Programas con arrays.

Algunos puntos importantes acerca de las matrices, arreglos o arrays de Java.

- En Java, todas las matrices se asignan dinámicamente. (Se analiza a continuación)
- Como las matrices/arrays son objetos en Java, cada array tiene asociado una variable de instancia de longitud (length) que contiene la cantidad de elementos que la matriz puede contener. (En otras palabras, length contiene el tamaño de la matriz.)
- Una variable array en Java se declara como otras variables con corchetes [] después del tipo de datos.
- Las variables en el array están ordenadas y cada una tiene un índice que comienza desde 0.
- El array Java también se puede usar como un campo estático, una variable local o un parámetro de método.
- El tamaño de un array debe especificarse mediante un valor int y no, long o short.
- La superclase directa de un tipo de array es Object.
- Cada tipo de array implementa las interfaces Cloneable y java.io.Serializable.
- El array puede contener tipos de datos primitivos así como también objetos de una clase según la definición del array. En el caso de los tipos de datos primitivos, los valores reales se almacenan en ubicaciones de memoria contigua. En el caso de los objetos de una clase, los objetos reales se almacenan en heap.

1. Qué es un Array en Java

Una array o arreglo es una colección de variables del mismo tipo, a la que se hace referencia por un nombre común. En Java, los arrays pueden tener una o más dimensiones, aunque el array unidimensional es el más común.

Los arrays se usan para una variedad de propósitos porque ofrecen un medio conveniente de agrupar variables relacionadas. Por ejemplo, puede usar una matriz para mantener un registro de la temperatura alta diaria durante un mes, una lista de promedios de precios de acciones o una lista de tu colección de libros de programación.

La ventaja principal de un array es que organiza los datos de tal manera que puede ser manipulado fácilmente. Por ejemplo, si tiene un array que contiene los ingresos de un grupo seleccionado de hogares, es fácil calcular el ingreso promedio haciendo un ciclo a través del array. Además, los arrays organizan los datos de tal manera que se pueden ordenar fácilmente.

Aunque los arrays en Java se pueden usar como matrices en otros lenguajes de programación, tienen un atributo especial: se implementan como objetos. Este hecho es una de las razones por las que la discusión de los arrays se pospuso hasta que se introdujeron los objetos. Al implementar arrays como objetos, se obtienen varias ventajas importantes, una de las cuales es que los arrays no utilizados pueden ser recolectados.

2. Arrays unidimensionales

Un array unidimensional es una lista de variables relacionadas. Tales listas son comunes en la programación. Por ejemplo, puede usar un array unidimensional para almacenar los números de cuenta de los usuarios activos en una red. Otro array podría usarse para almacenar los promedios de bateo actuales para un equipo de béisbol.

La forma general de declarar un arreglo unidimensional es:

```
tipo nombre-array[];
tipo [] nombre-array;
```

La declaración de un array tiene dos componentes: el tipo y el nombre.

tipo declara el tipo de elemento del array. El tipo de elemento determina el tipo de datos de cada elemento que comprende la matriz. Al igual que la matriz de tipo int, también podemos crear una matriz de otros tipos de datos primitivos como char, float, double..etc o tipo de datos definido por el usuario (objetos de una clase). Por lo tanto, el tipo de elemento para la matriz determina el tipo de datos que la matriz contendrá.

Ejemplo:

```
// ambas son declaraciones válidas
int intArray[];
int[] intArray;

//Tipo de datos primitivos

byte byteArray[];
short shortArray[];
boolean booleanArray[];
long longArray[];
float floatArray[];
double doubleArray[];
char charArray[];

//Tipos de datos definidos por el usuario

// una serie de referencias a objetos de
// la clase MyClass (una clase creada por
// el usuario)

MyClass myClassArray[];

Object[]  ao,        // array de Object
Collection[] ca;  // array de Collection
```

Aunque la primera declaración anterior establece el hecho de que intArray es una variable de matriz, en realidad no existe una matriz. Simplemente le dice al compilador que esta variable

(intArray) contendrá una matriz del tipo entero. Para vincular intArray con una matriz física real de enteros, debe asignar una usando new y asignarlo a intArray. Ya veremos…

2.1. Instanciando un array en Java

Cuando un array se declara, solo se crea una referencia del array. Para realmente crear o dar memoria al array (a partir de aquí solo mencionaré a array, y no matriz o arreglo), puede crear un array de la siguiente manera:

```
nombre-array = new tipo ;
```

- tipo especifica el tipo de datos que se asignará
- tamaño especifica el número de elementos en el array
- nombre-array es el nombre de la variable del array vinculado al mismo.
- Es decir, para usar new para asignar un array, debe especificar el tipo y la cantidad de elementos a asignar.

