Tema 10: Estructuras de datos avanzados

¿Qué aprenderás?

- Usar las estructuras ArrayList y HashMap.
- Usar los métodos típicos para trabajar con estas estructuras.
- Modificar los métodos hashCode y equals del HashMap.

¿Sabías que…?

- Tanto los ArrayList como los HashMap se incluyeron en la especificación 1.2 de Java.
- Estas dos estructuras no pueden trabajar con tipos de datos simples. Debemos usar sus Wrappers, en un proceso que se le conoce como Boxing.
- Normalmente los métodos hashCode y equals se sobreescriben de manera que compartan los criterios para considerar cuándo dos objetos son iguales.

1. Introducción

Hasta ahora nuestros programas constaban de variables cuyos tipos de datos nos permitían guardar un único valor alfanumérico. Disponemos también de los arrays y las cadenas de caracteres (String) para poder almacenar varios valores de un mismo tipo. Incluso los objetos nos ofrecen una forma de almacenamiento de información, mediante los atributos.

Estos no son los únicos mecanismos que nos brinda Java para poder almacenar información. Hay otras estructuras avanzadas, como pueden ser las listas, colas, pilas, árboles, etc., cada una de ellas con sus métodos de acceso a los datos. En este capítulo nos centraremos en el funcionamiento de las listas (ArrayList) y colecciones (HashMap).

2. La clase ArrayList

Un ArrayList lo podemos comparar con un array estático de los que se han visto en temas anteriores, por el hecho de permitirnos guardar varios valores de un mismo tipo bajo un único nombre de variable. Pero hay una diferencia muy importante entre los array y los ArrayList: éste último es dinámico. Esto quiere decir que su tamaño puede cambiar a lo largo de la ejecución del programa.

Si recordamos los arrays, en su declaración debíamos especificar el tamaño que tendrá, y este no cambiará en todo el programa. Esto puede suponer un gasto innecesario de memoria, o puede ocurrir que nos quedemos cortos al querer añadir un nuevo elemento al array. Con los arrays dinámicos, los ArraysList, este problema desaparece, ya que su tamaño se adaptará a su contenido. Durante la ejecución del programa podemos añadir y eliminar tantos elementos como queramos.

Para usar ArrayList en nuestro programa, tendremos que importar la librería java.util.ArrayList.

2.1. Declaración y método add

La declaración del ArrayList es la siguiente:

```
ArrayList miLista = new ArrayList();
```

Declaración de un ArrayList vacío

La instrucción anterior crea un ArrayList vacío. Creándolo de esta forma, el ArrayList puede contener objetos de cualquier tipo. Para añadir elementos, usaremos el método add():

```
miLista.add("Hola");
miLista.add(31);
miLista.add('d');
miLista.add(5.5);
```

Adición de elementos en un ArrayList.

El método add añade al final del ArrayList un nuevo elemento. En este ejemplo, el primer objeto que se añade al ArrayList es el String "Hola". El resto no son objetos. Son datos de tipos básicos pero el compilador los convierte automáticamente en objetos de su clase contenedora antes de añadirlos al ArrayList.

Un ArratList al que se le pueden asignar elementos de distinto tipo puede tener alguna complicación a la hora de trabajar con él. Por eso, una alternativa a esta declaración es indicar el tipo de objetos que contiene. En este caso, el ArrayList sólo podrá contener objetos de ese tipo.

```
ArrayList<tipo> miLista = new ArrayList<tipo>();
```

Declaración de un ArrayList especificando el tipo de datos que contendrá.

En la instrucción anterior, "tipo" debe ser una clase. Indica el tipo de objetos que contendrá el array.

No se pueden usar tipos primitivos. Para un tipo primitivo se debe utilizar su clase contenedora. Por ejemplo, si queremos que el ArrayList contenga enteros:

```
ArrayList<Integer> miLista = new ArrayList<Integer>();
```

Declaración de un ArrayList que guardará números enteros.

Hemos visto que podemos añadir elementos a un ArrayList con el método add. Los elementos nuevos siempre se añadirán al final del ArrayList. Si queremos añadir un elemento en un lugar concreto, podemos usar la misma función add con otros parámetros:

```
miLista.add(0, 1000);
```

Adición de un valor en una posición concreta de un ArrayList.

