ESTRUCTURAS DE DATOS
CURSO 2022-23
Práctica 3: Listas

1. OBJETIVOS:

- Definir un TAD Lista Ordinal de Productos a partir del TAD lista ordinal de enteros incluido en las transparencias de clase.
- Usar el TAD lista ordinal de Productos para definir un TAD Factura.
- Usar las listas ordinales que proporciona la biblioteca estándar de Java, para volver a definir el TAD Factura.
- Definir un TAD Conjunto, utilizando una lista enlazada.
- Usar el TAD Conjunto.

2. Definición del TAD lista ordinal de productos.

2.1. El paquete *listaordinal* en el proyecto "ED 3 Practica. Listas".

Descargue el archivo "ED 3 Practica. Listas.zip" de *Moodle* y descomprímalo. Abra el proyecto con *IntelliJ* y, si aparece un mensaje de error, recuerde definir correctamente el JDK (puede consultar cómo en los enunciados de las prácticas anteriores). En la carpeta *src* podrá ver que los archivos se organizan en dos paquetes: *listaordinal* y *conjunto*, que se corresponden con dos partes diferentes de esta práctica. En esta primera parte trabajaremos con el paquete *listaordinal*, cuyos archivos son:

- **Nodo**, **ListaOrdinal**: son las clases que implementan el TAD Lista Ordinal de enteros a través de listas enlazadas. Tienen la misma implementación que las transparencias de clase.
- **Iterador**: clase para definir iteradores sobre la lista ordinal, también implementada de la misma manera que las transparencias de clase.
- **Producto**: es la clase en la que se definen los datos de un producto que se vende en una tienda: descripción del producto, precio por unidad y número de unidades de ese producto vendidas. Se definen dos constructores, uno con los tres datos anteriores y otro para cuando se vende sólo una unidad del producto. Además, proporciona las siguientes operaciones: *getDescripcion*, *getPrecio*, *getUnidades*, *setPrecio*, *setUnidades*, *mostrar* e *equals*.

  La operación *equals* se utiliza para comprobar si dos objetos *Producto* se refieren al mismo producto. Lo cual es cierto si tienen la misma descripción y el mismo precio.
- **PruebasLista** es una clase con un método *main* vacío en el que se irán realizando las pruebas que se vayan pidiendo en esta práctica.

  Nota: se pueden ir realizando en esta clase todas las pruebas que se deseen, pero **la práctica se entregará exclusivamente con las pruebas que se piden en este enunciado**.

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

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 *Producto*. Algunos consejos son:

- Esencialmente, lo que hay que hacer es cambiar el tipo de dato *int* por *Producto*.
- Hay que tener cuidado con las comparaciones que se realizan con el dato en los métodos *borrar* y *posicion*. Mientras que para comparar dos *int* se utiliza == o *!=*, para comparar dos objetos *Producto* se utiliza el método *equals*.
- Mientras que un *int* se visualiza correctamente utilizando el método *System.out.print,* para visualizar un objeto *Producto* se utiliza su método *mostrar*.

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

- Crear cuatro objetos *Producto* con los siguientes valores para descripción, precio y unidades:
  - "Mesa escritorio" (2 unidades) a 185 € c/u.
  - "Silla oficina" (3 unidades) a 95.9 € c/u.
  - "Mesa cocina" (1 unidad) a 125 € c/u.
  - "Sillón reclinable" (2 unidades) a 230 € c/u.
- Insertarlos ese orden en una lista ordinal de productos.
- Visualizar el contenido de la lista utilizando el método *mostrar*.
- Eliminar las tres sillas de oficina.
- Volver a mostrar el contenido de la lista.

