Estructuras encadenadas

Introducción

Java, como otros lenguajes modernos orientados a objetos, trabaja fundamentalmente en base a referencias. Cuando se declara una variable de una clase determinada (no de un tipo primitivo), lo que se está creando es una entidad que sirve para referenciar un objeto de esa clase, cuando este se cree.

En un objeto, los campos que no sean de un tipo primitivo, almacenan referencias a otros objetos, como en el ejemplo mostrado en la Ilustración 1, en el que los campos *nombre*, *primerApellido* y *segundoApellido*, de la clase *Persona*, y los campos *matrícula* y *marca*, de la clase *Coche*, son referencias a objetos de la clase *String*, mientras que el campo *propietario*, de la clase *Coche*, es una referencia a un objeto de clase *Persona*.

```
public class Persona{
    private String nombre;
    private String primerApellido
    private String segundoApellido;
}
…
```

```
public class Coche{
    private Persona propietario;
    private String matrícula
    private String marca;
}
…
```

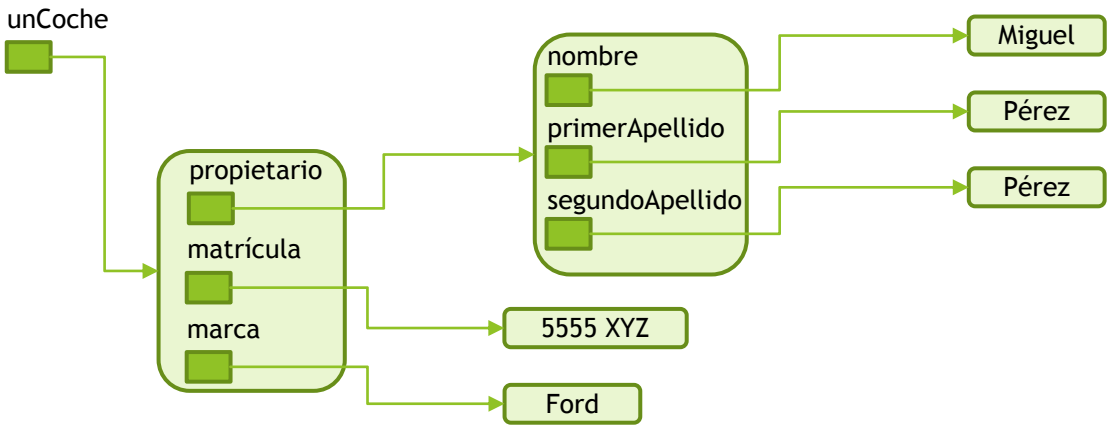

Ilustración 1

Por medio de las referencias, los objetos se encadenan unos con otros, formando estructuras más o menos complejas, dependiendo de las necesidades de la información a representar.

Una estructura encadenada está formada por un conjunto de objetos, generalmente, conocidos como nodos de la estructura, que se referencian unos a otros. Deben existir objetos terminales, que no

referencian a otros objetos, así como al menos un objeto inicial (según la estructura puede denominarse primero, frente, cima, raíz, etc.) que es referenciado desde "fuera" de la estructura y sirve como "punto de entrada" a la misma (en la Ilustración 1, el objeto inicial, el primero de la cadena, es el referenciado por la variable *unCoche*).

Estructuras recursivas

Si un objeto, de una clase *X*, tiene campos que son referencias a otros objetos, cabe plantearse si no podrían ser referencias a otros objetos de la misma clase *X*. En el ejemplo mostrado en la Ilustración 2 se hace precisamente eso, al añadir a la clase *Persona* un campo, *padre*, que es, a su vez, una referencia a un objeto de la clase *Persona*. La idea es que los objetos de la clase *Persona* tengan acceso a información sobre el padre (persona) de la persona representada.

```
public class Persona{
    …
    private Persona padre;
}
```

