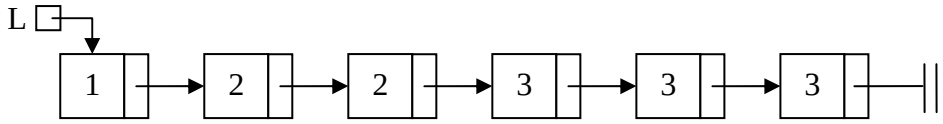
ESTRUCTURA DE DATOS Y ALGORITMOS **5-Febrero-2004**
Titulación: Ingeniero Técnico en Informática de Gestión **Curso: 2º**

1. Ejercicio (3 puntos)

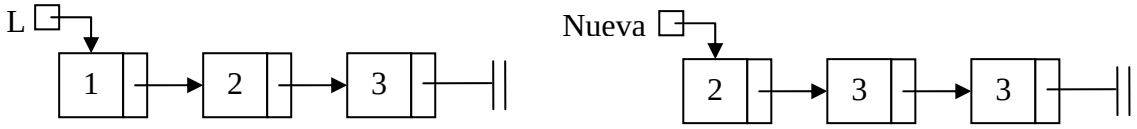
Diseñar y escribir en Java un subprograma que tenga como argumento de entrada una lista *L*, de la que se sabe que tiene nodos repetidos y estos están juntos. El subprograma creará otra lista cuyos nodos contendrán la información repetida en la lista *L y* eliminará dichos nodos de la lista *L*. También se requiere:

- Definir los tipos de datos para representar las listas.
- Especificar los métodos que se vayan a utilizar.
- Implementar los métodos relacionados con la eliminación, en caso de su uso.

Por ejemplo:



Después de ejecutar el subprograma:



1.1. Definir los tipos de datos:

```
class Node {
   Object element;
   Node next;
}
public class LinkedList{
   Node top
}
```

```
public class LinkedListItr {
   LinkedList theList;
   Node actual;
   Node anterior;
}
```

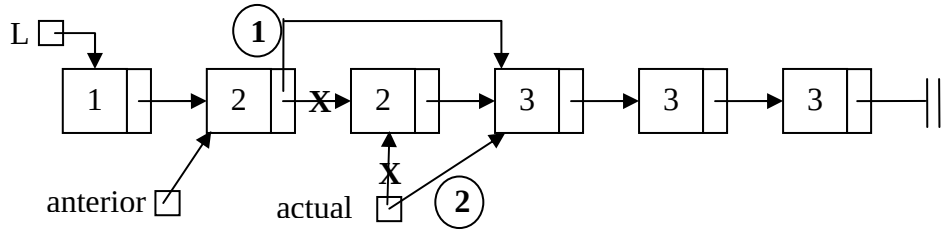
1.2. Especificar los métodos que se van a utilizar

public **LinkedList**(); //Construye una lista enlazada vacia
public **LinkedListItr**(LinkedList l);
// Construye una clase iteradora y le asocia una lista enlazada
public void **advance**();
// avanza en la lista con la variable actual, pasando al siguiente nodo del nodo actual
public boolean **isOnList**();
// verifica si la variable actual referencia un nodo de la lista, en caso afirmativo devuelve true
// y en caso contrario false
public void **first**();
//actualiza la variable actual, el cuál referencia el primer nodo de la lista
public Object **getActual**();
// devuelve el objeto que esta referenciado en el nodo que referencia actual
public Object **getAnterior**(); // igual que el anterior, pero el que referencia la variable anterior
public void **removeActual**();
// borra el nodo que es referenciado por actual de la lista y actual queda referenciando el
// siguiente nodo
public void **insert**(Object obj);
// inserta un nuevo nodo con la información obj en la lista, después del nodo que referencia la
// variable actual

1.3. Diseño e implementación:

```
Diseño quitarRepetidos
      Crear lista vacia(nueva)
      Situarse en el primer elemento de la lista L
      Mientras recorro la lista L // o no haya visitado todos los nodos hacer
        Avanzar en la lista L
        objAnterior = recuperar la información del nodo anterior de la lista L
        objActual = recuperar la información del nodo actual de la lista L
        mientras objAnterior y objActual sean iguales hacer
            insertar en la lista nueva la información objActual
            (1.1)eliminar de la lista L el nodo que referencia actual
            recuperarActual(actual)
        fin mientras
      fin mientras

Diseño 1.1 eliminarActual (método de la clase iteradora)
      // en la clase iteradora mantendremos dos variables anterior y actual
    hacer 1. que el siguiente del nodo anterior sea el siguiente del nodo actual
    hacer 2. que la variable actual referencie un nodo de la lista, exactamente
            el siguiente nodo de anterior
```