Ejemplo:

```
int intArray[]; //declarando un array
intArray = new int; // asignando memoria al array
```

o

```
int[] intArray = new int; // combinando ambas declaraciones en una
```

Nota:

Los elementos en la matriz asignada por se inicializarán automáticamente a cero (para tipos numéricos), false (para booleano) o null (para tipos de referencia).

Obtener un array es un proceso de dos pasos. Primero, debe declarar una variable del tipo de array deseado. En segundo lugar, debe asignar la memoria que mantendrá el array, usar y asignarla a la variable del array. Por lo tanto, en Java, todos los arrays se asignan dinámicamente.

2.2. Array Literal

En una situación en la que ya se conoce el tamaño y los elementos del array, se pueden usar literales del array.

```
int[] intArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 };
 // Declarando un array literal
```

La longitud de este array determina la longitud del array creado.

No es necesario escribir new int[] en las últimas versiones de Java

2.3. Accediendo a los elementos del Array usando el bucle for

A cada elemento del array se accede a través de su índice. El índice comienza con 0 y termina en (tamaño total del array) -1. Se puede acceder a todos los elementos de la matriz usando el bucle for en Java.

```
//acceder a los elementos del array
 for (int i = 0; i < arr.length; i++)
 System.out.println("Elemento en el índice " + i + " : "+ arr);
```

Ejemplo:

```
// Programa Java para ilustrar la creación de un array de enteros,
// coloca algunos valores en la matriz, e imprime cada valor
 class DemoArray
{
    public static void main (String[] args)
    {
      // declara un array de enteros.
      int[] arr;
      // asignando memoria para 5 enteros.
      arr = new int;
      // inicializa el primer elemento del array
      arr = 10;
      // inicializa el segundo elemento del array
      arr = 20;
      // y así...
      arr = 30;
      arr = 40;
      arr = 50;

      // accediendo a los elementos del array
      for (int i = 0; i < arr.length; i++)
         System.out.println("Elemento en el índice " + i +
                                    " : "+ arr);
    }
}
```

Salida:

```
Elemento en el índice 0 : 10
```

```
Elemento en el índice 1 : 20
Elemento en el índice 2 : 30
Elemento en el índice 3 : 40
Elemento en el índice 4 : 50
```

3. Uso de length en Arrays

Debido a que los arreglos se implementan como objetos, cada array tiene asociado una variable de instancia de longitud (length) que contiene la cantidad de elementos que el array puede contener. (En otras palabras, length contiene el tamaño del array.) Aquí hay un programa que demuestra esta propiedad:

```
// Demostrando el uso de length en Arrays
class DemoArray
{
    public static void main(String args[])
    {
        int lista[]= new int ;
        int num[]={1,2,3};
        int tabla[][]={
                {1,2,3},
                {4,5},
                {6,7,8,9}
        };
        System.out.println("Longitud de lista: "+lista.length);
        System.out.println("Longitud de num: " +num.length);
        System.out.println("Longitud de tabla: "+tabla.length);
        System.out.println("Longitud de tabla: " +tabla.length);
        System.out.println("Longitud de tabla: " +tabla.length);
        System.out.println("Longitud de tabla: " +tabla.length);
        //Usando length para inicializar lista
        for (int i=0; i < lista.length; i++)
            lista=i*i;
        System.out.print("La lista es: ");
        //Ahora usamos length para mostrar lista
        for (int i=0; i < lista.length; i++)
            System.out.print(lista+ " ");
        System.out.println();
    }
```

```
}
```

Salida:

```
Longitud de lista: 10
Longitud de num: 3
Longitud de tabla: 3
Longitud de tabla: 3
Longitud de tabla: 2
Longitud de tabla: 4
La lista es: 0 1 4 9 16 25 36 49 64 81
```

Fuente: https://javadesdecero.es/

Class Arrays

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Arrays.html

https://www.geeksforgeeks.org/array-class-in-java/

equals()

