ESTRUCTURAS DE DATOS
CURSO 2018-19.
Práctica 2: Listas.

1. OBJETIVOS:

- Construir un TAD lista ordinal para objetos *Paciente* a partir del TAD lista ordinal para enteros visto en clase.
- Ampliar el TAD lista ordinal de pacientes con algún nuevo método.
- Utilizar el TAD lista ordinal de pacientes en diferentes aplicaciones.
- Definir un TAD cola de prioridades para objetos *Paciente*.
- Utilizar para el TAD cola de prioridades una estructura de datos compleja: una lista de colas.

2. Definición del TAD Lista Ordinal para Pacientes.

2.1. Construcción del proyecto *Practica2ListaPacientes*.

Construya un nuevo proyecto en *IntelliJ*, con el nombre *Practica2ListaPacientes*. En el directorio *src* de dicho proyecto, incluya los archivos obtenidos al descomprimir el archivo *Practica2ListaPacientes* de la plataforma *Moodle*. Eche un primer vistazo al contenido de las clases:

- Clase ***Paciente*** en la que se definen los datos de un paciente en un hospital, junto con los métodos necesarios para manipular dichos datos: *getNombre*, *setNombre*, *getSintomas*, *setSintomas*, *darAlta* (para dar de alta a un paciente), *estaAlta* (para saber si un paciente está de alta) y *verPaciente* (para visualizar los datos de un paciente).
- Clase ***ListaOrdinal*** en la que se define el TAD lista ordinal para números enteros, implementada con una lista enlazada de una forma similar a como se ha visto en clase.
- Clase ***Nodo*** para definir los nodos que constituyen la lista enlazada con la que se implementa el TAD lista ordinal.
- Clase ***Iterador*** para definir iteradores sobre la lista ordinal.

2.2. Definición de un TAD lista ordinal de objetos *Paciente*.

Realizar las modificaciones oportunas en las clases *Nodo*, *ListaOrdinal* e *Iterador* para que el TAD lista ordinal de enteros se transforme en un TAD lista ordinal de objetos *Paciente*.

A continuación, se probará su correcto funcionamiento en el *main* de la clase *Aplicación*. Con este objetivo, se incluirá código para realizar las siguientes pruebas:

- Crear cuatro objetos *Paciente* con los siguientes contenidos:
    - Alberto (nombre), Frecuentes mareos (síntomas).

- Ana (nombre), Infección resistente (síntomas).
- Eva (nombre), Problemas digestivos (síntomas).
- Ernesto (nombre), Problemas cardiovasculares (síntomas).

- Insertarlos en el mismo orden en una lista ordinal de *Pacientes*.
- Dar de alta a los pacientes Ana y Eva.
- Visualizar el contenido de la lista. Para ello, se deberá utilizar un *iterador*.

El resultado de la ejecución será el siguiente:

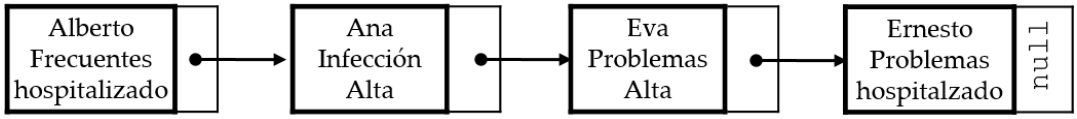
PACIENTES DEL HOSPITAL:
Nombre: Alberto. Síntomas: Frecuentes mareos. Estado: hospitalizado
Nombre: Ana. Síntomas: Infección resistente. Estado: alta
Nombre: Eva. Síntomas: Problemas digestivos. Estado: alta
Nombre: Ernesto. Síntomas: Problemas cardiovasculares. Estado: hospitalizado
---

2.3. Ampliación del TAD Lista ordinal de Pacientes.

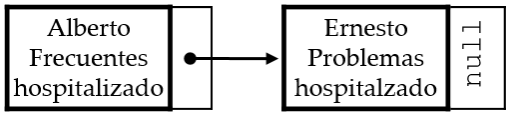
En este apartado, se definirá un nuevo método en la clase *ListaOrdinal*:

```
public void borrarAltas()
```

Este método recorrerá la lista enlazada de pacientes para eliminar todos aquellos que se encuentren en estado de alta hospitalaria. Por ejemplo, si se ejecuta este método sobre la siguiente lista de pacientes:



La lista pasa a contener:



Una vez realizado el método, se incluirá código en el *main* de la clase *Aplicación* para comprobar su correcto funcionamiento:

- Utilizar el método *borrarAltas* sobre la lista.
- Visualizar el contenido de la lista, utilizando un iterador.

