Anexo I.- Character y String

Cuando trabajamos con caracteres se hacen necesarias algunas determinadas funcionalidades para poder trabajar de forma cómoda. Las clases **Character** y **String** disponen de una serie de propiedades y métodos que nos pueden facilitar mucho el trabajo con caracteres o cadenas de caracteres.

Character

Podemos representar un carácter de dos formas diferentes:

- char c;
- Character c;

La primera instrucción declara una variable del tipo primitivo char, y por tanto no tiene propiedades ni métodos asociados.
La segunda instrucción declara una variable de tipo Character, que es una clase y por tanto tiene propiedades y métodos que podemos utilizar.

La mayoría de métodos de la clase Character son estáticos y por tanto no es necesario crear un objeto de este tipo. Podemos ejecutar sus métodos posando el nombre de la clase "punto" y el nombre del método. Por example: Character.isDigit('0');

Métodos estáticos más usados

- boolean Character.**isLowerCase**(char c). Comprueba si el carácter está en minúsculas.
- boolean Character.**isUpperCase**(char c). Comprueba si el carácter está en mayúsculas.
- boolean Character.**isDigit**(char c). Comprueba si el carácter es un dígito.
- boolean Character.**isLetter**(char c). Comprueba si el carácter es una letra.
- boolean Character.**isSpaceChar**(char c). Comprueba si el carácter es un espacio en blanco.
- char Character.**toLowerCase**(char c). Convierte el carácter a minúsculas.
- char Character.**toUpperCase**(char c). Convierte el carácter a mayúsculas.
- Character Character.**valueOf**(char c). Devuelve una instancia de Character representando el carácter especificado.
- int Character.**getNumericValue**(char c). Devuelve el valor numérico del carácter recibido como parámetro.
- String Character.**toString**(char c). Convierte el carácter a un String.

Podemos crear un objeto de tipo Character de la siguiente forma:
Character c = new Character('G');

String

La clase String permite representar cadenas de caracteres, por lo tanto es ideal por almacenar palabras, frases o textos.

Podemos representar una cadena de caracteres de las siguientes formas:

- String s = "Hola";
- String s = new String("Hola");

Aunque parece lo mismo, la forma de crearse el objeto no es igual. La segunda forma siempre crea un nuevo objeto en el heap, una zona de memoria especial para las variables dinámicas, mientras que la primera forma puede crear o no un nuevo objeto (si no lo crea lo reutiliza del String Pool, una memoria caché diseñada para reciclar Strings).

Métodos más usados

- char **charAt**(int i). Retorna el carácter que ocupa la posición i.
- int **compareTo**(String anotherString). Compara lexicográficamente (alfabéticamente) el String actual con el String pasado como parámetro. Suele utilizarse para ordenar cadenas de caracteres.
  - Devuelve < 0, si la cadena que invoca al método es menor lexicográficamente que la cadena recibida como parámetro.
  - Devuelve == 0 si las dos cadenas son iguales lexicográficamente.
  - Devuelve > 0, si la cadena que invoca al método es mayor lexicográficamente que la cadena recibida como parámetro.
- String **concat**(String s). Concatena la cadena actual con la cadena recibida como parámetro.
- boolean **contains**(CharSequence s). Comprueba si la cadena actual contiene la cadena recibida como parámetro.
- boolean **equals**(Object o). Comprueba si el valor del String es igual al del objeto o.
- booean **equalsIgnoreCase**(String s). Comprueba si el valor del String es igual, sin tener en cuenta las mayúsculas, a la String recibido como parámetro.
- byte[] **getBytes**(Charset charset). Codifica el String en una secuencia de bytes utilizando el sistema de codificación de caracteres recibido como parámetro.
- int **hashCode**(). Devuelve un hash code para el String.
- int **indexOf**(…). Múltiples variantes. Devuelve el índice de la primera ocurrencia encontrada del carácter/String en el valor recibido como parámetro. Si no se encuentra ninguna coincidencia devuelve -1.
- int **length**(). Devuelve la longitud de la String.
- String **replace**(char oldChar, char newChar). Cambia todas las ocurrencias del carácter oldChar por el carácter newChar.
- String **replaceFirst**(String regex, String replacement). Cambia la primera ocurrencia que coincida con la expresión regular regex por el substring replacement.
- String **replace**(CharSequence target, CharSequence replacement). Cambia cada substring que coincida con el target por la secuencia de caracters indicada por replacement.
- String[] **split**(String regex). Parte el String varias partes según las reglas indicadas en la expresión regular regex.
- String **substring**(int beginIndex). Devuelve un String que empieza desde el carácter número

beginIndex hasta el final.

- String **substring**(int beginIndex, int endIndex). Devuelve un String que empieza desde el carácter número beginIndex hasta el carácter número endIndex.
- char[] **toCharArray**(). Convierte el String en un array de caracteres.
- String **toLowerCase**(). Convierte el String a minúsculas.
- String **toUpperCase**(). Coverteix el String a mayúsculas.
- String **trim**(). Devuelve el mismo String pero sin los espacios en blanco del inicio y del final.
- String String.**ValueOf**(…). Múltiples opciones. Devuelve como String el tipo de datos pasado como parámetro.