

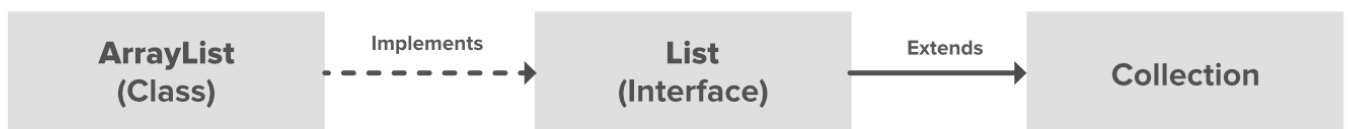
Índice

- [Índice](#)
- [Definición y creación de un ArrayList](#)
- [Métodos y propiedades generales](#)
- [Añadir datos a la colección](#)
 - [Añadir elementos desde el constructor](#)
 - [Añadir elementos a partir de otras colecciones](#)
 - [Añadir elementos mediante código](#)
 - [Eliminar elementos mediante código](#)
- [Recorrer la colección](#)
- [Búsqueda de elementos](#)
- [Obtención de subcolecciones](#)
- [Ordenación de elementos](#)
 - [Con funciones de Collection](#)
 - [Con expresiones lambda](#)
 - [Con API Stream](#)
- [Webgrafía](#)

Definición y creación de un ArrayList

La clase ArrayList en Java permite almacenar datos en memoria de forma similar a los Arrays con la ventaja de que el numero de elementos que almacena lo hace de forma **dinámica**, es decir, que **no es necesario declarar su tamaño como pasa con los Arrays**. Los elementos pueden añadirse o eliminarse según necesidad.

La clase ArrayList implementa la interface List, la cual extiende la interface `java.util.Collection`.



Para crear un ArrayList, podemos importar o bien solo la clase `java.util.ArrayList` o todo el paquete `java.util`. Un ArrayList se puede declarar de diferentes elementos u objetos (String,

Integer, float, Boolean, Object...). Además **existen 3 sobrecargas del constructor** que veremos a continuación.

Para el ejemplo que vamos a seguir en este manual, utilizaremos el IDE Netbeans y declararemos un `ArrayList` de objetos *Coche* al que llamaremos *garaje* y en el que almacenaremos objetos de la clase *Coche*, la cual deberemos de crear también con el siguiente código:

```

// Creamos una clase Coche que usaremos para el ejemplo y
// se guardará en Coche.java
public class Coche
{
    //Atributos de la clase
    private String marca;
    private String modelo;
    private String version;
    private double precio;

    //Constructor con el mismo nombre de la clase
    public Coche(String marca,
                  String modelo,
                  String version,
                  double precio) {
        this.marca = marca;
        this.modelo = modelo;
        this.version = version;
        this.precio = precio;
    }

    //Métodos de la clase
    public String getMarca() {
        return marca;
    }

    public String getModelo() {
        return modelo;
    }

    public String getVersion() {
        return version;
    }

    public double getPrecio() {
        return precio;
    }

    //String de la clase
    @Override
    public String toString() {
        return "Coche: " + marca + " / "
            + modelo + " / "
            + version + " / "
            + precio;
    }
}

```

Una vez creada la clase *Coche*, declaramos el ArrayList *garaje* y le añadimos 3 coches de la siguiente manera:

```
// SOBRECARGA 1

// Importamos la clase ArrayList
import java.util.ArrayList;

// Creamos 3 objetos Coche
Coche seatArosa = new Coche("Seat", "Arosa", "2.0", 25000);
Coche fordFocus = new Coche("Ford", "Focus", "1600", 21000);
Coche audiA4 = new Coche("Audi", "A4", "v6", 45000);

// Creamos un ArrayList garaje al que le añadimos los 3 coches
ArrayList<Coche> garaje = new ArrayList<Coche>();
```

Al hacerlo de esta manera, estamos creando un ArrayList sin longitud, es decir, es una lista vacía, aunque por defecto tiene una capacidad de 10. Esto no quiere decir que sólo podamos añadir 10 objetos, sino que se dimensiona con una capacidad de 10 para optimizar recursos (aunque la lista está vacía, si se consultara el tamaño de la lista sería 0). Si más adelante queremos añadir objetos, el ArrayList es capaz de redimensionarse.

