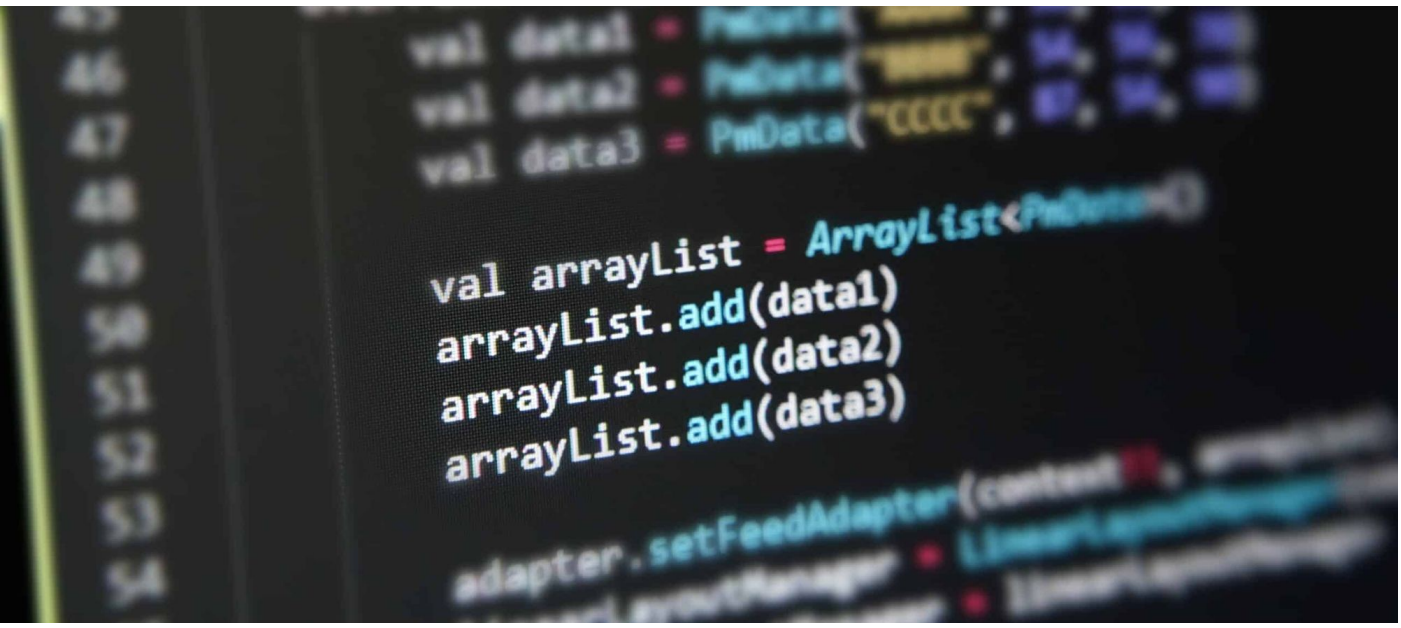

1. Arrays unidimensionales

Date	Empty
Status	Not started
Type	Empty
Units	4. Estructuras de almacenamiento

Arrays

- Arrays unidimensionales
- Acceso a los elementos de un array
- Modificar los elementos de un array
- Recorrer los elementos de un array
- Operaciones de búsqueda
- Clonar un array
- Comparar arrays completos

Arrays

Los arrays de datos son variables que guardan en su interior un conjunto de datos, bien primitivos o referencias a objetos, de forma

que son accesibles desde un punto concreto. El punto negativo que tienen este tipo de colecciones es que su longitud no es alterable una vez está creada la colección.

Arrays unidimensionales

Los arrays unidimensionales son aquellos que guardan datos únicos, es decir cada posición es un puntero al dato, bien sea primitivo o referencia a un objeto

Los arrays pueden ser definidos bien con el número de posiciones que admiten, o con los elementos que formar parte de la colección

```
// la declaración incluye las variables que formarán parte de
l objeto array int [] numeros = {0,1,2,3,4,5,6,7,8,9} // la d
eclaración no incluye los datos que formarán parte del array
pero si se indica el número de posiciones máximas que podrá t
ener el array int [] numeros = new int[9]
```

Se pueden declar arrays que guarden tipos concretos o arrays que guarden tipos múltiples

```
char[] v1; boolean[] v2; byte[] v3; short[] v4; int[] v5; lon
g[] v6; float[] v7; double[] v8; BigInteger[] v9; BigDecimal
[] v10; String[] s; Character[] a1; Boolean[] a2; Byte[] a3;
Short[] a4; Integer[] a5; Long[] a6; Float[] a7; Double[] a8;
Object[] o1;
```

Algunos ejemplo de definición

```
String[] palabras = {"ejemplo","de","array","unidimensiona
l","de","palabras"} Object datos = {"ejemplo", 3,false,"palab
ra"};
```

Hay que tener en cuenta que si un array es definido en un punto del programa, utilizado por n instrucciones y vuelvo a ser definido, la operación que se lleva a cabo es la de reinicio del array y todos los componentes que formaban parte del mismo

Acceso a los elementos de un array

En el primer ejemplo de array definido

```
int [] numeros = {0,1,2,3,4,5,6,7,8,9}
```

Para poder acceder a los elementos del array simplemente hay que indicar cual es la posición a que se quiere acceder:

```
numeros[0] // devolverá el valor 0
```