El resultado de la ejecución será el siguiente:

```
---------- PRODUCTOS EN LA LISTA -----------
  Mesa escritorio (2 unidades) a 185.0 € c/u. Importe: 370.0 €
  Silla oficina (3 unidades) a 95.9 € c/u. Importe: 287.7 €
  Mesa cocina (1 unidades) a 125.0 € c/u. Importe: 125.0 €
  Sillón reclinable (2 unidades) a 230.0 € c/u. Importe: 460.0 €
Después de borrar las sillas de oficina:
  Mesa escritorio (2 unidades) a 185.0 € c/u. Importe: 370.0 €
  Mesa cocina (1 unidades) a 125.0 € c/u. Importe: 125.0 €
  Sillón reclinable (2 unidades) a 230.0 € c/u. Importe: 460.0 €
```

La clase *ListaOrdinal* así obtenida se utilizará en el siguiente apartado 3. En la realización del mismo, no será válido añadir ningún método más a esta clase, ni cualquier otra modificación.

3. Definición del TAD *Factura*.

Se pretende definir un TAD en el que se encapsulen los datos de una factura: DNI del cliente (*dni*), fecha de expedición (*fecha*), productos vendidos (*productos*) y si la factura está cobrada o no (*cobrada*). Para implementar los productos vendidos se utiliza el TAD Lista ordinal de productos definido en el apartado anterior de esta práctica.

Este TAD se implementa en la clase ***Factura*** contenida en el paquete *listaordinal*. Además de los atributos, anteriormente comentados, esta clase proporciona las siguientes operaciones públicas:

- Constructor con el DNI del cliente y la fecha.
- *getDNI*, *getFecha*, *estaCobrada* (determina si la factura está cobrada o no) y *cobrada* (marca la factura como ya cobrada).

En los apartados que siguen, se pretende definir nuevos métodos que amplíen la interfaz del TAD *Factura*.

3.1. Método *añadirProducto.*

Añadir a la clase *Factura* el método:

```
public void añadirProducto(Producto producto)
```

Que incorpore a la factura el producto recibido como parámetro. En caso de que el producto ya se encontrara en la factura, lo que hay que hacer es incrementar el número de unidades de ese producto en la lista. Por ejemplo, si se desea incluir "silla oficina" (2 unidades) a 95 € c/u, cuando en la factura ya existe "silla oficina" (4 unidades) a 95 € c/u, lo que habrá que hacer es dejar en la factura un producto "silla oficina" (6 unidades) a 95 € c/u.

En definitiva, este método primero buscará en la factura el producto recibido como parámetro para modificarlo si lo encuentra, o añadir un producto nuevo si no lo encuentra.

Este método puede realizarse de varias maneras, pero en ningún caso se considerará como válido recorrer la lista de productos utilizando el método *getElemento*, ya que para eso la solución adecuada es hacer uso de un iterador.

Recuerde que para comprobar si dos objetos *Producto* se refieren al mismo producto, se utiliza el método *equals* de la clase *Producto*.

3.2. Método *mostrar.*

Añadir a la clase *Factura* el método:

```
public void mostrar()
```

Que visualiza el contenido de la factura con el siguiente formato:

```
FACTURA de: 12345678A. Fecha: 17/03/2021
  Mesa escritorio (2 unidades) a 185.0 € c/u. Importe: 370.0 €
  Silla oficina (4 unidades) a 95.9 € c/u. Importe: 383.6 €
  Mesa cocina (1 unidades) a 125.0 € c/u. Importe: 125.0 €
  Sillón reclinable (2 unidades) a 230.0 € c/u. Importe: 460.0 €
```

A continuación, se realizarán las siguientes pruebas en la clase *PruebasLista*:

- Se creará una factura para el cliente con DNI "12345678B", expedida en la fecha "17/03/2021".
- Se añadirán a dicha factura los cuatro productos creados en el apartado 2.2.
- Se creará un nuevo producto: "Silla oficina" (1 unidad) a 95.9 € c/u.
- Se añadirá este último producto a la factura.
- Se mostrará el contenido de la factura con el método *mostrar* de la clase *Factura*.

El resultado de la ejecución será justo el que aparece en este apartado 3.2. como ejemplo de formato.

Recuerde que todas las pruebas que se pidan en este enunciado deberán permanecer en el proyecto al entregar las prácticas.

3.3. Método *importeTotal*

Añadir a la clase *Factura* el método:

```
public float importeTotal()
```

Que determine importe total de la factura, teniendo en cuenta el precio por unidad y el número de unidades de cada producto.

