TALLER DE DESARROLLO 2

PROYECTO FINAL

2022

UNIVERSIDAD AUTÓNOMA DE CHIAPAS

FACULTAD DE NEGOCIOS CAMPUS IV

INGENIERÍA EN DESARROLLO Y TECNOLOGÍAS DE SOFTWARE

TALLER DE DESARROLLO 2

PROYECTO FINAL

INTEGRANTES

PEDRO OCTAVIO CULEBRO PRADO

JOSÉ EDUARDO OROZCO CARDENAS

SAMUEL SANCHEZ GUZMAN

EMILIA ZUÑIGA LOSADA

DOCENTE

MTRO. ERWIN BERMUDEZ CASILLAS

GRADO Y GRUPO

4° “D”

TAPACHULA DE CÓRDOVA Y ORDOÑEZ, CHIAPAS A 15 DE MAYO DE 2022

1.ÍNDICE DE CONTENIDO

2. INTRODUCCIÓN

Dentro del desarrollo de tecnologías, los puntos de venta (POS), están basados en un software inteligente que cualquier empresa puede utilizar para satisfacer las necesidades de administración.

A manera de proyecto final, para la materia de Taller de Desarrollo 2 correspondiente a la Licenciatura en Ingeniería de Desarrollo y Tecnologías de Software, se nos presenta la siguiente problemática:

La Compañía "MalairStore" desea tener un Punto de Venta para controlar el inventario de los productos y al mismo tiempo conocer las ventas.

Entendiendo el contexto habido dentro de esta compañía (*3.1*), nos pusimos manos a la obra con la realización del software de punto de venta, mismo que será programado en Java y conectado a una base de datos de PostgreSQL.

Para la creación de este sistema fue muy importante realizar un estudio detallado de los problemas que afectan el funcionamiento de la organización de cara al cliente y en su medio ambiente, para tener claridad de lo que se iba a desarrollar, fue necesario considerar todos los pasos a efectuar tanto por el administrador como por el personal de ventas, concretando así a la idea de la implementación.

Teniendo presente lo anterior, nos adentramos en la investigación para documentar todos los conocimientos adquiridos y puestos en práctica, presentando así el presente reporte escrito, con los siguientes objetivos:

Objetivo general:

Entregar como pruebas del trabajo realizado en el salón de clases, la solución al problema presentado.

Objetivos específicos:

Realizar con el propósito de la integración con compañeros del salón, el trabajo por equipos, en el que se debatirán y analizarán tanto el orden como la lógica empleada para la creación del software. Así como entregar, a manera de conclusiones, las observaciones finales y las experiencias obtenidas antes, después y durante la realización de este trabajo.

Sin nada más que agregar, damos paso a lo siguiente…

3. DEFINICIÓN DEL MARCO TEÓRICO

3.1. Delimitación del tema de la investigación.

Diseño, creación y documentación de un software de punto de venta para la automatización de procesos de una tienda de autoservicio de tipo club.

3.2 Definición del tipo de investigación.

El tipo de investigación seleccionado para este proyecto es de investigación aplicada, debido a que emplearemos los recursos teóricos que recabamos para la resolución de un problema, en pro del desarrollo científico y la mejora cultural tanto de nuestro equipo como de todo aquel que se interese en nuestro proyecto.

3.3 Presentación del problema.

3.3.1. Narrativa general:

La compañía 'MalairStore' presenta una serie de deficiencias, debido a su anticuado sistema de administración; dentro de las necesidades que tiene la compañía está conocer las entradas y salidas de los productos que venden de tal manera que necesitan un manejo de inventario preciso, esto surge porque existen algunos productos sobrantes y otros faltantes al control que llevan actualmente y cómo conocer la fecha de ingreso o de egreso de los productos, no pueden determinar dónde está la falla. Otra necesidad es conocer los montos de las ventas, para esto ocupa que se lleve un control por día pero también requieren que se pueda generar uno por rango de días, de tal manera que puedan conocer los ingresos de efectivo que tuvieron.

Tomando esto en cuenta, realizamos un listado de acciones analizadas, que serán tomadas en cuenta para la resolución de este caso.

