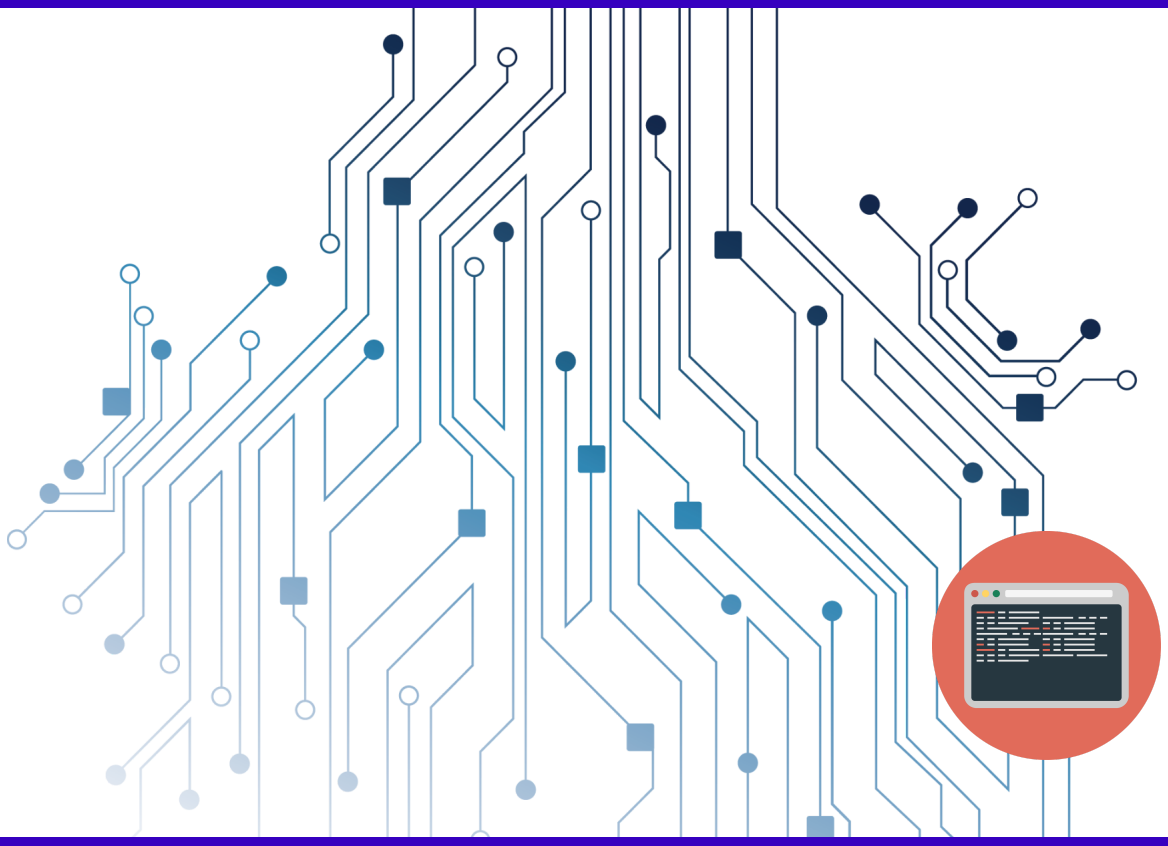
CFGS APLICACIONES MULTIPLATAFORMA - PROGRAMACIÓN

UNIDAD 5

COLECCIONES



DIEGO VALERO ARÉVALO
BASADO EN APUNTES Y EJERCICIOS PROPORCIONADOS POR
Mª CARMEN DÍAZ GONZÁLEZ - IES VIRGEN DE LA PALOMA

U5 - **COLECCIONES**
ÍNDICE

5.1 - COLECCIONES

Las **colecciones** son **objetos que almacenan referencias a otros objetos**, dicho de otra manera, podemos considerarlos **"arrays de objetos"**.

