

**Estructuras de datos. Paquete
java.util
© ADR Infor SL**

Indice

Competencias y Resultados de Aprendizaje desarrollados en esta unidad	3
Estructuras de datos. Paquete java.util	4
Objetivos	4
Clase StringBuffer	4
Clase StringTokenizer	8
Clase Vector	12
Clase Hashtable	20
Ejercicios	26
Ejercicio 1. StringBuffer	26
Datos a mostrar por consola:	26
Ejercicio 2. StringTokenizer	27
Datos a mostrar por consola:	27
Ejercicio 3. Vectores	28
Lo necesario para comenzar	29
Datos a mostrar por consola:	29
Ejercicio 4. Buscar ficheros	29
Lo necesario para comenzar	29
Recomendaciones	30
Datos a mostrar por consola:	30
Ejercicio 5. HashTable	31
Recomendaciones	32
Datos a mostrar por consola:	32
Ejercicio 6. Contar palabras	33
Recomendaciones	33
Datos a mostrar por consola	34
Recursos	35
Enlaces de Interés	35

Competencias y Resultados de Aprendizaje desarrollados en esta unidad

Competencia:

Conocer las estructuras de datos más comunes en Java

Resultados de Aprendizaje:

- Conocer la clase StringBuffer y sus usos
- Conocer la clase StringTokenizer y sus usos
- Conocer los vectores y sus usos
- Conocer la clase HashTable y sus usos
- Manejar correctamente la clase StringBuffer

Estructuras de datos. Paquete java.util

Objetivos

- Entender la estructura de datos que Java utiliza.
- Estudiar el paquete java.util y entender su uso.

Clase StringBuffer

Una StringBuffer es una variable referenciada asociada a un objeto de la clase `java.lang.StringBuffer`.

Es **similar a una String** y, por lo tanto, sirve para **almacenar cadenas de caracteres**, pero con **dos diferencias**:

- Su tamaño y contenido pueden modificarse en tiempo de ejecución de programa.
- Debe crearse con alguno de sus constructores asociados. No sirve declararla como si fuera una variable primitiva.

Una StringBuffer está indexada, es decir, cada uno de sus caracteres tiene asociado un índice: **0 para el primero**, 1 para el segundo, etc.

CLASE ASOCIADA

`java.lang.StringBuffer` que hereda directamente de `Object`.

CONSTRUCTORES

- **StringBuffer():** construye una StringBuffer vacía y con una capacidad por defecto de 16 caracteres. La capacidad indica la máxima cantidad de caracteres que es capaz de almacenar. Se modifica dinámicamente a medida que se va llegando al máximo. A partir de 34 caracteres la capacidad se iguala al número de caracteres de la StringBuffer.
- **StringBuffer(int capacidad):** ídem anterior pero con la capacidad que se le pasa al argumento.
- **StringBuffer(String str):** construye una StringBuffer en base a la String que se le pasa como argumento.

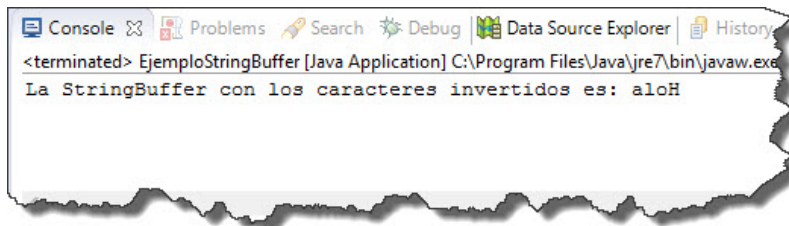
MÉTODOS PRINCIPALES

- **int capacity():** devuelve la capacidad de una StringBuffer. Es una propiedad dinámica y, por tanto, su valor puede cambiar en tiempo de ejecución.
- **int length():** devuelve el número de caracteres de la StringBuffer.
- **StringBuffer reverse():** invierte el orden de los caracteres de la StringBuffer sobre la que se aplica. No devuelve una nueva con los caracteres invertidos, sino que modifica la original.

```
package unidad09.ejemplos;

public class EjemploStringBuffer{
    public static void main(String args[]){
        StringBuffer sb=new StringBuffer("Hola");
        /* no se crea una nueva StringBuffer, sino que se modifica
        * la que se ve afectada por el método reverse. No se necesita
        * almacenar el tipo de retorno de método en otra variable.
        */
        sb.reverse();
        System.out.println("La StringBuffer con los caracteres invertidos es: " +sb);
    }
}
```

Por consola:



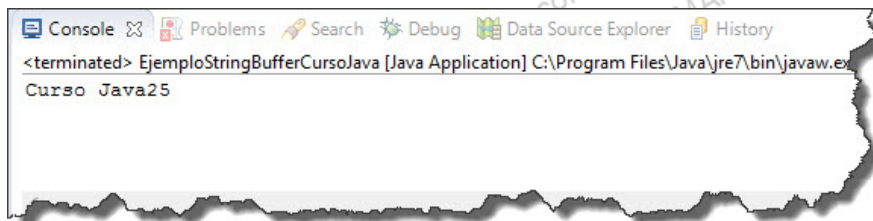
- **void setCharAt(int indice, char ch):** cambia el carácter asociado al índice indicado en el primer argumento por el carácter que se le pasa al segundo. Si el índice es negativo o superior a su (longitud -1) se lanza una StringIndexOutOfBoundsException.
- **StringBuffer deleteCharAt(int indice):** elimina el carácter de la StringBuffer indicado en el argumento.
- **char charAt(int indice):** devuelve el carácter asociado a la posición del entero que se le pasa al argumento.
- **void setLength(int nuevaLongitud):** modifica la longitud de la StringBuffer sobre la que se aplica. Si la nueva longitud es menor, se trunca.

- **String toString():** convierte una StringBuffer en una String.
- **StringBuffer append(...):** es un método sobrecargado (consultar la API). Modifica la StringBuffer sobre la que se aplica agregándole al final una String o la representación en forma de String de un dato asociado a una variable primitiva (byte, short, ..., boolean).

```
package unidad09.ejemplos;

public class EjemploStringBufferCursoJava {
    public static void main(String args[]) {
        StringBuffer sb = new StringBuffer();
        // Se emplea append(String s)
        sb.append("Curso Java");
        // Se emplea append(int i)
        sb.append(25);
        System.out.println(sb);
    }
}
```

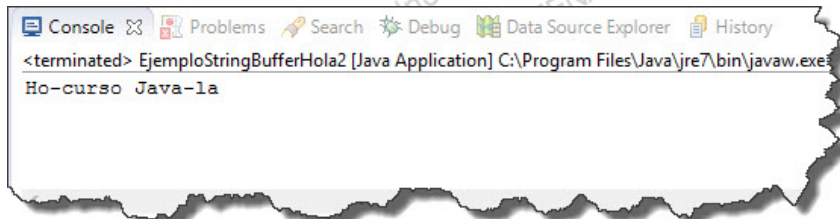
Por consola:



- **StringBuffer insert(int inicio,...):** modifica la StringBuffer sobre la que se aplica insertando justo antes de la posición especificada en el primer argumento, la String o la representación en forma de String de una variable asociada al tipo de dato primitivo especificado en el segundo argumento. Consultar la API.

```
package unidad09.ejemplos; public class EjemploStringBufferHola2 {
    public static void main(String args[]){
        StringBuffer sb=new StringBuffer("Hola");
        sb.insert(2, "-curso Java-");
        System.out.println(sb);
    }
}
```

Por consola:

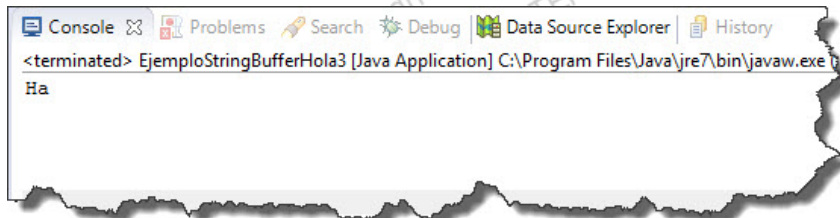


- **StringBuffer delete(int inicio, int fin):** modifica la StringBuffer sobre la que se aplica eliminando la subcadena de caracteres comprendida entre los dos índices asociados a los argumentos. En un intervalo semiabierto[...).

```
package unidad09.ejemplos;

public class EjemploStringBufferHola3 {
    public static void main(String args[]) {
        StringBuffer sb=new StringBuffer("Hola");
        sb.delete(1, 3);
        System.out.println(sb);
    }
}
```

Por consola:



- **StringBuffer replace(int indiceIni, int indiceFin, String str):** modifica la StringBuffer sobre la que se aplica reemplazando los caracteres de la subcadena comprendidos entre los índices especificados en el primer y segundo argumento por la String del tercero. Es un intervalo [...).
- **String substring(int indiceIni, int indiceFin):** devuelve una String obtenida a partir del índice inicial incluido asociado al primer argumento y del índice final excluido asociado al segundo argumento. Es un intervalo semiabierto [indiceIni, indiceFin).

Ejemplo 1: se muestra un código que trabaja con aspectos básicos de una StringBuffer. **Todos los códigos de este tema se guardarán en c:\cursojava\tema9 o en el workspace correspondiente, si se emplea IDE.**

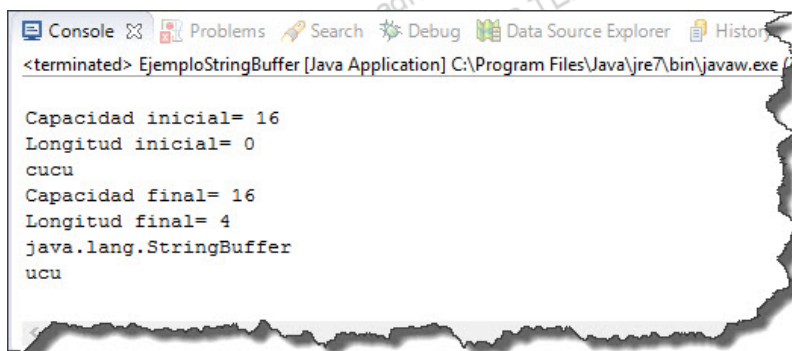
```

package unidad09.ejemplos;

public class EjemploStringBuffer{
    public static void main(String args[]){
        StringBuffer sb=new StringBuffer();
        //Por consola aparecerá una línea vacía
        System.out.println(sb);
        System.out.println("Capacidad inicial= "+sb.capacity());
        System.out.println("Longitud inicial= "+sb.length());
        sb.append("cucu");
        System.out.println(sb);
        System.out.println("Capacidad final= "+sb.capacity());
        System.out.println("Longitud final= "+sb.length());
        /*El método "Class getClass()" pertenece a la clase Object
        *y, por tanto, puede aplicarse a cualquier objeto que herede de
        *Object. El método "String getName()" devuelve el nombre de la
        *clase a la que pertenece el objeto sb.
        */
        System.out.println(sb.getClass().getName());
        sb.deleteCharAt(0);
        System.out.println(sb);
    }
}

```

Por consola:



```

<terminated> EjemploStringBuffer [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe

Capacidad inicial= 16
Longitud inicial= 0
cucu
Capacidad final= 16
Longitud final= 4
java.lang.StringBuffer
ucu

```

Ahora conviene hacer el primer ejercicio del tema

Clase StringTokenizer

Una `StringTokenizer` es una variable referenciada asociada a un objeto de la clase `java.util.StringTokenizer`. Se utiliza como analizador de Strings del siguiente modo:

Divide una String en tokens. Estos tokens son **cadenas de texto** generadas **en base a un carácter delimitador** que, por defecto, es el espacio en blanco.

Ejemplo:

Dada la cadena de texto **`String texto="Hoy es sabado y mañana domingo"`**

Si se crea una `StringTokenizer` en base a esta String, se tienen seis tokens que se corresponden con cada una de las palabras que la componen.

**Hoy
es
sabado
y
mañana
domingo**

Como carácter delimitador de los tokens, por defecto, se emplea el espacio en blanco.

En cambio, si se utilizará 'n' como carácter delimitador se tendrían tres tokens:

**Hoy es sabado y maña
a domi
go**

CLASE ASOCIADA

`java.util.StringTokenizer` que hereda directamente de `Object` e **implementa** la interface **`java.util Enumeration`**. Esto implica que cualquier objeto `StringTokenizer` puede usar los métodos declarados en la interface. Ir a la API.

CONSTRUCTORES

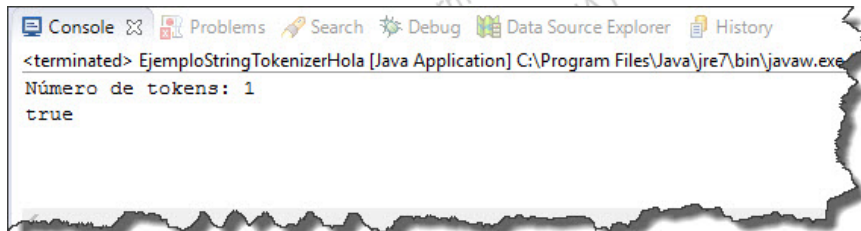
- **`StringTokenizer(String str)`**: crea una `StringTokenizer` en base a la String asociada al argumento. El delimitador de tokens por defecto es el espacio en blanco.
- **`StringTokenizer(String str, String delim)`**: crea una `StringTokenizer` en base a la String asociada al primer argumento y especificando en el segundo argumento el delimitador de tokens deseado.

MÉTODOS PRINCIPALES

- **`boolean hasMoreTokens()`**: devuelve **true si se detecta algún token** en la `StringTokenizer` sobre la que se aplica; si no, devuelve false. Hay que tener en cuenta lo siguiente: si se construye una `StringTokenizer` mediante una String compuesta de una palabra, pese a no aparecer el carácter delimitador, el método devuelve true. Suele emplearse dentro de un bucle para recorrer los tokens que componen la `StringTokenizer`.

```
package unidad09.ejemplos;
import java.util.StringTokenizer;
public class EjemploStringTokenizerHola{
    public static void main(String args[]){
        String texto = "Hola";
        StringTokenizer st = new StringTokenizer(texto);
        System.out.println("Número de tokens: " + st.countTokens());
        System.out.println(st.hasMoreTokens());
    }
}
```

Por consola:



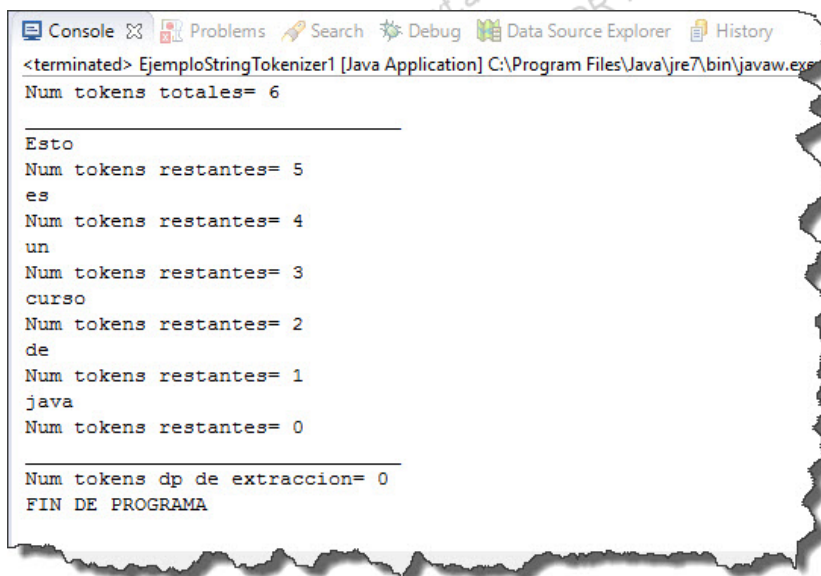
- **int countTokens():** devuelve el número de tokens de la StringTokenizer sobre la que se aplica en base al delimitador empleado en el constructor.
- **String nextToken():** devuelve una String asociada al token de la StringTokenizer que se está analizando y provoca el avance al siguiente token.

