Estructuras de Control

Entrada y salida de información

Entrada por argumentos

El método main visto anteriormente recibe una serie de parámetros cuyo valor se obtiene de los argumentos indicados en la ejecución del programa.

```
public class Argumentos {

    public static void main(String[] args) {
        System.out.println(args[0] + args[1] + args[2] + args[3]);
    }

}
```

```
java Argumentos 1 2 3 4
```

Entrada y salida por consola

Flujo estándar de salida: ***System.out***

- System.out.write(datos) → Imprime los bytes enviados como argumento.
- System.out.print(obj) → Se transforma obj en un conjunto de bytes y se imprime con el método anterior.
- System.out.println(obj) → Igual que el anterior pero añade un salto de línea al final.
- System.out.printf(String cadena, [argumento1],...,[argumentoN]) → Imprime la cadena con el formato indicado.

Entrada y salida por consola

Entrada mediante la clase ***Scanner***. Divide la entrada de información en *tokens* que, por defecto, vienen separados por espacios. Alguno de los métodos que proporciona son:

- nextLine() → Devuelve la línea completa como un String.
- next() → Retorna el siguiente token como un String.
- nextXXXX() → Retorna el siguiente token como el tipo indicado (siendo XXXX el tipo : nextInt(), nextFloat(),...)
- hasNext() → Devuelve true si el escáner puede devolver otro token.
- hasNextXXXX() → Devuelve true si el próximo token puede interpretarse como el tipo indicado (siendo XXX el tipo: hasNextInt(), hasNextDouble(),...)

Clase Scanner

¿Cómo funciona el elemento Scanner?

Al crear un objeto de tipo Scanner debemos indicar sobre qué entrada queremos definirlo (nosotros lo utilizaremos sobre la entrada por teclado, que se indica con ***System.in***):

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

Desde este momento el objeto sc lo usaremos para ‘observar’ lo que el usuario introduce por teclado.

Clase Scanner

Los valores que introducimos por teclado en consola se interpretan por Scanner de dos formas:

- En la mayoría de métodos de Scanner, se separa la entrada de texto en *tokens*, que serán los valores separados por espacios.
  Por ejemplo, ante la entrada “Buenos días a todos” tendremos 4 tokens:
  - Buenos
  - días
  - a
  - todos
- También existen métodos de Scanner que nos devuelven todo lo ingresado hasta pulsar un enter , como el método ***nextLine()***

Clase Scanner

El objeto de tipo Scanner tiene un buffer de memoria, donde se van interpretando los *tokens* que se obtienen de la entrada por teclado.

Si no existe un valor en el buffer de memoria es cuando se queda esperando a que el usuario introduzca un valor por consola.

Los métodos hasNextXXX (por ejemplo hasNextInt()) evalúan el contenido del buffer de memoria pero no retiran lo que está ahí contenido.

Los metodos nextXXX (por ejemplo nextInt()) retiran el elemento del buffer de memoria intentando realizar una conversión al tipo indicado (en el caso de nextInt() intenta convertirlo a Int)

Clase Scanner

Veamos el siguiente programa.

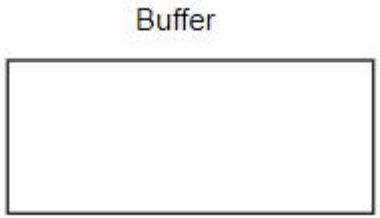
Pedirá un número y, si no se indica un valor entero, seguirá pidiendo el valor en bucle.

```java
import java.util.Scanner;

public class EjemploScanner {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int num=0;
        Scanner sc= new Scanner(System.in);

        boolean esUnNumero=false;

        //Bucle para pedir un valor mientras no se introduzca un número
        while(esUnNumero==false)
        {
            System.out.println("Introduce un número");
            if(sc.hasNextInt())
            {
                num = sc.nextInt();
                esUnNumero=true;
            }
        }

        System.out.println("El número indicado es " + num);

        sc.close();
    }//main

}//clase
```

Clase Scanner

En la primera ejecución del bucle, la ejecución del programa se detiene en el ***sc.hasNextInt()*** y espera que el usuario introduzca un valor, ya que el buffer de sc está vacío.

