**Conocimiento: Iteración sobre colecciones de objetos.**

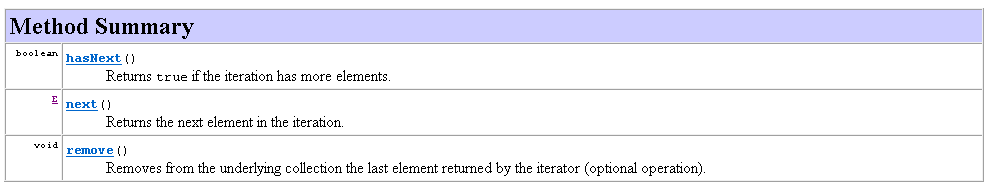
**Acceso a Collection a través de in iterador**

Normalmente se desea recorrer una colección con un ciclo de iteración. Para este propósito existe la interfaz **Iterator o ListIterator**,

**Interfaz Iterator**

Iterator permite ejecutar un ciclo a lo largo de una colección para obtener o eliminar elementos.

Métodos de Iterator:

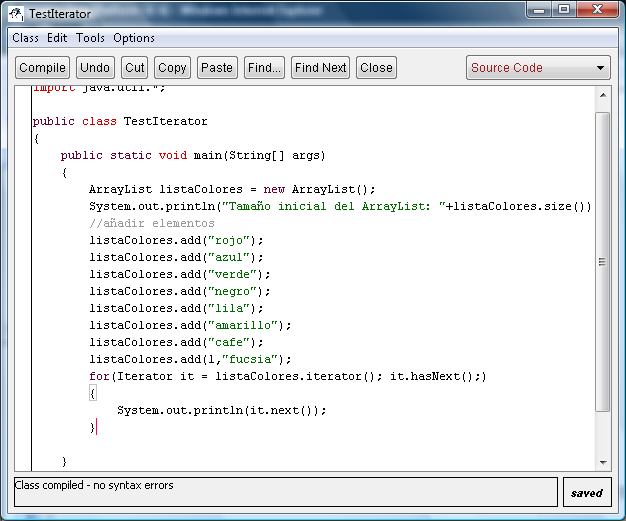
* El método **hasNext ()** verifica si existe un siguiente elemento en la colección, devolviendo true si así es o false en caso contrario.
* El método **next ()** permite obtener el siguiente elemento disponible en la colección.
* El método **remove ()** permite eliminar el objeto actual desde la colección.

Antes de poder acceder a una colección por medio de un iterador, se debe obtener uno. En java, cada clase de colecciones proporciona un método **iterator ()** que devuelve un iterador al inicio de la colección. Utilizando este objeto iterador se puede acceder a cada elemento, uno por uno.

Pasos a seguir para usar un iterador:

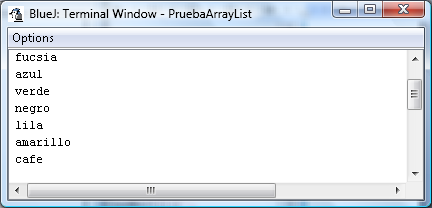
1. Se obtiene un iterador al principio de la colección llamando al método **iterator()** de dicha colección.
2. Se ejecuta un ciclo que llama a hasNext(). Se ejecuta este ciclo mientras **hasNext()** devuelva true.
3. Dentro del ciclo se obtiene a cada elemento llamando a **next().**

Ejemplo de uso de iterator() al recorrer un ArrayList para imprimirlo:



Se utiliza un ciclo for para recorrer la colección a través del objeto iterador, definido por la interfaz Iterator.

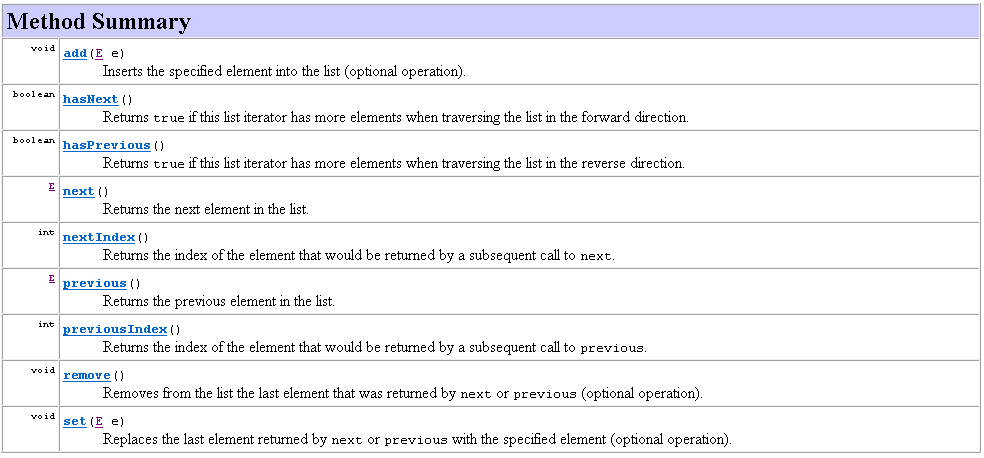
El resultado de ejecutar este código es:



**Interfaz ListIterator**

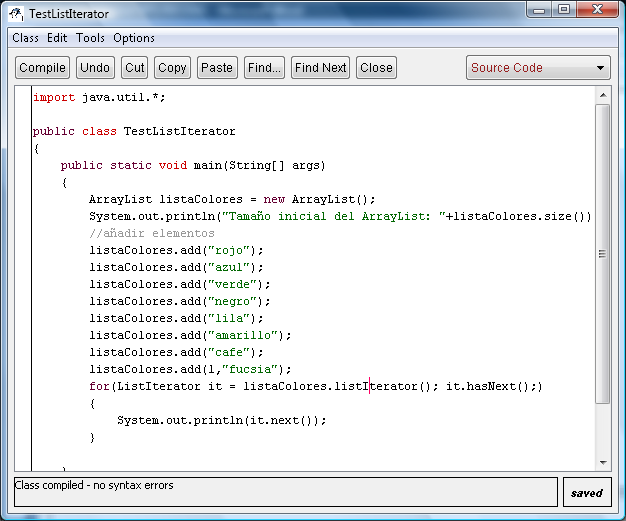
Para colecciones que implementen la interfaz List también se puede obtener un iterador llamando a la interfaz ListIterator, permitiendo acceder a la colección en forma bidireccional, hacia adelante o hacia atrás, así como modificar un elemento.

Métodos de ListIterator



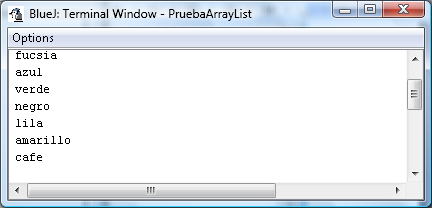
* El método **add(E)** agrega un elemento en la colección, frente al actual
* El método **hasNext ()** verifica si existe un siguiente elemento en la colección, devolviendo true si así es o false en caso contrario.
* El método **hasPrevious ()** verifica si existe un siguiente elemento en la colección, devolviendo true si así es o false en caso contrario.
* El método **next ()** permite obtener el siguiente elemento disponible en la colección
* El método **nextIndex()** permite obtener el índice del siguiente elemento disponible
* El método **previous()** permite obtener el elemento anterior al actual
* El método **previousIndex()** permite obtener el índice del elemento anterior disponible
* El método **remove ()** permite eliminar el objeto actual desde la colección.
* El métodos **set(E)** permite cambiar el elemento actual por el del parámetro

Ejemplo de uso de ListIterator para recorrer un ArrayList:



Se utiliza un ciclo for para recorrer la colección a través del objeto iterador, definido por la interfaz ListIterator.

El resultado de la ejecución de este código es:

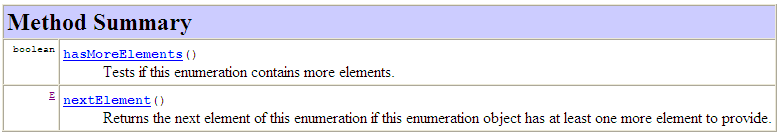


**Interfaz Enumeration**

La interfaz Enumeration permite iterar a través de los elementos de un contenedor como es el caso de un objeto Vector.

Un objeto que implementa la interfaz de Enumeración genera una serie de elementos, uno a la vez. Las sucesivas llamadas al método nextElement retorno sucesivos elementos de la serie

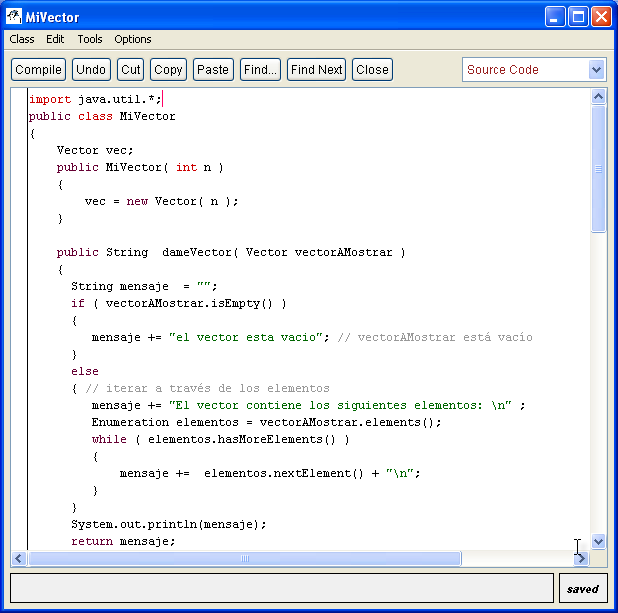
Los métodos de esta interfaz son los siguientes:



* El método **hasMoreElements()** verifica si en la enumeración hay más elementos.
* El método **nextElement()** permite obtener el siguiente elemento de la enumeración.

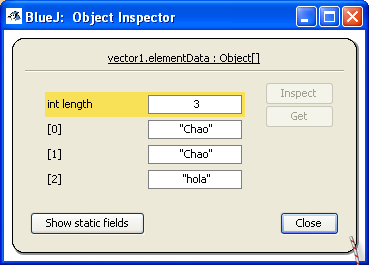
La funcionalidad de esta interfaz es replicada por la interfaz Iterator.

Ejemplo de uso de Enumeration con Vector:



En este ejemplo para poder recorrer el Vector se utiliza Enumeration.

Dado el objeto de la clase Vector con los siguientes elementos:



El resultado de ejecutar este código con el vector anterior es:

