CAPITULO 4. AGRUPAR OBJETOS

4.1 Agrupar objetos en colecciones de tamaño flexible

Frecuentemente se necesitan agrupar los objetos en colecciones; muchas veces el número de elementos almacenados en la colección varía a lo largo del tiempo. Una solución adecuada sería aquella que no requiera que conozcamos anticipadamente la cantidad de elementos que queremos agrupar o bien, establecer un límite mayor que dicho número.

4.3 Una primera visita a las bibliotecas de clases

Los lenguajes orientados a objetos suelen estar acompañados de *bibliotecas de clases.* Estas bibliotecas contienen varios cientos o miles de clases diferentes que han demostrado ser de gran ayuda para los desarrolladores en un amplio rango de proyectos diferentes. Java cuenta con varias de estas bibliotecas y las denomina *paquetes* (packages).

La clase ArrayList, que es un ejemplo de una clase *colección* está definida en el paquete *java.util*. Las **colecciones de objetos** son objetos que pueden almacenar un número arbitrario de otros objetos.

El acceso a una clase de una biblioteca de Java se obtiene mediante la *sentencia import*, para ArrayList sería:

```
import java.util.ArrayList;
```

Esta sentencia hace que la clase ArrayList del paquete java.util esté disponible para nuestra clase. Las sentencias import deben ubicarse en el texto de la clase, siempre antes del comienzo de la declaración de la clase. Una vez que el nombre de una clase ha sido importado desde un paquete de esta manera, podemos usar dicha clase tal como si fuera una de nuestras propias clases, por ejemplo como en la siguiente declaración:

```
private ArrayList<String> notas;
```

Aquí se observa una nueva construcción: la mención de String entre símbolos de menor “<” y de mayor “>”: <String>.

Cuando usamos colecciones, debemos especificar dos tipos: el tipo propio de la colección y el tipo de los elementos que planeamos almacenar en la colección. Para instanciar un nuevo objeto se necesita especificar de nuevo el tipo completo con el tipo de elemento entre los símbolos de menor y de mayor, seguido de los paréntesis para la lista de parámetros:

```
notas = new ArrayList<String>();
```

Las clases similares a ArrayList que se parametrizan con un segundo tipo se denominan *clases genéricas*.

La clase ArrayList declara muchos métodos entre ellos: add, size y get. add sirve para añadir nuevos elementos, size devuelve el número total de elementos y get devuelve un elemento, sin eliminarlo de la colección.

4.4 Estructuras de objetos con colecciones

Características importantes de la clase ArrayList son:

- Es capaz de aumentar su capacidad interna tanto como se requiera: cuando se agregan más elementos, simplemente hace suficiente espacio para ellos.
- Mantiene su propia cuenta privada de la cantidad de elementos almacenados, que se obtiene mediante *size*.
- Mantiene el orden de los elementos que se agregan, por lo que más tarde se pueden recuperar en el mismo orden.

4.5 Clases genéricas

Las clases genéricas no definen un tipo único en Java sino potencialmente muchos tipos. Por ejemplo, la clase ArrayList puede usarse para especificar un *ArrayList de Strings*, un *ArrayList de Personas*, un *ArrayList de Rectángulos*, o un ArrayList de cualquier otra clase. Cada ArrayList en particular es un tipo distinto que puede usarse en declaraciones de campos, parámetros y tipos de retorno.

Hay que tener en cuenta que ArrayList<Persona> y ArrayList<MaquinaDeBoletos> son tipos diferentes. Los campos no pueden ser asignados uno a otro, aun cuando sus tipos deriven de la misma clase.

4.6 Numeración dentro de las colecciones

Los elementos almacenados en las colecciones tienen una numeración implícita o posicionamiento que comienza a partir de cero. La posición que ocupa un objeto en una colección se conoce como su *índice*. El primer elemento que se agrega a una colección tiene por índice al número 0, el segundo tiene al número 1, y así sucesivamente hasta size -1.