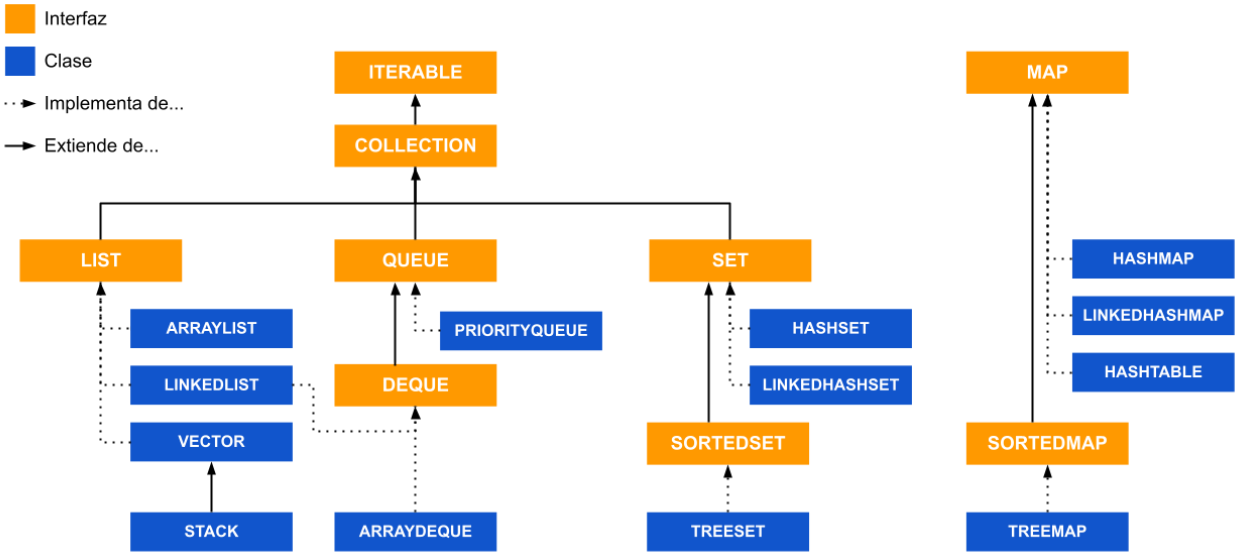
A diferencia de los arrays que conocemos, las colecciones son **dinámicas**, que quiere decir que **no trabajan con un tamaño fijo** y permiten añadir y eliminar objetos durante la ejecución del programa.

Java incluye en el paquete `java.util` un amplio conjunto de clases para la creación y tratamiento de colecciones. Todas ellas proporcionan una serie de métodos para realizar las operaciones básicas sobre una colección, como son añadir objetos, eliminarlos, obtenerlos, localizarlos, iterar a través de la colección,...

En esta unidad veremos las siguientes interfaces y sus derivaciones:

List	Set	Map

De las cuáles puedes ver su relación en el siguiente diagrama:



Como ves, aunque `MAP` sea considerada una colección de objetos, **no extiende** de la interfaz `COLLECTION`, pero se suele añadir a este repertorio por su funcionalidad.

Todas las clases que ves en el diagrama cumplen el mismo propósito, pero cada una varía y tiene sus características. Vamos a ir viendo poco a poco muchas de ellas.

5.2 - ARRAYLIST

Los `arraylist` son **colecciones dinámicas** que implementan **listas de tamaño variable**, cuyo tamaño por defecto es `0`. Es similar a un array con las ventajas de que **su tamaño crece dinámicamente**

conforme se añaden elementos (no es necesario fijar su tamaño al crearlo) y permite almacenar datos y objetos de cualquier tipo.

Aunque los `arraylist` permiten combinar tipos de datos diferentes, **no es recomendable** para una buena organización y manipulación, ya que esta colección trata a sus elementos como **objetos**, ignorando si son tipos primitivos o no. Para ello podemos especificar el tipo de objetos del `arraylist` con el modificador `<tipo>` que puedes ver en el siguiente apartado: **5.2.1 - *Declarar un Arraylist***.

>> 5.2.1- DECLARAR UN ARRAYLIST

Antes de poder declarar cualquier `arraylist`, debemos **importar la clase**. Después podremos declararla como hacemos con el resto:

```
import java.util.Arraylist;

ArrayList<tipo> nombreAL = new ArrayList<tipo>();
```

`ArrayList`	Nuestro objeto será de tipo `ArrayList`.
`<tipo>`	Aunque no es obligatorio, es recomendable **especificar el tipo de objetos** que contendrá el `arraylist`. Podemos **no ponerlo**, aceptar **cualquier tipo con `<E>`** (**tipo genérico**) o **especificar directamente el tipo**.
`nombreAL`	Asignamos un **nombre** al mismo.
`new`	El operador `new` asigna durante la ejecución del programa un **espacio de memoria al objeto**, y almacena esta **referencia** en una variable.
`ArrayList<tipo>();`	La clase en la que se va a basar el objeto, que será `ArrayList`. Si especificamos el tipo de objetos en la declaración, es recomendable colocarlo aquí también.

