




## 5. Otras colecciones

 Date	Empty
 Status	Not started
 Type	Empty
Units	4. Estructuras de almacenamiento

Además de las colecciones vistas en el punto anterior, existen numerosos tipos más, donde podemos destacar dos: HashSet y Stak

### HashSet

Una colección HashSet permite crear listas de datos con índices de acceso. Del mismo modo que un ArrayList se trata de una colección no sincronizada. A diferencia de las colecciones, un HashSet no puede contener objetos duplicados, ya que cada uno tiene un hashCode asignado

El constructor para poder crear una lista es:

```
HashSet<Integer> lista = new HashSet<String>(); HashSet<Persona> lista = new HashSet<Persona>();
```

Donde se indica el tipo de dato que guardará la lista.

Los métodos utilizados son:

```
// Añadir el elemento "a" en la lista (si ya existe, no se añ
ade): lista.add(new Integer(a)); // Vaciar el conjunto "c": l
ista.clear(); // Comprobar si en la lista existe el valor ent
ero "a": lista.contains(new Integer(a)) → boolean // Conocer
si el conjunto "c" está vacía: lista.isEmpty()→ boolean // Ob
tener el número de elementos que tiene la lista: lista.size()
→ int // Eliminar el valor entero "n" en la lista: lista.remo
ve(new Integer(n));
```

En el caso de añadir objetos creados, es recomendable la sobreescritura del método hashCode()

```
public class Persona { int clave; String nombre, apellido; in
t telefono; public Persona(String nombre, String apellido, in
t telefono) { this.nombre = nombre; this.apellido = apellido;
this.telefono = telefono; } public String getNombre() { retur
n nombre; } public String getApellido() { return apellido; }
public int getTelefono() { return telefono; } @Override publi
c int hashCode() { int codigo; codigo = getNombre().hashCode
(); codigo += getTelefono(); return codigo; } }
```

Para poder recorrer los elementos de un HashMap se utiliza un objeto Iterator sobre la lista

```
Iterator<String> elementos = lista.iterator(); for (Iterator
<String> it = elementos; it.hasNext(); ) { String s = it.next
(); System.out.println(s); }
```

## Stack

Un Stack permite apilar objetos.

```
Stack<Integer> a = new Stack<Integer>();
```

Los métodos utilizados son:

```
// Vaciar la pila: a.clear(); // Comprobar si la pila está vacía: a.isEmpty() // Elimina el elemento superior de la pila: a.pop(); // Consultar el último apilado: a.peek(); // Apilar un entero: a.push(8); // obtener todos los elementos de una pila a.elements() // obtener el elemento de la posición 2 a.elementAt(2) // clonar la pila a.clone() // evaluar si el elemento 2 está contenido en la pila a.contains(2) // añadir el elemento 2 a.add(5) // obtener el primer elemento a.firstElement() // obtener el último elemento a.lastElement()
```

Para poder recorrer los elementos de una pila

```
Iterator<Integer> iterator = numeros.iterator(); for (Integer n: numeros) { System.out.println(n); }
```

O bien

```
Enumeration<Integer> enumeration = numeros.elements(); while (enumeration.hasMoreElements()) { System.out.println(enumeration.nextElement()); }
```