracteres (*String*).
9. Leer datos de teclado (*Scanner*).
10. Paquetes y librerías del API de Java.

Hola Mundo en Java

```
public class Saludo{
     public static void main(String[] args){
          System.out.println(“Bienvenido a Java”);
     }
}
```

- El nombre del fichero igual que el nombre de la clase con la extensión *.java*: **Saludo.java**
- Compilador (***javac***) genera el fichero ***bytecode*** con extensión .class
  - *javac Saludo.java*
- El programa en *bytecode* se ejecuta en la máquina virtual (*Java Virtual Machine*, JVM)
  - *java Saludo*

Aspectos básicos de la sintaxis

```
/** Esto es un comentario javadoc
  * @autor José Luis Berenguel
  */
public class Saludo{
     /* Comentario multilínea
      * La función main es el punto de entrada para el programa en la JVM */
     public static void main(String[] args){
          System.out.println("Bienvenido a Java"); //Mensaje de bienvenida
          System.out.println("Va a ser una experiencia única \n\t¡Comencemos!");
     }//fin_main
}//fin_class
```

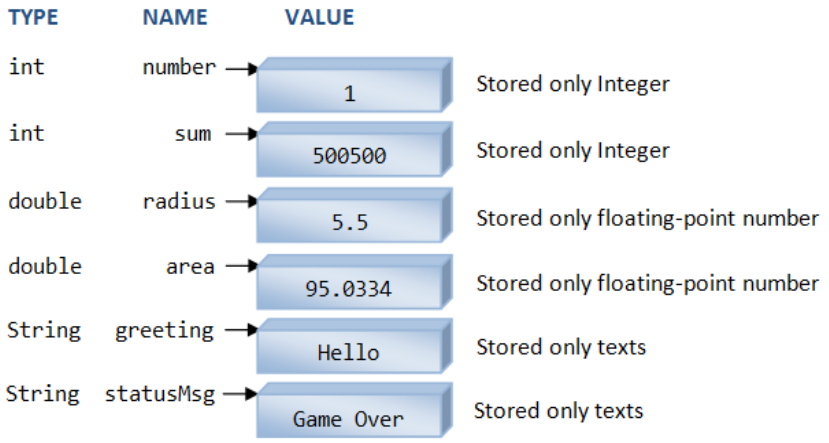
Secuencia de escape	Significado
\b	Retroceso
\n	Salto de línea
\t	Tabulación horizontal
\\	Barra invertida \
\'	Comilla simple
\"	Comilla doble

Tipos de datos primitivos

Tipo	Tamaño	Rango de representación
boolean	-	true (verdadero) o false (falso).
char	16 bits	Cualquier símbolo o carácter UNICODE.
byte	8 bits	Desde **-128 (-2^7)** a **+127 ($+2^7-1$)**.
short	16 bits	Desde **-32.768 (-2^{15})** a **+32.767 ($+2^{15}-1$)**.
int	32 bits	Desde **-2.147.483.648 (-2^{31})** a **+2.147.483.647 ($+2^{31}-1$)**.
long	64 bits	Desde **-2^{63}** a **$+2^{63}-1$**.
float	32 bits	Número en coma flotante utilizando la representación IEEE 754-1985 (revisión del estándar de Agosto de 2008).
double	64 bits	Número en coma flotante utilizando la representación IEEE 754-1985 (revisión del estándar de Agosto de 2008).

Variables

- Una variable es un espacio físico en memoria donde se puede almacenar un dato.
- Se distinguen dos tipos de variables en Java, en función del tipo de dato que almacenan:
  - Tipo **primitivo**:contienen el dato almacenado.
  - Tipo **objeto**: contienen una referencia al objeto.



