Java API and Arrays

Clase `String`

`String` se encuentra definida en el paquete `java.lang` .

Creando objetos `String`

Se pueden crear `String` a través del operador `new` o utilizando *literal values* con comillas dobles

Cuando se utiliza el operador `new` la cadena no considera el pool de `String` .

Cuando se utilizan *literal values* se considera el pool de `String` .

`String` tiene constructores que permiten crear un `String` a partir de `StringBuilder` o `StringBuffer` .

`null` es un *literal value* para objetos.

Inmutabilidad de `String`

Una vez creado un objeto `String` , no se puede modificar.

Ninguno de los métodos definidos en `Strig` manipula los elementos del arreglo de valores de la cadena. Cada que estos métodos parecieran modificar, en realidad se crea otro objeto.

Un `String` internamente maneja un arreglo de `char` .

Métodos de `String`

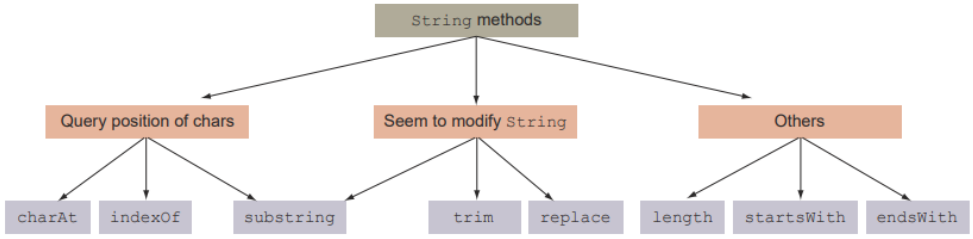


Figure 4.8 Categorization of the `String` methods

`charAt(int index):char`

Este método permite obtener el caracter en el índice especificado.

`indexOf(char|String):int index`

Este método busca una coincidencia entre el argumento introducido, ya sea un `char` o `String` . El retorno es el index inicial de la coincidencia. Si no hay resultados se retorna -1.

Este método inicia de 0 hasta el final de la cadena, para cambiar la posición de inicio de busqueda el método es `indexOf(char myChar, startIndex)`

`substring(int index): String`

Retorna un substring a partir de la posición indicada

Es posible indicar la posición de termino pasando el segundo parámetro y este no se incluye, es excluyente.

`substring(int startInclusive, int endExclusive)`

`trim():String`

El método `trim()` retorna un `String` y remueve los espacios al inicio y al final de una cadena.

`replace():String`

El método retorna un `String` reemplazando todas las coincidencias de caracter con otro caracter; o con `String.replace(charToFind|StringToFind, charToReplace|StringToReplace)`

No se pueden mezclar los tipos, de lo contrario no compila, únicamente se puede reemplazar con el mismo tipo.

`length():String`

Este método permite conocer el tamaño de un `String` .

`startsWith(String):boolean & endsWith(String):boolean`

Determinan si la cadena empieza o termina con una cadena determinada.

Con `starWith(String,int startIndex)` se modifica donde comienza.

Encadenamiento de métodos

Los métodos se evalúan de izquierda a derecha

Operadores

■ Concatenation: `+` and `+=`
■ Equality:
`==` and `!=`

Cuando se realiza un concatenación se crea un nuevo objeto, no olvidar la inmutabilidad.

Las expresiones se leen de izquierda a derecha, por lo que si hay dos `Number` a la izquierda, se realizara a suma antes que la concatenación.

La suma de números se puede forzar con parentecis.

Si al concatenar una variable es `null` , se imprime `nullmicadena` .

Determinando la igualdad de `Strings`

`equals()` retorna `true` si el valor representa la misma secuencia de caracteres que el objeto comparado.

El operador `==` y `!=` compara las referencias.

Igualdad de los valores retornados en los métodos

Las cadenas retornadas en métodos no son puestos en el pool de `String` .

`StringBuilder` - cadenas mutables

La clase se encuentra definida en el paquete `java.lang` y tiene una secuencia mutable de caracteres.

