



Instituto tecnológico de Orizaba

Sistemas Computacionales

Estructura de Datos

Unidad 6

Profesora: María Jacinta Martínez Castillo

Integrantes del equipo:

Moran De la Cruz Aziel 21011006

Jiménez Jiménez Carlos Yael 21010975

Sebastián Brito García 21010929

Bandala Hernández Sebastián 21010921

Introducción

Una lista enlazada es una estructura de datos fundamental en ciencias de la computación. Se utiliza para almacenar y organizar una colección de elementos de manera secuencial. A diferencia de otros tipos de estructuras de datos, como los arreglos o matrices, una lista enlazada no requiere un bloque contiguo de memoria para almacenar sus elementos.

En una lista enlazada, cada elemento, llamado nodo, consta de dos partes principales: el valor del elemento en sí y un puntero que apunta al siguiente nodo en la secuencia. Esta conexión entre nodos es lo que permite que la lista enlazada se mantenga ordenada y accesible.

El primer nodo de la lista se conoce como el nodo cabeza, mientras que el último nodo de la lista apunta a un valor especial llamado nulo o null, indicando el final de la secuencia.

La principal ventaja de las listas enlazadas es que permiten la inserción y eliminación eficiente de elementos en cualquier posición, ya que solo es necesario actualizar los punteros de los nodos afectados. Sin embargo, su acceso secuencial puede ser menos eficiente que en estructuras de datos basadas en arreglos, ya que no ofrecen acceso directo a elementos en una posición específica.

Existen diferentes tipos de listas enlazadas, como las listas simplemente enlazadas, donde cada nodo tiene un puntero que apunta al siguiente nodo; las listas doblemente enlazadas, en las que cada nodo tiene un puntero al siguiente nodo y otro al nodo anterior; y las listas circulares enlazadas, en las que el último nodo apunta al primer nodo, creando un ciclo.

Las listas enlazadas son ampliamente utilizadas en la implementación de otras estructuras de datos más complejas, como pilas, colas y grafos. También se

utilizan en aplicaciones donde se requiere una manipulación dinámica de datos y se desconoce la cantidad exacta de elementos que se almacenarán.

Competencia específica:

Con base a los temas que hemos visto en clase, aprendemos y elaboramos nuestros programas con los diferentes tipos de estructuras de datos que pudimos ver, esto con el fin de almacenar diferentes datos que el usuario puede dar a una lavadora, precios y sus tipos de marcas, nosotros como programadores debemos almacenarlos en nodos.

Marco Teórico:

Listas Enlazadas: Las listas enlazadas son estructuras de datos dinámicos que permiten almacenar y organizar elementos de forma secuencial. En lugar de utilizar una ubicación contigua en memoria como en los arreglos, las listas enlazadas están compuestas por nodos enlazados entre sí. Cada nodo contiene un elemento de datos y una referencia al siguiente nodo en la lista.

La ventaja principal de las listas enlazadas es su capacidad para permitir la inserción y eliminación eficiente de elementos en cualquier posición de la lista, a diferencia de los arreglos donde se requiere un desplazamiento de elementos. Esto se debe a que los nodos enlazados pueden modificarse enlazando o desenlazando sus referencias, sin afectar al resto de la lista.

Nodos: Los nodos son las unidades básicas de una lista enlazada. Cada nodo contiene un elemento de datos y una referencia al siguiente nodo en la lista. Además, en el caso de las listas enlazadas doblemente enlazadas, también tiene una referencia al nodo anterior.

Los nodos actúan como contenedores para almacenar y organizar los datos en la lista enlazada. Cada nodo puede ser accedido a través de una referencia, lo que permite acceder a su elemento de datos ya su referencia al siguiente (y anterior) nodo. Al insertar o eliminar elementos en la lista, es necesario ajustar las referencias entre los nodos para mantener la estructura coherente.

Herencia: La herencia es un concepto fundamental de la programación orientada a objetos (POO). Permite crear jerarquías de clases, donde las clases derivadas (subclases) heredan propiedades y comportamientos de una clase base (superclase). La herencia permite reutilizar código, facilitar la extensibilidad y promover la coherencia en la estructura del programa.

Materiales utilizados

Los materiales utilizados para esta práctica son:

Computadora

NetBeans 8.2

Desarrollo

Para empezar el programa como está hecho en base a herencia necesitamos crear una clase padre llamada Electrodomésticos que forma parte del paquete Clase Padre.

