****

Estructuras de Datos

Unidad ll

Presentado por

Luis David Palacio Diaz

7502210079

Docente

Jhon Arrieta Arrieta

Universidad de Cartagena

Facultad de Ingeniería – Ingeniería de Software

III Semestre – Cartagena

2023

TABLA DE CONTENIDO

[**Introducción** 4](#_Toc135333305)

[**Objetivos** 5](#_Toc135333306)

[**Justificación** 6](#_Toc135333307)

[**DESARROLLO DE LA ACTIVIDAD:** 7](#_Toc135333308)

## **Explicación de los Algoritmos de ordenamiento: ……………………………………………….**

1. **Ordenamiento de burbuja:……………………………………………………………………….**
2. **Ordenamiento por inserción (Insertion sort)…………………………………………………**
3. **Ordenamiento por selección (Selection sort):……………………………………………...**
4. **Ordenamiento Shell (Shell sort):………………………………………………………………**
5. **Ordenamiento por casilleros (Counting sort):……………………………………………...**
6. **Ordenamiento de panqueques (Pancake sorting):…………………………………………**
7. **Ordenamiento de mezcla (Merge sort): ……………………………………………………...**
8. **Ordenamiento rápido (Quick sort):…………………………………………………………....**
9. **Ordenamiento Gnome Sort (Stupid sort):……………………………………………………**
10. **Ordenamiento Comb Sort:………………………………………………………………………**

**¿Qué es una Lista? …………………………………………………………………………………**

**Desarrollar un ejemplo en al menos 3 lenguajes de programación, en el cual se pueda usar una lista personalizada y sus operaciones básicas……………………………………**

**Tomar al menos 3 algoritmos de ordenamiento (los mismos ejemplos que realizó en el punto #1.1) y probarlos con el ejemplo de la lista del punto 1.2………………………….**

## **Ejercicios prácticos: 50% de la nota de la actividad………………………………………...**

[**Bibliografía** 71](#_Toc135333309)

## **Introducción**

En esta unidad se busca solucionar problemas planteados con el objetivo de incentivar a los estudiantes a leer, investigar y desarrollar de manera autónoma ejercicios que representen un desafío clave para nuestro crecimiento. El enfoque se centra en aplicar los conocimientos previos sobre arreglos y complementarlos con los conocimientos próximos a aprender.

Además se buscar aplicar el conocimiento adquirido de la investigación de los algoritmos de ordenamiento en tres lenguajes diferentes y explicar cada uno de ellos para comprenden mejor el funcionamiento de estos.

## **Objetivos**

El objetivo principal de este trabajo es adquirir y aplicar conocimientos sobre los algoritmos de ordenamiento en los lenguajes de programación sugeridos, esto permitirá comprender el funcionamiento básico de estos algoritmos y resolver eficazmente los problemas planteados en este trabajo.

## **Justificación**

Este presente trabajo surge con la necesidad de adquirir y expandir los conocimientos sobre los algoritmos de ordenamiento y su implementación en diferentes lenguajes de programación. Como estudiantes podemos resolver adecuadamente cada uno de los problemas planteados y podemos fortalecer y profundizar nuestros conocimientos de manera práctica, demostrando su capacidad de organización y aplicando las utilidades de los temas estudiados a lo largo del proceso de la actividad. Esto les permitirá mejorar sus habilidades de resolución de problemas y su comprensión de los conceptos teóricos mediante su aplicación práctica en situaciones concretas.

## **DESARROLLO DE LA ACTIVIDAD:**

Realizar una investigación documental sobre los siguientes temas y responder las preguntas descritas en este documento.

Con el fin de validar que el estudiante ha realizado un correcto y responsable proceso de estudio, algunas preguntas deben ir acompañadas de un ejemplo simple escrito en al menos 3 de los siguientes lenguajes de programación, estrictamente necesario que cada pregunta deba tener su respectivo:

* 1. Java
  2. C#
  3. C++
  4. Python
  5. JavaScript
  6. Dart
  7. Kotlin
  8. Ruby
  9. Go
  10. Rust
  11. Swift
  12. Delphi
  13. TypeScript
  14. PHP

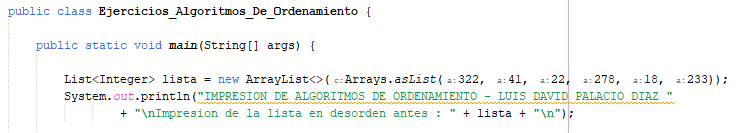
**IMPORTANTE**: El ejemplo debe ser el mismo para los 3 lenguajes por cada pregunta, no se aceptan ejemplos diferentes por cada lenguaje para una misma pregunta.

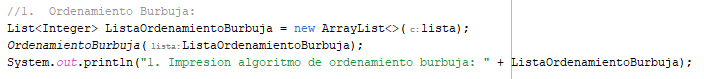
1. Responda las siguientes preguntas, explicando y argumentando con sus propias palabras cada ítem, es ideal, pero no es necesario, que realice un esquema o gráfico que pueda complementar dicha explicación.
   1. Explicar al menos 10 de los siguientes algoritmos de ordenamiento. Debe usar un único conjunto de datos (array o lista) y escribir un ejemplo de cada algoritmo en al menos 3 lenguajes de programación de los descritos anteriormente. Se debe adjuntar a este informe (subiendo a la plataforma SIMA-PESAD) un archivo comprimido con el código de los ejemplos.

## **Explicación de los Algoritmos de ordenamiento:**

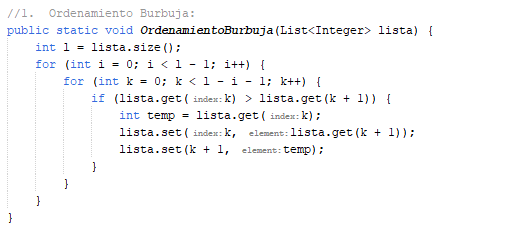
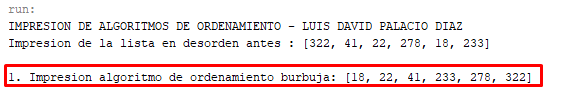
Después de haber consultado varias fuentes de información entre videos, libros, paginas, hemos logrado aprender un poco sobre los algoritmos de ordenamiento por eso pasaremos a realizar una explicación de lo que hemos entendido de cada tipo de algoritmo de ordenamiento, solo hemos seleccionado diez (10) de los veinte (20) tipos de ordenamiento ya que es lo que nos pide el enunciado.

1. **Ordenamiento de burbuja:**

Este algoritmo consiste en darle orden a una lista de elementos, su forma de proceder es intercambiar elementos entre ellos para ir validando si están en el orden incorrecto, básicamente funciona como un bucle ya que se estará realizando varias veces la operación hasta que la información quede totalmente organizada.

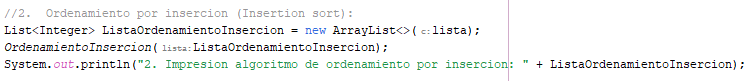
En esta sección del código lo que inicialmente creamos es una lista llamada lista y la vamos a inicializar con los valores aleatorios de 322, 41, 22, 278, 18 y 233, solo crearemos una lista con 5 elementos para todos los tipos de algoritmos de ordenamiento que utilizaremos, esto para evitar escribir mucho más código y hacer el proceso algo menos complejos, luego de crear nuestra lista pasaremos a imprimir nuestra lista y le agregamos el mensaje de impresión de la lista en desorden antes, esto lo hacemos para que se pueda evidenciar al imprimir como lucia la lista antes y después.

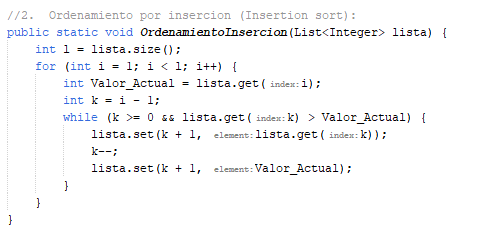
Luego en esta siguiente sección se crea una nueva lista que recibe por nombre ListaOrdenamientoBurbuja, la cual vamos a inicializar con los mismo valores que nuestra lista original llamada lista, de esta manera podremos comparar los resultados al final, después de esto hemos pasado a crear una función llamada OrdenamientoBurbuja, pasando la ListaOrdenamientoBurbuja como argumento, (Cabe destacar que cuando nos referimos a la palabra “argumento” estamos diciéndole a nuestro código que tome esa lista como entrada para que nuestra función pueda procesarla) por ultimo ingresaremos un mensaje donde estamos indicando el resultado de nuestro algoritmo de ordenamiento burbuja con la lista en su respectivo orden de menor a mayor.

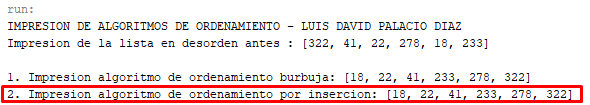
En la primera línea de código estamos definiendo nuestra función llamada OrdenamientoBurbuja, la cual es declarada de forma publica lo cual nos indica que puede ser accedido desde cualquier parte del código, además se ha definido estático por que puede ser llamado directamente en la clase que esta definido sin necesidad de crear un objeto de esa clase, luego con el tipo de dato void la cual nos está indicando que nuestra función no va retornar ningún valor; luego en la siguiente línea de código donde tenemos lista.size estamos declarando una variable 1 que almacenara el tamaño de nuestra lista, esto significa que contiene la cantidad de elementos de esta lista, luego con el bucle for y lo que esta contenido dentro de este bloque de código de nuestro bucle lo que estamos realizando en esta sección son iteraciones desde nuestro índice cero hasta nuestro penúltimo elemento de la lista, sabemos que la letra i hace referencia a nuestro índice en cada una de las iteraciones, luego en la siguiente línea de código del bucle for anidado lo que estamos haciendo es comparar los elementos adyacentes y realizar el respectivo intercambiando de elementos en caso que sea necesario (Cuando hablamos de adyacente estamos refiriéndonos a que se van a ir comparando los elementos que se encuentran uno al lado del otro) luego en esta mismo bloque de código hemos asignado la variable K la cual estará representando nuestro índice cada vez que se realice una iteración, entonces creamos una condición con if en la cual estamos indicando que si el elemento que esta en nuestra posición k es aun mayor que el elemento en nuestra posición K + 1, en caso de ser verdadero (true) se ejecutara el código que se encuentra en el bloque de nuestro condicional if; en la siguiente línea de código creamos nuestra variable temporal llamada temp, la cual va almacenar de manera temporal los valores antes de intercambiar los elementos, el valor que tomara temp es el valor de nuestro elemento en la posición K, luego en las ultimas dos líneas de código lo que estaremos haciendo es en la primera reemplazar el valor de nuestro elemento en la posición K con el valor de nuestro elemento en la posición K + 1, con esto se puede hacer el intercambio entre los elementos que se encuentran adyacentes, es decir unos al lado de otros, para finalizar en esta última línea de código lo que se esta haciendo es reemplazar el valor de nuestro elemento K + 1 con el valor que esta guardado en nuestra variable temporal llamada temp, para asi poder completar la operación de intercambio entre los diferentes elementos.

