élogos

GESTIÓN DE COLECCIONES

élogos

ÍNDICE

GESTIÓN DE COLECCIONES

1. Colecciones de tipo lista y tabla

Tipos de colecciones

Una colección es una agrupación dinámica de objetos. Es dinámica porque, a diferencia de un array, el tamaño de la colección es variable y se pueden añadir y eliminar objetos en tiempo de ejecución.

Existen tres tipos de colecciones:

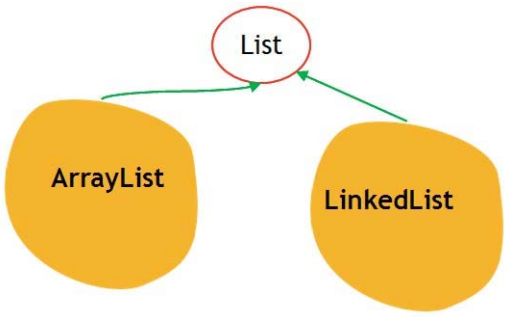
- **Listas**. Son similares a los arrays. Cada elemento tiene asociada una posición según el orden en el que se ha añadido. La interfaz java.util.List define los principales métodos para manejar listas. Entre las clases de tipo lista más importates tenemos ArrayList, LinkedList y Vector

- **Tablas**. En una tabla los objetos tienen asociada una clave que lo identifica. La interfaz java.util.Map define los principales métodos para manejar tablas, siendo las principales clases que la implementan Hashtable y HashMap

- **Conjuntos**. Un conjunto es una agrupación de objetos sin orden y sin clave asociada. Los métodos para el tratamiento de este tipo de colecciones se encuentran en java.util.Set, siendo HashSet la principal clase de este tipo

Interface List

La interfaz List proporciona los métodos para manipular una lista. Entre sus principales implementaciones están:



Métodos de la interfaz List

Object get (int index):

Recupera el elemento que se encuentra en la posición indicada, manteniéndolo dentro de la lista

boolean add (Object element):

Recupera el elemento que se encuentra en la posición indicada, manteniéndolo dentro de la lista

void add (int index, Object element):

añade un elemento en la posición indicada por index desplazando hacia la derecha el resto de los elementos existentes en el contenedor.

Object set (int index, Object element):

inserta en la posición indicada por index sobreescribiendo el elemento que hubiese en dicha posición. Devuelve el elemento que había antes en dicha posición.

Object remove(int index):

Elimina el elemento que ocupa la posición indicada, desplazando hacia la izquierda los que se encuentran a continuación de éste.

int indexOf (Object element) o int lastIndexOf (Object element):

ambos métodos realizan la búsqueda de un elemento dentro de la lista devolviendo en qué posición se encuentra.

La interfaz ListIterator

Permite recorrer la lista utilizando un interface Iterator especial que permite el recorrido hacia adelante y hacia atrás:

```
public interface ListIterator extends Iterator {
        boolean hasNext();
        Object next();
        Boolean hasPrevious();
        Object previous();
}
```

La clase ArrayList

La clase ArrayList es un ejemplo de implementación de interfaz List. Para crear una colección de este tipo que almacene, por ejemplo, cadenas de caracteres, utilizaremos la expresión:

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

A través del método add() comentado anteriormente añadimos elementos:

```
lista.add("Juan");

lista.add("Pedro");

lista.add("Ana");
```

Después, podrá recorrerse con un for estándar:

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

o con un for each:

```
for(String s:lista){
   System.out.println(s);
}
```

La interfaz Map

Los mapas, también conocidos como tablas, se gestionan a través de la interfaz Map.

Entre las principales implementaciones de Map tenemos:

- **Hashtable**. Colección que almacena objetos con una clave asociada que lo identifica. Los métodos para la manipulación de la colección son sincronizados, lo que hace que sea un tipo de colección adecuada en aplicaciones multitarea
- **HashMap**. Es muy similar a Hashtable, si bien sus métodos no son sincronizados, lo que hace que ofrezca un mejor rendimiento en aplicaciones monotarea

Métodos:

Entre los principales métodos de la interfaz Map tenemos los siguientes:

- Object **put** (Object key, Object value). Añade a la colección el objeto especificado en el segundo parámetro, asociándole la clave indicada como primer parámetro.
- Object **get**(Object key). Recupera el objeto que tiene la clave indicada. Si no existiera ningún objeto con esa clave, devolverá null
- boolean **containsKey** (Object key) . Indica si la clave está siendo utilizada por algún objeto
- Object **remove** (Object key). Elimina el objeto que tiene asociada dicha clave
- int **size**(). Indica el número de elementos de la colección
- Collection **values**(). Devuelve un Collection con los valores (objetos) existentes en la colección. El Collection se puede recorrer con un for each

La clase HashMap

Como hemos indicado anteriormente, HashMap es una de las principales implementaciones de de Map., donde los objetos almacenados en la colección se encuentran mapeados a una clave. La colección utiliza el valor devuelto por el método hashCode() para distinguir unas claves de otras, este valor, conocido como código hash, identifica de forma única a un objeto cargado en memoria.

