

# UT7: Estructuras de datos dinámicas

---

PROGRAMACIÓN DAM/DAW

# 1. Introducción.

---

Una colección es un objeto que almacena un conjunto de referencias a otros objetos, dicho de otra manera, es un especie de array de objetos.

Sin embargo, a diferencia de los array, las colecciones son dinámicas, en el sentido de que no tienen un tamaño fijo y permiten añadir y eliminar objetos en tiempo de ejecución.

# 1. Introducción.

---

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:

- Añadir objetos a la colección.
- Eliminar objetos de la colección.
- Obtener un objeto de la colección.
- Localizar un objeto en la colección.
- Iterar a través de una colección.

# Introducción.



## Vectores: Definición

Es un array que crece automáticamente cuando se alcanza la dimensión inicial máxima.

Se encuentra en el paquete `java.util.*`

# Constructores

Podemos encontrar 3 constructores

1. `Vector v=new Vector(10, 5);`

Tenemos un Vector con una dimensión de 10 y si se almacenan 11 elementos crece 5. Su dimensión final sería 15.

# Constructores





















2. `Vector v=new Vector(10)`

Si se insertan 11 elementos, la dimensión del vector se duplica.

3. `Vector v=new Vector()`

La dimensión inicial es 10 y si se rebasa será de 11, luego 12, 13, etc.

# Métodos

 <b>add</b> (Object e)	boolean	
 <b>add</b> (int index, Object element)	void	
 <b>addAll</b> (Collection c)	boolean	
 <b>addAll</b> (int index, Collection c)	boolean	
 <b>addElement</b> (Object obj)	void	
 <b>capacity</b> ()	int	
 <b>clear</b> ()	void	
 <b>clone</b> ()	Object	
 <b>contains</b> (Object o)	boolean	
 <b>containsAll</b> (Collection c)	boolean	
 <b>copyInto</b> (Object[] anArray)	void	
 <b>elementAt</b> (int index)	Object	
 <b>elements</b> ()	Enumeration	
 <b>ensureCapacity</b> (int minCapacity)	void	
 <b>equals</b> (Object o)	boolean	
 <b>firstElement</b> ()	Object	
 <b>get</b> (int index)	Object	



# Métodos

Entre los más importantes encontramos:

- `add(object e)`: Añade un objeto al final del vector
- `add(int index, object e)`: Inserta el objeto en la posición indicada por el índice, desplazando los elementos a la derecha.
- `addElement(Object e)`: Añade el objeto al final y se incrementa el tamaño del Vector.
- `capacity()`: Devuelve la capacidad actual del vector

# Métodos

`clear()`: Borra todos los elementos del Vector

`elementAt(int index)`: Devuelve el elemento que se encuentra en la posición del índice

`firstElement()` Devuelve el primer elemento del Vector

`get(int index)`: Devuelve el elemento que se encuentra en la posición del índice

# Métodos

`isEmpty()`: Comprueba si está vacío

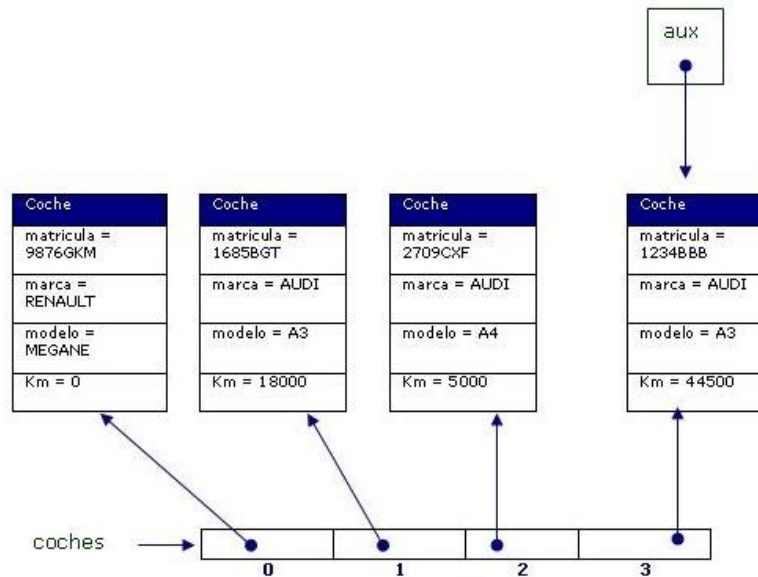
`removeElementAt(int index)`: Borra el elemento que se encuentra en la posición del índice

`toArray()`: Convierte el Vector en un Array

`ToString()`: Devuelve una cadena que representa al vector

# La clase ArrayList.

Representa una colección basada en índices, en la que cada objeto de la misma tiene asociado un número (índice) según la posición que ocupa dentro de la colección, siendo 0 la posición del primer elemento.



