Estructuras de Datos
Grado en Ingeniería Informática en Sistemas de Información
Enseñanzas Prácticas y de Desarrollo
EPD-5: Listas, Pilas y Colas

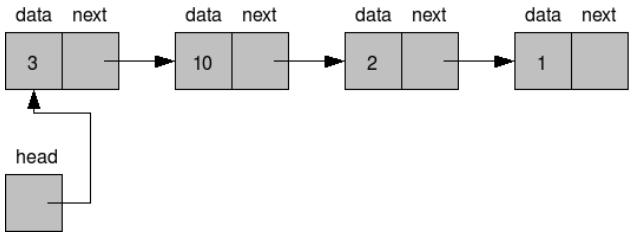
Objetivos

- Implementar una lista enlazada
- Introducir el conceptos colas
- Implementar colas y pilas basadas en listas

Conceptos

1. El Concepto de Lista Enlazada

Una lista enlazada es una estructura lineal que almacena una colección de elementos generalmente llamados *nodos*, en donde cada nodo puede almacenar datos y enlazar con otros nodos. De esta manera los nodos pueden localizarse en cualquier parte de la memoria, utilizando la referencia que lo relaciona con otro nodo dentro de la estructura. El siguiente esquema ilustra el concepto de lista enlazada:



Las listas enlazadas son estructuras dinámicas que se utilizan para almacenar datos que están cambiando constantemente.
Las listas enlazadas permiten almacenar información en posiciones de memoria que no sean contiguas; para almacenar la información contienen elementos llamados nodos. Estos nodos poseen dos campos uno para almacenar la información o valor del elemento y otro para el enlace que determina la posición del siguiente elemento o nodo de la lista.
Para poder implementar una lista, necesitamos de dos clases, una para los nodos y otra para la lista en si.
La clase nodo tendrá la siguiente estructura:

```
clase Nodo{
        Elemento element;
        Nodo next;
        //metodos setters y getters
}
```

Se puede ver como el atributo nodo es de tipo Nodo, lo cual no permite enlazar un nodo con otro. Esta será, por lo tanto, una *clase auto-referenciada,* es decir, una clase con al menos un campo cuyo tipo de referencia es el nombre de la misma clase.

Una *lista enlazada simple* es una colección de nodos que tienen una sola dirección y que en conjunto forman una estructura de datos lineal. Cada nodo es un objeto compuesto que guarda una referencia a un elemento (dato) y una referencia a otro nodo (dirección).
La referencia que guarda un nodo a otro nodo se puede considerar un *enlace* o un *puntero* hacia el segundo nodo y el salto que los relaciona recibe el nombre de *salto de enlace* o *salto de puntero*.

El primer nodo de una lista recibe el nombre de ***cabeza*** y el último es llamado ***cola*** (es el único nodo con la referencia a otro objeto como nula).

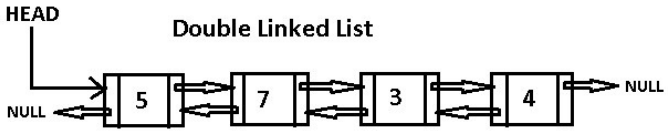
Un nodo de una lista enlazada simple puede determinar quien se encuentra después de él pero no puede determinar quien se encuentra antes, ya que solo cuenta con la dirección del nodo siguiente pero no del anterior.

Para poder implementar una lista, una clase deberá tener como atributo por lo menos una referencia a la cabeza de la lista, y es recomendable guardar referencia a la cola de la lista también. Esto ser hará a través de dos atributos de tipo nodo. Otro atributo que es recomendable incluir es un entero que guarde el tamaño de la lista.

