Manual técnico del programa

Proyecto #1

Estructuras de datos

Integrantes:

Oscar Zamora Ramírez

Esteban Rojas Herrera

Sebastián José Brenes Murillo

**Clase Ferretería**

La clase ferretería tiene un atributo que se llama listaSección. Los métodos de esta clase utilizan la delegación para llamar los métodos a usar y envían varios parámetros a dichos métodos. Esto para poder realizar las funciones necesarias.

**Método agregarSección:** recibe dos parámetros: 1 parámetro Dato de objeto Sección y 1 parámetro entero Opción. Delega al método tipoInserción de listaSeccion y le envía estos dos parámetros.

**Método agregarProductoSección:** recibe 3 tres parámetros: 1 parámetro int numero, 1 parámetro int opc y 1 parámetro Dato de objeto pilaProdcuto. Delega al método agregarProducto de listaSeccion y le envía estos 3 parámetros.

**Método buscarSección:** recibe 1 parámetro int numero. Llama al método buscarSeccion de listaSeccion y le envía el parámetro.

**Método eliminarSección:** recibe 1 parámetro int numero. Llama al método eliminarSeccion de listaSeccion y le envía el parámetro.

**Método eliminarProducto:** recibe 1 parámetro int numero y 1 parámetro string nombre. Llama al método eliminarProducto de listaSeccion y le envía los dos parámetros.

**Método mostrarSecciones:** Llama al método toString de listaSeccion.

**Método mostrarProductoSeccion:** recibe 1 parámetro int numero. Llama al método mostrarProductoSeccion y le envía el parámetro.

**Método agregarDatoCarrito:** regresa un objeto Producto. Recibe 2 parámetros: 1 parámetro int numero y 1 parámetro string nombre. Llama al método agregarDatoCarrito de listaSeccion y le envía los dos parámetros.

**Método cargarListaSeccion:** recibe 1 parámetro string archivoListaSeccion. Delega al método cargarListaSeccion de listaSeccion y le envía el parámetro.

**Método nombreSeccion:** llama al método getNombre de listaSeccion.

**Método guardarListaSeccion:** recibe 1 parámetro string nombreArchivo. Delega al método guardarListaSeccion y le envía el parámetro.

**Método guardarListaCategoria:** delega al método guardarListaCategoria de listaSeccion.

**Método guardarArchivosPila:** delega al método guardarArchivoPila de listaSeccion.

**Clase listaSucursal**

**Método agregarDato:** Agrega una nueva sucursal a la ferreteria. Recibe 1 parámetro Dato de objeto Ferreteria que se envía al nodoFerretería para almacenarlo.

**Método eliminarElemento:** Borra una sucursal usando el id. Recibe 1 parámetro string id. Es un método de tipo string que recorre la lista mientras no esté vacía comparando el valor del método “getId” de cada nodo con el parámetro recibido por el método. Si los dos valores son iguales elimina dicho nodo, imprime un aviso de que se la sucursal se eliminó y cambia el valor de la variable bool “encontrado” a true. Si lo valores son distintos se avanza al siguiente nodo mientras que el mismo no esté vacío y la variable “encontrado” mantenga el valor false. Si se recorre toda la lista y la variable “encontrado” mantiene el valor “false” se imprime un aviso de que no se encontró una sucursal con ese id. Si la lista está vacía se imprime un aviso de que la lista está vacía.

**Método empty:** regresa un valor NULL si el nodo está vacío.

**Método mostrarLista:** muestra las sucursales existentes. Si la lista no está vacía la recorre y, mientras que el valor del nodo actual no sea NULL, llama al método toString de la clase ferreteria. Si la lista está vacía se imprime un aviso de que la lista está vacía.

**Método buscarSucursal:** busca una sucursal usando el id. Recibe 1 parámetro string id. Se declara una variable bool llamada “encontrado” y se le asigna un valor false. Luego se empieza a recorrer la lista, siempre y cuando el valor del nodo actual no sea NULL y la variable “encontrado” siga siendo false, y se compara el valor del método getId del nodo con el parámetro recibido. Si estos valores son iguales se cambia el valor de la variable “encontrado” a true. Al final del método se retorna un valor int 1 la variable “encontrado” es igual a true indicando que se encontró la sucursal buscada, de no ser así se retorna un valor int 0.

**Método agregarSección:** agrega una nueva sección a una sucursal específica. Agrega una sección a una sucursal específica. Recibe 3 parámetros: 1 parámetro string id, un parámetro Dato de objeto Sección y 1 parámetro int opc. Se declara e inicializa la variable “encontrado” con un valor false y se empieza a recorrer la lista mientras que el nodo actual no esté vacío y la variable “encontrado” siga siendo false. En cada nodo se compara el valor del método getDato con el parámetro string id. si estos dos valores son iguales se delega al método agregarSeccion de la clase ferreteria y se le envían el parámetro int opc y el parámetro de objeto Sección.

**Método BucarSeccion:** busca una sección dentro una sucursal específica. Recibe 2 parámetros: 1 parámetro string id y 1 parámetro int numero. Se declara una variable bool llamada “encontrado” y se le asigna un valor false. Luego se empieza a recorrer la lista, siempre y cuando el valor del nodo actual no sea NULL y la variable “encontrado” siga siendo false, y se compara el valor del método getId del nodo con el parámetro recibido. También se llama al método buscarSeccion enviándole el parámetro “numero”. Si método buscarSeccion retorna un valor int 1 y los valores de getId y del parámetro id son iguales se cambia el valor de la variable “encontrado” a true. Al final del método se retorna un valor int 1 la variable “encontrado” es igual a true indicando que se encontró la sección buscada, de no ser así se retorna un valor int 0.

**Método agregarProductoSeccion:** agrega un nuevo producto a una sección que está dentro de una sucursal específica. Recibe 4 parámetros: 1 parámetro string id, 1 parámetro int numero, 1 parámetro int opc y 1 parámetro de objeto pilaProducto. Se declara una variable bool llamada “encontrado” se inicializa en false. Luego se establece un ciclo que recorre la lista siempre que el nodo actual sea diferente de NULL y la variable “encontrado” tenga un valor false. Dentro del ciclo una condicional compara el valor del método getId del nodo actual con el parámetro id para comprobar si es la sucursal correcta. Si los valores son iguales se cambia el valor de “encontrado” a true y se delega al método agregarProductoSecccion de la clase ferreteria y se le envía los parámetros numero, Dato y opc.

