Introducción

Representar y manejar textos es algo habitual dentro de una aplicación informática.

Las cadenas de caracteres se almacenan en Kotlin utilizando el tipo **String**. Normalmente el valor de un **String**, es una secuencia de caracteres entre dobles comillas.

```
var textoMenu = "Archivo"
```

Un objeto String, no solo almacena el valor de una cadena de caracteres, sino que tiene numerosos métodos para operar con dicho valor y con otros relacionados.

Los String son **inmutables**. Una vez que se inicializa una cadena, no puede cambiar su valor ni asignarle un nuevo valor. Todas las operaciones que transforman cadenas devuelven sus resultados en un **nuevo objeto String**, dejando la cadena original sin cambios.

En cualquier aplicación, siempre que necesites una cadena, crearás un objeto de esta clase y podrás utilizar todas las operaciones que ofrece (métodos):

- Algunos de los **métodos** son: **get**, **length, substring**, **indexOf**, **lastIndexOf**, **endsWith**, **startsWith**, **replace**.

- Otros métodos son los que permiten la **comparación** entre dos objetos String: **equals** y **compareTo**.

Podemos ver todas las funcionalidades que nos ofrece **String** en:
https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-string/

Caracteres de un String

Podemos interpretar un **String** como una secuencia de elementos Char. Esto te permitirá leer los caracteres en sus posiciones a través de get() o su operador equivalente de acceso en corchetes string[índice], siempre teniendo en cuenta que la primera posición es la 0.

Si se indica una posición que sobrepasa la longitud de la cadena, se lanza la excepción **IndexOutOfBoundsException.**

```
var cadena :String = "Cadena de texto";
println(cadena[0]);    // C
println(cadena.get(0)); // C
```

También es posible recorrer los caracteres de un String a través del bucle for:

```
for(char in cadena){
    print(char)
}
```

Incluso usar la propiedad índices para obtener el rango de índices que sirvan de referencia en el bucle for:

```
for(index in cadena.indices){
    println(cadena[index])
}
```

Obtener la longitud de una cadena

Para obtener la longitud de una cadena utilizaremos el método **length()**. Hay que tener en cuenta que devolverá el número de caracteres incluyendo los espacios.

También podremos saber si una cadena está vacía con el método **isEmpty()**. Recuerda que nunca será lo mismo que la cadena esté vacía a que no haya sido instanciada.

Con la función **trim(),** se eliminan los espacios que existan al principio y al final, pero no entre medias.

```
fun main(){
    val cadenaVacia = ""
    // Podemos comprobar si la cadena está vacía (length() == 0)
    if (cadenaVacia.isEmpty()){
        println("cadena vacia")
    }

    /**
     * Si cadenaNula es null
     *      --> VALOR DE: cadenaNula?.isEmpty() esto será null
     *      --> CONDICION: cadenaNula?.isEmpty() == true, será false
     */
    var cadenaNula: String? = null

    val cadena = "esto es una cadena"
    println(cadena.length) // 18

    val cadenaConEspacios = " espacios "
    println(cadenaConEspacios)  // " espacios "
    println(cadenaConEspacios.trim()) // "espacios"
}
```

Concatenación de cadenas

Concatenar es unir el contenido de una cadena con el de otra, lo que te permite generar una nueva cadena. También, te permite generar una nueva cadena entre un **String** y otro tipo. Se puede hacer de varias maneras, por ejemplo, utilizando el **operador suma** (+).

```
var st1 = "me gusta" + " Kotlin"
var st2 = "Cantidad de ahorros" + 536.4
var st3 = "Resultado: " + (2+4)
```

Puedes usar combinación de variables o literales si así lo deseas. Incluso, puedes usar el operador de asignación y adición en su forma simplificada:

```
var st4 = "Nueva concatenación"
st4 += '!'  // "Nueva concatenación!"
```

➔ El uso de '+' es **INEFICIENTE**. La alternativa es utilizar la clase **StringBuilder** (lo veremos más adelante).

Estudia el siguiente ejemplo:

```
var s = "Jesus"
s = s + " Fernandez" // Jesus Fernandez
```

Parece que **s** se *modifica* añadiendo "Fernandez" pero en realidad lo que sucede es que:

- Crea un nuevo objeto **String** con el contenido de **s** y " Fernandez".

