Programación

UD 6: Estructuras de datos dinámicas

Ciclo Formativo de Grado Superior
Desarrollo de Aplicaciones Web

Francisco Pérez

Fondo Social Europeo

Estructuras de datos dinámicas

1.- Estructuras de datos dinámicas

1.- Estructuras de datos dinámicas

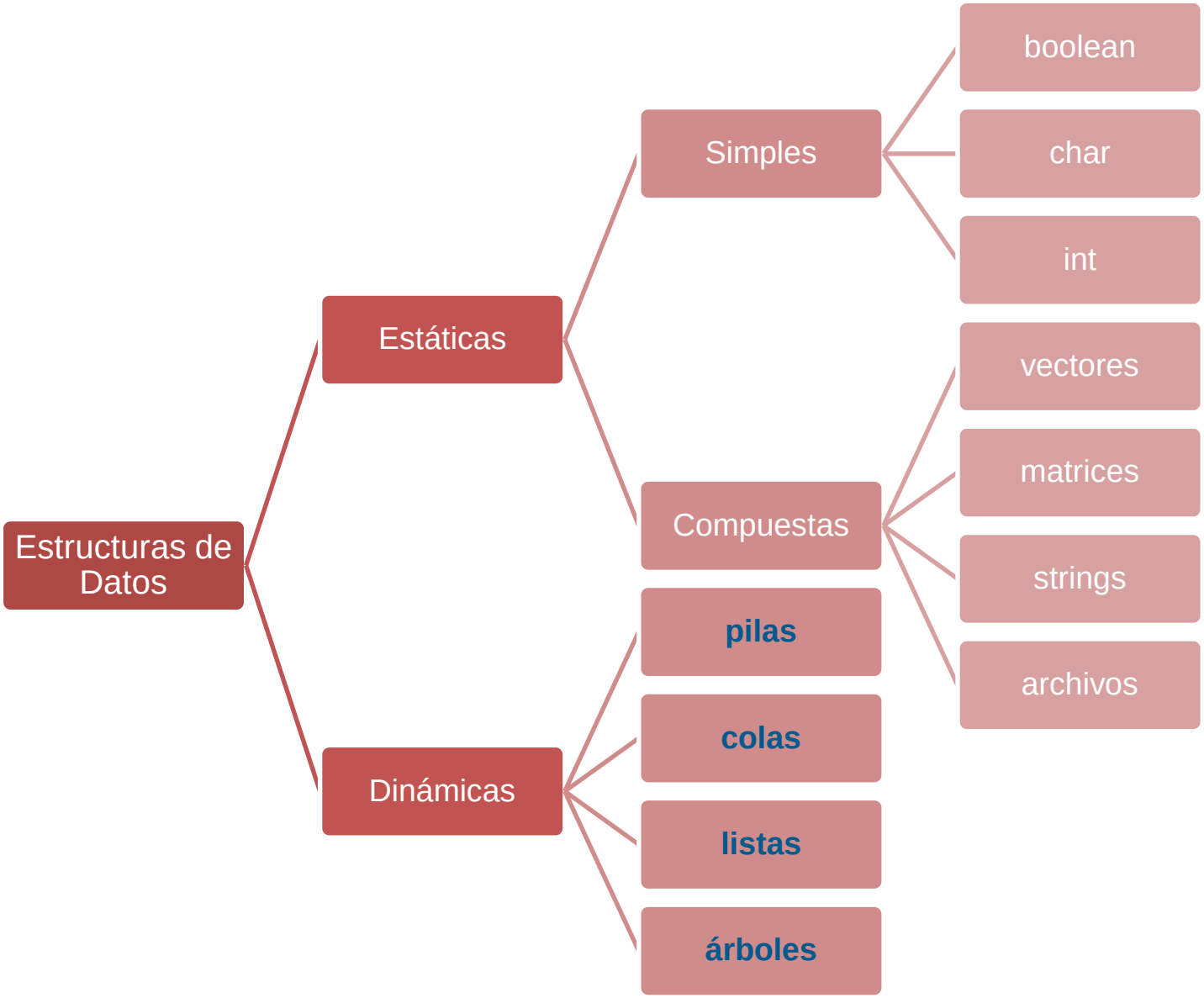
Estructuras de datos estáticas

Las estructuras estáticas son aquellas en las que el tamaño ocupado en memoria se define con anterioridad a la ejecución del programa que los usa, de forma que su dimensión no puede modificarse durante la misma (p.e., un vector o una matriz) aunque no necesariamente se tenga que utilizar toda la memoria reservada al inicio (en todos los lenguajes de programación las estructuras estáticas se representan en memoria de forma contigua).