Creación de objetos

Existen varios constructores sobrecargados:

- `StringBuilder()` → No args → Inicia con una capacidad por defecto de 16 caracteres.
- `StringBuilder(Object)`
- `StringBuilder(int capacity)` → Capacidad inicial de la cadena
- `StringBuilder("string")`

Métodos de la clase `StringBuilder`

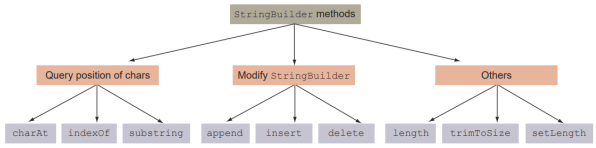


Figure 4.14 Categorization of `StringBuilder` methods

`append(any, int startInclusive, int endExclusive)`

Añade el valor especificado al final de la cadena.

El método se encuentra sobrecargado y acepta todos los primitivos, arrays, objectos.

Cuando se envía un objeto como parámetro se utiliza su método `toString()`

`insert(int index, any)`

Permite insertar los datos en una posición determinada

Figure 4.15 illustrates the previous code.

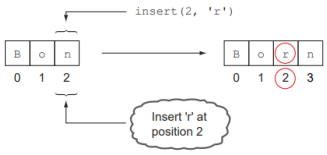


Figure 4.15 Inserting a `char` using the method `insert` in `StringBuilder`

Otra forma es `insert(int index, any , anyStartInclusive, int anyEndExclusive)`

`delete() deleteCharAt()`

Remueve una cadena de caracteres o un solo caracter en una posición.

`delete(int startIndexInclusive, int endIndexExclusive`

`deleteCharAt(int index)`

`reverse()`

Invierte la secuencia de caracteres.

No se puede utilizar este método para invertir un substring.

`replace()`

Reemplaza un secuencia de caracteres identificados por sus posiciones .

`replace(int startInclusive, int endExclusive, "cadenaReemplazo")`

Arreglos

Un arreglo es un objeto que almacena una colección de valores.

Un arreglo puede almacenar dos tipos de datos:

- Datos primitivos → En si mismos sin referencias.
- Referencias de objetos → Almacena una colección de referencias que apuntan a los objetos.

Los arreglos multidimensionales corresponden a arreglos que contienen arreglos. Teóricamente no existe un número límite de dimensiones anidadas.

Declaración de arreglos

Una declaración de arreglos debe incluir el tipo, y la variable.

- `int intArray[];`
- `String[] strArray;`
- `int[] multiArray[];`
- `int[][] multiArray;`
- `int multiArray[][];`

La creación de esta variable apunta a `null`

El tamaño del arreglo no se define en la variable → No compila

Instancia de arreglos

Se debe definir el tamaño del arreglo, de lo contrario no compilará. Los arreglos multidimensionales pueden sólo definir el tamaño del primer nivel.

El tamaño del arreglo debe ser un `int` , y se aceptan expresiones que retornen un `int`

Una vez que se ha instanciado, sus dimensiones no puede cambiar.

- `intArray = new int[2];`
- `strArray = new String[4];`
- `multiArr = new int[2][3];`
- `multiArr = new int[2][]`
- `multiArr = new int[2*3][Math.max(2,3)];`

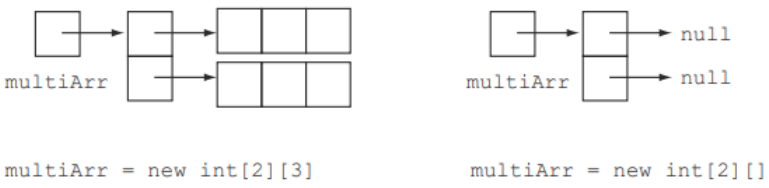


Figure 4.24 The difference in array allocation of a two-dimensional array when it's allocated using values for only one of its dimensions and for both of its dimensions

Los valores por defecto de un arreglo de objetos son `null`

En el caso de primitivos:

- Integers → 0

- Decimal → 0.0
- Boolean → `false`
- Char → `\u0000`

Inicialización de arreglos

Se le llama inicializar un arreglo cuando se asignan datos a los espacios que definimos en la instancia.