Aquí por último podemos ver que cuando imprimimos en consola podemos observar el resultado de nuestro algoritmo de ordenamiento burbuja en el cual se puede apreciar como estaba en desorden antes y como quedo organizado luego de menor a mayor.

1. **Ordenamiento por inserción (Insertion sort)**

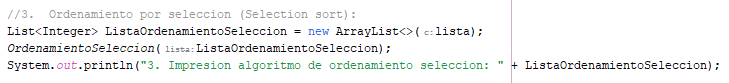
Este algoritmo consiste en darle un orden a una lista de elementos de la siguiente manera: En una lista desordenada toma el primer elemento (Elemento cero) de una lista y empieza a realizar una comparación del primer elemento con el elemento ubicado a su izquierda, como obviamente en la posición cero a su izquierda no hay nada pasaría al siguiente elemento (posición uno) para validar si el numero que esta a su izquierda es mayor que el numero actual y se realiza el cambio, básicamente se repite este proceso una y otra vez hasta que todo quede organizado, normalmente lo que se va ubicando a la izquierda es lo que se ha ido organizando y lo de la derecha es lo que estaba en desorden, básicamente es ir comparando el valor actual con el de la izquierda para poder llegar a su orden.

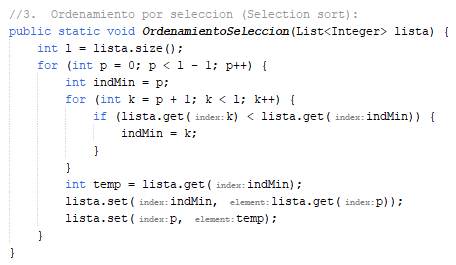
En este siguiente algoritmo de ordenamiento de inserción o en ingles Insertion sort, pasamos a crear una nueva lista que recibe por nombre ListaOrdenamientoInsercion, la cual vamos a inicializar con los mismo valores que nuestra lista original llamada lista, de esta manera podremos comparar los resultados al final, después de esto hemos pasado a crear una función llamada OrdenamientoInsercion, pasando la ListaOrdenamientoInsercion como argumento, (Cabe destacar que anteriormente hemos explicado a que se refiere la palabra “argumento”) por ultimo ingresaremos un mensaje donde estamos indicando el resultado de nuestro algoritmo de ordenamiento inserción con la lista en su respectivo orden de menor a mayor.

En esta sección del código como siempre realizamos se debe declarar el método que en este caso lo llamaremos OrdenamientoInsersacion, este método igual que el anterior y que todos los que vienen siempre reciben una lista de enteros como parámetro, esto nos damos cuenta por List<Integer>, después de esto pasamos a obtener la longitud de la lista uno mediante lista.size esto ya anteriormente se ha explicado entonces continuamos con un bucle for que realizara las respectivas iteraciones desde nuestro segundo elemento de la lista hasta el último elemento, esto lo hacemos gracias a que inicializamos la variable i = 1 y realizamos la condición de salida i < 1, luego en la siguiente línea de código dentro del bucle almacenaremos el valor actual que recibe nuestro elemento, esta la guardaremos en la variable Valor\_Actual usando en lista.get(i) este método nos sirve para poder acceder al valor de un elemento en una posición especifica de nuestra lista, luego de esto se procede a establecer una variable K la cual vamos a inicializar de la siguiente manera i – 1 esto es para ir comparando el valor actual con el elemento anterior, por ejemplo si tenemos que i es igual a 4, entonces k se inicializa obviamente en 3 ya que 4 – 1 es igual a 3, entonces comparara el valor actual de nuestro elemento con la posición numero cuatro de nuestra lista, de esta manera se realiza la operación para poder insertar este en la posición correcta, con este breve ejemplo explico para que se utiliza lo anteriormente mencionado, continuando con la explicación seguimos a la siguiente línea donde hemos iniciado un bucle while que estará en ejecución mientras que k sea mayor o igual a cero y que nuestro elemento en el índice K de nuestra lista sea mayor que el Valor\_Actual, esto es para poder buscar la posición correcta donde se insertara nuestro valor actual en la sección ordenada de nuestra lista, luego en nuestro bucle while se ira desplazando cada elemento que sea mayor que nuestro valor actual y se moverá una posición hacia la derecha para ir haciendo un espacio para nuestro valor actual, después de salir del bucle while, hemos encontrado la posición correcta para insertar el valor actual, así se procede hacer la inserción del valor actual en la posición k + 1, de esta manera en el bucle for se continuara repitiendo el proceso hasta que todos los elementos de la lista se encuentren ordenados en su posición correcta.

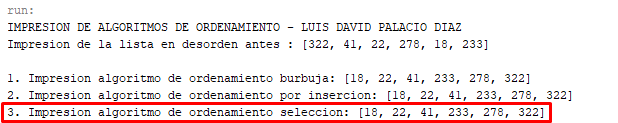
Aquí por último podemos ver que cuando imprimimos en consola observamos el resultado de nuestro algoritmo de ordenamiento por inserción en el cual se puede apreciar cómo estaba en desorden antes y como quedo organizado luego de menor a mayor.

1. **Ordenamiento por selección (Selection sort):**

Este método consiste en darle un orden a una lista de elementos los cuales se encuentran en desorden, la principal tarea es tomar el valor mas grande o pequeño e ir organizándolo en su respectivo orden, mas detalladamente podríamos decir que si tenemos una lista de elementos en desorden pues lo que haremos será buscar su respectivo elemento mas pequeño en la lista y lo iremos intercambiando con el primer elemento de nuestra lista y así iremos buscando siempre el valor mas pequeño y lo iremos asignando en la posición correspondiente, básicamente siempre es la misma tarea, buscara el valor más pequeño en cualquier posición y la asignara en el orden correspondiente.

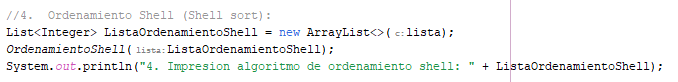
En este algoritmo de ordenamiento por selección o en ingles Selection sort, pasamos a crear una nueva lista que recibe por nombre ListaOrdenamientoSeleccion, la cual vamos a inicializar con los mismo valores que nuestra lista original llamada lista, de esta manera podremos comparar los resultados al final, después de esto hemos pasado a crear una función llamada OrdenamientoSeleccion, pasando la ListaOrdenamientoSeleccion como argumento, (Cabe destacar que anteriormente hemos explicado a que se refiere la palabra “argumento”) por ultimo ingresaremos un mensaje donde estamos indicando el resultado de nuestro algoritmo de ordenamiento por selección con la lista en su respectivo orden de menor a mayor.

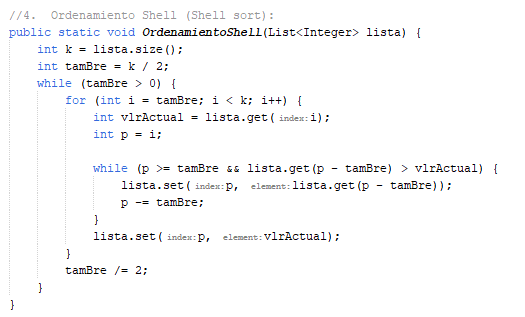
En esta sección del código realizamos la declaración del el método, en este caso lo llamaremos OrdenamientoSeleccion, este método igual que el anterior recibe una lista de enteros como parámetro, esto nos damos cuenta por List<Integer>, después de esto pasamos a obtener la longitud de la lista mediante lista.size y almacenamos la cantidad de elemento en la variable uno, luego en el bucle for podemos evidenciar que realizara iteraciones desde el índice cero tal cual como se indica en el bucle hasta su penúltimo elemento (l – 1), sabemos que la variable p = 0 estará representando el índice del elemento actual, luego con int intMin = p; estaremos inicializando la variable indMin con nuestro valor de P el cual va representar el índice actual del elemento mas pequeño, luego el bucle for interior realiza iteraciones sobre los elementos restantes de nuestra lista a partir del siguiente índice después de p, donde k representa nuestro índice del elemento actual en el bucle interior, luego en el condicional if, se comparan los elementos de nuestro índice k con el valor mas pequeño, si el valor de nuestro elemento en el índice K es menor se actualizara nuestra variable indMin con el valor de K y así indicara que ha hallado un nuevo elemento mas pequeño, seguido de esto se guardara el valor del elemento mínimo en nuestra variable temp (temporal), luego en la siguiente línea de código, lo que estaremos realizando es intercambiar la posición de los elementos en la lista, el valor se asignara al nuestro índice p al lugar del elemento mas pequeño, eso significa en sencillas palabras que el elemento más pequeño encontrado ahora ocupara la posición actual de p y de esta manera se estará moviendo el elemento que hallamos antes en esa posición al lugar donde hallamos el elemento mas pequeño y por ultimo en esta línea de código que sigue se asignara el valor que hemos guardado en nuestra variable temp (temporal) y se moverá al lugar donde se hallaba nuestro elemento en el índice P, de esta manera hemos logrado completar el intercambio de posiciones, aunque suene un poco complejo de entender la explicación es la manera mas sencilla entre todas las maneras que plantee el escribir esta descripción de lo que sucede en este algoritmo de ordenamiento de selección.



Aquí por último podemos ver que cuando imprimimos en consola observamos el resultado de nuestro algoritmo de ordenamiento por selección en el cual se puede apreciar cómo estaba en desorden antes y como quedo organizado luego de menor a mayor.

1. **Ordenamiento Shell (Shell sort):**

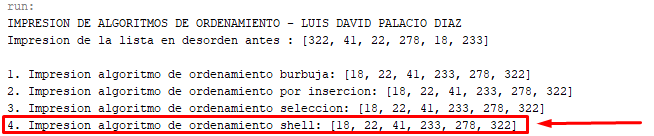
El algoritmo de ordenamiento Shell básicamente consiste en dividir una lista en subgrupos para poder facilitar el trabajo ya que aplicara el ordenamiento por inserción en cada subgrupo, a medida que nuestro algoritmo vaya avanzando el subgrupo se hará mas pequeño para poder mover los elementos grandes a la posición final, básicamente es parecido al de inserción solo que divide los elementos en 2 grupos partiendo de la mitad de los elementos para que el trabajo se realice de manera mucho más fácil y rápida.