Compare cadenas para averiguar si son iguales:

```
String myStr1 = "Hello";

String myStr2 = "Hello";

String myStr3 = "Another String";

System.out.println(myStr1.equals(myStr2)); // Returns true because
they are equal

System.out.println(myStr1.equals(myStr3)); // false
```

Diferencia entre el método == y .equals () en Java

En general, tanto **.equals()** como el operador "==" en Java se utilizan para comparar objetos para verificar la igualdad, pero estas son algunas de las diferencias entre los dos:

- La principal diferencia entre el método **.equals()** y el operador == es que uno es un método y el otro es el operador.
- Podemos usar operadores == para la comparación de referencias (comparación de direcciones) y el método **.equals()** para la comparación de contenido. En palabras simples, == comprueba si ambos objetos apuntan a la misma ubicación de memoria, mientras que **.equals()** evalúa la comparación de valores en los objetos.

Si una clase no anula el método equals, entonces de forma predeterminada utiliza el método equals (Object o) de la clase principal más cercana que ha anulado este método.

Ejemplo de codificación:

```
// Java program to understand
// the concept of == operator
public class Test {
    public static void main(String[] args)
    {
        String s1 = "HELLO";
        String s2 = "HELLO";
        String s3 =  new String("HELLO");
        System.out.println(s1 == s2); // true
        System.out.println(s1 == s3); // false
        System.out.println(s1.equals(s2)); // true
        System.out.println(s1.equals(s3)); // true
    }
}
```

Java.util.Arrays.equals() in Java

Syntax:

public static boolean equals(int[] a, int[] a2)

Parameters:

a - one array to be tested for equality

a2 - the other array to be tested for equality

Returns:

true if the two arrays are equal

Other Variants:

public static boolean equals(byte[] a, byte[] a2)

public static boolean equals(short[] a, short[] a2)

public static boolean equals(long[] a, long[] a2)

public static boolean equals(float[] a, float[] a2)

public static boolean equals(double[] a, double[] a2)

public static boolean equals(char[] a, char[] a2)

public static boolean equals(boolean[] a, boolean[] a2)

public static boolean equals(Object[] a, Object[] a2)

```
// Java program to demonstrate working of Arrays.equals()

import java.util.Arrays;

public class ArrayEqualDemo
{
    public static void main(String[] args)
    {
        // Let us create different integers arrays
```

```
        int[] arr1 = new int [] {1, 2, 3, 4};
        int[] arr2 = new int [] {1, 2, 3, 4};
        int[] arr3 = new int [] {1, 2, 4, 3};

        System.out.println("is arr1 equals to arr2 : " +
                                Arrays.equals(arr1, arr2));
        System.out.println("is arr1 equals to arr3 : " +
                                Arrays.equals(arr1, arr3));
    }
}


// Java program to demonstrate working of Arrays.equals()
// for user defined objects.

import java.util.Arrays;


public class ArrayEqualDemo
{
    public static void main (String[] args)
    {
        Student [] arr1 = {new Student(111, "bbbb", "london"),
                           new Student(131, "aaaa", "nyc"),
                           new Student(121, "cccc", "jaipur")};

        Student [] arr2 = {new Student(111, "bbbb", "london"),
                           new Student(131, "aaaa", "nyc"),
                           new Student(121, "cccc", "jaipur")};

        Student [] arr3 = {new Student(111, "bbbb", "london"),
                           new Student(121, "dddd", "jaipur"),
                           new Student(131, "aaaa", "nyc"),
                           };

        System.out.println("is arr1 equals to arr2 : " +
                                    Arrays.equals(arr1, arr2));
        System.out.println("is arr1 equals to arr3 : " +
                                    Arrays.equals(arr1, arr3));
    }
}

// A class to represent a student.
class Student
{
    int rollno;
    String name, address;

    // Constructor
    public Student(int rollno, String name,
                                String address)
    {
        this.rollno = rollno;
        this.name = name;
        this.address = address;
    }

    @Override
    public boolean equals(Object obj) {
```

```
        // typecast obj to Student so that we can compare students
        Student s = (Student) obj;

        return this.rollno == s.rollno && this.name.equals(s.name)
                              && this.address.equals(s.address);
    }
}
```

Output:

is arr1 equals to arr2 : true

is arr1 equals to arr3 : false

Arrays.deepToString() in Java

Devuelve una representación de cadena del "contenido profundo" de la matriz especificada. Si la matriz contiene otras matrices como elementos, la representación de cadena contiene su contenido y así sucesivamente. Este método está diseñado para convertir matrices multidimensionales en cadenas. El método simple toString () funciona bien para arreglos simples, pero no para arreglos multidimensionales. Este método está diseñado para convertir matrices multidimensionales en cadenas.

