Capítulo 6

La API de Colecciones

Muchos algoritmos requieren el uso de una representación de datos apropiada para ser eficientes. Esta representación y las operaciones permitidas con ella se conocen con el nombre de *estructura de datos*. Cada estructura de datos permite la inserción arbitraria pero difiere en cuanto al modo en que permite acceder a los miembros del grupo. Algunas estructuras permiten accesos y borrados arbitrarios, mientras que otras imponen restricciones, como por ejemplo que se permita acceder solo al elemento del grupo más recientemente insertado o menos recientemente insertado.

Como parte de Java, se proporciona una librería de soporte conocida con el nombre de *API de Colecciones*. La mayor parte de la API de Colecciones reside en `java.util`. Esta API proporciona una serie de estructuras de datos. También proporciona algunos algoritmos genéricos, como por ejemplo de ordenación. La API de Colecciones hace un uso intensivo de la herencia.

Nuestro objetivo principal es describir, en términos generales, algunos ejemplos y aplicaciones de las estructuras de datos. Nuestro objetivo secundario es explicar los fundamentos de la API de Colecciones, para poder utilizarla en la Parte Tres. No trataremos sobre la teoría que subyace a una implementación eficiente de la API de Colecciones hasta la Parte Cuatro, en cuyo momento proporcionaremos implementaciones simplificadas de algunos elementos fundamentales de la API de Colecciones. Pero no representa ningún problema el retrasar las explicaciones sobre la API de Colecciones hasta después de que hayamos aprendido a utilizarla. No necesitamos saber *cómo* está implementado algo, siempre y cuando estemos seguros de que *está* implementado.

En este capítulo, veremos

- Estructuras de datos comunes, sus operaciones permitidas y sus tiempos de ejecución.
- Algunas aplicaciones de las estructura de datos.
- La organización de la API de Colecciones y su integración con el resto del lenguaje.

6.1 Introducción

Las estructuras de datos nos permiten conseguir un objetivo importante de la programación orientada a objetos: la reutilización de componentes. Las estructuras de datos descritas en esta sección (e implementadas posteriormente en la Parte Cuatro) tienen usos recurrentes. Una vez que cada estructura de datos ha sido implementada, se la puede emplear todas las veces que se quiera en diversas aplicaciones.

Una *estructura de datos* es una representación de los datos y de las operaciones permitidas con esos datos. Muchas de las estructuras de datos comunes, aunque no todas ellas, almacenan una colección de objetos y luego proporcionan métodos para añadir un nuevo objeto de la colección, eliminar un objeto existente o acceder a uno de los objetos contenidos en la misma.

Una estructura de datos es una representación de datos y de las operaciones permitidas con dichos datos.

En este capítulo, vamos a examinar algunas de las estructuras de datos fundamentales y sus aplicaciones. Utilizando un protocolo de alto nivel, escribiremos operaciones típicas que normalmente están soportadas por las estructuras de datos y describiremos también brevemente sus usos. Siempre que sea posible, daremos una estimación del coste de implementar de manera eficiente estas operaciones. Esta estimación se basará a menudo en la analogía con aplicaciones no informáticas de la estructura de datos. Nuestro protocolo de alto nivel usualmente soporta tan solo un conjunto fundamental de operaciones básicas. Posteriormente, cuando describamos los fundamentos de cómo pueden implementarse las estructuras de datos (en general, existirán múltiples ideas que compiten entre sí), podremos centrarnos más fácilmente en los detalles algorítmicos independientes del lenguaje si restringimos el conjunto de operaciones a un conjunto mínimo fundamental.

Las estructuras de datos nos permiten reutilizar los componentes.

Como ejemplo, la Figura 6.1 ilustra un protocolo genérico que muchas estructuras de datos tienden a seguir. No vamos a utilizar realmente este protocolo de forma directa en ningún código. Sin embargo, una jerarquía de estructuras de datos basada en la herencia podría utilizar esta clase como punto de partida.

La API de colecciones es una librería de estructuras de datos y algoritmos cuya disponibilidad está garantizada.

