

# Java. Práctica 2

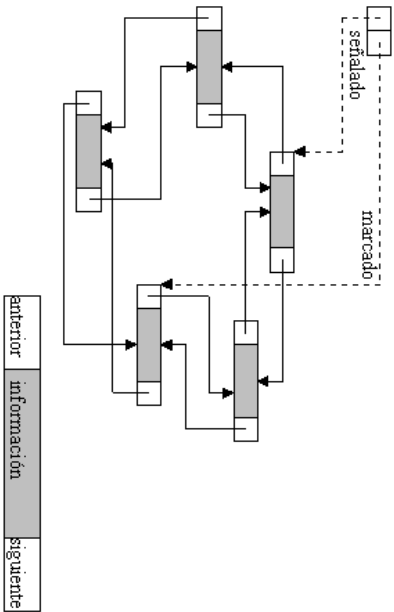
**Ejercicio 1.** Directorio de trabajo: `c:\java\util\lista`. Crear un paquete llamado `lista` que contenga la clase `Lista` capaz de almacenar una lista de objetos con un protocolo similar al de la lista de enteros de la práctica 1. Así mismo, incluir en el paquete las clases `Cola` y `Pila`.

**Ejercicio 2.** Directorio de trabajo: `c:\java\usolista`. Generar una aplicación que haga uso del paquete `lista` anterior. Utilizar *Writers* para introducir enteros en una lista. Para poder utilizar el paquete deberá estar incluido el directorio `c:\java\util` en el `CLASSPATH`.

**Ejercicio 3.** Directorio de trabajo: `c:\java\entrada`. Colocar la clase `ES` en el directorio `c:\java\util\est` para poderla utilizar como un paquete llamado `est`. Crear una aplicación que reciba datos desde el teclado a través de la clase `ES` y los muestre por el dispositivo de salida con un formato adecuado.

**Ejercicio 4.** Construir la clase `Anillo` para mantener información en una estructura circular. El anillo permitirá mantener dos referencias, una a un elemento del anillo que se denominará **señalado** (o `null` si no hay ningún elemento en el anillo), y otra a otro elemento del anillo que se denominará **marcado** (o `null` si no hay ninguno marcado).

Cada elemento del anillo (perteneciente a una clase `NodoAnillo`), mantiene referencias al **elemento anterior del anillo**, al **elemento siguiente**, y la **información** que se desea guardar. El gráfico muestra un anillo con 5 elementos



Los mensajes mínimos a responder por el anillo son:

- ☐ `void inserta(Object)`  
Inserta un elemento. Se insertará tras el señalado y el nuevo pasa a ser el señalado.
- ☐ `void borra()`  
Borra el elemento señalado. El nuevo señalado pasa a ser el siguiente o `null`. Si el marcado es el elemento borrado, el marcado pasa a ser `null`.
- ☐ `void marca()`  
Marca elemento señalado en ese momento.
- ☐ `void rotaDerecha()`  
Rota a la derecha de manera que el señalado pasa a ser el anterior al señalado y el marcado no varía.
- ☐ `void rotaIzquierda()`  
Rota a la izquierda de manera que el señalado pasa a ser el siguiente al señalado y el marcado no varía.

- ☐ `void rotaHastaMarca()`  
Rota hasta la marca de manera que el señalado pasa a ser el marcado.
- ☐ `boolean esVacia()`  
Devuelve cierto si no hay elementos en el anillo y falso en otro caso.
- ☐ `boolean esMarcado()`  
Devuelve cierto si el marcado es el señalado y falso en otro caso.

**Ejercicio 5.** Construir una clase `AnilloE` que se comporte como un `Anillo` pero que disponga además de un método para girar a la derecha (`void rotaDerecha(int)`) un número dado de veces.

**Ejercicio 6.** Crear la clase `Ruleta`. Una ruleta utiliza una especie de Anillo para guardar información de 37 elementos numerados de 0 a 36. La creación de la ruleta debe ser tal que los números queden repartidos aleatoriamente por el anillo (utilizar para ello el método `random()` de la clase `Math`). Crear un constructor que inicialice adecuadamente la ruleta.

La clase `Ruleta` debe implementar además los siguientes mensajes:

- ☐ `int gira()`  
Devuelve un número de la ruleta. Para ello, toma un valor aleatorio entre 1 y 37 y gira a la derecha tantas veces como indica dicho número. El valor que quede señalado será el que se devuelva.
- ☐ `int bola()`  
Gira la ruleta (según el método anterior) y *marca* el número *señalado*. Gira nuevamente la ruleta y anota el nuevo número *señalado* (seas). Ahora, se coloca como *señalado* el que está *marcado* y se gira a la derecha tantas veces como indica el número *s*. El valor devuelto será el que se *señale* en este momento.

Realizar una aplicación que simule la ejecución del método `bola()` un determinado número de veces y muestre cuantas veces sale cada número de la ruleta.

**Ejercicio 7.** Implementar para las clases del paquete `lista` lo necesario para que puedan utilizarse a través de la interfaz `Enumeration`. Las clases del paquete deberán disponer del método `elements()` que devuelva un objeto que implemente la interfaz `Enumeration`.

**Ejercicio 8.** Construir una aplicación que tome tres números como argumentos devolviendo la suma de todos ellos. Se debe producir una excepción si no hay tres argumentos o si alguno de los argumentos no es convertible a un número.

**Ejercicio 9.** Utilizando una `Pila` de `Double` y la clase `StringTokenizer` crear una aplicación que tome como argumento una cadena que representa una expresión en formato prefijo y la evalúa devolviendo por la salida estándar su resultado. Por ejemplo:

```
java Evalua "4 5 * 5 - 2 3 + / 2 **"
```

produce como resultado 6.

NOTA: Sólo se admiten como operaciones `+`, `-`, `*`, `/`  
Se supone que en la expresión los símbolos están separados por espacios en blanco.  
Producir una excepción si algo que no es una operación tampoco es convertible a un número.