Conjuntos en Java

Esta obra, cuyo autor es Marcos Martínez Rus, ha sido publicada bajo una
licencia de Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional

ÍNDICE

BY NC SA

1. Introducción

Los conjuntos en Java nos van a permitir crear colecciones dinámicas de datos, todos del mismo tipo, que no tienen un orden y no permiten duplicados. La ventaja es que no hace falta definir, en tiempo de compilación, el número de elementos que va a tener el conjunto (similar a lo que sucede con las listas).

Una peculiaridad de Java es que tenemos que trabajar con las clases *wrapper* para tipos primitivos ya que lo que almacenan son objetos. Por ello, para trabajar con tipos de datos primitivos debemos hacer uso de las clases `Integer`, `Float`, `Boolean`, etcétera.

Nosotros, para el trabajo con conjuntos nos vamos a centrar en el uso de `HashSet`. A continuación se indican una serie de métodos pero hay muchos más… ¡investiga!.

2. Declaración e inicialización

El trabajo con conjuntos en Java no se limita a la interfaz `HashSet`. No obstante, es la más utilizada y, por ello, nos centraremos en ella.

Para declarar conjuntos en Java podemos utilizar la siguiente sintaxis:

```
Set<tipoDeDato> nombreLista = new HashSet<tipoDeDato>();
```

Para poder trabajar con conjuntos, en Java, es necesario importar el paquete `java.util` que es donde se encuentra la interfaz `Set` (entre otras).

Por ejemplo, si queremos almacenar los nombres y apellidos de una familia, podríamos implementar el siguiente código:

```
Set<String> miembrosFamilia = new HashSet<String>();
```

3. Métodos heredados de Collection

Sabemos que las interfaces `List<E>` y `Set<E>` heredan de la interfaz `Collection<E>`. De esta manera, tenemos una serie de métodos comunes que son los siguientes y que podemos utilizar en ambos tipos de colecciones. Supondremos que el elemento `e` es un objeto de la clase `E`.

- `boolean` **`add`**`(E e)`. Añade un nuevo elemento al final de la lista.
- `boolean` **`remove`**`(E e)`. Elimina la primera ocurrencia del elemento indicado.
- `boolean` **`contains`**`(E e)`. Comprueba si el elemento especificado está en la colección.
- `void` **`clear`**`()`. Elimina todos los elementos de la colección.
- `int` **`size`**`()`. Devuelve el número de elementos en la colección.
- `boolean` **`isEmpty`**`(Collection<?> c)`. Comprueba si la colección está vacía.

- `boolean` **`addAll`**`(Collection<?> c)` . Añade todos los elementos de la colección c.
- `boolean` **`removeAll`**`(Collection<?> c)`. Elimina todos los elementos de la colección c.
- `boolean` **`containsAll`**`(Collection<?> c)`. Comprueba si coinciden las colecciones.
- `boolean` **`retainAll`**`(Collection<?> c)`. Elimina todos los elementos a no ser que estén en c. Es lo que se conoce como intersección de conjuntos.

4. Tipos de conjuntos

Podemos utilizar diferentes implementaciones para el uso de conjuntos. Como ya hemos visto, la más eficiente es `HashSet<E>` que utiliza una implementación con *tabla hash* (por ahora, se nos escapa ese concepto). Pero si queremos que los elementos queden ordenados podemos usar `TreeSet<E>` (implementación con árbol).

CC BY NC SA