- El nuevo objeto **String** queda referenciado por **s** y el anterior objeto que contiene "Jesus" queda **desreferenciado**, siendo candidatos a ser eliminados de la memoria mediante un recolector de basura o garbage collector, que trabaja en un segundo plano de forma transparente para el programador. Esto hace que la liberación de la memoria no sea una preocupación de primer nivel para un programador y que pueda centrar toda su atención en escribir código de calidad.

Se puede demostrar que la concatenación de objetos **String** con el **operador** + es **ineficiente**, ya que los objetos **String** al ser inmutables, el compilador hace que se genere un nuevo objeto **String** cada vez que se necesita modificar.

- Es una **mala práctica** hacer lo siguiente:

```
var s = s1 + s2 + s3;
```

- NO pasa lo mismo con el siguiente ejemplo:

```
var s = "Esto " + "es " + "un ejemplo";
```

En este caso, como se usan constantes, NO se realiza la reserva de memoria, sino que el compilador lo agrupa todo en un único objeto **String**.

Para realizar la concatenación citada en caso anterior, se deberán utilizar objetos de una clase implementada para este propósito llamada **StringBuilder**.

- Esta clase tiene varios métodos que modifican la cadena. Por ejemplo, con el método **append** puedes añadir texto.

- Al ser su contenido modificable, **NO se crean nuevos objetos** como si ocurre en el caso de objetos **String**.

- El método **toString** permite devolver un objeto **String** con la cadena que contiene el objeto **StringBuilder**.

Ejemplo: Dados dos String con los apellidos, construir un String con el formato Apellido1 Apellido2, Nombre usando **StringBuilder**:

```
var strApellido1 = "Pérez"
var strApellido2 = "Gómez"

// Compongo una cadena con el nombre completo
var sbNombreCompleto = StringBuilder("");

sbNombreCompleto.append(strApellido1);// Agrego Pérez
sbNombreCompleto.append(" ");       // Agrego espacio
sbNombreCompleto.append(strApellido2);// Agrego Gómez

sbNombreCompleto.append(",");       // Agrego coma
sbNombreCompleto.append(" ");       // Agrego espacio
sbNombreCompleto.append("Luis");       // Agrego Luis

// Obtengo un objeto String a partir de uno StringBuffer
var strNombreCompleto = sbNombreCompleto.toString();

println("Nombre completo: "+ strNombreCompleto);// Pérez Gómez, Luis
```

Se recomienda, por tanto, usar la clase **StringBuilder** para concatenar cadenas de caracteres, en lugar de usar el operador (+)

Importante:

➔ NO usar **equals** ó == para comparar dos objetos **StringBuffer** porque no funcionará tal y como se espera.

La razón es que este método no está sobreescrito en la clase **StringBuffer** (la sobreescritura de métodos tiene que ver con la herencia, que se verá más adelante)

Comparación de cadenas

Las cadenas de texto, como "casi" todos los objetos en Kotlin, pueden ser comparadas con el operador `==` y el método **equals**, ya que internamente el operador `==` llama al método equals.

El método **equals** que compara el contenido del objeto:

- devuelve ***true*** si son iguales (igual longitud e igual contenido de ambas cadenas).
- devuelve ***false*** si no son exactamente iguales

```
var nombre = "Asociación Española de Programadores Informáticos"
var nombreCorto = "AEPI"
if (nombre.equals(nombreCorto)) {
   println("Las dos cadenas son iguales")
}
else {
   println("Las dos cadenas de texto son diferentes")
}
```

El método **equals(other: String, ignoreCase: Boolean)** , pasando true como segundo parámetro, funciona de la misma manera que el *equals*, pero sin tener en cuenta mayúsculas y minúsculas.

```
var nombre = "Asociación Española de Programadores Informáticos"
var nombre2 = "asociación Española De programadores informáticoS"
if (nombre.equals(nombre2, true)) {
   println("Las dos cadenas son iguales")
}
else {
   println("Las dos cadenas de texto son diferentes")
}
```

Para saber si una cadena es mayor o menor que la otra, existe el método **compareTo**. Este método compara ambas cadenas, considerando el orden alfabético:

- Si la primera cadena es mayor en orden alfabético que la segunda, devuelve un **valor mayor que 0**.
- Si son **iguales** devuelve **0**.
- Si la primera es menor en orden alfabético que la segunda, devuelve un **valor menor que 0**.

```
var st1 = "AAA"
var st2 = "BBB"

// Comparo las dos cadenas alfabeticamente
var ret = st1.compareTo(st2)
if (ret > 0){
        println("$st1 es mayor que $st2")
}
else if (ret <0) {
        println("$st2 es mayor que $st1")
}
else {
        println("$st1 y $st2 son iguales")
}
```