Declaración de variables:

Folio: Es una variable estática de tipo entero que se inicializa con el valor 100. Al ser estática, pertenece a la clase en lugar de a instancias individuales de la clase.

Tipo: Es una variable protegida (accesible desde la clase y sus subclases) de tipo String que representa el tipo de electrodoméstico.

Marca: Es una variable protegida de tipo String que representa la marca del electrodoméstico.

Potencia: Es una variable protegida de tipo double que almacena la potencia del electrodoméstico en kilovatios por hora (kW/h).

Constructores:

Electrodomesticos(): Es un constructor vacío (sin parámetros) de la clase Electrodomesticos.

Electrodomesticos(String tipo, String marca, double potencia): Es un constructor que toma tres parámetros (tipo, marca y potencia) y los utiliza para inicializar las variables correspondientes. También incrementa el valor de folio en 1.

Métodos de acceso:

getTipo(): Devuelve el valor de la variable tipo.

setTipo(String tipo): Establece el valor de la variable tipo según el parámetro proporcionado.

getMarca(): Devuelve el valor de la variable marca.

setMarca(String marca): Establece el valor de la variable marca según el parámetro proporcionado.

getPotencia(): Devuelve el valor de la variable potencia.

setPotencia(double potencia): Establece el valor de la variable potencia según el parámetro proporcionado.

getFolio(): Devuelve el valor de la variable estática folio.

Métodos de cálculo:

calcularConsumo(int horas): Calcula el consumo de energía en kilovatios por hora (kW/h) basado en el número de horas proporcionado como parámetro y la potencia del electrodoméstico. Devuelve el resultado del cálculo.

calcularCosteConsumo(int horas, double costeHora): Calcula el costo del consumo de energía basado en el número de horas y el costo por hora proporcionados como parámetros, junto con la potencia del electrodoméstico. Devuelve el resultado del cálculo.

Método toString():

Sobrescribe el método toString() heredado de la clase Object para proporcionar una representación en forma de cadena de la instancia de Electrodomesticos. Devuelve una cadena que incluye el valor de folio, tipo, marca y potencia del electrodoméstico.

En resumen, esta clase Electrodomesticos es una representación básica de un electrodoméstico con sus atributos y métodos asociados, como obtener y establecer valores, calcular el consumo de energía y el costo, y una representación en forma de cadena de texto.

Ahora en la clase hija llamada Lavadora que extiende a la clase Electrodomésticos del paquete ClasePadre.

Declaración de variables adicionales:

precio: Es una variable privada de tipo double que representa el precio de la lavadora.

aguaCaliente: Es una variable privada de tipo booleano que indica si la lavadora tiene la opción de agua caliente.

horas: Es una variable privada de tipo entero que representa el número de horas de uso de la lavadora.

costeHora: Es una variable privada de tipo double que representa el costo por hora de uso de la lavadora.

Constructores:

Lavadora(): Es un constructor vacío (sin parámetros) de la clase Lavadora. Llama al constructor vacío de la clase padre (Electrodomesticos) y luego incrementa el valor de folio en 1.

Lavadora(String marca, double potencia): Es un constructor que toma dos parámetros (marca y potencia) y los utiliza para inicializar las variables correspondientes. Establece el tipo como "Lavadora" y la opción de agua caliente como false por defecto.

Lavadora(String marca, double precio, double potencia, boolean aguaCaliente): Es un constructor que toma cuatro parámetros (marca, precio, potencia y aguaCaliente) y los utiliza para inicializar las variables correspondientes.

Métodos de acceso adicionales:

setPrecio(double precio): Establece el valor de la variable precio según el parámetro proporcionado.

getPrecio(): Devuelve el valor de la variable precio.

setAguaCaliente(boolean aguaCaliente): Establece el valor de la variable aguaCaliente según el parámetro proporcionado.

isAguaCaliente(): Devuelve el valor de la variable aguaCaliente.

setHoras(int horas): Establece el valor de la variable horas según el parámetro proporcionado.

getHoras(): Devuelve el valor de la variable horas.

setCosteHora(double costeHora): Establece el valor de la variable costeHora según el parámetro proporcionado.

getCosteHora(): Devuelve el valor de la variable costeHora.

Método toString():

Sobrescribe el método toString() heredado de la clase Electrodomésticos para proporcionar una representación en forma de cadena de la instancia de Lavadora. Además de los detalles heredados de la clase padre, esta representación incluye el precio, si la lavadora tiene la opción de agua caliente, el consumo de energía y el costo del consumo.