Cualquiera sea la posición en la lista, los punteros **inicio** y **fin** apuntan siempre al primer y último elemento. El campo *tamaño* contendrá al numero de elementos de la lista cualquiera sea la operación efectuada sobre la lista.
Así que el esquema de una clase para implementar una lista será:

```
clase Lista {
        Nodo inicio; //cabeza de la lista
        Nodo fin; //cola de la lista
        int tamanho; //tamaño de la lista
        //metodos para la gestion de la lista
}
```

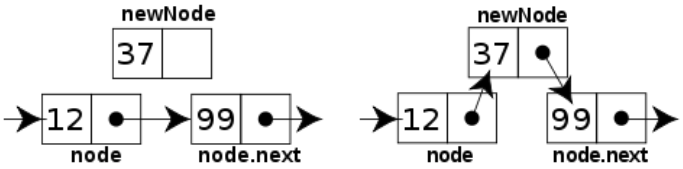
Una lista doblemente enlazada, o doble, es una lista en al cual los nodos enlazan tanto con el nodo anterior como el siguiente en la lista, como en el siguiente esquema:



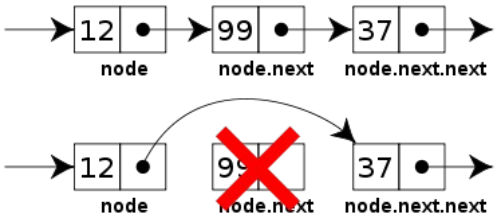
Las operaciones básicas que deberá proporcionar una lista son la inserción, el borrado y devolver el elemento en una posición especifica. Recuerde que por defecto los elementos se añaden al final de la lista, pero existe la posibilidad de añadir un elemento especificando su posición en la lista.
En estas operaciones deberemos de poner cuidado en no perder ninguna referencias a los elementos de la lista. Por ejemplos, los pasos a seguir para insertar un elemento al principio de la lista son:

1. Crear un nuevo nodo
2. Insertar el nuevo elemento en el nodo creado en el paso 1
3. Hacer que el nuevo nodo apunte a la antigua cabeza de la lista
4. Actualizar el atributo “inicio” para que apunte al nuevo nodo

Si, por ejemplo, no ejecutamos el paso 4, el nuevo nodo no será insertado en la lista, que que el atributo inicio seguiría apuntando, erróneamente, al segundo nodo, e de esta forma no sería posible acceder al nuevo nodo. El siguiente esquema ilustra como insertar un nodo en una posición determinada de la lista:



Y el siguiente esquema ilustra como borrar un nodo de una lista:



Recuerdo que la inserción y borrado en primera y última posición son casos especiales, y se tienen que implementar en métodos específicos.

2. Implementación de una Pila basada en Listas

Tal y como se ha expuesto en las clases de EB, es posible implementar una pila utilizando un array. Hay que destacar que esta implementación tiene un defecto: la pila tiene una capacidad limitada.
Para obviar a esta limitación, se puede utilizar una lista enlazada, debiendo implementar una clase auxiliar que modela la lista que se va a utilizar. De esta forma la pila tendrá una estructura parecida a la que se muestra en la figura 1.

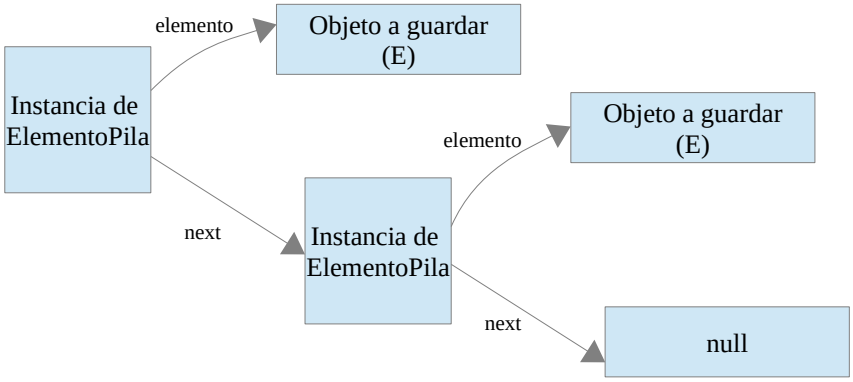


Figura 1: Estructura de lista enlazada.

