Los arrays en Java son suficientes para guardar tipos básicos de datos, y objetos de una determinada clase cuyo número conocemos de antemano. Algunas veces deseamos guardar objetos en un array pero no sabemos cuantos objetos vamos a guardar. Una solución es la de crear un array cuya dimensión sea más grande que el número de elementos que necesitamos guardar. La clase *Vector* nos proporciona una solución alternativa a este problema. Un vector es similar a un array, la diferencia estriba en que un vector crece automáticamente cuando alcanza la dimensión inicial máxima. Además, proporciona métodos adicionales para añadir, eliminar elementos, e insertar elementos entre otros dos existentes.

Crear un vector

Para usar la clase *Vector* hemos de poner al principo del archivo del código fuente la siguiente sentencia import

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

Cuando creamos un *vector* u objeto de la clase *Vector*, podemos especificar su dimensión inicial, y cuanto crecerá si rebasamos dicha dimensión.

```
Vector vector=new Vector(20, 5);
```

Tenemos un vector con una dimensión inicial de 20 elementos. Si rebasamos dicha dimensión y guardamos 21 elementos la dimensión del vector crece a 25.

Al segundo constructor, solamente se le pasa la dimensión inicial.

```
Vector vector=new Vector(20);
```

Si se rebasa la dimensión inicial guardando 21 elementos, la dimensión del vector se duplica. El programador ha de tener cuidado con este constructor, ya que si se pretende guardar un número grande de elementos se tiene que especificar el incremento de la capacidad del vector, si no se quiere desperdiciar inútilmente la memoria el ordenador.

Con el tercer constructor, se crea un vector cuya dimensión inicial es 10.

```
Vector vector=new Vector();
```

La dimensión del vector se duplica si se rebasa la dimensión inicial, por ejemplo, cuando se pretende guardar once elementos.

Añadir elementos al vector

Hay dos formas de añadir elementos a un vector. Podemos añadir un elemento a continuación del último elemento del vector, mediante la función miembro *addElement*.

```
v.addElement("uno");
```

Podemos también insertar un elemento en una determinada posición, mediante *insertElementAt*. El segundo parámetro o índice, indica el lugar que ocupará el nuevo objeto. Si tratamos de insertar un elemento en una posición que no existe todavía obtenemos una excepción del tipo *ArrayIndexOutOfBounds*. Por ejemplo, si tratamos de insertar un elemento en la posición 9 cuando el vector solamente tiene cinco elementos.

Para insertar el string "tres" en la tercera posición del vector *v*, escribimos

```
        v.insertElementAt("tres", 2);
```

En la siguiente porción de código, se crea un vector con una capacidad inicial de 10 elementos, valor por defecto, y se le añaden o insertan objetos de la clase *String*.

```
        Vector v=new Vector();
        v.addElement("uno");
        v.addElement("dos");
        v.addElement("cuatro");
        v.addElement("cinco");
        v.addElement("seis");
        v.addElement("siete");
        v.addElement("ocho");
        v.addElement("nueve");
        v.addElement("diez");
        v.addElement("once");
        v.addElement("doce");
        v.insertElementAt("tres", 2);
```

Para saber cuantos elementos guarda un vector, se llama a la función miembro *size*. Para saber la dimensión actual de un vector se llama a la función miembro *capacity*. Por ejemplo, en la porción de código hemos guardado 12 elementos en el vector *v*. La dimensión de *v* es 20, ya que se ha superado la dimensión inicial de 10 establecida en la llamada al tercer constructor cuando se ha creado el vector *v*.

```
        System.out.println("nº de elementos "+v.size());
        System.out.println("dimensión "+v.capacity());
```

Podemos eliminar todos los elementos de un vector, llamando a la función miembro *removeAllElements*. O bien, podemos eliminar un elemento concreto, por ejemplo el que guarda el string "tres".

```
        v.removeElement("tres");
```

Podemos eliminar dicho elemento, si especificamos su índice.

```
        v.removeElementAt(2);
```

Acceso a los elementos de un vector

El acceso a los elementos de un vector no es tan sencillo como el acceso a los elementos de un array. En vez de dar un índice, usamos la función miembro *elementAt*. Por ejemplo, *v.elementAt(4)* sería equivalente a *v[4]*, si *v* fuese un array.

