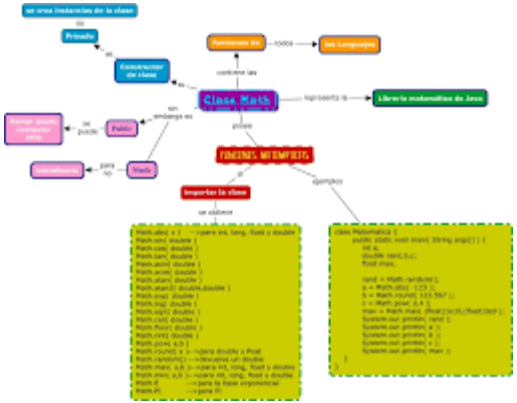
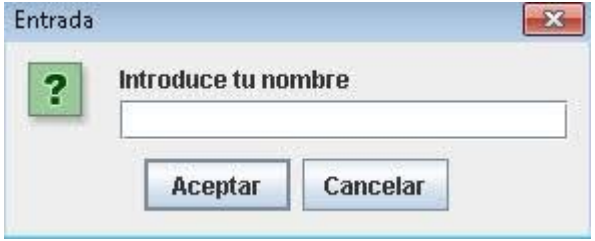
La clase Math y el proceso de Entrada/Salida Java String

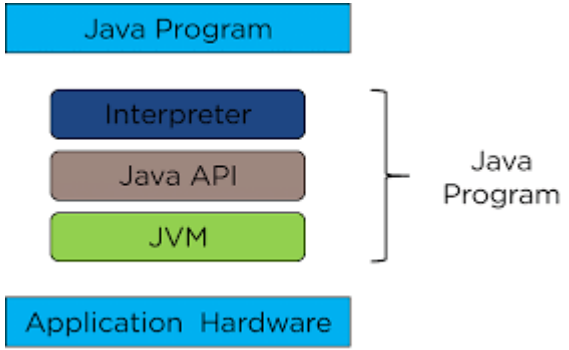




Encarnación Sánchez Gallego

API de Java

- Son las bibliotecas de clases que vienen predefinidas para que podamos usarlas.
- La API de Java es muy grande. Por tanto el conocimiento en profundidad del API no es tarea trivial, pero a la vez es imprescindible si se quieren abordar desarrollos extensos.
- La aproximación debe realizarse de forma progresiva, teniendo un conocimiento general de las capacidades globales del API, para ir adquiriendo una mayor especialización en función de las necesidades. Antes de intentar resolver cada problema de programación es conveniente reflexionar que packages del API pueden ayudarnos a resolverlo (si es que no está resuelto completamente).



La clase Math

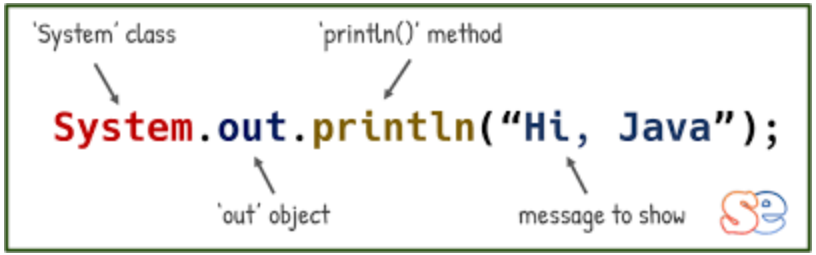
- Esta clase representa la librería matemática de Java.

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

¿Cómo se usa?

```
System.out.println("Pi es " + Math.PI);

System.out.println("e es " + Math.E);
```



Entrada de datos en Java

Se realiza con la clase Scanner de la API de Java. Esta clase tiene tres métodos más usados:

- nextLine(): permite introducir texto
- nextInt(): permite introducir enteros
- nextDouble(): permite introducir decimales.