3.3.2. Posibles soluciones a la problemática

-Creación de clases y métodos para su implementación en ventanas separadas por categorías, apuntando hacia una navegación eficiente.

-Extracción directa de los datos de la base para su posterior búsqueda, modificación y eliminación; así también la inserción de nuevos datos totalmente funcionales e

implementados de forma automática por medio de órdenes de sql aplicadas de manera interna en el código fuente.

-Funciones específicas de consulta para averiguar el rendimiento de la empresa por día y por rangos de fecha.

-Manejador de ventas y de inventario.

3.3.3. Presentación de hipótesis para la resolución

-La implementación de software de punto de venta en la empresa antes mencionada, satisfacerá las necesidades antes presentadas y entregará a los trabajadores y encargados las facilidades requeridas para llevar a cabo sus labores de manera eficiente.

-El desarrollo y documentación de este proyecto aportará a la mejora de metodologías de trabajo, tanto de los colaboradores de este proyecto, como de los lectores interesados.

3.4 Solución del problema.

Para poder mantener un inventario que sea funcional y donde se puedan ver las pérdidas si las hubiera, se creó una tabla para todos los productos que entren a la tienda se agregó una columna donde se ingresa el total de los productos, al momento de ingresar una venta esta columna no se podrá modificar, por lo tanto se agrega una segunda columna esta se verá afectada por cada producto vendido, así mismo cuando se haga alguna devolución que básicamente es lo mismo que una cancelación del producto, de esta forma se puede saber cuántos son los artículos que entregó el proveedor, en el caso de las salidas se consideran todos los productos que se cobraron desde caja, se muestra una tabla con todos esos productos clasificados por el código de barras y si la persona encargada de la caja no cobra un producto, al momento de checar el inventario se podrá ver que no cuadra y que hay producto faltante, pues la cantidad vendida al sumarla con el stock actual del producto debe ser igual a la cantidad que el proveedor entregó en la tienda.

Así mismo se tienen los reportes diarios, donde se puede visualizar todos los productos que se vendieron durante un solo día, en este caso sería el día actual, puede ser de ayudar para cuando se hace el cierre de la venta diaria y ver si los artículos que se vendieron cuadran con las ventas, por otro lado están las ventas por rangos de fecha, en este caso se pueden saber las ventas de ciertos días, especificando la fecha de inicio y la fecha final, en este caso también se observa el total de las ventas, este es un valor monetario que nos hace la suma de todos los montos de los productos.

4.CONTENIDO DE LA INVESTIGACIÓN.

4.1 Github

Consiste en una herramienta o servicio en la web que permite a los desarrolladores almacenar y administrar sus códigos en la nube, así como llevar un registro y control de los cambios que se realicen al mismo. Específicamente, es un sistema de control de versión distribuida, es decir, el código fuente es posible encontrarlo en la computadora de cualquier desarrollador esto permite un fácil acceso a las ramas y fusiones, es importante comprender los conceptos anteriores pues son base para entender el funcionamiento de github, es posible llevar un control del proyecto principal mediante una bifurcación, esto es denominado repositorio que se puede comprender como la rama master la cual contendrá todos los cambios realizados a través de las fusiones, estas son posibles a través de la creación de las ramas de todos los colaboradores del proyecto, por ello se dice que github es una herramienta de desarrollo colaborativo pues permite a más de un desarrollar hacer cambios al código pero usando su rama personal donde los cambios realizados no afectan a la rama master. Github posee una interfaz fácil de comprender donde especifica cada una de las acciones que se pueden realizar, una de ellas es ver cuando alguien ha hecho un commit, cuando se hayan realizado cambios en cualquiera de las ramas o solicitudes de extracción de código. Para fines del proyecto se implementó esta herramienta en NetBeans donde proporciona herramientas para subir cambios o extraerlos ya sea de nuestra rama o las de los colaboradores.