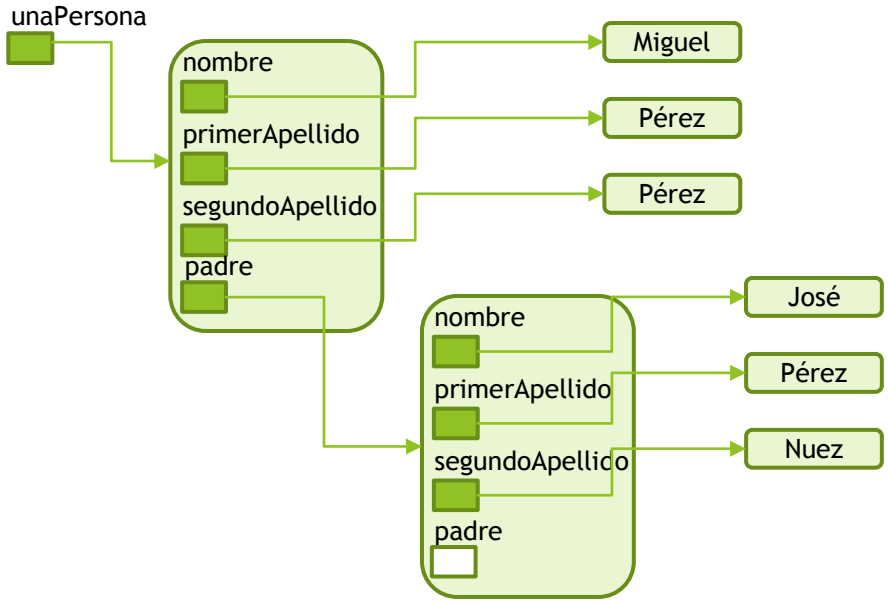

Ilustración 2

Esta definición da lugar a una estructura recursiva que puede expandirse indefinidamente, formando una cadena (Ilustración 3) en la que cada objeto de la clase *Persona* referencia a otro objeto de la clase *Persona*. Naturalmente, en la praxis computacional esta estructura no puede ser infinita, de tal

manera que la recursión debe acabar con una "condición de base", que, necesariamente, es un objeto Persona que no referencia a su padre (campo de referencia con valor *null*).