Syntax:

public static String deepToString(Object[] arr)

arr - An array whose string representation is needed

This function returns string representation of arr[].

It returns "null" if the specified array is null.

Example:

```
Let us suppose that we are making a 2-D array of
3 rows and 3 column.
    2   3   4
    5   6   7
    2   4   9


If use deepToString() method to print the 2-D array,
we will get string representation as :-
[[2,3,4], [5,6,7], [2,4,9]]
```

Printing multidimensional Array:

```
// A Java program to print 2D array using deepToString()
```

```
import java.util.Arrays;

public class GfG
{
    public static void main(String[] args)
    {
        // Create a 2D array
        int[][] mat = new int[2][2];
        mat[0][0] = 99;
        mat[0][1] = 151;
        mat[1][0] = 30;
        mat[1][1] = 5;

        // print 2D integer array using deepToString()
        System.out.println(Arrays.deepToString(mat));
    }
}
```

Output:

[[99, 151], [30, 5]]

toString() vs deepToString()

toString () funciona bien para matrices unidimensionales, pero no funciona para matrices multidimensionales.

```
// Java program to demonstrate that toString works if we
// want to print single dimensional array, but doesn't work
// for multidimensional array.
import java.util.Arrays;
public class Deeptostring
{
    public static void main(String[] args)
    {
        // Trying to print array of strings using toString
        String[] strs = new String[] {"practice.geeksforgeeks.org",
                                      "quiz.geeksforgeeks.org"
                                     };
        System.out.println(Arrays.toString(strs));

        // Trying to print multidimensional array using
        // toString
        int[][] mat = new int[2][2];
        mat[0][0] = 99;
        mat[0][1] = 151;
        mat[1][0] = 30;
        mat[1][1] = 50;
        System.out.println(Arrays.toString(mat));
    }
}
```

Output:

[practice.geeksforgeeks.org, quiz.geeksforgeeks.org]

[[I@15db9742, [I@6d06d69c]

deepToString () funciona tanto para unidimensionales como para multidimensionales, pero no funciona para una matriz unidimensional de primitivas

```
// Java program to demonstrate that deepToString(strs))
// works for single dimensional arrays also, but doesn't
// work single dimensional array of primitive types.
import java.util.Arrays;
public class Deeptostring
{
    public static void main(String[] args)
    {
        String[] strs = new String[] {"practice.geeksforgeeks.org",
                                      "quiz.geeksforgeeks.org"
                                     };
        System.out.println(Arrays.deepToString(strs));

        Integer []  arr1 = {10, 20, 30, 40};
        System.out.println(Arrays.deepToString(arr1));

        /* Uncommenting below code would cause error as
           deepToString() doesn't work for primitive types
        int []  arr2 = {10, 20, 30, 40};
        System.out.println(Arrays.deepToString(arr2));   */
    }
}
```

Output:

[practice.geeksforgeeks.org, quiz.geeksforgeeks.org]

[10, 20, 30, 40]

Arrays.fill() in Java

Este método asigna el valor del tipo de datos especificado a cada elemento del rango especificado de la matriz especificada.

Syntax:

// Makes all elements of a[] equal to "val"

public static void fill(int[] a, int val)

// Makes elements from from_Index (inclusive) to to_Index

// (exclusive) equal to "val"

public static void fill(int[] a, int from_Index, int to_Index, int val)

This method doesn't return any value.

Exceptions it Throws:

IllegalArgumentException - if from_Index > to_Index

ArrayIndexOutOfBoundsException - if from_Index a.length

```
// Java program to fill a subarray of given array
import java.util.Arrays;

public class Main
{
```

```
    public static void main(String[] args)
    {
        int ar[] = {2, 2, 1, 8, 3, 2, 2, 4, 2};

        // To fill complete array with a particular
        // value
        Arrays.fill(ar, 10);
        System.out.println("Array completely filled" +
                  " with 10\n" + Arrays.toString(ar));
    }
}
```

Output:

Array completely filled with 10

[10, 10, 10, 10, 10, 10, 10, 10, 10]

Podemos llenar parte de la matriz.

```
// Java program to fill a subarray array with
// given value.
import java.util.Arrays;

public class Main
{
    public static void main(String[] args)
    {
        int ar[] = {2, 2, 2, 2, 2, 2, 2, 2, 2};

        // Fill from index 1 to index 4.
        Arrays.fill(ar, 1, 5, 10);

        System.out.println(Arrays.toString(ar));
    }
}
```

Output:

[2, 10, 10, 10, 10, 2, 2, 2, 2]

Podemos llenar una matriz multidimensional

Podemos usar un bucle para llenar una matriz multidimensional.

