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. Las 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);
  }
}
```

Observa 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. 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"

OPERACIONES(MÉTODOS) CON CADENAS

La clase String tiene muchísimos métodos, algún ejemplo:

int length()

Devuelve la longitud de una cadena. Observa que para la clase String length es un método y por tanto lleva paréntesis, en cambio, en los arrays, length es un atributo y

por tanto no lleva paréntesis.
int compareTo(String cad) . Devuelve:

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

Recuerda que para comparar dos cadenas A y B, 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.

Ejemplo: uso de length() y compareTo()

```
class Unidad2{
  public static void main(String args[]){
     String str1= new String("hola");
     String str2= "Adios";
     System.out.println("la cadena " +str1 + " tiene una longitud: " + str1.length());
     System.out.println("la cadena " +str2 + " tiene una longitud: " + str2.length());
      if(str1.compareTo(str2)!=0)
         System.out.println("la cadena " +str1 + " es diferente a " + str2);
  }
}
```

Ejemplo: uso de equals()

```
class Unidad2{
  public static void main(String args[]){
     String str1= "hola";
     String str2= "Hola";
     String str3= "hola";
     System.out.println("str1 = str2?" +str1.equals(str2));
     System.out.println("str1 = str3?" +str1.equals(str3));
     System.out.println("str2 = str3?" +str1.equals(str2));
  }
}
```

Ejemplo: uso de substring()
Observa que el 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));
    }
}
```

Ejemplo: uso de charAt()

```
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));
    }
}
```

La clase String tiene muchísimos métodos. cuando necesites hacer algo, como siempre, echas un vistazo al API haber si hay algún método que cumpla tus expectativas.

STRINGS Y LOS OPERADORES == Y !=

Los operadores == y != se utilizan 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 objetos. 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 contien el mismo texto");
      else
         System.out.println("s1 y s2 NO contienen el mismo texto");

   }

}
```

un caso especial es cuando asignamos a una referencia un literal String directamente, es decir, sin new. 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, aparace 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 contien 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

Y vista esta curiosidad, trabajando con Strings lo que nos interesa normalmente es comparar por contenido y por tanto lo mejor es utilizar siempre equals()

REFLEXIONANDO SOBRE LA CREACIÓN DE OBJETOS STRING

¿String s="hola"; es exactamente lo mismo que String s =new String("hola");?

Hay un matiz que los diferencia.
Con String s="hola";
la dirección del objeto literal string "hola" se copia en s

Con String s=new String("hola");
se copia el string "hola" en un nuevo objeto String, por tanto la dirección del objeto literal string "hola" es diferente que la dirección que contiene s

Podemos constatar esto con el siguiente ejemplo observando que sólo es cierto el primer if

```
class Unidad2 {
   public static void main(String[] args) {
      String s1="hola";
      String s2="hola";
      String s3= new String("hola");
      if(s1==s2)
         System.out.println("s1 y s2 referencian al mismo objeto");
      if(s1==s3)
         System.out.println("s1 y s3 referencian al mismo objeto");
      if(s2==s3)
         System.out.println("s3 y s3 referencian al mismo objeto");
   }
}
```

otro ejemplo del que debemos entender el resultado

```
class Unidad2 {
   public static void main(String[] args) {
      String s1=new String("hola");
      if(s1 == "hola" )
         System.out.println("s1 y literal hola tienen misma referencia");
      if(s1.equals("hola" ))
         System.out.println("s1 y literal hola tienen mismo contenido");
   }
 }
```

Pero a efectos prácticos, como lo que nos interesa de los String es el contenido y como además son inmutables, podemos considerar las instrucciones anteriores equivalentes.

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

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 cadena de 'lectura y escritura', son los objetos de la clase StringBuffer y StringBuilder, pero en general, es más usada la clase String.

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 "."

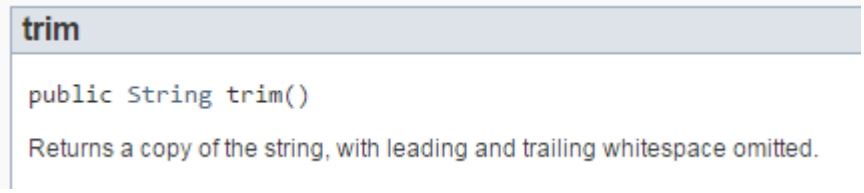
Realmente crearé un "nuevo" String a partir del "viejo" NO MODIFICO EL VIEJO

```
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()



trim() quitá 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");
  }
```

```
}
```

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.

EJERCICIO 1: 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: usando useDelimiter se pide conseguir el siguiente efecto

```
L:\Programacion>java Unidad2
introduce un número con parte entera, coma y parte decimal ej.-2,7
56,78
Parte entera: 56  parte decimal:  78
```

EJERCICIO 2: 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 3: Se pide mejorar el main() que pide un número complejo al usuario de forma que admita introducirlo en su forma binomial

El main() antiguo era el siguiente:

```
import java.io.Console;
class Unidad2{
        public static void main(String[] args){
                Console teclado=System.console();
                String parteReal;
                String parteImaginaria;
```

```
                //creamos número a
                System.out.println("Número a");
                parteReal=teclado.readLine("\tparte real:");
                parteImaginaria=teclado.readLine("\tparte imaginaria:");
                Complejo a= new Complejo(Double.parseDouble(parteReal),Double.parseDouble(parteImaginaria));

                //creamos número b
                System.out.println("Número b");
                parteReal=teclado.readLine("\tparte real:");
                parteImaginaria=teclado.readLine("\tparte imaginaria:");
                Complejo b= new Complejo(Double.parseDouble(parteReal),Double.parseDouble(parteImaginaria));

                //probar suma a y b
                a.sumar(b);
                String suma=a.convertirAString();
                System.out.println("\nsuma de a y b: "+ suma);
        }
}
```

ahora utiliza la capacidad de la clase Scanner para escanear double con nextDouble() para 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)
```