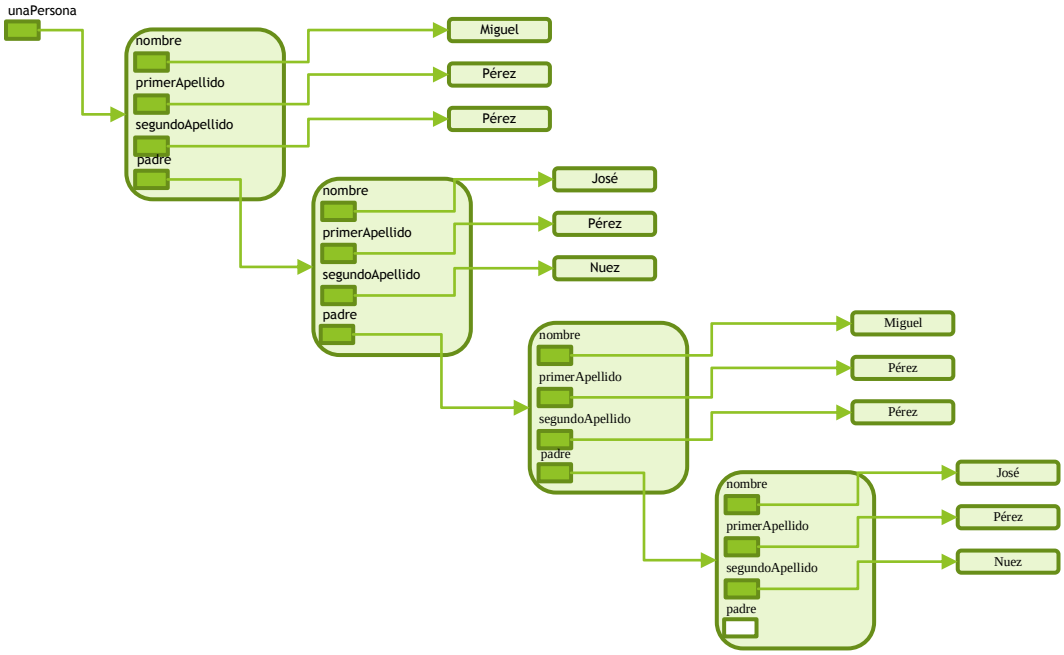

Ilustración 3

Igual que un objeto puede contener muchas referencias a objetos de otras clases, también puede contener múltiples referencias a objetos de la misma clase, como en el ejemplo de la Ilustración 4, en que se añade a la clase *Persona* un segundo campo, *madre*, para referenciar objetos de la clase *Persona*, obteniéndose así una estructura no lineal (un "árbol" genealógico), en oposición a la estructura lineal de la Ilustración 3.

El encadenamiento otorga una gran flexibilidad en la creación de estructuras adaptadas a las necesidades de la información a representar, y permite implementar estructuras de gran complejidad.

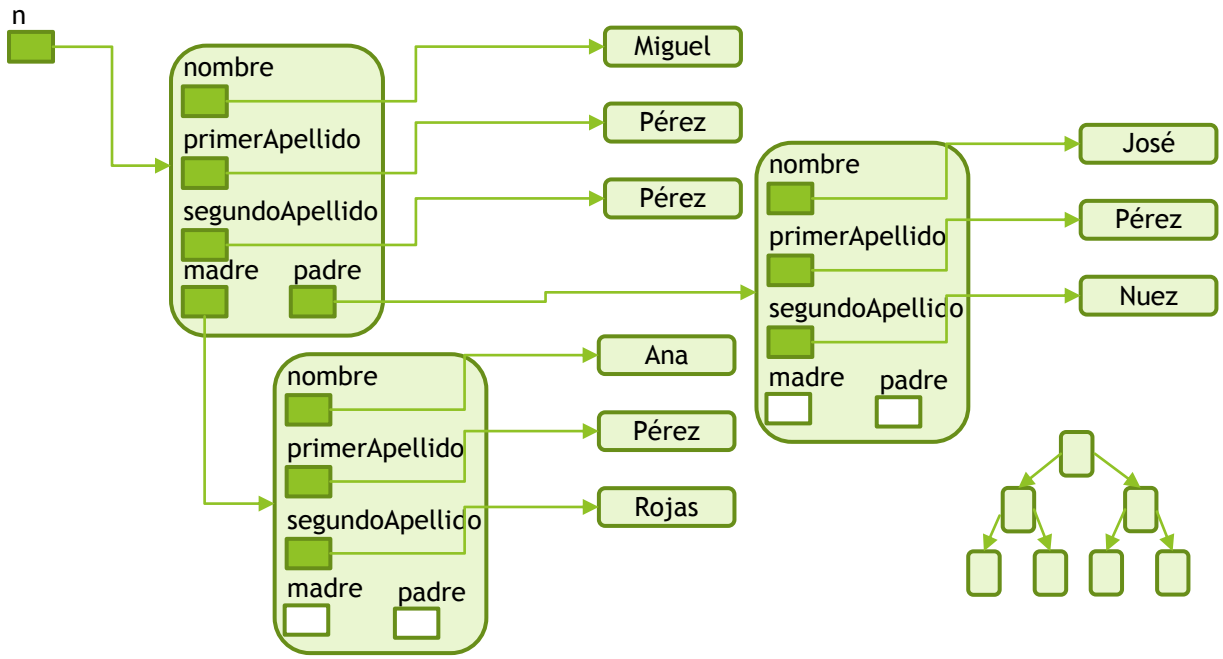

Ilustración 4

Ventajas de las estructuras encadenadas

Una estructura encadenada está formada por un conjunto de objetos, generalmente, conocidos como nodos de la estructura, que se referencian unos a otros. Deben existir objetos terminales, que no referencian a otros objetos, así como al menos un objeto inicial (según la estructura puede denominarse primero, frente, cima, raíz, etc.) que es referenciado desde "fuera" de la estructura y sirve como "punto de entrada" a la misma (en Ilustración 3, el objeto inicial, el primero de la cadena, es el referenciado por la variable *unaPersona*). Las estructuras encadenadas presentan diversas ventajas, y, también, algunos inconvenientes.