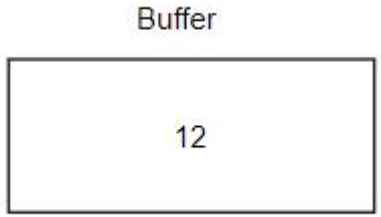
```
//Bucle para pedir un valor mientras no se introduzca un número
while(esUnNumero==false)
{
    System.out.println("Introduce un número");
    if(sc.hasNextInt())
    {
        num = sc.nextInt();
        esUnNumero=true;
    }
}
```



Clase Scanner

Si se introduce un valor numérico entero (por ejemplo 12), el *hasNextInt()* devuelve un valor ***True*** con lo que la ejecución entra en el if

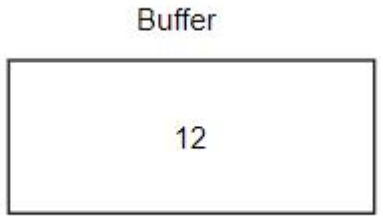
```
//Bucle para pedir un valor mientras no se introduzca un número
while(esUnNumero==false)
{
    System.out.println("Introduce un número");
    if(sc.hasNextInt())
    {
        num = sc.nextInt();
        esUnNumero=true;
    }
}
```



Clase Scanner

Cuando se ejecuta el ***sc.nextInt()*** no se queda esperando a que el usuario introduzca un valor ya que ese valor 12 aun sigue dentro del buffer de sc.

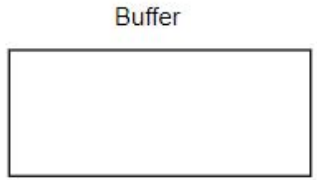
```
//Bucle para pedir un valor mientras no se introduzca un número
while(esUnNumero==false)
{
    System.out.println("Introduce un número");
    if(sc.hasNextInt())
    {
        num = sc.nextInt();
        esUnNumero=true;
    }
}
```



Clase Scanner

Al ejecutarse esa línea ***num=sc.nextInt()*** se obtiene el valor del Buffer, se guarda en ***num*** y se retira del buffer el *token* actual (el 12)
Como se cambia el valor de ***esUnNumero*** a true no se mantendría en el bucle y terminaría el programa

```
//Bucle para pedir un valor mientras no se introduzca un número
while(esUnNumero==false)
{
    System.out.println("Introduce un número");
    if(sc.hasNextInt())
    {
        num = sc.nextInt();
        esUnNumero=true;
    }
}
```

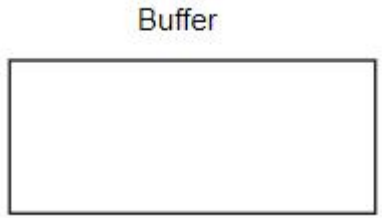


Clase Scanner

Pero, ¿qué pasaría si lo que introduce el usuario es un valor no entero

En la primera ejecución el buffer está vacío y, por ejemplo, introducimos el valor no numérico “el”

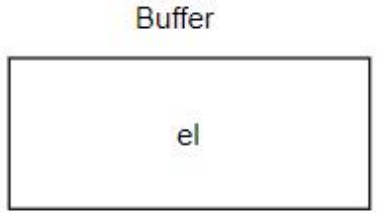
```
//Bucle para pedir un valor mientras no se introduzca un número
while(esUnNumero==false)
{
    System.out.println("Introduce un número");
    if(sc.hasNextInt())
    {
        num = sc.nextInt();
        esUnNumero=true;
    }
}
```



Clase Scanner

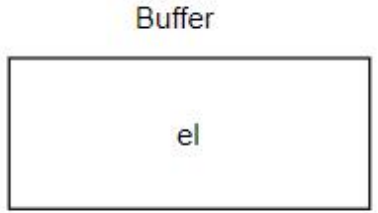
El método ***hasNextInt()*** devuelve un false, con lo que la ejecución no entra en el código del if y no se retira el valor del buffer.
Se mantiene el valor de **esUnNumero** a **False** con lo que se repetiría la ejecución del código del bucle

```
//Bucle para pedir un valor mientras no se introduzca un número
while(esUnNumero==false)
{
    System.out.println("Introduce un número");
    if(sc.hasNextInt())
    {
        num = sc.nextInt();
        esUnNumero=true;
    }
}
```



Clase Scanner

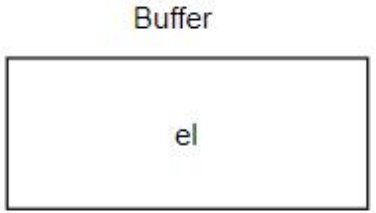
La siguiente ejecución de ***hasNextInt()*** no se queda esperando a que el usuario introduzca un valor (ya que el buffer aún mantiene el valor “el”) con lo que nos quedamos en un bucle infinito donde sólo se nos mostraría el mensaje “Introduce un número” pero sin esperar contestación del usuario.