Ejemplo 1: se muestra un código que trabaja con aspectos básicos de una StringTokenizer

```

package unidad09.ejemplos;
import java.util.*;
public class EjemploStringTokenizer1 {
    public static void main(String args[]){
        String texto="Esto es un curso de java";
        StringTokenizer st=new StringTokenizer(texto);
        System.out.println("Num tokens totales= "+st.countTokens());
        System.out.println("_____");
        while(st.hasMoreTokens()){
            System.out.println(st.nextToken());
            System.out.println("Num tokens restantes= "+st.countTokens());
        }
        System.out.println("_____");
        System.out.println("Num tokens dp de extraccion= "+st.countTokens());
        /*
        * Si se repite el bucle, no se muestra nada, pues la StringTokenizer
        * sobre la que se aplica el método hasMoreTokens() ya no contiene
        * tokens
        */
        while(st.hasMoreTokens()){
            System.out.println(st.nextToken());
            System.out.println("Num tokens= "+st.countTokens());
        }
        System.out.println("FIN DE PROGRAMA");
    }
}

```

Por consola:


```

<terminated> EjemploStringTokenizer1 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe
Num tokens totales= 6

_____
Esto
Num tokens restantes= 5
es
Num tokens restantes= 4
un
Num tokens restantes= 3
curso
Num tokens restantes= 2
de
Num tokens restantes= 1
java
Num tokens restantes= 0

_____
Num tokens dp de extraccion= 0
FIN DE PROGRAMA

```

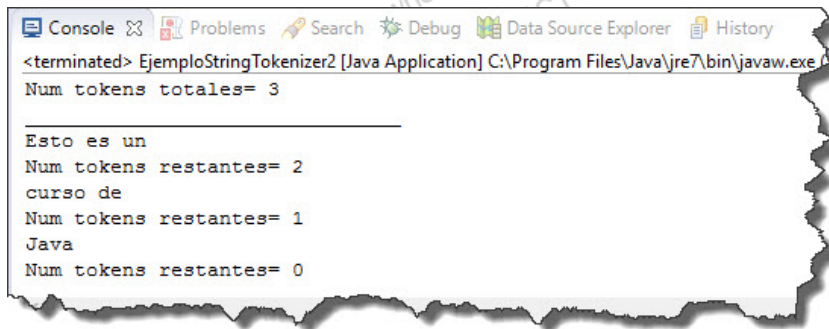
Ejemplo 2: ahora el carácter delimitador de tokens va a ser el asterisco (*)

```

package unidad09.ejemplos;
import java.util.*;
public class EjemploStringTokenizer2{
    public static void main(String args[]){
        String texto="Esto es un*curso de*Java";
        StringTokenizer st=new StringTokenizer(texto,"*");
        System.out.println("Num tokens totales= "+st.countTokens());
        System.out.println("_____");
        while(st.hasMoreTokens()){
            System.out.println(st.nextToken());
            System.out.println("Num tokens restantes= "+st.countTokens());
        }
    }
}

```

Por consola:



```

<terminated> EjemploStringTokenizer2 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe
Num tokens totales= 3
_____
Esto es un
Num tokens restantes= 2
curso de
Num tokens restantes= 1
Java
Num tokens restantes= 0

```

Ahora conviene hacer el segundo ejercicio

Clase Vector

Las clases Vector y Hashtable han sido las precursoras de otras más modernas y eficaces que han ido surgiendo en las sucesivas versiones del J2SE. Durante el curso se estudiarán estas dos clases como ejemplo de otras más modernas de similar funcionamiento.

Un Vector es una estructura de almacenamiento de objetos que pueden pertenecer a distintas clases, cuyo número de elementos puede modificarse en tiempo de ejecución, en función de las necesidades de almacenamiento del programador.

Los elementos de un Vector están indexados al igual que los de un array.

Diferencias entre un array y un Vector:

- Un array sólo puede almacenar datos asociados al mismo tipo de variable primitiva o también objetos pero pertenecientes a la misma clase. Un Vector sólo puede contener objetos aunque pueden ser de distintas clases. Nunca variable primitivas.
- El número de elementos de un array es fijo una vez construido; el de un Vector puede variar.
- Distinta forma de construcción: un array utiliza [] y se inicializa nombrando a cada elemento mientras que un Vector emplea el constructor que interese al programador y se inicializa mediante el método "void addElement(Object obj)"

CLASE ASOCIADA

java.util.Vector. Implementa por construcción varias interfaces, una muy importante es Collection. Declara métodos que permiten trabajar cómodamente con estructuras que almacenan un conjunto de elementos.

el paquete **java.util** es muy importante ya que **contiene clases e interfaces** utilizadas para trabajar con estructuras de almacenamiento dinámicas o **colecciones** (su número de elementos puede variar en tiempo de ejecución de programa).

Actualmente se utilizan clases más modernas, así se prefiere la clase ArrayList o LinkedList a Vector y HashMap, LinkedHashMap o TreeMap a Hashtable.

Los métodos y la utilidad de estas clases son muy similares a los de sus precursoras. Por ello y dado que esto es un curso de iniciación, se ha optado por estudiar sólo las clases Vector y Hashtable.

El trabajo con colecciones se basa en un grupo de interfaces de java.util que son implementadas, por construcción, por la mayoría de clases relacionadas con colecciones. Estas interfaces se dividen en dos grandes conjuntos:

- Aquellas que **heredan de Collection**: las clases que las implementan representan una colección de objetos simples. Collection, a su vez, tienen dos subinterfaces List y Set. Una clase que implementa List se comporta como una lista de cosas que admite repetición. Ejemplos: Vector, ArrayList y LinkedList. Una clase que implementa Set se comporta como una lista de cosas que no admite repetición. Ejemplos: HashSet, LinkedHashSet, TreeSet, etc.
- Aquellas que **heredan de Map**: las clases que las implementan representan una colección de objetos en forma de pares clave/valor. Ejemplos: Hashtable, HashMap, LinkedHashMap, TreeMap, etc.

CONSTRUCTORES

- **Vector()**: crea un Vector sin elementos con una capacidad inicial de 10. La capacidad indica el número de elementos que puede almacenar el Vector. El concepto de capacidad es similar al de una StringBuffer. Si se necesita más, se va aumentando automáticamente.
- **Vector(int capacidadInicial)**: ídem que el anterior pero fijando la capacidad. Si se necesita más, se va aumentando automáticamente.

MÉTODOS

- **int capacity()**: devuelve la capacidad del Vector.
- **int size()**: devuelve el número de elementos del Vector.
- **void addElement(Object obj)**: es uno de los métodos empleados para agregar objetos a un Vector. Lo agrega al final del último introducido e incrementa en uno su tamaño. Como todo objeto, independientemente de la clase a la que pertenezca, hereda de la clase Object (clase raíz), cumple los requisitos del argumento. **MUY USADO**

El casting a Object se realiza de forma automática, no es necesario realizar un casting explícito a Object cuando se hace un **Cast** de una subclase (por ejemplo String o cualquier otra) a una superclase (Object).

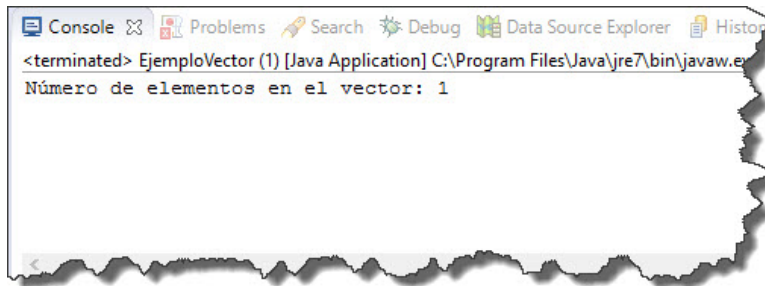
En cambio, sí es necesario hacer **Cast** de una superclase a una subclase; si no se producirá un error de compilación.

Esto se verá al estudiar los métodos **Object clone()** y el método **Object elementAt(Object obj)**.

```
package unidad09.ejemplos;
import java.util.*;

public class EjemploVector{
    public static void main(String args[]){
        Vector v=new Vector();
        //Añade un elemento al vector v
        v.addElement("hola");
        System.out.println("Número de elementos en el vector: " + v.size()
    );
    }
}
```

Por consola:



- **Object clone():** realiza una copia del vector sobre el que se aplica.

```
package unidad09.ejemplos;
import java.util.*;

public class EjemploVector{
    public static void main(String args[]){
        Vector v=new Vector();
        //Añade un elemento al vector v
        v.addElement("hola");
        v.addElement(new Float((float)12));
        /*
         * Hay que hacer un casting entre clases para que el
         * Object que devuelve el método se convierta en un Vector.
         * Este casting es posible cuando entre las clases
         * implicadas (Object y Vector en este caso)
         * existe una relación de herencia.
         * Así, por ejemplo, no puede realizarse un casting entre
         * una instancia de Integer y una String pues ambas
         * no están ligadas por herencia
         */
        Vector vCopia = (Vector)v.clone();
    }
}
```

- **Object elementAt(int índice):** devuelve el elemento del Vector asociado al índice que se le pasa al argumento, en forma de Object. Si se quiere recuperar el elemento del Vector con el tipo que se agregó, debe hacerse un casting. Empleado para mostrar los elementos del Vector con ayuda de un bucle for. **MUY USADO**

viewnext.adrformacion.com © ADR info
VICTOR TENA PALOMARES

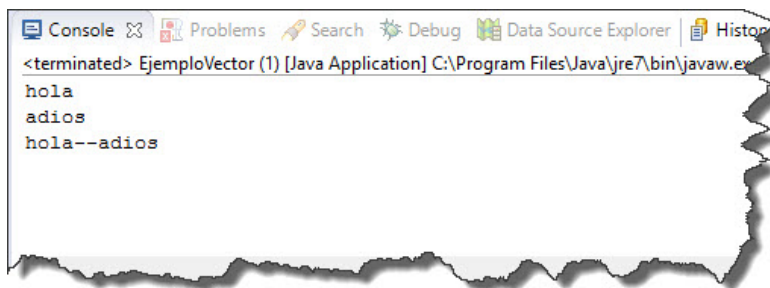
```

package unidad09.ejemplos;
import java.util.*;

public class EjemploVector{
    public static void main(String args[]){
        Vector v=new Vector();
        //Añade un elemento al vector v
        v.addElement("hola");
        v.addElement("adios");
        // Para mostrar por consola no es necesario el Cast a String
        for(int i=0;i<v.size();i++){
            System.out.println(v.elementAt(i));
        }
        /*
        * Si se quiere recuperar algún elemento del Vector tal y como
        * fue agregado debe hacerse un Cast.
        */
        String saludo = (String)v.elementAt(0);
        String despedida = (String)v.elementAt(1);
        System.out.println(saludo + "--" + despedida);
    }
}

```

Por consola



- **void removeElement(Object obj):** elimina el primer objeto del Vector coincidente con el argumento del método y desplaza el resto de elementos. Para eliminar todos, se usaría un bucle for.


```

package unidad09.ejemplos;
import java.util.*;

public class EjemploVector{
    public static void main(String args[]){
        Vector v=new Vector();
        //Añade un elemento al vector v
        v.addElement("hola");
        v.addElement("adios");
        v.addElement("hola");
        System.out.println("El número de elementos del vector es:v" + v.size());
        v.remove(0);
        System.out.println("El número de elementos del vector es:v" + v.size());
    }
}

```

Por consola:



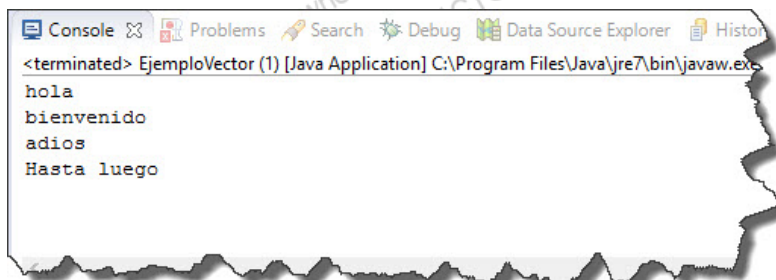
- **void removeElementAt(int índice):** elimina el elemento del Vector asociado al índice que se le pasa al argumento.
- **void removeAllElements():** elimina todos los elementos del Vector sobre el que se aplica.
- **boolean contains(Object obj):** indica si el Vector sobre el que se aplica contiene o no el objeto que se le pasa al argumento. Si lo contiene devuelve true, sino lo contiene devuelve false.
MUY USADO
- **boolean isEmpty():** devuelve true si el Vector sobre el que se aplica no tiene elementos.
- **Object firstElement():** devuelve el primer elemento del Vector como un Object.
- **Object lastElement():** devuelve el último elemento del Vector como un Object.
- **void setElementAt(Object obj, int índice):** cambia el elemento del Vector asociado al índice del segundo argumento por el objeto asociado al primer argumento del método.

- **void insertElementAt(Object obj, int índice):** inserta el elemento asociado al primer argumento en la posición inmediatamente anterior al índice asociado al segundo argumento.

```
package unidad09.ejemplos;
import java.util.Vector;

public class EjemploVector {
    public static void main(String args[]) {
        Vector<String> v = new Vector<String>();
        // Añade un elemento al vector v
        v.addElement("hola");
        v.addElement("hasta luego");
        v.insertElementAt("bienvenido", 1);
        v.insertElementAt("adios", 2);
        for (int i=0;i<v.size();i++) {
            System.out.println(v.elementAt(i));
        }
    }
}
```

Por consola:



- **int indexOf(Object obj):** devuelve el índice asociado a la primera vez que aparece el elemento que se le pasa al argumento.
- **Enumeration elements():** devuelve una interface Enumeration que contiene los elementos del Vector sobre el que se aplica. Son métodos que permiten recorrer el Vector y acceder a sus elementos. No suele emplearse mucho. Son los siguientes:
 - **boolean hasMoreElements():** devuelve true si el objeto Enumeration tiene elementos.
 - **Object nextElement():** devuelve el siguiente elemento del objeto Enumeration sobre el que se aplica y provoca el avance al siguiente elemento.

Se trata de crear un array de enteros que contenga los diez primeros números naturales sin contar el cero.

Luego, se crea un Vector y se inicializa con los elementos del array, adecuadamente convertidos en objetos, para que puedan formar parte del Vector.