El resultado de las nuevas pruebas será el siguiente:

DESPUÉS DE BORRAR LAS ALTAS:
Nombre: Alberto. Síntomas: Frecuentes mareos. Estado: hospitalizado
Nombre: Ernesto. Síntomas: Problemas cardiovasculares. Estado: hospitalizado
---

2.4. Utilización del TAD Lista ordinal de Pacientes.

Se desea realizar una aplicación que utiliza una lista de pacientes y que necesita visualizar los datos de los pacientes con alta hospitalaria. Para ello, se definirá una función en la clase *Aplicacion* que reciba como parámetro una lista de pacientes y que realice dicha visualización.

```
... void verAltas(ListaOrdinal lista)
```

A continuación, se comprobará el correcto funcionamiento de esta función en el *main*:

- Volver a insertar en la lista a los pacientes Ana y Eva.
- Ejecutar la función *verAltas* pasándole como parámetro la lista de pacientes anterior.

El resultado de las nuevas pruebas será el siguiente:

> PACIENTES DE ALTA:
> Nombre: Ana. Síntomas: Infección resistente. Estado: alta
> Nombre: Eva. Síntomas: Problemas digestivos. Estado: alta
> ----------

Ahora se desea realizar una aplicación que necesita buscar todos los pacientes cuyos síntomas encajen con una cadena de caracteres que se pase como parámetro. Para ello, se definirá una función en la clase *Aplicación* que reciba como parámetro una lista de pacientes y una cadena de caracteres, y devuelva otra lista de pacientes conteniendo todos los pacientes cuyo síntoma encaje con el *String* recibido.

```
... ListaOrdinal pacientesSintoma(ListaOrdinal lista, String sintoma)
```

Basta con que el *String* que se envía como parámetro se encuentre dentro de los síntomas de un paciente para que sea considerado. Por ejemplo, si se envía como parámetro "mareo", el paciente Alberto irá en la lista devuelta, ya que sus síntomas son "Frecuentes mareos". Para conseguir este comportamiento, se recomienda utilizar el método *indexOf(String)* de la clase *String* (consultar dicho método en el API de la clase *String*).

A continuación, se comprobará el correcto funcionamiento de esta función incluyendo código en el *main* de la clase *Aplicación*:

- Ejecutar la función *pacientesSintoma* pasándole la lista de pacientes anterior y la cadena "mareo".
- Visualizar el contenido de la lista devuelta utilizando un iterador.

El resultado de las nuevas pruebas será el siguiente:

> PACIENTES CON MAREO:
> Nombre: Alberto. Síntomas: Frecuentes mareos. Estado: hospitalizado
> ----------

3. Definición del TAD Cola de Prioridades para Pacientes.

En el servicio de urgencias de un hospital atienden a los enfermos en base a dos cuestiones:

- El nivel de urgencia de cada uno de ellos. Un enfermo con mucha urgencia es atendido antes que otro con poca urgencia, por mucho que este último lleve esperando.
- Si dos enfermos tienen el mismo nivel de urgencia, se atiende primero al que haya llegado en primer lugar.

Se desea informatizar el proceso de atención a los pacientes mediante una aplicación. La idea es que cuando un paciente llega, es atendido por un médico que determina el nivel de urgencia de dicho paciente (prioridad) y lo ingresa en la cola de espera con dicha prioridad.

La estructura de datos que debe utilizar esta aplicación es una cola de prioridades, en la que un paciente se encola detrás de todos los que tienen su misma prioridad, y en la que los pacientes que tienen más prioridad van antes. Por ejemplo, sean los siguientes pacientes, que llegan en este orden al servicio de urgencias:

Alberto (4)
Ana (2)
Eva (2)
Felipe (1)
Ernesto (4)
Sonia (2)

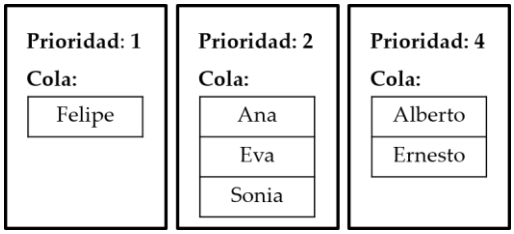
Cuando llegan se les asocia un nivel de prioridad, que es el que figura entre paréntesis. Suponemos que el nivel máximo de prioridad es el 1, y que cuanto mayor es el número que expresa la prioridad, menor urgencia tiene.

Los pacientes serían encolados con el siguiente orden de izquierda a derecha:

Felipe(1) Ana(2) Eva(2) Sonia(2) Alberto(4) Ernesto(4)

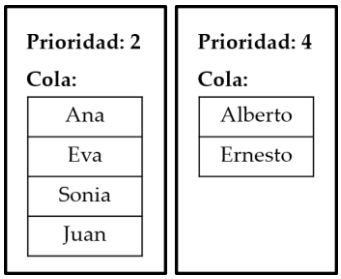
Con lo que Felipe sería el primer paciente en ser atendido, luego Ana, etc.