Distribución

Dado que los objetos en una estructura encadenada se referencian unos a otros, no necesitan ocupar posiciones contiguas en el espacio de almacenamiento, sino que pueden estar distribuidos por todo él. Esto permite aprovechar un espacio de almacenamiento fragmentado. Por ejemplo, en lenguajes como C++ y otros, que no usan las referencias como elemento básico de la representación de la información, si queremos almacenar en un *array* 100 objetos de tamaño *T*, necesitamos que en la memoria haya un

bloque contiguo de tamaño mayor o igual a *100*T*; sin embargo, si los almacenamos usando una estructura encadenada, solo necesitaremos que haya 100 bloques, contiguos o separados, de tamaño mayor o igual a *T*.

En lenguajes como Java, el problema se suaviza porque el uso intensivo de referencias hace que solo necesitemos espacio para un *array* de tamaño 100 veces el de una referencia, estando los objetos igualmente distribuidos por todo el *heap*, a menos que sean de un tipo primitivo, que son los únicos que se almacenarían directamente en el *array*.

Dinamismo

Siguiendo con el ejemplo del apartado anterior, si hemos almacenado 100 objetos de la clase *X* en un bloque contiguo, y de repente, durante la ejecución de programa, hace falta almacenar un objeto más, necesitaríamos, en el mejor de los casos, que se dispusiese de espacio a continuación del ya utilizado, aunque no suele ser tan fácil. Lo corriente es que si hemos almacenado los objetos en un *array* (*X[100]*), necesitaremos crear un *array* de mayor tamaño y copiar los objetos ya existentes a ese nuevo *array*, con lo que, aparte del tiempo que consuma hacer la copia, durante un intervalo necesitaremos disponer del doble de memoria de la que realmente hace falta para representar la información. Nuevamente, en Java el problema se suaviza porque el *array* solo contendría referencias, que ocupan, normalmente, mucho menos espacio que los objetos en sí. No obstante, con una estructura encadenada, lo único que necesitamos es que haya espacio disponible, en cualquier parte, para crear un objeto más.

Pero es más, si los objetos siguen un determinado orden, y en una colección de objetos ya existentes hay que insertar uno más en una posición concreta con respecto al resto, cuando usamos ubicación contigua, hay que hacer un desplazamiento de elementos para abrir hueco donde insertar el nuevo objeto, o la referencia al nuevo objeto, como muestra la Ilustración 5.

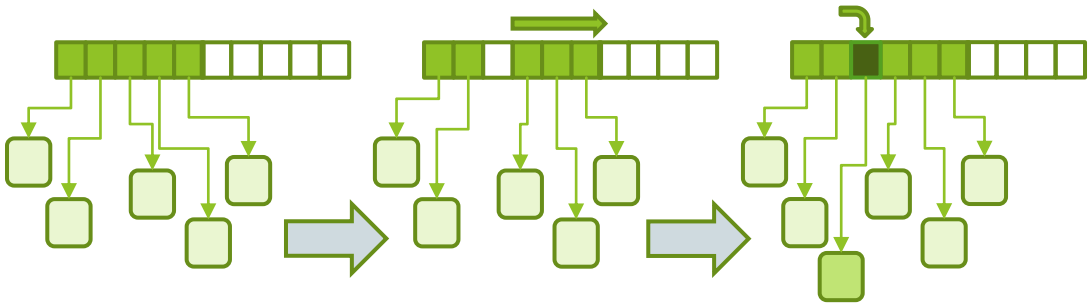

Ilustración 5

Sin embargo, cuando los objetos están encadenados entre sí, no es necesario hacer ningún desplazamiento, sino ajustar algunas referencias para que el nuevo nodo quede en la posición adecuada, tal como se muestra en la Ilustración 6.