Para acceder a todos lo elementos del vector, escribimos un código semejante al empleado para acceder a todos los elementos de un array.

```
        for(int i=0; i<v.size(); i++){
            System.out.print(v.elementAt(i)+"\t");
        }
```

Existe otra alternativa, que es la de usar las funciones del interface *Enumeration*. Este interface declara dos funciones pero no implementa ninguna de ellas. Una *Enumeration* nos permite acceder a los elementos de una estructura de datos de forma secuencial.

```
public interface Enumeration {
    boolean hasMoreElements();
    Object nextElement();
}
```

La función miembro *elements* de la clase *Vector* devuelve un objeto de la clase *VectorEnumerator* que implementa el interface *Enumeration* y tiene que definir las dos funciones *hasMoreElements* y *nextElement*.

```
final class VectorEnumerator implements Enumeration {
    Vector vector;
    int count;
    VectorEnumerator(Vector v) {
        vector = v;
        count = 0;
    }
    public boolean hasMoreElements() {
        //...
    }
    public Object nextElement() {
        //...
    }
}
```

El objeto *enum* devuelto por la función miembro *elements* es de la clase *VectorEnumerator*, sin embargo no podemos escribir

```
VectorEnumerator enum=v.elements();
```

porque *VectorEnumerator* no es una clase pública. Como podemos ver en su definición, no tiene la palabra reservada **public** delante de **class**. Sin embargo, podemos guardar un objeto de la clase *VectorEnumerator* en una variable *enum* del tipo *Enumeration*, por que la clase implementa dicho interface.

```
Enumeration enum=v.elements();
while(enum.hasMoreElements()){
    System.out.print(enum.nextElement()+"\t");
}
```

Desde el objeto *enum* devuelto por la función miembro *elements* de la clase *Vector* llamamos a las funciones miembro *hasMoreElements* y *nextElement* de la clase *VectorEnumerator*. La función *hasMoreElements* devuelve **true** mientras haya todavía más elementos que se puedan acceder en el vector *v*. Cuando se ha llegado al último elemento del vector, devuelve **false**. La función *nextElement* devuelve una referencia al próximo elemento en la estructura de datos. Esta función devuelve una referencia a un objeto de la clase base *Object*, que el programador precisará en ciertos casos, como veremos más abajo, promocionar (casting) a la clase adecuada.

Para buscar objetos en un vector se puede usar una *Enumeration* y hecer una comparación elemento por elemento mediante equals, tal como vemos en la siguiente porción de código

```
Enumeration enum=v.elements();
while(enum.hasMoreElements()){
    String elemento=(String)enum.nextElement();
    if(elemento.equals("tres")){
        System.out.println("Encontrado tres");
        break;
    }
}
```

Podemos usar alternativamente, la función miembro *contains* para este propósito.

```
if(v.contains("tres")){
    System.out.println("Encontrado tres");
}
```

Algunos de los métodos de la clase Vector se muestran a continuación:

Vector ( ) Constructor: crea un vector inicialmente vacío
void addElement (Objet obj) Inserta el objeto especificado al final del vector
void setElementAt (Object obj, int indíce) Inserta el objeto específicado en el vector en la posición específicada
Object remove (int indíce) Elimina el objeto que se encuentra en la posición específicada y lo regresa
boolean removeElement (Object obj) Elimina la primera occurencia del objeto específicado en el vector
void removeElementAt (int indíce) Elimina el objeto específicado en el índice del vector
void clear ( ) Elimina todos los objetos del vector
boolean contains (Object obj) Regresa verdadero si el objeto dado pertenece al vector
int indexOf (Object obj) Regresa el índice del objeto específicado. Regresa -1 si no fue encontrado el objeto
Object elementAt (int indíce) Regresa el componente en el índice específicado
boolean isEmpty ( ) Regresa verdadero si el vector no contiene elementos
int size ( ) Regresa el número de elementos en el vector