UT 6 : CLASES CONTENEDORAS - WRAPPERS

En Java existen una serie de clases predefinidas equivalentes a los tipos primitivos denominadas wrappers, clases contenedoras, clases envolventes o envoltorios. La principal diferencia entre un tipo primitivo y un wrapper es que este último es una clase. Cuando trabajamos con wrappers estamos trabajando con objetos mientras que cuanto trabajamos con tipos primitivos obviamente no.

Debemos tener en cuenta que es sensiblemente distinto trabajar con un objeto que con un tipo primitivo, en especial cuando le pasamos una variable a un método como argumento o parámetro y es de un tipo primitivo se le pasa siempre por valor, mientras que cuando le pasamos un wrapper , al ser un objeto, se lo estamos pasando por referencia, luego las modificaciones que se pudieran efectuar sobre el mismo permanecen.

El identificador de cada una de estas clases es el mismo que el del tipo primitivo correspondiente pero con la letra inicial en mayúsculas (salvo int - Integer y char - Character). Cada una de estas clases declara un conjunto de métodos de gran utilidad.

Tipo primitivo	**Clase contenedora**
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

El uso de estas clases puede ser especialmente interesante para realizar determinadas operaciones mediante los métodos que implementan.

LA CLASE CHARACTER

La mayoría de las veces, si se usa un valor de carácter, se usas el tipo primitivo char. Por ejemplo:

```
char ch = 'a';
char uniChar = '\u039A'; // Unicode para mayúscula Griego omega
char[] charArray ={ 'a', 'b', 'c', 'd', 'e' }; // un array de chars
```

Hay ocasiones, sin embargo, en que necesita usar un char como un objeto—por ejemplo, como un argumento de un método donde se espera un objeto. Java provee una clase *de envoltura o wrapper* para este propósito. Un objeto de tipo Character contiene un único campo, cuyo tipo es char. Esta clase Character también ofrece un número de métodos de clase (estáticos) útiles para manipular caracteres.

Se puede crear un objeto Character con el constructor Character:

```
Character ch = new Character('a');
```

La siguiente tabla lista algunos de los métodos más útiles en la clase Character, pero no están todos.

Métodos Útiles en la Clase `Character`

Método	Descripción
`boolean isLetter(char ch)` `boolean isDigit(char ch)`	Determina si el valor char especificado es una letra o un dígito, respectivamente.
`boolean isWhiteSpace(char ch)`	Determina si el valor char especificado es un espacio en blanco.
`boolean isUpperCase(char ch)` `boolean isLowerCase(char ch)`	Determina si el valor char especificado es mayúscula o minúscula, respectivamente.
`char toUpperCase(char ch)` `char toLowerCase(char ch)`	Devuelve la forma mayúscula o minúscula del valor de carácter especificado.
`toString(char ch)`	Devuelve un objeto `String` representando el valor del carácter especificado—esto es, una cadena de un carácter.

Traslaciones de carácter/dígito

```
int i = Character.digit( c,base );
char c = Character.forDigit( i,base );
```

Conversión del objeto String en un tipo primitivo

Cada una de estas wrapper classes o clases envoltorio (excepto Character) tiene un método que permite convertir desde String al tipo primitivo. Simplemente hay que llamar al método de la wrapper class adecuada y el String se convierte al tipo básico.

Por ejemplo:
Convierte el contenido de la variable micadena a una variable int mientero. La conversión es fácil. La conversión de cada tipo implica el empleo de un método único para cada clase wrapper. Todas las conversiones, salvo las de Boolean se hacen con un método de nombre similar, pero todos los métodos tienen nombres distintos:

byte - Byte.parseByte(aString)
short - Short.parseShort(aString)
int - Integer.parseInt(aString)
long - Long.parseLong(aString)
float - Float.parseFloat(aString)

String micadena = "12345";
int mientero = Integer.parseInt(micadena);
double - Double.parseDouble(aString)
boolean - Boolean.getBoolean(aString)

Hay una excepción, la clase Character no tiene este método, para convertir un String en char hay que llamar al método charAt de la clase String. Si el contenido del String no puede ser convertido al tipo requerido, salta la excepción NumberFormatException, que al ser una excepción en tiempo de ejecución no requiere un bloque try-catch.

(“Conversión a caracter: ” + myString.charAt(0));

Secuencias de Escape

Un carácter precedido por una pleca invertida (\) es una secuencia de escape y tiene significado especial para el compilador. La siguiente tabla muestra las secuencias de escape de Java:

Secuencias de Escape

Secuenca de Escape	Descripción
`\t`	Inserta una tabulación en el texto en esta posición.
`\b`	Inserta un espacio de regreso (backspace) en el texto en esta posición.
`\n`	Inserta una nueva línea en el texto en esta posición.
`\r`	Inserta un regreso de carro en el texto en esta posición.
`\f`	Inserta una alimentación de forma en el texto en esta posición.
`\'`	Inserta un carácter de comilla simple en el texto en esta posición.
`\"`	Inserta un carácter de comilla doble en el texto en esta posición.
`\\`	Inserta un carácter de pleta en el texto en esta posición.

Cuando se encuentra una secuencia de escape en una instrucción de impresión, el compilador la interpreta acordemente. Por ejemplo, si desea poner comillas dentro de comillas debe usar la secuencia de escape, \", en las comillas interiores. Para imprimir esta oración

She said "Hello!" to me. escribiría
System.out.println("She said \"Hello!\" to me.");