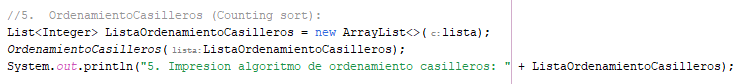
En este punto realizamos el ejemplo del algoritmo de ordenamiento shell, el cual como podemos notar parece una mejora del algoritmo de inserción, lo primero es obtener el tamaño de la lista por medio de lista.size y vamos almacenar ese valor en nuestra variable k, luego inicializamos nuestra variable tamBre con la mitad que tenemos en la variable k, esta variable representara el tamaño inicial de nuestra variable tamBre la cual se utilizara para realizar comparaciones entre los elementos, luego iniciamos nuestro bucle while que estará en ejecución mientras que nuestra variable tamBre sea mayor que cero, lo que hacemos con este bucle es controlar el proceso de reducción del tamaño de nuestra variable hasta que sea igual a cero para lo cual el algoritmo en ese momento habrá terminado, obviamente dentro del bucle se realizaran iteraciones en los elementos de la lista, por la cual empezara desde la variable tamBre hasta nuestra variable k, de esta manera se irán comparando los elementos distantes en cada una de la iteraciones; de esta manera en cada iteración se ira guardando el valor actual en nuestra variable vlrActual y se guardara su posición en la variable creada p, luego de esto se inicia un bucle while que se ejecutara mientras que nuestra variable p sea mayor o igual a la variable tamBre y mientras que el elemento en nuestra posición p – tamBre sea mayor que nuestra variable vlrActual, de esta manera se realizaran diferentes comparaciones y se irán desplazando los elementos que sean mayores que vlrActual hacia la derecha, de esta manera se creara un espacio para poder insertar el valor actual en la respectiva posición correcta, una vez que se halle la posición correcta se utiliza el método set para asignara el vlrActual en esa posición, luego de utilizar el bucle interno se va reducir a la mitad el tamaña de nuestra variable tamBre realizando una división por dos, de esta manera se repite el bucle while hasta que tamBre sea igual a cero y este punto habrá finalizado.

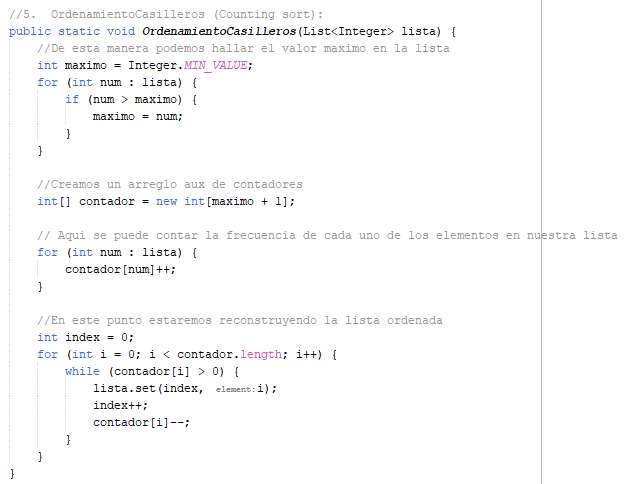
A continuación se muestra la impresión por consola de esta manera repitiendo el proceso como en los anteriores algoritmos de ordenamiento se pasará a

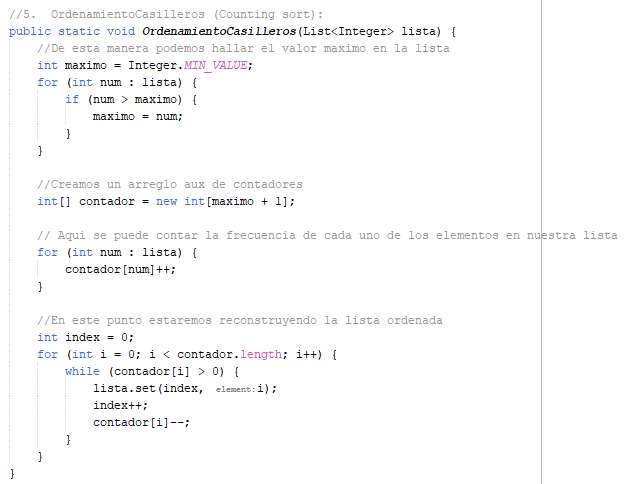
imprimir el resultado tal cual como se puede apreciar en la siguiente imagen.



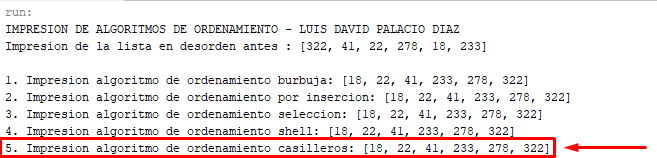
1. **Ordenamiento por casilleros (Counting sort):**

Este método no utiliza la comparación como en los casos anteriores, lo que en realidad se hace es ordenar los elementos agrupándolos en una clasificación y luego pasaremos a utilizar la clasificación para volver a armar nuestros elementos, pero en una lista de forma ordenada.



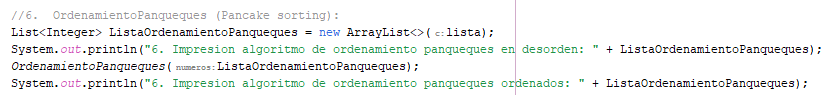


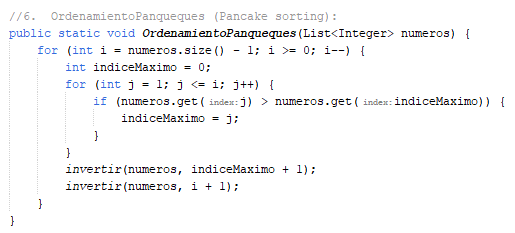
En este algoritmo de ordenamiento como siempre creamos nuestra lista la cual le llamamos ListaOrdenamientoCasilleros la cual se va a inicializar con los elementos que están ubicados en nuestra lista original llamada lista, como ya sabemos como funciona todo este proceso dentro del método main iremos directamente al punto de la explicación del código fuera del método main ya que aquí es donde sucede la magia; bueno para empezar en este código lo que buscamos es el valor máximo de nuestra lista en la cual se estará realizando distintas iteraciones en los elementos y a su vez se ira actualizando nuestra variable máximo en caso que se encuentre algún numero mayor, de esta manera se nos permite determinar el rango de los valores existentes en nuestra lista, luego hemos creado un arreglo el cual le asignamos el nombre de contador, el tamaño de este es la variable máximo + 1, este arreglo lo estaríamos utilizando como un arreglo auxiliar para almacenar la frecuencia de cada uno de los elementos en la lista, luego se recorrerá la lista nuevamente y se ira incrementando el contador correspondiente a cada uno de los numero hallados, luego de contar la frecuencia de cada uno de los números se procede a reconstruir la lista ordenada utilizando la variable index para poder validar la posición actual en la lista, luego realizaremos iteraciones en el arreglo llamado contador y en cada índice se implementara nuestro bucle while mientras que el contador sea mayor que cero, luego dentro del bucle se establecerá el numero i en la posición actual de la lista y se incrementa index y luego se decrementara el contador, este proceso se seguirá repitiendo hasta que la todos elementos sean procesados y se pueda reconstruir la lista de manera ordenada y de esta manera finaliza el proceso, a continuación se relaciona la imagen de la impresión en consola.



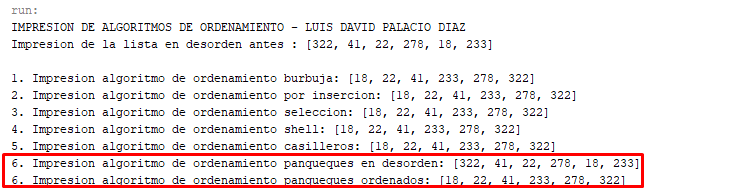
1. **Ordenamiento de panqueques (Pancake sorting):**

Este algoritmo básicamente consiste en algo sencillo como es darle un orden a una lista de manera ascendente o descendente de esta manera se ubicarán en su manera correcta según sea la preferencia, básicamente es como decir invertir la posición de los elementos sea del menor al mayor o del mayor al menor, pero básicamente es buscar el elemento mas grande de la lista y colocarlo al final o al inicio según sea la forma en que sea requerido.



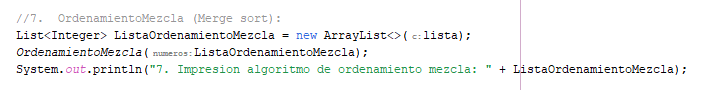


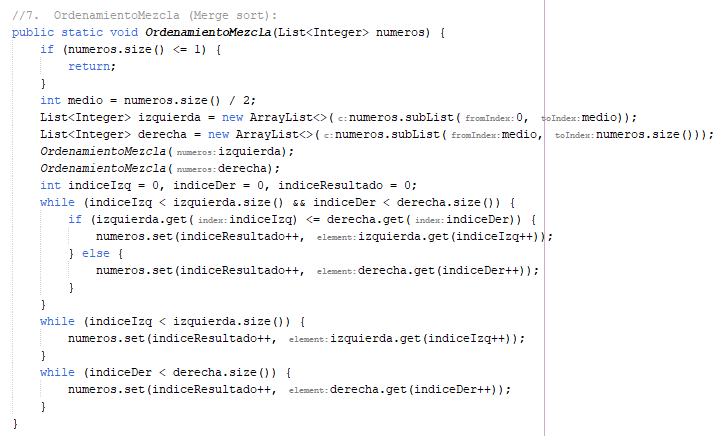
En la primera sección del código lo que hacemos es recorrer la lista números en orden inverso, de esta manera estaríamos empezando desde nuestro ultimo elemento en la lista e ir retrocediendo hasta nuestro primer elemento, esto lo realizamos gracias al bucle for que tiene una variable i la cual es inicializada con numeros.size -1 y en cada iteración se ira decrementando hasta que alcance la posición cero, luego pasamos a declarar nuestra variable indiceMaximo la cual se inicializa en cero, esta variable que mencionamos anteriormente la estamos utilizando para poder rastrear el índice del elemento máximo que es hallado en cada iteración de nuestro bucle interno, luego pasamos a crear un nuevo bucle for con la variable j, que va desde 1 hasta i, este bucle estará recorriendo los elementos desde nuestro segundo elemento hasta el elemento actual en la iteración del bucle externo, de esta manera dentro de nuestro bucle se estará realizando una comparación con nuestro elemento del índice máximo y en caso que se halle algún elemento de mayor valor se actualizara nuestra variable de indiceMaximo con el valor de j, después de hallar el índice del elemento máximo dentro de los elementos que no están ordenados pues se pasara a invertir la sublista mediamente el método invertir con nuestros parámetros de números y indiceMaximo + 1, luego invertiremos nuevamente la lista hasta la posición i+1 nuevamente aplicando el método de invertir; haciendo un poco de énfasis en la sección del código donde se explica el método invertir pues lo utilizamos para poder invertir una sublista de los elementos que se encuentran dentro de una lista principal, ya sabemos que se crea una sublista mediante el método subList de nuestra clase List, como tal esta contiene los elementos desde el índice cero hasta nuestro índice k -1 luego realizan iteraciones sobre la mitad de los elementos de la sublista utilizando nuestro bucle for, luego mediamente la variable temp se almacenar de manera transitoria el valor de un elemento mientras se van realizando los intercambios, al finalizar el bucle la sublista se habrá invertido de manera correcta en su respectivo lugar dentro de la lista principal, a continuación se relaciona la impresión por consola en el cual se muestran los elemento en desorden y luego se evidencia como quedan en su respectivo orden.