Para implementar una cola de prioridades, se propone utilizar una lista de colas, es decir, una lista en la que cada elemento contiene una cola con todos los pacientes de una misma prioridad. De manera que el ejemplo anterior quedaría como sigue en nuestra estructura de datos:

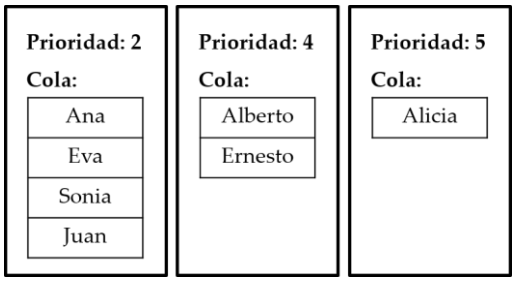


Si ahora se atiende a un paciente, sería Felipe, que sería desencolado de la primera de las colas. Además, esta cola se vacía y desaparece, quedando la estructura así:

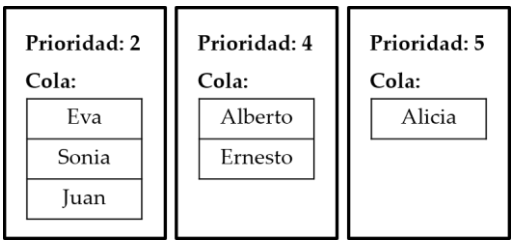
Si ahora llega un nuevo paciente al servicio de urgencias se encolaría en la correspondiente cola según su nivel de prioridad. Por ejemplo, supongamos que llega Juan con nivel de prioridad 2:



Cuando llega un paciente con un nivel de prioridad nuevo, hay que crear una nueva cola y colocarla en la lista de forma ordenada según su prioridad. Por ejemplo, supongamos que llega Alicia con nivel de prioridad 5:



Finalmente, si ahora se atiende a un nuevo paciente, sería Ana, que se desencola de la primera cola:



3.1 Construcción del proyecto Practica2ColaPriPacientes.

Construya un nuevo proyecto en *IntelliJ*, con el nombre *Practica2ColaPriPacientes*. En el directorio *src* de dicho proyecto incluya los archivos obtenidos al descomprimir el archivo *Practica2ColaPriPacientes* de la plataforma *Moodle*. Eche un primer vistazo a las clases que contiene:

- Clase ***Paciente***, que es la misma que en el apartado anterior.
- Clase ***Cola*** en la que se define el TAD cola para números enteros, implementada con una lista enlazada de una forma similar a como se ha visto en clase.
- Clase ***Nodo*** para definir los nodos que constituyen la lista enlazada con la que se implementa el TAD cola.
- Clase ***NodoPrioridad*** para definir los nodos que constituyen una lista enlazada para un TAD lista calificada con clave entera y dato entero.

3.2 Construcción del TAD cola de pacientes.

Como el objetivo es construir una lista de colas de pacientes, comenzaremos definiendo el TAD cola de pacientes. Para ello, se modificarán las clases *Nodo* y *Cola*, de manera que manejen objetos *Paciente* en lugar de enteros.

A continuación, se probará su correcto funcionamiento incluyendo código en el *main* de la clase *Aplicación*:

- Crear tres objetos *Paciente* con los siguientes contenidos:
  - Alberto (nombre), Frecuentes mareos (síntomas).
  - Ana (nombre), Infección resistente (síntomas).
  - Eva (nombre), Problemas digestivos (síntomas).
- Insertarlos en el mismo orden en una cola de pacientes.

- Visualizar el contenido de la cola mediante el método *verCola*.
- Desencolar elementos de la cola hasta que se quede vacía (el método *colaVacia* devuelve *true*).
- Visualizar nuevamente el contenido de la cola.

El resultado de la ejecución será el siguiente:

```
CONTENIDO DE LA COLA:
Nombre: Alberto.  Síntomas: Frecuentes mareos.  Estado: hospitalizado
Nombre: Ana.  Síntomas: Infección resistente.  Estado: hospitalizado
Nombre: Eva.  Síntomas: Problemas digestivos.  Estado: hospitalizado
------------------------
CONTENIDO DE LA COLA:
------------------------
```

3.3 Construcción del TAD cola de prioridades de pacientes.

La cola de prioridades de pacientes es una lista calificada en la que cada elemento tiene:

- Una clave, que es la prioridad.
- Un dato, que es la cola de pacientes asociados a esa prioridad.

Además, los elementos están ordenados por el campo clave y no puede haber elementos repetidos, como corresponde a una lista calificada.