Vamos ahora a añadir los 3 coches creados y a mostrar que están en la lista:

```
garaje.add(seatArosa); // Añadimos elementos a la lista garaje
garaje.add(fordFocus);
garaje.add(audiA4);

for (Coche coche : garaje) // Recorremos el ArrayList y mostramos sus objetos
{
    System.out.println(coche);
}
```

Si por el contrario, quisieramos darle una longitud inicial al ArrayList, la declaración sería la siguiente:

```
// SOBRECARGA 2

// Creamos el objeto ArrayList con una capacidad de 30
ArrayList<Coche> garaje = new ArrayList<Coche>(30);
```

Si sabemos de antemano cual va a ser la capacidad que se necesita de ArrayList, es mejor crearlo de esta manera, ya que aunque un ArrayList es una lista dinámica y permite el crecimiento agregando elementos, conlleva un coste muy alto de recursos cada vez que se redimensiona.

Existe una tercera sobrecarga del constructor `ArrayList(Collection<? extends E> c)` donde podemos crear/aumentar el ArrayList pasándole una colección específica. De esta manera, el ArrayList se generará con los valores de dicha colección según se los vaya pasando el iterador. Por ejemplo:

```
// SOBRECARGA 3

// Creamos una lista parking a partir de la lista garaje
ArrayList<Coche> parking = new ArrayList<Coche>(garaje);

// Mostramos la lista parking donde se ve que tiene los
// mismos coches que el ArrayList garaje
for (Coche coche : parking) {
    System.out.println(coche);
}
```

Métodos y propiedades generales

En la documentación oficial de Oracle se pueden encontrar todos los métodos de la

Class `ArrayList<E>` (<https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#ArrayList-java.util.Collection->). En este manual indicamos los más utilizados que son:

```
// Añadir un objeto Coche al final del ArrayList
garaje.add(fordFocus);

// Añade el elemento al ArrayList en la posición 'n'
garaje.add(3, setArosa);

// Devuelve el numero de elementos del ArrayList
garaje.size();

// Devuelve el elemento que esta en la posición '2' del ArrayList
garaje.get(2);

// Comprueba si existe del objeto Coche que se le pasa como parametro
garaje.contains(audiA4);

// Devuelve la posición del primer objeto Coche fordFocus encontrado en el ArrayList
garaje.indexOf(fordFocus);

// Devuelve la posición del último objeto Coche fordFocus en el ArrayList
garaje.lastIndexOf(fordFocus);

// Borra el elemento de la posición '2' del ArrayList
garaje.remove(2);

// Borra el primer objeto Coche fordFocus encontrado en el ArrayList que se le pasa como parametro
garaje.remove(fordFocus);

//Borra todos los elementos de ArrayList
garaje.clear();

// Devuelve True si el ArrayList esta vacio. Sino Devuelve False
garaje.isEmpty();

// Pasa el ArrayList a un Array
Object[] array = garaje.toArray();
```

Añadir datos a la colección

Aunque previamente hemos mostrado algunos métodos para añadir y eliminar datos, vamos a describirlos con más detalle en este epígrafe.

Añadir elementos desde el constructor

La clase ArrayList es una colección que no permite desde los constructores 1 `public ArrayList()` y 2 `public ArrayList(int initialCapacity)` añadir elementos en su creación.

Añadir elementos a partir de otras colecciones

Sólo a través del constructor 3 `public ArrayList(Collection<? extends E> c)` es posible, tal y como hemos visto en el ejemplo anterior:

```
// Creamos una lista parking a partir de la lista garaje
ArrayList<Coche> parking = new ArrayList<Coche>(garaje);

// Mostramos la lista parking donde se ve que tiene los mismos coches
// que el ArrayList garaje
for (Coche coche : parking) {
    System.out.println(coche);
}
```

Añadir elementos mediante código

Aun con todo lo anterior, lo interesante de los ArrayList es su **capacidad de adaptación a añadir nuevos elementos a la lista**. Por tanto, sea cual sea la forma en la que hemos creado el ArrayList, podemos añadir elementos a la misma con los siguiente métodos:

`.add(E e)`

El método `.add(E e)` añade el elemento que indiquemos en su argumento al final del ArrayList e incrementa su longitud en +1:

```
// Revisamos la cantidad de elementos del ArrayList garaje. Nos devolverá valor = 3
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Creamos un nuevo objeto Coche
Coche opelCorsa = new Coche("Opel", "Corsa", "2000", 27000);

// Añadimos el nuevo Coche al ArrayList
garaje.add(opelCorsa);

// Revisamos de nuevo la cantidad de elementos del ArrayList
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Recorremos el ArrayList para que nos muestre sus elementos
for (Coche coche : garaje) {
    System.out.println(coche);
}
```

`.add(int index, E element)`

El método `.add(int index, E element)` es una variante del anterior en donde podemos especificar en el primer argumento la posición en la que queremos insertar el nuevo elemento dentro del

ArrayList:

```
// Revisamos la cantidad de elementos del ArrayList garaje. Nos devolverá valor = 4
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Recorremos el ArrayList para que nos muestre sus elementos
for (Coche coche : garaje) {
    System.out.println(coche);
}

// Creamos un nuevo objeto Coche
Coche hondaAccord = new Coche("Honda", "Accord", "1800", 31000);

// Añadimos el nuevo Coche al ArrayList en la posición 2
garaje.add(2, hondaAccord);

// Revisamos de nuevo la cantidad de elementos del ArrayList para ver que se ha incrementado en
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Recorremos el ArrayList para que nos muestre sus elementos y comprobar
// que en la posición 2 se encuentra el nuevo coche
for (Coche coche : garaje) {
    System.out.println(coche);
}
```

.addAll(Collection<? extends E> c)

El método `.addAll(Collection<? extends E> c)` añade una lista completa de elementos que indiquemos en su argumento al final del ArrayList e incrementa su longitud en la misma cantidad de elementos que tiene la lista del argumento:


```
// Revisamos la cantidad de elementos del ArrayList garaje. Nos devolverá valor = 5
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Revisamos la cantidad de elementos del ArrayList parking. Nos devolverá valor = 3
int cantidadElementosEnArrayList = parking.size();
System.out.println("La cantidad de elementos en el ArrayList parking es: "
    + cantidadElementosEnArrayList);

// Aplicamos el método
garaje.addAll(parking);

// Revisamos de nuevo la cantidad de elementos del ArrayList
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Recorremos el ArrayList para que nos muestre sus elementos
for (Coche coche : garaje) {
    System.out.println(coche);
}
```

.addAll(int index, Collection<? extends E> c)

El método `.addAll(int index, Collection<? extends E> c)` es una variante del anterior en donde podemos especificar en el primer argumento la posición en la que queremos insertar la lista de elementos dentro del ArrayList:

```
// Revisamos la cantidad de elementos del ArrayList garaje. Nos devolverá valor = 8
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Revisamos la cantidad de elementos del ArrayList parking. Nos devolverá valor = 3
int cantidadElementosEnArrayList = parking.size();
System.out.println("La cantidad de elementos en el ArrayList parking es: "
    + cantidadElementosEnArrayList);

// Recorremos el ArrayList para que nos muestre sus elementos
for (Coche coche : garaje) {
    System.out.println(coche);
}

// Aplicamos el método insertando la lista en la posición 2
garaje.addAll(2, parking);

// Revisamos de nuevo la cantidad de elementos del ArrayList
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Recorremos el ArrayList para que nos muestre sus elementos
for (Coche coche : garaje) {
    System.out.println(coche);
}
```

Eliminar elementos mediante código

De la misma manera que añadir elementos a ArrayList es lo que lo hace interesante, también tiene la capacidad de poder eliminarlos. Hay **5 métodos** para eliminar elementos de un ArrayList:

.remove(int index)

El método `.remove(int index)` elimina el elemento que se encuentra en la posición que le indiquemos del ArrayList:

```
// Revisamos la cantidad de elementos del ArrayList garaje. Nos devolverá valor = 11
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Recorremos el ArrayList para que nos muestre sus elementos
for (Coche coche : garaje) {
    System.out.println(coche);
}

// Eliminamos el elemento de la posición 2
garaje.remove(2);

// Revisamos de nuevo la cantidad de elementos del ArrayList
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Recorremos el ArrayList para que nos muestre sus elementos
for (Coche coche : garaje) {
    System.out.println(coche);
}
```

.remove(Object o)

