

LA CLASE STRING

El término *String* significa "*cadena*" (de caracteres). En muchos lenguajes, como en C, array de caracteres y *String* es lo mismo. En java el *String* se implementó como una clase propia para que fuera más fácil su manejo. Los literales *String* se representan como una tira de caracteres entre comillas dobles, por ejemplo, "hola" es un *String* y "hola\n" es otro *String*. Estuvimos utilizando literales *String* desde el principio

```
System.out.println("hola mundo");
```

Utilizaremos indistintamente el término en inglés "*String*" como su traducción "*cadena*" (ou "*cadea*").

CONSTRUCCIÓN DE CADENAS

Varias posibilidades:

1. con *new*, al igual que cualquier otro objeto.
`String cadena1 = new String("Hola");`
2. pasando como argumento al constructor una cadena ya existente
`String cadena2 = new String(cadena1);`
3. o para el caso 1 se puede utilizar la forma compacta
`String cadena1= "Hola";`
4. otras posibilidades que irán saliendo poco a poco. Echa un vistazo al API y observa que hay muchas versiones del constructor.

Ejemplo: Ejecuta el siguiente código

```
class Unidad2{
    public static void main(String args[]){
        String str1= new String("hola");
        String str2= "Adios";
        String str3=new String(str1);
        String str4=str1;

        System.out.println(str1);
        System.out.println(str2);
        System.out.println(str3);
        System.out.println(str4);
    }
}
```

Recuerda que *str1*, *str2*, *str3* y *str4* no son strings, son referencias a Strings, recuerda que una cosa es el objeto, y otra una referencia a un objeto. Como siempre, informalmente podremos hablar a veces del string *str1*, pero cuando el contexto o el rigor lo requiera hay que distinguir entre referencia y objeto. Y dicho esto, tienes que entender en el ejemplo anterior que:

- *str1* y *str4* referencian al mismo objeto *String* que contiene el valor "Hola"
- *str3* referencia a un objeto diferente al anterior, ubicado en otra posición de memoria, aunque ese objeto también contenga el valor "hola"

VISUALIZAR CON *identityHashCode()* LA REFERENCIA DE UN STRING

Por su importancia, volveremos a comentar rápidamente conceptos ya vistos

Visualizar la referencia de un objeto

Supongamos una referencia *x*, como ya sabemos, `print(x)` es equivalente a `print(x.toString())`

Observa la afirmación anterior con el siguiente ejemplo, y observa también que el método `toString()` de *MiClase* está comentado.

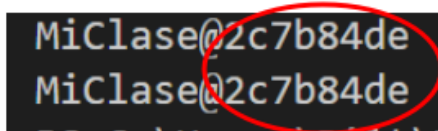
```

class MiClase {
    int x = 34;
    // public String toString(){return ""+x;}
}

class Unidad2 {
    public static void main(String[] args) {
        MiClase mc = new MiClase();
        System.out.println(mc);
        System.out.println(mc.toString());
    }
}

```

SALIDA:



→ Las referencias en hexadecimal

Por lo tanto, si una clase no tiene un `toString()` definido, java utiliza una versión por defecto que consiste en imprimir `nombredeclase@referenciaenhexadecimal`

Escribir un método `toString()`

Si escribimos un `toString()` podemos modificar el comportamiento por defecto. El trabajo habitual de un método `toString()` es devolver el estado del objeto(su valores) en un `String`, por ejemplo, en el caso anterior, visualizamos el valor del atributo `x`

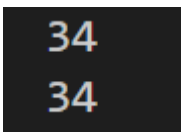
```

class MiClase {
    int x = 34;
    public String toString(){return ""+x;}
}

class Unidad2 {
    public static void main(String[] args) {
        MiClase mc = new MiClase();
        System.out.println(mc);
        System.out.println(mc.toString());
    }
}

```

SALIDA:



El método `toString()` de la clase `String` enseña el contenido del `String`.