Clase Scanner

Para solucionar esto tenemos que eliminar el valor que está en el Buffer de sc en el caso en que no sea un entero (en un else de ***hasNextInt***())

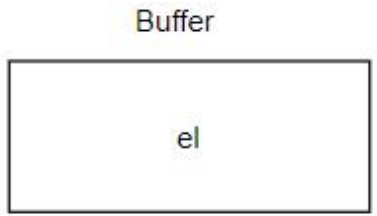
```
//Bucle para pedir un valor mientras no se introduzca un número
while(esUnNumero==false)
{
    System.out.println("Introduce un número");
    if(sc.hasNextInt())
    {
        num = sc.nextInt();
        esUnNumero=true;
    }
    else
    {
        sc.next();
        esUnNumero=false;
    }
}
```



Clase Scanner

Con este código y el ejemplo anterior, la ejecución entraría por el **else** y el ***sc.next()*** (que obtiene el valor del buffer como un **String**) se encargaría de quitar el valor no numérico del Buffer.
En este caso ni siquiera se guarda el valor de retorno de ***sc.next()*** en ninguna variable al no necesitarlo.

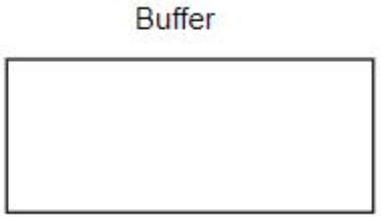
```
//Bucle para pedir un valor mientras no se introduzca un número
while(esUnNumero==false)
{
    System.out.println("Introduce un número");
    if(sc.hasNextInt())
    {
        num = sc.nextInt();
        esUnNumero=true;
    }
    else
    {
        sc.next();
        esUnNumero=false;
    }
}
```



Clase Scanner

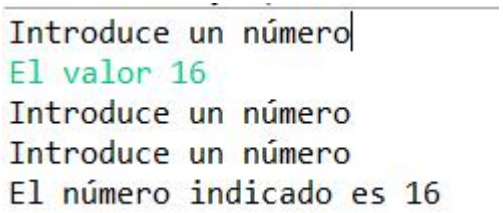
Como esUnNumero tiene un valor False se mantiene dentro del bucle y, en la siguiente ejecución de sc.hasNextInt() el programa se queda esperando un valor por consola del usuario ya que el Buffer está vacío.

```
//Bucle para pedir un valor mientras no se introduzca un número
while(esUnNumero==false)
{
    System.out.println("Introduce un número");
    if(sc.hasNextInt())
    {
        num = sc.nextInt();
        esUnNumero=true;
    }
    else
    {
        sc.next();
        esUnNumero=false;
    }
}
```



Clase Scanner

Hay que tener en cuenta que tanto ***hasNextInt***() como ***nextInt***() o ***next***() son métodos que actúan sobre tokens.
Si con este código el texto que introducimos es “El valor 16” la salida por consola sería la siguiente:



Clase Scanner

La explicación es que se evalúa en tokens (separaciones de las palabras separadas por espacios) con lo cual:

En la primera ejecución del bucle, lo que hay en el buffer es “El”, con lo que al no ser numérico se elimina del buffer con el ***sc.next***().

En la segunda ejecución del bucle lo que hay ahora en el Buffer es “valor”. Al igual que antes se elimina con el ***sc.next***().

En la tercera ejecución del bucle, el Buffer tiene el valor “16”, con lo que sí es un entero y termina el bucle y el programa.