Estructuras de datos dinámicas

Por el contrario, ciertas estructuras de datos pueden crecer o decrecer en tamaño, durante la ejecución, dependiendo de las necesidades de la aplicación, sin que el programador pueda o deba determinarlo previamente: son las llamadas estructuras dinámicas. Las estructuras dinámicas no tienen teóricamente limitaciones en su tamaño, salvo la única restricción de la memoria disponible en el computador.

1.- Estructuras de datos dinámicas



2.- ¿Qué son las colecciones?

2.- ¿Qué son las colecciones?

Una colección representa un grupo de objetos.

Estos objetos son conocidos como elementos.

Cuando queremos trabajar con un conjunto de elementos, necesitamos un almacén donde poder guardarlos.

En Java, se emplea la interfaz genérica Collection para este propósito.

```
import java.util.*
```

2.- ¿Qué son las colecciones?

Gracias a la interfaz Collection, podemos almacenar cualquier tipo de objeto y podemos usar una serie de métodos comunes, como pueden ser:

- ✓ Añadir
- ✓ Eliminar
- ✓ Obtener el tamaño de la colección
- ✓ etc.

Partiendo de la interfaz genérica Collection extienden otra serie de interfaces genéricas.

Estas subinterfaces aportan distintas funcionalidades sobre la interfaz anterior.

3.- Tipos de colecciones

3.- Tipos de colecciones

Tipos de colecciones en Java:

- Set
  - HashSet
  - TreeSet
  - LinkedHashSet
- List
  - **ArrayList**
  - Vector
  - LinkedList
- Map
  - **HashMap**
  - TreeMap
  - LinkedHashMap
- Queue
  - **PriorityQueue**
- Stack
  - **Stack**

3.- Tipos de colecciones

3.1.- List:

La interfaz List define una sucesión de elementos. A diferencia de la interfaz Set, la interfaz List sí admite elementos duplicados. A parte de los métodos heredados de Collection, añade métodos que permiten mejorar los siguientes puntos:

- Acceso posicional a elementos: manipula elementos en función de su posición en la lista.
- Búsqueda de elementos: busca un elemento concreto de la lista y devuelve su posición.
- Iteración sobre elementos: mejora el *Iterator* por defecto.
- Rango de operación: permite realizar ciertas operaciones sobre rangos de elementos dentro de la propia lista.

Dentro de la interfaz List existen los siguientes tipos de implementaciones realizadas dentro de la plataforma Java:

ArrayList
Vector
LinkedList

3.- Tipos de colecciones

ArrayList:

La clase ArrayList permite el almacenamiento de datos en memoria de forma similar a los arrays convencionales pero con la gran ventaja de que el número de elementos que puede almacenar es dinámico.

Para declarar un Arraylist:

```
ArrayList<nombreClase> nombreLista;
```

En caso de almacenar datos de un tipo básico de Java (char, int, double, etc**…) se debe especificar el nombre de la clase asociada**: Character, Integer, Double, etc.. Por ejemplo:

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

3.- Tipos de colecciones

ArrayList (Métodos)

Boolean add(Object elementoaInsertar)

-Los elementos que se van añadiendo se colocan después del último elemento.
-El primer elemento que se añada se colocará en la posición 0

```
ArrayList<String> listaPaises = new ArrayList();
listaPaises.add("España");   //Ocupa la posición 0
listaPaises.add("Francia");  //Ocupa la posición 1
listaPaises.add("Portugal"); //Ocupa la posición 2

//Se pueden crear ArrayList para guardar datos numéricos
ArrayList<Integer> edades = new ArrayList();
edades.add(22);
edades.add(31);
edades.add(18);
```

3.- Tipos de colecciones

ArrayList (Métodos)

void add(int posicion, Object elementoaInsertar)

-Inserta un elemento en una determinada posición desplazando el elemento que se encontraba en esa posición y todos los siguientes, una posición más.