Método calcularConsumo(int horas):

Sobrescribe el método calcularConsumo() de la clase padre Electrodomésticos. Calcula el consumo de energía en kilovatios por hora (kW/h) basado en el número de horas proporcionado como parámetro, considerando la potencia de la lavadora y si tiene la opción de agua caliente. Si tiene agua caliente, se suma un 20% adicional a la potencia. Devuelve el resultado del cálculo.

Como es un programa de herencia que utiliza las listas enlazadas necesitamos crear una clase Nodo que se utiliza en el paquete EntradaSalida.

Declaración de variables:

info: Es una variable pública de tipo genérico T que almacena la información contenida en el nodo.

sig: Es una variable pública de tipo `Nodo<T>` que representa el enlace al siguiente nodo en una estructura de datos.

Constructor:

`Nodo(T info)`: Es un constructor que toma un parámetro info y lo utiliza para inicializar la variable info. El enlace sig se establece como null.

Métodos de acceso:

`getInfo()`: Devuelve el valor de la variable info.

`setInfo(T info)`: Establece el valor de la variable info según el parámetro proporcionado.

`getSig()`: Devuelve el enlace al siguiente nodo, es decir, el valor de la variable sig.

`setSig(Nodo<T> sig)`: Establece el enlace al siguiente nodo según el parámetro proporcionado.

La clase Nodo se utiliza comúnmente en estructuras de datos como listas enlazadas, donde cada nodo contiene un elemento de datos y un enlace al siguiente nodo en la secuencia. En este caso, la clase Nodo se extiende a partir de la clase `Electrodomesticos` del paquete `ClasePadre`, lo que significa que los objetos creados a partir de esta clase también heredan los atributos y métodos de la clase `Electrodomesticos`.

Ahora necesitamos una clase donde se hara todas las operaciones del Nodo, es una clase llamada DatosDesordenados en el paquete EntradaSalida.

Declaración de variables:

inicio: Es una variable privada de tipo genérico `Nodo<T>` que representa el inicio de la lista enlazada.

f: Es una variable privada de tipo genérico `Nodo<T>` que representa el final de la lista enlazada.

Constructor:

`DatosDesordenados()`: Es el constructor de la clase. Inicializa el inicio de la lista enlazada como null.

Métodos:

`ListaVacia()`: Verifica si la lista enlazada está vacía. Devuelve true si el inicio es null, lo que indica que la lista está vacía.

`insertFrente(T dato)`: Inserta un nuevo nodo con el dato proporcionado al frente de la lista enlazada. Si la lista está vacía, el nodo insertado se convierte en el inicio y el final de la lista. Si la lista no está vacía, se inserta el nodo al frente y se actualiza el enlace del nodo anterior.

`imprimeLista()`: Genera una representación en forma de cadena de la lista enlazada. Recorre la lista enlazada desde el inicio hasta el final, concatenando los valores de cada nodo en la cadena de salida.

`insertFinal(T dato)`: Inserta un nuevo nodo con el dato proporcionado al final de la lista enlazada. Si la lista está vacía, el nodo insertado se convierte en el inicio. Si

la lista no está vacía, se inserta el nodo al final y se actualiza el enlace del nodo anterior al nuevo nodo.

`eliminaLista(Nodo<T> pos)`: Elimina el nodo proporcionado de la lista enlazada. Si el nodo a eliminar es el inicio, se actualiza el inicio. Si el nodo a eliminar es el final, se actualiza el final. En otros casos, se actualiza el enlace del nodo anterior al nodo siguiente y se libera la memoria del nodo a eliminar.

`dirAntes(Nodo<T> pos)`: Devuelve el nodo anterior al nodo proporcionado en la lista enlazada. Recorre la lista enlazada desde el inicio hasta encontrar el nodo cuyo enlace siguiente sea igual al nodo proporcionado.

`modificar(Nodo aux)`: Modifica el valor de información del nodo proporcionado con un nuevo dato. Se utiliza la clase `Tools` para leer el nuevo dato.

`busSecLista(int dato)`: Realiza una búsqueda secuencial en la lista enlazada para encontrar un nodo cuyo atributo folio sea igual al dato proporcionado. Devuelve el primer nodo encontrado que cumple con esta condición.

Necesitamos un menú donde se ejecute nuestro programa para ello creamos una clase llamada `MenuElectrodomesticos` en el paquete `EntradaSalida`.

|

importaciones:

`javax.swing.JOptionPane`: Importa la clase `JOptionPane` para mostrar ventanas de diálogo.

