Ayudantía 10

Alexander Inostroza – alexander.inostroza@usm.cl

Strings

Los Strings en Java, a diferencia de Python, son objetos de una clase, y esta clase tiene sus métodos:

Method	Description	Return Type
charAt()	Returns the character at the specified index (position)	char
codePointAt()	Returns the Unicode of the character at the specified index	int
codePointBefore()	Returns the Unicode of the character before the specified index	int
codePointCount()	Returns the number of Unicode values found in a string.	int
compareTo()	Compares two strings lexicographically	int
compareToIgnoreCase()	Compares two strings lexicographically, ignoring case differences	int
concat()	Appends a string to the end of another string	String
contains()	Checks whether a string contains a sequence of characters	boolean
contentEquals()	Checks whether a string contains the exact same sequence of characters of the specified CharSequence or StringBuffer	boolean
copyValueOf()	Returns a String that represents the characters of the character array	String
endsWith()	Checks whether a string ends with the specified character(s)	boolean
equals()	Compares two strings. Returns true if the strings are equal, and false if not	boolean
equalsIgnoreCase()	Compares two strings, ignoring case considerations	boolean
format()	Returns a formatted string using the specified locale, format string, and arguments	String
getBytes()	Converts a string into an array of bytes	byte[]
getChars()	Copies characters from a string to an array of chars	void
hashCode()	Returns the hash code of a string	int

indexOf()	Returns the position of the first found occurrence of specified characters in a string	int
intern()	Returns the canonical representation for the string object	String
isEmpty()	Checks whether a string is empty or not	boolean
join()	Joins one or more strings with a specified separator	String
lastIndexOf()	Returns the position of the last found occurrence of specified characters in a string	int
length()	Returns the length of a specified string	int
matches()	Searches a string for a match against a regular expression, and returns the matches	boolean
offsetByCodePoints()	Returns the index within this String that is offset from the given index by codePointOffset code points	int
regionMatches()	Tests if two string regions are equal	boolean
replace()	Searches a string for a specified value, and returns a new string where the specified values are replaced	String
replaceAll()	Replaces each substring of this string that matches the given regular expression with the given replacement	String
replaceFirst()	Replaces the first occurrence of a substring that matches the given regular expression with the given replacement	String
split()	Splits a string into an array of substrings	String[]
startsWith()	Checks whether a string starts with specified characters	boolean
subSequence()	Returns a new character sequence that is a subsequence of this sequence	CharSequence
substring()	Returns a new string which is the substring of a specified string	String
toCharArray()	Converts this string to a new character array	char[]
toLowerCase()	Converts a string to lower case letters	String
toString()	Returns the value of a String object	String
toUpperCase()	Converts a string to upper case letters	String
trim()	Removes whitespace from both ends of a string	String
valueOf()	Returns the string representation of the specified value	String

Fuente: https://www.w3schools.com/java/java_ref_string.asp

Documentación completa de Strings aquí.

Arrays

Los arreglos en Java son el análogo a listas de Python, pero:

- Son de tamaño fijo.
- Sólo pueden tener elementos de un tipo.

```java
public class Main {
    public static void main(String[] args) {
        int arreglo[]; // declaracion de un array
        arreglo = new int[10]; // definiendo tamaño y asignando memoria
        for(int i=0; i<10; i++){ // recorriendo con indices
            arreglo[i] = i*i; // asignacion de valores
        }
        int[] arreglo2 = new int[]{1,2,3,4,5,6,7,8,9,10}; // arreglo literal
        int arreglo3[] = {10,9,8,7,6,5,4,3,2,1};

        int arr[][] = {{2, 7, 9}, {3, 6, 1}, {7, 4, 2}}; // en 2D

        System.out.println(arr.length);
    }
}
```

Recursión

La recursión es una técnica utilizada en programación que utiliza funciones llamándose a sí mismas.

```java
public class Main {
    public static int potencia(int base, int exponente){
        if(exponente == 0) return 1;
        return base * potencia(base, exponente - 1);
    }

    public static boolean palindromo(String palabra){
        if(palabra.length() <= 1) return true;
        return palabra.charAt(0) == palabra.charAt(palabra.length() - 1)
            && palindromo(palabra.substring(1, palabra.length() - 1));
    }
}
```

Ejercicios

1. Programe las siguientes funciones de manera **recursiva**:

a) **Fib(n)**, que entrega el n-ésimo número en la serie de Fibonacci, definida como:

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}$$

Algunos ejemplos:

Fib(2) = 1	Fib(5) = 5
Fib(3) = 2	Fib(6) = 8
Fib(4) = 3	Fib(7) = 13

b) **BinToDec(n)** ¸que recibe un número entero compuesto de 0s y 1s (número binario) y lo convierte a un número decimal.

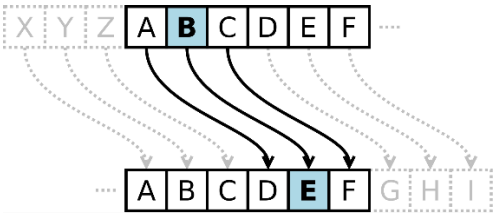
101001	→	41
1101	→	13
1000100	→	68

c) **Ordenar(arr,n)**, con arr un arreglo de enteros sin repetición y n la cantidad de elementos. Esta función debe tomar el mayor elemento e intercambiarlo con el último del arreglo en cada iteración, hasta que deba ordenar sólo un elemento.

d) **Suma(arr)**, con arr un arreglo de enteros; que entrega el resultado de sumar todos los valores del arreglo arr.

2. **Cifrado César**

El cifrado César es una de las técnicas de cifrado más simples y más usadas, consiste en reemplazar cada letra del mensaje que se quiere codificar por la letra que está a una distancia x en el alfabeto. Puede verse también como una rotación del alfabeto.



a) Escriba la función **Cifrar(texto, n)**, que aplicará cifrado César al **texto**, moviendo cada letra una distancia **n**.
b) Escriba un programa main que lea el archivo **cifrar.txt**, le aplique cifrado César con **n = 3** y escriba el mensaje cifrado en un archivo **cifrado.txt**.

3. **Búsqueda binaria**

En el proceso de buscar un elemento en una lista de datos ordenados, si la cantidad de datos es demasiado grande, puede que tardemos mucho tiempo en encontrar nuestro dato, por ejemplo, si tenemos un arreglo de 1 millón de datos y buscamos de forma secuencial, y el dato que buscamos resulta ser el último, la búsqueda será exitosa después de 1 millón de comparaciones.

Hay opciones para optimizar esta búsqueda, una muy buena es la búsqueda binaria recursiva: Aprovechando que los datos están ordenados, revisamos el número del medio de la lista,

- Si el número es menor al buscado, iteramos para los números a su derecha como un sub-arreglo.
- Si el número es mayor al buscado, iteramos para los números a su izquierda como un sub-arreglo.
- Si el número es igual al buscado, lo encontramos. 😄
- Si el tamaño del arreglo es 1, es más fácil aún.

Un GIF demostrativo.

Dado un arreglo ordenado de n números únicos, defina la función **binarySearch(arr,num,pos_i,pos_f)** , que tras realizar búsqueda binaria recursiva, entregue el índice del número buscado. Si no lo encuentra debe retornar **-1**.

4. **N puntos**

Dada una clase **Function**, que representa una función y tiene un método estático **eval(x)**, que evalúa la función en el valor x. Escriba el método **n_values(a,b,n)** , que entrega **n** valores equiespaciados del intervalo **[a,b]** evaluados en la función. ¿Para qué podría servir n_values?

5. **Pendiente: Sopa de letras y sumatoria**