Después, proporcionaremos una descripción de la interfaz que la API de Colecciones proporciona para estas estructuras de datos. No queremos decir en modo alguno que la API de Colecciones represente la mejor forma de hacer las cosas. Sin embargo, lo que sí representa es una librería de estructuras de datos y algoritmos cuya disponibilidad está garantiza. Su uso ilustra también algunos de los problemas fundamentales que tendremos que resolver en cuanto empecemos a tener en cuenta los aspectos teóricos.

Retrasaremos la consideración de cómo implementar las estructuras de datos de manera eficiente a la Parte Cuatro. En ese punto proporcionaremos, como parte del paquete `weiss.nonstandard`, algunas implementaciones alternativas de estructuras de datos que se ajustan a los protocolos simples desarrollados en este capítulo. También desarrollaremos una implementación para los componentes básicos de la API de Colecciones descritos en el capítulo, dentro del paquete `weiss.util`. Por tanto, estaremos separando la interfaz de la API de Colecciones (es decir, lo que hace, que es lo que describimos en este capítulo) de su implementación (es decir, cómo se hace, lo que abordaremos en la Parte Cuatro). Esta técnica –la separación de la interfaz de la implementación– forma parte del paradigma de orientación a objetos. El usuario de la estructura de datos solo necesita ver las operaciones disponibles, no la implementación. Recuerde que estos son los conceptos de encapsulación y ocultamiento de la información que se utilizan en la programación orientada a objetos.

El resto de este capítulo se organiza de la forma siguiente: en primer lugar, expondremos los fundamentos del *patrón iterador*, que se usa a lo largo de toda la API de Colecciones. Después, explicaremos la interfaz de los contenedores e iteradores de la API de Colecciones. A continuación describiremos algunos algoritmos de la API de Colecciones y, finalmente, examinaremos algunas otras estructura de datos, muchas de las cuales está soportadas en la API de Colecciones.

```
1  package weiss.nonstandard;
2
3  // protocolo SimpleContainer
4  public interface SimpleContainer<AnyType>
5  {
6     void insert( AnyType x );
7     void remove( AnyType x );
8     AnyType find( AnyType x );
9
10    boolean isEmpty( );
11    void makeEmpty( );
12 }
```

Figura 6.1 Protocolo genérico para muchas estructuras de datos.

6.2 El patrón iterador

Un objeto *iterador* controla la iteración a través de una colección.

La API de Colecciones hace uso de una técnica común conocida con el nombre de *patrón iterador*. Así que, antes de comenzar con las explicaciones sobre la API de Colecciones, vamos a examinar las ideas que subyacen al patrón iterador.

Considere el problema de imprimir los elementos de una colección. Típicamente, la colección será una matriz, por lo que si asumimos que el objeto `v` es una matriz, podemos imprimir fácilmente su contenido con un código como el siguiente:[1]

```
for( int i = 0; i < v.length; i++ )
System.out.println( v[ i ] );
```

En este bucle, `i` es un objeto iterador, porque es el objeto que se emplea para controlar la iteración. Sin embargo, utilizar el entero `i` como iterador restringe el diseño: solo podemos almacenar la colección en una estructura de tipo matricial. Una alternativa más flexible consiste en diseñar una clase iteradora que encapsule una posición dentro de una colección. La clase iteradora proporciona métodos para recorrer la colección.

Cuando programamos de acuerdo con una interfaz, escribimos código que utilice los métodos más abstractos posible. Estos métodos se aplicarán a los tipos concretos actuales.

La clave es el concepto de programación de acuerdo con una interfaz: queremos que el código que realice el acceso al contenedor sea lo más independiente posible del tipo de contenedor. Esto se lleva a cabo empleando únicamente métodos que sean comunes para todos los contenedores y sus iteradores.

Hay muchos posibles diseños distintos de iterador. Si sustituimos `int i` por `IteratorType itr`, entonces el bucle anterior nos quedaría

[1] El bucle `for` avanzado añadido en Java 5 constituye simplemente una sintaxis adicional. El compilador expande el bucle `for` avanzado para obtener el código que se muestra aquí.