Implementación:

```
public void eliminarActual()
{
   anterior.next = actual.next;
   actual = anterior.next;
}

public static LinkedListItr quitarRepetidos(LinkedListItr itrLista)
{
   LinkedList listaRepetidos = new LinkedList();
   LinkedListItr itrListaRepetidos = new LinkedListItr (listaRepetidos);

   itrLista.first();
   while (itrLista.isOnList())
   {
     itrLista.avanzar();
     Object objAnterior = itrLista.getAnterior();
     Object objActual = itrLista.getActual();
     while (itrLista.isOnList() && objAnterior.equals(objActual))
     {
       itrListaRepetidos.insert(objActual);
       itrLista.removeActual();
       objActual = itrLista.getActual();
     }
   }

   return itrListaRepetidos;
}
```

2. Ejercicio. (3 puntos)

1. **Especificar / Parametrizar**
   Entrada: A, un árbol binario de enteros
   Salida: Vacía
   Post: Se ha escrito en pantalla los pesos de los nodos izquierdos y derechos.

   public void pesosIzquierdosYDerechos();

   A la hora de tratar un nodo debemos conocer si es nodo izquierdo o derecho → INMERSION
   Entrada: A, un árbol binario de enteros,
   esIzq, true si el nodo raíz de A es nodo izquierdo y false, en otro caso
   Salida: peso de los nodos izquierdos y peso de los nodos derechos →
   Se necesita una clase Contexto para poder devolver dos valores

```
public class Contexto {
        int pesoIzq;
        int pesoDch;
}
```

   public Contexto pesosIzquierdosYDerechos(BTNode A, boolean esIzq)

2. **Diseño**
   Casos triviales
   Si Vacío(A) → devolver new Contexto(0, 0)
   Si Hoja(A) → **Si** esIzquierdo **entonces** devolver new Contexto(Raíz(A),0)
   si no devolver new Contexto(0,Raíz(A))
   Caso general
   Si no Vacío(A) y no Hoja(A) →

```
contextoIzq = pesosIzquierdosYDerechos(Rama_izq(A), true);
contextoDch = pesosIzquierdosYDerechos(Rama_dch(A), false);
contexto = new Contexto(contextoIzq.pesoIzq + contextoDch.pesoIzq,
                        contextoIzq.pesoDch + contextoDch.pesoDch);
Si esIzquierdo entonces
    contexto.pesoIzq += Raíz(A);
si no
    contexto.pesoDch += Raíz(A);
fin si
devolver contexto;
```

3. Implementación

```
public void pesosIzquierdosYDerechos() {
   Contexto contexto = pesosIzquierdosYDerechos(bTree.root, true);
   System.out.println("Peso de nodos izquierdos:",contexto.getPesoIzq());
   System.out.println("Peso de nodos derechos:",contexto.getPesoDch());
}

private Contexto pesosIzquierdosYDerechos(BTNode nodo, boolean esIzq){
   if ( nodo == null) return new Contexto(0, 0);
   else {
      int raiz = ((Integer) nodo.content).intValue();

      if (nodo.left==null && nodo.right==null){
        if (esIzq)
            return new Contexto(raiz, 0);
        else
            return new Contexto(0, raiz);
      }
      else {
        Contexto contextoIzq = pesosIzquierdosYDerechos(nodo.left, true);
        Contexto contextoDch = pesosIzquierdosYDerechos(nodo.right, false);
        Contexto contexto =
            new Contexto(contextoIzq.getPesoIzq() + contextoDch.getPesoIzq(),
                         contextoIzq.getPesoDch() + contextoDch.getPesoDch());
        if(esIzq) contexto.sumaPesoIzq(raiz);
        else contexto.sumaPesoDch(raiz);

        return contexto;
     }
  }
}

public class Contexto{
   int pesoIzq;
   int pesoDch;

   public Contexto(int pesoIzq, int pesoDch){
      this.pesoIzq = pesoIzq;
      this.pesoDch = pesoDch;
   }

   public int getPesoIzq(){
      return pesoIzq;
   }

   public int getPesoDch(){
      return pesoDch;
   }

   public void sumaPesoIzq(int peso){
      pesoIzq += peso;
   }

   public void sumaPesoDch(int peso){
      pesoDch += peso;
   }
}
```

3. Ejercicio (4 puntos)

1. Definir en Java una representación adecuada para la clase **ColaConPrioridad** basada en el tipo de **cola**, de tal forma que todas las operaciones anteriormente especificadas sean de O(1). (Podéis suponer que las operaciones del tipo de cola son de orden constante y que el número de prioridades también es constante).

```
public class colaConPrioridad {
    // Una cola de n prioridades se puede implementar como un array de n
    // posiciones donde en cada posición se almacena una cola
    public final int MAX_PRIORIDAD = 5;
    Queue[] colaPrioridad = new Queue[MAX_PRIORIDAD+1];
}
```

2. Implementar las operaciones:
   `cargarDatos`, `primeroEnMinuto` y `borrarPrimeroEnMinuto`
   según la representación elegida.