Entrada y salida por consola

```java
import java.util.Scanner;

public class EntradaSalida {

    public static void main(String[] args) {
        System.out.println("Introduzca el texto a reproducir");
        Scanner entradaEsc=new Scanner(System.in);
        String entradaTeclado=entradaEsc.nextLine();
        System.out.println(entradaTeclado);
        entradaEsc.close();

    }

}
```

Print, println y printf

El método **System.out.print** imprime un mensaje en la consola sin hacer un salto de línea al final. El cursor se mantiene en la misma línea después de que se imprime el mensaje, por lo que lo siguiente que imprimas aparecerá seguido en la misma línea.

El método **System.out.println** imprime un mensaje en la consola y luego realiza un salto de línea. El cursor pasa a la siguiente línea después de imprimir el mensaje, con lo que lo siguiente que imprimas aparecerá en una nueva línea.

El método **System.out.printf** se utiliza para imprimir texto formateado en la consola. A diferencia de println, que solo imprime un mensaje, printf te permite controlar el formato exacto de los datos que imprimes (número de decimales, alineación, longitud, etc.).

Printf

Formatos más comunes:
%d: Para enteros decimales (int, long).
Ejemplo: System.out.printf("%d", 25); imprimirá 25.

%f: Para números de punto flotante (float, double).
Ejemplo: System.out.printf("%.2f", 123.456); imprimirá 123.46 (con 2 decimales).

%s: Para cadenas de texto (String).
Ejemplo: System.out.printf("%s", "Hola"); imprimirá Hola.

%c: Para caracteres (char).
Ejemplo: System.out.printf("%c", 'A'); imprimirá A.

Printf (2)

%n: Para saltar a una nueva línea (equivalente a \n).

%%: Para imprimir el símbolo de porcentaje (%).
Ejemplo: System.out.printf("Porcentaje: 90%%"); imprimirá Porcentaje: 90%.

Modificadores adicionales:
.nf: Controla el número de decimales que deseas imprimir.
Ejemplo: %.2f imprimirá un número flotante con 2 decimales.

%nd: Para especificar el ancho mínimo del campo en el que se imprimirá el número.
Ejemplo: %5d imprimirá un número entero en un espacio de al menos 5 caracteres.

Estructuras de Selección

If, else, else if

If: Se evalúa lo indicado entre paréntesis y, si el valor de retorno es verdadero, ejecutará el bloque de instrucciones entre llaves.

```
if(a>b)
{
    mayor=a;
    menor=b;
}
```

Estructuras de Selección

If, else, else if

If-else: Similar a la anterior. Si el resultado evaluado es false se ejecuta el bloque de código tras el else.

```
if(a>b)
{
    mayor=a;
}
else
{
    mayor=b;
}
```

Estructuras de Selección

If, else, else if

If-else if: Las sentencias if-else se pueden anidar.

```
if(dia=="Lunes")
{
    //
}else if(dia=="Martes")
{
    //
}else if(dia=="Miercoles")
{
    //
}else
{
    //
}
```

Estructuras de Selección

Switch: Mediante la sentencia switch se puede evaluar una variable ejecutando diferentes sentencias dependiendo de su valor.

La sentencia de control puede evaluar:

- Tipos primitivos: byte, int, char, short,...
- Tipos enumerados
- Strings
- Clases envoltorio: Byte, Integer, Character y Byte

La sentencia ***break*** se utiliza para salir de la ejecución de sentencias dentro de un bloque evaluado.

Estructuras de Selección

Switch

```java
//Dias que faltan hasta el fin de semana
int diaSemana = 1;// 1-L, 2-M,...
switch (diaSemana) {
case 1:
    System.out.println("Lunes");
case 2:
    System.out.println("Martes");
case 3:
    System.out.println("Miercoles");
case 4:
    System.out.println("Jueves");
case 5:
    System.out.println("Viernes");
    break;
default:
    System.out.println("Fin de semana");
    break;
}
```

Estructuras de repetición

for

La sentencia for proporciona una forma de iterar sobre un determinado rango de valores. Está formada por:

- Una expresión de inicialización (se ejecuta al principio inicializando la variable de control al valor inicial)
- Una expresión de condición que debe cumplirse para que se mantenga el bucle
- Una expresión de incremento/decremento sobre la variable de control para conseguir la finalización del bucle

Estructuras de repetición

for

```
for(int i=0;i<4;i++)
{
    System.out.println("Valor incremento" + i);
}

for(int i=10;i>3;i--)
{
    System.out.println("Valor decremento" + i);
}


for( ; ; ) //Bucle infinito
{

}
```