>> 5.2.2- AÑADIR DATOS A UN ARRAYLIST

`nombreAL.add(dato);`	Simplemente el método `.add` y lo que queramos añadir entre los paréntesis. Acepta cualquier tipo de primitivas e incluso podemos declarar objetos directamente.

EJEMPLO

.java

```
1  listado.add(-25);
2  listado.add(3.4);
3  listado.add('a');
4  listado.add("hola");
5  listado.add(new Persona("28751533Q", "María", "González Collado", 37));
6
7  //E incluso declarar objetos y añadirlos con su variable:
8  Persona p = new Persona("16834954R", "Raquel", "Dobón Pérez", 62));
9  listado.add(p);
```

>> 5.2.3- MÉTODOS DE ACCESO Y MANIPULACIÓN DE UN ARRAYLIST

La clase **`ArrayList`** ofrece muchos métodos diferentes para poder hacer operaciones con sus elementos. Vamos a ver los más utilizados:

`int .size();`	Devuelve el número total de elementos del **`arraylist`**.
`E .get(index);`	Devuelve la referencia al elemento en la posición **`index`**.
`void .clear();`	Vacía el **`arraylist`** y establece el tamaño a `0`.
`boolean .isEmpty();`	Devuelve true si el **`arraylist`** está vacío.
`boolean .add(dato);`	Como ya hemos visto, inserta un dato en la lista, siempre en la **última posición**.
`void .add(index, dato);`	Inserta el **`dato`** en la posición **`index`** de la lista. D**esplaza una posición** todos los demás elementos de la lista (**no sustituye ni borra otros elementos**).
`void .set(index, dato);`	Sustituye el elemento en la posición **`index`** por el nuevo **`dato`**.
`boolean .contains(dato);`	Busca el **`dato`** en la lista y devuelve **true** si existe. Este método utiliza **`.equals()`** para comparar los objetos.
`int .indexOf(dato);`	Busca el **`dato`** en la lista, empezando por el principio, y devuelve el **índice** dónde se encuentre. Devuelve **`-1`** si **no existe**. Este método utiliza **`.equals()`** para comparar los objetos.
`int .lastIndexOf(dato);`	Igual que el anterior pero **devuelve el índice de la última coincidencia**, ignorando las anteriores.
`E .remove(index);`	**Elimina** el elemento en la posición **`index`** y lo devuelve.
`boolean .remove(dato);`	**Elimina la primera ocurrencia** de **`dato`** en la lista. Devuelve **`true`** si lo ha encontrado y eliminado, y si no, **`false`**. Este método utiliza **`.equals()`** para comparar los objetos.
`void .remove(index);`	**Elimina** el objeto de la lista que se encuentra en la posición **`index`**. Es más rápido que el método anterior ya que no necesita recorrer toda la lista.

Puedes encontrar muchas más información consultando el **API** de Java de la clase **`ArrayList`**.

>> 5.2.4- RECORRER UN ARRAYLIST

Hay varias maneras diferentes en las que se puede **iterar** (recorrer) una colección del tipo ArrayList. La más fácil es como recorrer un array normal, con un bucle for, pero hay otras dos maneras útiles: a través de un bucle **`for-each`** o un **`iterador`**.

· 5.2.4.1- BUCLE FOR-EACH

Este tipo de bucle es una manera resumida de **`for`** que no usa contadores ni límites, simplemente se limita a acceder a cada uno de los elementos existentes en una colección.

```
for(tipo v : colección){
   instrucciones...
}
```

`tipo`	Debemos especificar de qué **tipo** son **los elementos** de la colección que vamos a recorrer.
`v`	Damos un **nombre** para poder **acceder a las posiciones** (se suele usar `v`). Esto actúa de medio para **devolver el valor** de cada elemento.
`colección`	Establecemos la **colección** que queremos recorrer.