**Método eliminarSeccion:** elimina una sección dentro de una sucursal específica. Recibe 2 parámetros: 1 parámetro string id y 1 parámetro int numero. Se declara una variable bool llamada “encontrado” y se inicializa con un valor false. Luego se inicia un ciclo que recorra la lista siempre que el valor del nodo actual sea diferente de NULL y que el valor de la variable “encontrado” sea false. Dentro del ciclo una condicional compara el valor del método getId del nodo actual con el parámetro id, si son iguales el valor de “encontrado” se cambia por true y se llama al método eliminarSeccion de la clase ferreteria enviándole el parámetro numero y el valor retornado se almacena en una variable int “x”. Cuando el ciclo se rompe se retorna el valor de x.

**Método eliminarProducto:** elimina un producto perteneciente a una sección que se encuentra dentro de una sucursal específica. Recibe 3 parámetros: 1 parámetro int numero, 1 parámetro string id y 1 parámetro string nombre. Se declara una variable bool llamada “encontrado” y se inicializa con un valor false. Luego se inicia un ciclo que recorra la lista siempre que el valor del nodo actual sea diferente de NULL y que el valor de la variable “encontrado” sea false. Dentro del ciclo una condicional compara el valor del método getId del nodo actual con el parámetro “id”, si son iguales se cambia el valor de la variable “encontrado” a true y se llama al método eliminarProducto de la clase ferreteria y se le envían los parámetros “numero” y “nombre”. El valor retornado por el método eliminarProducto se almacena en una variable int “x” que se retorna una vez el ciclo se rompe.

**Método mostrarSecciones:** muestra las secciones existentes dentro de una sucursal. Recibe 1 parámetro string id. Se declara una variable bool llamada “encontrado” y se inicializa con un valor false. Luego se inicia un ciclo que recorra la lista siempre que el valor del nodo actual sea diferente de NULL y que el valor de la variable “encontrado” sea false. Dentro del ciclo una condicional compara el valor del método getId del nodo actual con el parámetro id. Si son iguales se cambia el valor de “encontrado” a true y se llama al método mostrarSecciones. El valor retornado por dicho método se almacena en una variable string “x” que se retornará una vez finalice el ciclo. Si al finalizar el ciclo el valor de “encontrado” sigue siendo false la variable “x” almacenará un aviso de que no existe una sucursal con ese id.

**Método mostrarProductosSeccion:** muestra los productos existentes en una sección almacenada en una sucursal específica. Recibe 2 parámetros: 1 parámetro string id y 1 parámetro int numero. Se declara una variable bool llamada “encontrado” y se inicializa con un valor false. Luego se inicia un ciclo que recorra la lista siempre que el valor del nodo actual sea diferente de NULL y que el valor de la variable “encontrado” sea false. Dentro del ciclo una condicional compara el valor del método getId del nodo actual con el parámetro id. Si los valores son iguales se cambia el valor de “encontrado” a true y llama al método mostrarProductosSeccion de la clase ferreteria y se le envía el parámetro “numero”. El valor retornado por dicha variable se almacena en una variable string “x” que se imprimirá al finalizar el ciclo. Si al finalizar el ciclo la variable “encontrado” tiene un valor false la variable “x” pasará a almacenar un aviso de que no existe una sucursal con es id.

**Método agregarDatoCarrito:** agrega un producto de una sección de una sucursal y lo coloca en el carrito de compras. Recibe 3 parámetros: 1 parámetro int numero, 1 parámetro string id y un parámetro string nombre. Se declara una variable bool llamada “encontrado” y se inicializa con un valor false, también se declara una variable de puntero “P”. Luego se inicia un ciclo que recorra la lista siempre que el valor del nodo actual sea diferente de NULL y que el valor de la variable “encontrado” sea false. Dentro del ciclo una condicional compara el valor del método getId del nodo actual con el parámetro id. Si los valores son iguales se cmabia el valor de “encontrado” a true y se llama al método agregarDatoCarrito al que se le envían los parámetros numero y nombre. El objeto producto retornado se almacena en la variable “P” y al finalizar el ciclo se retorna la variable P.

**Método cargarListaSucursal:** se coloca el ifstream Archivo para declarar a Archivo como la entrada de datos del archivo al código y luego se hace el llamado para abrir el archivo “Lista Sucursales.txt”. Después se colocan 2 condicionales para comprobar que el archivo existe. Si no existe un archivo llamado “Lista Sucursales.txt” se imprime un avis de que no se pudo abrir el archivo. Si el archivo “Lista Sucursales.txt” sí existe se inicializa un ciclo que va a funcionar mientras el archivo tenga datos.

Dentro del ciclo se declaran las variables string “id” y string “ubicación” que se usarán para almacenar los datos obtenidos de los archivos. Con un getline de recupera el valor de “id” hasta el “;” y “ubicacion” hasta el “;” de la cadena de texto. Luego se coloca Archivo.ignore para saltar a la siguiente cadena de texto y repetir el proceso. A continuación, se coloca una condicional que dicta que, si el dato guardado que se va a recuperar no es un espacio en blanco, entonces declara una variable F de objeto Ferreteria y se le guardan por parámetros las variables “id” y “ubicación” para luego delegar al método cargarListaSeccion, al que se le envía “id”, de la clase listaSeccion y delegar al método agregarDato, al que se le envía “F”, de la clase listaSucursal.

**Método guardarDatosSucursal:** se declara la variable “salidaC” como la salida de los valores del programa hacia los archivos y la variable “actual” de objeto nodoFerreteria y se iguala al valor de nodoCabeza. Luego se llama al método open de archivos para abra o cree “Lista Sucursales.txt”. Después se coloca un ciclo que funcionará mientras que la lista no esté vacía. Dentro del ciclo se llama al método getid de la clase ferreteria para que guarde el valor del id del nodo actual dentro del archivo y después le coloque un “;” como diferenciador. Los mismo se hace con el método getUbicacion de ferreteria y por último de guarda un espacio en blanco en el archivo antes de avanzar al siguiente nodo. Al final del método se llama a la función close de archivos para cerrar el archivo.

**Método guardarDatosSeccion:** se declara una variable actual de objeto nodoFerreteria y se iguala a “nodoCabeza” (una variable declarada en el .h de listaSucursal). Después coloca un ciclo que funcionará mientras “actual” sea diferente de vacío. Dentro del ciclo se delega al método guardarListaSeccion, al que se le envía el id de “actual”, de la clase listaSeccion y luego se iguala a “actual” al nodo siguiente.

**Método guardarDatosCategoria:** se declara una variable actual de objeto nodoFerreteria y se iguala a “nodoCabeza” (una variable declarada en el .h de listaSucursal). Después coloca un ciclo que funcionará mientras “actual” sea diferente de vacío. Dentro del ciclo se delega al método guardarListaCategoria de la clase listaCategoria y luego se iguala a “actual” al nodo siguiente.