Método `main`:

`main(String[] args)`: Este método es el punto de entrada de la aplicación. Llama al método `listasElectro()` para iniciar el menú de electrodomésticos.

Método `listasElectro()`:

Este método maneja el menú de opciones para interactuar con la lista de electrodomésticos.

Utiliza un bucle do-while para repetir el menú hasta que el usuario elija la opción "Salir".

Dentro del bucle, se utiliza un bloque switch para realizar acciones basadas en la opción seleccionada por el usuario.

Las opciones disponibles son:

"FRENTE": Inserta un nuevo electrodoméstico al frente de la lista llamando al método `insertFrente()` de la clase `DatosDesordenados`. Luego muestra los datos actualizados de la lista.

"FINAL": Inserta un nuevo electrodoméstico al final de la lista llamando al método `insertFinal()` de la clase `DatosDesordenados`. Luego muestra los datos actualizados de la lista.

"BUSCAR": Permite buscar un electrodoméstico en la lista por su folio. Solicita al usuario que ingrese el folio y luego llama al método `busSecLista()` de la clase `DatosDesordenados` para buscar el electrodoméstico. Muestra los datos del electrodoméstico encontrado si existe.

"ELIMINAR": Permite eliminar un electrodoméstico de la lista por su folio. Solicita al usuario que ingrese el folio del electrodoméstico a eliminar y luego llama al método `eliminaLista()` de la clase `DatosDesordenados` para eliminarlo de la lista.

"MODIFICAR": Permite modificar los atributos de un electrodoméstico en la lista por su folio. Solicita al usuario que ingrese el folio del electrodoméstico a modificar y luego muestra un submenú para seleccionar qué atributo modificar (marca, potencia o precio). Luego de realizar la modificación, muestra los datos actualizados de la lista.

"IMPRIMIR": Imprime los datos de la lista enlazada llamando al método `imprimeLista()` de la clase `DatosDesordenados`.

El bucle continúa hasta que el usuario seleccione la opción "SALIR".

Método Datos():

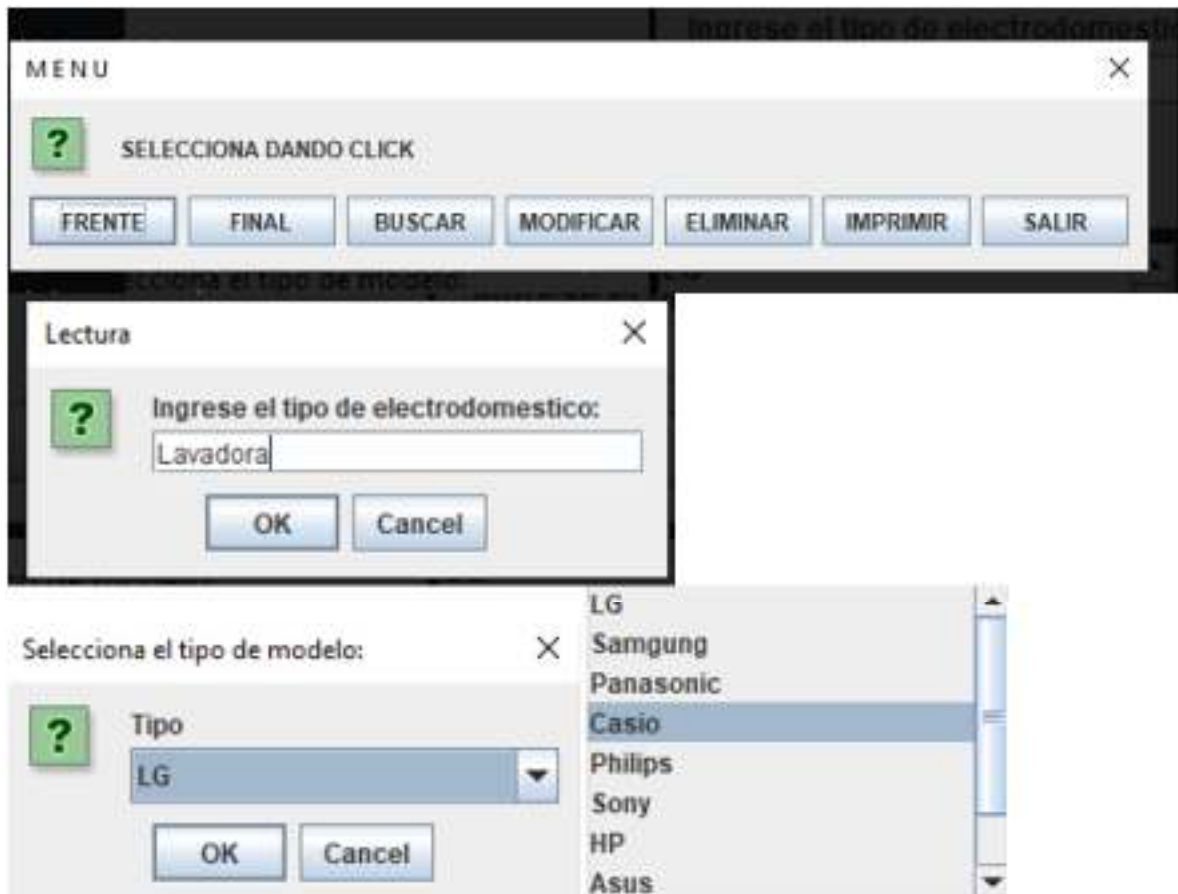
Este método se utiliza para solicitar y crear un objeto Lavadora con los datos ingresados por el usuario.

Utiliza métodos de la clase Tools para leer y validar los valores de los atributos del electrodoméstico.

Retorna el objeto Lavadora creado con los datos ingresados.

En resumen, esta clase MenuElectrodomesticos implementa un menú interactivo para realizar diversas operaciones (agregar, buscar, eliminar, modificar, imprimir) en una lista enlazada de electrodomésticos de tipo Lavadora, utilizando la clase DatosDesordenados para almacenar y manipular los datos con sus respectivas Tools.

Ejecución del programa




Lectura ✕

 Potencia:

75


OK Cancel

Lectura ✕

 ¿Ocupas agua caliente?

Yes No

Lectura ✕

 Precio:

4567

OK Cancel


Lectura ✕

 Horas:

7

OK Cancel

Lectura ✕

 Costo por hora:

9

OK Cancel


Salida ✕

 Datos:
Electrodomesticos (Folio= 191, Tipo= Lavadora, Marca= LG, Potencia= 75.0 kWh, Precio= \$4567.0, Agua Caliente= Si, Consumo= 288.0, Costo del consumo= \$1200.0) ->

OK

Buscar:

Lectura [X]

 Ingresa el folio que desees buscar:

Solista [X]

 Datos Encontrados:


Electrodomésticos [Folio= 101, Tipo= Lavadora, Marca= LG, Potencia= 75.0 KWts, Precio= \$4567.0, Agua Caliente= true, Consumo= 390.0, Costo del consumo= \$1200.0]

Modificar:


Lectura [X]

 Ingrese el número de folio a modificar:

MENU [X]

 SELECCIONA EL APARTADO QUE DESEE S MODIFICAR

Solista [X]

 Datos Actualizados:

Electrodomésticos [Folio= 101, Tipo= Lavadora, Marca= Panasonic, Potencia= 42.0 KWts, Precio= \$8888.0, Agua Caliente= true, Consumo= 201.6, Costo del consumo= \$672.0,->

Eliminar:

Lectura

?

Ingresa el folio que desees eliminar:

101

OK

Cancel

Salida

?

Lista Vacía

OK

Conclusión:

En conclusión el programa que fue elaborado en este reporte se lleva a cabo de diferentes temas que hemos aprendido en estas unidades para después juntarlos y crearlo de esta manera, debemos de tomar en cuenta esto ya que los subtemas de listas enlazadas, nodos y herencia son elementos fundamentales en la programación y son ampliamente utilizados en el desarrollo de aplicaciones en Java.

Las listas enlazadas proporcionaron una estructura de datos dinámica y eficiente para almacenar y organizar elementos en secuencia. Su capacidad para permitir la inserción y eliminación eficiente de elementos en cualquier posición las hace muy útiles en situaciones donde se requiere una manipulación frecuente de datos.

Los nodos son los componentes básicos de las listas enlazadas, y cada nodo contiene un elemento de datos y una referencia al siguiente nodo. Los nodos permiten enlazar los elementos en una lista, lo que facilita el acceso y la gestión de los datos. Además, en las listas enlazadas doblemente enlazadas, los nodos también tienen una referencia al nodo anterior, lo que permite una navegación bidireccional.

La herencia es un mecanismo poderoso que permite crear jerarquías de clases en las que las subclases heredan propiedades y comportamientos de una clase base. Esto promueve la reutilización de código y facilita la extensibilidad del programa. En el contexto de un programa de Java, la herencia se utiliza para modelar relaciones entre objetos y para agregar características adicionales a través de las subclases.