## Creación de un ArrayList.

Para crear un objeto ArrayList utilizamos la expresión:

```
ArrayList variable_objeto= new  
    ArrayList();
```

Ejemplo:

```
ArrayList v= new ArrayList();
```

Una vez creado, podemos hacer uso de los métodos de la clase ArrayList para realizar operaciones habituales con una colección.

## Métodos de la clase ArrayList.

**boolean add (Object o):** Añade un nuevo objeto a la colección y lo sitúa al final de la misma, devolviendo el valor true.

**boolean add(int indice, Object o):** Añade el objeto al ArrayList en la posición especificada por índice, desplazando hacia delante el resto de los elementos de la colección.

## Métodos de la clase ArrayList.

**Object get(int indice):** Devuelve el objeto que ocupa la posición indicada. Hay que tener en cuenta que el tipo de devolución es Object, por tanto, para guardar la referencia al objeto devuelto en una variable de su tipo será necesario realizar una conversión explícita.

**Object remove(int indice):** Elimina de la colección el objeto que ocupa la posición indicada, desplazando hacia atrás los elementos de las posiciones siguientes. Devuelve el objeto eliminado.

## Métodos de la clase ArrayList.

**void clear():** Elimina todos los elementos de la colección.

**int indexOf(Object o):** Localiza en el ArrayList el objeto indicado como parámetro, devolviendo su posición. En caso de que el objeto no se encuentre en la colección, la llamada al método devolverá como resultado el valor -1.



## Métodos de la clase ArrayList.

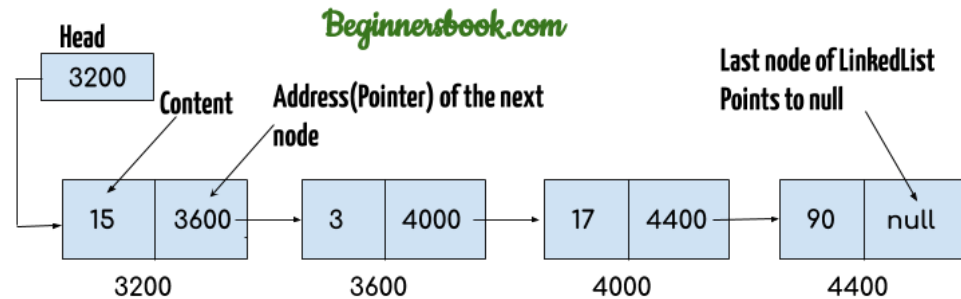
**int size():** Devuelve el número de elementos almacenados en la colección. Utilizando este método conjuntamente con `get()`, se puede recorrer la colección completa.

# La clase LinkedList

Cada elemento de la Lista de Enlaces se llama Nodo.

Cada Nodo de la ListaEnlazada contiene dos elementos:

- 1) Contenido del elemento
- 2) Puntero/Dirección/Referencia al siguiente Nodo de la LinkedList.



# Métodos de la clase LinkedList

void	<b>addFirst(E e)</b> Inserta el elemento especificado al principio de esta lista.
void	<b>addLast(E e)</b> Agrega el elemento especificado al final de esta lista.
void	<b>clear()</b> Elimina todos los elementos de esta lista.

# Métodos de la clase LinkedList

**E** `remove(int index)`

Elimina el elemento en la posición especificada en esta lista.

`boolean` `remove(Object o)`

Elimina la primera aparición del elemento especificado de esta lista, si está presente.

**E** `removeFirst()`

Elimina y devuelve el primer elemento de esta lista.

`boolean` `removeFirstOccurrence(Object o)`

Elimina la primera aparición del elemento especificado en esta lista (al recorrer la lista de principio a fin).

**E** `removeLast()`

Elimina y devuelve el último elemento de esta lista.