Modificar los elementos de un array

En el caso de querer guardar un dato dentro de una posición concreta, simplemente hay que igualar esa posición al valor que se quiera guardar. Hay que tener en cuenta que esta operación sobreescribe el valor que estaba previamente

```
numeros[0]=12 // guardará el valor 12 en la posición 0
```

Recorrer los elementos de un array

Normalmente para poder recorrer una colección de datos se utiliza una estructura for o foreach donde en cada iteración se accede al siguiente valor.

```
int [] numeros = {1,2,3,4,5,6,7,8,9,10} for (int i = 0; i < n
umeros.length;i++){ System.out.println (String.format(“La pos
ición %d tiene asignado el valor %d",i,numeros[i])); }
```

Existe una forma más sencilla de hacer esto que es con el bucle foreach

```
for (int num: numeros){ ystem.out.println (num)); }
```

Uno de los fallos más comunes es el de recorrerlos de forma incorrecta, intentando acceder a posiciones que no existen y por lo tanto obteniendo fallos de acceso o tambien llamadas

ArrayIndexOutOfBoundsException. Este tipo de fallo se produce cuando se quiere acceder a una posición de array que en realidad no existe. Para ello se utiliza el bucle for visto anteriormente desde 0 hasta array.length

```
String[] palabras = {"ejemplo","de","array","unidimensiona
l","de","palabras"}; // este array tiene 6 elementos, pero te
niendo en cuenta que la primera posición es la 0, // si se in
tenta acceder a la posición 6 obtendriamos el error mencionad
o anteriormente palabras[6] //ArrayIndexOutOfBoundsException
```

Existe la posibilidad de trabajar con arrays de forma manual para realizar diversas tareas como por ejemplo ordenar sus elementos, pero en java existe un paquete llamado Array que facilita mucho el trabajo

Operaciones de búsqueda

Para poder realizar búsquedas dentro de los elementos de un array se puede hacer de dos formas:

- Recorriendo el array de forma manual evaluando elemento a elemento

Esta forma de proceder es la unión de diferentes técnicas como son el recorrido de un array y la evaluación individual de cada elemento

Sea el arra int[] numeros = {4, 1, 2, 6, 8, 3, 4, 89, 67, 32, 12};

1. Se recorre el array

```
int[] num = {2, 4, 6, 7, 9, 1}; Arrays.sort(num); for (int n
: num) { }
```

1. Se evalúa cada elemento preguntando si es el buscado

```
int[] num = {2, 4, 6, 7, 9, 1}; Arrays.sort(num); for (int n
: num) { if (n == 7) { } }
```

1. En caso afirmativo se rompe el bucle

```
int[] num = {2, 4, 6, 7, 9, 1}; int iteracion=0; Arrays.sort
```

```
(num); for (int n : num) { if (n == 7) { break; } iteracio
n++; } System.out.printf("Han sido necesarias %d iteracione
s",iteracion);
```

- Convirtiéndolo en el una lista y ejecutando un método específico

En el ejemplo anterior hay que recorrer todos los elementos hasta encontrar el concreto, pero existe la posibilidad gracias a la clase Arrays de convertir el elemento en una lista (será explicada más adelante)

1. Se convierte el array en una lista y se pregunta a la lista si contiene un valor determinado

```
int[] num = {2, 4, 6, 7, 9, 1}; boolean contenido = Arrays.as
List(num).contains(1);
```

Otras operaciones comunes con arrays son:

Para poder ordenar un array simplemente hay que ejecutar el método sort

```
int[] numeros = {4, 1, 2, 6, 8, 3, 4, 89, 67, 32, 12}; Syste
m.out.println("Antes de la organizacion:"); for (int temp : n
umeros) { System.out.println(temp); } Arrays.sort(numeros); S
ystem.out.println("Después de la organizacion:"); for (int te
mp : numeros) { System.out.println(temp); }
```

Para poder copiar un array, creando uno nuevo y definiendo una nueva longitud se ejecuta el método copyOf:

```
int[] numeros = {4, 1, 2, 6, 8, 3, 4, 89, 67, 32, 12}; Syste
m.out.println(numeros.length); int[] numerosNuevos = Arrays.c
opyOf(numeros, 20); System.out.println(numerosNuevos.length);
System.out.println(numerosNuevos[19]);
```

Clonar un array

Para clonar arrays se utiliza el método clone

```
int[] numeros = {4, 1, 2, 6, 8, 3, 4, 89, 67, 32, 12}; int[]
numerosNuevos = numeros.clone(); System.out.println(numerosNu
evos.length);
```

Comparar arrays completos

Para comparar arrays se ejecuta el método equal

```
int[] numeros = {4, 1, 2, 6, 8, 3, 4, 89, 67, 32, 12}; int[]
numerosDos = {6, 7, 8, 2, 123, 53, 231, 23}; boolean iguales
= Arrays.equals(numeros, numerosDos); System.out.println(igua
les); int[] numerosTres = {4, 1, 2, 6, 8, 3, 4, 89, 67, 32, 1
2}; int[] numerosCuatro = {4, 1, 2, 6, 8, 3, 4, 89, 67, 32, 1
2}; boolean igualesDos = Arrays.equals(numerosTres, numerosCu
atro); System.out.println(igualesDos);
```