Entrada de datos en Java

1.-Para la entrada de datos en Java debemos llamar a la clase: java.util.Scanner

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

2.- Declarar un objeto tipo Scanner:

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

System.out.println("Introduce tu nombre");

String nombreusuario = entrada.nextLine();
```

Ejemplo:

```
public class ejercicio1 {

    public static void main(String[] args) {

     Scanner entrada = new Scanner(System.in);

        System.out.println("Introduce tu nombre");

        String nombreusuario = entrada.nextLine();

      System.out.println("Te llamas " + nombreusuario);
```

JOptionPane

Esta clase hace realmente fácil el mostrar ventanas standards para mostrar y pedir información a los usuarios. Estas ventanas llamadas Dialogs, se muestran en forma “Modal” que significa que bloquean la aplicación hasta que son cerradas.

JOptionPane es parte de la librería Swing para el manejo de interfaces gráficas. Esta librería es muy compleja, pero en este tutorial veremos lo suficiente para mostrar y recibir información del usuario. Más adelante veremos Swing en más detalle.

showMessageDialog(Component parentComponent, Object message)

Este método crea una ventana que muestra un mensaje entregado en el parámetro *message*. El parámetro *parentComponent* es para indicar la ventana padre. En estos primeros tutoriales usaremos null en este parámetro.

```
import javax.swing.JOptionPane;
    public class HelloWorld {
        public static void main(String[] args) {
            JOptionPane.showMessageDialog(null, "Hello World");
```

String showInputDialog(Object message)

Este método es una función que muestra una ventana con una caja de texto y dos botones: Aceptar y Cancelar. Si oprimimos aceptar, recibiremos un String con el contenido de la caja de texto como retorno de la función. Si oprimimos cancelar, recibiremos un null como resultado.

```
import javax.swing.JOptionPane;
public class ShowInputDialogExample {
        public static void main(String[] args) {
                String name = JOptionPane.showInputDialog("Type your name please");
                JOptionPane.showMessageDialog(null, "Hello " + name);
        }
```

Ejemplo: Usted es mayor de edad?

```
public static void main(String[] args) {
        int edad = Integer.parseInt(JOptionPane.showInputDialog("Ingrese su edad"));
        String respuesta = (edad >= 18) ? "Usted es mayor de edad" : "Usted es menor de edad";
        JOptionPane.showMessageDialog(null, respuesta);
 }
```


Strings en Java


#include <stdio.h>
int main()
{
printf("Hello World");
return 42;
}

Clase String. Principales métodos

Uno de los tipos de datos más importantes de Java es **String**. *String* define y admite cadenas de caracteres. En algunos otros lenguajes de programación, una cadena o string es una matriz o array de caracteres. Este no es el caso con Java. En Java, **los String son objetos**.

En realidad, has estado usando la clase String desde el comienzo del curso, pero no lo sabías. Cuando crea un literal de cadena, en realidad está creando un objeto String. Por ejemplo, en la declaración:

```
System.out.println("En Java, los String son objetos");
```

Construcción del String (I)

Puede construir un *String* igual que construye cualquier otro tipo de objeto: utilizando new y llamando al constructor *String*. Por ejemplo:

```
String str = new String("Hola");
```

Esto crea un objeto *String* llamado *str* que contiene la cadena de caracteres “Hola”. También puedes construir una *String* desde otro *String*. Por ejemplo:

```
String str = new String("Hola");
String str2 = new String(str);
```

Después de que esta secuencia se ejecuta, str2 también contendrá la cadena de caracteres “Hola”. Otra forma fácil de crear una cadena se muestra aquí:

```
String str = "Estoy aprendiendo sobre String en JavadesdeCero.";
```

Construcción del String (II)

En este caso, *str* se inicializa en la secuencia de caracteres “*Estoy aprendiendo sobre String en JavadesdeCero*.”. Una vez que haya creado un objeto String, puede usarlo en cualquier lugar que permita una cadena entrecomillada. Por ejemplo, puede usar un objeto String como argumento para *println()*, como se muestra en este ejemplo:

```
// Uso de String
class DemoString
{
    public static void main(String args[])
    {
        //Declaración de String de diferentes maneras
        String str1=new String("En Java, los String son objetos");
        String str2=new String("Se construyen de varias maneras");
        String str3=new String(str2);

        System.out.println(str1);
        System.out.println(str2);
        System.out.println(str3);

    }
 }
```

Construcción del String (III)

La salida del programa se muestra a continuación:

```
En Java, los String son objetos
Se construyen de varias maneras
Se construyen de varias maneras
```

Métodos de la clase String (I)

- **int length()**: Devuelve la cantidad de caracteres del String.

