# PRÁCTICA 02

La práctica corresponde al:
***Tema 3: Estructuras de Datos Lineales***

Peso en la nota final:
**6%**

## Objetivos

Se espera que el alumno se familiarice con la implementación interna de algunas de las estructuras de datos que se incluyen en el *Java Collections Framework*, así como el contenido completo de una de las ramas de la jerarquía y las posibilidades que existen para extender la jerarquía con nuevas implementaciones.
Al igual que en anteriores prácticas se trabajará con la Genericidad de Java, que permite definir los tipos de datos a almacenar en una colección. Es una característica muy importante que mantienen todos los componentes del *Java Collections Framework*.
A partir de esta práctica en adelante, una vez que se ha trabajado el concepto de complejidad computacional en prácticas anteriores, se incorporará de manera implícita a la forma de trabajo y evaluación de las prácticas. Esto implica que se valorará como un aspecto más de las respuestas aportadas, penalizando aquellas que incluyan una complejidad excesivamente superior a la necesaria para la solución del problema.

## Conceptos Teóricos

Se pretende crear una nueva estructura de datos que nos permita almacenar elementos inicialmente para realizar luego su procesamiento en un determinado orden. En este caso se desea tener una estructura que, una vez incluidos estos elementos, permita procesarlos en el mismo orden en que se incluyeron, pero haciendo esto repetidas veces. Solo se dejará de incluir este elemento en la repetición si se elimina de forma explícita. No debe existir limitación en el número de elementos que se puedan incluir.
Este es el ejemplo del funcionamiento del algoritmo de Round Robin.

### Aproximaciones al problema

La estructura que correspondería al problema planteado es una cola. Existen varias opciones para su solución:

- Una opción podría ser emplear la clase `LinkedList`, ya incluida en el lenguaje Java. Revisando las características de la misma podemos ver que ya implementa el interfaz `Queue` y no impone restricciones de tamaño. Para poder repetir el procesamiento de los mismos elementos habría que extender su comportamiento para que al extraer un elemento y se añada por el otro extremo si no se indica lo contrario.
- **Implementar una estructura de datos propia con esas características.** Esta es la opción que se empleará en esta práctica.

## Trabajo del Alumno

# 1. Programa

Se solicita implementar una nueva clase que implemente una cola circular doblemente enlazada. Para ello:

1. Se creará una nueva clase de nombre `ColaCircularEnlazada` que extienda `AbstractQueue`.

2. Se creará una clase anidada a la primera, de nombre `Nodo`, que constituya la estructura que permitirá contener los elementos que se quieran incluir en la cola. Esta clase es la que permitirá enlazar los contenidos en la cola. Deberá contener por ello, tanto el dato a almacenar, como referencias al nodo siguiente y al nodo anterior en la cola, y si se desea sus correspondientes métodos de get y set para cada dato.
   Se debe tener en cuenta que la clase debe contener al menos una referencia al primer `Nodo` de la cola.

- Se implementarán todos los métodos necesarios para que la clase `ColaCircularEnlazada` sea una estructura funcional y modificable integrada en el framework. Esto implica que se deberá disponer de métodos para añadir, borrar y consultar. Se recuerda que, en una cola, esas operaciones solo se realizan en los extremos. Por requerimiento del *Collections Framework* también se deberá implementar una clase interna para disponer de un iterador. Este iterador deberá implementar también el método `remove()`.
- Puesto que se evaluará la eficiencia computacional de la clase, el alumno puede elegir sobreescribir el comportamiento de los métodos por defecto de la clase `AbstractQueue` de los que considere que puede mejorar su complejidad

  Para ver la definición del funcionamiento del resto de métodos, se puede revisar la documentación oficial del interfaz Queue. Se puede tomar como punto de partida la implementación de ejemplo de “Cola Enlazada” vista en la teoría de esta asignatura. La diferencia con esa es que en este caso se solicita que la cola sea circular y enlazada en los dos sentidos.

- Para completar la funcionalidad requerida, se implementará un método de nombre `iteradorCircular()` que implementará la iteración continua por los contenidos de la cola. Este iterador siempre indica que existe un elemento siguiente, a no ser que la cola esté vacía. Devolverá los elementos en el mismo orden que se insertaron (FIFO) y se diferenciará del iterador estándar en que, una vez alcanzado el último elemento, considerará el siguiente como el primer elemento insertado, continuando la iteración de forma normal.