La instrucción anterior añadirá el entero 1000 en la primera posición del ArrayList (posición 0). Hay que decir que la posición que especifiquemos debe estar comprendida entre 0 y ArrayList.size()-1, sino se producirá una excepción.

2.2. Métodos de la clase ArrayList

En el apartado anterior ya hemos visto cómo crear un ArrayList y cómo añadir elementos en él, mediante el método add. Vamos a ver otros métodos igualmente útiles para trabajar con nuestros ArrayList.

Para saber el tamaño del ArrayList, es decir, cuántos elementos tiene actualmente, usaremos el método size:

```
miLista.size();
```

Consulta del tamaño de un ArrayList.

Si queremos acceder al elemento que ocupa la posición x en el ArrayList, usaremos el método get. Éste nos devuelve un objeto del mismo tipo que el que hay en la posición accedida del ArrayList.

```
ArrayList<String> listaPalabras = new ArrayList<String>();
…
String palabra = listaPalabras.get(6);
```

Declaración de un ArrayList y acceso a la posición 6.

La instrucción anterior devuelve el String que hay almacenado en la posición 6 del ArrayList, y lo asigna a una variable “palabra”. Hay que tener en cuenta que la primera posición de un ArrayList es la 0. Por tanto, en el método get las posiciones válidas van de 0 a ArrayList.size()-1. Cualquier posición fuera de este rango lanzará una excepción.

A modo de resumen, en la siguiente tabla podemos ver los principales métodos de la clase ArrayList, algunos de ellos ya comentados:

size()	Devuelve un entero conel número de elementos del ArrayList.
add(X)	Añade el objeto X al final del ArrayList.
add(posición, X)	Inserta el objeto X en la posición indicada del ArrayList.
get(posición)	Devuelve el elemento que está en la posición indicada.
remove(posición)	Elimina el elemento que se encuentra en la posición indicada. Devuelve el elemento eliminado.
remove(X)	Elimina la primera ocurrencia del objeto X. Devuelve true si el elemento está en la lista.
clear()	Elimina todos los elementos del ArrayList.
set(posición, X)	Sustituye el elemento que se encuentra en la posición indicada por el objeto X. Devuelve el elemento sustituido.
contains(X)	Comprueba si la colección contiene al objeto X. Devuelve true o false.
indexOf(X)	Devuelve la posición del objeto X. Si no existe devuelve -1.

3. La clase HashMap

Un HashMap es una implementación de la interfaz Map de Java. Es una colección de objetos. El HashMap nos permite almacenar pares clave/valor, de tal manera que una clave sólo puede tener un valor, es decir, las claves no se repiten. Si añadimos un elemento cuya clave ya existe, no se generará un nuevo elemento en el HashMap, sino que se reescribe el valor que pertenece a la clave existente. Hay que tener en cuenta que el HasMap no admite valores nulos.

La particularidad de los HashMap radica en cómo se ordenan sus elementos. Se utiliza un algoritmo que, a partir de la clave, genera otra clave “hash”, que será la que determine el orden en que se almacenarán en memoria los elementos.

Esta forma de ordenar los objetos hace que el tiempo de ejecución de operaciones como coger un elemento o añadir uno nuevo permanezca constante independientemente del tamaño del HashMap.

3.1. Declaración

Para usar HashMap en nuestro programa, tendremos que importar la librería java.util.HashMap. La declaración del HashMap es la siguiente:

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

Declaración de un HashMap en que las claves serán de tipo String y los valores de tipo Integer.

La instrucción anterior crea un HashMap cuyos elementos tendrán una clave de tipo String y un valor de tipo Integer. Como pasa con los ArrayList, en los HashMap no se pueden usar tipos primitivos (int), se debe utilizar su clase contenedora (Integer).

Podemos declarar un HashMap sin especificar el tipo de datos de las claves y los valores a almacenar. Esto nos permite añadir cualquier tipo de objeto, lo cual no es recomendable, pues para mostrar los datos del HashMap podemos tener problemas de conversión de tipos de datos.

3.2. Añadir elementos a un HashMap

Una vez se ha creado un HashMap, vamos a introducir elementos en él. Para ello usaremos el método put de la siguiente manera:

```
map.put("Pedro", 12345);
map.put("Luis", 45321);
map.put("Susana", 99999);
map.put("Maria", 34521);
map.put("Manuel", 123);
map.put("Manuel", 12333);
```

Adición de claves y valores en un HashMap cpn el método put.