```
"Javadesdecero.es".length(); // retorna 16
```

- **Char charAt(int i)**: Devuelve el carácter en el índice *i*.

```
System.out.println("Javadesdecero.es".charAt(3)); // retorna 'a'
```

- **String substring (int i)**: Devuelve la subcadena del i-ésimo carácter de índice al final.

```
"Javadesdecero.es".substring(4); // retorna desdecero.es
```

- **String substring (int i, int j)**: Devuelve la subcadena del índice *i* a *j-1*.

```
"Javadesdecero.es".substring(4,9); // retorna desde
```

- **String concat(String str)**: Concatena la cadena especificada al final de esta cadena.

```
        String s1 = "Java";
String s2 = "desdeCero;
String salida = s1.concat(s2); // retorna "JavadesdeCero"
```

Métodos de la clase String (II)

- **int indexOf (String s)**: Devuelve el índice dentro de la cadena de la primera aparición de la cadena especificada.

```
String s = "Java desde Cero";
int salida = s.indexOf("Cero"); // retorna 11
```

- **int indexOf (String s, int i)**: Devuelve el índice dentro de la cadena de la primera aparición de la cadena especificada, comenzando en el índice especificado.

```
String s = "Java desde Cero";
int salida = s.indexOf('a',3); //retorna 3
```

- **Int lastIndexOf (int ch)**: Devuelve el índice dentro de la cadena de la última aparición de la cadena especificada.

```
String s = "Java desde Cero";
int salida = s.lastIndexOf('a'); // retorna 3
```

- **boolean equals (Objeto otroObjeto)**: Compara este String con el objeto especificado.

```
Boolean salida = "Java".equals("Java"); // retorna true
Boolean salida = "Java".equals("java"); // retorna false
```

Métodos de la clase String (III)

- **int compareTo (String otroString)**: Compara dos cadenas lexicográficamente.

```
int salida = s1.compareTo(s2); // donde s1 y s2 son
                    // strings que se comparan

Esto devuelve la diferencia s1-s2. Si :
                                salida < 0 // s1 es menor que s2
                                salida = 0 // s1 y s2 son iguales
                                salida > 0 // s1 es mayor que s2
```

- **int compareToIgnoreCase (String otroString)**: Compara dos cadenas lexicográficamente, ignorando las consideraciones *case*.

```
int salida = s1.compareToIgnoreCase(s2); // donde s1 y s2 son
                    // strings que se comparan

Esto devuelve la diferencia s1-s2. Si :
                                salida < 0 // s1 es menor que s2
                                salida = 0 // s1 y s2 son iguales
                                salida > 0 // s1 es mayor que s2
```

Métodos de la clase String (IV)

- **String toLowerCase()**: Convierte todos los caracteres de String a minúsculas.

```
String palabra1 = "HoLa";
String palabra2 = palabra1.toLowerCase(); // retorna "hola"
```

- **String toUpperCase()**: Convierte todos los caracteres de String a mayúsculas.

```
String palabra1 = "HoLa";
String palabra2 = palabra1.toUpperCase(); // retorna "HOLA"
```

- **String trim()**: Devuelve la copia de la cadena, eliminando espacios en blanco en ambos extremos. No afecta los espacios en blanco en el medio.

```
String palabra1 = " Java desde Cero ";
String palabra2 = palabra1.trim(); // retorna "Java desde Cero"
```

- **String replace (char oldChar, char newChar)**: Devuelve una nueva cadena al reemplazar todas las ocurrencias de oldChar con newChar.