EJEMPLO

.java

```java
ArrayList<String> miLista = new ArrayList<String>();

miLista.add("hola");
miLista.add("casa");
miLista.add("árbol");

for(String v : miLista){
    System.out.println(v+", ");
}
```

Console

```
hola, casa, árbol,
```

· 5.2.4.1- ITERADOR

Usando un objeto `Iterator` que permite **recorrer listas como si fuese un índice**. **Se necesita importar la clase antes**.

```java
import java.util.Iterator;

Iterator nombreIterator = colección.iterator();
```

`Iterator nombreIterator`	Creamos el objeto de tipo `Iterator` y le damos un nombre.
`colección.iterator();`	Accedemos a la colección que queremos iterar y le añadimos el método `.iterator()`.

Tiene dos métodos principales:

`.hasNext()`	**Verifica** si hay más elementos después del actual.
`.next()`	**Devuelve** el objeto actual y **avanza al siguiente**.

EJEMPLO

.java

```java
import java.util.Iterator;

ArrayList<String> miLista = new ArrayList<String>();
Iterator iterador = miLista.iterator();

miLista.add("hola");
miLista.add("casa");
miLista.add("árbol");

while(iterador.hasNext()){
    System.out.println(iterador.next()+", ");
}
```

Console

```
hola, casa, árbol,
```

Para recorrer el `arraylist` con un `iterator`, comprobamos si hay más elementos después del actual con `.hasNext()`, y si es así, imprime el actual y mueve el iterador al siguiente de la lista con `.next()`.

5.3 - HASHMAP

Un **`HashMap`** es una **colección dinámica** que no tiene posiciones, sino que guarda los elementos en función de una **clave** y un **valor** que le corresponde. El acceso a estos es a través de la clave, que **no pueden repetirse ni ser nulas**, al contrario que los valores.

> Los **`HashMap`** no guardan el orden ni garantizan el orden de inserción cuando los recorremos.

>> 5.3.1- DECLARAR UN HASHMAP

Antes de poder declarar cualquier **`HashMap`**, debemos **importar la clase**. Después podremos declararla como hacemos con el resto:

```
import java.util.HashMap;

HashMap<tipoClave, tipoValor> nombreHM = new HashMap<tipoClave, tipoValor>();
```

`HashMap`	Nuestro objeto será de tipo **`HashMap`**.
`<tipoClave, tipoValor>`	Debemos **especificar el tipo de las claves y de los valores** que contendrá el **`HashMap`**. para que el programa sepa cómo trabajar con ellos. > Los **`HashMap`** **no aceptan tipos primitivos**, por lo que debemos usar **wrappers** (clases que representan primitivos, como **`Integer`**).
`nombreHM`	Asignamos un **nombre** al mismo.
`new`	El operador **`new`** asigna durante la ejecución del programa un **espacio de memoria al objeto**, y almacena esta **referencia** en una variable.
`HashMap<tipoClave, tipoValor>();`	La clase en la que se va a basar el objeto, que será **`HashMap`**. Se añaden también los tipos de las claves y los valores.

>> 5.3.2- AÑADIR DATOS A UN HASHMAP

`nombreHM.put(clave, valor);`	Simplemente con el método **`.put`** y añadimos **la clave y su valor** entre los paréntesis.

EJEMPLO

`.java`

```
1  import java.util.HashMap;
2
3  HashMap<Integer, String> miHashMap = new HashMap<Integer, String>();
```

```
4
5  miHashMap.put(1, "Primer valor");
```

>> 5.3.3- MÉTODOS DE ACCESO Y MANIPULACIÓN DE UN HASHMAP

La clase **`HashMap`** ofrece muchos métodos diferentes para poder hacer operaciones con sus elementos. Vamos a ver los más utilizados:

`.get(clave)`	Este método nos devuelve el **valor de la clave** que le pasemos. Si no existe la clave, devuelve **`null`**.
`.put(clave, valor)`	Como ya hemos visto, con esto **añadimos una clave y su valor** al **`HashMap`**. Si ya existe la clave, **sustituye el valor**.
`.remove(clave)`	**Elimina la clave y su valor** del **`HashMap`**. Si no existe la clave, devuelve **`null`**.
`.keySet()`	Devuelve un conjunto **`set`** con **todas las claves**.
`.values()`	Devuelve una **colección** con **todos los valores** (los valores pueden estar duplicados, a diferencia de las claves).
`.entrySet()`	Devuelve una **colección** con todos los pares de **clave-valor**.
`.containsKey(clave)`	Devuelve **true** si el **`HashMap`** **contiene la clave indicada** y **false** en caso contrario.