En ningún caso se considerará como válido recorrer la lista de productos utilizando el método *getElemento*, ya que para eso la solución adecuada es hacer uso de un iterador.

Incorporar al método *mostrar* una última línea que incluya el importe total de la factura, según el siguiente formato:

```
IMPORTE TOTAL: 158.66 €
```

Volver a ejecutar la prueba del apartado anterior. De manera que se obtenga el siguiente resultado al mostrar la factura:

```
FACTURA de: 12345678A. Fecha: 17/03/2021
  Mesa escritorio (2 unidades) a 185.0 € c/u. Importe: 370.0 €
  Silla oficina (4 unidades) a 95.9 € c/u. Importe: 383.6 €
  Mesa cocina (1 unidades) a 125.0 € c/u. Importe: 125.0 €
  Sillón reclinable (2 unidades) a 230.0 € c/u. Importe: 460.0 €
IMPORTE TOTAL: 1338.6 €
```

3.4. Método *eliminarProducto.*

Añadir a la clase *Factura* el método:

```
public int eliminarProducto(Producto producto)
```

Que elimine de la factura el producto recibido como parámetro y devuelva el número de unidades del producto eliminadas.

Si el producto a eliminar se encuentra en la factura, lo que hay que hacer es decrementar el número de unidades de ese producto en la lista. Por ejemplo, si se desea eliminar “silla oficina” (2 unidades) a 95 € c/u, cuando en la factura existe “silla oficina” (4 unidades) a 95 € c/u, lo que habrá que hacer es dejar en la factura ese mismo producto con 2 unidades, y devolver el valor 2 (se han borrado 2 unidades).

Si se desea eliminar tantas unidades o más de las que hay en la factura, se eliminará completamente el producto. Por ejemplo, si se desea eliminar “silla oficina” (6 unidades) a 95 € c/u, cuando en la factura existe “silla oficina” (4 unidades) a 95 € c/u, lo que habrá que hacer es eliminar ese producto de la factura, y devolver el valor 4 (sólo se han podido eliminar 4 unidades).

Si el producto a eliminar no se encuentra en la factura, ésta no sufrirá ninguna modificación y el método devolverá 0.

En definitiva, este método primero buscará en la factura el producto recibido como parámetro, para modificarlo o borrarlo si lo encuentra.

Este método puede realizarse de varias maneras, pero en ningún caso se considerará como válido recorrer la lista de productos utilizando el método *getElemento*, ya que para eso la solución adecuada es hacer uso de un iterador.

Recuerde que para comprobar si dos objetos *Producto* se refieren al mismo producto, se utiliza el método *equals* de la clase *Producto*.

A continuación, se probará este método en la clase *PruebasLista*, de la siguiente manera:

- Se crearán dos nuevos productos:
  - "Silla oficina" (2 unidades) a 95.9 € c/u.
  - "Sillón reclinable" (3 unidades) a 230 € c/u.
- Se eliminarán ambos productos de la factura.
- Se mostrará el número de unidades de productos que se han eliminado.
- Se mostrará el contenido de la factura con el método *mostrar* de la clase *Factura*.

El resultado de la ejecución será el siguiente:

```
Se han eliminado 4 unidades
FACTURA de: 12345678A. Fecha: 17/03/2021
  Mesa escritorio (2 unidades) a 185.0 € c/u. Importe: 370.0 €
  Silla oficina (2 unidades) a 95.9 € c/u. Importe: 191.8 €
  Mesa cocina (1 unidades) a 125.0 € c/u. Importe: 125.0 €
IMPORTE TOTAL: 686.8 €
```

3.5. Método *mayoresPrecios.*

Añadir a la clase *Factura* el método:

```
public ListaOrdinal mayoresPrecios(float precio)
```

Que devuelva una lista con todos los productos de la factura que tienen un precio por unidad superior al precio recibido en el parámetro. Por ejemplo, si sobre la factura mostrada en la ejecución anterior, se ejecuta el método *mayoresPrecios(100)*, se obtiene una lista ordinal con dos productos: "Mesa escritorio" (2 unidades) a 185 € c/u y "Mesa cocina" (1 unidad) a 125 € c/u.