**Método guardarArchivosPila:** se declara una variable actual de objeto nodoFerreteria y se iguala a “nodoCabeza” (una variable declarada en el .h de listaSucursal). Después coloca un ciclo que funcionará mientras “actual” sea diferente de vacío. Dentro del ciclo se delega al método guardarArchivosPila y luego se iguala a “actual” al nodo siguiente.

**Clase Sección**

**Método toString:** imprime el número y el nombre de la sección.

**Método agregarProducto:** recibe 2 parámetros: 1 parámetro Dato de objeto pilaProducto y 1 parámetro int opc. Delega al método tipoInsercion de la clase listaCategoria y le envía el parámetro dato y el parámetro opc.

**Método eliminarProducto:** recibe 1 parámetro string nombre. Llama al método eliminarDato de la clase listaCategoria y le envía el parámetro nombre. Retorna el valor string que él método eliminarDato regrese.

**Método mostrarProductosSeccion:** llama al método al método toString de la clase listaCategoria y retorna el valor string recibido.

**Método agregarDatoCarrito:** recibe 1 parámetro string nombre. Llama al método agregarDatoCarrito de la clase listaCategoria y le envía el parámetro nombre. Retorna el valor string enviado por el método.

**Método guardarListaCategoria:** recibe 1 parámetro string nombreArchivo. Delega al método guardarListaCategoria de la clase listaCategoria y le envía el parámetro nombreArchivo.

**Método guardarArchivoProducto:** delega al método guardarArchivoProducto de la clase listaCategoria.

**Método cargarListaCategoria:** recibe 1 parámetro string nombreArchivo. Delega al método cargarListaCategoria y le envía el parámetro nombreArchivo.

**Clase ListaSeccion**

**Método insertarDatoUltimo:** se utiliza para insertar una nueva seccion en la última posición de la lista. Recibe 1 parámetro Dato de objeto Sección. Se declara una variable “nuevo” de tipo nodoSeccion y se le asigna el parámetro Dato recibido por el método junto con dos variables nodoSeccion, “primero” y “ultimo”, que almacenan el valor del primer nodo y el ultimo nodo de la lista respectivamente. Luego una condicional evalúa la variable “primero” para comprobar si está vacía o no. Si el valor de la variable es igual a NULL entonces a las variables “primero” y “ultimo” se les asigna la variable “nuevo” como su valor y se hace que el valor del método setSig de “primero” sea “primero” y que el valor del método setAtras de primero sea “ultimo”.

Si el valor de la variable “primero” no es NULL significa que la lista no está vacía, por lo que se asigna a “nuevo” como el valor del método setSig de “ultimo”. Luego, el valor del método setAtras de “nuevo” se iguala a la variable “ultimo” y el valor del método setSig de “nuevo” se iguala a la variable “primero”. Por último, la variable “ultimo” se iguala a “nuevo” y le valor del método setAtras de “primero” se iguala a ultimo.

**Método insertarDatoPrimero:** inserta una nueva sección en la primera posición de la lista. Recibe 1 parámetro Dato de objeto Sección. Se declara una variable “nuevo” de tipo nodoSeccion y se le asigna el parámetro Dato recibido por el método junto con dos variables nodoSeccion, “primero” y “ultimo”, que almacenan el valor del primer nodo y el ultimo nodo de la lista respectivamente. Luego una condicional evalúa la variable “primero” para comprobar si está vacía o no. Si el valor de la variable es igual a NULL entonces a las variables “primero” y “ultimo” se les asigna la variable “nuevo” como su valor y se hace que el valor del método setSig de “primero” sea “primero” y que el valor del método setAtras de primero sea “ultimo”.

Si el valor de la variable “primero” no es NULL significa que la lista no está vacía, por lo que se asigna a “nuevo” como el valor del método setAtras de “primero”. Luego el valor del método setSig de “nuevo” se iguala a “primero” y el valor del método setAtras de “nuevo” se iguala a “ultimo”. Por último, se hace que el nuevo valor de “primero” sea “nuevo” y el valor del método setSig de “ultimo” se iguala a primero.

**Método toString:** imprime los datos de la sección desde el primero al último. Se crea una variable “actual” de tipo nodoSeccion y se iguala al primer valor de la lista. Si la lista está vacía se guarda en un variable string “x” un aviso de que la lista está vacía. Si la lista tiene elementos se inicia un ciclo que llama al método toString de la clase Seccion y almacena en la variable “x” los datos recibidos por el mismo; luego se avanza al siguiente nodo. Este ciclo se repite hasta que el nodo vuelva a la primera posición de la lista. Al final se retorna la variable “x”.

**Método toStringAtras:** imprime los datos de la sección desde el último al primero. Se crea una variable “actual” de tipo nodoSeccion y se iguala al último valor de la lista. Si la lista está vacía se guarda en un variable string “x” un aviso de que la lista está vacía. Si la lista tiene elementos se inicia un ciclo que llama al método toString de la clase Seccion y almacena en la variable “x” los datos recibidos por el mismo; luego se avanza al nodo anterior. Este ciclo se repite hasta que el nodo vuelva a la última posición de la lista. Al final se retorna la variable “x”.

**Método eliminarDato:** elimina una sección específica. Recibe 1 parámetro int. Se declara una variable “actual” de tipo nodoSeccion que se iguala a “primero” y una variable “anterior” de tipo nodoSeccion que se iguala a NULL. También se declara una variable bool “encontrado” que se iguala a false y una variable string “x”. Si la variable “primero” es igual a NULL se guarda un aviso de que la lista esta vacía dentro de “x”. Si “primero” es diferente a NULL se inicia un ciclo que se repetirá mientras “actual” sea distinto de NULL y “encontrado” sea igual a false.

Dentro del ciclo se compara el valor del método getNumero de la clase Sección con el parámetro numero. Si los valores son distintos, “anterior” se iguala a “actual” y “actual” avanza al siguiente nodo. Si los valores son iguales el valor de “encontrado” se iguala a true, “x” almacena el mensaje “Eliminado” y se colocan 4 condicionales:

* Si los valores de “primero” y “ultimo” son iguales: primero se iguala a NULL dejando la lista vacía.
* Si los valores de “actual” y “primero” son iguales: “primero” se iguala al siguiente nodo, el método setAtras de “primero” se iguala a “ultimo” y el método setSig de “ultimo” se iguala a “primero”.
* Si los valores de “actual” y “ultimo” son iguales: “ultimo se iguala a “anterior”, el método setSig de “ultimo” se iguala a “primero” y el método setAtras de “primero” se iguala a “ultimo”.
* Si no se cumple ninguna de las condiciones anteriores: el valor del método setSig de “anterior” se iguala al valor del método setSig de “actual” y el valor del método setAtras llamado por el método getSig de “actual” se iguala a anterior.