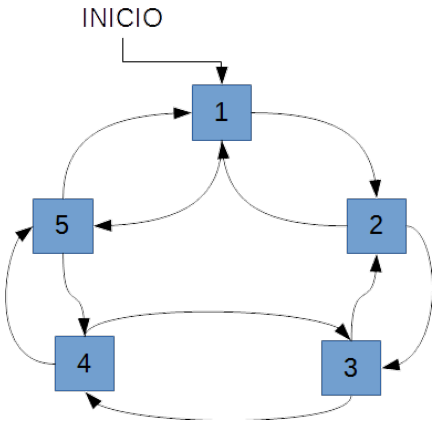


*Fig. 1: Esquema representando la cola enlazada pedida en el enunciado*

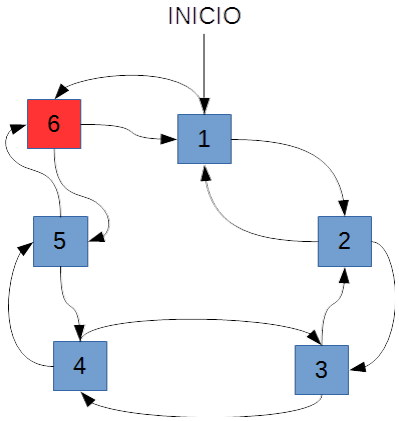


*Fig. 2:Esquema representando la inserción de un 6º elemento en la cola enlazada*

## 2. Informe

Se pide al alumno que, una vez completada la clase a programar, realice el un informe sencillo en el que se incluya un razonamiento de la complejidad algorítmica de cada uno de los métodos que se han implementado. No es necesario realizar un análisis completo y detallado de cada método (es decir, implementar las pruebas de tiempo de ejecución para el análisis de complejidad), pero sí incluir un razonamiento que relacione el código fuente que se facilita y a la complejidad que se considera que tiene el mismo.

La extensión del informe será de aproximadamente 5 páginas (incluyendo portadas, índices, gráficas...etc.).

# Condiciones de Entrega

- La práctica se realizará en grupos de 1 o 2 personas.
- La entrega se hará SOLAMENTE por medio de la plataforma UBUVirtual. Cada miembro del grupo deberá subir su propia copia de la solución. No se admitirán soluciones fuera de plazo o por otros medios de entrega (e-mail, mensajería interna, …).
- La entrega incluirá la información de quienes son los autores de la misma, tanto en los ficheros de código fuente como en los documentos asociados.
- Cada entrega consistirá en un fichero comprimido (formato .zip), con la estructura de nombre "`Apellidos1Nombre1_Apellidos2Nombre2`"
- Qué deberá incluir el fichero comprimido:
  - Código fuente de la solución (dentro de la estructura de paquetes necesaria)
  - Documentación en formato Javadoc
  - Informe en formato PDF (en aquellas prácticas que se solicite).
- No hace falta entregar ficheros binarios

# Criterios de Evaluación

- **Corrección del funcionamiento**. Se valorará como aspecto más importante que el funcionamiento de la estructura de datos implementada cumpla con todas las condiciones solicitadas, tanto en la documentación oficial como en el enunciado. Se facilitan al alumno un subconjunto de los tests que se emplearán en la corrección, pero se aconseja que el alumno realice de forma adicional los que crea conveniente.
- **Complejidad algorítmica**: Como parte fundamental de los conocimientos de la asignatura, se considerará la complejidad algorítmica de las soluciones proporcionadas el segundo aspecto más importante a valorar en los ejercicios. Penalizará en la nota aquellas soluciones que, a pesar de proporcionar salidas correctas, lo hagan con una complejidad mucho mayor de la estrictamente necesaria.
- **Informe**: se tendrá en cuenta la corrección del informe de complejidad algorítmica de la solución. No es necesario un informe excesivamente extenso, sino que se centre en la comparación de la complejidad algorítmica solicitada. La extensión aproximada es de 5 páginas.
- **Corrección del código**. Ausencia de *warnings* u otros elementos no deseables como variables no utilizadas, código que no se emplea, catch vacíos, ausencia de marcas sobre los métodos (*@Override*, @...) etc.
- **Documentación**. Las explicaciones y razonamientos que se incluyan a modo de comentarios en el código que entrega el alumno. Se solicita que el alumno entregue adicionalmente la documentación generada en formato javadoc, junto con los ficheros de código fuente.