En ningún caso se considerará como válido recorrer la lista de productos utilizando el método *getElemento*, ya que para eso la solución adecuada es hacer uso de un iterador.

A continuación, se probará este método en la clase *PruebasLista* de la siguiente manera:

- Se solicitará a la factura una lista de productos con un precio superior a 100 € por unidad.
- Se visualizará un mensaje indicativo del número de productos obtenidos.
- Se mostrará el contenido de la lista obtenida.

El resultado de la ejecución será el siguiente.

```
Se han obtenido 2 productos con precio mayor a 100 € por unidad
  Mesa escritorio (2 unidades) a 185.0 € c/u. Importe: 370.0 €
  Mesa cocina (1 unidades) a 125.0 € c/u. Importe: 125.0 €
```

Comentario final del TAD *Factura*: nótese que en las listas de productos que hemos utilizado, no puede haber elementos repetidos. Por tanto, probablemente habríamos conseguido una mejor solución si hubiéramos tomado como base una lista calificada de productos, para lo cual tendríamos que haber definido un código a cada producto, que sirviese de clave.

4. Definición del TAD *Factura* usando la biblioteca de Java

En este apartado se volverá a definir el TAD *Factura*, pero en lugar de codificar a medida las clases *ListaOrdinal*, *Nodo* e *Iterador*, aprovecharemos la biblioteca estándar de Java para definir la lista de productos. Más concretamente, utilizaremos las clases *LinkedList* e *Iterator* de dicha biblioteca. Como además queremos una lista de productos y un iterador de productos, las parametrizaremos con la clase *Producto*: l*inkedList<Producto>* e *Iterator<Producto>*

En primer lugar, se hará una copia de la clase *Factura*, llamándola *FacturaBib*. Para ello, lo más fácil es hacer esta copia dentro de *IntelliJ*, seleccionado en la ventana del proyecto la clase *Factura* y copiándola en el paquete *listaordinal*. En cualquier caso, lo importante es que tanto el archivo como la clase de la copia se llamen *FacturaBib*.

En *FacturaBib*:

- Utilizar *LinkedList<Producto>* en lugar de *ListaOrdinal*. La interfaz de *LinkedList* puede consultarse en el API de Java o en las transparencias de la asignatura.
- Utilizar *Iterator<Producto>* en lugar de *Iterador*. En este caso los métodos no cambian, siguen siendo *hasNext* y *next*.
- Tanto *LinkedList* como *Iterator* tendrán que ser importadas al comienzo de *FacturaBib*, aunque esta labor es facilitada por *IntelliJ* tan pronto como se usan dichas clases.
- En el método *mostrar* de la clase *Factura* nos encontramos con el problema de que la clase *LinkedList* no tiene ningún equivalente al método *mostrar*. Al utilizar una clase de biblioteca, nos tenemos que adaptar a lo que nos proporciona. En este caso, tendremos que mostrar los datos de los productos utilizando iteradores (*Iterator<Producto>*)

Una vez codificada la clase *FacturaBib*, la probamos el método *main* de *PruebasLista*:

- Crear una factura de biblioteca (*FacturaBib*) para un cliente con DNI “88888888A” y expedida en la fecha “08/08/2008”
- Crear tres nuevos productos:
  - “Armario” (5 unidades) a 385 € c/u.
  - “Cama” (3 unidades) a 255 c/u €.
  - “Cama” (2 unidades) a 255 c/u €.
  - “Armario” (1 unidad) a 385 c/u €.
- Añadir los tres primeros productos a la factura de biblioteca.
- Eliminar el cuarto producto de la factura de biblioteca.
- Mostrar el contenido de la factura de biblioteca.
- Pedirle a la factura de biblioteca una lista de productos (*LinkedList<Productos>*) con un precio superior a 250 € por unidad.
- Visualizar un mensaje indicativo del número de productos obtenidos.
- Mostrar el contenido de la lista obtenida.