1) Rellenar matriz 2D

```
// Java program to fill a multidimensional array with
// given value.
import java.util.Arrays;

public class Main
{
    public static void main(String[] args)
    {
        int [][]ar = new int [3][4];

        // Fill each row with 10.
        for (int[] row : ar)
            Arrays.fill(row, 10);

        System.out.println(Arrays.deepToString(ar));
```

```
    }
}
```
Output:

[[10, 10, 10, 10], [10, 10, 10, 10], [10, 10, 10, 10]]

2) Fill 3D Array

```
// Java program to fill a multidimensional array with
// given value.

import java.util.Arrays;

class GFG {

    public static void main(String[] args) {
        int[][][] ar = new int[3][4][5];

        // Fill each row with -1.
        for (int[][] row : ar) {
            for (int[] rowColumn : row) {
                Arrays.fill(rowColumn, -1);
            }
        }

        System.out.println(Arrays.deepToString(ar));
    }
}
```
Output:

[[[-1, -1, -1, -1, -1], [-1, -1, -1, -1, -1], [-1, -1, -1, -1, -1], [-1, -1, -1, -1, -1]], [[-1, -1, -1, -1, -1], [-1, -1, -1, -1, -1], [-1, -1, -1, -1, -1], [-1, -1, -1, -1, -1]], [[-1, -1, -1, -1, -1], [-1, -1, -1, -1, -1], [-1, -1, -1, -1, -1], [-1, -1, -1, -1, -1]]]

Arrays.sort() in Java

Una clase de matriz es una clase que contiene métodos estáticos que se utilizan con matrices para buscar, ordenar, comparar, insertar elementos o devolver una representación de cadena de una matriz. en una matriz. Entonces, especifiquemos las funciones primero y luego discutiremos las mismas. Son los siguientes que han estado presentes en la clase java.util.Arrays. Aquí discutiremos diferentes gráficos usando el método sort () de la clase Arrays.

El método Arrays.sort () consta de dos variaciones, una en la que no pasamos ningún argumento en el que clasifica la matriz completa, ya sea una matriz de enteros o una matriz de caracteres, pero si se supone que debemos ordenar una parte específica utilizando este método de la clase Arrays, entonces lo sobrecargamos y pasamos el índice inicial y el último a la matriz.

Syntax: sort() Method

Arrays.sort();

Syntax: Overloaded sort() Method

public static void sort(int[] arr, int from_Index, int to_Index) ;

Parameters: It takes three parameters as can be perceived from the syntax which are as follows:

The array to be sorted

The index of the first element, inclusive, to be sorted (Referred to as from_index)

The index of the last element, exclusive, to be sorted (Referred to as last_index)

Return Type: It does not return any value.

Example 1:

```
// Java Program to Sort Array of Integers
// by Default Sorts in an Ascending Order
// using Arrays.sort() Method

// Importing Arrays class from the utility class
import java.util.Arrays;

// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Custom input array
        int[] arr = { 13, 7, 6, 45, 21, 9, 101, 102 };

        // Applying sort() method over to above array
        // by passing the array as an argument
        Arrays.sort(arr);

        // Printing the array after sorting
        System.out.println("Modified arr[] : %s",
                           Arrays.toString(arr));
    }
}
```

Output:

Modified arr[] : [6, 7, 9, 13, 21, 45, 101, 102]

Example 2

```
// Java program to Sort a Subarray in Array
// Using Arrays.sort() method

// Importing Arrays class from java.util package
import java.util.Arrays;

// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Custom input array
        // It contains 8 elements as follows
        int[] arr = { 13, 7, 6, 45, 21, 9, 2, 100 };

        // Sort subarray from index 1 to 4, i.e.,
        // only sort subarray {7, 6, 45, 21} and
```

```
        // keep other elements as it is.
        Arrays.sort(arr, 1, 5);

        // Printing the updated array which is
        // sorted after 2 index inclusive till 5th index
        System.out.println("Modified arr[] : %s",
                           Arrays.toString(arr));
    }
}
```