Si se intenta acceder a un elemento de una colección que está fuera de los índices válidos del ArrayList se obtendrá un mensaje del error denominado *desbordamiento*, que en Java será *IndexOutBoundsException.*

4.7 Eliminar un elemento de una Colección

La clase ArrayList tiene un método remove que toma como parámetro el índice del elemento o el propio elemento a eliminar.

Una complicación del proceso de eliminación es que se modifican los valores de los índices de los restantes objetos que están almacenados en la colección.

4.8 Procesar Una colección completa

4.8.1 El *ciclo for-each*

Un **ciclo o bucle** puede usarse para ejecutar repetidamente un bloque de sentencias sin tener que escribirlas varias *veces.*

for-each es un tipo de bucle. Las acciones de un *ciclo for-each* se pueden resumir en el siguiente pseudocódigo:

```
for (TipoDelElemento elemento: colección) {
        cuerpo del ciclo
}
```

Consta de dos partes: un encabezado de ciclo (la primera línea del ciclo) y un cuerpo a continuación del encabezado. El cuerpo contiene aquellas sentencias que deseamos llevar a cabo una y otra vez. Una forma de entender este bucle sería:

```
Para cada elemento en la colección hacer: {
        cuerpo del ciclo
 }
```

En cada vuelta, antes de que la sentencia se ejecute, la variable *elemento* se configura para contener uno de los elementos de la lista: primero el del índice 0, luego el del índice 1, y así sucesivamente. Llamaremos *variable de bucle* a la variable que se usará para contener los elementos de la lista.

El tipo de la variable de bucle debe ser el mismo que el tipo del elemento declarado para la colección que estamos usando, a continuación aparecen dos puntos y la variable que contiene la colección que deseamos procesar. Cada elemento de esta colección será asignado en su turno a la variable de ciclo, y para cada una de estas asignaciones el cuerpo del ciclo se ejecutará una sola vez.

4.8.2 El *ciclo while*

Un ciclo *while* consiste en un encabezado de ciclo y un cuerpo, y el cuerpo puede ejecutarse repetidamente. La estructura de un *ciclo while* sería:

```
while (condición del ciclo) {
        cuerpo del ciclo
}
```

El *ciclo while* comienza con la palabra clave *while*, seguida de una condición. Este ciclo es más flexible que el *ciclo for-each.* Puede recorrer un número variable de elementos de la colección, dependiendo de la condición del bucle. La condición es una expresión lógica que se usa para determinar si el cuerpo debe ejecutarse por lo menos una vez. Si la condición se evalúa verdadera, se ejecuta el cuerpo del ciclo. Cada vez que se ejecuta el cuerpo del ciclo, la condición se vuelve a controlar nuevamente. Este proceso continúa repetidamente hasta que la condición resulta falsa, que es el punto en el que se salta del cuerpo del ciclo y la ejecución continúa con la sentencia que esté ubicada inmediatamente después del cuerpo.

Hay que declarar fuera del ciclo una variable para el índice e iniciarlo por nuestros propios medios a 0 para acceder al primer elemento de la lista. También tenemos que llevar nuestra propia cuenta del índice para recordar la posición en que estábamos.

Existe un operador especial para incrementar una variable numérica en 1: var++; que es equivalente a: var = var + 1;

Hay dos puntos más a destacar, el *ciclo while* no necesita estar relacionado con una colección y aunque se use el ciclo para procesar una colección, no necesitamos procesar cada uno de sus elementos.

El método contains(<string>) de String devuelve verdadero si una cadena contiene a otra pasada como parámetro.

4.8.3 Recorrer una colección

Existe una tercera variante para recorrer una colección, que está entre medio de los ciclos *while* y *for-each*. Usa un ciclo while para llevar a cabo el recorrido y un *objeto iterador* en lugar de una variable entera como índice para mantener el rastro de la posición en la lista.