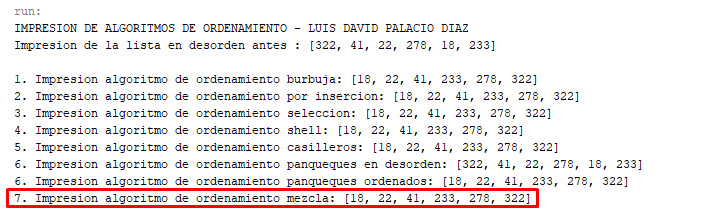
1. **Ordenamiento de mezcla (Merge sort):**

Este algoritmo básicamente tiene unas 3 fases la cual consiste primero en dividir nuestra lista original en dos partes que contengan la misma cantidad de elementos, luego se procede a ordenar en cada mitad de las sublistas para que el proceso no sea tan complejo y se haga mas sencillo y al final solo se debe combinar para obtener una única lista ordenada, ya luego miraremos de que manera lo queremos ordenar si de manera ascendente o descendente, básicamente este es uno de los algoritmos de ordenamiento mas eficiente por su manera de organizar los elementos en sublistas para luego darle su orden de manera más rápida.



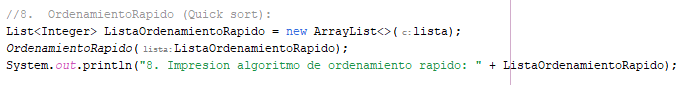


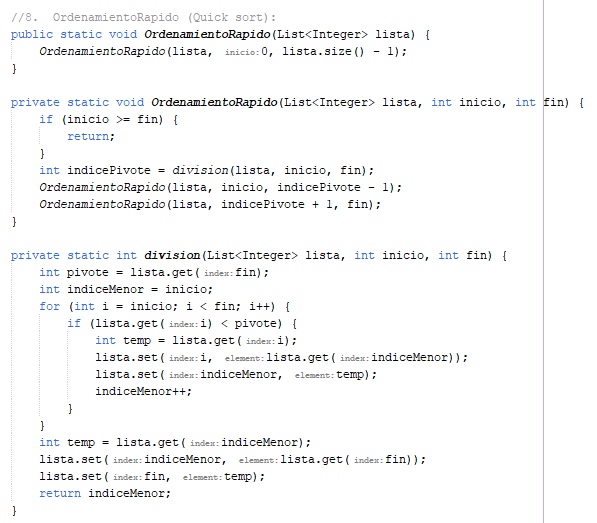
En este algoritmo de ordenamiento por mezcla lo primero que hacemos es crear una función llamada OrdenamientoMezcla la cual va recibir una lista de numero que se desea ordenar, esta comprobara si nuestra lista tiene un tamaño menor o igual a uno, en caso que sea así pues significa que la lista ya estará ordenada o que simplemente no contiene elemento, por lo cual no será necesario realizar mas operaciones pero en caso contrario que la lista posea mas de un elemento se calculara el punto medio de esta y realizara por llamarlo así una repartición o división en la cual existirán dos sublistas la cual hemos llamado izquierda y derecha, la primera sublista va poseer los elementos desde el inicio que seria la posición cero hasta el punto medio y la derecha va a obtener el restante hasta el final, luego se realizara el llamado de la función OrdenamientoMezcla para poder ordenar ambas sublistas tanto la izquierda y la derecha, luego de haberse organizado ambas sublistas se combinaran los resultados de ambas en una única lista ordenadas, en esto aplicaremos los tres índices indiceIzq, indiceDer y indiceResultado, todos se van a inicializar en cero, luego aplicaremos un bucle while para poder ir comparando los elementos de las 2 sublistas, mientras ambos índices sean menores que el tamaño real de la sublista se irán comparando elemento actual de esta sublista izquierda con el de la sublista derecha, de esta manera si el elemento de la primera sublista es menor o igual se colocara en la posición actual de nuestra lista original llamada números y se incrementara el índice de la primera sublita y el índice del resultado, así mismo sucede con la sublista derecha, si el elemento es menor se colocara en la posición actual y se incrementaran los índices que corresponden, luego de salir de este bucle puede existir algunos elementos restantes para lo cual utilizaremos otros dos bucles while para poder agregar los elementos restantes a ambas sublistas, luego de realizar este proceso deberían quedar todos ordenados en una lista original llamada números, aquí todos los elementos se deben encontrar de manera ordenada para poder finalizar el proceso, a continuación se puede evidenciar el proceso finalizado con su respectiva impresión en consola, se encuentra de forma ordenada.



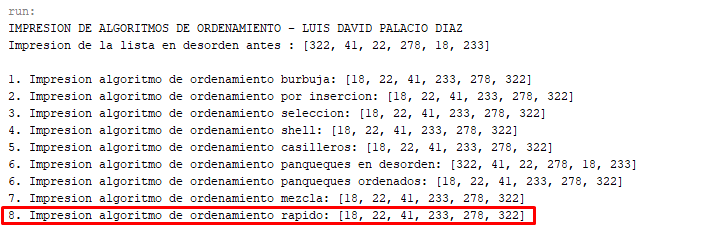
1. **Ordenamiento rápido (Quick sort):**

Este suele ser uno de los algoritmos de ordenamiento mas eficientes ya que divide la lista en dos para poder realizar el orden de los elementos de una manera mas sencilla, lo que hace es tomar un elemento de nuestra lista y lo va comparando con cada posición y así va ubicando los elementos de tal manera que a la izquierda se ubican los números menores que este y a la derecha se ubican los números mayores a este, siendo así este proceso se repite una y otra vez hasta que toda la lista quede totalmente organizada.



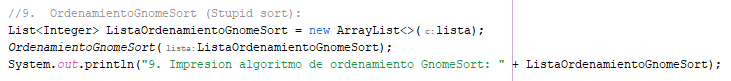


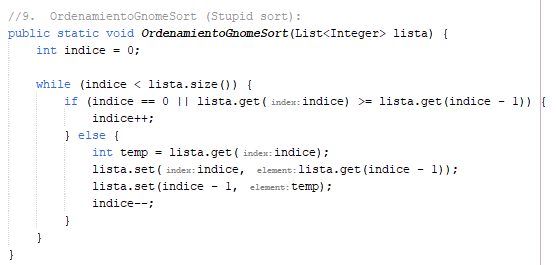
Para el ordenamiento rápido comenzamos haciendo el método con la palabra static que pertenecería a una clase como tal y no sería una instancia, el void es su declaración que no devolverá ningún valor, en este ordenamiento utilizaremos una lista de tipo entero y la función privada de OrdenamientoRapido() es parte de la función para indicar como se llama e iniciar el proceso. Iniciamos con OrdenamientoRapido (listas,0, lista.size()-1) que pondrá de parámetros la lista original de los actores ya previamente asignados y tomamos el indice cero que es el inicio de la lista que comenzara ordenar, tenemos lista.size() que nos va a ordenar la lista y retornará el tamaño de la misma menos 1 que va a obtener último índice de los elementos que estén en la lista, luego tenemos la siguiente función que abarcará en ella la condición ya que se toma el índice del inicio y el final para que se verifique si este índice es mayor o igual al índice final, si es así, entonces no hay elementos por tanto no sería necesario hacer la operación, pero si el índice es menor que el índice final entonces se llama a la función división () y se divide la lista en dos parte del elemento pivote menos 1 en OrdenamientoRapido(lista, inicio, indicePivote-1), adicional ordenamos las partes separadas se ordena desde el su arreglo a la derecha más 1 hasta llegar al índice final, a continuación se presenta la impresión en consola.

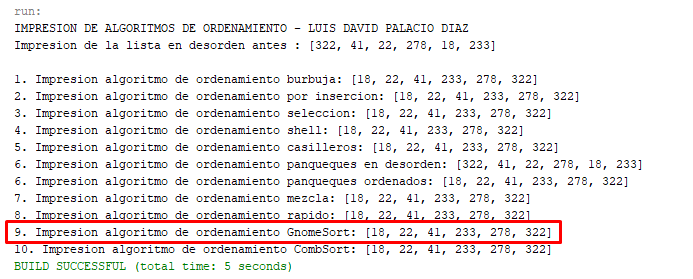


1. **Ordenamiento Gnome Sort (Stupid sort):**

Este algoritmo también es conocido como el ordenamiento estúpido, pero no por su nombre debe ser necesariamente eso, por el contrario es muy interesante ya que este tipo de ordenamiento va comparando los elementos dentro de la lista, este siempre comienza en la posición cero de nuestra lista, en este proceso se evidencia que evalúa al tamaño del elemento, si el elemento actual es menor que nuestro elemento anterior intercambiara nuestros elementos y retrocederá de posición, esto lo hará para poder comparar todos lo elementos de la lista y darle un buen orden, siempre se repetirá el mismo proceso hasta que la lista quede bien organizada.