Declaración de variables

- El tipo de dato puede ser un tipo primitivo u objeto.
- El nombre de la variable o **identificador** debe:
  - Comenzar por una letra, subrayado(_) o dólar ($).
  - Sin espacios, signos de puntuación y secuencias de escape.
  - No puede ser una palabra reservada.

```
tipo_dato identificador;
tipo_dato variable1, variable2, variable3;
```

Válido	*No válido*
`int edad, código;`	`boolean 6h; //empieza por un número`
`short b1;`	`int numero inicial; // contiene espacio`
`char un_caracter;`	`long class; //utiliza una palabra reservada`

Ámbito de las variables

- El ámbito de una variable determina su "**visibilidad**".
- Según dónde esté declarada, se distinguen:
  - **Atributo de clase**: declarada al principio de una clase, fuera de cualquier método. Son accesibles desde cualquier método de la misma clase.
  - **Variable local**: declarada dentro de cualquier bloque de instrucciones delimitado entre llaves ({…}).

```
public class Ejemplo{
     private int p; //atributo de clase
     public void metodo(..){
          char t; //variable local al método
          if(..){
               long c; //variable local al if
          }
     }
}
```

Convenciones para identificadores

- Java distingue mayúsculas y minúsculas: **Alumno** y **alumno** son identificadores diferentes.
- Deben ser lo más descriptivos posibles.

Identificador	*Convención*	*Ejemplo*
Nombre de variable	Comienza por letra minúscula, y si está formado por varias palabras, se colocan juntas y todas las siguientes a partir de la segunda comenzarán por mayúsculas	NumAlumnosMatriculados suma
Nombre de constante	Con todas sus letras en mayúsculas, separando las palabras con el guión bajo, y además por convenio el guión bajo no se utiliza en ningún otro sitio.	TAMAÑO_MAX, PI
Nombre de una clase	Comienza por letra mayúscula.	String, MiClase
Nombre de función o método	Sigue la misma convención que los nombres de variables.	calcularPotencia()

Inicialización de variables

- **Asignación**: consiste en dar un valor a una variable.
- Puede realizarse en la misma sentencia de declaración o posteriormente en cualquier parte del código.
- Según el tipo de dato de la variable, el valor asignado está restringido a un conjunto de valores válidos.

```
variable = expresión;
```

```
int p, k, v;
p=30;
k=p+20;
v=k*p;
```

```
int edad=33; //declara e inicializa la variable
boolean a, b=true; //declara dos variables pero se inicializa solo una
```

Constantes

- Su valor no puede ser modificado tras la inicialización.
- Por convención sus identificadores se escriben en mayúsculas.
- Para declarar una constante se utiliza la palabra reservada ***final***:

```
final tipo_dato NOMBRE_CTE=valor;
```

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

Palabras reservadas

- Son aquellos *tokens* que tienen asignada una función específica en el lenguaje.

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

Literales

- **Literales *boolean***: true/false.
- **Literales *char***:
  - Se escriben entre comillas simples: 'A', 'a', '7', '?','+'.
  - Representaciones numéricas del carácter en Unicode.

```
class LiteralChar {
        public static void main (String[] args) {
                char a1 = 'A';
                char a2 = '\115';    //Código octal del carácter
                char a3 = '\u0055'; //Código hexadecimal del carácter
                char a4 = 231;       //Código entero del carácter
                System.out.print("a1 = " + a1);
                System.out.print('\n');
                System.out.print("a2 = " + a2);
                System.out.print('\n');
                System.out.print("a3 = " + a3);
                System.out.print('\n');
                System.out.print("a4 = " + a4);
                System.out.print('\n');
        }
}
```

Literales

- **Literales enteros**: cualquier secuencia de dígitos (0-9) que puede llevar delante signo o no.
  - Se aplican a los tipos primitivos (*byte*, *short*, *int*, *long*) con la precaución de cada uno de su rango de valores.
  - Se pueden especificar en **decimal**, **hexadecimal** y **octal**.

```
int p=90;
int q=0132; //Se considera octal si comienza por 0 (cero)
int r=0X5A; //Se considera hexadecimal si comienza por 0x o 0X
int s=0x5A;
int t=0X5a;
int u=0x5a;
```