La mayoría de las clases del API java, tienen escrito un `toString()` que enseña el estado del objeto ya no suele ser muy útil visualizar `nombredeclase@referencia`

```

class Unidad2 {
    public static void main(String[] args) {
        String s="hola";
        BigInteger bi=new BigInteger("123");
        BigDecimal bd= new BigDecimal("99.88");
        System.out.println(s+ " "+ bi+ " "+bd);
    }
}

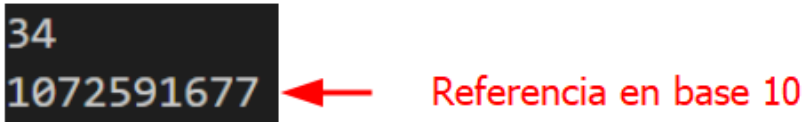
```

La utilidad `System.identityHashCode()`.

¿Si una clase tiene escrito el método `toString()` cómo puedo observar su referencia?

Con la utilidad `System.identityHashCode()`.

```
class MiClase{
    int x=34;
    public String toString(){return ""+x;}
}
class Unidad2 {
    public static void main(String[] args) {
        MiClase mc= new MiClase();
        System.out.println(mc);
        System.out.println(System.identityHashCode(mc));
        String s="hola";
        System.out.println(s);
        System.out.println(System.identityHashCode(s));
    }
}
```

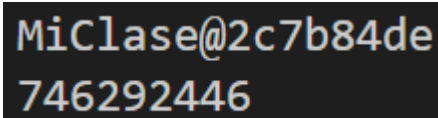


34
1072591677 ← Referencia en base 10

La referencia que imprime `identityHashCode()` es la misma que la que imprime la versión por defecto del `toString()`. La diferencia es sólo visual ya que `identityHashCode()` lo hace en base 10 y `toString()` en base 16 con `nombredeclase@` delante, pero el valor, es el mismo.

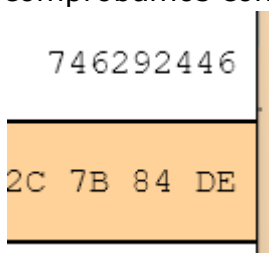
Podemos comprobar esta afirmación con una calculadora

```
class MiClase {
    int x = 34;
}
class Unidad2 {
    public static void main(String[] args) {
        MiClase mc = new MiClase();
        System.out.println(mc);
        System.out.println(System.identityHashCode(mc));
    }
}
```



MiClase@2c7b84de
746292446

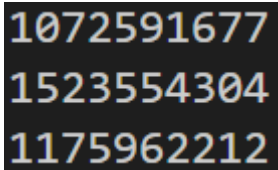
comprobamos con la calculadora



746292446
2C 7B 84 DE

Entonces, para ver la referencia que contiene una variable String, puedo utilizar esta utilidad

```
class Unidad2 {
    public static void main(String[] args) {
        String s="hola";
        BigInteger bi=new BigInteger("123");
        BigDecimal bd= new BigDecimal("99.88");
        System.out.println(System.identityHashCode(s));
        System.out.println(System.identityHashCode(bi));
        System.out.println(System.identityHashCode(bd));
    }
}
```



1072591677
1523554304
1175962212

OPERACIONES(MÉTODOS) CON CADENAS

La clase String tiene muchísimos métodos, algún ejemplo:

int length()

Devuelve la longitud de una cadena.

Ejemplo:

```
class Unidad2{
    public static void main(String args[]){
        String str1="uno";
        String str2= "unos";
        System.out.println("la cadena " +str1 + " tiene una longitud: " + str1.length());
        System.out.println("la cadena " +str2 + " tiene una longitud: " + str2.length());
    }
}
```

int compareTo(String cad) . Devuelve:

- un entero <0 si la cadena que invoca al método es menor que cad
- 0 si es igual
- un entero >0 si es mayor.

Los métodos que comparan Strings trabajan con los valores enteros Unicode de los caracteres. Para comparar dos cadenas A y B vamos comparando de posición en posición: se comparan el primer carácter de A y el primero de B según su valor unicode, si es el mismo carácter se pasa a mirar el siguiente carácter de A y B, etc. La comparación entre caracteres es tipo orden alfabético como cuando hago búsquedas en diccionario. Surgen matices, para lo que hay que tener en cuenta los valores unicode.

