6.B. Cadenas de caracteres.

1. Cadenas de caracteres.

1.3. Operaciones avanzadas con cadenas de caracteres (III).

¿Cómo comprobarías que la cadena "3" es mayor que 0? No puedes comparar directamente una cadena con un número, así que necesitarás aprender cómo convertir cadenas que contienen números a tipos de datos numéricos (`int`, `short`, `long`, `float` o `double`). Esta es una operación habitual en todos los lenguajes de programación, y Java, para este propósito, ofrece los métodos `valueOf`, existentes en todas las clases envoltorio descendientes de la clase `Number: Integer, Long, Short, Float` y `Double`.

Veamos un ejemplo de su uso para un número de doble precisión, para el resto de las clases es similar:

```
String c="1234.5678";
```

```
double n;
```

```
try {
```

```
    n=Double.valueOf(c).doubleValue();
```

```
} catch (NumberFormatException e)
```

```
{  /* Código a ejecutar si no se puede convertir */  }
```

Fíjate en el código anterior, en él puedes comprobar cómo la cadena c contiene en su interior un número, pero escrito con dígitos numéricos (caracteres). El código escrito esta destinado a transformar la cadena en número, usando el método `valueOf`. Este método lanzará la excepción `NumberFormatException` si no consigue convertir el texto a número. De momento no te preocupes demasiado acerca de este nuevo concepto, aunque es probable que ya te haya sucedido; más adelante dedicaremos todo un tema al tratamiento de excepciones en Java. En el siguiente código, tienes un ejemplo más completo, donde aparecen también ejemplos para las otros tipos numéricos:

```
import javax.swing.JOptionPane;

/**
 * Ejemplo en el que se muestra la conversión de varias cadenas de texto, que
 * contienen números, a números.
 * @author Salvador Romero Villegas
 */

public class EjemplosConversionStringANumero {

   boolean operacionCancelada;

   /**
    * Constructor de la clase.
    */
   public EjemplosConversionStringANumero() {
      setOperacionCancelada(false);
   }

   /**
```

```
 * Método que permite comprobar si la última operación tipo Pedir ha sido
 * cancelada.
 * @return true si la última operación realizada ha sido cancelada, false
 * en otro caso.
 */
public boolean isOperacionCancelada() {
    return operacionCancelada;
}

/**
 * Método que permite cambiar el estado de la variable privada
 * operacionCancelada. Este método es privado y solo debe usarse desde
 * un método propio de esta clase.
 * @param operacionCancelada True o false, el nuevo estado para la variable.
 */
private void setOperacionCancelada(boolean operacionCancelada) {
    this.operacionCancelada = operacionCancelada;
}

/**
 * Clase que pide al usuario que introduzca un número. El número esperado
 * es un número de doble precisión, en cualquiera de sus formatos. Admitirá
 * números como: 2E10 (2*10^10); 2,45; etc.
 * @param titulo
 * @param mensaje
 * @return
 */
public double PedirNumeroDouble(String titulo, String mensaje) {
    double d = 0;
    setOperacionCancelada(false);
    boolean NumeroValido = false;
    do {

        String s = (String) JOptionPane.showInputDialog(null, mensaje,
                titulo, JOptionPane.PLAIN_MESSAGE, null, null, "");

        if (s != null) {
            try {
                d = Double.valueOf(s).doubleValue();
```

```
                NumeroValido = true;
            } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(null, "El número introducido no es válido.", "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
        else { NumeroValido=true; // Cancelado
            setOperacionCancelada(true);
        }
    } while (!NumeroValido);
    return d;
}

/**
 * Clase que pide al usuario que introduzca un número. El número esperado
 * es un número de precisión sencilla, en cualquiera de sus formatos.
 * @param titulo
 * @param mensaje
 * @return
 */
public float PedirNumeroFloat (String titulo, String mensaje) {
    float d = 0;
    setOperacionCancelada(false);
    boolean NumeroValido = false;
    do {

        String s = (String) JOptionPane.showInputDialog(null, mensaje,
                titulo, JOptionPane.PLAIN_MESSAGE, null, null, "");

        if (s != null) {
            try {
                d = Float.valueOf(s).floatValue();
                NumeroValido = true;
            } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(null, "El número introducido no es válido.", "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
        else { NumeroValido=true; // Cancelado
            setOperacionCancelada(true);
        }
```

```
    } while (!NumeroValido);
    return d;
}

/**
 * Clase que pide al usuario que introduzca un número. El número esperado
 * es un número entero.
 * @param titulo
 * @param mensaje
 * @return
 */
public int PedirNumeroInteger (String titulo, String mensaje) {
    int d = 0;
    setOperacionCancelada(false);
    boolean NumeroValido = false;
    do {

        String s = (String) JOptionPane.showInputDialog(null, mensaje,
                titulo, JOptionPane.PLAIN_MESSAGE, null, null, "");

        if (s != null) {
            try {
                d = Integer.valueOf(s).intValue();
                NumeroValido = true;
            } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(null, "El número introducido no es válido.", "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
        else { NumeroValido=true; // Cancelado
              setOperacionCancelada(true);
        }
    } while (!NumeroValido);
    return d;
}

/**
 * Clase que pide al usuario que introduzca un número. El número esperado
 * es un número entero.
 * @param titulo
```

```
 * @param mensaje
 * @return
 */
public long PedirNumeroLong (String titulo, String mensaje) {
   long d = 0;
   setOperacionCancelada(false);
   boolean NumeroValido = false;
   do {

      String s = (String) JOptionPane.showInputDialog(null, mensaje,
            titulo, JOptionPane.PLAIN_MESSAGE, null, null, "");

      if (s != null) {
         try {
            d = Long.valueOf(s).longValue();
            NumeroValido = true;
         } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(null, "El número introducido no es válido.", "Error", JOptionPane.ERROR_MESSAGE);
         }
      }
      else { NumeroValido=true; // Cancelado
           setOperacionCancelada(true);
      }
   } while (!NumeroValido);
   return d;
}

 /**
 * Clase que pide al usuario que introduzca un número. El número esperado
 * es un número entero corto.
 * @param titulo
 * @param mensaje
 * @return
 */
public short PedirNumeroShort (String titulo, String mensaje) {
   short d = 0;
   setOperacionCancelada(false);
   boolean NumeroValido = false;
   do {
```

```
        String s = (String) JOptionPane.showInputDialog(null, mensaje,
                titulo, JOptionPane.PLAIN_MESSAGE, null, null, "");

        if (s != null) {
           try {
              d = Short.valueOf(s).shortValue();
              NumeroValido = true;
           } catch (NumberFormatException e) {
              JOptionPane.showMessageDialog(null, "El número introducido no es válido.", "Error", JOptionPane.ERROR_MESSAGE);
           }
        }
        else { NumeroValido=true; // Cancelado
              setOperacionCancelada(true);
        }
     } while (!NumeroValido);
     return d;
  }
}
```