   A) **Diseño** cargarDatos

```
abrir fichero
mientras haya lineas hacer
    datos = leerDatos();
    trabajo = crearTrabajo(datos)
    introducir en la cola de prioridad el trabajo, según su prioridad
fin mientras
```

   Implementación*:*

```
public void cargarDatos(String f){
    Token fichero = new Token(f);
    while (fichero.hasMoreTokens()){
       Datos datos = fichero.getToken();
       Trabajo trabajo = new Trabajo(datos.getIdTrabajo, datos.getMinuto);
       insertarElemento(trabajo, datos.getPrioridad());
    }
}
```

   B) **Diseño** primeroEnMinuto(m)

```
si (cola de impresión es vacia) entonces devolver null;
sino
    numCola = (2.1)buscarColaPrimeroEnMinuto(m);
    si (numCola != 0) entonces
            devolver primer elemento de la cola de prioridad numCola
    sino devolver null
```

   Diseño 2.1 buscarColaPrimeroEnMinuto(m)

```
P=1; encontrado = false;
Mientras (no encontrado y P<=5 hacer
    si (no es vacía la cola de prioridad P) entonces
            Obtener primero de la cola de prioridad P;
            si (minuto de llegada <= m) entonces
                    encontrado = true;
            sino incrementar P;
    sino incrementar P
Fim mientras;
Si (encontrado) entonces devolver P sino devolver 0;
```

Implementación:

```
public Trabajo primeroEnMinuto(int m){
    if (isEmpty()) return null;
    int pos = (2,1) buscarColaPrimeroEnMinuto(m);
    if (pos!=0)
            return (Trabajo) colaPrioridad[pos].front();
    else return null;
}
public int buscarColaPrimeroEnMinuto(int m){
    int p=1;
    boolean encontrado = false;
    Trabajo trabajo;
    while (!encontrado && p < MAX_PRIORIDAD+1){
       if ( !colaPrioridad[p].isEmpty()){
            trabajo = (Trabajo) colaPrioridad[p].front();
            if (trabajo.getMinuto() <= m)
                    encontrado = true;
            else
                    p++;
       }
       else p++;
    }
    if (encontrado) return p;
    else return 0;
}
```

C) **Diseño** borrarPrimeroEnMinuto(m)

```
si (no es vacia la cola de impresión) entonces
    numCola = (2.1) buscarColaPrimeroEnMinuto(m);
    si ( numCola !=0)
            eliminar elemento cabecera de la cola numCola;
```

Implementación:

```
public void borrarPrimeroEnMinuto(int m){
    int pos = buscarColaPrimeroEnMinuto(m);
    if ( pos !=0)
            colaPrioridad[pos].pop();
}
```

3. A partir del fichero **colaTrabajos.txt** dónde cada línea contiene:

Diseñar e implementar un subprograma que escriba en pantalla, los números de trabajo y sus respectivos minutos de impresión. Dicha relación deberá estar ordenada según el minuto de impresión.

Ejemplo:

Para el siguiente fichero:

10	1	6
40	5	8
33	3	11
55	4	13
77	4	20
86	4	21
99	5	22
32	3	23

Obtendríamos en la pantalla:

TRABAJO	MINUTO IMPRESION
10	6
40	9
33	12
55	15
77	20
32	23
86	26
99	29

***Especificación*:**

Entrada: Un fichero de texto, por linea (trabajo, prioridad, minuto de llegada)
Salida: Listado en pantalla de pares (trabajo, minuto de impresión)
Pre: El fichero está ordenado por minuto de llegada
Post: El listado de salida está ordenado por minuto de impresión

***Diseño**:*

```
Inicializar minutoDeImpresión(tiempo) a 0
Inicializar cola de impresión
Guardar en la cola de impresión todos los trabajos del fichero
Si (no vacia la cola de impresión (ci)) entonces
        Obtener primero en llegar
        Se imprime en cuanto llegué(escribir en pantalla)
        Borrar primero en llegar
        Actualizar tiempo ← minuto que llegó a la cola + 3
Fin si
Mientras (no vacia la cola de impresión (ci)) hacer
        Obtener el de mayor prioridad de los que llegaron hasta m
        Si (hay trabajo) entonces
                Trabajo se imprime en minuto tiempo (escribir en pantalla)
                Borrarlo de la cola de impresión
                Actualizar tiempo ← tiempo + 3
        Si no
                Actualizar tiempo ← tiempo + 1
        Fin si
Fin mientras
```

***Implementación*:**

```
public static void main(String[] args){
        String path = "C:\\Trabajos\\datos\\);
        int tiempo = 0;
        Trabajo trabajo;
        ColaConPrioridad colaImpresion = new ColaConPrioridad();
        colaImpresion.cargarDatos(path+"datos.txt");
        System.out.println("Trabajo \t Minuto Impresión");
        if (!colaImpresion.esVacia()){
                trabajo = colaImpresión.primero();
                System.out.println(trabajo.getIdTrabajo+"\t"+trabajo.getMinuto());
                colaImpresion.borrarElemento();
                tiempo = trabajo.getMinuto() + 3;
        }
        while (!colaImpresion.esVacia()){
                trabajo = colaImpresion.primeroEnMinuto(tiempo);
                if ( trabajo != null){
                        System.out.println(trabajo.getIdTrabajo+"\t"+tiempo);
                        colaImpresion.borrarPrimeroEnMinuto(tiempo);
                        tiempo += 3;
                }
                else
                        tiempo ++;
        }
}
```