En la figura se muestra una pila de dos elementos. Nótese que el atributo next del último elemento de una pila apunta a null.
De esta forma, la pila tendría la siguiente estructura:

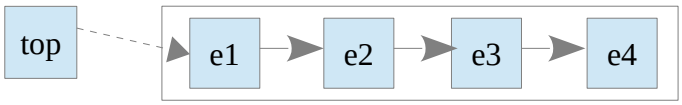


Figura 2: Ejemplo de pila implementada con una lista enlazada.

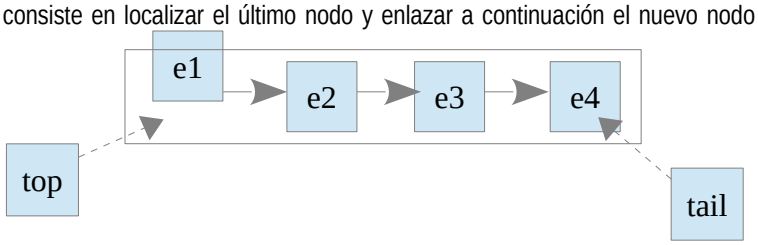
La implementación de la pila tendrá que tener por lo menos un atributo, *top* que apunta al primer elemento de la pila.

Hay que subrayar que la pila debe permitir almacenar cualquier tipo de objeto, e implementar por lo menos los métodos del ADT de pila visto en clase de EB.

3. El Concepto de Cola

Una cola (queue) es una estructura de datos similar a la pila, pero donde la inserción y eliminación de los elementos se efectúa según el principio FIFO (First In First Out). Es decir, el primer elemento en salir es el primero que entró en la cola.

Como en el caso de la pila, la implementación basada en array de una cola tiene el defecto de establecer el número máximo de elementos que la estructura puede contener. En esta práctica, implementaremos una cola con una lista enlazada.
Como en el caso de la pila, la estructura que definiremos tendrá un atributo *top*, que apunta a la cabeza de la cola, que será el elemento más antiguo. En el caso de la cola, tendremos que mantener otro atributo *tail* que apunte al otro extremo de la cola, es decir que representa la última posición de la cola (el siguiente nodo de tail es null).
Por tanto, la extracción de la cola consiste en devolver el objeto asociado al nodo de la cabeza y desenlazarlo. Por el contrario, la inserción de un nuevo objeto consiste en localizar el último nodo y enlazar a continuación el nuevo nodo con dicho objeto. Para



Set y SortedSetFigura 3: Ejemplo de cola implementada con una lista enlazada.

evitar tener que recorrer la lista siempre para encontrar el último nodo, se mantiene el atributo tail que lo referencia.
Como en el caso de la pila, los nodos en una lista enlazada, se enlazan unos con otros usando un atributo next.

De esta forma, la estructura de la cola se establecerá por medio de dos atributos; uno top, que apunta a la cabeza de la cola, por donde se extraen los elementos; y otro, tail, por donde se introducen los elementos. De esta manera, se cumple la estructura FIFO. En la figura 3 se puede apreciar la estructura que tendrá una cola implementada con una lista enlazada. Los atributos top y tail apuntan a los extremos de la cola.

Bibliografía Básica

- Fundamentos de programación: Algoritmos, Estructuras de datos y objetos. L. Joyanes. MacGraw-Hill, 2003, pag. 465-482.
- http://opendatastructures.org/ods-java/3_Linked_Lists.html
- http://opendatastructures.org/ods-java/1_2_Interfaces.html#666
- http://opendatastructures.org/ods-java/1_2_Interfaces.html#SECTION00421000000000000000
- https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm

Experimentos

E1. Analice el siguiente código, correspondiente a una interfaz para una lista:

```
public interface ILinkedList<E> {

    void add(E newElement, int index) throws IndexOutOfBoundsException;
    void addEnd(E newElement);
    void addFront(E newElement);
    E getElement(int index) throws IndexOutOfBoundsException;
    boolean isEmpty();
    void remove(int index) throws IndexOutOfBoundsException;
    void removeEnd();
    void removeFront();
    int size();
    @Override
    String toString();
}
```

¿Por qué se puede lanzar la excepción IndexOutOfBoundsException?