Estructuras de repetición

while

Mediante esta instrucción se ejecuta el código entre llaves mientras la evaluación de la condición entre paréntesis sea cierta.

```
int cont=0;
while(cont<10)
{
    System.out.println(cont);
    cont++;
}
```

Estructuras de repetición

do while

A diferencia del while, mediante esta instrucción se ejecuta primero el bloque de instrucciones y luego se evalúa la condición.

```
int cont=0;
do
{
    System.out.println(cont);
    cont++;
}
while(cont<10);
```

Sentencias de salto

Java incorpora tres sentencias de salto: break, continue y return, que transfieren el control a otra parte del programa. Veremos break y continue:

Sentencias de salto. Break

La sentencia **break** (además de su uso en un switch) fuerza la finalización inmediata de un ciclo, evitando la expresión condicional y el resto de código dentro del cuerpo del ciclo. Cuando se encuentra una sentencia break dentro de un ciclo, el ciclo termina y el control del programa se transfiere a la sentencia que sigue al ciclo.

Cuando la sentencia break se utiliza dentro de un conjunto de ciclos anidados, solamente se saldrá del ciclo en el que esté situado, si es el interno, no afecta al ciclo superior.

El siguiente ejemplo calcula si un número introducido es primo o no, recorriendo los números desde el 2 hasta el anterior a dicho número. Si alguno de esos números es divisor del número introducido (resto de su división es cero) ya no será primo, y podemos abandonar el bucle.

Sentencias de salto. Break

```
Scanner teclado = new Scanner (System.in);
System.out.println("Introduce un entero positivo:");
int valor = teclado.nextInt();
int i;
boolean primo=true;

for (i=2; i< valor; i++) {    // se pueden omitir las llaves
      if (valor % i == 0) {primo = false; break; }
}
if (primo) System.out.println ("Es primo");
else System.out.println ("No es primo");
```

Sentencias de salto. Break

NOTA: El uso de break en los bucles puede ser considerado un **mal estilo de programación** ya que, entre muchas líneas de código, a veces es difícil encontrar la lógica de la condición de un bucle. Por ello, suele ser **recomendable controlar la condición** que provoca el break desde la condición de ejecución del bucle (por ejemplo, convirtiendo un for en un while).

```
Scanner teclado = new Scanner (System.in);
System.out.println("Introduce un entero positivo:");
int valor = teclado.nextInt();
int i=2;
boolean primo=true;
while (primo && i < valor ) {
    if (valor % i == 0) primo = false;
    i++;
}
if (primo) System.out.println ("Es primo");
else System.out.println ("No es primo");
```

Sentencias de salto. Continue

Algunas veces es útil forzar una nueva iteración de un ciclo sin concluir completamente el procesamiento de la iteración actual, es decir, un salto desde un punto del bloque hasta el final del ciclo. Eso lo hacemos con la sentencia **continue**.

Al igual que el caso del break no es muy aconsejable abusar de su uso.

La clase String

En el anterior tema, ***Elementos de un programa informático***, vimos los diferentes tipos de datos que podemos utilizar. Entre ellos está, para representar las cadenas de texto, la clase String.

La definición es igual a lo visto con otros tipos (tipo y nombre de la variable)

```
String cadena;
```

La asignación de valores de forma directa se realiza mediante las comillas dobles “

```
cadena="Esto es una cadena de texto";
```

La clase String

A diferencia de los tipos primitivos (int, char, boolean,...), la clase String (al igual que las Clases Envoltorio) provee de una serie de métodos, ya que en este caso no es un dato primitivo, si no que es un objeto. Algunos métodos interesantes que provee son:

- length → Indica la longitud de la cadena
- charAt(i) → Devuelve el caracter de la cadena en la posición i. **Hay que tener en cuenta que las posiciones empiezan en 0**

La clase String

```
String cadena="Esta es una prueba de cadena";
```

E	s	t	a		e	s		u	n	a		p	r	u	e	b	a		d	e		c	a	d	e	n	a
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

```
System.out.println(x: cadena.length());//28

System.out.print(c: cadena.charAt(index: 3) ); //a
```

La clase String

Con estos dos métodos podríamos, por ejemplo, recorrer la cadena de texto. Siempre teniendo en cuenta que comienza en la posición 0 (en la cadena de ejemplo, las posiciones van de 0 a 27 ya que la longitud es 28).

```
for(int i=0;i<cadena.length();i++)
{
    System.out.println(cadena.charAt(i));
}
```

La clase String

Para realizar conversiones (por ejemplo de elemento de tipo caracter a tipo String) podemos utilizar el método valueOf()

```
String caracterEnCadena=String.valueOf('A');
```

Así tendríamos en la variable *caracterEnCadena* la cadena “A”

Este método se puede usar con otras clases de tipo Envoltorio (por ejemplo Integer.valueOf(“5”) convertirá el valor “5” en un valor entero 5).