- `strArray[1] = "summer";`
- `strArray[0][2] = "winter";`

Cuando se trata de acceder a un index que no existe se lanza la runtime exception `ArrayIndexOutOfBoundsException`

El compilador no verifica los índices y tamaños de arreglos, por lo que incluso puedes poner índices negativos y compilará, pero se espera `ArrayIndexOutOfBoundsException`

El código no compilará si no se utiliza `char, byte, short, int` para indicar los índices.

Combinación de declaración, instancia e inicialización

Los pasos anteriores se pueden combinar.

Si el arreglo de declara explícitamente, no se le debe asignar dimensiones, de lo contrario no compilará.

- `int intArray[] = new int[] {0,1};`
- `int multiArray[][] = new int[][]{ {1,0}, {2,3} };`
- `int intArray[] = {0,1};`
- `String[] strArray = {"summer","winter"};`
- `int multiArray[][] = { {0,1}, {2,3,4} };`

Arreglos multidimensionales asimétricos

Figure 4.25 shows this asymmetrical array.

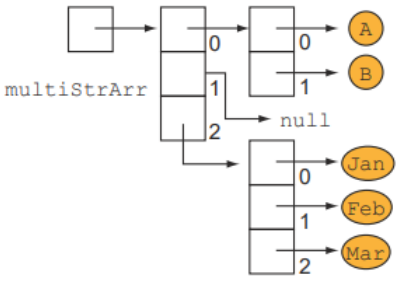


Figure 4.25 An asymmetrical array

Arreglos de interfaces y clases

Interfaces

Sus elementos por defecto son `null` y puede almacenar objetos que implementen dicha interfaz.

Clases abstractas

Sus elementos por defecto son `null` y puede almacenar objetos de las clases concretas que extienden de la clase.

`Object`

Debido a que todas las clases extienden de `java.lang.Object` , los arreglos de tipo `Object` pueden referirse a cualquier objeto.

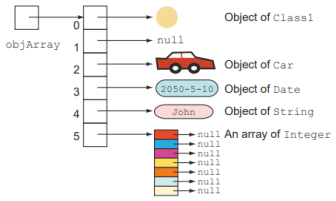


Figure 4.26 An array of class `Object`

Miembros de un arreglo

Un arreglo tiene los siguientes miembros públicos:

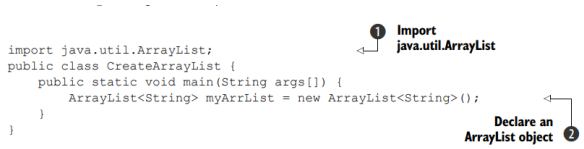
- Atributo `length` que almacena el número de elementos del arreglo
- Método `clone()` que sobreescribe al que se encuentra en la clase `Object` . El tipo de retorno es el mism que el del arreglo original.

ArrayList

Características

- Implementa la interfaz `List`
- Permite añadir valores `null`
- Permite duplicados
- Mantiene el orden de inserción
- Se puede utilizar `Iterator` ó `ListIterator` para iterar los elementos.
- Soporta genéricos, lo que brinda seguridad de tipo.
- Se requiere importar de `java.util.ArrayList`

Creando un `ArrayList`




```
ArrayList<String> myArrList = new ArrayList<>();
```

Missing object type on right of = works in Java version 7 and above

Un `ArrayList` internamente maneja `Arrays` , pero provee los beneficios de un arreglo dinámico.

Añadiendo elementos

Se utiliza el método `add(elementos)` para añadir el item al final del arreglo

Se utiliza el método `add(int index,elementos)` para agregar el elemento en la posición específica y la lista se desplaza.