`.getKey()`	Devuelve la **clave** de un **`entrySet`** que contenga **una pareja clave-valor**. Sólo se aplica a esta, no a todo el **`HashMap`**.
	EJEMPLO
	.java
	`for(Map.Entry pareja : nombreHM.entrySet()){`
	`  System.out.println(pareja.getKey());`
	`}`

`.getValue()`	Exactamente igual que **`.getKey()`**, pero devuelve el **valor** del **`entrySet`** .

Veamos con un ejemplo el uso de varios de estos métodos:

EJEMPLO

Ejemplo.java

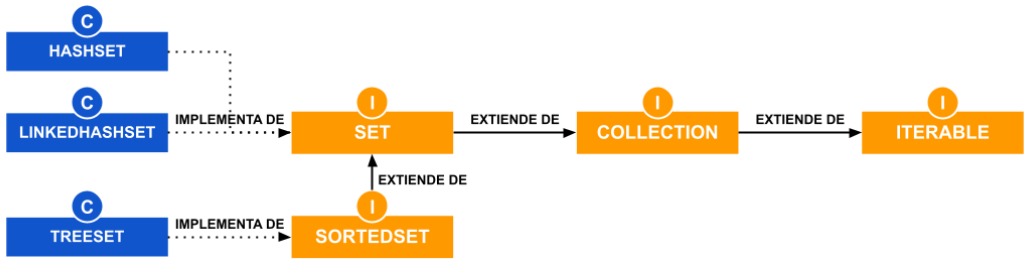
```
1  import java.util.HashMap;
2  import java.util.Map.Entry;
3
4  public class Ejemplo{
5     public static void main(String[] args){
6
7        Hashmap<Integer, String> miHashMap = new ArrayList<Integer, String>();
8
9        miHashMap.put(921, "Juan Alcántara");
10       miHashMap.put(555, "Ana Díaz");
11       miHashMap.put(212, "José Moreno");
12
13       System.out.println("HashMap accediendo por su clave:");
14       for(Integer key : miHashMap.keySet()){
15          System.out.println("Clave: "+key+", Valor: "+miHashMap.get(key));
16       }
```

```java

        System.out.println("\nHashMap accediendo por pares (clave-valor):");
        for(Entry<Integer, String> par : miHashMap.entrySet()){
            System.out.println("Clave: "+par.getKey()+", Valor: "+par.getValue();
        }

    }
}
```

Console

```
HashMap accediendo por su clave:
Clave: 921, Valor: Juan Alcántara
Clave: 555, Valor: Ana Díaz
Clave: 212, Valor: José Moreno

HashMap accediendo por pares (clave-valor):
Clave: 921, Valor: Juan Alcántara
Clave: 555, Valor: Ana Díaz
Clave: 212, Valor: José Moreno
```