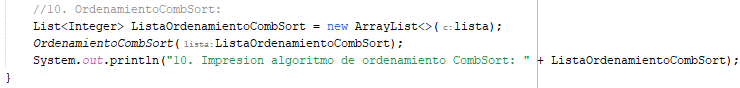


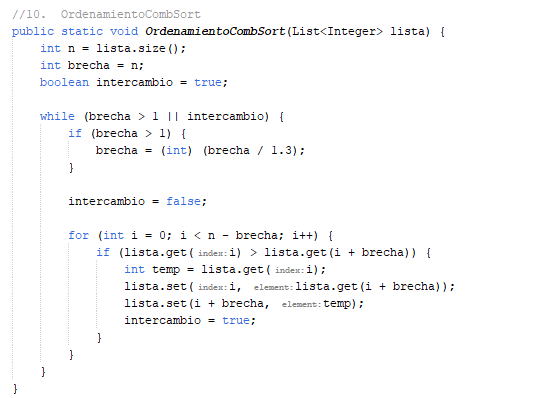


Bueno este algoritmo su nombre es un poco extraño se hace llamar en español el algoritmo estúpido o quizás le podríamos llamar el algoritmo tonto, en este caso podemos evidenciar que hemos inicializado una variable llamada índice en cero, esta variable como tal podrá representar la posición actual en la lista que estamos ordenando, esto empezara con un bucle while el cual se ejecutara mientras que nuestro índice sea menor que el tamaño de nuestra lista, de esta manera se podrá asegurar de manera satisfactoria que se puedan recorrer absolutamente todos los elementos que estén en la lista y así poder realizar comparaciones y todos los intercambios que sean necesarios, luego dentro del bucle se podrá validar si el índice es igual a cero o si en realidad el elemento en la posición índice es mayor o igual que el elemento que esta en la posición de índice - , en caso que se cumpla esta condición significara que los elementos están en orden o por otro lado que el índice esta en la posición inicial, por lo cual el índice será incrementado en uno para poder avanzar al siguiente elemento, luego si la condición anterior no se cumple se puede decir que el elemento en la posición de índice es menor que el elemento en la posición índice – 1 y estarán en el orden incorrecto, como tal en este caso se realizara un intercambio en los elementos con la variable que se ha implementado en los otros algoritmos llamada temporal, temp, luego elemento en la posición dice se va mover una posición menos es decir estará realizando el respectivo intercambio con el elemento anterior y el índice disminuirá en uno para poder volver a comprar el elemento que recién se intercambio con el anterior, luego el bucle continuara hasta que índice pueda recorrer absolutamente toda lista y que se haya verificado todos los pares de los elementos que se encuentran uno al lado del otro, después de completar este ultimo proceso se puede decir que la lista quedo ordenada y se concluirá todo el algoritmo, a continuación se presenta la impresión en consola de este algoritmo.

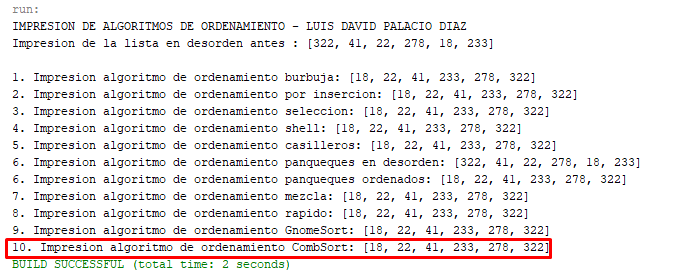
1. **Ordenamiento Comb Sort:**

El algoritmo de ordenamiento Comb Sort consiste básicamente en dividir la cantidad de elementos de una lista entre el factor de encogimiento, que casi siempre suele ser 1.3 (“Es de vital importancia aclarar que se puede usar cualquier valor para el factor de encogimiento pero en este caso seleccionamos el más usado normalmente”) Para validar que siempre el número más pequeño se encuentra en la posición cero (0) se deben comparar los extremos para que de esta manera siempre el elemento más pequeño se encuentre en la posición inicial cero, por ejemplo si nosotros construimos una lista de 8 elementos lo que debemos hacer es una división por la cantidad de elementos que tiene nuestra lista ejemplo que nuestra lista posea 8 elementos y dividirlo por el factor de encogimiento que normalmente suele ser 1.3 lo que daría un resultado de 6,15 lo que debemos hacer es redondear el resultado a 6 para poder trabajar bien con las posiciones de nuestra lista, entonces al tener como resultado 6 lo que haremos será comparar el valor de nuestro primer elemento en la posición cero (0) hasta nuestro elemento en la posición seis (6) hará una comparación del tamaño del elemento y en caso que la posición seis (6) sea menor que la posición (cero) hará un intercambio si no seguirá realizando comparaciones entre el siguiente elemento de la lista que seria a la izquierda la posición numero uno (1) y a la derecha la posición siete (7) que seguía después de la posición anteriormente verificada, en caso de la posición numero siete sea menor que la posición uno (1) realizara el intercambio de los elementos, luego como no tiene posición de elemento numero ocho (8) lo que se realizara es una nueva división, dividiendo el resultado de la anterior división que fue 6 y lo dividimos por el factor de encogimiento que es 1.3 esto daría como resultado 4,61 y si redondeamos nos quedaría cinco (5) luego nuevamente repetimos el proceso desde la posición numero cero (0) hasta la posición numero cinco (5) de esta manera se debe seguir realizando la explicación para poder ordenar todos los elementos de la lista.





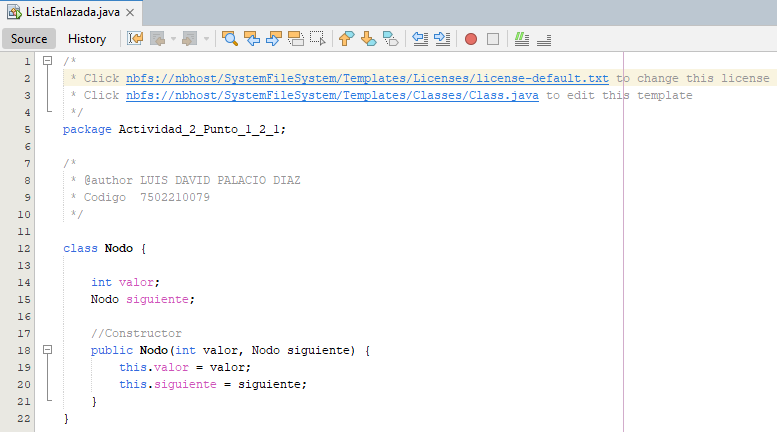
En este algoritmo vamos a obtener como siempre primeramente el tamaño de

la lista y lo guardaremos en la variable n, luego inicializamos nuestra variable llamada brecha con el valor de n, esta como tal representara la distancia entre los elementos que se estarán comparando y ordenando, adicional creamos una variable llamada intercambiando en true para poder controlar si se realizara algún cambio internamente en el bucle, luego dentro del bucle while el cual hemos creado se pasa a validar si la brecha es mayor que el numero uno y en caso de ser así se va actualizar pero dividiendo por un factor de reducción el cual en este caso es 1.3 cabe aclarar que se puede utilizar cualquier otro valor pero este suele ser el más usado, luego de realizar esta operación se pasa a establecer intercambio en false antes de pasar a iniciar el bucle interno eso se hace para poder validar si hubo algún intercambio y así poder determinar si se deben realizar mas iteraciones, seguido de esto pasamos a crear un bucle for el cual ira recorriendo todos los elementos de la lista teniendo en cuenta nuestra brecha actual, el nombre brecha lo hemos asignado así pero puede ser reemplazo por cualquier otro nombre, continuando en la explicación se compararan los elementos en las posiciones i e i + brecha y en caso que el elemento en i sea mayor se realizara el intercambio utilizando otra variable temporal la cual asignaremos el nombre de temp, después de realizar todo este proceso en el bucle interno se valida si se realizo algún intercambio, en caso que haya sucedido nada pues la lista quedara completamente ordenada y se puede salir del bucle y pasara a seguir repitiendo el proceso para ir reduciendo la brecha hasta que se cumpla la condición de salida de este mismo, así de esta manera al final se lograra ordenar de manera eficiente este algoritmo, a continuación se relaciona la impresión por consola de este algoritmo de manera ordenada.

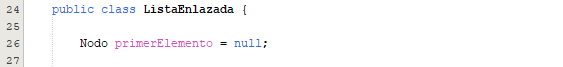
* 1. **¿Qué es una Lista?**

Una lista es una colección ordenada de elementos los cuales tienen su orden y un lugar específico, en las listas puedes agregar, eliminar, cambiar el orden, acceder a cada elemento de la lista o incluso agregar elementos nuevos.

* + 1. **Desarrollar un ejemplo en al menos 3 lenguajes de programación, en el cual se pueda usar una lista personalizada y sus operaciones básicas (no es la que trae el lenguaje dentro de su SDK). El ejemplo debe ser el mismo escrito en los 3 lenguajes que usted elija.**

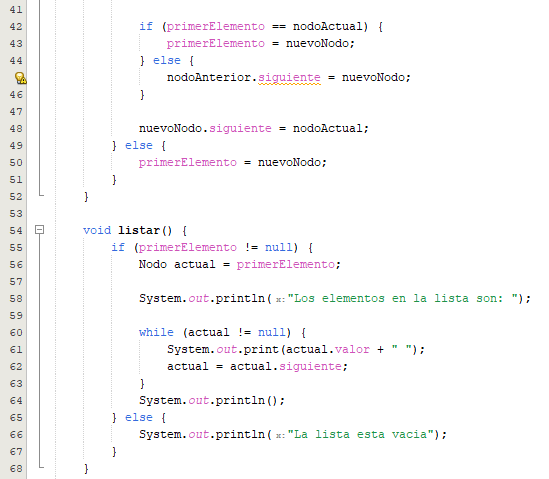
A continuación, se presenta primero el código y luego la explicación

En este código estaremos implementando una lista enlazada y en ella vamos a implementar algunas operaciones como listar los elementos, agregar, buscar, eliminar y limpiar la lista, lo primero que hacemos es crear nuestro paquete el cual le asignamos el nombre de Actividad\_2\_Punto\_1\_2\_1, seguido de esto pasamos a la explicación de la clase llamada nodo, esta misma representara los nodos individuales de nuestra lista enlazada, nos damos cuenta que cada noto tendrá un valor entero el cual le asignamos la variable valor y una referencia al siguiente nodo la cual definimos con la variable llamada siguiente, podemos evidenciar el constructor de la clase la cual se utiliza para poder crear un nuevo nodo y establecer sus valores.