El método `.remove(Object o)` elimina el primer elemento del ArrayList que coincide con el elemento pasado como argumento al método:

```
// Revisamos la cantidad de elementos del ArrayList garaje.
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Recorremos el ArrayList para que nos muestre sus elementos
for (Coche coche : garaje) {
    System.out.println(coche);
}

// Eliminamos el elemento fordFocus
garaje.remove(fordFocus);

// Revisamos de nuevo la cantidad de elementos del ArrayList
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Recorremos el ArrayList para que nos muestre sus elementos
for (Coche coche : garaje) {
    System.out.println(coche);
}
```

.removeAll(Collection<?> c)

El método `.removeAll(Collection<?> c)` elimina del ArrayList todos aquellos elementos que coinciden con los indicados en la lista que pasamos como argumento al método:

```
// Revisamos la cantidad de elementos del ArrayList garaje.
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Recorremos el ArrayList para que nos muestre sus elementos
for (Coche coche : garaje) {
    System.out.println(coche);
}

// Eliminamos la lista parking
garaje.removeAll(parking);

// Revisamos de nuevo la cantidad de elementos del ArrayList
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Recorremos el ArrayList para que nos muestre sus elementos
for (Coche coche : garaje) {
    System.out.println(coche);
}
```

.removeRange(int fromIndex, int toIndex)

El método `.removeRange(int fromIndex, int toIndex)` elimina del ArrayList todos aquellos elementos que se encuentran entre la posición inicial `fromIndex` y la posición final `toIndex`:

```
// Revisamos la cantidad de elementos del ArrayList garaje.
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Añadimos la lista parking para incrementar la cantidad de elementos en ArrayList
garaje.addAll(parking);

// Recorremos el ArrayList para que nos muestre sus elementos
for (Coche coche : garaje) {
    System.out.println(coche);
}

// Eliminamos de la posición 1 a 3
garaje.removeRange(1,3);

// Revisamos de nuevo la cantidad de elementos del ArrayList
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Recorremos el ArrayList para que nos muestre sus elementos
for (Coche coche : garaje) {
    System.out.println(coche);
}
```

.removeIf(Predicate<? super E> filter)

El método `.removeIf(Predicate<? super E> filter)` elimina del ArrayList todos aquellos elementos que cumplen con el predicado (la condición) descrita como argumento en el método e indicada como expresión lambda:

```
// Revisamos la cantidad de elementos del ArrayList garaje.  
int cantidadElementosEnArrayList = garaje.size();  
System.out.println("La cantidad de elementos en el ArrayList garaje es: "  
    + cantidadElementosEnArrayList);  
  
// Añadimos la lista parking para incrementar la cantidad de elementos en ArrayList  
garaje.addAll(parking);  
  
// Recorremos el ArrayList para que nos muestre sus elementos  
for (Coche coche : garaje) {  
    System.out.println(coche);  
}  
  
// Eliminamos aquellos que tengan un precio > 30000  
garaje.removeIf(coche -> (coche.getPrecio() > 30000));  
  
// Revisamos de nuevo la cantidad de elementos del ArrayList  
int cantidadElementosEnArrayList = garaje.size();  
System.out.println("La cantidad de elementos en el ArrayList garaje es: "  
    + cantidadElementosEnArrayList);  
  
// Recorremos el ArrayList para que nos muestre sus elementos  
for (Coche coche : garaje) {  
    System.out.println(coche);  
}
```

Recorrer la colección

Aunque ya hemos visto a lo largo de este manual una forma de recorrer un ArrayList, vamos a explicar en este epígrafe las distintas maneras que tenemos de obtener los datos que contiene el mismo:

Bucle for

El bucle `for` recorre el ArrayList posición a posición. Podemos recorrerlo entre las posiciones que indiquemos o especificar que lo recorra hasta el final con la propiedad `.size()`:

```
// Revisamos la cantidad de elementos del ArrayList garaje.
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Añadimos la lista parking para incrementar la cantidad de elementos en ArrayList
garaje.addAll(parking);

// Recorremos el ArrayList desde la posición 1 a 3
for (int i=1; i<=3; i++) {
    System.out.println(garaje.get(i));
}

// Recorremos el ArrayList completo
for (int i=0; i<garaje.size(); i++) {
    System.out.println(garaje.get(i));
}
```

Método .forEach()

El método `.forEach()` nos permite simplificar el código cuando queremos recorrer por completo el ArrayList:

```
// Revisamos la cantidad de elementos del ArrayList garaje.
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Recorremos todo el ArrayList con una expresión lambda
garaje.forEach((coche) -> System.out.println(coche));
```