Un **iterador** es un objeto que proporciona funcionalidad para recorrer todos los elementos de una colección.

El método iterator de ArrayList devuelve un objeto Iterator. La clase Iterator también está definida en el paquete java.util. Un Iterator provee dos métodos para recorrer una colección: hasNext y next . La manera de uso en pseudocódigo sería:

Iterator<*TipoDelElemento*> it = miColeccion.iterator ();
while (it.hasNext ( )) {
Invocar it. next () *para obtener el siguiente elemento*
Hacer algo con dicho elemento
}

Iterator también es de tipo genérico por lo que hay que parametrizarlo con el tipo de los elementos de la colección. Luego usamos dicho iterador para controlar repetidamente si hay más elementos (it.hastNext ( )) y para obtener el siguiente elemento (it.next ( )). Un punto a destacar es que le pedimos al iterador que devuelva el siguiente elemento y no a la colección.

4.8.4 Comparación de acceso entre índices e iteradotes

El *ciclo for-ea ch*, es la técnica estándar que se usa si deben procesarse todos los elementos de una colección porque es el más breve para este caso. El bucle while (con un índice y el método get) y el iterador tienen la ventaja de que la iteración puede ser detenida más fácilmente en mitad del proceso, de modo que son mejores cuando se quiere procesar sólo una parte de una colección.

Para algunas colecciones, es imposible o muy ineficiente acceder a elementos individuales mediante un índice y se accede mediante for-each o iterador.

El iterador está disponible para todas las colecciones de las clases de las bibliotecas de Java y es un patrón importante que se usará a menudo.

Null. Se usa la palabra reservada **null** de Java para significar que «no hay objeto», es decir cuando una variable objeto no está haciendo referencia realmente ningún objeto. Un campo que no haya sido inicializado explícitamente contendrá el valor por defecto null.

4.10.3 Objetos anónimos

La siguiente sentencia ilustra el uso de objetos anónimos: miColeccion.add(new Elemento());

Aquí estamos haciendo dos cosas: crear un nuevo objeto Elemento y pasar este nuevo objeto al método add de miColeccion. Podríamos haber escrito lo mismo en dos líneas de código, una para declarar el nuevo elemento y otra para pasárselo a la colección. Ambas versiones son equivalentes, pero la primera versión evita declarar una variable que puede tener un uso muy limitado. Se crea un objeto anónimo, un objeto sin nombre, pasándoselo directamente al método que lo utiliza.

Dos objetos String s1 y s2 pueden compararse para ver si son iguales mediante la expresión lógica: s1.equals(s2).

4.12 Colecciones de tamaño fijo

A veces conocemos anticipadamente cuántos elementos deseamos almacenar en la colección y este número permanece invariable durante la vida de la colección. En tales circunstancias, tenemos la opción de elegir usar una colección de objetos de tamaño fijo. Una colección de tamaño fijo se denomina *array* o *vector*. Un **vector** es un tipo especial de colección que puede almacenar un número fijo de elementos.

Se obtienen ventajas, en compensación, con respecto a las clases de colecciones de tamaño flexible:
• El acceso a los elementos de un vector es generalmente más eficiente que el acceso a los elementos de una colección de tamaño flexible.
• Los vectores son capaces de almacenar objetos o valores de tipos primitivos. Las colecciones de tamaño flexible sólo pueden almacenar objetos (sin tener en cuenta el autoboxing).

Otra característica distintiva de los vectores es que tienen una sintaxis especial en Java, ya que a sus elementos se accede mediante “[]”.

4.12.2 Declaración de variables vectores

La característica distintiva de la declaración de una variable de tipo vector es un par de corchetes que forman parte del nombre del tipo: int[ ]. Este detalle indica que la variable declarada es de tipo *vector de enteros.* Decimos que int es el tipo base de este vector en particular. La declaración de una variable vector no crea en sí misma un objeto vector, sólo reserva un espacio de memoria para que en un próximo paso, usando el operador new, se cree el vector tal como con los otros objetos.