Cambiar las letras de mayúscula a minúsculas

También podemos pasar una cadena de texto a minúsculas o mayúsculas. La clase **String** tiene métodos para realizar ambas operaciones, devolviendo siempre el resultado final, se tratan de los métodos: **lowerCase()** y **upperCase().**

```
var nombre = "Asociación Española de Programadores Informáticos"
var nombreMinusculas = nombre.lowerCase()
println(nombreMinusculas)  // asociación española de programadores informáticos

var nombreMayusculas = nombre.upperCase()
println(nombreMayusculas) // ASOCIACIÓN ESPAÑOLA DE PROGRAMADORES INFORMÁTICOS
```

Subcadenas

Una subcadena es el fragmento que extraemos de una cadena más larga.

La clase **String** en proporciona diferentes formas de hacerlo a través de las diferentes sobrecargas del método **substring()**, dependiendo de los parámetros que se le pasen.

Tendremos que tener en cuenta que la primera posición de una cadena será la posición 0.

Si las posiciones no son válidas, se lanza la excepción **IndexOutOfBoundsException**.

```
var nombre = "Asociación Española de Programadores Informáticos"

// Devuelve la subcadena desde la posición 10 en adelante
var subcadena = nombre.substring(10)
println(subcadena)    // Española de Programadores Informáticos

// Devuelve la subcadena entre dos posiciones dadas
subcadena = nombre.substring(5, 10)
println(subcadena)    // ación
```

Búsqueda en una cadena

También podemos buscar *subcadenas* dentro de una *cadena*. Podremos buscar desde un carácter hasta una cadena de longitud indeterminada, usando el método: **indexOf**(), el cual devuelve la primera posición en la que aparece una determinada cadena pasada por parámetro, en el caso de que la cadena buscada no se encuentre, devuelve **-1**.

El método **indexOf** está sobrescrito, tal que admite también, los parámetros:

- startIndex, tal que se puede buscar a partir de una posición en concreta
- ignoreCase, si true, se ignoran mayúsculas de minúsculas.

Otros métodos de la clase *String*, para búsqueda en una cadena son:

- **lastIndexOf:** Devuelve la **última posición** en la que aparece una determinada cadena pasada por parámetro Es casi idéntica a la anterior, sólo que busca desde el final.
- **endsWith:** Devuelve ***true*** si la cadena termina con un determinado texto.
- **startsWith:** Devuelve ***true*** si la cadena empieza con un determinado texto.

```
var nombre = "Asociación Española de Programadores Informáticos"
var posicion = nombre.indexOf("Española")
if (posicion == -1) {
   println("La palabra 'Española' no se encuentra en la cadena")
}
else {
   println("Se ha encontrado la cadena 'Española' en la posición: $posicion ")  //11
}
```

Sustitución en una cadena

Podemos reemplazar fragmentos de la cadena de texto por otros que pasemos como parámetros al método **replace(),** que devolverá la cadena resultante.

```
var nombre = "Asociación Española de Programadores Informáticos"
var valorAntiguo = "Programadores"
var reemplazo = "Desarrolladores"

nombre = nombre.replace(valorAntiguo, reemplazo)
println(nombre)  // Asociación Española de Desarrolladores Informáticos
```

String con múltiples líneas

En ocasiones nos interesa crear literales de String que posean múltiples líneas. Esto lo conseguimos usando la sintaxis de triple comillas (""")

```
fun main() {

    val welcome= """
        ¡Bienvenido a clase de Programación!
        Aprenderás
        los conceptos básicos sobre el lenguaje
        y las herramientas necesarias para probarlo
    """
    println(welcomeText)
}
```

Otras funciones de String

La clase **String** tiene múltiples funciones, que te ofrecen diferentes funcionalidades como, por ejemplo:

- removeSuffix: Si la cadena termina con el sufijo dado, devuelve una copia de esta cadena sin el sufijo.

- removePrefix: Si la cadena comienza con el prefijo dado, devuelve una copia de esta cadena sin el prefijo.

- removeSurroundings: Elimina la cadena delimitadora dada, tanto del inicio como del final de esta cadena si y solo si comienza y termina con el delimitador.

```
println("mo_dancing.jpg".removeSuffix(".jpg")) // mo_dancing
println("HelloWorld".removePrefix("Hello")) //World
println("__OJO__".removeSurrounding("__")) // OJO
```