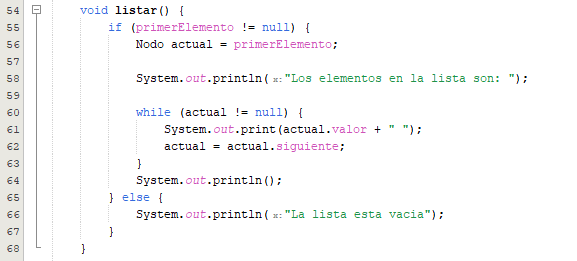


En esta siguiente sección se define la clase ListaEnlazada la cual representa en si la lista enlazada, esta tiene un campo llamado primerElemento la cual es una referencia de nuestro primer nodo en la lista, como tal ese primer elemento se establecerá como nulo (null), lo que obviamente nos va a indicar que la lista se encuentra vacía.

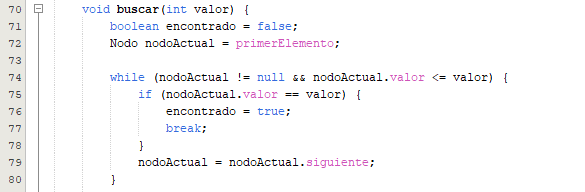


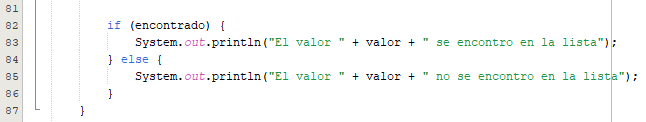


En este método agregar lo usamos para poder insertar un nuevo elemento en nuestra lista enlazada, este recibe un valor entero como parámetro, continuando con la explicación lo primero es crear un nuevo nodo el cual le asignamos la variable nuevoNodo con el valor proporcionado y la referencia al siguiente nodo la cual esta establecida como nulo (null), luego se valida si la lista esta vacía mediante primerElemento == null, en caso que la lista no se encuentre vacía se va proceder a recorrer toda la lista para poder encontrar la posición correcta para poder insertar el nuevo nodo en orden ascendente, de esta manera si el primer elemento de la lista es igual al nodo actual pues se pasara a actualizar el primerElemento para que apunte al nuevo nodo creado, si esto no sucede entonces se actualizara la referencia siguiente del nodoAnterior para que pueda apuntar al nuevo nodo, por ultimo se actualizara la referencia siguiente del nuevo nodo para que este pueda apuntar al nodo actual.

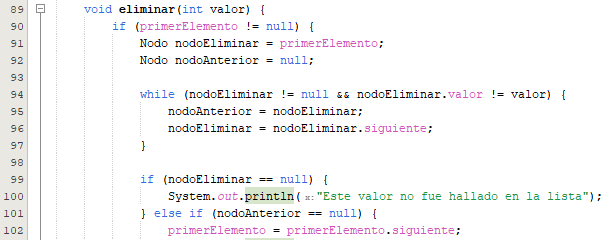


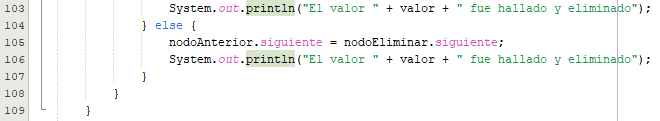
Aquí en este método listar se utilizara para poder imprimir todos los elementos que se encuentran dentro de la lista enlazada, esto se empieza verificando que la lista no se encuentra vacía mediante primerElemento != null, en tal caso que la lista no se encuentre vacía pues básicamente se va pasar a recorrer la lista desde nuestro primer nodo hasta el ultimo nodo, así se ira imprimiendo el valor de cada uno de los nodos en la consola, en tal caso que la lista se encuentre vacía se pasara a imprimir un mensaje que indique que la lista se encuentra vacía.



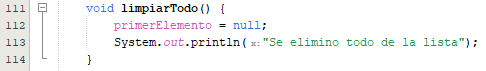


En este método buscar lo utilizamos para poder buscar un valor en nuestra lista enlazada, esta recibe un entero como parámetro, luego comienza estableciendo una bandera encontrado como false y un nodoActual como el primer nodo de la lista, luego vemos que se recorre la lista mientras el nodo actual no sea nulo y mientras que el valor del nodoActual sea menor o igual que el valor buscado, en caso que encuentre el valor se actualizara la bandera encontrado a true y pasara a interrumpirse el bucle luego al finalizar se imprime un mensaje indicando si se encontró o no el valor en la lista, luego saltaremos al método de eliminar.

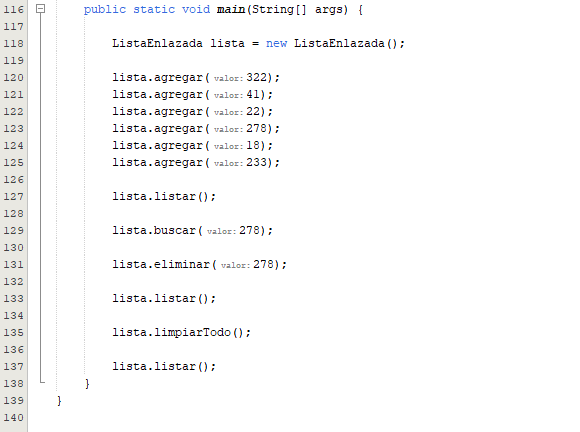




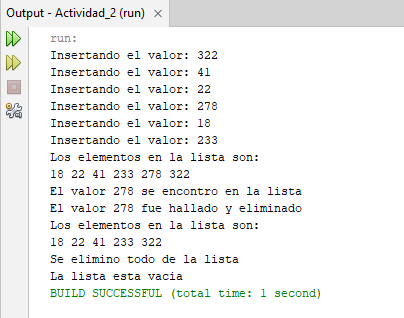
Luego en este método eliminar se utilizará para poder eliminar el valor que nosotros queremos selecciona en la lista enlazada, este recibe como parámetro un entero, comienza verificando si la lista no esta vacía mediante primerElemento != null, luego se recorrerá la lista buscando el nodo con el valor que le estamos solicitando, en caso que se encuentre el valor se actualizara la referencia siguiente del nodoAnterior para poder saltar el nodo a eliminar, pero si el nodo a eliminar es el primer nodo de nuestra lista se actualizara primerElemento para que apunte al siguiente nodo, al finalizar se imprimirá un mensaje indicando que si se encuentre y se elimina el valor de la lista, ahora pasaremos a el método limpiarTodo.



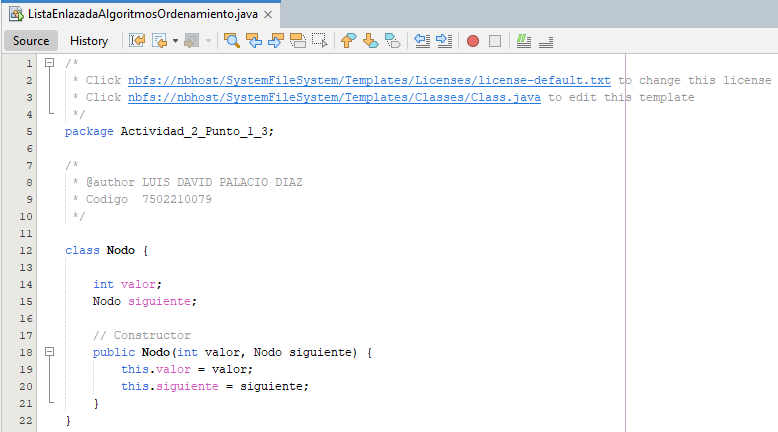
En este método limpiarTodo se utilizará para eliminar todos los elementos que se encuentren en la lista enlazada, para eso se establecerá primerElemento como nulo (null) lo que esto indica que la lista esta vacía, luego se imprime un mensaje diciendo que se eliminaron todos los elementos de la lista.



En esta ultima sección final del código se esta implementando el método main el cual es el punto de entrada del programa, aquí sea creara una instancia de la clases ListaEnlazada llamada lista, seguido de esto se agregan varios elementos a la lista utilizando el método agregar, después de cada operación de agregado se llamara al método listar el cual nos va a mostrar todos los elementos de la lista en ese momento, luego se implementa el método de buscar para poder buscar el valor el cual le hemos pasado que en este caso es 278, lo que hará es buscar y mediante el método eliminar para poder quitar de la lista ese valor, luego se llama nuevamente al método listar para poder mostrar los elementos actualizados obviamente quitando de la lista el elemento que se seleccionó a eliminar, por ultimo se llama el método limpiarTodo para eliminar todos los elementos de nuestra lista y por ultimo se llama nuevamente al método lista para que quede demostrado que la lista quedo vacía, por ultimo se imprime en consola como se puede evidenciar en la siguiente imagen.



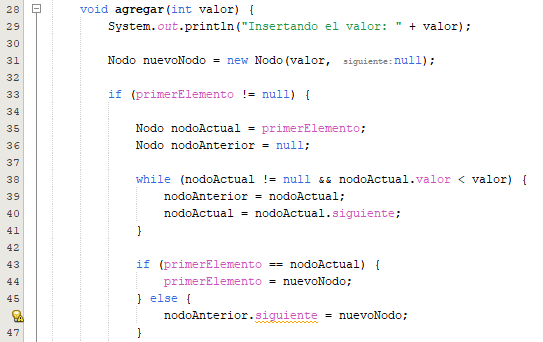
* 1. **Tomar al menos 3 algoritmos de ordenamiento (los mismos ejemplos que realizó en el punto #1.1) y probarlos con el ejemplo de la lista del punto 1.2**

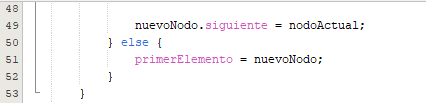


En este código estaremos implementando una lista enlazada y en ella vamos a implementar algunas operaciones como listar los elementos, agregar, buscar, eliminar y limpiar la lista, lo primero que hacemos es crear nuestro paquete el cual le asignamos el nombre de Actividad\_2\_Punto\_1\_3, seguido de esto pasamos a la explicación de la clase llamada nodo, esta misma representara los nodos individuales de nuestra lista enlazada, nos damos cuenta que cada noto tendrá un valor entero el cual le asignamos la variable valor y una referencia al siguiente nodo la cual definimos con la variable llamada siguiente, podemos evidenciar el constructor de la clase la cual se utiliza para poder crear un nuevo nodo y establecer sus valores.

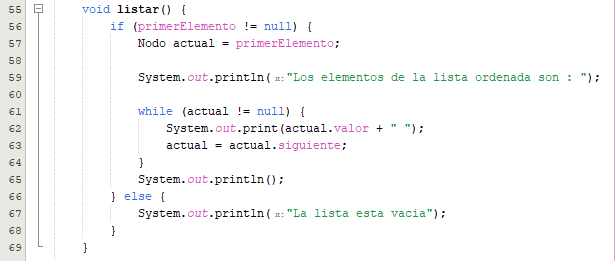


En esta siguiente sección se define la clase ListaEnlazadaAlgoritmos Ordenamieto la cual representa en si la lista enlazada, esta tiene un campo llamado primerElemento la cual es una referencia de nuestro primer nodo en la lista, como tal ese primer elemento se establecerá como nulo (null), lo que obviamente nos va a indicar que la lista se encuentra vacía.