Literales

- **Literales *float* y *double***: son aquellos que:
  - contienen una parte entera y/o decimal. Se utiliza el punto para separar ambas partes aunque alguna de ellas puede estar vacía;
  - o poseen un exponente especificado tras la letra **e** o **E**;
  - o van precedidos por una letra de tipo, **f** o **F**(*float*), **d** o **D** (*double*). Si no se especifica letra de tipo (f, F, d, D) el literal se considera de tipo *double*.

```
float p=3.14; //error de compilación ya que el literal se considera double
float p=3.14f;//correcto
```

```
double a=.56;
double b=137.;
double c=158.4e-3;
double d=53234D;
```

Valores por defecto de los atributos de clase

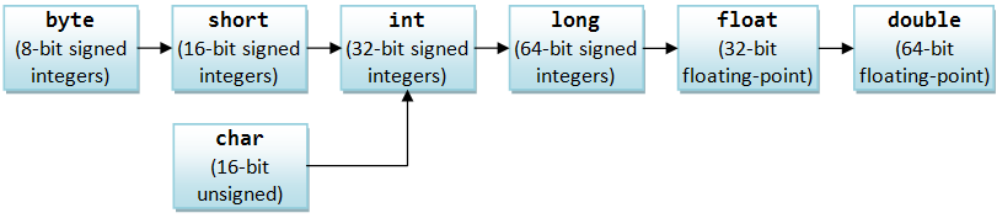
- Las variables locales deben ser inicializadas explícitamente antes de su utilización.
- Los atributos de clase se inicializan implícitamente a unos valores por defecto por el **constructor de la clase**.

Tipo de variable	*Valor por defecto*
byte, short, int, long	0
char	'\u0000'
float, double	0.0
boolean	false
objeto	null

```
void metodo(){
     int n;
     n=n+1; //error de compilación: variable local no inicializada
}
```

Conversiones de tipo

- Consiste en almacenar en una variable un dato de un tipo diferente al declarado para dicha variable.
- En Java es posible realizar conversiones entre todos los tipos primitivos, excepto *boolean*.
- Conversiones implícitas: Las realiza el compilador de manera automática.
- Conversiones explícitas o ***casting***: el programador debe indicar cuando se debe realizar esta conversión ya que puede producir pérdida de precisión en los datos.



Orders of Implicit Type-Casting for Primitives

Conversiones implícitas

- El tipo de la variable destino debe ser de tamaño igual o superior al tipo de origen, excepto:
  - Cuando la variable destino es entera y el origen es decimal.
  - Cuando la variable destino es *char* y el origen es numérico.

```
int k=5, p;
short s=10;
char c='ñ';
float h;
p=c; //conversión implícita de char a int
h=k; //conversión implícita de int a float
k=s; //conversión implícita de short a int
```

```
int n;
long c=20;
float ft=2.4f;
char k;
byte s=4;
n=c; //error de conversión implícita de long a int
k=s; //error de conversión implícita de byte a char
n=ft; //error de conversión implícita de float a int
```

Conversiones explícitas o casting

- Cuando no se reúnen las condiciones para una conversión implícita, puede realizarse la conversión mediante la expresión:

```
variable_destino=(tipo_destino) dato_origen;
```

```
char c;
byte k;
int p=400;
double d=34.6;
c=(char)d; //se elimina la parte decimal (truncado)
k=(byte)p; //se produce una pérdida de datos, pero la conversión es posible
```

No es posible la conversión entre variables de tipo objeto, pero sí la asignación de un objeto a una variable de una clase diferente si entre ellas existe una relación de herencia. Estudiaremos este tema más adelante.

Operadores

- Se clasifican según la operación que realizan en:
  - Operadores aritméticos.
  - Operadores de asignación.
  - Operadores condicionales.
  - Operadores lógicos.
  - Operadores a nivel de bits.