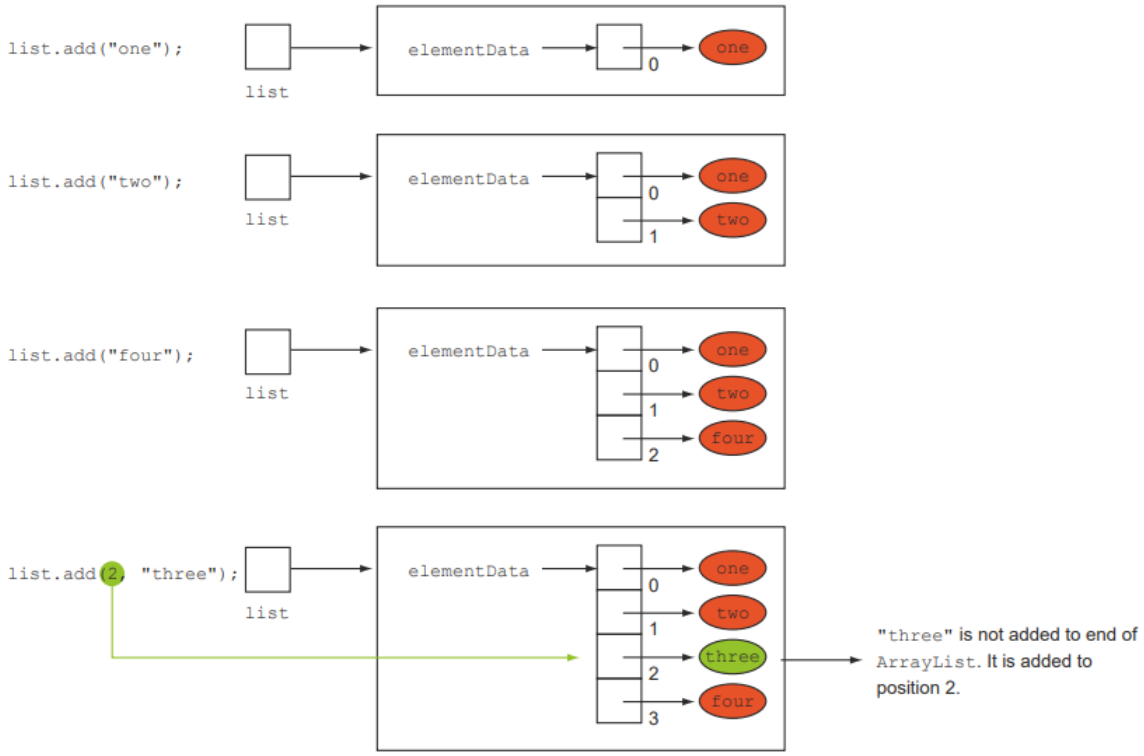


Figure 4.28 Code that adds elements to the end of an `ArrayList` and at a specified position

Accediendo a los elementos de un `ArrayList`

Se utiliza el método `get(index)` o loops o `ListIterator`.

En caso de que no exista el index se lanza `java.lang.IndexOutOfBoundsException`

```
import java.util.ArrayList;
import java.util.ListIterator;
public class AccessArrayListUsingListIterator {
    public static void main(String args[]) {
        ArrayList<String> myArrList = new ArrayList<String>();
        myArrList.add("One");
        myArrList.add("Two");
        myArrList.add("Four");
        myArrList.add(2, "Three");
        ListIterator<String> iterator = myArrList.listIterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

1 Get the iterator

2 Use hasNext() to check whether more elements exist

3 Call next() to get the next item from iterator

Modificando los elementos de un `ArrayList`

Para modificar se utiliza el método `set(index, elemento)`

Otra manera es a través de los métodos particulares de cada objeto.

Borrando elementos

- `remove(int index)` → Remueve el elemento en la posición indicada
- `remove(Object o)` → Remueve la primera coincidencia si se presenta.

Limpiando los elementos

`clear()` → Método para eliminar la lista.

Otros métodos

`size()` → Retorna el número de elementos en la lista.

`contains(Object o)` → Retorna `true` si la lista contienen el elemento especificado.

`indexOf(Object o)` → Retorna el index de la primera coincidencia o `-1` si no se tiene el elemento.

`lastIndexOf(Object o)` → Retorna el index de la última coincidencia o `-1` si no se tiene el elemento.

`clone()` → retorna una nueva copia del arreglo, no de los elementos en él.

`toArray()` → Crea un nuevo arreglo con las referencias.

Comparando objetos para la equidad

El método `equals()` está definido en la clase `java.lang.Object` y por defecto compara la referencia entre objetos.

Para hacer Override del método `equals()` es necesario que reciba como parámetro `Object obj` , de lo contrario se estaría haciendo una sobrecarga.

Cuando sobreescribes el método `equals()` tambien se debería sobreescribir el método `hashCode()`