Seguro que ya te vas familiarizando con este embrollo y encontrarás interesante todas estas operaciones. Ahora te planteamos otro reto: imagina que tienes que mostrar el precio de un producto por pantalla. Generalmente, si un producto vale, por ejemplo 3,3 euros, el precio se debe mostrar como "`3,30 €`", es decir, se le añade un cero extra al final para mostrar las centésimas. Con lo que sabemos hasta ahora, usando la concatenación en Java, podemos conseguir que una cadena se concatene a un número flotante, pero el resultado no será el esperado. Prueba el siguiente código:

```
float precio=3.3f;
```

```
System.out.println("El precio es: "+precio+"€");
```

Si has probado el código anterior, habrás comprobado que el resultado no muestra "`3,30 €`" sino que muestra "`3,3 €`". ¿Cómo lo solucionamos? Podríamos dedicar bastantes líneas de código hasta conseguir algo que realmente funcione, pero no es necesario, dado que Java y otros lenguajes de programación (como C), disponen de lo que se denomina formateado de cadenas. En Java podemos "formatear" cadenas a través del método estático `format` disponible en el objeto `String`. Este método permite crear una cadena proyectando los argumentos en un formato específico de salida. Lo mejor es verlo con un ejemplo, veamos cuál sería la solución al problema planteado antes:

```
float precio=3.3f;
```

```
String salida=string.format ("El precio es: %.2f €", precio));
```

```
System.out.println(salida);
```

El formato de salida, también denominado "cadena de formato", es el primer argumento del método `format`. La variable `precio`, situada como segundo argumento, es la variable que se proyectará en la salida siguiendo un formato concreto. Seguro que te preguntarás, ¿qué es "`%.2f`"? Pues es un especificador de formato, e indica cómo se deben formatear o proyectar los argumentos que hay después de la cadena de formato en el método `format`.

Debes conocer

Es necesario que conozcas bien el método format y los especificadores de formato. Por ese motivo, te pedimos que estudies el siguiente anexo:

Anexo I.- Formateado de cadenas en Java.

Sintaxis de las cadenas de formato y uso del método `format`.

En Java, el método estático `format` de la clase `String` permite formatear los datos que se muestran al usuario o la usuaria de la aplicación. El método `format` tiene los siguientes argumentos:

- Cadena de formato. Cadena que especifica cómo será el formato de salida, en ella se mezclará texto normal con especificadores de formato, que indicarán cómo se debe formatear los datos.

- Lista de argumentos. Variables que contienen los datos cuyos datos se formatearán. Tiene que haber tantos argumentos como especificadores de formato haya en la cadena de formato.

Los especificadores de formato comienzan siempre por "`%`", es lo que se denomina un carácter de escape (carácter que sirve para indicar que lo que hay a continuación no es texto normal, sino algo especial). El especificador de formato debe llevar como mínimo el símbolo "`%`" y un carácter que indica la conversión a realizar, por ejemplo "`%d`".