Operadores aritméticos

Operador	*Descripción*	*Ejemplo*
+	Suma dos valores numéricos.	`int c;` `c=2+5;`
-	Resta dos valores numéricos.	`int c;` `c=2-5;`
*	Multiplica dos números.	`int c;` `c=2*5;`

/	Divide dos números. El tipo de resultado depende de los operandos, pues en caso de que sean enteros, el resultado de la división siempre será entero.	`int a=8,b=5;`
		`float x,y;`
		`x=a/b;  //El resultado es 1`
		`y=a/3.0f; //El resultado es 2.66`

%	Calcula el resto de la división entre dos números.	`int c;` `c=7%2;  //El resultado es 1`
++	Incrementa una variable numérica en una unidad y deposita el resultado en la variable.	`int c=3;` `c++;  //Equivale a c=c+1;`
--	Decrementa una variable en una unidad y deposita el resultado en la variable.	`int c=3;` `c--;  //Equivale a c=c-1;`

Otros usos del operador +

- Para **concatenar cadenas de caracteres**.
- Para unir una cadena de caracteres con el valor contenido en una variable que no sea de tipo texto.

```
System.out.println("Hola "+"Juan"); //Mostraría: Hola Juan
```

```
int r, int n=5;
r=n*n;
System.out.println("El cuadrado de "+n+"es"+r);//Mostrará: El cuadrado de 5 es 25
```

Operadores ++ y --

- Pueden aparecer delante de la variable (prefijo) o detrás (postfijo).
  - **Prefijo**: el incremento/decremento del valor de la variable se realiza antes de que se tome su valor.
  - **Postfijo**: el incremento/decremento del valor de la variable se realiza después de que se tome su valor.

```
int a=3, b;
b=a++; //b=3, a=4
```

```
int a=3, b;
b=++a; //b=4, a=4
```

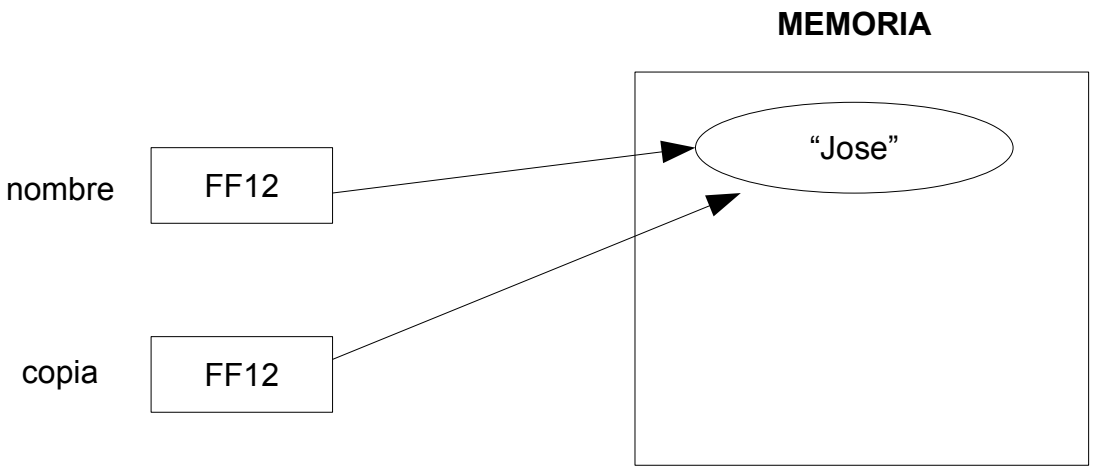
Operadores de asignación

Operador	*Descripción*	*Ejemplo*
=	Asigna la expresión de la derecha al operando situado a la izquierda del operador.	`int c;` `c=8*5;`
+=	Suma la expresión de la derecha a la variable situada a la izquierda del operador.	`int c=4;` `c+=5; //Equivale a c=c+5`
-=	Resta la expresión de la derecha a la variable situada a la izquierda del operador.	`int c=3;` `c-=2; //Equivale a c=c-2`
*=	Multiplica la expresión de la derecha a la variable situada a la izquierda del operador.	`int a=8;` `a*=2; //Equivale a a=a*2`
/=	Divide la expresión de la derecha a la variable situada a la izquierda del operador.	`int c=7;` `c/=2; //Equivale a c=c/2`
%=	Calcula el resto de la división entre la variable situada a la izquierda y la expresión de la derecha, depositando el resultado en la variable.	`int c=6;` `c%=3; //Equivale a c=c%3`