3.3.1. Definir los nodos de la lista calificada.

La clase *NodoPrioridad* define los nodos de una lista calificada en la que tanto la clave como el dato son valores enteros. Se modificará dicha clase para utilizar una prioridad como clave y una cola de pacientes como dato.

3.3.2. Definir el TAD cola de prioridades de pacientes.

Para ello, se utilizará la clase *ColaPrioridades*, que contiene una referencia al primer *NodoPrioridad* de la lista. Aunque se trata de una lista calificada, la abstracción de este TAD requiere de las siguientes operaciones:

- Encolar a un paciente según su prioridad.
- Desencolar al paciente que le toca ser atendido.
- Ver el contenido de la cola de pacientes, desde el primero al último

Por tanto, la siguiente actividad de esta práctica es definir estas tres operaciones en la clase *ColaPrioridades*. A continuación, se detallan las mismas:

- `public void encolar(int prioridad, Paciente paciente)`

  Se busca si existe la cola con la prioridad recibida como parámetro, en cuyo caso se encola en ella al paciente. Si no existe dicha cola, se crea, se inserta en la lista en el sitio adecuado según su prioridad y se encola en ella al paciente. Se recomienda revisar los ejemplos vistos al principio del apartado 3: encolar a Juan con prioridad 2 y encolar a Alicia con prioridad 5.

- `public Paciente desencolar()`

  Se desencola al paciente de la primera cola de la lista, es decir, de la cola que tiene más prioridad. Si dicha cola se queda vacía, se elimina de la lista. Si no existe ninguna cola, es decir, ningún paciente, este método devuelve *null*. Se recomienda revisar los ejemplos de desencolar vistos al principio del apartado 3: desencolar a Felipe y desencolar a Ana.

- `public void verColaPrioridades()`
  Visualiza los datos de todos los pacientes que están en la cola de prioridades, comenzando por el que será atendido en primer lugar y terminando por el último. La estructura de la visualización será la siguiente:

  Prioridad n:
  Lista pacientes con prioridad n
  Prioridad i:
  Lista pacientes con prioridad i

3.4 Utilizar el TAD cola de prioridades de pacientes.

A continuación, se probará el correcto funcionamiento en el *main* de la clase *Aplicación*. Para ello, se incluirá código para realizar las siguientes pruebas:

- Crear una cola de prioridades y encolar en ella a los tres pacientes creados anteriormente: Alberto con prioridad 4, Ana con prioridad 2 y Eva con prioridad 4.
- Visualizar el contenido de la cola de prioridades mediante el método *verColaPrioridades*.
- Desencolar un paciente y visualizar los datos del paciente que se ha desencolado.
- Visualizar nuevamente el contenido de la cola de prioridades.
- Crear un nuevo paciente: Ernesto (nombre), Problemas cardiovasculares (síntomas)
- Encolar a Ernesto con prioridad 1.
- Visualizar nuevamente el contenido de la cola de prioridades.

El resultado de la ejecución será el siguiente:

Prioridad 2:
Nombre: Ana. Síntomas: Infección resistente. Estado: hospitalizado
Prioridad 4:
Nombre: Alberto. Síntomas: Frecuentes mareos. Estado: hospitalizado
Nombre: Eva. Síntomas: Problemas digestivos. Estado: hospitalizado
---
PACIENTE ATENDIDO:
Nombre: Ana. Síntomas: Infección resistente. Estado: hospitalizado
---
Prioridad 4:
Nombre: Alberto. Síntomas: Frecuentes mareos. Estado: hospitalizado
Nombre: Eva. Síntomas: Problemas digestivos. Estado: hospitalizado
---
LLEGA ERNESTO CON PRIORIDAD 1

Prioridad 1:
Nombre: Ernesto. Síntomas: Problemas cardiovasculares. Estado: hospitalizado
Prioridad 4:
Nombre: Alberto. Síntomas: Frecuentes mareos. Estado: hospitalizado
Nombre: Eva. Síntomas: Problemas digestivos. Estado: hospitalizado
---

4. Entrega de la práctica.

Se entregarán los dos proyectos *IntelliJ* resultantes de hacer la práctica: *Practica2ListaPacientes* y *Practica2ColaPriPacientes*, **que tendrán que tener exactamente esos nombres**.

Para entregarlos, se comprimirá cada proyecto en un archivo ZIP y se subirán a la plataforma ambos archivos ZIP. Por tanto, **cada proyecto irá en un archivo ZIP diferente**. Los nombres de dichos archivos ZIP serán los mismos: *Practica2ListaPacientes.zip* y *Practica2ColaPriPacientes.zip*.

No olvide que los proyectos deben incluir las pruebas solicitadas en este enunciado, mediante el código de las funciones *main*.