Al final se retorna “x”.

**Método buscarSeccion:** busca una sección específica en la lista. Recibe 1 parámetro int numero. Se declara una variable “actual” de tipo nodoSeccion que se iguala a “primero” y se declara una variable bool “encontrado” que se iguala a false. Luego se inicia un ciclo que funcionará mientras el valor de “actual” no sea NULL y el valor de “encontrado” sea false. Dentro del ciclo hay 2 condicionales:

* Si el valor del método getnumero de la clase Seccion es igual al parámetro numero se cambia el valor de “encontrado” a true.
* Si no se cumple la condición anterior: “actual” se iguala al siguiente nodo.

Al final se colocan dos condicionales que dictan que si el valor de “encontrado” es true retorne un int 1, y si no es así retorne un int 0.

**Método agregraProducto:** agrega un producto a una sección específica. Recibe 3 parámetros: 1 parámetro int numero, 1 parámetro int opc y 1 parámetro Dato de objeto pilaProducto. Se declara una variable “actual” de tipo nodoSeccion que se iguala a “primero” y se declara una variable bool “encontrado” que se iguala a false. Luego se inicia un ciclo que funcionará mientras el valor de “actual” no sea NULL y el valor de “encontrado” sea false. Dentro del ciclo hay 2 condicionales:

* Si el valor del método getnumero de la clase Seccion es igual al parámetro numero se cambia el valor de “encontrado” a true y se delega al método agregarProducto de la clase Sección a que se le envían los parámetros “Dato” y “opc”.
* Si no se cumple la condición anterior: “actual” se iguala al siguiente nodo.

**Método eliminarProducto:** elimina un producto de una sección específica. Recibe 2 parámetros: 1 parámetro string nombre y 1 parámetro int numero. Se declara una variable “actual” de tipo nodoSeccion que se iguala a “primero”, se declara una variable bool “encontrado” que se iguala a false y se declara una variable string “x”. Luego se inicia un ciclo que funcionará mientras el valor de “actual” no sea NULL y el valor de “encontrado” sea false. Dentro del ciclo hay 2 condicionales:

* Si el valor del método getnumero de la clase Seccion es igual al parámetro numero se cambia el valor de “encontrado” a true y se delega al método agregarProducto de la clase Sección al que se le envían los parámetros “Dato” y “opc”. El valor string recibido de este método se guarda en “x”.
* Si no se cumple la condición anterior: “actual” se iguala al siguiente nodo.

Al final retorna “x”.

**Método mostrarProductosSeccion:** muestra los productos guardados dentro de una sección específica. Recibe 1 parámetro int numero. Se declara una variable “actual” de tipo nodoSeccion que se iguala a “primero”, se declara una variable bool “encontrado” que se iguala a false y se declara una variable string “x”. Luego se inicia un ciclo que funcionará mientras el valor de “actual” no sea NULL y el valor de “encontrado” sea false. Dentro del ciclo hay 2 condicionales:

* Si el valor del método getnumero de la clase Seccion es igual al parámetro numero se cambia el valor de “encontrado” a true y se delega al método mostrarProductosSeccion de la clase Sección. El valor string recibido de este método se guarda en “x”.
* Si no se cumple la condición anterior: “actual” se iguala al siguiente nodo.

Fuera del ciclo se coloca 1 condicional que comprobará el valor de “encontrado”. Si el valor de “encontrado” es false se guardará en “x” un aviso de que no existe una sección con es id. Al final se retorna “x”.

**Método agregarDatoCarrito:** guardará el valor de un producto dentro de una sección específica en el carrito de compras. Recibe 2 parámetros: 1 parámetro string nombre y 1 parámetro int numero. Se declara una variable “actual” de tipo nodoSeccion que se iguala a “primero”, se declara una variable bool “encontrado” que se iguala a false y se declara una variable “P” de objeto Producto. Luego se inicia un ciclo que funcionará mientras el valor de “actual” no sea NULL y el valor de “encontrado” sea false. Dentro del ciclo hay 2 condicionales:

* Si el valor del método getnumero de la clase Seccion es igual al parámetro numero se cambia el valor de “encontrado” a true y se delega al método agregarProducto de la clase Sección al que se le envía el parámetro “nombre”. El valor objeto producto recibido de este método se guarda en “P”.
* Si no se cumple la condición anterior: “actual” se iguala al siguiente nodo.

Al final se retorna “P”.

**Método tipoInsercion:** determina el tipo de guardado que se va a usar para la nueva sección. Recibe 2 parámetros: 1 parámetro Dato de objeto Sección y 1 parámetro int opc. Dentro del método hay 2 condicionales:

* Si el valor del parámetro “opc” es igual a un int 1: se delega al método insertarDatoUltimo de la clase listaSeccion y se le envía el parámetro “Dato”.
* Si no se cumple la primera condición: se delega al método insertarDatoPrimero de la clase listaSeccion y se le envía el parámetro “Dato”.

**Método cargarListaSeccion:** recibe 1 parámetro string “archivoListaSeccion”. se coloca el ifstream Archivo para declarar a Archivo como la entrada de datos del archivo al código y luego se hace el llamado para abrir el archivo “archivoListaSeccion” “.txt”. Después se colocan 2 condicionales para comprobar que el archivo existe. Si no existe un archivo llamado “archivoListaSeccion” “.txt” se imprime un avis de que no se pudo abrir el archivo. Si el archivo “archivoListaSeccion” “.txt” sí existe se inicializa un ciclo que va a funcionar mientras el archivo tenga datos.

Dentro del ciclo se declaran las variables string “nombre” y int “numero” que se usarán para almacenar los datos obtenidos de los archivos. Con un getline de recupera el valor de “nombre” hasta el “;” de la cadena de texto También se recupera. Luego se coloca Archivo.ignore para saltar a la siguiente cadena de texto y repetir el proceso. A continuación, se coloca una condicional que dicta que, si el dato guardado que se va a recuperar no es un espacio en blanco, entonces declara una variable “S” de objeto Seccion y se le guardan por parámetros las variables “numero” y “nombre” para luego delegar al método cargarListaCategoria, al que se le envía “nombre”, de la clase listaSeccion y delegar al método insertarDatoUltimo, al que se le envía “S”, de la clase listaSecciones.

Por último, se llama al método close de archivos para cerrar el archivo.