Asignación de referencias

- Cuando se utiliza el operador de asignación con variables tipo objeto, **no se copian los valores** de un objeto a otro como ocurre con las variables de tipo primitivo, **sino que se copian sus referencias**.
- En objetos inmutables no se apreciaría la diferencia pero hay que **tener especial cuidado con objetos mutables**.

```
String nombre=new String(“Jose”);
String copia=nombre;
```



Operadores condicionales

Operador	*Descripción*
==	Compara dos valores, en caso de que sean iguales el resultado de la operación será `true`.
<	Si el operando de la izquierda es menor que el de la derecha, el resultado es `true`.
>	Si el operando de la izquierda es mayor que el de la derecha, el resultado es `true`.
<=	Si el operando de la izquierda es menor o igual que el de la derecha, el resultado es `true`.
>=	Si el operando de la izquierda es mayor o igual que el de la derecha, el resultado es `true`.
!=	Si el valor de los operandos es diferente, el resultado es `true`.

Los operadores de comparación (<, >, <= y >=) únicamente pueden utilizarse para comparar enteros, puntos flotantes y caracteres (*char*). Si se utilizan con referencias a objetos, se producirá un error de compilación.

Los operadores de igualdad == y desigualdad != aplicado a variables de tipo objeto comparan las referencias y no los objetos. Esto hace que si comparamos dos variables que referencian a dos objetos iguales, el resultado de la comparación resultará ser falsa. Para comprobar la igualdad entre objetos, las clases proporcionan un método llamado *equals()*.

Operador condicional ternario

- Asigna un valor a una variable, entre dos posibles, en función del cumplimiento de una condición:

```
tipo variable = (condición)?valor_si_true:valor_si_false;
```

```
int k=4;
String s=(k%2==0)?“par”:“impar”;
System.out.println(s); //Mostraría: par
```

Operadores lógicos

Operador	*Descripción*
&&	Operador lógico AND. El resultado será `true` si los dos operandos son `true`, en cualquier otro caso el resultado será `false`.
||	Operador lógico OR. El resultado será `true` si alguno de los operandos es `true`.
!	Operador lógico NOT. Actúa sobre un único operando `boolean`, dando como resultado el valor contrario al que tenga el operando.

Los operadores && y || funcionan en modo cortocircuito, esto significa que si el primer operando determina el resultado de la operación, el segundo operando no será evaluado.

```
int p=4, f=2;
if((p>0) || (++f>0)){
     p++;
}
System.out.println("p vale"+p); //Mostraría: p vale 5
System.out.println("f vale"+f); //Mostraría: f vale 2
```

Operadores a nivel de bits

Operador	*Descripción*
&	Operador lógico AND. Realiza la operación AND entre los operandos, bit a bit.
|	Operador lógico OR. Realiza la operación OR entre los operandos, bit a bit.
^	Operador lógico OR exclusiva. Realiza la operación OR exclusiva entre los operandos, bit a bit.
~	Operador NOT. Invierte el estado de los bits del operando.
<<	Desplazamiento a la izquierda, rellenando con ceros los bits que quedan libres a la derecha.
>>	Desplazamiento a la derecha, rellenando con el bit mayor (de signo) los bits que quedan libres a la izquierda.
>>>	Desplazamiento a la derecha, rellenando con ceros los bits que quedan libres a la izquierda.