Ejemplo:

```
class Unidad2{
    public static void main(String args[]){
        String str1= "huevo";
        String str2= "Huevo";

        //las mayúsculas son más pequeñas que las minúsculas
        System.out.println("huevo es mayor que Huevo:"+str1.compareTo(str2));

        //huevo y huevos son iguales excepto por la s final. Es mayor la más larga
    }
}
```

```

    str2="huevos";
    System.out.println("huevo es menor que  huevos:"+ str1.compareTo(str2));

    //no tiene porque ser mayor el más largo
    str1="Huevos";
    str2="huevo";
    System.out.println("Huevos es menor que  huevo:"+ str1.compareTo(str2));

    //los números dentro de un String  no son más que caracteres unicode
    //con cadenas de igual longitud hay coincidencia con comparación aritmética
    str1="999";
    str2="123";
    System.out.println("999 es mayor que  123:"+str1.compareTo(str2));

    //pero ojo con Strings que contienen números cuando son de diferente longitud

    str1="999";
    str2="1234";
    System.out.println("999 ¿es mayor que!  1234:"+str1.compareTo(str2));

}
}

```

Para razonar los comportamiento anteriores comprueba los valores unicode en una lista unicode por ejemplo:
https://en.wikipedia.org/wiki/List_of_Unicode_characters

equals()

Decide si dos Strings son iguales por su contenido, no por su referencia de memoria.

```

class Unidad2{
    public static void main(String args[]){
        String str1= new String("hola");
        String str2= new String("Hola");
        String str3= new String("hola");
        System.out.println("Demostrar que equals con Strings trabaja por contenido");
        System.out.println("Primero Observo que son 3 objetos diferentes, cada uno con su referencia");
        System.out.println("Referencia de str1: "+System.identityHashCode(str1));
        System.out.println("Referencia de str2: "+System.identityHashCode(str2));
        System.out.println("Referencia de str3: "+System.identityHashCode(str3));
        System.out.println("Ahora constato que equals compara contenido, no referencias");
        System.out.println("str1 igual a str2?" +str1.equals(str2));
        System.out.println("str1 igual a str3?" +str1.equals(str3));
        System.out.println("str2 igual a str3?" +str1.equals(str2));
    }
}

```

```
}
}
```

el efecto de equals() también se puede hacer con compareTo(), por ejemplo:

```
System.out.println("str1 igual a str2?" +(str1.compareTo(str2)==0));
```

acceso a cada carácter individual por índice

Para trabajar con muchos métodos de la clase String es necesario saber que a cada carácter de un String se le asocia un número llamado índice relacionado con la posición del carácter en el String

Por ejemplo, analicemos los índices del String "hola"

| | | | |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| h | o | l | a |

el índice de 'h' es 0, el de 'o' 1, etc.

charAt()

le indico un índice y me devuelve el carácter en él almacenado

Ejemplo:

```
class Unidad2{
    public static void main(String args[]){
        String str= "hola mundo";
        System.out.println(str.charAt(0));
        System.out.println(str.charAt(1));
        System.out.println(str.charAt(2));
    }
}
```

substring()

nos permite extraer un substring del string original utilizando un par de índices

Ejemplo: Observa que el índice límite inferior es inclusivo y el superior exclusivo

```
class Unidad2{
    public static void main(String args[]){
        String str= "hola mundo";
        System.out.println("de 0 a 2: " +str.substring(0, 2));
        System.out.println("de 2 a 6: " +str.substring(2, 6));
    }
}
```

indexOf()

le indico un carácter o string y me devuelve un índice asociado

```
class Unidad2{
    public static void main(String args[]){

        String texto="HOLA MUNDO MI EMAIL ES YO@YO.ES";
        //primer índice que contiene M
        System.out.println(texto.indexOf('M'));
        //primer índice que contiene M a partir del índice 12
        System.out.println(texto.indexOf('M',12));
        //primer índice que contiene MA
        System.out.println(texto.indexOf("MA"));
        //primer índice que contiene MA a partir del índice 16 no existe y devuelve -1
        System.out.println(texto.indexOf("MA",16));

    }
}
```