-Si se intenta insertar en una posición que no existe se produce una excepción del tipo *IndexOutOfBoundsException*

```
ArrayList<String> listaPaises = new ArrayList();
listaPaises.add("España");
listaPaises.add("Francia");
listaPaises.add("Portugal");
//El orden hasta ahora es: España, Francia, Portugal
listaPaises.add(1, "Italia");
//El orden ahora es: España, Italia, Francia, Portugal
```

3.- Tipos de colecciones

ArrayList (Métodos)

Object get(int posicion)

-Permite obtener el elemento almacenado en una determinada posición

```
System.out.println(listaPaises.get(3));
//Siguiendo el ejemplo anterior, mostraría: Portugal
```

Object set(int posicion, Object nuevoElemento)

-Permite modificar un elemento que previamente ha sido almacenado en la lista con «add».

- El primer parámetro indica la posición que ocupa el elemento a modificar.
- El segundo parámetro especifica el nuevo elemento que ocupará dicha posición sustituyendo al elemento anterior.

```
listaPaises.set(1, "Alemania");
```

3.- Tipos de colecciones

ArrayList (Métodos)

int indexOf(Object elementoBuscado)

-Retorna la posición que ocupa el elemento que se indique por parámetro
-Si el elemento se encuentra en más de una posición, retorna la posición del primero.
-El método «lastIndexOf» obtiene la posición del último encontrado.

```
String paisBuscado = "Francia";
int pos = listaPaises.indexOf(paisBuscado);
if(pos!=-1)
   System.out.println(paisBuscado + “  en la posición: "+pos);
else
   System.out.println(paisBuscado + " no se ha encontrado");
```

3.- Tipos de colecciones

ArrayList (Métodos)

Tenemos el método size() que nos devuelve el número de elementos que contiene la lista, lo cual es ideal para poder recorrerla.

```
for(int i=0; i<listaPaises.size(); i++)
    System.out.println(listaPaises.get(i));
```

Los ArrayList también pueden recorrerse utilizando un «Iterator», que es un objeto que nos permite desplazarnos a través de una colección:

```
Iterator iter = listaPaises.iterator( );
while ( iter.hasNext( ) ) { //True si existen más elementos
        System.out.println(iter.next( ));  //devuelve elemento
}                                          //y apunta al siguiente
```

3.- Tipos de colecciones

ArrayList (Métodos)

MÉTODO	DESCRIPCIÓN
size()	Devuelve el número de elementos (int)
add(X)	Añade el objeto X al final. Devuelve true.
add(posición, X)	Inserta el objeto X en la posición indicada.
get(posicion)	Devuelve el elemento que está en la posición indicada.
remove(posicion)	Elimina el elemento que se encuentra en la posición indicada. Devuelve el elemento eliminado.
remove(X)	Elimina la primera ocurrencia del objeto X. Devuelve true si el elemento está en la lista.
clear()	Elimina todos los elementos.
set(posición, X)	Sustituye el elemento que se encuentra en la posición indicada por el objeto X. Devuelve el elemento sustituido.
contains(X)	Comprueba si la colección contiene al objeto X. Devuelve true o false.
indexOf(X)	Devuelve la posición del objeto X. Si no existe devuelve -1
isEmpty(X)	Retorna true si la lista está vacía

Los puedes consultar todos en:
https://docs.oracle.com/javase/9/docs/api/java/util/ArrayList.html

3.- Tipos de colecciones

3.2.- Map:

La interfaz Map asocia pares de claves y valores.

Esta interfaz no puede contener claves duplicadas y; cada una de dichas claves, sólo puede tener asociado un valor como máximo.