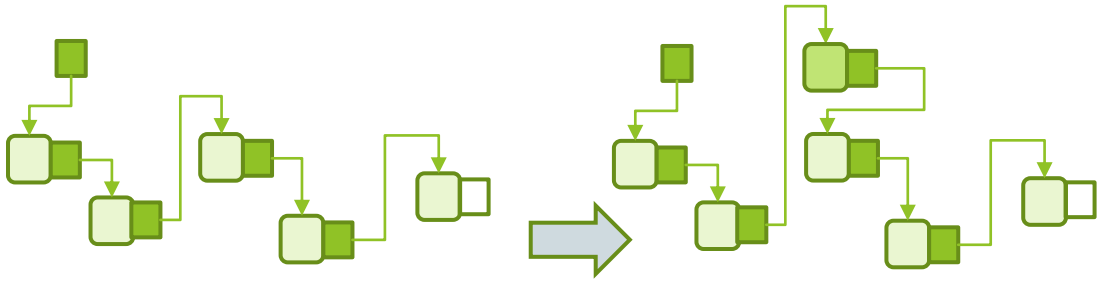

Ilustración 6

Consideraciones equivalentes se podrían hacer respecto a la extracción de un elemento de entre un conjunto de ellos: en la ubicación contigua habría que desplazar los elementos posteriores para tapar el hueco dejado por el que se extrae, mientras que en una estructura encadenada basta con modificar alguna(s) referencias para "puentearlo".

Flexibilidad

En el apartado anterior, se hizo referencia a la posibilidad de tener una colección de objetos que "sigan un determinado orden". Esto significa que el elemento "más pequeño" según ese orden tiene que estar el primero, y, a cada elemento, le sigue uno, que es igual o mayor que él, y, menor o igual que los que están después.

En ubicación contigua, este orden se traslada directamente al espacio físico: si hablamos de un array, el elemento más pequeño está en la posición 0, el siguiente en la 1, y así, sucesivamente. En una estructura encadenada, el orden se refleja en los encadenamientos: el nodo inicial contiene el valor más pequeño y referencia al nodo que contiene el siguiente elemento en el orden establecido.

El problema, es que, como en la ubicación contigua el orden lógico se corresponde con el orden físico, los elementos sólo pueden estar en un único orden, mientras que en una estructura encadenada basta con añadir campos de encadenamiento adicionales para poder reflejar órdenes diferentes con los mismos elementos, tal como muestra la Ilustración 7, en la que una colección de ristras está encadenada en dos órdenes diferentes: alfabético y por longitud, cada uno, además, con una referencia de inicio diferente.

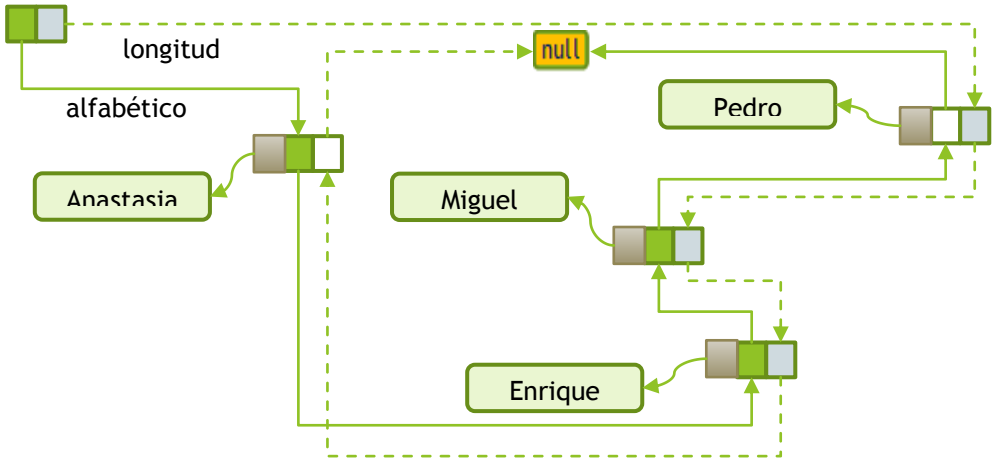

Ilustración 7

En el ejemplo de la Ilustración 7 se tienen dos cadenas lineales superpuestas, usando los mismos nodos. Además, como se mostró en la estructura no lineal de la Ilustración 4, se pueden añadir campos de encadenamiento con el propósito de crear estructuras de mayor complejidad, permitiendo un alto grado de flexibilidad para adaptarse a los requerimientos de un problema concreto.