El resultado de las nuevas pruebas será el siguiente:

```
FACTURA de: 88888888A. Fecha: 08/08/2008
  Armario (4 unidades) a 385.0 € c/u. Importe: 1540.0 €
  Cama (5 unidades) a 255.0 € c/u. Importe: 1275.0 €
IMPORTE TOTAL: 2815.0 €
```

```
Se han obtenido 2 productos con precio mayor a 250 € por unidad
  Armario (4 unidades) a 385.0 € c/u. Importe: 1540.0 €
  Cama (5 unidades) a 255.0 € c/u. Importe: 1275.0 €
```

5. Definición del TAD Conjunto de enteros no negativos.

En este apartado definiremos un TAD *Conjunto*, cuyos elementos serán números enteros no negativos. Una característica importante de los conjuntos es que no pueden tener elementos repetidos.

La implementación de este TAD se realizará mediante una lista enlazada, en la que cada nodo contiene un elemento, es decir, un entero no negativo.

5.1 El paquete *conjunto* en el proyecto "ED 3 Practica. Listas".

En esta segunda parte de la práctica trabajaremos con los archivos del paquete *conjunto*, que son los siguientes:

- **Nodo**: implementación de los nodos utilizados en la lista enlazada. Cada nodo contiene un entero no negativo.
- **Conjunto**: implementación de la clase que implementa el TAD *Conjunto*. Puede observarse que ya se encuentran implementados los métodos *vacio*, *insertar*, *contiene*, *borrar*, *getNumElementos* y *mostrar*. Si se observan estos métodos, puede verse que son prácticamente iguales a los del TAD *Lista Calificada* de las transparencias. En ambos casos, los elementos se encuentran ordenados de menor a mayor, y no puede haber dos elementos con el mismo valor entero. La diferencia con las listas calificadas es que en los conjuntos cada nodo sólo tiene el número entero, que además actúa como clave.
- **Algoritmos:** clase para implementar código que utilice el TAD *Conjunto*
- **PruebasConjunto** es una clase con un método *main* vacío en el que se irán realizando las pruebas que se vayan pidiendo en esta práctica.

  Nota: se pueden ir realizando en esta clase todas las pruebas que se deseen, pero **la práctica se entregará exclusivamente con las pruebas que se piden en este enunciado**.

Realice las siguientes pruebas en el *main* de la clase *PruebasConjunto*:

- Crear un objeto de la clase *Conjunto*.
- Insertar en el conjunto los siguientes elementos: 4, 6, 2, 4, -2.
- Mostrar el conjunto resultante.

Obtendrá como resultado:

```
Conjunto de trabajo:
[ 2, 4, 6 ]
```

5.2 Definir el método *toArray*.

Añadir a la clase *Conjunto* el método:

```
public int[] toArray()
```

Que devuelva un *array* de enteros con todos los elementos del conjunto. Este *array* tendrá el mismo tamaño que número de elementos tenga el conjunto; y los elementos estarán ordenados igual que en la lista enlazada, es decir, de menor a mayor.

Probar este método en el método *main* de *PruebasConjunto*:

- Utilizar el método *toArray* con el conjunto definido en el apartado anterior.
- Visualizar el contenido del *array* obtenido.

Obtendrá como resultado:

```
El array contiene: 2  4  6
```

5.3 Definir el método *mayor*.

Añadir a la clase *Conjunto* el método:

```
public int mayor()
```

Que devuelva el elemento mayor del conjunto. En caso de que el conjunto esté vacío, este método devolverá -1.

Para realizar este apartado es obligatorio hacerlo recorriendo la lista enlazada. **No es válido utilizar el método *toArray* para recorrer luego el *array*.**

Probar este método en el método *main* de *PruebasConjunto*:

- Ejecutar este método sobre el conjunto de trabajo definido.