Finalmente, se muestran por consola sus elementos.

```
package unidad09.ejemplos;
import java.util.*;

public class EjemploVector1 {
    public static void main(String args[]) {
        int enteros[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        Vector v = new Vector();
        for (int i = 0; i < enteros.length; i++) {
            v.addElement(new Integer(enteros[i]));
            System.out.print(v.elementAt(i) + " ");
        }
        System.out.println("\nFIN DE PROGRAMA");
    }
}
```

Por consola:



Ahora se pretende crear un Vector que contenga una String, una StringBuffer y un Integer.

A continuación se muestran sus elementos.

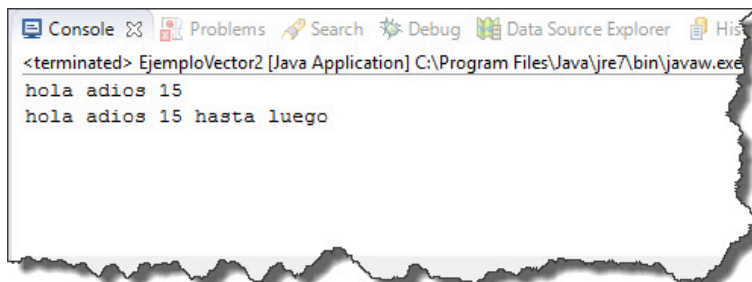
Luego, se crea una copia del Vector, y a la copia, se le agrega una String.

Finalmente se mostrarán por consola los elementos del Vector copia.

```
package unidad09.ejemplos;
import java.util.*;

public class EjemploVector2{
    public static void main(String args[]){
        Vector v=new Vector();
        v.addElement("hola");
        v.addElement(new StringBuffer("adios"));
        v.addElement(new Integer(15));
        for(int i=0;i<v.size();i++)
            System.out.print(v.elementAt(i)+" ");
        System.out.println();
        //Casting obligatorio
        Vector vClon=(Vector)v.clone();
        vClon.addElement("hasta luego");
        for(int i=0;i<vClon.size();i++)
            System.out.print(vClon.elementAt(i)+" ");
    }
}
```

Por consola:



Ahora conviene hacer el tercer ejercicio

Clase Hashtable

Es una **estructura de datos que permite relacionar una clave con un valor** de forma que, mediante la clave se obtiene el valor. Un Hashtable es algo similar a una tabla con dos columnas: la primera corresponde a las claves y la segunda a los valores. Cada fila de esta tabla sería un elemento del Hashtable o un par clave/valor.

Ejemplo:

CLAVE	VALOR
16592696	APTO
16606886	APTO
16587456	NO APTO
16325745	NO APTO

Cualquier **objeto distinto de null puede ser tanto clave como valor**. Para poder acceder a un valor almacenado en la tabla, debe conocerse la clave. Asociado a un Hashtable se tienen dos propiedades a tener en cuenta:

Capacidad:

- asociada a la cantidad de datos (pares clave/valor) que puede almacenar un Hashtable.
- si el número de elementos del Hashtable es mayor que la capacidad por el factor de crecimiento, la capacidad inicial del Hashtable aumenta adecuadamente para no tener problemas de saturación, realizando una llamada automática al método `rehash()`.

Factor de crecimiento o de carga (load factor):

- asociado a lo que debe aumentar la capacidad de un Hashtable cuando se está cerca del límite de su capacidad inicial. Es un tipo de dato primitivo `float` que pertenece al intervalo `[0,1]`.
- no conviene asignar a un Hashtable factores de carga muy superiores a `0.75` ya que se ralentiza la búsqueda de los valores.

una clave puede repetirse aunque no tiene mucho sentido hacerlo. Cuando ocurra, el valor asociado a una clave repetida, se corresponde al último elemento introducido en el Hashtable.

Un valor también puede repetirse. Esto si tiene más sentido como se ha visto en el ejemplo anterior

CLASE ASOCIADA

`java.util.Hashtable`. Implementa por construcción varias interfaces. Consultar la API. Una muy importante es `Map`. Declara métodos que permiten trabajar cómodamente con estructuras que almacenan datos en forma de pares clave/valor.

CONSTRUCTORES

- **Hashtable():** crea un Hashtable vacío con una capacidad de 10 y un factor de carga por defecto de 0.75.
- **Hashtable(int capacidadInicial):** ídem anterior, pero fijando la capacidad inicial.
- **Hashtable(int capacidadInicial, float factorCarga):** ídem anterior, pero fijando el factor de carga mediante el segundo argumento.

MÉTODOS

- **int size():** devuelve el número de elementos o tamaño del Hashtable (número de pares clave/valor).
- **Object put(Object clave, Object valor):** inicializa un Hashtable con pares clave/valor. **MUY USADO.**
- **Object get(Object clave):** devuelve el valor asociado a la clave que se le pasa al argumento. Si la clave no existe, devuelve null. **MUY USADO.**
- **boolean isEmpty():** devuelve true si el Hashtable sobre el que se aplica no tiene elementos. Si no, false.
- **void clear():** elimina todos los pares clave/valor del Hashtable sobre el que se aplica.
- **Object remove(Object clave):** elimina el par clave/valor en base a la clave que se le pasa al argumento. Si hay varios pares con una misma clave, se eliminan.
- **boolean containsKey(Object clave):** devuelve true si la clave que se le pasa al argumento existe; false en caso contrario.
- **boolean contains(Object valor):** devuelve true si hay alguna clave asociada con el valor que se le pasa al argumento; false en caso contrario.
- **Enumeration keys():** devuelve una interface Enumeration que contiene las claves del Hashtable.
- **Enumeration elements():** devuelve una interface Enumeration que contiene los valores del Hashtable. Enumeration es una interface perteneciente al paquete java.util que declara métodos que permiten recorrer una colección de objetos. Estos métodos son necesarios ya que un Hashtable no está indexado como un Vector o un array. Son los que se explicaron cuando se estudió la clase Vector:
 - **boolean hasMoreElements():** devuelve true si el objeto Enumeration tiene elementos.

- **Object nextElement():** devuelve el siguiente elemento del objeto Enumeration sobre el que se aplica y provoca el avance al siguiente par clave/valor.

Se muestra un código que trabaja con aspectos básicos de un Hashtable. Se parte de una clase pública de nombre EjemploHashtable1 con un solo método: el main. Deben realizarse las siguientes operaciones:

- Crear un Hashtable h con el constructor sin argumentos y mostrar su tamaño por consola.
- Añadir pares clave/valor expresados en forma de String siguiendo el esquema:

Clave	Valor
1	uno
2	dos
3	tres
4	cuatro
5	cinco
6	seis

- Mostrar por consola los pares clave/valor del Hashtable y su tamaño.
- Agregar al Hashtable tres pares clave/valor que continúen la secuencia inicial y volver a mostrar por consola todos los pares clave/valor, pero usando los métodos de la interface Enumeration. Además, mostrar el nuevo tamaño.
- Almacenar en las variables mediante las que fueron agregadas al Hashtable los valores correspondientes a las claves impares.
- Después mostrarlos por consola.
- Eliminar los pares clave/valor asociados a las claves 1 y 3 y mostrar el nuevo tamaño del Hashtable.
- Eliminar todos los pares clave/valor del Hashtable y mostrar el nuevo tamaño del Hashtable.

```
package unidad09.ejemplos;

import java.util.Enumeration;
import java.util.Hashtable;

public class EjemploHashtable1 {
    public static void main (String args[]){
        // Crear Hashtable y mostrar su tamaño inicial
        Hashtable h = new Hashtable();
        System.out.println("Tamaño inicial: " + h.size());
        // Agregar pares claves/valor
        System.out.println("*****");
        System.out.println("Pares del Hashtable: ");
        h.put(1, "uno");
        h.put(2, "dos");
        h.put(3, "tres");
```

```

h.put(4, "cuatro");
h.put(5, "cinco");
h.put(6, "seis");
// Mostrar directamente y sin orden los pares del Hashtable

System.out.println(h);
System.out.println("Tamaño: " + h.size());
System.out.println("*****");
// Agregar tres pares nuevos
h.put(7, "siete");
h.put(8, "ocho");
h.put(9, "nueve");

Enumeration claves = h.keys();
Enumeration valores = h.elements();

/*
 * Mostrar los pares del hashtable con los métodos
 * de enumeration
 */
System.out.println("*****");
System.out.println("Nuevos Pares del Hashtable: ");
while(claves.hasMoreElements()){
    System.out.println(claves.nextElement() + "-" + valores.nextElement());
}
System.out.println("Tamaño: " + h.size());

// Extraer los valores de las claves impares
System.out.println("*****");
System.out.println("Valores claves impares del Hashtable: ");
String valor1 = (String)h.get(1);
String valor3 = (String)h.get(3);
String valor5 = (String)h.get(5);
String valor7 = (String)h.get(7);
String valor9 = (String)h.get(9);
System.out.println(valor1 + " " + valor3 + " " +
    valor5 + " " + valor7 + " " + valor9 );
System.out.println("*****");

// Eliminar los pares 1 y 3
System.out.println("*****");
System.out.println("Eliminar los pares 1 y 3 del Hashtable: ");
h.remove(1);
h.remove(3);
System.out.println("Tamaño: " + h.size());
System.out.println("*****");

// Eliminar todos los pares
System.out.println("*****");
System.out.println("Eliminar todos los pares del Hashtable: ");
h.clear();
System.out.println("Tamaño: " + h.size());
System.out.println("FIN DEL PROGRAMA");

```

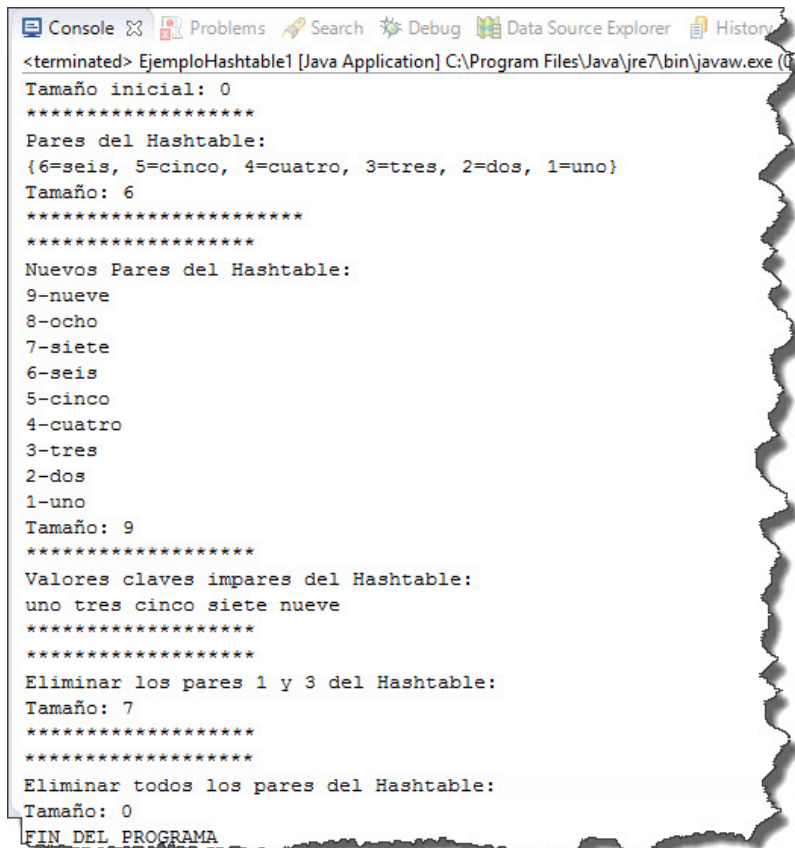


```

    }
}

```

Por consola:



```

<terminated> EjemploHashtable1 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (0
Tamaño inicial: 0
*****
Pares del Hashtable:
{6=seis, 5=cinco, 4=cuatro, 3=tres, 2=dos, 1=uno}
Tamaño: 6
*****
*****
Nuevos Pares del Hashtable:
9-nueve
8-ocho
7-siete
6-seis
5-cinco
4-cuatro
3-tres
2-dos
1-uno
Tamaño: 9
*****
Valores claves impares del Hashtable:
uno tres cinco siete nueve
*****
*****
Eliminar los pares 1 y 3 del Hashtable:
Tamaño: 7
*****
*****
Eliminar todos los pares del Hashtable:
Tamaño: 0
FIN DEL PROGRAMA

```

Ahora conviene hacer el resto de ejercicios del tema.

Ejercicios

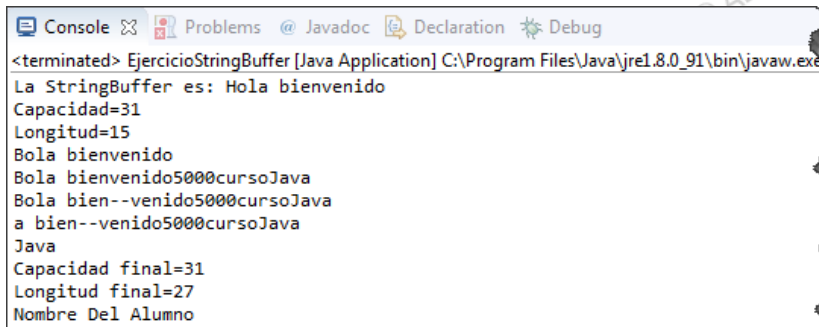
Ejercicio 1. StringBuffer

35

Realizar un programa que contenga una clase pública de nombre **EjercicioStringBuffer** con un solo método: el main. Su código debe hacer lo siguiente:

- **Crear una StringBuffer** mediante el tercer constructor y la String "Hola, bienvenido" y **mostrarla por consola**.
- **Mostrar por consola su capacidad y longitud**.
- **Sustituir el primer carácter** de la StringBuffer anterior **por 'B'** y **mostrarla por consola**.
- **Agregar al final** de la StringBuffer anterior **el numero entero 5000** almacenado en una variable primitiva int y la String "cursoJava" y **mostrarla por consola**.
- **Insertar** en la StringBuffer anterior y justo después de la subpalabra "bien" la String "--". **Mostrarla por consola**.
- **Eliminar los tres primeros caracteres** de la anterior StringBuffer y **mostrarla por consola**.
- **Almacenar** en una String **los cuatro últimos caracteres** de la StringBuffer resultante y **mostrarla la String por consola**.
- **Mostrar por consola la capacidad y la longitud** de la StringBuffer final.
- **Crear un array** de tres StringBuffer **que contenga vuestro nombre, primer apellido y segundo apellido**. Luego **mostrar por consola sus elementos**.