Siguiendo con el HashMap creado de ejemplo, con un String como clave y un Integer como valor, las líneas de código anterior almacenarían 5 nuevos elementos en el HashMap. Observad que los dos últimos elementos comparten la misma clave “Manuel”. Recordemos que el HashMap no contiene claves repetidas, por tanto, al ejecutarse la última instrucción, no se crea un nuevo elemento en el HashMap, sino que se modifica el valor del elemento con la clave “Manuel”.

El método put recibe dos parámetros, el primer indica la clave y el segundo el valor del objeto que se va a insertar en el HashMap. Los tipos de ambos parámetros deben concordar con los empleados en la declaración del HashMap. Si no se produce un error de compilación.

3.3. Recorrer los elementos de un HashMap

Una operación típica que se hace con los HashMap es recorrerlos y mostrar los datos que almacena. El siguiente fragmento de código lo hace:

```
Iterator it = map.keySet().iterator();
while(it.hasNext()){
        String key = (String) it.next();
        System.out.println("Clave: " + key + " -> Valor: " + map.get(key));
}
```

Uso de la clase Iterator para recorrer los elementos de un HashMap.

Para recorrer el HashMap se ha declarador un iterador en la primera instrucción. La interfaz Iterator nos ofrece un mecanismo para acceder secuencialmente a los objetos de una colección. Su uso extendido radica en la gran variedad de tipologías de colecciones, además de que los elementos almacenados en ellas no son homogéneos.

Volviendo al código de ejemplo, el iterador lo creamos sobre una colección que contiene las claves de los elementos del HashMap. La función keySet() nos devuelve el conjunto de claves que hay en el HashMap.

Para movernos por la colección de claves usando el iterador creado, se usa un bucle while con la condición de entrada “it.hasNext()”. El método hasNext() de la interfaz Iterator devuelve true si la iteración tiene más elementos.

Dentro del bucle, movemos el iterador al elemento siguiente con la instrucción it.next(). Como estamos recorriendo la colección de claves del HashMap, y éstas son de tipo String, almacenamos el contenido del iterador en una variable “key” de tipo String.

Para acceder al valor de cada elemento del HashMap usamos el método get(key). Como parámetro debemos especificar la clave que queremos buscar en el HashMap. El método nos devolverá el valor asociado a dicha clave, si existe, nulo en otro caso.

Si se ejecuta el código anterior, se puede observar que los datos se muestran sin un orden previsible, como podría ser alfabéticamente. Esto es debido al algoritmo usado por el HashMap para almacenar eficientemente sus elementos.

3.4. Métodos de la clase HashMap

En la siguiente tabla podemos ver los principales métodos de la clase HashMap:

clear()	Vacía el HashMap.
containsKey(key)	Indica si el HashMap contiene un elemento con la clave key.
containsValue(value)	Indica si el HashMap contiene un elemento con el valor value.
entrySet()	Devuelve una copia del HashMap en forma de colección. Cualquier cambio en la colección afecta directamente al HashMap.
get(key)	Devuelve el valor asociado a la clave key, o nulo si no existe la clave.
isEmpty()	Indica si el HashMap está vacío.
keySet()	Devuelve una colección con las claves del HashMap.
put(key, value)	Añade un elemento al HashMap con la clave key y el valor value. Si ya existe un elemento con clave key, sustituye su valor a value.
putAll(map)	Copia todos los elementos de map en el HashMap que invoca el método.
remove(key)	Elimina el elemento con clave key del HashMap si existe.
size()	Devuelve un número entero que es el número de elementos que tiene el HashMap
values()	Devuelve una colección con los valores que contiene el HashMap.

3.5. Métodos equals y hashCode

Estos dos métodos son los que debemos usar si queremos tener control sobre cuándo dos elementos se consideran iguales. Ambos métodos son proporcionados por la interfaz Map, los cuales tendremos que sobrescribir para programar el comportamiento que queremos que tengan.