Operadores de bits en C++ y Java: http://www.zator.com/Cpp/E4_9_3.htm

Orden de precedencia

Operadores postfijos	→	**expr++, expr--**
Operadores prefijos y unarios	→	**++expr , --expr , +expr , -expr , ~ , !**
Creación y cast	→	**new , (tipo)expr**
Multiplicativos	→	*** , / , %**
Aditivos	→	**+ , -**
Desplazamiento (bits)	→	**<< , >> , >>>**
Relacionales	→	**< , <= , > , >=**
Igualdad	→	**= = , !=**
Conjunción bits (AND)	→	**&**
Disy. excl. bits (XOR)	→	**^**
Disy. incl. bits (OR)	→	**|**
Conjunción lógica	→	**&&**
Disyunción lógica	→	**||**
Condicional	→	**? :**
Asignación	→	**+= , -= ,+= , *= , /= , %= , &= , |= , ^= , <<= , >>= , >>>=**

Regla de asociatividad

- Todos los operadores son asociativos por la izquierda, es decir, a igual nivel de precedencia y a falta de paréntesis, se realizarán en el mismo orden que están escritos, excepto la asignación, que es asociativa por la derecha.

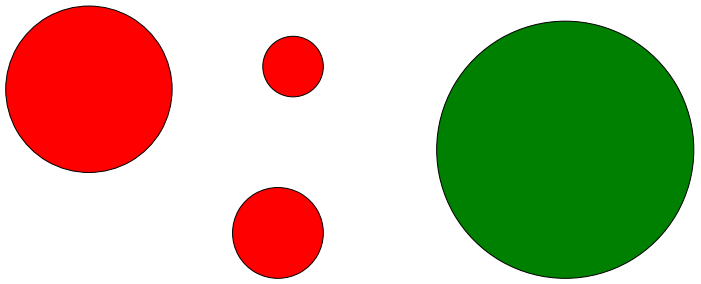
```
b * b - 4 * a * c < 0
((b * b) - ((4 * a) * c)) < 0
```

```
a = b = c = 1
a = (b = (c = 1))
```

Construcción de tipos objeto

- Un **objeto** es una entidad lógica que tiene una serie de propiedades llamadas **atributos** y sobre el que se puede realizar una serie de operaciones a través de sus **métodos**.
- Los objetos se definen por medio de las **clases**. Un objeto es una **instancia** de una clase.
- El operador **new** inicializa un objeto por medio del constructor de la clase y devuelve la referencia al objeto.

```
public class Circulo{
     double radio;
     String color;
     public Circulo(){
          radio=1.0;
          color=”rojo”;
     }
     public void setRadio(double r){
          this.radio=r;
     }
     //resto de métodos de la clase
}
```



Cadenas de caracteres: String

- En Java, las cadenas de caracteres se emplean por medio de la clase ***String***.
- Los literales ***String*** se delimitan por comillas dobles.
- Podemos declarar variables ***String*** de la siguiente forma:

```
String nombre=”Jose Luis”; //declaración e inicialización implícita
String apellidos=new String(“García Sanz”); //inicialización explícita llamando al
constructor de la clase
```

Métodos de la clase String

- Concatenar dos cadenas: operador + y método **concat()**
- Obtener una subcadena: **substring()**
- Buscar una subcadena o un carácter: **indexOf()**
- Obtener el carácter en una posición: **charAt()**
- Obtener la longitud de una cadena: **length()**
- Comprobar si empieza o termina con una secuencia de caracteres: **startsWith()**, **endsWith()**
- Reemplazar parte de una cadena por otra: **replaceAll()**
- Comparar cadenas: **equals()**, **equalsIgnoreCase()**
- Cambiar a mayúsculas/minúsculas: **toUpperCase()**, **toLowerCase()**
- Convertir un tipo primitivo numérico a String: **valueOf()**