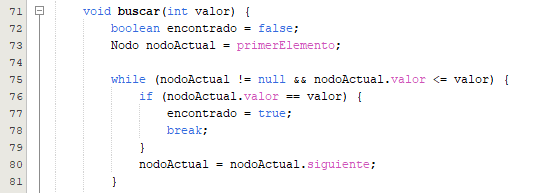


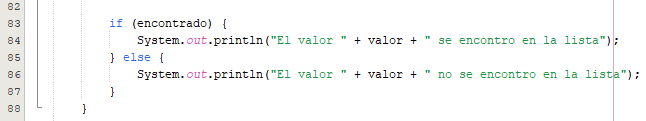


En este método agregar lo usamos para poder insertar un nuevo elemento en nuestra lista enlazada, este recibe un valor entero como parámetro, continuando con la explicación lo primero es crear un nuevo nodo el cual le asignamos la variable nuevoNodo con el valor proporcionado y la referencia al siguiente nodo la cual está establecida como nulo (null), luego se valida si la lista está vacía mediante primerElemento == null, en caso que la lista no se encuentre vacía se va proceder a recorrer toda la lista para poder encontrar la posición correcta para poder insertar el nuevo nodo en orden ascendente, de esta manera si el primer elemento de la lista es igual al nodo actual pues se pasara a actualizar el primerElemento para que apunte al nuevo nodo creado, si esto no sucede entonces se actualizara la referencia siguiente del nodoAnterior para que pueda apuntar al nuevo nodo, por último se actualizara la referencia siguiente del nuevo nodo para que este pueda apuntar al nodo actual.

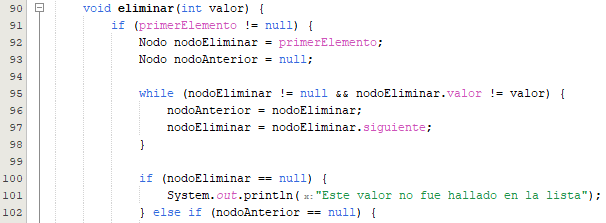


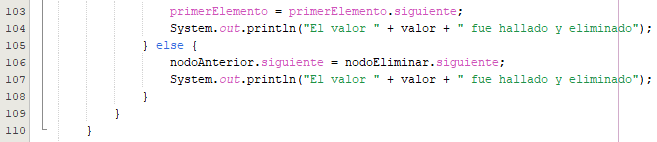
Aquí en este método listar se utilizara para poder imprimir todos los elementos que se encuentran dentro de la lista enlazada, esto se empieza verificando que la lista no se encuentra vacía mediante primerElemento != null, en tal caso que la lista no se encuentre vacía pues básicamente se va pasar a recorrer la lista desde nuestro primer nodo hasta el último nodo, así se ira imprimiendo el valor de cada uno de los nodos en la consola, en tal caso que la lista se encuentre vacía se pasara a imprimir un mensaje que indique que la lista se encuentra vacía.



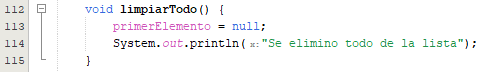


En este método buscar lo utilizamos para poder buscar un valor en nuestra lista enlazada, esta recibe un entero como parámetro, luego comienza estableciendo una bandera encontrado como false y un nodoActual como el primer nodo de la lista, luego vemos que se recorre la lista mientras el nodo actual no sea nulo y mientras que el valor del nodoActual sea menor o igual que el valor buscado, en caso que encuentre el valor se actualizara la bandera encontrado a true y pasara a interrumpirse el bucle luego al finalizar se imprime un mensaje indicando si se encontró o no el valor en la lista, luego saltaremos al método de eliminar.





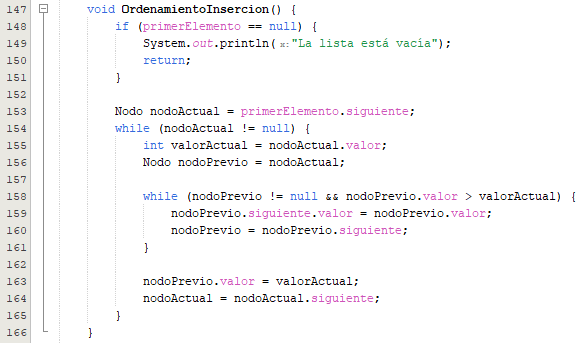
Luego en este método eliminar se utilizará para poder eliminar el valor que nosotros queremos selecciona en la lista enlazada, este recibe como parámetro un entero, comienza verificando si la lista no está vacía mediante primerElemento != null, luego se recorrerá la lista buscando el nodo con el valor que le estamos solicitando, en caso que se encuentre el valor se actualizara la referencia siguiente del nodoAnterior para poder saltar el nodo a eliminar, pero si el nodo a eliminar es el primer nodo de nuestra lista se actualizara primerElemento para que apunte al siguiente nodo, al finalizar se imprimirá un mensaje indicando que si se encuentre y se elimina el valor de la lista, ahora pasaremos a el método limpiarTodo.



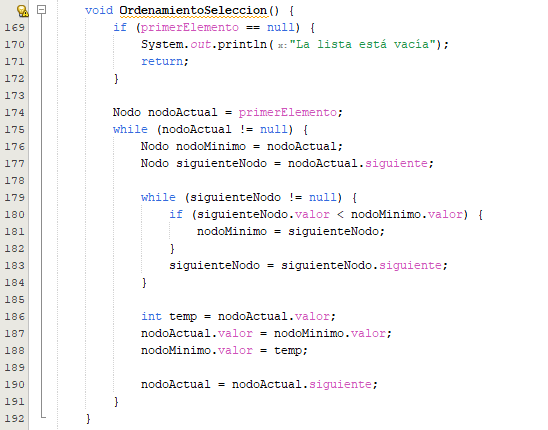
En este método limpiarTodo se utilizará para eliminar todos los elementos que se encuentren en la lista enlazada, para eso se establecerá primerElemento como nulo (null) lo que esto indica que la lista está vacía, luego se imprime un mensaje diciendo que se eliminaron todos los elementos de la lista.



Aquí hemos creado el método llamado OrdenamientoBurbuja el cual se implementa el algoritmo de ordenamiento burbuja para poder obtener los elementos de la lista enlazada, aquí se empieza validando si la lista esta vacía por medio de primerElemento == null, si la lista esta vacía se imprime un mensaje y regresa, de lo contrario utiliza un bucle do-while para poder iterar el proceso de ordenamiento hasta que no se realicen mas intercambios, dentro del bucle while para poder comparar cada par de nodos adyacentes y realizar los respectivos intercambios en caso de ser necesario, pero si el valor del nodo actual es mayor que el valor del siguiente nodo automáticamente se intercambian los valores, al finalizar este proceso se ordenar los elementos de la lista de manera ascendente.



Aquí en este método de OrdenamientoInsercion se implementa el algoritmo de ordenamiento por inserción para ordenar los elementos de la lista enlazada, se empieza verificando si la lista se encuentra vacía por medio de primerElemento == null, en caso que la lista este vacía se imprime un mensaje y regresa, en caso contrario se implementara el bucle while para pode recorrer la lista a partir del segundo nodo nodoActual == primerElemento.siguiente, dentro del bucle se guardara el valorActual del nodo actual, luego se utilizara otro bucle while para poder ubicar la posición correcta para insertar el valor actual pero en orden ascendente, los nodos anteriores se desplazaran hacia adelante para hacer espacio para el nuevo valor, luego al finalizar el valor actual se colocara en la posición correcta.

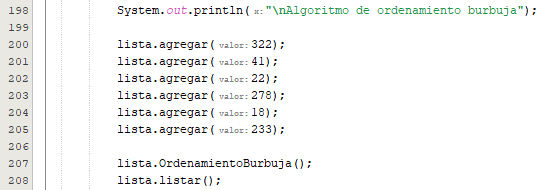


En este método de OrdenamientoSeleccion implementa el algoritmo de

ordenamiento por selección para poder ordenar los elementos de la lista enlazada, esto comienza validando como en los anteriores algoritmos validando si la lista esta vacía, de igual manera se implementa un bucle while para poder recorrer la lista y seleccionar el elemento mínimo en cada iteración, dentro del bucle se busca el nodo con el valor mínimo nodoMinimo comparando el valor del siguiente nodo con el valor mínimo actual, al finalizar esta búsqueda se intercambia el valor del nodo actual con el valor mínimo y por ultimo se mueve al siguiente nodo y se repite el proceso hasta que todos los elementos estén ordenados.

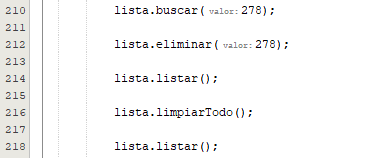


Aquí creamos una instancia de la lista y se crea un objeto de la clase ListaEnlazadaAlgoritmosOrdenamiento el cual representara la lista enlazada, esto va permitir poder utilizar los métodos y cualquier otra función definidos en esa clase, a continuación, se vamos a explicar el primer algoritmo

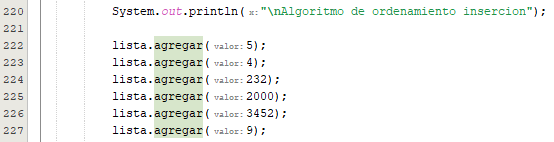


Se muestra un mensaje por consola diciendo que se utilizara el algoritmo de

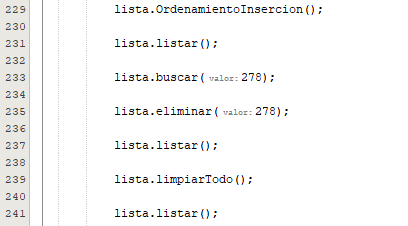
Ordenamiento burbuja, luego se agregarán varios elementos a nuestra lista utilizando el método de agregar, después de esto se llama al método OrdenamientoBurbuja para poder ordenar los elementos de la lista utilizando este algoritmo, finalmente se llamará al método listar para poder mostrar los elementos ordenados en la lista, adicional utilizamos algunas operaciones básicas que se muestran a continuación.