**Método guardarListaSeccion:** recibe 1 parámetro string “nombreArchivo”. Se declara la variable “salidaC” como la salida de los valores del programa hacia los archivos y la variable “actual” de objeto nodoFerreteria y se iguala al valor de nodoCabeza. Luego se llama al método open de archivos para abra o cree un archivo de nombre “nombreArchivo” “.txt”. Después se coloca un ciclo que funcionará mientras que la lista no esté vacía. Dentro del ciclo se llama al método getnombre de la clase Seccion para que guarde el valor del nombre del nodo actual dentro del archivo y después le coloque un “;” como diferenciador. Los mismo se hace con el método getnumero de la clase Seccion y por último de guarda un espacio en blanco en el archivo antes de avanzar al siguiente nodo. Al final del método se llama a la función close de archivos para cerrar el archivo.

**Método guardarListaCategoria:** se declara una variable actual de objeto nodoSeccion y se iguala a “primero” (una variable declarada en el .h de listaSucursal). Después coloca un ciclo que funcionará mientras “actual” sea diferente de vacío. Dentro del ciclo se delega al método guardarListaCategoria, al que se le envía el valor “nombre” de la clase Seccion y luego se coloca la condicional que dice que si “actual” es igual a “ultimo” no haga nada. Luego se iguala a “actual” al nodo siguiente.

**Método guardarArchivoPila:** se declara una variable actual de objeto nodoSeccion y se iguala a “primero” (una variable declarada en el .h de listaSucursal). Después coloca un ciclo que funcionará mientras “actual” sea diferente de vacío. Dentro del ciclo se delega al método guardarArchivoProducto y luego se coloca la condicional que dice que si “actual” es igual a “ultimo” no haga nada. Luego se iguala a “actual” al nodo siguiente.

**Clase pilaProducto**

**Método llenarPila:** guarda un objeto Producto en la pila de productos. recibe 1 parámetro “P” de objeto Producto. Se llama al método push de pila y se le envía el parámetro “P”.

**Método toString:** imprime el stack de productos. Declara una variable “aux” de stack de Productos y la iguala al stack “pila” de productos guardada. Inicia un ciclo que se recorra mientras que el stack “aux” no esté vacío. Dentro del ciclo se llama al método toString de la clase Producto para que imprima el primer valor del stack “aux” y luego se elimina el primer elemento del stack “aux”.

**Método eliminarProducto:** llama al método pop del stack “pila” para que elimine el primer elemento del stack.

**Método mostrarDatosPila:** imprime el valor del método getNombre de la clase pilaProducto seguido por la cantidad de elementos del stack “pila”.

**Método empty:** regresa un valor NULL si la pila está vacía.

**Método cargarPilaProducto:** recibe 1 parámetro string “archivoPilaProducto”. Se declara la variable Archivo para declarar la salida de los datos del archivo hacia al programa. Luego se llama el método open de archivo para abrir el archivo de nombre “archivoPilaProducto” “.txt”. luego se colocan dos condicionales para comprobar si el archivo existe o no. Si el archivo no existe se imprime un aviso. Si el archivo existe entra a un ciclo que se realiza mientras que el archivo tenga datos.

Dentro del ciclo se declaran las variables string “Nombre” y int “Codigo” que se usarán para almacenar los datos obtenidos de los archivos. Con un getline de recupera el valor de “Nombre” hasta el “;” de la cadena de texto. También se recupera el dato “Codigo” para enviárselo al Producto. Luego se coloca Archivo.ignore para saltar a la siguiente cadena de texto y repetir el proceso. A continuación, se coloca una condicional que dicta que, si el dato guardado que se va a recuperar no es un espacio en blanco, entonces declara una variable “P” de objeto Producto y se le guardan por parámetros las variables “Nombre” y “Codigo” para luego delegar al método push, al que se le envía “P”, de la clase pilaProducto.

Por último, se llama al método close de archivos para cerrar el archivo.

**Método guardarPilaProducto:** recibe 1 parámetro string “archivoPilaProducto”. Se declara la variable “salidaC” como la salida de los valores del programa hacia los archivos. Luego se llama al método open de archivos para que abra o cree un archivo de nombre “archivoPilaProducto” “.txt”. Luego se coloca un ciclo que funcionará mientras que el stack “pila” no esté vacío. Si no está vacío se guarda dentro del archivo el valor de los métodos getNombre y getCodigo de la clase pilaProducto del primer producto de la pila. Después se borra el primer producto de la pila.

**Clase listaCategoria**

**Método toString:** imprime la lista de categorías de productos. Se declaró una variable “actual” de objeto nodoCategoria y se iguala al valor de “primero” (variable de objeto nodoCategoria declarada en el .h). Se colocan 2 condicionales:

* Si el valor de primero no es NULL: se inicia un ciclo que funciona mientras que el valor de “actual” no sea NULL. Dentro del ciclo se llama al método mostrarDatosPila de la clase pilaProducto guardada en este nodo y se imprime el valor retornado para luego avanzar al siguiente nodo.
* Si no se cumple la primera condición: se imprime un mensaje de que la lista está vacía.

**Método insertarDatoUltimo:** guarda una categoría de productos en el ultimo nodo de la lista. Recibe 1 parámetro Dato de objeto pilaProducto. Se declara una variable “nuevo” de objeto nodoCategoria y en ella se guarda el valor del parámetro “Dato” junto con dos variables de nodoCategoria, “primero” y “ultimo”, que apunten al primer y ultimo nodo de la lista. Se colocan 2 condicionales:

* Si el valor de “primero” es igual a NULL: “primero” se iguala al valor de “nuevo”. Luego, el valor del método setSig de “primero” se iguala a NULL, el valor del método setatras de “primero” se iguala a NULL y el valor de “ultimo” se iguala al valor de “nuevo”.
* Si no se cumple la primera condición: el valor del método setSig de “ultimo” se iguala a “nuevo”, el valor del método setSig de “nuevo” se iguala a NULL, el valor del método setatras de “nuevo” se iguala a “ultimo” y el valor de “ultimo” se iguala al valor de “nuevo”.

**Método insertarDatoPrimero:** guarda una categoría de productos en el primer nodo de la lista. Recibe 1 parámetro “Dato” de objeto pilaProducto. Se declara una variable “nuevo” de objeto nodoCategoria y en ella se guarda el valor del parámetro “Dato” junto con dos variables de nodoCategoria, “primero” y “ultimo”, que apunten al primer y último nodo de la lista. Se colocan 2 condicionales:

* Si el valor de “primero” es igual a NULL: “primero” se iguala al valor de “nuevo”. Luego, el valor del método setSig de “primero” se iguala a NULL, el valor del método setatras de “primero” se iguala a NULL y el valor de “ultimo” se iguala al valor de “nuevo”.
* Si no se cumple la primera condición: el valor del método setatras de “primero” se iguala a nuevo, el valor del método setSig de “nuevo” se iguala a primero, el valor del método setatras de “nuevo” se iguala a “NULL” y el valor de “primero” se iguala al valor de “nuevo”.