## La clase Hashtable.

Esta clase representa un tipo de colección basada en claves, donde los objetos almacenados en la misma no tienen asociado un índice numérico basado en su posición, sino una clave que lo identifica de forma única dentro de la colección. Una clave puede ser cualquier tipo de objeto.

## La clase Hashtable.

La utilización de las colecciones basadas en claves resulta útil en aquellas aplicaciones en las que se requiera realizar búsquedas de objetos a partir de un dato que lo identifica. Por ejemplo, si se va a gestionar una colección de objetos de tipo “Empleado”, puede resultar más práctico, almacenarlos en un hashtable, asociándoles como clave el “dni”, que guardarlos en un ArrayList en el que cada empleado se le asigna un índice según el orden de almacenamiento.

## Creación de un Hashtable.

La creación de un objeto hashtable se realiza utilizando el constructor sin parámetros de la clase:

```
Hashtable  
variable_objeto=new Hashtable();
```

Por ejemplo:

```
Hashtable tb=new  
Hashtable();
```

## Métodos de la clase Hashtable.

**Object put(object key, Object valor):** Añade a la colección el objeto valor, asignándole la clave especificada por key. En caso de que exista esa clave en la colección, el objeto que tenía asignada esa clave se sustituye por el nuevo objeto valor, devolviendo el objeto sustituido.



## Métodos de la clase Hashtable.

**boolean containsKey(Object key):** Indica si la clave especificada existe o no en la colección.

**Object get(Object key):** Devuelve el valor que tiene asociado la clave que se indica en el parámetro. En caso de que no exista ningún objeto con esa clave asociada, devolverá *null*.

# Métodos de la clase Hashtable.

---

**Object remove(Object key):** Elimina de la colección el valor cuya clave se especifica en el parámetro. En caso de que no exista ningún objeto con esa clave, no hará nada y devolverá *null*, si existe, eliminará el objeto y devolverá una referencia al mismo.

# Métodos de la clase Hashtable.

---

**int size():** Devuelve el número de objetos almacenados en la colección.

**Enumeration keys():** Devuelve un objeto enumeration que permite iterar sobre el conjunto de claves.

# Iteración de un Hashtable.

---

Al no estar basado en índices, un Hashtable no se puede recorrer utilizando una instrucción *for* con una variable que recorra las posiciones de los objetos.

Esto no significa que no se puede iterar sobre un Hashtable, se puede hacer a través de un objeto enumeration.

Enumeration es un objeto que implementa la interfaz ***java.util.Enumeration***.

# Iteración de un Hashtable.

---

Los métodos proporcionados por la interfaz Enumeration permiten recorrer una colección de objetos asociada y acceder a cada uno de sus elementos. En el caso concreto del método *keys()* de la clase Hashtable, el código Enumeration devuelto nos permite recorrer la colección de claves del Hashtable.

Métodos de la interfaz Enumeration:

# Iteración de un Hashtable.

---

**Object nextElement():** La llamada este método sobre un objeto Enumeration provoca que ésta pase a apuntar al siguiente objeto de la colección, devolviendo el nuevo objeto apuntado.

**boolean hasMoreElements():** Indica si hay más elementos por recorrer en la colección. Cuando el objeto enumeration esté apuntando al último elemento, la llamada a este método devolverá false.

# Colecciones de tipos genéricos.

---

La utilización de colecciones basadas en tipos genéricos proporciona un mecanismo que permite notificar al compilador el tipo de los objetos que va a ser almacenado en la colección. Esto supone dos mejoras respecto al funcionamiento de las colecciones:

# Colecciones de tipos genéricos.

---

Cualquier instrucción que intente almacenar en la colección un objeto de un tipo que no sea el especificado provocará un error de compilación.

Dado que se conoce el tipo de objeto almacenado en la colección, no será necesario realizar una conversión explícita durante su recuperación.



# Sintaxis.

---

Para especificar el tipo de objeto a utilizar en una colección se debe indicar dicho tipo en la declaración de la variable de colección:

```
tipo_colección <tipo_objeto> variable= new tipo_colección <tipo_objeto>();
```

Ejemplo:

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

# Sintaxis.

---

En el caso de una colección de tipo Hashtable, donde además de los elementos de la colección (objeto valor) se almacenan claves(objeto clave) asociadas a cada elemento, la utilización de genéricos permite especificar el tipo del elemento como clave:

```
Hashtable<String,Empleado>tb=new Hashtable<String,Empleado>;
```