Métodos de la clase String

```
String concat = “Hola”+”Jose”;
String substr = “El empleado encargado”.substring(3,11); //”empleado”
char c = “Empleado”.charAt(2); // 'p'
int longitud = “Empleado”.length(); // 8
boolean comienza = “Empleado”.startsWith(“Em”); //true
boolean termina = “Empleado”.endsWith(“a”); //false
String reemplaza = “El empleado”.replaceAll(“e”,”X”); //”El XmplXado”
```

Leer datos de teclado: Scanner

- El constructor de Scanner recibe un objeto *InputStream*.
  - *Scanner sc = new Scanner(System.in);*
- Se lee la cadena hasta la pulsación de la tecla “enter”.
- La cadena es dividida en “tokens”.
- El separador de tokens es por defecto el espacio en blanco.
  - **String next()**: devuelve el siguiente token.
  - **String nextLine()**: devuelve toda la cadena hasta el salto de línea.
  - **boolean hasNext()**: indica si existe o no un token.
  - **xxx nextXxx()**: devuelve el token como un tipo básico especificado por Xxx, por ejemplo, **nextInt()** o **nextFloat()**.
  - **boolean hasNextXxx()**: indica si existe un token de tipo Xxx.
  - **void useDelimiter(String d)**: establece un nuevo delimitador.

Leer datos de teclado: Scanner

- Supongamos que en el siguiente programa el usuario introdujera la cadena “José Luis”.

```
    public static void main(String[] args) {
         String nombre;
         int edad;
         Scanner sc = new Scanner(System.in);
         System.out.println("Introduzca su nombre:");
         nombre = sc.next();
         System.out.println("Indique su edad:");
         edad = sc.nextInt();
         System.out.println("Su nombre es "+nombre+" y tiene "+edad+" años");
    }
```

Al intentar convertir un token a un tipo que no le corresponde, los métodos de la clase Scanner lanzan la excepción InputMismatchException.

Leer datos de teclado: Scanner

- El siguiente programa valida un dato de entrada de tipo entero en un rango establecido:

```
Scanner sc = new Scanner(System.in);
int valor, min=-5, max=5;
do{
     while(!sc.hasNextInt())
          sc.next();
     valor = sc.nextInt();
}while(valor<5 || max>5);
```

Organización en paquetes

- Las clases se organizan y agrupan en **paquetes**.
- Los paquetes permiten crear ***bibliotecas*** y facilitan la **reutilización** del código.
  - Crear directorios: incluirlos en el PATH/CLASSPATH.
  - Situar el fichero .java dentro del directorio correspondiente.
  - Escribir el paquete en el fichero .java (primera sentencia):
    - ***package nombre_paquete***;

```
package mipaquete;
public class MiClase{...}
```

```
mipaquete.MiClase m = new mipaquete.MiClase();
```

```
import mipaquete;
MiClase m = new MiClase();
```

Organización en paquetes

- ***package*** e ***import*** permiten dividir el espacio de nombres de las clases de manera que sea único y global.
- Convención: la primera parte del nombre de un paquete es el dominio Internet del creador de la biblioteca, dado la vuelta:
  - ***com.joseberenguel.mipaquete;***
  - ***com/joseberenguel/mipaquete/<archivos.java>***

Librerías de Java

- Algunos de los paquetes más importantes que ofrece el lenguaje Java son:
  - ***java.io.*** Para operaciones de entrada y salida.
  - ***java.lang.*** Clases básicas del lenguaje (***Object***, envoltorios de tipos primitivos: ***Integer***, ***Long***, ***Float***, ***Double***…). No es necesario hace *import* para utilizarla.
  - ***java.util.*** Clases de utilidad general (***Date***, ***Scanner***).
  - ***java.math.*** Para operaciones matemáticas.
  - ***java.time***. Para manejar fechas y horas (***LocalDate***, ***LocalTime***).
  - ***java.swing.*** Para la construcción de interfaces gráficas con ***Swing***.
  - ***java.net.*** Para la programación de aplicaciones en red.
  - ***java.sql.*** Para programar el acceso a bases de datos.

Unidad 2. El lenguaje Java

DUDAS Y PREGUNTAS