```
String palabra1 = "yavadesdecero";
String palabra2 = palabra1.replace('y' ,'j'); //retorna javadesdecero
```

Resumen

Métodos de String	Función que realizan
String(...)	Constructores para crear Strings a partir de arrays de bytes o de caracteres (ver documentación on-line)
String(String str) y String(StringBuffer sb)	Costructores a partir de un objeto String o StringBuffer
charAt(int)	Devuelve el carácter en la posición especificada
getChars(int, int, char[], int)	Copia los caracteres indicados en la posición indicada de un array de caracteres
indexOf(String, [int])	Devuelve la posición en la que aparece por primera vez un String en otro String, a partir de una posición dada (opcional)
lastIndexOf(String, [int])	Devuelve la última vez que un String aparece en otro empezando en una posición y hacia el principio
length()	Devuelve el número de caracteres de la cadena
replace(char, char)	Sustituye un carácter por otro en un String
startsWith(String)	Indica si un String comienza con otro String o no
substring(int, int)	Devuelve un String extraído de otro
toLowerCase()	Convierte en minúsculas (puede tener en cuenta el locale)
toUpperCase()	Convierte en mayúsculas (puede tener en cuenta el locale)
trim()	Elimina los espacios en blanco al comienzo y final de la cadena
valueOf()	Devuelve la representación como String de sus argumento. Admite Object, arrays de caracteres y los tipos primitivos

Métodos de la clase String.
Método split Manejo de cadenas

En Java el método split(String regEx) nos permite dividir una cadena en base a las ocurrencias de una expresión regular definida dentro. Ya que es una expresión regular puede ser simplemente una letra, coma, signo o una construcción más elaborada.

Hay que tomar en cuenta que existen símbolos especiales en las expresiones regulares como el punto, el signo más, los corchetes, etc. No se pueden agregar literalmente ya que dan problema. En caso de querer usarlos como cadenas se debe indicar usando \\. Por ejemplo \\. \\+ \\[etc.

Métodos de la clase String. Método split
Manejo de cadenas

```
public class SplitExample1 {


 public static void main(String[] args) {


    //Se puede dividir por medio de comas o palabras
    String dias = "Lunes,Martes,Miercoles,Jueves,Viernes,Sabado,Domingo";
    String diaArray[] = dias.split(",");


    System.out.println("--Ejemplo 1--");
    for(String dia : diaArray){
            System.out.println(dia);
    }
}
```

Salida de código:

```
--Ejemplo 1--
Lunes
Martes
Miercoles
Jueves
Viernes
Sabado
Domingo
```

```
//El punto se usa en las expresiones regulares por lo que
//si se desea usar como tal se debe definir con \\
//Otros valors son + [ ] ? etc.
String diasPunto = "Lunes.Martes.Miercoles.Jueves.Viernes.Sabado.Domingo";
String diaPuntoArray[] = diasPunto.split("\\.");


System.out.println("--Ejemplo 2--");
for(String diaPunto : diaPuntoArray){
          System.out.println(diaPunto);
}
```

Salida de código:

```
Lunes
Martes
Miercoles
Jueves
Viernes
Sabado
Domingo
```

Expresiones regulares (I)

Una expresión regular define un patrón de búsqueda para cadenas de caracteres.

La podemos utilizar para comprobar si una cadena contiene o coincide con el patrón. El contenido de la cadena de caracteres puede coincidir con el patrón 0, 1 o más veces.

Algunos ejemplos de uso de expresiones regulares pueden ser:

- para comprobar que la fecha leída cumple el patrón dd/mm/aaaa
- para comprobar que un NIF está formado por 8 cifras, un guión y una letra
- para comprobar que una dirección de correo electrónico es una dirección válida.
- para comprobar que una contraseña cumple unas determinadas condiciones.
- Para comprobar que una URL es válida.
- Para comprobar cuántas veces se repite dentro de la cadena una secuencia de caracteres determinada.
- Etc. Etc.

El patrón se busca en el String de izquierda a derecha. Cuando se determina que un carácter cumple con el patrón este carácter ya no vuelve a intervenir en la comprobación.

Símbolos comunes en expresiones regulares

Expresión	Descripción
.	Un punto indica cualquier carácter
^expresión	El símbolo ^ indica el principio del String. En este caso el String debe contener la expresión al principio.
expresión$	El símbolo $ indica el final del String. En este caso el String debe contener la expresión al final.
[abc]	Los corchetes representan una definición de conjunto. En este ejemplo el String debe contener las letras a ó b ó c.
[abc][12]	El String debe contener las letras a ó b ó c seguidas de 1 ó 2
[^abc]	El símbolo ^ dentro de los corchetes indica negación. En este caso el String debe contener cualquier carácter excepto a ó b ó c.
[a-z1-9]	Rango. Indica las letras minúsculas desde la a hasta la z (ambas incluidas) y los dígitos desde el 1 hasta el 9 (ambos incluidos)
A|B	El carácter | es un OR. A ó B
AB	Concatenación. A seguida de B

Meta caracteres

Expresión	Descripción
\d	Dígito. Equivale a [0-9]
\D	No dígito. Equivale a [^0-9]
\s	Espacio en blanco. Equivale a [\t\n\x0b\r\f]
\S	No espacio en blanco. Equivale a [^\s]
\w	Una letra mayúscula o minúscula, un dígito o el carácter _ Equivale a [a-zA-Z0-9_]
\W	Equivale a [^\w]
\b	Límite de una palabra.

En Java debemos usar una doble barra invertida \\

Por ejemplo para utilizar \w tendremos que escribir \\w. Si queremos indicar que la barra invertida en un carácter de la expresión regular tendremos que escribir \\\\.

Cuantificadores

Expresión	Descripción
{X}	Indica que lo que va justo antes de las llaves se repite X veces
{X,Y}	Indica que lo que va justo antes de las llaves se repite mínimo X veces y máximo Y veces. También podemos poner {X,} indicando que se repite un mínimo de X veces sin límite máximo.
*	Indica 0 ó más veces. Equivale a {0,}
+	Indica 1 ó más veces. Equivale a {1,}
?	Indica 0 ó 1 veces. Equivale a {0,1}

Clase Pattern y Matcher

Para usar expresiones regulares en Java se usa el package **java.util.regex**

Contiene las clases **Pattern** y **Matcher** y la excepción **PatternSyntaxException**.

Clase **Pattern:** Un objeto de esta clase representa la expresión regular. Contiene el método compile(String regex) que recibe como parámetro la expresión regular y devuelve un objeto de la clase Pattern.

La clase **Matcher:** Esta clase compara el String y la expresión regular. Contienen el método matches(CharSequence input) que recibe como parámetro el String a validar y devuelve true si coincide con el patrón. El método find() indica si el String contienen el patrón.

Ejemplos de expresiones regulares

1. Comprobar si el String *cadena* contiene exactamente el patrón (matches) “abc”

```
Pattern pat = Pattern.compile("abc");
Matcher mat = pat.matcher(cadena);
if (mat.matches()) {
    System.out.println("SI");
} else {
    System.out.println("NO");
}
```

2. Comprobar si el String *cadena* contiene “abc”

```
Pattern pat = Pattern.compile(".*abc.*");
Matcher mat = pat.matcher(cadena);
if (mat.matches()) {
    System.out.println("SI");
} else {
    System.out.println("NO");
}
```

3. Comprobar si el String *cadena* empieza por "abc"

```
Pattern pat = Pattern.compile("^abc.*");
 Matcher mat = pat.matcher(cadena);
 if (mat.matches()) {
     System.out.println("Válido");
 } else {
     System.out.println("No Válido");
 }
```

4. Comprobar si el String *cadena* empieza por "abc" ó "Abc"

```
Pattern pat = Pattern.compile("^[aA]bc.*");
 Matcher mat = pat.matcher(cadena);
 if (mat.matches()) {
     System.out.println("SI");
 } else {
     System.out.println("NO");
 }
```

5. Comprobar si el String *cadena* está formado por un mínimo de 5 letras mayúsculas o minúsculas y un máximo de 10.