La clase String tiene muchísimos métodos

Iremos usando nuevos métodos a medida que avancemos en el curso, en cualquier caso, cuando necesites hacer algo con Strings, como siempre, echas un vistazo al API para localizar algún método que cumpla tus expectativas. Como último ejemplo fíjate en la potencia del método `replace()`

```
class Unidad2 {  
  
    public static void main(String[] args) {  
        String cantidad="2,34";  
        System.out.println(cantidad.replace(",", "."));  
        System.out.println(cantidad.replace(",", ".0"));  
        System.out.println(cantidad.replace(",", ""));  
        System.out.println(cantidad.replace("2,3", "99"));  
    }  
}
```

EL LITERAL STRING ES UN OBJETO Y SE LE PUEDE APLICAR EL OPERADOR PUNTO

sabes que el operador punto nos permite acceder a los atributos y métodos de objetos. Hasta ahora siempre lo aplicamos a variables referencia porque no manejamos ningún objeto literal, pero por ejemplo se puede usar con los literales String. Fíjate que en el siguiente código no hay ni una variable referencia

```
class Unidad2 {  
  
    public static void main(String[] args) {  
        System.out.println("hola mundo".length());  
        System.out.println("hola mundo".charAt(0));  
        System.out.println("hola mundo".charAt(9));  
        System.out.println("hola mundo".substring(2, 7));  
        System.out.println("huevos".compareTo("Huevos"));  
        System.out.println("999".compareTo("123"));  
        System.out.println("999".compareTo("1234"));  
    }  
}
```

STRINGS Y LOS OPERADORES == Y !=

Los operadores `==` y `!=` los utilizamos para comparar expresiones de tipo primitivo

por ejemplo

```
int a=3;  
a==5; //es false  
a==3; //es true  
8.9==9.0; // es false  
false!=true; // es true
```

etc.

también los podemos usar para comparar referencias pero no para comparar el contenido de los objetos. Es decir, con `==` comparamos el valor de las referencias, no el valor de los objetos a los que apuntan dichas referencias. Para comparar el contenido de dos objetos String se usa `equals()`.

El método `equals()` aplicado a Strings **no** dice si dos objetos son el mismo, dicen si tienen el mismo contenido.

por ejemplo

```
class Unidad2 {  
  
    public static void main(String[] args) {  
        String s1=new String("hola");  
        String s2=new String("hola");
```

```

        if(s1==s2)
            System.out.println("s1 y s2 referencian al mismo objeto");
        else
            System.out.println("s1 y s2 NO referencian al mismo objeto");
        if(s1.equals(s2))
            System.out.println("s1 y s2 contiene el mismo texto");
        else
            System.out.println("s1 y s2 NO contienen el mismo texto");
    }
}

```

Con `identityHashCode()` también observamos claramente que `s1` y `s2` referencia a objetos diferentes.

```

class Unidad2 {
    public static void main(String[] args) {
        String s1=new String("hola");
        String s2=new String("hola");
        System.out.println(System.identityHashCode(s1));
        System.out.println(System.identityHashCode(s2));
    }
}

```

Una consideración especial para los objetos literales String

Los literales String en java por razones de eficiencia son objetos especiales de forma que una vez que se crea uno, ya que es inmutable, cuando se vuelve usar el mismo literal no se crea en memoria uno nuevo. Por ejemplo, aparece en nuestro programa varias veces el literal String "hola", pero sólo se crea un objeto String "hola" y a partir de ese momento todos los literales "hola" de mi programa tienen la misma referencia ya que realmente es el mismo objeto.

```

class Unidad2 {
    public static void main(String[] args) {
        String s1="hola";
        String s2="hola";
        if(s1==s2)
            System.out.println("s1 y s2 referencian al mismo objeto");
        else
            System.out.println("s1 y s2 NO referencian al mismo objeto");
        if(s1.equals(s2))
            System.out.println("s1 y s2 contiene el mismo texto");
        else
            System.out.println("s1 y s2 NO contienen el mismo texto");
    }
}

```

Ahora `s1==s2` es true porque java sólo creó una vez el literal "hola". El primer "hola" y el segundo "hola" son el mismo objeto