La conversión se indica con un simple carácter, y señala al método `format` cómo debe ser formateado el argumento. Dependiendo del tipo de dato podemos usar unas conversiones u otras. Veamos las conversiones más utilizadas:

Listado de conversiones más utillizada y ejemplos.				
Tipo de conversión	**Especificación de formato**	**Tipos de datos aplicables**	**Ejemplo**	**Resultado del ejemplo**

Valor lógico o booleano.	"`%b`" o "`%B`"	`Boolean` (cuando se usan otros tipos de datos siempre lo formateará escribiendo `true`).	`boolean b=true;`
			`String d= String.format("Resultado: %b", b);`
			`System.out.println (d);`
				Resultado: `true`

Cadena de caracteres.	"`%s`" o "`%S`"	Cualquiera, se convertirá el objeto a cadena si es posible (invocando el método `toString`).	`String cad="hola mundo";`
			`String d= String.format("Resultado: %s", cad);`
			`System.out.println (d);`
				Resultado: `hola mundo`

Entero decimal	"`%d`"	Un tipo de dato entero.	`int i=10;`
			`String d= String.format("Resultado: %d", i);`
			`System.out.println (d);`
				Resultado: `10`

Número en notación científica	"`%e`" o "`%E`"	Flotantes simples o dobles.	`double i=10.5;`
			`String d= String.format("Resultado: %E", i);`
			`System.out.println (d);`
				Resultado: `1.050000E+01`

Número decimal	"`%f`"	Flotantes simples o dobles.	`float i=10.5f;`
			`String d= String.format("Resultado: %f", i);`
			`System.out.println (d);`
				Resultado: `10,500000`

Número en notación científica o decimal (lo más corto)	"`%g`" o "`%G`"	Flotantes simples o dobles. El número se mostrará como decimal o en notación científica dependiendo de lo que sea más corto.	`double i=10.5;`
			`String d= String.format("Resultado: %g", i);`
			`System.out.println (d);`
				Resultado: `10.5000`

Ahora que ya hemos visto alguna de las conversiones existentes (las más importantes), veamos algunos modificadores que se le pueden aplicar a las conversiones, para ajustar como queremos que sea la salida. Los modificadores se sitúan entre el carácter de escape ("%") y la letra que indica el tipo de conversión (d, f, g, etc.).

Podemos especificar, por ejemplo, el número de caracteres que tendrá como mínimo la salida de una conversión. Si el dato mostrado no llega a ese ancho en caracteres, se rellenará con espacios (salvo que se especifique lo contrario):

```
%[Ancho]Conversión
```

El hecho de que esté entre corchetes significa que es opcional. Si queremos por ejemplo que la salida genere al menos 5 caracteres (poniendo espacios delante) podríamos ponerlo así:

```
String.format ("%5d",10);
```

Se mostrará el "10" pero también se añadirán 3 espacios delante para rellenar. Este tipo de modificador se puede usar con cualquier conversión.

Cuando se trata de conversiones de tipo numéricas con decimales, solo para tipos de datos que admitan decimales, podemos indicar también la precisión, que será el número de decimales mínimos que se mostrarán:

```
%[Ancho][.Precisión]Conversión
```

Como puedes ver, tanto el ancho como la precisión van entre corchetes, los corchetes no hay que ponerlos, solo indican que son modificaciones opcionales. Si queremos, por ejemplo, que la salida genere 3 decimales como mínimo, podremos ponerlo así:

```
String.format ("%.3f",4.2f);
```

Como el número indicado como parámetro solo tiene un decimal, el resultado se completará con ceros por la derecha, generando una cadena como la siguiente: "4,200".

Una cadena de formato puede contener varios especificadores de formato y varios argumentos. Veamos un ejemplo de una cadena con varios especificadores de formato:

```
String np="Lavadora";
```

```
int u=10;
```

```
float ppu = 302.4f;
```

```
float p=u*ppu;
```

```
String output=String.format("Producto: %s; Unidades: %d; Precio por unidad: %.2f €; Total: %.2f €", np, u, ppu, p);
```

```
System.out.println(output);
```

Cuando el orden de los argumentos es un poco complicado, porque se reutilizan varias veces en la cadena de formato los mismos argumentos, se puede recurrir a los índices de argumento. Se trata de especificar la posición del argumento a utilizar, indicando la posición del argumento (el primer argumento sería el 1 y no el 0) seguido por el símbolo del dólar ("$"). El índice se ubicaría al comienzo del especificador de formato, después del porcentaje, por ejemplo:

```
int i=10;
```

```
int j=20;
```

```
String d=String.format("%1$d multiplicado por %2$d (%1$dx%2$d) es %3$d",i,j,i*j);
```

```
System.out.println(d);
```

El ejemplo anterior mostraría por pantalla la cadena "10 multiplicado por 20 (10x20) es 200". Los índices de argumento se pueden usar con todas las conversiones, y es compatible con otros modificadores de formato (incluida la precisión).

Para saber más

Si quieres profundizar en los especificadores de formato puedes acceder a la siguiente página (en inglés), donde encontrarás información adicional acerca de la sintaxis de los especificadores de formato en Java:

Sintaxis de los especificadores de formato.