Dentro de la interfaz Map existen los siguientes tipos de implementaciones realizadas dentro de la plataforma Java:

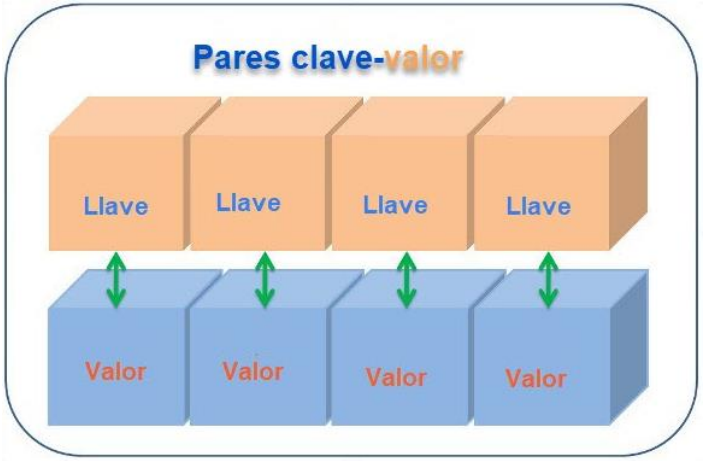
HashMap
TreeMap
LinkedHashMap

3.- Tipos de colecciones

HashMap:

Esta implementación almacena las claves en una tabla hash. En esta tabla, los valores se almacenan en un mapa formando un par clave-valor. El valor se puede recuperar usando la clave pasándola al método correcto.

De la misma forma que en el caso anterior del ArrayList, es imposible utilizar tipos de datos primitivos como double o int, por lo que en su lugar utilizaremos sus clases contenedoras (Integer o Double).



3.- Tipos de colecciones

HashMap:

Se crea con: HashMap<Clave,Valor> Nombre = new HashMap<Clave,Valor>();

```
public class Principal {

    public static void main(String[] args) {

        //Definimos un mapa cuya clave es un entero y cuyo dato es un String.
        HashMap <Integer, String> map = new HashMap <Integer, String> ();
        map.put(46, "España");
        map.put(22, "Italia");
        map.put(13, "Francia");

        //Imprime {22=Italia, 13=Francia, 46=España}
        System.out.println(map);

        //Recupera "Italia"
        System.out.println(map.get(22));

        //Elimina a Francia, porque nos tiran los camiones de fruta
        map.remove(13);

        //Como ya no existe, nos devolvería null
        System.out.println(map.get(13));

        //Imprime {22=Italia, 46=España}
        System.out.println(map);
    }

}
```

3.- Tipos de colecciones

HashMap (Métodos)

MÉTODO	DESCRIPCIÓN
clear()	Borra todo el mapa
containsKey(K)	Devuelve true si el mapa contiene la clave K
containsValue(V)	Devuelve true si el mapa contiene el valor V
get(K)	Devuelve el valor asociado a la clave K
put(K, V)	Inserta en el mapa el valor V que será accesible mediante la clave K
isEmpty()	Devuelve true si el mapa está vacío
remove(K)	Borra la clave K del mapa
size()	Devuelve el tamaño del mapa