También puedo usar `System.identityHashCode()` para demostrar que los literales sólo se crean una vez.

```

class Unidad2 {
    public static void main(String[] args) {
        String s1="hola";
        String s2="hola";
        System.out.println(System.identityHashCode(s1));
        System.out.println(System.identityHashCode(s2));
        System.out.println(System.identityHashCode("hola"));
    }
}

```


en cambio ahora por efecto del new se crearon 2 objetos adicionales

```
class Unidad2 {
    public static void main(String[] args) {
        String s1=new String("hola");
        String s2=new String("hola");
        System.out.println(System.identityHashCode(s1));
        System.out.println(System.identityHashCode(s2));
        System.out.println(System.identityHashCode("hola"));
    }
}
```

CONCLUSIÓN: Y vista esta curiosidad, simplemente quedarnos con la idea de que usar literales string es eficiente y que respecto a las comparaciones trabajando con Strings lo que nos interesa normalmente es comparar por contenido y por tanto lo mejor es utilizar siempre equals()

STRINGS Y EL OPERADOR +

Ya vimos un montón de ejemplos con este operador, recuerda que su precedencia es de izquierda a derecha y que funciona aritméticamente cuando sus dos operandos son números, pero, con que uno sólo de los operandos sea un String pasa a funcionar como concatenador promocionando automáticamente los operandos que no son String a String. Ejemplo:

```
System.out.println(4+1+"-"+4+1);
```

Imprime la cadena 5-41

UN OBJETO STRING NO PUEDE MODIFICARSE, ES INMUTABLE

Ya manejamos anteriormente otros objetos inmutables:BigDecimal y BigInteger. Una vez que se crea un string en memoria, no puede modificarse. Si queremos cambiar el contenido de un string, lo que podemos hacer es crear uno nuevo a partir del viejo con los cambios deseados. Recuerda que al hacer esta operación, el string 'viejo' si no tiene asociada ninguna variable referencia, será eliminado de memoria por el recolector de basura. Esta característica 'chocante' de los String hace que su implementación interna sea más eficiente. En cualquier caso también hay en Java objetos que almacenan cadenas de caracteres y que son de 'lectura y escritura' como los objetos de la clase StringBuffer y StringBuilder.

Ejemplo: Usando la clase String, queremos modificar una cadena de caracteres que inicialmente vale "soy una cabritilla abandonada" de forma que queremos añadirle un punto y final "."

Realmente crearé un "nuevo" String a partir del "viejo" NO MODIFICO EL VIEJO

```
class Unidad2{
    public static void main(String args[]){
        String s1="soy una cabritilla abandonada";
        String s2;

        //añado un punto a s2
        s2=s1+".";
        System.out.println(s1);
        System.out.println(s2);
    }
}
```

```
}
}
```

El programa anterior podría hacerse con una única variable Referencia String

```
class Unidad2{
    public static void main(String args[]){
        String s1="soy una cabritilla abandonada";

        System.out.println(s1);
        //añado un punto
        s1=s1+ ".";
        System.out.println(s1);
    }
}
```

Ejemplo: Observamos inmutabilidad usando el método trim(). Observa como indica *Returns a Copy*

trim

```
public String trim()
```

Returns a copy of the string, with leading and trailing whitespace omitted.

trim() quita los espacios de delante y del final de los strings

```
class Unidad2{
    public static void main(String args[]){
        String str1= " ho la ";
        System.out.println(str1+"adios");
        System.out.println(str1.trim()+"adios");
        System.out.println(str1+"adios");
    }
}
```

Podemos comprobar lo anterior usando identityHashCode()

```
class Unidad2 {
    public static void main(String args[]) {
        String str1 = " ho la ";
        System.out.println(System.identityHashCode(str1));
        String str2 = str1.trim();
        System.out.println(System.identityHashCode(str1));
        System.out.println(System.identityHashCode(str2));
    }
}
```

observa que str1.trim() no cambia el String referenciado por str1 si no que genera uno nuevo tal y como demuestra el código anterior ya que en el último println() se vuelve a obtener el resultado inicial. ES DECIR: No hay en la clase String ningún método que actualice el String, lo que hay son métodos que generan un nuevo String. Similarmente recuerda que por ejemplo add() de BigInteger/BigDecimal devolvía un nuevo objeto. BigInteger y BigDecimal también son INMUTABLES.