```
Pattern pat = Pattern.compile("[a-zA-Z]{5,10}");
 Matcher mat = pat.matcher(cadena);
 if (mat.matches()) {
     System.out.println("SI");
 } else {
     System.out.println("NO");
 }
```

6. Comprobar si el String *cadena* no empieza por un dígito

```
Pattern pat = Pattern.compile("^[^\\d].*");
 Matcher mat = pat.matcher(cadena);
 if (mat.matches()) {
     System.out.println("SI");
 } else {
     System.out.println("NO");
 }
```

7. Comprobar si el String *cadena* no acaba con un dígito

```
Pattern pat = Pattern.compile(".*[^\\d]$");
Matcher mat = pat.matcher(cadena);
if (mat.matches()) {
    System.out.println("SI");
} else {
    System.out.println("NO");
}
```

8. Comprobar si el String *cadena* solo contienen los caracteres a ó b

```
Pattern pat = Pattern.compile("(a|b)+");
Matcher mat = pat.matcher(cadena);
if (mat.matches()) {
    System.out.println("SI");
} else {
    System.out.println("NO");
}
```

Ejemplo: expresión regular para comprobar si un email es válido

```
package ejemplo1;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Ejemplo1 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        String email;
        System.out.print("Introduce email: ");
        email = sc.nextLine();
        Pattern pat = Pattern.compile("^[\\w-]+(\\.[\\w-]+)*@[A-Za-z0-9]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$");
        Matcher mat = pat.matcher(email);
        if(mat.find()){
            System.out.println("Correo Válido");
        }else{
            System.out.println("Correo No Válido");
    }
   }
}
```

"^[\\w-]+(\\.[\\w-]+)*@[A-Za-z0-9]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$"

La explicación de cada parte de la expresión regular es la siguiente:

[\\w-]+	Inicio del email El signo + indica que debe aparecer uno o más de los caracteres entre corchetes: \\w indica caracteres de la A a la Z tanto mayúsculas como minúsculas, dígitos del 0 al 9 y el carácter _ Carácter – En lugar de usar \\w podemos escribir el rango de caracteres con lo que esta expresión quedaría así: [A-Za-z0-9-_]+
(\\.[\\w-]+)*	A continuación: El * indica que este grupo puede aparecer cero o más veces. El email puede contener de forma opcional un punto seguido de uno o más de los caracteres entre corchetes.
@	A continuación debe contener el carácter @
[A-Za-z0-9]+	Después de la @ el email debe contener uno o más de los caracteres que aparecen entre los corchetes
(\\.[A-Za-z0-9]+)*	Seguido (opcional, 0 ó más veces) de un punto y 1 ó más de los caracteres entre corchetes
(\\.[A-Za-z]{2,})	Seguido de un punto y al menos 2 de los caracteres que aparecen entre corchetes (final del email)

Secuencias de escape en Java

Una secuencia de escape esta formada por una barra inversa seguida de una letra, un carácter o de una combinación de dígitos.
Una secuencia de escape siempre representa un solo carácter aunque se escriba con dos o más caracteres.
Se utilizan para realizar acciones como salto de línea o para usar caracteres no imprimibles.

Algunas secuencias de escape definidas en Java son:

\n -----> Nueva Linea.
\t -----> Tabulador.
\r -----> Retroceso de Carro.
\f -----> Comienzo de Pagina.
\b -----> Borrado a la Izquierda.
\\ -----> El carácter barra inversa (\ ).
\' -----> El carácter prima simple (').
\" -----> El carácter prima doble (").

Secuencias de escape en Java. Ejemplos

```
System.out.println("Juan\nVictor\nAlfonso\nEnrique");
```

Salida por pantalla:

```
Juan
Victor
Alfonso
Enrique
```

```
System.out.println("Lunes\rMartes, Miércoles");
```

Salida por pantalla:

```
Martes, Miércoles
```

```
System.out.println("Lunes\bMartes");
```

Salida por pantalla:

```
LuneMartes
```

```
System.out.println("Lunes\tMartes\tMiércoles");
```

Salida por pantalla:

```
Lunes    Martes    Miércoles
```

```
System.out.println("\"Lunes\",\"Martes\",\'Miércoles\'");
```

Salida por pantalla:

```
"Lunes","Martes",'Miércoles'
```