Output:

Modified arr[] : [13, 6, 7, 21, 45, 9, 2, 100]

Example 3

```
// Java program to Sort a Subarray in Descending order
// Using Arrays.sort()

// Importing Collections class and arrays classes
// from java.util package
import java.util.Arrays;
import java.util.Collections;

// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Note that we have Integer here instead of
        // int[] as Collections.reverseOrder doesn't
        // work for primitive types.
        Integer[] arr = { 13, 7, 6, 45, 21, 9, 2, 100 };

        // Sorts arr[] in descending order using
        // reverseOrder() method of Collections class
        // in Array.sort() as an argument to it
        Arrays.sort(arr, Collections.reverseOrder());

        // Printing the array as generated above
        System.out.println("Modified arr[] : %s",
                           Arrays.toString(arr));
    }
}
```

Output:

Modified arr[] : [100, 45, 21, 13, 9, 7, 6, 2]

Example 4

```
// Java program to sort an array of strings
// in ascending and descending orders (Alphabetical order)
// Using Arrays.sort()

// Importing arrays and Collections class
// from java.util class
import java.util.Arrays;
import java.util.Collections;
```

```
// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Custom input string
        String arr[] = { "practice.geeksforgeeks.org",
                         "quiz.geeksforgeeks.org",
                         "code.geeksforgeeks.org" };

        // Sorts arr[] in ascending order
        Arrays.sort(arr);
        System.out.println("Modified arr[] : \n%s\n\n",
                           Arrays.toString(arr));

        // Sorts arr[] in descending order
        Arrays.sort(arr, Collections.reverseOrder());

        // Lastly printing the above array
        System.out.println("Modified arr[] : \n%s\n\n",
                           Arrays.toString(arr));
    }
}
```

Output:

Modified arr[] :

Modified arr[] :

[quiz.geeksforgeeks.org, practice.geeksforgeeks.org, code.geeksforgeeks.org]

Ahora, por último, implementaremos el método sort () al máximo porque aquí declararemos nuestra propia creteria definida con la ayuda de la interfaz Comparator.

Example 5

```
// Java program to demonstrate Working of
// Comparator interface

// Importing required classes
import java.io.*;
import java.lang.*;
import java.util.*;

// Class 1
// A class to represent a student.
class Student {
    int rollno;
    String name, address;

    // Constructor
    public Student(int rollno, String name, String address)
    {
        // This keyword refers to current object itself
        this.rollno = rollno;
        this.name = name;
        this.address = address;
```

```
    }

    // Used to print student details in main()
    public String toString()
    {
        return this.rollno + " " + this.name + " "
            + this.address;
    }
}

// Class 2
// Helper class extending Compatator interface
class Sortbyroll implements Comparator<Student> {
    // Used for sorting in ascending order of
    // roll number
    public int compare(Student a, Student b)
    {
        return a.rollno - b.rollno;
    }
}

// Class 3
// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        Student[] arr
            = { new Student(111, "bbbb", "london"),
                new Student(131, "aaaa", "nyc"),
                new Student(121, "cccc", "jaipur") };

        System.out.println("Unsorted");

        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);

        // Sorting on basic as per class 1 created
        // (user-defined)
        Arrays.sort(arr, new Sortbyroll());

        System.out.println("\nSorted by rollno");

        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);
    }
}
```

Output:

Unsorted

111 bbbb london

131 aaaa nyc

121 cccc jaipur

Sorted by rollno

111 bbbb london

121 cccc jaipur

131 aaaa nyc

Nota: Existe una ligera diferencia entre Arrays.sort () y Collections.sort (). Arrays.sort funciona para matrices que también pueden ser de tipo de datos primitivos. Collections.sort () funciona para colecciones de objetos como ArrayList, LinkedList, etc.

Fuente: https://www.geeksforgeeks.org/

toString()

Class System

https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/lang/System.html

arraycopy()

Junit

https://testingandlearning.home.blog/2019/01/30/how-to-use-junit-with-netbeans/

Funciones predeterminadas de Java.

Funciones definidas por el usuario.

Variables locales y globales.

Parámetros y argumentos.

Creación e inclusión de librerías con funciones definidas por el usuario.

Funciones como procedimientos o subrutinas.

Codificación de programas con funciones.