otro ejemplo que demuestra inmutabilidad con método replace()

```
class Unidad2 {
    public static void main(String[] args) {
        String s1="Bueno ,bonito y barato";
        String s2=s1.replace("b", "v");
```

```

        System.out.println(s2);
        System.out.println(s1);
        s1=s1.replace(" ","");
        System.out.println(s1);
    }
}

```

EJERCICIO U2_B11_E1: Recuerda que la clase Scanner usa el concepto de delimitador para separar los tokens entre sí. Por defecto Scanner usa el delimitador *whitespace*(blanco+tab+salto línea) pero podemos cambiar esto con el método `useDelimiter()`. Observa en este ejemplo el uso de `useDelimiter()`

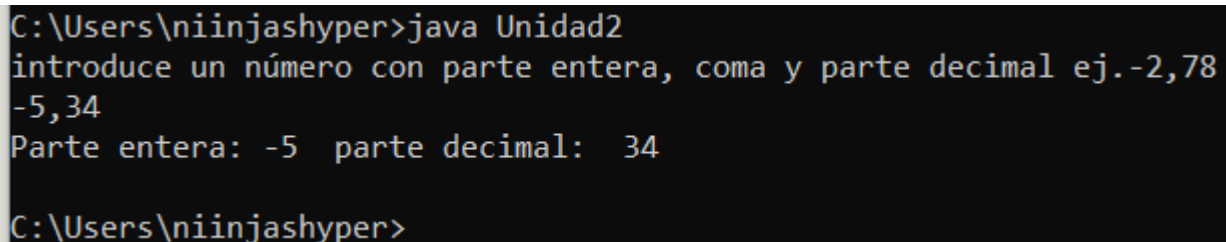
```

import java.util.Scanner;
public class Unidad2{
    public static void main(String[] args){
        String s="HOLA,ADIOS,CHAO";
        Scanner sc = new Scanner(s);
        sc.useDelimiter(",");
        System.out.println(sc.next() + " " + sc.next()+ " "+sc.next());

    }
}

```

SE PIDE: Escribir un programa que lea del teclado un string que contenga un número con coma decimal y obtener su parte entera y decimal usando un scanner (no pasando a double).



```

C:\Users\niinjashyper>java Unidad2
introduce un número con parte entera, coma y parte decimal ej.-2,78
-5,34
Parte entera: -5  parte decimal:  34

C:\Users\niinjashyper>

```

Hacerlo de 2 formas:

1. Usando Console para leer de teclado y el string leído luego lo analizas con un objeto Scanner
2. Usando Scanner para leer de teclado(por lo tanto manejas dos objetos Scanner en el programa, uno para leer de teclado y otro para analizar el texto)

EJERCICIO U2_B11_E2: vuelve a resolver el ejercicio anterior pero ahora basándote el método `substring()` teniendo en cuenta que el usuario siempre va a teclear

exactamente dos decimales (y cualquier parte entera).

EJERCICIO U2_B11_E3: Se pide mejorar el main() que pide un número complejo al usuario de forma que admita introducirlo en su forma binomial