4.12.3 Creación de objetos vector

La siguiente sentencia muestra como se asocia una variable vector con un objeto vector:

```
obejetoVector = new int[tamaño] ;
```

La forma general de la construcción de un objeto vector es: new *tipo[expresión-entera]*. La elección del tipo especifica de qué tipo serán todos los elementos que se almacenarán en el vector. La expresión entera especifica el tamaño del vector.

Cuando se asigna un objeto vector a una variable vector, el tipo del objeto vector debe coincidir con la declaración del tipo de la variable.

Cuando se crea un vector no crea tantos objetos como tiene capacidad para almacenar, solo crea una colección de tamaño fijo que es capaz de almacenar dichos objetos.

4.12.4 Usar objetos vector

Se accede a los elementos individuales de un objeto vector mediante un *índice.* Un índice es una expresión entera escrita entre un par de corchetes a continuación del nombre de una variable vector.

Los valores válidos para una expresión que funciona como índice dependen de la longitud del vector en el que se usarán. Los índices de los vectores siempre comienzan por cero y van hasta el valor del tamaño del vector menos uno.

Las expresiones que seleccionan un elemento de un vector se pueden usar en cualquier lugar que requiera un valor del tipo base del vector. Esto quiere decir que podemos usarlas, por ejemplo, en ambos lados de una asignación.

El uso de un índice de un vector en el lado izquierdo de una asignación es equivalente a un método de modificación (o método *set)* del vector porque cambiará el contenido del mismo. Los restantes usos del índice son equivalentes a los métodos de acceso (o métodos *get*).

4.12.6 El ciclo for

Java define dos variantes para el *ciclo for,* el bucle *for-each,* y el *bucle for,* que es una estructura de control repetitiva alternativa que resulta particularmente adecuada cuando:

- queremos ejecutar un conjunto de sentencias un número exacto de veces.
- necesitamos una variable dentro del ciclo cuyo valor cambie en una cantidad fija, generalmente en 1, en cada iteración.

Es común el uso del *ciclo for* cuando queremos hacer algo con cada elemento de un vector tal como imprimir el contenido de cada elemento. Un *ciclo for* tiene la siguiente forma general:

```
for (inicialización; condición; acción modificadora) {
   setencias a repetir
}
```

En este *ciclo for,* los paréntesis contienen tres secciones distintas separadas por símbolos de punto y coma (;).

Todos los vectores contienen un campo length que contiene el valor del tamaño del vector. El valor de este campo coincide siempre con el valor entero usado para crear el objeto vector. Por eso la condición del bucle suele usar el operador menor que "<" para controlar el valor del índice respecto de la longitud del vector.

En general, cuando deseamos acceder a cada elemento de un vector, el encabezado del bucle for tendrá la siguiente forma:

```
for(int indice = 0; indice < vector.length; indice ++)
```

¿Qué ciclo debo usar?

- Si se necesita recorrer todos los elementos de una colección, el *ciclo for-each* es, casi siempre, el ciclo más elegante para usar, aunque no provee una variable contadora de ciclo.

• Si se tiene un bucle que no está relacionado con una colección (pero lleva a cabo un conjunto de acciones repetidamente), el *ciclo for-each* no resulta útil. En este caso, se puede elegir entre el *ciclo for* y el *ciclo while*. El *ciclo for-each* es sólo para colecciones.

• El *ciclo for* es bueno si conoce anticipadamente la cantidad de repeticiones necesarias. Esta información puede estar dada por una variable, pero no puede modificarse durante la ejecución del ciclo. Este ciclo también resulta muy bueno cuando necesita usar explícitamente una variable contadora.

• El *ciclo while* será el adecuado si, al comienzo del ciclo, no se conoce la cantidad de iteraciones que se deben realizar. El fin del ciclo puede determinarse previamente mediante alguna condición.