5.4 - CLASES SET



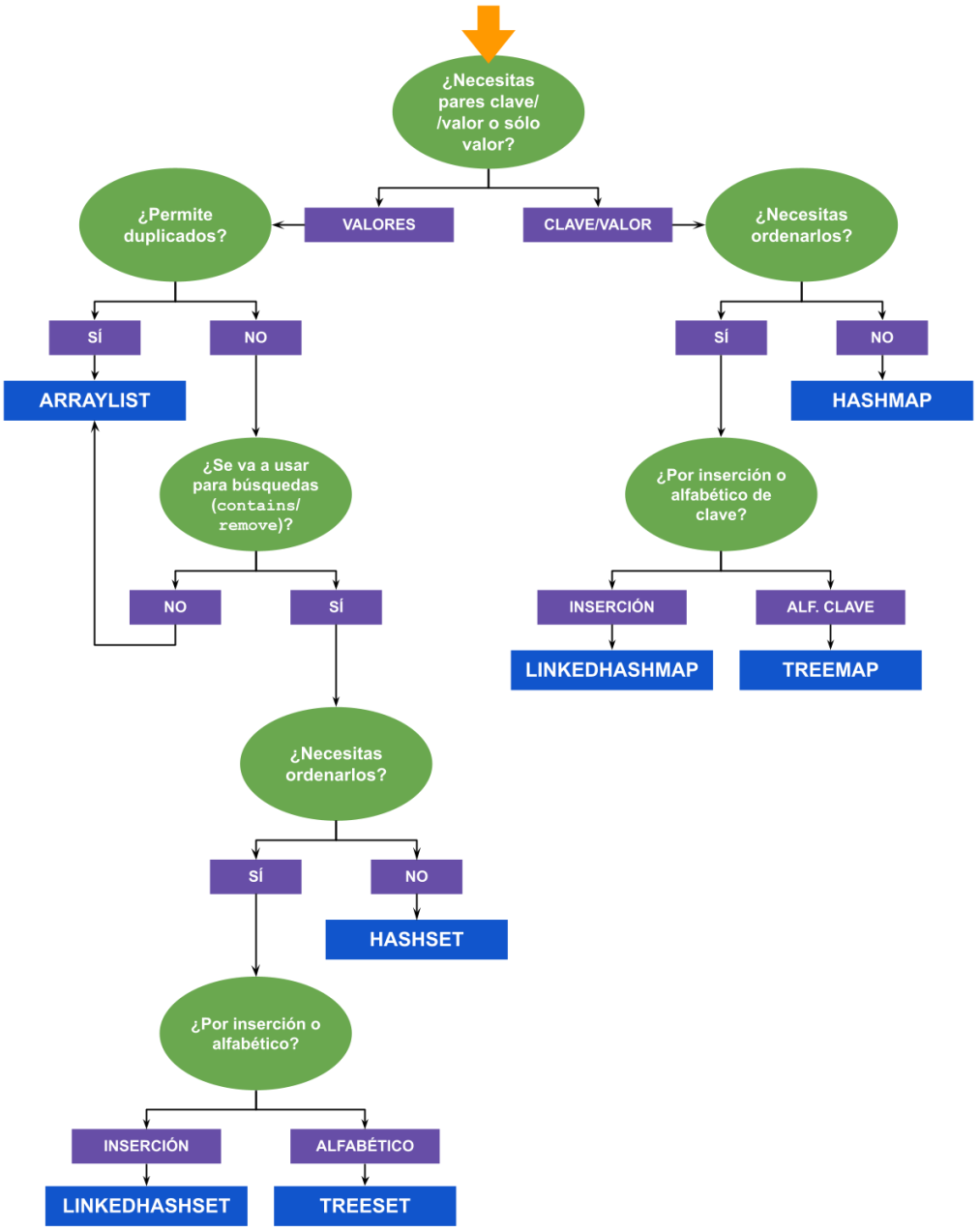
`Set` es un tipo de interfaz usado en colecciones que **no permite duplicados (a diferencia de `List`)**. Las **tres clases** que implementan de esta y sus diferencias son:

HashSet	LinkedHashSet	TreeSet (a través de SortedSet)
No mantiene el orden de inserción.	**Ordena los datos** en **orden alfabético ASCII** automáticamente.	**Mantiene** el orden de inserción.

Aunque en una clase de la interfaz `Set` no se admiten duplicados, esto nos lo podemos saltar si damos **dos objetos que aunque cambien de nombre tengan el mismo contenido**. Para evitar esto usamos la interfaz `Comparator<T>` y su método `compare()`, el cual podemos **sobreescribir** y establecer las reglas de comparación. Puedes aprender más sobre este en el **Anexo:** ***Implementación de Métodos Recomendados***.