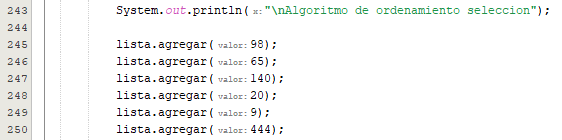
Mediante estas operaciones se puede llamar al método buscar para que este busque el valor que le hemos pasado el cual es 278 en la lista, luego llamamos al método de eliminar para quitar este elemento de nuestra lista, seguido utilizamos el método listar para poder nuevamente obtener la lista y que se muestre la nueva lista actualizada después de haber eliminado el elemento 278, por ultimo utilizamos el método limpiarTodo para poder vaciar la lista y por ultimo empleamos nuevamente el método de listar y se puede evidenciar la lista vacía.



Aquí podemos se muestra un mensaje por consola diciendo que se utilizara el algoritmo de OrdenamientoInsercion, luego se agregarán varios elementos a nuestra lista utilizando el método de agregar, después de esto se llama al método OrdenamientoInsercion para poder ordenar los elementos de la lista utilizando este algoritmo, finalmente se llamará al método listar para poder mostrar los elementos ordenados en la lista, adicional utilizamos algunas operaciones básicas que se muestran a continuación.

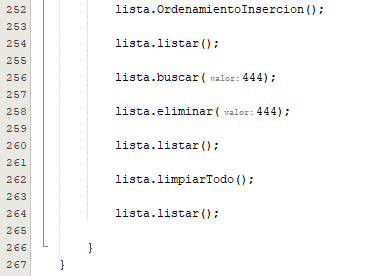


Mediante estas operaciones se puede llamar al método buscar para que este busque el valor que le hemos pasado el cual es 278 en la lista, luego llamamos al método de eliminar para quitar este elemento de nuestra lista, seguido utilizamos el método listar para poder nuevamente obtener la lista y que se muestre la nueva lista actualizada después de haber eliminado el elemento 278, por ultimo utilizamos el método limpiarTodo para poder vaciar la lista y por ultimo empleamos nuevamente el método de listar y se puede evidenciar la lista vacía. (En este caso nos va a imprimir que no hallo el elemento y por lo tanto no lo va a eliminar lo que va suceder es que básicamente nos va mostrar la misma lista ya que no pudo hallar ni eliminar nada, obviamente al final va limpiar toda la lista y esta quedara vacía, mas adelante en las imágenes se vera la impresión en consola.



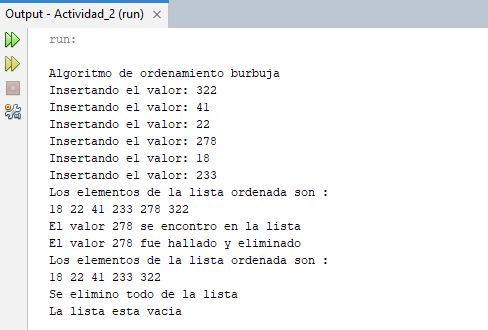
Se muestra un mensaje por consola diciendo que se utilizara el algoritmo de

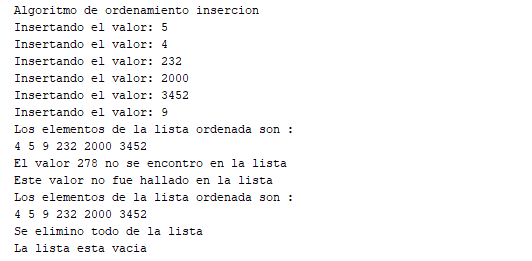
Ordenamiento selección, luego se agregarán varios elementos a nuestra lista utilizando el método de agregar, después de esto se llama al método OrdenamientoSeleccion para poder ordenar los elementos de la lista utilizando este algoritmo, finalmente se llamará al método listar para poder mostrar los elementos ordenados en la lista, adicional utilizamos algunas operaciones básicas que se muestran a continuación.

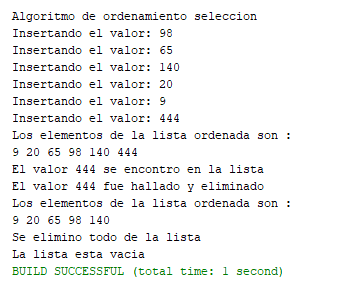


Mediante estas operaciones se puede llamar al método buscar para que este busque el valor que le hemos pasado el cual es 444 en la lista, luego llamamos al método de eliminar para quitar este elemento de nuestra lista, seguido utilizamos el método listar para poder nuevamente obtener la lista y que se muestre la nueva lista actualizada después de haber eliminado el elemento 444, por ultimo utilizamos el método limpiarTodo para poder vaciar la lista y por ultimo empleamos nuevamente el método de listar y se puede evidenciar la lista vacía, a continuación se puede evidenciar la impresión en consola de todos estos anteriores algoritmos de ordenamiento:

**IMPRESIÓN EN CONSOLA DE LOS ANTERIORES ALGORITMOS DE ORDENAMIENTO**





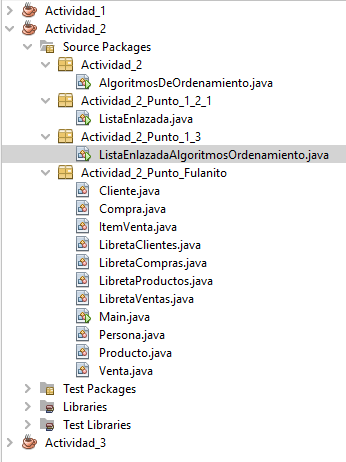


## **Ejercicios prácticos: 50% de la nota de la actividad**

Realizar un programa que permita simular el siguiente proceso solo utilizando listas personalizadas (la lista construida en el punto #1.2:

Una persona llamada Fulanito de Tal realiza una actividad comercial basada en la compra y venta de productos.

1. Fulanito anota en una libreta información importante por cada producto que compra, no sin antes revisar si dicho producto ya había sido comprando (anotado) anteriormente, algunos de los datos son: código, nombre, marca, color, precio de compra, precio de venta, porcentaje de descuento máximo, unidades en existencia, métricas de medida (kg, litros, metros, unidad) y categoría.
2. Fulanito anota en otros libreta información importante sobre cada compra que hace por cada producto: consecutivo, fecha, proveedor al que le compró, el código del producto que ha comprado, el precio que lo ha comprado, las cantidades (kg, litros, metros, unidades) y realiza el cálculo de la cantidad de dinero que debe pagar por esa compra, para eso tiene en cuenta el impuesto del IVA que es del 19% para todos los productos, anotando el valor a pagar sin IVA, el valor a pagar por concepto de IVA y el valor total a pagar.
3. Por cada compra que Fulanito realiza toma la libreta donde tiene anotados los productos que ha comprado, busca el producto por su código para actualizar el precio de compra, el precio de venta (el 40% del precio de compra), el porcentaje máximo de descuento y las unidades de existencia (suma las unidades compradas a las unidades actuales)
4. Fulanito inicia su proceso de mercadeo de sus productos con el fin de obtener personas (Clientes) interesados en comprarle. Para evitar publicitar productos que no tiene en su inventario, Fulanito toma como base la información que ha ido anotando en sus libretas, entonces, busca la información anotada previamente en sus libretas para consultar los productos cuyas unidades de existencias tienen un valor mayo que 0. Luego de haber hecho ese filtro de datos, los ordena alfabéticamente y los agrupa por categorías, consignando la información en un archivo el cual envía a una empresa de publicidad para que hagan difusión de esa información de forma masiva.
5. Fulanito posee una libreta donde anota todos los clientes que ha conseguido, algunos son solo referidos (no han comprado productos aun), en esa libreta va anotando la cédula, el nombre y los dos apellidos, el género, la fecha de nacimiento, el número telefónico, el email y la dirección, pero también anota el estado de cada cliente como VIP (los que ha comprado algo) (y como Referido (a los que no)
6. Fulanito toma esos datos para enviar información de los productos a sus clientes por medio de mensajes de texto, correos electrónicos, folletos de impresos o llamadas telefónicas.
7. Cuanto un cliente está interesado en comprar algún producto, Fulanito toma otra libreta y anota la siguiente información: consecutivo, fecha, cédula del cliente, medio de pago, modalidad (directa o domicilio). Por cada artículo a comprar Fulanito anota: código del producto, precio de venta, cantidad a comprar según las unidades de medida, el valor de descuento aplicado. Luego por cada producto realiza el cálculo de la cantidad de dinero que debe cobrar por esa venta, teniendo en cuenta el impuesto del IVA, anotando el resultado de los siguientes cálculos: El valor a cobrar sin IVA, el valor que debe descontar, el valor a cobrar por IVA, y el valor total a cobrar.
8. En caso de que la venta se realice, Fulanito consulta la libreta de ventas para buscar el Código de esa venta y modifica al estado de la venta para cambiarle el valor por éxito, pero, si la venta no se realiza, entonces, Fulanito cambia el estado a cancelada y agrega un breve motivo.
9. Luego de actualizar la venta, Fulanito toma la libreta de productos para buscar cada uno de los productos vendidos para disminuir las cantidades existentes de acuerdo con las cantidades vendidas.
10. Por último, Fulanito toma la libreta de clientes para cambiarle pasar el cliente a VIP en caso de que esta sea su primera compra.
11. Al finalizar cada venta, Fulanito revisa la información de los productos mostrando sólo aquellos cuya existencia es inferior a 5 (según la unidad de medida) y toma es información para realizar compras y abastecerse su inventario con eso artículos.
12. Con el objeto de motivar a los clientes y aumentar las ventas, Fulanito revisa la libreta de ventas para buscar cuales son los clientes que más compras han hecho (superior al promedio de compras) para informarles que tienen un 10% de descuento en la próxima vente
13. Los clientes que han comprado poco (menos del promedio de compras) les informa que tienen un 15% en su próxima compra
14. Y a los clientes que no han comprado les informa que tienen un 25% en su primera compra
15. Por último, Fulanito consulta la información que tiene en sus libretas para poder calcular el balance financiero de su negocio, pues le interese conocer la siguiente información:
    * Total, de dinero invertido en los productos que ha comprado desde el inicio.
    * Total, de dinero que ha recuperado en ventas desde el inicio
    * Total, de dinero ganancia que ha obtenido desde el inicio
    * Total, de dinero en descuento que ha generado desde el inicio
    * Total, de dinero que ha pagado como impuesto IVA
16. Para saber en qué productos debe invertir más en las próximas compras, Fulanito consulta los productos que superan el 70% de las ventas.
17. Los productos que no superan el 70% de las ventas les aplica un 35% de descuento



Link Git Hub