**Método BuscarProducto:** busca un producto dentro de una categoría en específica. Recibe 1 parámetro string nombre. Se declara una variable “actual” de tipo nodoCategoria que se iguala a “primero” y se declara una variable bool “encontrado” que se iguala a false. Luego se colocan 2 condicionales que revisan si la lista está vacía. Si el valor de “primero” es igual a NULL se imprime un aviso de que la lista está vacía. Si el valor de “primero” es diferente a NULL se inicia un ciclo que funcionará mientras el valor de “actual” no sea NULL y el valor de “encontrado” sea false.

Dentro del ciclo hay una condicional que dicta que si el valor del método getNombre de la clase Producto es igual al parámetro “nombre” se llama e imprime el método toString de la clase Producto y se cambia el valor de “encontrado” a true. Sin importar la condicional anterior se cumple o no, “actual” se iguala al siguiente nodo. Si al finalizar el ciclo el valor de “encontrado” sigue siendo false se imprime un aviso de que no se encontró ningún producto con ese nombre.

**Método eliminarDato:** elimina una categoría de productos de la lista. Recibe 1 parámetro string nombre. Se declara una variable “actual” de objeto nodoCategoria que se iguala a “primero”, también se declara una variable “anterior” de objeto nodoCategoria que se iguala a “NULL” y por último se declara una variable bool “encontrado” que se iguala a false. Luego se colocan dos condicionales comprueban si la lista está vacía. Si “primero” es igual a NULL se imprime un aviso de que la lista está vacía. Si “primero” es diferente de NULL se inicia un ciclo que funcionará mientras que el valor de “actual” sea diferente de NULL y el valor de “encontrado” sea false. Dentro del ciclo hay 2 condicionales que analizan si el valor del método getNombre de la clase Producto es igual al valor del parámetro “nombre”. Si ambos valores son distintos “anterior” se iguala a “actual” y “actual” se iguala al nodo siguiente. Por otro lado, si ambos valores son iguales se colocan 4 condicionales:

* Si el valor de “anterior” es igual a NULL: “primero” se iguala al valor del método getSig de “primero”, el método setSig de “actual” se iguala a NULL y “actual” se iguala a “primero”.
* Si el valor de “actual” es igual al valor de “primero”: “primero” se iguala al método getSig de “primero” y el método setatras de “primero” se iguala a NULL.
* Si el valor de “actual” es igual a “ultimo”: el método setSig de “anterior” se iguala a NULL y el valor de “ultimo” se iguala a “anterior”.
* Si no se cumplió ninguna de las condicionales anteriores: el método setSig de “anterior” se iguala al método getSig de “actual” y el método setatras del método getSig de “actual” se iguala a “anterior”.

Sin importar a cuál de las 4 condicionales anteriores entre el código se imprimirá un mensaje indicando que se eliminó correctamente el dato.

**Método agregarDatoCarrito:** agrega un producto de una categoría específica al carrito de compras recibe 1 parámetro string nombre. Se declara una variable “actual” de tipo nodoCategoria que se iguala a “primero”, se declara una variable bool “encontrado” que se iguala a false y se declara una variable “P” de objeto Producto. Luego se inicia un ciclo que funcionará mientras el valor de “actual” no sea NULL y el valor de “encontrado” sea false. Dentro del ciclo hay 1 condicional que evalúa si el valor del método getNombre de la clase pilaProducto es igual al parámetro nombre.

De ser así se coloca otra condicional que evalúa si el stack de productos de la clase pilaProducto está vacío. Si no lo está guarda un producto almacenado en el stack en la variable “P” y se cambia el valor de “encontrado” a true.

Al final se retorna “P”.

**Método tipoInsercion:** determina el tipo de inserción que se va a usar para introducir la categoría. Recibe 2 parámetros: 1 parámetro Dato de objeto pilaProducto y 1 parámetro int opc. Se colocan dos condicionales:

* Si le valor del parámetro “opc” es igual a un int 1: se delega al método insertarDatoUltimo de la clase listaCategoria y se le envía el parámetro “Dato”.
* Si no se cumple la condicional anterior: se delega al método insertarDatoPrimero de la clase listaCategoria y se le envía el parámetro “Dato”.

**Método cargarListaCategoria:** recibe 1 parámetro string.

Método guardarListaCategoria: recibe 1 parámetro string.

Método guardarArchivoProducto: no recibe ningún parámetro.

**Clase Producto**

**Método toString:** imprime el nombre y el código del objeto producto.

**Clase pilaCarrito**

**Método llenarPila:** llena la pila de objetos Producto. Recibe 1 parámetro P de objeto producto. Inicia un ciclo que va a funcionar mientras el stack “pila” no esté vacío. Dentro del ciclo se llama al método push de “pila” y se le envía el parámetro “P”.

**Método toString:** imprime la pila de objetos producto. Declara una variable “aux” de tipo stack y se iguala al stack “pila”. Luego se inicia un ciclo que funcionará mientras el stack “aux” no esté vacío. Dentro del ciclo se llamará al método toString de la clase Producto a través del método top de “aux” para imprimir el primer valor del stack y luego se eliminará el primer valor del stack “aux” con el método pop.

**Método eliminarProducto:** elimina un valor del stack “pila”. Se llama al método pop del stack “pila” para eliminar el primer valor del mismo stack “pila”.

**Método empty:** regresa un valor NULL si la pila está vacía.

Método cargarPilaProducto:

Método guardarPilaProducto:

**Clase interfaz**

**Método agregarSucursal:** se declaran las variables string id y string ubicación para que el usuario las ingrese como los datos de la nueva sucursal. Posteriormente a su ingreso por parte del usuario se guardan por parámetros en una variable “F” de tipo ferreteria. Finalmente se llama al método agregarDato de la clase listaSucursal y se le envía “F”.

**Método agregarSeccion:** Se declaran las variables string “id”, string “nombre” e int “opc”. Luego se le pide al usuario que ingrese el “id” de la sucursal a la que desea añadirle una sección. Una vez ingresado el dato se colocan dos condicionales que evalúan si el id ingresado pertenece a una sucursal existente. Si no pertenece a ninguna sucursal imprime un aviso de que no hay una sucursal con ese id.