El código que utilizamos anteriormente era más o menos:

```
import java.util.Scanner;
class Complejo{
    private double real;
    private double imag;
    Complejo(){
        this.real=0;
        this.imag=0;
        //o mejor
        //this(0,0);
    }
    Complejo(double real,double imag){
        this.real=real;
        this.imag=imag;
    }
    void setReal(double real){
        this.real=real;
    }
    double getReal(){
        return this.real;
    }
    void setImag(double imag){
        this.imag=imag;
    }
    double getImag(){
        return this.imag;
    }
    public String toString(){
        //return this.real + " + " + this.imaginaria+"i";//falla si la parte imaginaria es negativa
        return this.real +(this.imag<0?" ":"")+this.imag+"i";
    }
    Complejo sumar(Complejo b){
        return new Complejo(this.real+b.real,this.imag+b.imag);
    }
}

class Unidad2{
    public static void main(String[] args){
        Scanner sc =new Scanner(System.in);
        double parteReal;
        double parteImaginaria;

        //creamos número a
        System.out.println("Número a");
        System.out.print("\tparte real:");
        parteReal=sc.nextDouble();
        System.out.print("\tparte imaginaria:");
        parteImaginaria=sc.nextDouble();
        Complejo a= new Complejo(parteReal,parteImaginaria);

        //creamos número b
        System.out.println("Número b");
        System.out.print("\tparte real:");
        parteReal=sc.nextDouble();
        System.out.print("\tparte imaginaria:");
        parteImaginaria=sc.nextDouble();
        Complejo b= new Complejo(parteReal,parteImaginaria);
        //probar suma a y b
        Complejo c= a.sumar(b);
        System.out.println("\nsuma de a y b: "+ c);
    }
}
```

ahora debes utilizar la capacidad de la clase Scanner para escanear double con nextDouble() y conseguir algo similar a lo siguiente:

```
L:\Programacion\complejo>java Unidad2
Introduce números complejos en formato binomial, por ejemplo -3 +4i
Observa restricciones: signos pegados a numeros y al menos un espacio entre real
e imaginaria
    Numero a: -3 +4i
    Numero b: 2 -2i

suma de a y b: -1.0 + 2.0i
```

Observa que con lo que sabemos imponemos al usuario un formato concreto de entrada ya que el programa es "delicado" por ejemplo si no dejamos espacio en blanco en el medio rompe

```
L:\Programacion\complejo>java Unidad2
Introduce números complejos en formato binomial, por ejemplo -3 +4i
Observa restricciones: signos pegados a numeros y al menos un espacio entre real
e imaginaria
    Numero a: 3+3i
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Unknown Source)
```

Igual que en el ejercicio 1 puedes utilizar 2 objetos Scanner o bien un objeto Console y otro Scanner

EJERCICIO U2_B11_E4:

introducimos por teclado 3 líneas de venta. Cada línea de venta consta de:

- precio del artículo que tiene exactamente 5 caracteres incluyendo el punto decimal si se usa
- un espacio en blanco
- número de unidades del artículo que se escribe exactamente con dos caracteres, puede haber un cero a la izquierda
- un espacio en blanco
- una descripción del artículo

Tras introducir las tres líneas se imprime un comprobante de cada línea y el importante total de la venta (ojo, a lo mejor en tu sistema tienes que teclear los double con coma)

```
L:\Programacion>java Unidad2
datos línea de venta 1: 12.05 02 papel
datos línea de venta 2: 00002 10 gomas
datos línea de venta 3: 100.5 01 puntero laser
    2 unidades del artículo papel a 12.05 la unidad
    10 unidades del artículo gomas a 2.0 la unidad
    1 unidades del artículo puntero laser a 100.5 la unidad
Importe total: 144.6
```

Resuelve el problema de dos formas

1. leyendo cada línea y procesándola con substring. Aquí es importante tener en cuenta la estructura fija de cada línea de venta

2. obteniendo los campos directamente con los métodos next de Scanner

Otras clases para trabajar con cadenas de caracteres.

Hay varias. Las otras dos más ampliamente usadas son StringBuffer y StringBuilder que se diferencian de la clase String porque son MUTABLES. Por otro lado, simplificando un poco, StringBuilder es una mejora de StringBuffer así que veremos un ejemplo de StringBuilder.

StringBuilder tiene un método setCharAt

setCharAt devuelve void y al usarlo por tanto es fácil enter que estamos haciendo una modificación sobre el objeto this

setCharAt

```
public void setCharAt(int index,  
                     char ch)
```

Demostramos que tras hacer un setChar() el objeto es el mismo

```
class Unidad2 {  
    public static void main(String args[]) {  
        StringBuilder strb = new StringBuilder("hola");  
        System.out.println(strb);  
        System.out.println(System.identityHashCode(strb));  
        strb.setCharAt(0, 'b');  
        strb.setCharAt(2, 't');  
        System.out.println(strb);  
        System.out.println(System.identityHashCode(strb));  
    }  
}
```