5.5 - IMPLEMENTACIÓN DE COLECCIONES

A veces es complicado saber **cuál es la colección más óptima** para nuestro proyecto, por eso puedes usar este diagrama para orientarte y elegir la más adecuada:



U5 - **BATERÍA DE EJERCICIOS**

1	Crea una biblioteca de funciones (librería) para arrayList de números enteros que contenga las siguientes funciones: a) generaArrayListInt: Genera un arrayList de tamaño n con números aleatorios cuyo intervalo (mínimo y máximo) se indica como parámetro. b) minimoArrayListInt: Devuelve el mínimo del arrayList que se pasa como parámetro. c) maximoArrayListInt: Devuelve el máximo del arrayList que se pasa como parámetro. d) mediaArrayListInt: Devuelve la media del arrayList que se pasa como parámetro. e) estaEnArrayListInt: Dice si un número está o no dentro de un arrayList. f) posicionEnArrayList: Busca un número en un arrayList y devuelve la posición (el índice) en la que se encuentra. g) volteaArrayListInt: Le da la vuelta a un arrayList. Crea una clase para probar todas las funciones de la librería, además está podrá ser utilizada en cualquier otro ejercicio.
2	Crea un programa para realizar cálculos relacionados con la altura (en metros) de personas. Se pedirá la altura de personas hasta que se introduzca un 0, y se almacenarán en un arrayList las alturas introducidas por teclado. Luego mostrará la altura media, máxima y mínima así como cuántas personas miden por encima y por debajo de la media.
3	Crea un programa que cree un arrayList de enteros y luego muestre el siguiente menú con distintas opciones: a) Mostrar valores. (inicialmente no tendrá ningún valor) b) Introducir valor al final. c) Introducir valor en una posición. d) Salir. La opción ‘a’ mostrará todos los valores por pantalla. La opción ‘b’ pedirá un valor V, y luego escribirá V en el arrayList La opción ‘c’ pedirá un valor V y una posición P, luego escribirá V en la posición P del arrayList. El menú se repetirá indefinidamente hasta que el usuario elija la opción ‘d’ que terminará el programa.
4	Crea un programa que permita al usuario almacenar una secuencia aritmética en un arrayList y luego mostrarla. Una secuencia aritmética es una serie de números que comienza por un valor inicial V, y continúa con incrementos de I. Por ejemplo, con V=1 e I=2, la secuencia sería 1, 3, 5, 7, 9... Con V=7 e I=10, la secuencia sería 7, 17, 27, 37... El programa solicitará al usuario V, I además de N (número de valores a crear)
5	Crea un programa que pida al usuario valores hasta que se introduzca -1 y los guarde en un arrayList. Luego muestra el arrayList por pantalla.
6	Crea un programa que cree un arrayList de enteros e introduzca la siguiente secuencia de valores: 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, etc. hasta introducir 10 diez veces, y luego la muestre por pantalla.