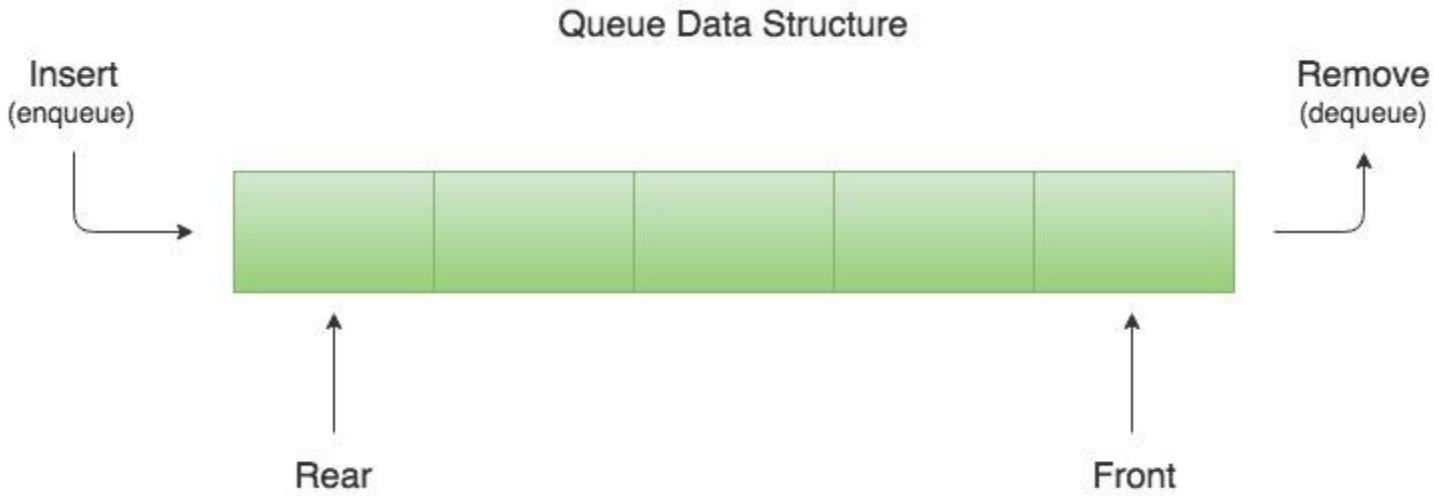
Los puedes consultar todos en:
https://docs.oracle.com/javase/9/docs/api/java/util/HashMap.html

3.- Tipos de colecciones

3.3.- Queue (Cola):

Una cola es una estructura de datos First In First Out (FIFO). Simula una cola en la vida real. Sí, la que podrías haber visto frente a un cine, un centro comercial, un metro o un autobús.

Al igual que las colas en la vida real, los elementos nuevos en una estructura de datos de cola se agregan en la parte posterior y se eliminan de la parte frontal. Una cola se puede visualizar como se muestra en la figura a continuación.



3.- Tipos de colecciones

Queue:

```
public static void main(String[] args) {

    //Definimos una cola que se crea como un objeto LinkedList
    Queue <String> colacine = new LinkedList<String>();
    colacine.add("Juan");
    colacine.add("Perico");
    colacine.add("Andrés");

    //Devuelve Juan
    System.out.println("El primero de la cola es: " + colacine.element());

    //Vamos a matar a Juan, que ya estaba muy mayor
    colacine.remove();

    //Tras el entierro de Juan, ahora el primero de la cola es Perico
    System.out.println("El primero de la cola es: " + colacine.element());

    //Imprime la cola resultante , que es [Perico, Andrés]
    System.out.println(colacine);

    //Ahora llega Tardon, que se sitúa al final de la cola
    colacine.add("Tardon");

    //Imprime la nueva cola, que es [Perico, Andrés, Tardon]
    System.out.println(colacine);

}
```

3.- Tipos de colecciones

Queue (métodos):

MÉTODO	DESCRIPCIÓN
add()	Inserta un elemento al final de la cola. Devuelve true.
element()	Devuelve, pero no elimina, la cabeza de la cola.
remove()	Elimina, y devuelve, la cabeza de la cola.

Además de todos los métodos heredados de la clase **Collection**: clear, contains, isEmpty, size, … y también se puede recorrer con un «Iterator» como el ArrayList:

```
Iterator iter = cola.iterator();
while (iter.hasNext()) {
    System.out.println(iter.next());
}
```

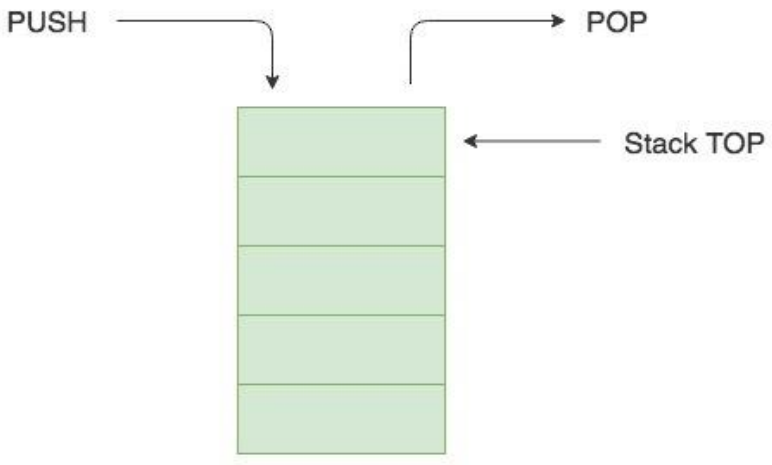
Los puedes consultar todos en:
https://docs.oracle.com/javase/9/docs/api/java/util/Queue.html

3.- Tipos de colecciones

3.4.- Stack (Pila):

Una pila es una estructura de datos Last In First Out (LIFO). Soporta dos operaciones básicas llamadas push y pop.