Si sí hay una sucursal con ese id se le pide al usuario que inserte el numero de la sección a agregar. Se coloca un ciclo que comprueba si el valor digitado sí es un valor numérico, de no serlo se imprime un aviso de que el valor no es válido. Luego se solicita al usuario que ingrese el nombre de la nueva sección y que decida si desea insertarla al inicio o al final de la lista. Una vez todo ha sido digitado se guardan los datos en una variable S de tipo Sección que recibe los valores “numero” y “nombre” por parámetros. Por último, se delega al método agregarSeccion de listaSucursal y se le envían las variables “id”, “S” y “opc”.

**Método agregarProducto:** se declaran las variables string “id”, string “nombre”, int “numero”, int “código”, int “n” e int “opc”. Se le pide al usuario el “id” de la sucursal a la que desea acceder. Una condicional llama al método buscarSucursal de listaSucursal y le envía el id digitado para buscar la sucursal digitada. Si no hay ninguna sucursal con ese id se imprime un aviso.

Si encuentra una sucursal con es id le pide que digite el numero de la sección a la que desea ingresarle el producto. Luego comprueba, con un ciclo, que el valor digitado sea válido. Una vez digitado un valor aceptable se llama al método BuscarSeccion de listaSucursal y se le envía el parámetro “id” y “numero” para comprobar si existe dicha sección. De no existir se imprime un aviso.

Si existe la sección dentro de la sucursal se declara una variable pilaP de objeto pilaProducto y se le guarda por parámetros nombre del producto. Luego se le pide al usuario que digite cuantos productos desea guardar, esto se guarda en “n" y se evalúa con un ciclo si el dato digitado es un valor válido, de no serlo se imprime un aviso. A continuación, se inicia un ciclo que se repite la “n” cantidad de veces digitada por el usuario. En este ciclo se le pide al usuario que ingrese el código de cada producto y se guardará en la variable “código”, variable que se evaluará para determinar si el valor es válido. Una vez digitado y evaluado el código se declara la variable P de tipo Producto y se almacenan por parámetros las variables “nombre” y “código” para luego llamar el método llenarPila de pilaP y enviarle la variable P (este proceso se repite con cada código). Finalmente se sale del ciclo y se le pregunta al usuario que tipo de inserción desea usar para luego delegar al método agregarProductoSeccion de listaSucursal y enviarle las variables “id”, “numero”, “pilaP” y “opc”.

**Método eliminarSucursal:** se declara la variable string “id” y se le pide al usuario que digite la sucursal a eliminar en “id”. Luego se delega al método eliminarSucursal de listaSucursal y se le envía “id”.

**Método eliminarSección:** se declaran las variables string “id”, int “numero” y se imprime el método mostrarLista de listaSucursal. Luego se le pide al usuario que digite en “id” la sucursal a la que desea eliminarle una sección. Luego se usa una condicional y el método buscarSucursal de listaSucursal para comprobar que la sucursal exista, de no hacerlo se imprime un aviso. Luego se imprimen las secciones con el método mostrarSecciones de listaSucursal, al que se le envía “id”, y se le pide al usuario que digite en “numero” la sección a eliminar, valor que se revisará para comprobar que es válido. Después se usa una condicional donde se delega al método BuscarSeccion de listaSucursal, enviándole “id” y “numero”, para comprobar que exista. Si existe se delega al método eliminarSeccion de listaSucursal, al que se le envían “id” y “numero”, y se imprime lo que retorne. Si no existe se imprime un aviso.

**Método eliminarProducto:** se declaran las variables string “id”, string “nombre”, int “numero” e int “opc” y se imprime el método mostrarLista de listaSucursal. Luego se le pide al usuario que digite en “id” la sucursal a la que desea eliminarle un producto. Luego se usa una condicional y el método buscarSucursal de listaSucursal para comprobar que la sucursal exista, de no hacerlo se imprime un aviso. Luego se imprimen las secciones con el método mostrarSecciones de listaSucursal, al que se le envía “id”, y se le pide al usuario que digite en “numero” la sección a la que le desea eliminar el producto, valor que se revisará para comprobar que es válido. Después se usa una condicional donde se delega al método BuscarSeccion de listaSucursal, enviándole “id” y “numero”, para comprobar que exista. Si no existe se imprime un aviso. Si existe se imprimen los productos con el método mostraProductosSeccion, al que se le envía “id” y “numero”.

Luego se pide al usuario que ingrese el nombre del producto a eliminar y si desea eliminar todo el stack de productos. Los datos ingresados se guardan en “nombre” y “opc”, esta último se evaluará en un ciclo para comprobar que sea un valor válido (de no serlo se imprime un aviso). Si “opc” es igual a 1 se llama al método eliminarProducto de listaSucursal, al que se le envía “id”, “numero” y “nombre”, y se imprime lo que retorne.

**Método agregarDatoCarrito:** recibe 3 parámetros: 1 parámetro string id, 1 parámetro int numero y 1 parámetro carrito de objeto pilaCarrito. Se declaran las variables string “nombre”, int “opc” y una variable “P” de objeto Producto. Se inicia un ciclo que no parará mientras que “opc” sea igual a 1. Dentro del ciclo se le pide al usuario que digite el numero del producto a comprar que se almacena en “nombre”. Luego se delega al método agregarDatocarrito de listaSucursal, al que se le envían “id” “numero” y “nombre”, y lo retornado se almcena en “P”. Luego se delega al método llenarPila de pilaCarrito, al que se le envía “P”, y se le pide al usuario que digite 1 si desea otro producto o 0 para salir y la respuesta se almacena en “opc”, dato que se revisa en un ciclo para verificar que sea válido.

**Método menuAdministrativo:** se declaran las variables string “nombre”, int “opc”, int “numero”. Luego se crea un ciclo que funcionará mientras que “opc” tenga un valor distinto a 0. Dentro del ciclo se imprimen todas las opciones del menú administrativo y una petición al usuario administrativo para que digite, en “opc”, cuál opción quiere usar. Después un ciclo evalúa si el valor digitado en “opc” es válido. Luego se crea un switch de 9 opciones que redirigirán a una función específica.