E2. El siguiente fragmento de código correponde al método para añadir un elemento al principio de una lista

```
public void addFront(E newElement) {
        if (newElement != null) {
            Node<E> newNode = new Node<E>();
            newNode.setElem(newElement);
            if (firstNode != null) {
                newNode.setNext(firstNode);
                firstNode = newNode;
            } else {
                newNode.setNext(null);
                firstNode = newNode;
                lastNode = newNode;
            }
            this.size++;
        }
    }
```

¿Cómo se logra que el nuevo nodo se inserte al principio de la lista?

E3. Analice el siguiente fragmento de código correspondiente al borrado del primer elemento de una lista:

```
public void removeFront() {
        if (firstNode != null) {
            Node<E> oldTop = firstNode;
            firstNode = firstNode.getNext();
            oldTop.setNext(null);
            this.size--;
        }
    }
```

¿En qué línea del código se borra el primer nodo?

E4. Analice el siguiente código correspondiente a la interfaz IPila:

```
public interface Pila<E> {
    boolean isEmpty();
    E pop() throws PilaVaciaException;
    void push(E o);
    int size();
    String toString();
    E top() throws PilaVaciaException;
}
```

¿Por qué se pueden lanzar las dos excepciones? ¿Por qué, a contrario de lo que se ha visto en clase de teoría, el método push no lanza ninguna excepción?

E5. Analice el siguiente código correspondiente a la interfaz Iqueue:

```
public interface IQueue<E> {
    boolean buscar(E o);
    E dequeue() throws EmptyQueueException;
    void enqueue(E o);
    E front() throws EmptyQueueException;
    boolean isEmpty();
    int size();
}
```

¿Por qué se pueden lanzar las dos excepciones? ¿Cómo se podría implementar el método buscar?

Ejercicios

EJ1. Defina la excepción IndexOutOfBoundsException. Una forma de definir una excepción puede ser la siguiente:

```
public class MyException extends Exception{

    public MyException(String exc)
    {
        super(exc);
    }
    public String getMessage()
    {
        return super.getMessage();
    }
}
```

EJ2. Implemente una clase LinkedList que implemente una lista simple enlazada. La clase tendrá que implementar la siguiente interfaz:

```
public interface ILinkedList<E> {
    //añade un elemento en posición index, desplaza los siguientes
    void add(E newElement, int index) throws IndexOutOfBoundsException;
    //añade un elemento al final
    void addEnd(E newElement);
    //añade un elemento al principio
    void addFront(E newElement);
    //devuelve el elemento en posición index
    E getElement(int index) throws IndexOutOfBoundsException;
    boolean isEmpty();
    //borra elemento en posición index
    void remove(int index) throws IndexOutOfBoundsException;
    //borra último elemento
    void removeEnd();
    //borra primer elemento
    void removeFront();
    int size();
    String toString();
}
```

EJ3. Implemente la clase PilaEnlazada que implemente la interfaz vista en el experimento 4. La pila tendrá que ser implementada por medio de una lista enlazada.

EJ4. Cree un programa que demuestre el funcionamiento de los elementos desarrollados en los ejercicios 3 y 4.

EJ5. Implemente la clase LinkedQueue que implemente la interfaz IQueue vista en el experimento 5. La cola tendrá que ser implementada por medio de una lista enlazada.

EJ6. Cree un programa que demuestre el funcionamiento de los elementos desarrollados en el ejercicios 5.