El método equals nos devuelve un booleano que indica si dos elementos son iguales o no. Por defecto, se considera que dos elementos son iguales si tienen la misma clave. Podemos ir más allá y hacer que, por ejemplo, dos elementos se consideren iguales si tienen el mismo valor. Por ejemplo, consideramos dos personas iguales si tienen el mismo nombre y apellidos.

El método hashCode viene a hacer lo mismo que el método equals, es decir, compara dos elementos y nos dice si son iguales. Pero la comparación la hace más rápida, ya que utiliza un número entero. Dos objetos que sean iguales devuelven el mismo valor hash.

Cuando comparamos dos objetos en estructuras de tipo hash (HashMap, HashSet, etc.) primero se invoca al método hashCode y luego al equals. Si los métodos hashCode de cada objeto devuelve un entero diferente, se consideran los objetos distintos. En caso contrario, en que ambos objetos tengan el mismo código hash, Java ejecutará el método equals y revisará en detalle si se cumple la igualdad.

Veamos un ejemplo en el que creamos la clase Persona e implementamos los métodos hashCode y equals:

```
public class Persona {
        String nombre;
        String apellido;
        int edad;
        public Persona(String nombre, String apellido, int edad) {
                this.nombre = nombre;
                this.apellido = apellido;
                this.edad = edad;
        }
        //Getters y setters
        …
}
```

Clase Persona.

Vemos que tenemos los atributos nombre, apellido y edad. Vamos a considerar que dos personas son iguales si tienen el mismo valor en los atributos nombre y apellidos. Implementamos el método hashCode:

```
@Override
public int hashCode() {
        int result = this.nombre.hashCode() + this.apellido.hashCode();
        return result;
}
```

Ejemplo de método hashCode en la clase Persona.

Lo que hacemos es declarar un entero, cuyo valor vendrá de sumar el número hash del nombre y del apellido. Se puede ver que invocamos el método hashCode sobre los Strings nombre y apellidos. La clase String tiene el método hashCode implementado, como todas las clases de Java (lo heredan de la clase Object). Lo que nosotros podemos hacer es sobreescribirlo, como estamos haciendo en la clase Persona.

El método equals considerará los mismos criterios que hashCode, es decir, dos personas son iguales si tienen el mismo nombre y apellido.

```
@Override
public boolean equals(Object obj) {
	if (this == obj)
		return true;
	if (obj == null)
		return false;
	if (getClass() != obj.getClass())
		return false;
	Persona other = (Persona) obj;
	if (apellido == null) {
		if (other.apellido != null)
			return false;
	} else if (!apellido.equals(other.apellido))
		return false;
	if (nombre == null) {
		if (other.nombre != null)
			return false;
	} else if (!nombre.equals(other.nombre))
		return false;
	return true;
}
```

Ejemplo de método equals en la clase Persona.

Recursos y enlaces

- Clase ArrayList Java 10:
  https://docs.oracle.com/javase/10/docs/api/java/util/ArrayList.html

- Clase HashMap Java 10:
  https://docs.oracle.com/javase/10/docs/api/java/util/HashMap.html

Conceptos clave

- **ArrayList**: estructura que permite almacenar varios datos del mismo tipo. El tamaño de esta estructura es variable en función del número de elementos que contenga en cada momento.
- **HashMap**: colección de objetos que almacena pares clave/valor. Podemos acceder a los elementos del HashMap mediante la clave, ya que éstas tienen un valor único.
- **hashCode**: es un valor numérico entero que sirve para identificar un objeto. En las colecciones de tipo hash tiene especial importancia para identificar objetos iguales de forma rápida.
- **Método equals**: método que compara dos objetos y devuelve un booleano indicando si son iguales o no.

Test de autoevaluación

¿Qué hace la siguiente instrucción? ArrayList lista = new ArrayList()

a) Se crea un ArrayList de una posición
b) Se crea un ArrayList que puede almacenar cualquier tipo de objeto
c) Da un error de compilación, ya que no se ha especificado el tipo de dato que almacenará el ArrayList
d) Da un error de compilación, ya que no se ha especificado un tamaño inicial para el ArrayList

¿Cuál de las siguientes es una declaración errónea de un ArrayList?

a) ArrayList lista = new ArrayList()
b) ArrayList<String> lista = new ArrayList<String>()
c) ArrayList<Double> lista = new ArrayList<Double>()
d) ArrayList<int> lista = new ArrayList<int>()

¿Qué ocurre si intentamos añadir a un HashMap un elemento con una clave que ya existe?

a) Se modificará el valor del elemento con la clave que se intenta añadir
b) Se inserta a continuación del elemento con la misma clave
c) No se inserta nada
d) Se produce un error

Ponlo en práctica

Actividad 1

Crea una aplicación que use un ArrayList para almacenar números enteros y añade varios valores.