Obtendrá como resultado:

```
El mayor elemento es: 6
```

5.4 Definir el método *subconjunto*.

Añadir a la clase *Conjunto* el método:

```
public Conjunto subconjunto(int inferior, int superior)
```

Que devuelva un subconjunto formado por los elementos del conjunto que estén entre los límites inferior y superior, incluidos. Por ejemplo, si sobre el conjunto [4, 7, 9, 12, 21, 33] se ejecuta el método *subconjunto(9, 25)*, se obtendría como resultado el conjunto [9, 12, 21].

Para realizar este apartado es obligatorio hacerlo recorriendo la lista enlazada. **No es válido utilizar el método *toArray* para recorrer luego el *array***.

Se recomienda utilizar el método *insertar* de la clase *Conjunto* para construir el subconjunto resultante.

Probar este método en el método *main* de *PruebasConjunto*:

- Sobre el conjunto definido en las pruebas, ejecutar el método *subconjunto(3, 6)*.
- Mostrar el subconjunto resultante.

Obtendrá como resultado:

```
El subconjunto entre 3 y 6 es: [ 4, 6 ]
```

5.5 Definir el método *equals*.

Añadir a la clase *Conjunto* el método:

```
public boolean equals(Conjunto conjunto)
```

Que determine si el conjunto recibido como parámetro es exactamente igual que el conjunto que recibe el mensaje. Dos conjuntos son iguales si tienen el mismo número de elementos y además estos elementos coinciden exactamente.

Para realizar este apartado es obligatorio hacerlo recorriendo las listas enlazadas de ambos conjuntos, el que recibe el mensaje (*this*) y el parámetro (*otroConjunto*). Se irá avanzando paralelamente en ambas listas enlazadas y comprobando que en todo momento coincide el valor del entero.

No es válido utilizar el método *toArray* para recorrer luego el *array*. Tampoco se considerará una solución válida utilizar el método *contiene* para comprobar que todos los elementos de un conjunto están en el otro.

Probar este método en el método *main* de *PruebasConjunto*:

- Crear un nuevo conjunto e insertarle los elementos 2, 4.
- Comprobar si el conjunto de trabajo ([2, 4, 6]) es igual al conjunto nuevo.
- Añadir al conjunto nuevo el elemento 6.
- Comprobar si el conjunto de trabajo ([2, 4, 6]) es igual al conjunto nuevo.

Obtendrá como resultado:

```
Un conjunto es: [ 2, 4 ]
Es igual que el conjunto de trabajo? false
Un conjunto es: [ 2, 4, 6 ]
Es igual que el conjunto de trabajo? true
```

6. Uso del TAD Conjunto.

En una aplicación en la que se usan conjuntos de enteros no negativos, se necesita obtener la intersección de dos conjuntos. La clase *Algoritmos* se supone que pertenece a dicha aplicación.

6.1 Definir el método *interseccion*.

En la clase *Algoritmos* se definirá el siguiente método:

```
public Conjunto interseccion(Conjunto conjunto1, Conjunto conjunto2)
```

Que devuelva el conjunto intersección de los dos recibidos como parámetros. Este método puede utilizar todas las operaciones pertenecientes a la vista pública de la clase *Conjunto*.

Probar este método en el método *main* de *PruebasConjunto*:

- Crear otro nuevo conjunto e insertarle los elementos 4, 5.
- Crear un objeto de la clase *Algoritmos*.
- Utilizar este último objeto para calcular la intersección entre el conjunto de trabajo ([2, 4, 6]) y el que acabamos de crear.
- Mostrar el resultado de la intersección.

Obtendrá como resultado:

```
La interseccion de [ 4, 5 ] con el de trabajo es:
[ 4 ]
```

7. Entrega de la práctica.

Se entregará el proyecto *IntelliJ* resultante de hacer la práctica “ED 3 Practica. Listas”, **que tendrá que tener exactamente ese nombre**.

Para entregarlo, se comprimirá en un archivo, preferentemente ZIP, y se subirá a la plataforma. El nombre del archivo ZIP será el mismo: “ED 3 Practica. Listas.zip”.

No olvide que el proyecto debe incluir las pruebas solicitadas en este enunciado, mediante el código de los métodos *main* en las clases *PruebaLista* y *PruebaConjunto*