La clase String

Para las comparaciones entre Cadenas de texto e incluso caracteres es preferible no utilizar la comparación directa (==) debido a que lo que se compara con la doble igualdad es la posición de memoria en la que se guarda el valor.

Para realizar comparaciones directas podemos usar:

el método ***equals***

```
if(miCadena1.equals(miCadena2)==true)
    System.out.println("Son iguales");
else
    System.out.println("No son iguales");
```

La clase String

el método ***compareTo*** (o la variante ***compareToIgnoreCase***)

En este caso, el método devuelve un valor entero con el resultado de la comparación

- Un valor >0 si la primera cadena es mayor que la segunda
- Un valor <0 si la primera cadena es menor que la segunda
- Un valor 0 si las dos cadenas son iguales

```
if( miCadena1.compareTo(miCadena2)==0)
    System.out.println("Son iguales");
else
    System.out.println("No son iguales");
```

La clase String

Otros métodos interesantes de la clase String serían:

- Convertir a mayúsculas: cadena.toUppercase();
- Convertir a minúsculas: cadena.toLowerCase();
- Concatenar cadenas:
  - cadena1 + " " + cadena2;
  - cadena1.concat(" ").concat(cadena2);
- Obtener subcadenas (o fragmentos) de la cadena principal, indicándole un índice inicial y uno final:
  - String subcadena = cadena.substring(3,8);
  - Si se omite el segundo índice se obtendría la subcadena desde el índice indicado hasta el final: cadena.substring(3);
- Encontrar la posición en la cadena de un caracter o de una subcadena (si no se encuentra devolvería -1):
  - int pos = cadena.indexOf('x')

La clase String

Otros métodos interesantes de la clase String serían:

- Encontrar la posición en la cadena de un caracter o de una subcadena (si no se encuentra devolvería -1):
  - int pos = cadena.indexOf('x');
  - int pos = cadena.indexOf('x',posIni);
  - int pos = cadena.indexOf("texto");
  - int pos = cadena.indexOf("texto",posIni);
- Para reemplazar un caracter o una cadena dentro de otra cadena podemos usar el método replace:
  - String nuevaCadena = cadenaOriginal.replace("barco","coche");
- Para eliminar espacios en blanco podemos usar el método trim
  - String nuevaCadena = cadenaOriginal.trim();

La clase String

Un String es un objeto, y los objetos pueden no contener ningún valor (a diferencia de los tipos primitivos). Para hacer que una cadena no almacene nada le asignamos el valor **null**.
String s = null;

También podemos preguntar si una cadena es null. De hecho, en algunos casos será aconsejable hacerlo antes de llamar a algún método, ya que en caso de que la cadena sea null, se producirá una excepción y nuestro programa terminará abruptamente.

```
if (s!=null) System.out.println(s.length());
```

La clase String

String dispone de algún método que nos puede parecer que es lo mismo que preguntar por ==null pero no producen el mismo resultado:

- isEmpty () devuelve true si la cadena vacía, esto es "", sin ningún carácter, pero no es lo mismo que null.
- isBlank () devuelve true si la cadena contiene solo espacios en blanco, esto es: " ", o bien " ", etc. (también si es "", sin nada dentro devuelve true).

La clase String

Errores Frecuentes:

- Hay que comparar los String con equals(), compareTo(), compareToIgnoreCase(), pero nunca con ==.
- Los String comienzan en la posición cero, no en la 1.
- El último carácter de una cadena está en length()-1. Es erróneo cadena.charAt(cadena.length()).
- El método substring (x,y) no incluye la posición ‘y’, acaba en la anterior a ‘y’.
- Una operación sobre un String crea un nuevo String, no modifica el actual.