Método .iterator()

El método `.iterator()` es una construcción que se utiliza para recorrer colecciones. Es aplicable a ArrayList:

```
// Revisamos la cantidad de elementos del ArrayList garaje.
int cantidadElementosEnArrayList = garaje.size();
System.out.println("La cantidad de elementos en el ArrayList garaje es: "
    + cantidadElementosEnArrayList);

// Generamos el Iterator y lo arrancamos con un bucle while
Iterator<Coche> it = garaje.iterator();
while(it.hasNext())
    System.out.println(it.next());
```

Con expresiones lambda

En el caso de recorrer un ArrayList completo, las expresiones lambda solo se utilizan junto con el método `.forEach()` tal y como hemos visto. La expresión lambda que hay que utilizar es

```
(coche) -> System.out.println(coche)
```

En ella se indica que para cada elemento de la lista (que se le ha nombrado con la variable `coche` pero podría ser cualquier otro nombre) se realice la acción `System.out.println`.

Es en la búsqueda de elementos concretos donde estas expresiones tienen más potencial. Esto lo veremos en el siguiente punto.

Búsqueda de elementos

Con un bucle (for / foreach / Iterator)

Combinando un bucle `for / foreach / Iterator` junto con un condicional `if`, nos permite hacer una búsqueda dentro del ArrayList y cuando salte la condición, realizar la acción que programemos:

```
// Hacemos un bucle for de principio a fin y dentro realizamos un if
for (int i=0; i<garaje.size(); i++) {
    if (coche.getMarca().equals("Ford")) {
        System.out.println(coche);
    }
};
```

Con expresiones lambda

Con la misma funcionalidad que el anterior pero con menos código, existe una variante del método `.foreach()` en la que podemos especificar mediante expresiones lambda una condición que en caso de cumplirse nos permita realizar sobre ella la acción que explicitemos, en este caso es un sencillo `System.out.println`:

```
// Condicionamos la busqueda a mostrar aquellos que sean marca Ford
garaje.forEach((coche) -> {
    if (coche.getMarca().equals("Ford")) {
        System.out.println(coche);
    }
});
```

Con API Stream

Los API Stream nos permiten aplicar la llamada *Programación Funcional* a la búsqueda de un elemento en un ArrayList mediante el uso de `.filter` que se apoya en expresiones lambda:


```
// Condicionamos la búsqueda a mostrar aquellos que sean marca Ford
garaje.Stream().filter(coche -> coche.getMarca().equals("Ford"));
```

Obtención de subcolecciones

Entendemos como subcolección un nuevo ArrayList de menor longitud que el ArrayList original y cuyos elementos se encuentran ordenados (o no) en la misma posición relativa en ambos. Para obtenerlos podemos aplicar lo aprendido en el anterior punto **Búsqueda de elementos** donde buscaremos elemento a elemento y, cuando se cumpla una condición especificada, añadir los elementos encontrados a una nueva colección (o subcolección) tal y como se ha aprendido en **Añadir datos a la colección**.

Ordenación de elementos

Con funciones de Collection

Para la ordenación del ArrayList con la clase **Collections** implementamos la colección de la siguiente manera:





```
// Iniciamos la colección
import java.util.Collections;
```

Esta clase consta exclusivamente de métodos estáticos que operan o devuelven colecciones. Para la ordenación de elementos, utilizaremos el método **.sort()** contenido en la clase **Collections**, el cual implementa 2 sobrecargas:

.sort(List list)

Ordena la lista especificada en orden ascendente, según el orden natural de sus elementos:

```
Collections.sort(garaje);
garaje.forEach((coche) -> System.out.println(coche));
```

  ¡OJO!  Esto no va a funcionar porque el método **.sort()** se puede utilizar de esta manera sólo con tipos primitivos (String, Integer, float...). Para que funcione con un ArrayList de objetos, hemos de utilizar la interfaz **Comparator**. Lo vemos en el siguiente punto 

.sort(List list, Comparator<? super T> c)

Existen clases comparadoras predefinidas que nos permiten ordenar de alguna otra forma los ArrayList de tipos primitivos. Una de ellas muy utilizada es `Collections.reverseOrder()`, la cual permite ordenar de forma descendente un ArrayList. Vamos a hacer un ejemplo fuera de la línea de este manual -ya que este manual se está realizando sobre el objeto `Coche` - para que se vea:

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

lenguajesProgramacion.add("Java");
lenguajesProgramacion.add("C#");
lenguajesProgramacion.add("Go");
lenguajesProgramacion.add("Phyton");
lenguajesProgramacion.add("PHP");

System.out.println("ArrayList no ordenado: "
                    + lenguajesProgramacion);

Collections.sort(lenguajesProgramacion, Collections.reverseOrder());

System.out.println("Array ordenado de forma descendente: " + lenguajesProgramacion);
```

Para ordenar un ArrayList de objetos no primitivos, hemos de crear una clase vacía que implementará la interfaz `Comparator` y en la que diseñaremos el código de comparación de dos objetos. Primero importamos la clase `Comparator` con `import java.util.Comparator;` y a continuación escribimos:

```
public class CompararCoches implements Comparator<Coche>{

    @Override
    public int compare(Coche coche1, Coche coche2){
        if(coche1.getPrecio()<coche2.getPrecio()){
            return -1;
        }else if(coche1.getPrecio()>coche2.getPrecio()){
            return 0;
        }else{
            return 1;
        }
    }
}
```

De esta manera, nos permitirá ordenar el ArrayList `garaje` por precio ascendente de los coches o por cualquier atributo de la clase `Coche`:

```
Collections.sort(garaje, new CompararCoches());
garaje.forEach((coche) -> System.out.println(coche));
```

Con expresiones lambda

El uso de expresiones lambda nos permite simplificar el anterior código sin tener que incluir la clase `CompararCoches` . Esta clase, con una expresión lambda pasaría de esto:

```
public class CompararCoches implements Comparator<Coche>{

    @Override
    public int compare(Coche coche1, Coche coche2){
        if(coche1.getPrecio()<coche2.getPrecio()){
            return -1;
        }else if(coche1.getPrecio()<coche2.getPrecio()){
            return 0;
        }else{
            return 1;
        }
    }
}
```

a esto:

```
(Coche coche1, Coche coche2)->coche1.getPrecio().compareTo(coche2.getPrecio());
```

Por tanto se escribiría el código de esta forma:

```
garaje.sort((Coche coche1, Coche coche2)->coche1.getPrecio().compareTo(coche2.getPrecio()));
```

Y para ordenar por orden inverso, sólo deberíamos de cambiar el orden de los coches:

```
garaje.sort((Coche coche1, Coche coche2)->coche2.getPrecio().compareTo(coche1.getPrecio()));
```

Con API Stream

Ordenar un `ArrayList` con la programación funcional que nos permite API Stream es mucho más sencillo en el caso de un `ArrayList` de objetos, pero debemos hacerlo a una lista auxiliar. Para ello inicializaremos con `.stream()` , ordenaremos con `.sorted()` y escribiremos con `.collect(Collectors.toList())` en la nueva lista:

```
List<Coche> nuevoGaraje = garaje.stream().sorted().collect(Collectors.toList());
nuevoGaraje.forEach((Coche) -> System.out.println(Coche));
```

Interesante indicar dos funcionalidades más: orden invertido con `Comparator.reverseOrder()` y comparar por atributo con `Comparator.comparing(Class::getter)` :

```
// Con Comparator.reverseOrder()
List<Coche> nuevoGaraje = garaje.stream()
                                .sorted(Comparator.reverseOrder())
                                .collect(Collectors.toList());
nuevoGaraje.forEach((Coche) -> System.out.println(Coche));

// Con Comparator.comparing(Class::getter)
List<Coche> nuevoGaraje = garaje.stream()
                                .sorted(Comparator.comparing(Coche::getPrecio))
                                .collect(Collectors.toList());
nuevoGaraje.forEach((Coche) -> System.out.println(Coche));
```

Webgrafía

- <https://www.geeksforgeeks.org/arraylist-in-java/?ref=lbp>
- https://www.w3schools.com/java/java_arraylist.asp
- <https://guru99.es/how-to-use-arraylist-in-java/>
- <https://www.javadevjournal.com/java/java-arraylist/>
- <https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>
- <https://www.discoduroderoer.es/formas-de-ordenar-un-arraylist/>
- <https://www.baeldung.com/java-sorting>
- <https://www.javatpoint.com/how-to-sort-arraylist-in-java>