La operación de inserción agrega un elemento en la parte superior de la pila, y la operación de apertura elimina un elemento de la parte superior de la pila.



Stack Data Structure

(Elements are added and removed from the top)

3.- Tipos de colecciones

Stack:

```
public static void main(String[] args) {

    //Definimos una pila
    Stack<String> pilalibros = new Stack<String>();

    //Apilamos 3 lecturas indispensables
    pilalibros.push("El Quijote");
    pilalibros.push("La Guía telefónica");
    pilalibros.push("Biografía de Ana Obregón");

    //Leemos el ultimo libro de la pila mediante "peek"
    String ultimo = pilalibros.peek();

    //Devuelve "Biografía de Ana Obregón"
    System.out.println("El ultimo libro apilado es: " + ultimo);

    //Una mesa nos cojea por lo que desapilamos la biografía de Ana Obregón
    pilalibros.pop();

    //Ahora el ultimo libro apilado es "La Guía telefónica"
    ultimo = pilalibros.peek();
    System.out.println("Ahora el ultimo libro apilado es: " + ultimo);

}
```

3.- Tipos de colecciones

Stack (métodos):

MÉTODO	DESCRIPCIÓN
empty()	Devuelve True si la pila está vacía. False, en caso contrario.
peek()	Devuelve el elemento de la cima de la pila sin eliminarlo.
pop()	Elimina el elemento de la cima de la pila y lo devuelve.
push()	Inserta un elemento en la cima de la pila.
search()	Devuelve la posición del elemento en la pila.

Además de todos los métodos heredados de la clase **Collection**: clear, contains, isEmpty, size, … y también se puede recorrer con un «Iterator» como el ArrayList

```
Iterator iter = pilalibros.iterator();
while (iter.hasNext()) {
    System.out.println(iter.next());
}
```

Los puedes consultar todos en:
https://docs.oracle.com/javase/9/docs/api/java/util/Stack.html

ANEXO: Un nuevo bucle

Bucle ForEach

Además de todas las estructuras cíclicas que ya conoces (for, while, do-while), vamos a dar la bienvenida a una nueva que no habíamos visto hasta ahora, la estructura ForEach, que se suele utilizar también para recorrer estructuras de datos dinámicas.

Este bucle, que se caracteriza por el uso de los 2 puntos (:) se utiliza así, ahorrándonos la llamada al método "get" por cada uno de los nodos que recorremos:

```
public static void main(String[] args) {
    ArrayList<String> simpsons = new ArrayList<String>();
    simpsons.add("Homer");
    simpsons.add("Marge");
    simpsons.add("Bart");
    simpsons.add("Lisa");
    simpsons.add("Maggie");

    for (String nombre : simpsons) {
        System.out.println(nombre);
    }
}
```

```
Problems
<terminated>
Homer
Marge
Bart
Lisa
Maggie
```

ANEXO: Un nuevo bucle

Bucle ForEach

Este bucle es el que se suele utilizar también para recorrer un HashMap. Utilizaremos el método **"keyset"** para iterar sobre el conjunto de claves del HashMap y extraer toda la información clave-valor. Por ejemplo:

```
public static void main(String[] args) {
    Map<Integer, String> tenistas = new HashMap<>();
    tenistas.put(1,"Nadal");
    tenistas.put(2,"Yocovid");
    tenistas.put(3,"Federer");
    for (Integer key: tenistas.keySet()) {
        System.out.println(key + ":" + tenistas.get(key));
    }
}
```

```
Problems
<terminated> B
1:Nadal
2:Yocovid
3:Federer
```