Inconvenientes de las estructuras encadenadas

Consumo de memoria

Uno de los inconvenientes que, tradicionalmente, se le puede achacar a las estructuras encadenadas es un mayor consumo de memoria; el cálculo es sencillo: si necesitamos almacenar una colección de n elementos de información, cada uno de los cuales tiene un tamaño t, al almacenarlos en un *array* ocuparán un espacio de $n*t$. Si se almacenan en una estructura encadenada, la más sencilla, una lista con un solo encadenamiento en cada nodo, si el tamaño de un campo de encadenamiento es e, necesitará un espacio de $n*(t + e)$, más un bloque adicional de tamaño e para la variable que referencia al primer nodo; es decir, la estructura encadenada mínima necesita un espacio extra de tamaño $n*e + 1$ frente a la estructura contigua equivalente, con lo que, en términos de consumo de memoria, la idoneidad de la estructura encadenada dependería de la relación entre el tamaño de la información útil de un nodo, t, y el tamaño de los campos de encadenamiento de un nodo, e. Si $t \leq e$, la estructura encadenada estará, como mínimo, doblando las necesidades de espacio.

Sin embargo, cuando el lenguaje utilizado usa referencias de manera intensiva, la cuestión del consumo de memoria ya no queda tan clara, como muestra la Ilustración 8, en la que se observa que la representación típica de un array en Java ya incluye el espacio extra para las referencias, e incluso más, si está dimensionado a un tamaño mayor, que el número de objetos que realmente está referenciando.

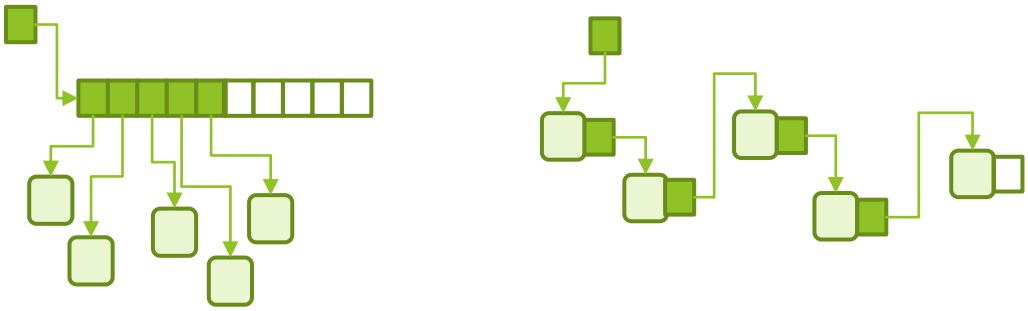

Ilustración 8

Direccionamiento

Hay un hecho en el que una estructura contigua del tipo de un *array* supera a una estructura encadenada, y es la capacidad de acceso a un elemento concreto. En una estructura encadena como la que se muestra en la Ilustración 8, acceder al último elemento de la estructura requiere pasar por todos los anteriores; en general, acceder a cualquier elemento requiere pasar por todos los anteriores. Esto significa que el coste (tiempo necesario) de acceder a los distintos elementos no es uniforme, sino que se incrementa a medida que el elemento está más atrás en la cadena.

En un *array*, por el contrario, el coste de acceder a cualquier elemento es siempre el mismo, ya que solo hay que hacer un pequeño cálculo a partir de su índice para saber en qué posición exacta de la memoria se encuentra.

Nota final

Aunque en este documento se han tratado las estructuras encadenadas usando ejemplos que, implícitamente, las ubican en el *heap*, en realidad el concepto de "encadenar" es independiente del espacio de almacenamiento utilizado, siempre que dicho espacio permita el direccionamiento directo de sus elementos. Por ejemplo, a los elementos almacenados en un *array* se les podrían añadir campos índice, que les permitieran referenciar a elementos situados en otras posiciones del *array*, formando así cadenas dentro del mismo.