7	Crea un programa que pida al usuario 10 cadenas de texto e introduzca las 5 primeros en un arrayList y los 5 últimos en otro arrayList. Por último, comparará ambos arrayList y le dirá al usuario si son iguales o no.
8	Supongamos una clase Producto con dos atributos: • String nombre • int cantidad Implementa esta clase con un constructor (con parámetros) además de los getters y setters de sus dos atributos, el método toString y el método equals. No es necesario comprobar los datos introducidos. A continuación, en el programa principal haz lo siguiente: 1. Crea 5 instancias de la Clase Producto. 2. Crea un ArrayList. 3. Añade las 5 instancias de Producto al ArrayList. 4. Visualiza el contenido de ArrayList utilizando Iterator. 5. Elimina dos elementos del ArrayList. 6. Inserta un nuevo objeto producto en la posición 2 de la lista. 7. Visualiza de nuevo el contenido de ArrayList utilizando for-each. 8. Elimina todos los valores del ArrayList y muestra el tamaño.
9	Crea un ArrayList con los nombres de 6 compañeros de clase. A continuación, muestra esos nombres por pantalla. Utiliza para ello un bucle for que recorra todo el ArrayList sin usar ningún índice.
10	Crear un programa que use ArrayList de números reales. El programa debe permitir mostrar un menú donde se pueda agregar un número, buscar un número, modificar un número, eliminar un número e insertar un número en una posición dada.
11	Implementa el control de acceso al área restringida de un programa. Se debe pedir un nombre de usuario y una contraseña. Si el usuario introduce los datos correctamente, el programa dirá “Ha accedido al área restringida”. El usuario tendrá un máximo de 3 oportunidades. Si se agotan las oportunidades el programa dirá “Lo siento, no tiene acceso al área restringida”. Los nombres de usuario con sus correspondientes contraseñas deben estar almacenados en una estructura de la clase HashMap.
12	Crea un mini-diccionario español-inglés que contenga, al menos, 20 palabras (con su correspondiente traducción). Utiliza un objeto de la clase HashMap para almacenar las parejas de palabras. El programa pedirá una palabra en español y dará la correspondiente traducción en inglés.
13	Realiza un programa que escoja al azar 5 palabras en español del mini- diccionario del ejercicio anterior. El programa irá pidiendo que el usuario teclee la traducción al inglés de cada una de las palabras y comprobará si son correctas. Al final, el programa deberá mostrar cuántas respuestas son válidas y cuántas erróneas.
14	Crear un programa para gestionar la plantilla de un equipo de futbol. El programa debe tener un HashMap con los jugadores actuales de la plantilla, de forma que la clave sea su número (que es único) y el valor su nombre. Será posible añadir nuevos jugadores (comprobar que no exista jugador con ese número para no machacarlo), eliminar jugadores, mostrar todos los jugadores y buscar si hay actualmente un jugador con el número indicado.
15	Crear un programa para gestionar el stock de productos en un almacén. Será necesario mantener información del nombre del producto, y las unidades del mismo. Se creará un menú que permita añadir productos, si el producto existe aumenta el stock, si no existe lo crea, eliminar productos, reducir el stock de un producto (teniendo en cuenta que no puede quedar stock negativo), buscar las unidades que hay de un determinado producto, listar todos los productos, y listar todos los productos junto con su stock.
16	Modificar el ejercicio 14, para que en vez de guardar de un jugador sólo el nombre, se almacene nombre y posición, que podrá ser (Delantero, Portero, Defensa o Medio_campo). Además, añadir funcionalidades para modificar la posición de un jugador, y modificar el nombre.
17	Crear un programa que inicialmente contenga una estructura de tipo HashMap, de forma que la clave sea un entero, y el valor otro entero. Se deberá mostrar el contenido completo del hashMap, posteriormente se modificarán los valores cuya clave sea par, multiplicando por dos el valor, se

	volverá a mostrar el contenido completo, y se modificarán los valores cuya clave sea impar, para sumar 10 al valor. Volviendo a mostrar el contenido completo.
18	Utilizando las clases creadas en el ejercicio 5 de la unidad 8 (Electrodomésticos) se deberá crear un hashMap para gestionar el almacén, teniendo en cuenta las siguientes características: • Todos los electrodomésticos serán almacenados en un único HashMap • La clave se asignará de la siguiente forma dependiendo del tipo de electrodoméstico: ◦ Lavadora: Lvx, siendo x un número consecutivo. ◦ Televisión: TVx ◦ Cualquier otro: Elx • Se tendrá un stock inicial de electrodomésticos • Se podrá añadir cualquier electrodoméstico, la clave deberá generarse automáticamente dependiendo del tipo. El resto de datos se pedirán al usuario. • Se podrán listar todos los electrodomésticos • Se podrán listar todas las lavadoras • Se podrán listar todos los televisores • Se podrán listar todos los electrodomésticos que no sean televisor ni lavadora. • Se podrán eliminar electrodomésticos, pero esa clave no será reutilizada en el futuro, simplemente se ignora. • Obtener el precio final de un electrodoméstico a partir de su clave. Todas las funcionalidades anteriores se realizarán mediante un menú que llamará a métodos siempre que sea posible. La clase Main debe quedar lo más clara posible.
19	Crea una clase Empleado con los siguientes atributos y constructor: private String nombre; private double sueldo; public Empleado(String nombre, double sueldo) Además tendrá los siguientes métodos: public String getNombre() public double getSueldo() Escribe un programa que utilizando una colección HashSet permita, a través de un menú, gestionar una lista de empleados de número ilimitado donde no se permitirán nombres de empleados repetidos y siendo el orden de los mismos totalmente irrelevante: 1.-Introducir empleado 2.-Listar empleados 3.-Eliminar empleado 4.-Borrar todos 5.-Mostrar número de empleados 6.-Buscar empleado 7.-Salir • Para introducir un empleado hará falta también introducir su sueldo. • Al listar los empleados se mostrará su nombre y sueldo. • Para eliminar un empleado será necesario introducir su nombre. • El programa no será sensible a mayúsculas