Datos a mostrar por consola:



```

<terminated> EjercicioStringBuffer [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
La StringBuffer es: Hola bienvenido
Capacidad=31
Longitud=15
Bola bienvenido
Bola bienvenido5000cursoJava
Bola bien--venido5000cursoJava
a bien--venido5000cursoJava
Java
Capacidad final=31
Longitud final=27
Nombre Del Alumno

```

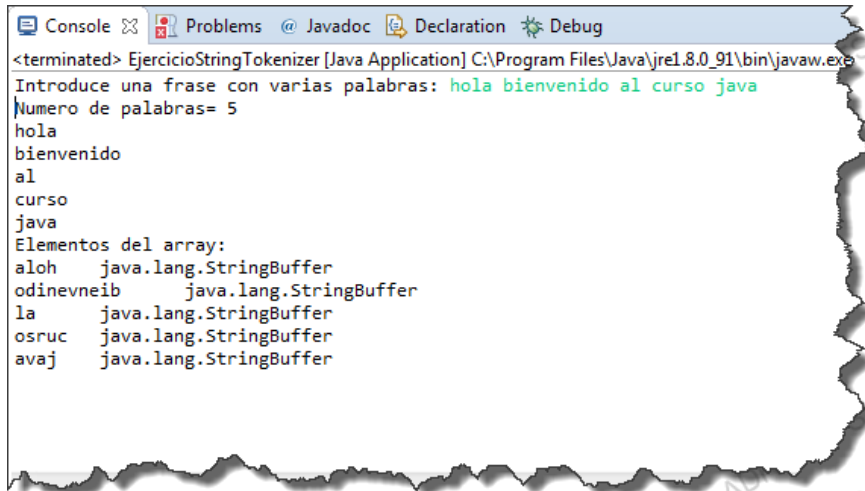
Ejercicio 2. StringTokenizer

25

Realizar un programa que contenga una clase pública de nombre **EjercicioStringTokenizer** con un solo método: el main. Su código debe hacer lo siguiente:

- **Solicitar** al usuario **una frase** con varias palabras
- **Mostrar** el **número de palabras** de la frase.
- **Mostrar** cada **palabra** en una **línea diferente**.
- **Almacenar** en un array de StringBuffer **las palabras** de la frase.
- **Modificar** las StringBuffer del array haciendo que **la primera letra pase a ser la última, la segunda la penúltima**, etc.
Ejemplo: si la frase es "Hola que tal estas", el array modificado debe contener [aloH, euq, lat, satse]
- **Mostrar los elementos del array anterior y la clase** a la que pertenecen mediante los métodos "Class getClass()" de java.lang Object y "String getName()" de java.lang.Class

Datos a mostrar por consola:



```

<terminated> EjercicioStringTokenizer [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Introduce una frase con varias palabras: hola bienvenido al curso java
Numero de palabras= 5
hola
bienvenido
al
curso
java
Elementos del array:
aloh      java.lang.StringBuffer
odinevneib java.lang.StringBuffer
la        java.lang.StringBuffer
osruc     java.lang.StringBuffer
avaj      java.lang.StringBuffer

```

Ejercicio 3. Vectores

40

Realizar un programa que contenga una clase pública de nombre **EjercicioVector1** con un solo método: el main. Su código debe hacer lo siguiente:

- **Crear un Vector v** utilizando el primer constructor, **que contenga a los planetas** del sistema solar ordenados alfabéticamente (Jupiter, Marte, Mercurio, Neptuno, Pluton, Saturno, Tierra, Urano, Venus) en forma de String
- **Mostrar la capacidad y tamaño** originales.
- **Comprobar** si el Vector **contiene** una String de nombre **“Saturno”**. Si la contiene, **mostrar** por consola la **posición** que ocupa dentro del Vector.
- **Mostrar** por consola el **primer** y **último** elemento del Vector.
- **Crear una StringBuffer** a partir de la String **“Madrid”**, **insertarla en el Vector** justo después del elemento **“Tierra”** y **mostrar** por consola la **capacidad** y el **tamaño** del Vector una vez insertada la StringBuffer
- **Agregar** al Vector **después del último** elemento **un objeto Integer** de nombre **“edad”** que almacene vuestra edad.
- **Mostrar** por consola los **nuevos elementos** del Vector.
- **Eliminar todos los elementos** del Vector.
- **Mostrar** por consola la **capacidad** y el **tamaño** del Vector **después de eliminar** todos sus elementos

Lo necesario para comenzar

1. Si los planetas no se agregan al **Vector** por orden alfabético, pueden ordenarse mediante el método estático **void sort(List lista)** de la clase **Collections** de java.util.
2. El argumento del método debe ser una objeto de la interface List (equivale a una clase que implemente la interface List). Como la clase Vector implementa por defecto dicha interface (ir a la API) es consistente pasarle al método un Vector. Muy usado para ordenar vectores que almacenan cadenas de texto.
3. Es similar al método sort(..) de la clase Arrays.

Datos a mostrar por consola:

```

<terminated> EjercicioVector1 [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Capacidad= 10
Num elem= 9

El vector contiene a Saturno
Posicion de Saturno= 5

Primer elemento= Jupiter
Ultimo elemento= Venus

Agregando Madrid
Nueva capacidad= 10
Nuevo num elem= 10

Agregando objeto Integer 28
Elementos del vector:
*****
Jupiter Marte Mercurio Neptuno Pluton Saturno Tierra LogroRo Urano Venus 28
*****

Ahora los borro todos
Capacidad final= 20
Num final elem= 0

```

Ejercicio 4. Buscar ficheros

30

Realizar un programa, siguiendo la estructura de métodos propuesta, que busque ficheros de una determinada extensión en un directorio del sistema local.

El programa se ejecutará desde consola pasándole como parámetros el directorio y la extensión.

Deberá mostrar por consola el listado de ficheros encontrados ordenados alfabéticamente y su número.

Lo necesario para comenzar

Se puede utilizar el siguiente esqueleto para crear la clase:

```
package unidad09.ejercicios;
import java.io.*;
import java.util.*;

public class BuscaFicheros{
    public static void main(String[] args){
        /*
         * Muestra por consola los ficheros.
         * Cada uno de ellos son elementos de un Vector llamado 'resultados' que
         * se obtiene después de invocar al método obtenerFicherosBuscados(...)
         * pasándole el directorio donde se busca y la extensión de los ficheros
         * buscados.
         * Si el usuario no introduce dos parámetros durante la ejecución
         * se informará de ello y se explicará cómo debe ejecutarse.
         */
    }

    private static boolean comprobarExtension(File f,String ext){
        /*
         * Asegura que el objeto File del primer argumento (representará
         * a un fichero) tiene la extensión especificada en el segundo.
         */
    }

    private static Vector obtenerFicherosBuscados(String dir,String ext){
        /*
         * Devuelve un Vector que contiene todos los ficheros de la extensión
         * buscada en el directorio especificado. Hace uso del método
         * comprobarExtension(...).
         */
    }
}
```

Recomendaciones

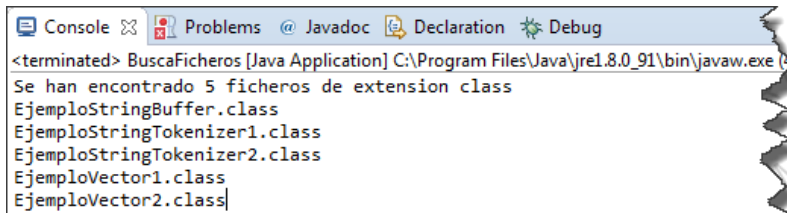
Por ejemplo, para buscar ficheros de extensión .class en el directorio asociado al tema 8, el código debería ejecutarse así:

```
java BuscaFicheros c:\cursojava\bin\unidad09\ejemplos.class
```

El directorio bin es el directorio de compilación de nuestro proyecto.

Datos a mostrar por consola:

Si se ejecuta de forma correcta:

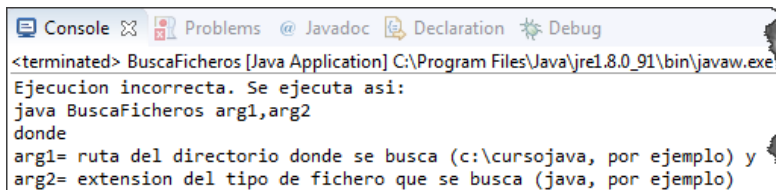


```

<terminated> BuscaFicheros [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Se han encontrado 5 ficheros de extension class
EjemploStringBuffer.class
EjemploStringTokenizer1.class
EjemploStringTokenizer2.class
EjemploVector1.class
EjemploVector2.class

```

Si se ejecuta de forma incorrecta, por ejemplo pasando menos de dos parámetros (si se le pasan más de dos sólo se consideran los dos primeros):



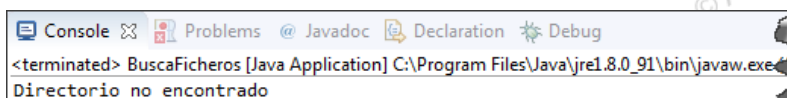
```

<terminated> BuscaFicheros [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Ejecucion incorrecta. Se ejecuta asi:
java BuscaFicheros arg1,arg2
donde
arg1= ruta del directorio donde se busca (c:\cursojava, por ejemplo) y
arg2= extension del tipo de fichero que se busca (java, por ejemplo)

```

Si se ejecuta de forma incorrecta, por ejemplo pasando un directorio de búsqueda que no existe:

Por consola:



```

<terminated> BuscaFicheros [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Directorio no encontrado

```

Ejercicio 5. HashTable

Realizar un programa que contenga una clase pública de nombre **EjercicioHashtable1** con un solo método: el main.

Su código **permitirá saber** a un usuario **si ha aprobado o suspendido un examen**.

El **funcionamiento** será el siguiente:

- El programa solicitará una clave al usuario y:
 - Si el usuario introduce erróneamente su clave, el programa debe mostrar "Clave incorrecta" y finalizar.
 - Si la introduce correctamente, se mostrará su calificación y un mensaje de "Enhorabuena" si ha aprobado o un "Lo siento" si ha suspendido.
- Después el programa finalizará.

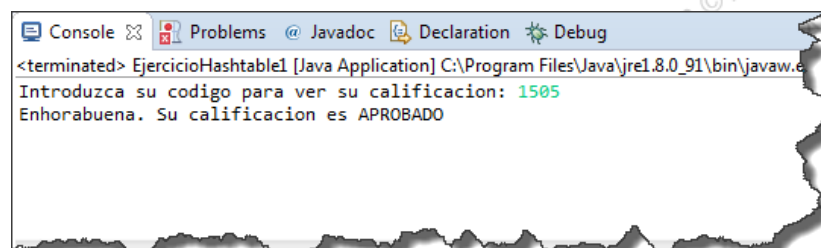
Recomendaciones

El Hashtable tendrá una capacidad de 10 y un factor de carga del 80% y contendrá los pares clave/valor indicados en la siguiente tabla.

Clave	Valor
1505	APROBADO
2800	SUSPENSO
1350	SUSPENSO
4200	APROBADO
3200	APROBADO

Datos a mostrar por consola:

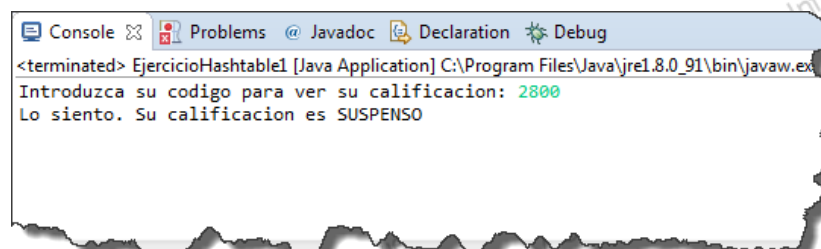
Por consola si se introduce una clave cuyo valor es APROBADO:



```

<terminated> EjercicioHashtable1 [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Introduzca su codigo para ver su calificacion: 1505
Enhorabuena. Su calificacion es APROBADO
  
```

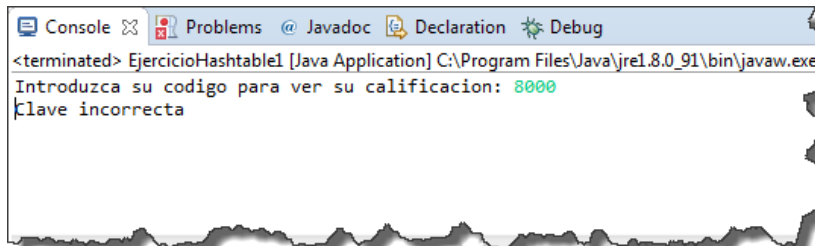
Por consola si se introduce una clave cuyo valor es SUSPENSO:



```

<terminated> EjercicioHashtable1 [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Introduzca su codigo para ver su calificacion: 2800
Lo siento. Su calificacion es SUSPENSO
  
```


Por consola si se introduce una clave que no existe:



Ejercicio 6. Contar palabras

30

Realizar un programa que contenga una clase pública de nombre CuentaPalabras con un solo método: el main.

El programa **se ejecutará desde consola** pasándole como parámetros la ruta del fichero que se lee y la del fichero en el que se escribe.

Su código hará lo siguiente:

- **Leer un fichero** de texto sin formato del sistema local que no contenga comas ni puntos ni puntos y coma ni dos puntos, etc. sólo palabras.
- **Contar** el número de veces que se **repite** cada palabra.
- **Ordenar alfabéticamente** las palabras.
- **Escribir** en otro fichero de nombre contador.txt **la palabra y el número de veces que aparece**.

Recomendaciones

1. En este ejercicio se recomienda el uso de **StringTokenizer**, **Vector**, **Hashtable**, el método estático **sort(..)** de la clase **Collections** y las clases del paquete **java.io** para leer y escribir ficheros
2. Cuando se resuelva, **se sugiere mejorar el programa** considerando que el contenido del fichero admite comas, puntos y comas, puntos, etc.

3. Una posible solución podría basarse en este **algoritmo**:

1. Crear un Hashtable para almacenar como claves las palabras del fichero origen y como valores el número de veces que se repiten.
2. Crear flujo y filtro para leer el fichero origen. Cada línea del mismo es una String, por tanto, puede crearse una StringTokenizer usando el delimitador por defecto. Luego, recorrer sus tokens e intentar obtener en el Hashtable la frecuencia de repetición de cada palabra teniendo en cuenta que si no hay ningún valor asociado a la palabra analizada devuelve null (esto implicará que la palabra es la primera vez que aparece). Si es distinto de null, la palabra ya ha aparecido antes. ¿Qué se hace? Se elimina el elemento del Hashtable cuya clave coincide con la palabra y se agrega un nuevo elemento teniendo en cuenta que el valor será un Integer, que almacenará un entero una unidad mayor que el del elemento eliminado.
3. Crear un Vector con las claves del Hashtable y ordenarlas mediante el método estático void sort(..) de la clase java.util.Collections.
4. Crear flujo y filtro para escribir en el fichero destino los pares del Hashtable por orden alfabético, en base a las claves. No olvidar vaciar y cerrar el filtro con flush() y close() respectivamente.

Datos a mostrar por consola

Para probar el código crear un fichero de nombre prueba.txt cuyo contenido sea este:

Hola que tal estas

Hola que tal con okal

Creo que estas bien

El contenido del fichero contador.txt será:

Creo: 1

Hola: 2

bien: 1

con: 1

estas: 2

okal: 1

que: 3

tal: 2

Recursos

Enlaces de Interés



<http://www.javahispano.org/portada/2011/8/1/tutorial-de-collections.html>

<http://www.javahispano.org/portada/2011/8/1/tutorial-de-collections.html>
Clases Collection. Estructuras de almacenamiento dinámico



<http://docs.oracle.com/javase/tutorial/collections/index.html>

<http://docs.oracle.com/javase/tutorial/collections/index.html>
Clases Collection. Estructuras de almacenamiento dinámico