Para crear un objeto HashMap, debemos indicar el tipo de clave y de objeto que se van a tratar:

```
HashMap<Integer, String> tabla = new HashMap<Integer, String>();
```

Mediante el método put() añadimos los elementos:

```
tabla.put(2345, “Juan”);

tabla.put(3411, “Javier”);
```

Dado que estas colecciones no tienen índices, no se pueden recorrer con un for estándar, pero podemos apoyarnos en Collection para recorrer los valores:

```
Collection<String> nombres= tabla.values();

for(String s:nombres){

    System.out.println(s);

}
```

La clase Hashtable

Hashtable es un subclase de Dictionary. La creación de un objeto Hashtable sería similar a la de un HashMap:

Hashtable <Integer, String> tabla = new Hashtable <Integer, String>();

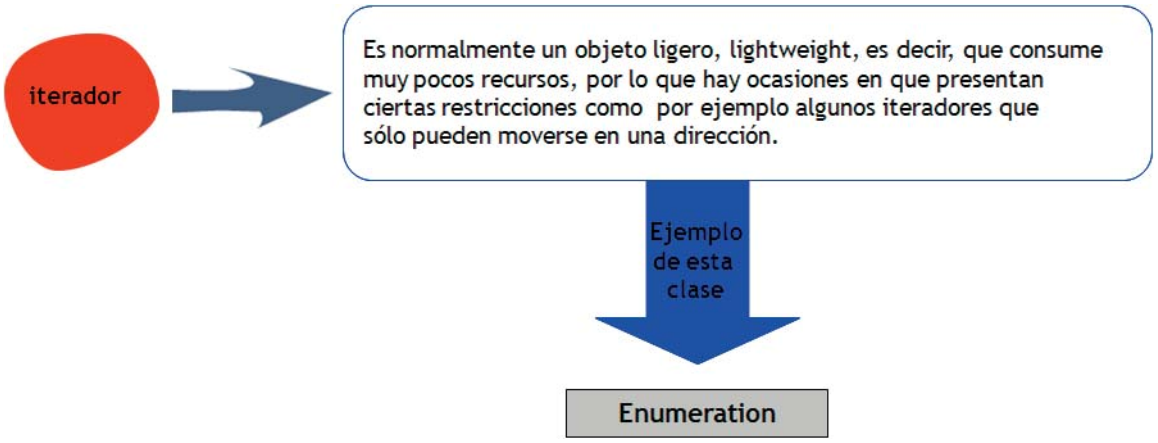
Además de los métodos definidos en Map, Hashtable incorpora otros métodos propios como

- Enumeration **keys**().Devuelve un objeto Enumeration para recorrer las claves
- Enumeration **elements**().Devuelve un objeto Enumeration para recorrer los elementos

2. Enumeraciones e iteraciones

Enumerador o iterador

El concepto de enumerador, o iterador, puede usarse para alcanzar el nivel de abstracción que se necesita en este caso. Es un objeto cuyo propósito consiste en desplazarse a través de un conjunto de objetos y seleccionar aquellos objetos adecuados sin que el programador que los usa tenga que conocer la estructura de la secuencia.



Métodos de enumerador o iterador

Las interfaces Iterator y Enumeration de java.util permiten trabajar con este tipo de objetos. Veamos, por ejemplo, los métodos de Enumeration:

- **nextElement**(). Nos devolverá el primer elemento en la lista, cuando lo aplicamos por primera vez. Para obtener siguiente elemento en la secuencia también o haremos a través del método nextElement().
- **hasMoreElements**(). Indica si hay más elementos en la secuencia

Ejemplo:

```
Hashtable <String, String> tabla;
Tabla=new Hashtable <String, String> ();
//rellenado de la tabla
:
//el método elements de Hashtable nos
//devuelve un Enumeration para recorrer
//los elementos
Enumeration<String> en=tabla.elements();
while(en.hasMoreElements(){
    System.out.println(en.nextElement());
}
```

3. Colecciones de tipos genéricos

Los tipos genéricos

Utilizando los genéricos se evitan las excepciones en tiempo de ejecución debido a la inserción de objetos de distinto tipo en las colecciones.

Los tipos genéricos solamente son evaluados en tiempo de compilación, en ejecución no existen aunque nos aseguramos que si el código compila correctamente, no nos encontraremos con problemas de casting entre objetos en la ejecución del código.

Ejemplo

Si por ejemplo declaramos la siguiente lista: List<Integer> miLista = new ArrayList<Integer>() el compilador no dejará que se inserte un objeto que no sea de tipo Integer.

Los tipos genéricos 2

Nuestro compilador sí aceptará clases derivadas, por ejemplo, si se declara una List<Map> se podrá insertar un tipo HashMap, TreeMap, etc. Es decir, admite la sobrecarga de métodos (polimorfismo).

Esto sería incorrecto: ArrayList<Map> list = new ArrayList<HashMap>().

Justo en el momento que hacemos la declaración (tipo Map) e instanciación (tipo HashMap) del objeto colección no se admite el polimorfismo.

Clase Padre de todos los genéricos

Es <?>, indica que cualquier genérico puede ser insertado en la colección. También se puede indicar el polimorfismo en los genéricos mediante <? extends Map>.

Donde se pueden utilizar los genéricos

Todo lo que extienda de Map se puede insertar en la colección:

Los genéricos también se pueden utilizar en la declaración de clases:

class MiColecion <T> { ...}), atributos de la clase (T instancia), constructores (MiColeccion (T ref)), métodos y tipos de retorno (public T bar (T ref) {}.

El compilador sustituirá el tipo por el que se cree cuando se instancie un objeto de la clase MiColeccion