* Case 1: delega al método agregarSucursal de la clase interfaz.
* Case 2: delega al método agregarSeccion de la clase interfaz.
* Case 3: delega al método agregarProducto de la clase interfaz.
* Case 4: delega al método eliminarSucursal de la clase interfaz.
* Case 5: delega al método eliminarSeccion de la clase interfaz.
* Case 6: delega al método eliminarProducto de la clase interfaz.
* Case 7: delega al método mostrarLista de la clase listaSucursal.
* Case 8: se le pide al usuario que digite el id de la sucursal que se almacena en “nombre”. Luego se imprime el método mostrarSecciones, al que se le envía “nombre”, de la clase listaSucursal.
* Case 9: se le pide al usuario que digite el id de la sucursal que se almacena en “nombre”. Luego se colocan dos condicionales y el método buscarSucursal, al que se le envía “nombre”, de la clase listaSucursal para comprobar si la sucursal existe. Si la sucursal no existe se imprime un aviso. Si sí existe se imprime el método mostrarSecciones, al que se le envía “nombre”, de la clase listaSucursal y se le pide al usuario que digite el numero de la sección que quiere ver. Posteriormente se imprime el método mostrarPorductosSeccion, al que se le envían “nombre” y “numero”, de la listaSucursal.
* Default: se imprime un aviso de opción incorrecta.

**Método menuCliente:** se declaran las variables int “opc” y se delega al método cargarListaSucursal de la clase listaSucursal. Luego se crea un ciclo que funcionará mientras que “opc” tenga un valor distinto a 0. Dentro del ciclo se imprimen todas las opciones del menú de para los clientes o usuarios y una petición a los mismos para que digite, en “opc”, cuál opción quiere usar. Después un ciclo evalúa si el valor digitado en “opc” es válido. Luego se crea un switch de 4 opciones que redirigirán a una función específica.

* Case 1: delega al método menuadministrativo de la clase interfaz.
* Case 2: delega al método menuCliente de la clase interfaz.
* Case 3: delega al método menuCaja de la clase interfaz.
* Case 0: imprime un aviso del programa cerrándose.
* Default: imprime un aviso de opción incorrecta.

Al final, fuera del ciclo, se delega a los métodos gauradarDatosSucursal, guardarDatosSeccion, guardarDatosCategoria y guardarArchivosPila, todos pertenecientes a la clase listaSucursal.

**Método Menu:** se declaran las variables int “opc”. Luego se crea un ciclo que funcionará mientras que “opc” tenga un valor distinto a 0. Dentro del ciclo se imprimen todas las opciones del menú principal y una petición al usuario para que digite, en “opc”, cuál opción quiere usar. Después un ciclo evalúa si el valor digitado en “opc” es válido. Luego se crea un switch de 4 opciones que redirigirán a una función específica.

* Case 1: delega al método menuAdministrativo de la clase interfaz.
* Case 2: delega al método menuCliente de la clase interfaz.
* Case 3: delega al método menuCaja de la clase interfaz.

**Método compra:** recibe 1 parámetro carrito de objeto pilaCarrito. Se declaran las variables string “id”, int “numero”, int “opc” y int “opc2”. Luego se imprime el método mostrarLista de listaSucursal y se le pide al usuario digite el “id” de la sucursal que desea consultar.

Después se inicia un ciclo que funcionará mientras que “opc2” sea deferente de 0. Dentro del ciclo se delega al método buscarSucursal de listaSucursal, al que se le envía “id”, y se usa una condicional para comprobar si la sucursal existe. Si no existe se imprime un aviso. Si existe se imprime el método mostrarSecciones, al que se le envía “id”, se le pide al usuario que digite el numero de la sección que quiere consultar. La respuesta se almacena en “numero” y se evalúa en un ciclo para comprobar que sea un número válido.

Usando otra condicional y el método BuscarSeccion de listaSucursal, al que se le envían “id” y “numero”, se comprueba si la sección digitada existe o no. Si no existe se imprime un aviso. Si sí existe se imprime el método mostrarProductosSeccion, al que se le envían “id2 y “numero”. Posteriormente se le pide al usuario que digite 1 para agregar un producto al carrito u cualquier otro número para no hacerlo. La respuesta se almacenará en “opc” y se evaluará en un ciclo para comprobar que es un valor válido. Si “opc” es igual a 1 se delega al método agregraDatoCarrito de la clase interfaz, al que se le envían “id”, “numero” y “carrito”.

Al final del método (sin salir del ciclo principal) se le dice al usuario que si quiere continuar digite cualquier número o 0 si quiere salir. Este valor se almacena en “opc2”.

**Método menuCaja:** se declara una variable int “opc”. Luego se imprime la cola de carritos llamando al método toString de la clase caja junto con las opciones disponibles en el menú de la caja. Se le pide al cliente que ingrese la opción de la acción que desea realizar. Esta respuesta se almacena en “opc”. Posteriormente se coloca un switch de 2 opciones que redirigirán a la opción específica solicitada:

* Case 1: se coloca un condicional para comprobar si la cola de carritos está vacía. Si está vacía se imprime un aviso. Si no está vacía se imprime el método eliminarProductosCarrito de la clase caja seguido por la impresión del método factura de la clase caja y por último se delega al método eliminarCarrito de la clase caja.
* Case 2

**Clase caja:**

**Método ingresarCarrito:** recibe 1 parámetro “carrito” de objeto pilaCarrito. Delega al método push del queue “colaCaja” (nombre de la cola de la caja) y se le envía el parámetro “carrito” para que lo guarde en el queue.

**Método eliminarCarrito:** delega al método pop del queue “colaCaja” para que elimine el primer valor del queue.

**Método empty:** regresa un valor NULL si la cola está vacía.

**Método factura:** imprime los datos necesarios para la factura de la compra. Estos datos son la fecha de la compra que se obtiene de la fecha y hora de la computadora utilizando la librería ctime, la cantidad de productos que se obtiene llamando al método cantProductos de la clase pilaCarrito para que imprima el tamaño del primer valor del queue “colaCaja” y por último la lista de productos comprados que se obtiene al llamar al método toString de la clase pilaCarrito para que imprima los productos guardados en primer valor del queue colaCaja.

**Método toString:** imprime el titulo para luego entrar a una condicional para comprobar que el queue colaCaja no está vacío. Si está vacío se imprime un aviso. Si no está vacío se declara una variable int “cant” que almacena el tamaño del queue y luego se coloca un ciclo que se repetirá la cantidad de veces que lo indique la variable “cant”. Dentro del ciclo se imprimirá la posición de los carritos existentes en el queue.

**Método eliminarProductosCarrito:** se declara una variable “aux” de tipo queue pilaCarrito y se iguala al queue colaCaja. Se inicia un ciclo que funcionará mientras la variable “aux” no esté vacía. Si “aux” está vacía se imprime un mensaje de que todos los productos del queue han sido eliminados. Si “aux” no está vacío se imprime un mensaje que indique que se está eliminando el producto y posteriormente se llama al método de eliminarProducto de la clase pilaProducto para que elimine el primer valor del queue “aux”.