A continuación, muestra el número total de elementos del ArrayList, y recórrelo.

Luego, pide al usuario que introduzca un valor para buscarlo en el ArrayList. Si lo encuentra, el programa preguntará si se quiere cambiar el valor. Si el usuario contesta afirmativamente, se le pedirá el nuevo valor y se introducirá en el ArrayList modificando el anterior.

Al final, se debe volver a mostrar el número total de elementos del ArrayList, y cuáles son.

Actividad 2

Crea la clase Alumno con los atributos nombre, apellidos y edad. Deberás implementar el constructor, todas los getters y setters, el método toString y los métodos hashCode y equals. El código hash de cada alumno se calculará sumando el código hash del nombre, el código hash del apellido y la edad. Por otro lado, dos alumnos se consideran iguales si tienen el mismo valor en sus atributos.

Crea una aplicación que use un HashMap para almacenar los alumnos (clave) y el curso al que van (valor).

Añade varios alumnos al HashMap y luego muestra el número total de alumnos y sus valores (datos del alumno y curso al que van).

Crea un nuevo alumno cuyos datos coincidan con alguno de los ya dados de alta en el HashMap, pero en un curso diferente, e intenta insertarlo.

Comprueba que el HashMap sigue teniendo el mismo tamaño y muestra todos los datos que contiene.

SOLUCIONARIOS

Test de autoevaluación

¿Qué hace la siguiente instrucción? ArrayList lista = new ArrayList()

e) Se crea un ArrayList de una posición

f) Se crea un ArrayList que puede almacenar cualquier tipo de objeto

g) Da un error de compilación, ya que no se ha especificado el tipo de dato que almacenará el ArrayList

h) Da un error de compilación, ya que no se ha especificado un tamaño inicial para el ArrayList

¿Cuál de las siguientes es una declaración errónea de un ArrayList?

i) ArrayList lista = new ArrayList()

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

k) ArrayList<Double> lista = new ArrayList<Double>()

l) ArrayList<int> lista = new ArrayList<int>()

¿Qué ocurre si intentamos añadir a un HashMap un elemento con una clave que ya existe?

m) Se modificará el valor del elemento con la clave que se intenta añadir

n) Se inserta a continuación del elemento con la misma clave

o) No se inserta nada

p) Se produce un error

Ponlo en práctica

Actividad 1

Crea una aplicación que use un ArrayList para almacenar números enteros y añade varios valores.

A continuación, muestra el número total de elementos del ArrayList, y recórrelo.

Luego, pide al usuario que introduzca un valor para buscarlo en el ArrayList. Si lo encuentra, el programa preguntará si se quiere cambiar el valor. Si el usuario contesta afirmativamente, se le pedirá el nuevo valor y se introducirá en el ArrayList modificando el anterior.

Al final, se debe volver a mostrar el número total de elementos del ArrayList, y cuáles son.

El solucionario está disponible en la versión interactiva del aula.

Actividad 2

Crea la clase Alumno con los atributos nombre, apellidos y edad. Deberás implementar el constructor, todas los getters y setters, el método toString y los métodos hashCode y equals. El código hash de cada alumno se calculará sumando el código hash del nombre, el código hash del apellido y la edad. Por otro lado, dos alumnos se consideran iguales si tienen el mismo valor en sus atributos.

Crea una aplicación que use un HashMap para almacenar los alumnos (clave) y el curso al que van (valor).

Añade varios alumnos al HashMap y luego muestra el número total de alumnos y sus valores (datos del alumno y curso al que van).

Crea un nuevo alumno cuyos datos coincidan con alguno de los ya dados de alta en el HashMap, pero en un curso diferente, e intenta insertarlo.

Comprueba que el HashMap sigue teniendo el mismo tamaño y muestra todos los datos que contiene.

El solucionario está disponible en la versión interactiva del aula.