# Capítulo 14

# Listas enlazadas

## 14.1 Referencias en objetos

En el capítulo pasado vimos que las variables de instancia de un objeto pueden ser arreglos y mencioné que pueden ser objetos también.

Una de las posibilidades más interesantes es que un objeto puede contener una referencia a otro objeto del mismo tipo. Existe una estructura de datos muy común, la **lista**, que aprovecha esta característica.

Las listas están compuestas de **nodos**, donde cada nodo contiene una referencia al próximo nodo en la lista. Además, cada nodo usualmente contiene una unidad de datos llamada **carga**. En nuestro primer ejemplo, la carga será simplemente un entero, pero más adelante vamos a escribir una lista **genérica** que puede contener objetos de cualquier tipo.

## 14.2 La clase `Nodo`

Como de costumbre, cuando escribimos una nueva clase, empezamos con las variables de instancia, uno o dos constructores y algún método que nos permita testear el mecanismo básico de crear y mostrar el nuevo tipo, como por ejemplo `toString`.

```
public class Nodo {
    int carga;
    Nodo prox;

    public Nodo () {
        carga = 0;
        prox = null;
    }
```

```
    public Nodo (int carga, Nodo prox) {
        this.carga = carga;
        this.prox = prox;
    }

    public String toString () {
        return carga + "";
    }
}
```

Las declaraciones de las variables de instancia se deducen naturalmente de la especificación, y el resto es mecánico a partir de las variables de instancia. La expresión `carga + ""` es una manera, rara pero concisa, de convertir un entero a un String.

Para probar la implementación hecha hasta ahora, pondremos algo así en el `main`:

```
Nodo nodo = new Nodo (1, null);
System.out.println (nodo);
```

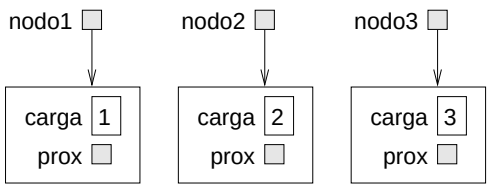
El resultado es simplemente

```
1
```

Para hacerlo interesante, ¡necesitamos una lista con más de un nodo!

```
Nodo nodo1 = new Nodo (1, null);
Nodo nodo2 = new Nodo (2, null);
Nodo nodo3 = new Nodo (3, null);
```

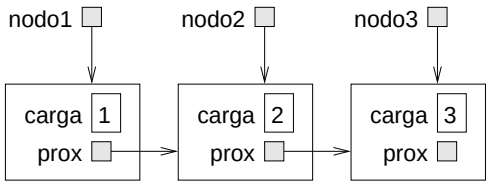
Este código crea tres nodos, pero no tenemos una lista aun ya que los nodos no están **enlazados**. El diagrama de estado se ve así:




Para enlazar los nodos, tenemos que hacer que el primer nodo referencie al segundo y el segundo al tercero.

```
nodo1.prox = nodo2;
nodo2.prox = nodo3;
nodo3.prox = null;
```

La referencia del tercer nodo es `null`, lo cual indica que es el final de la lista. Ahora el diagrama de estado se ve así:




Ahora ya sabemos cómo crear nodos y enlazarlos creando listas. Lo que puede no estar muy claro a esta altura es por qué hacer esto.

## 14.3 Listas como colecciones

El hecho que hace a las listas tan útiles es que son una manera de ensamblar muchos objetos en una sola entidad, llamada frecuentemente colección. En el ejemplo, el primer nodo de la lista sirve como una referencia a la lista entera.

Si queremos pasar la lista como parámetro a algún método, todo lo que necesitamos es pasar una referencia al primer nodo. Por ejemplo, el método `imprimirLista` toma un solo nodo como argumento. Empezando por el comienzo de la lista, imprime cada nodo hasta llegar al final (indicado por la referencia a `null`).

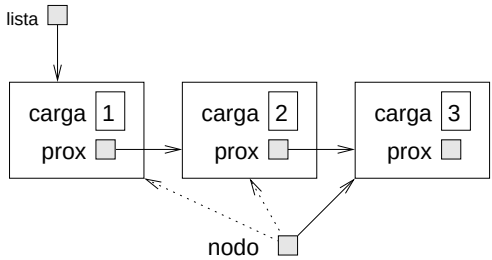
```
public static void imprimirLista (Nodo lista) {
    Nodo nodo = lista;

    while (nodo != null) {
        System.out.print (nodo);
        nodo = nodo.prox;
    }
    System.out.println ();
}
```

Para llamar a este método sólo tenemos que pasar una referencia al primer nodo:

```
imprimirLista (nodo1);
```

Dentro de `imprimirLista` tenemos una referencia al primer nodo de la lista, pero no tenemos ninguna variable que se refiera a los demás nodos. Tenemos que usar el valor de `prox` de cada nodo para llegar al próximo nodo.

Este diagrama de estado muestra el valor de `lista` y los valores que va tomando `nodo`:




Esta manera de moverse a través de una lista es llamada un **recorrido**, así como el patrón similar al moverse a través de los elementos de un arreglo. Es común usar un iterador como `nodo` para referenciar a cada nodo de la lista sucesivamente. La salida de este método es

```
123
```

Por convención, las listas se imprimen entre paréntesis con comas entre los elementos, como en `(1, 2, 3)`. A modo de ejercicio, modificá el método `imprimirLista` para que genere una salida con este formato.

Como otro ejercicio, reescribí `imprimirLista` usando un ciclo `for` en lugar de un ciclo `while`.

## 14.4 Listas y recursión

La recursión y las listas van juntas como las hamburguesas y las papas fritas. Por ejemplo, acá hay un algoritmo recursivo para imprimir el reverso de una lista:

1. Separar la lista en dos partes: el primer nodo (llamado la cabeza) y el resto (llamado la cola).

2. Imprimir el reverso de la cola.

3. Imprimir la cabeza.

Obviamente, el Paso 2, el llamado recursivo, asume que tenemos una forma de imprimir el reverso de una lista. Pero *si* asumimos que el llamado

recursivo funciona—el salto de fe—entonces podemos convencernos de que este algoritmo funciona.

Todo lo que necesitamos es el caso base, y una manera de probar que para cualquier lista, llegaremos eventualmente al caso base. Una elección natural para el caso base es una lista con un único elemento, aunque una mejor elección es la lista vacía, representada por null.

```
public static void imprimirInverso (Nodo lista) {
    if (lista == null) return;

    Nodo cabeza = lista;
    Nodo cola = lista.prox;

    imprimirInverso (cola);
    System.out.print (cabeza);
}
```

La primera línea se encarga del caso base sin hacer nada. Las siguientes dos líneas separan la lista en `cabeza` y `cola`. Las últimas dos líneas imprimen la lista.

Este método se llama exactamente igual que a `imprimirLista`:

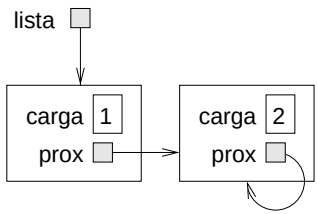
```
imprimirInverso(nodo1);
```

El resultado es el reverso de una lista.

¿Podemos demostrar que este método terminará siempre? En otras palabras, ¿llegará siempre al caso base? De hecho, la respuesta es que no. Existen algunas listas que harían que este método falle.

## 14.5 Listas infinitas

No hay manera de evitar que un nodo referencie a un nodo anterior de la lista, o incluso a sí mismo. Por ejemplo, esta figura muestra una lista de dos nodos, uno de los cuales se referencia a sí mismo.




Si llamamos `imprimirLista` en la lista, iterará por siempre.