Problemas

P1. Implemente una lista doblemente enlazada. La clase que implementará la lista se llamará DoubleLinkedList y deberá implementar la siguiente interfaz:

```
public interface IDoubleLinkedList<E> {
    //añade un elemento en posición index, desplaza los siguientes
    void add(E newElement, int index) throws IndexOutOfBoundsException;
    //añade un elemento al final
    void addEnd(E newElement);
    //añade un elemento al principio
    void addFront(E newElement);
    //devuelve el elemento en posición index
    E getElement(int index) throws IndexOutOfBoundsException;
    boolean isEmpty();
    //borra elemento en posición index
    void remove(int index) throws IndexOutOfBoundsException;
    //borra último elemento
    void removeEnd();
    //borra primer elemento
    void removeFront();
    int size();
     //imprime la lista desde el principio
    String toString();
    //imprime la lista desde el final
    String printFromLast();
    //busca un elemento
    boolean find(E element);
}
```

P2. Implementar la clase Pila que implemente la siguiente interfaz:

```
public interface IStack<E> {

    /**
     * Return the number of elements in the stack.
     * @return number of elements in the stack.
     */
    public int size();

    /**
     * Return whether the stack is empty.
     * @return true if the stack is empty, false otherwise.
     */
    public boolean isEmpty();

    /**
     * Inspect the element at the top of the stack.
     * @return top element in the stack.
     * @exception EmptyStackException if the stack is empty.
     */
    public E top()
            throws EmptyStackException;

    /**
     * Insert an element at the top of the stack.
     * @param element to be inserted.
     */
    public void push(E element) throws FullStackException;
```

```
    /**
     * Remove the top element from the stack.
     * @return element removed.
     * @exception EmptyStackException if the stack is empty.
     */
    public E pop()
            throws EmptyStackException;
}
```

La pila se implementará **utilizando un array** de tamaño 1000.
La excepción EmptyStackException es la misma que la implementada en el ejercicio 1, la excepción FullStackException deberá ser implementada. Esta última excepción deberá ser lanzada al intentar un elemento en una pila llena.
Cree un programa que demuestre el funcionamiento de la clase implementada.

P3. Implementar la clase Cola, que implemente la interfaz del ejercicio 5 utilizando un array de tamaño 1000. Como en el problema precedente, para el array se puede utilizar la clase Vector. En este caso, como en el problema 1, si se intenta insertar un elemento en una cola llena, se tendrá que lanzar una excepción.
Cree un programa que demuestre el funcionamiento de la clase implementada.

P4. Añade un método a la clase LinkedQueue que devuelva el objeto colocado en antepenúltimo lugar de la cola, sin extraerlo. Debe devolver null si la cola contiene dos o menos elementos. Comprueba que funcione correctamente.

P5. Añade un método a la clase LinkedQueue que reciba como parámetro un objeto y devuelva un entero indicando su posición en la cola (-1 si no está, 1 el primero, 2 el segundo, etc.) ¿Qué diferencia existe entre compararlos con equals o utilizando las referencias? Comprueba que funcione correctamente.

P6. Añade un método a la clase LinkedQueue que reciba como parámetro una referencia a un objeto QueueNode y un entero (void insertar(QueueNode sublista, int posicion)). El método debe insertar la lista que comienza con sublista justo a continuación del nodo cuya posición indique posicion. Por ejemplo, si en la lista [a-b-c-d-e] se desea insertar la sublista [x,y,z] en la posición 3, la lista resultante será [a-b-c-x-y-z-d-e]. Se seguirá el siguiente convenio:

1. Si posicion <=0, inserta la sublista justo al principio.
2. Si posicion >= num. de elementos, inserta la sublista justo al final.

Comprueba que funcione correctamente.

Ampliación de Bibliografía

- http://www.studytonight.com/data-structures/introduction-to-linked-list
- https://www.tutorialspoint.com/data_structures_algorithms/doubly_linked_list_algorithm.htm
- http://opendatastructures.org/ods-java/3_2_DLList_Doubly_Linked_Li.html
- Arrays https://docs.oracle.com/javase/6/docs/api/java/util/Arrays.html
- Vector https://docs.oracle.com/javase/7/docs/api/java/util/Vector.html
- http://www.cs.cmu.edu/~adamchik/15-121/lectures/Stacks%20and%20Queues/Stacks%20and%20Queues.html
- BufferedReader https://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html
- BufferedReader http://www.java2s.com/Tutorial/Java/0180__File/CreateBufferedReaderfromFileReader.htm