4.2 Variables

Las variables en java es el nombre que se le da a una ubicación de memoria, estos también son llamados identificadores por lo que es la unidad básica de almacenamiento de cualquier programa, mientras que los valores asignados a dicha variable pueden ser cambiados a lo largo de la ejecución del programa, de ahí proviene su nombre “variable”, pues es un valor que no siempre estará fijo. Es importante tener presente que las variables deben declararse antes de ser usadas en cualquier parte del programa. Para declarar una variable es necesario conocer los tipos de datos presentes en Java, también saber para qué se utilizará, estos pueden variar y cada una almacena diferentes tipos de datos, algunos de estos tipos

son los tipos primitivos int (32 bits), short (16 bits), byte (8 bits), long (64 bits), char (16 bits), boolean (1 bits), float (32 bits) y double (64 bits).

4.3 Clases

Las clases son una parte simple del lenguaje, prácticamente todo lo que se encuentra en Java forma parte de una clase, estas están formadas por los objetos, es conveniente usar las clases porque Java no soporta las funciones o variables globales. Básicamente las clases son un contenedor en donde se pueden definir los estados y las conductas de un objeto. Un ejemplo completo podría ser la clase persona, el objeto sería un individuo en particular que cuenta con atributos o propiedades por ejemplo nombre, apellido o edad, también tiene comportamientos como hablar, escuchar, comer, caminar y finalmente métodos de la clase, estas son sus funciones.

4.4 Objetos

Son fundamentales para el uso de Java y son las que forman las clases, este objeto debe cumplir con la condición de que debe ser real y algo en concreto o específico, este debe tener un estado y funcionamiento o bien los denominados métodos estos son las acciones a las que tendrá acceso el objeto dependiendo de su clase, de esta forma si el objeto es llamado este puede realizar una acción o bien modificar su estado, el algunas ocasiones no es necesario la declaración de un objeto es decir darle un nombre. Para tener más claro en qué consiste un objeto basta con tocar algo que tengamos a nuestro alrededor, como un lápiz, un dispositivo móvil, cualquier cosa que podamos tomar y que exista de forma real.

4.4.1 Tipo iterator

El iterador es un tipo de objeto que permite recorrer una lista y presentar todos los elementos que contiene, cuenta con dos métodos importantes para realizar sus operaciones estos son hasNext() y next (). De esta forma es posible visualizar los elementos que se hayan añadido a una lista sin depender del tipo que sean, agilizando la visualización de los mismos.

4.5 Atributos

Los atributos o variable miembro son las características individuales que diferencian un objeto de otro y determinan su apariencia, estado u otras cualidades. Los atributos se guardan en variables denominadas de instancia, y cada objeto particular puede tener valores distintos para estas variables.

Las variables de instancia también denominados miembros dato, son declaradas en la clase, pero sus valores son fijados y cambiados en el objeto.

Además de las variables de instancia hay variables de clase, las cuales se aplican a la clase y a todas sus instancias. Por ejemplo, el número de ruedas de un automóvil es el mismo cuatro, para todos los automóviles, también puede ser los que describen el estado de un objeto, con el que pueden ser de cualquier tipo de dato, para poder acceder o visibilizar a ellos es a través de los modificadores de acceso, las cuales son palabras reservadas que permiten especificar la forma de acceso, y estos son los modificadores: “private, public, protected, package private”.

4.6 Métodos

Los métodos describen el comportamiento de los objetos de una clase. Estos representan las operaciones que se pueden realizar con los objetos de la clase, constituyendo un mecanismo que se utiliza para implementar mensajes entre objetos, y esto es a través de la invocación de un método correspondiente al objeto, y su ejecución puede llevar a cambiar el estado del objeto. Se definen de la misma forma que las funciones normales, pero deben declararse dentro de la clase y su primer argumento siempre referencia a la instancia que la llama, de esta forma se afirma que los métodos son funciones, adjuntadas a objetos.

La Declaracion_del_metodo proporciona información sobre su nombre, la accesibilidad del método, el número de parámetros que recibe, etc.
El Cuerpo_del_metodo contiene el conjunto de sentencias que manipula los datos de cada objeto. Además, aplican los mismos modificadores de acceso que para los atributos.

4.6.1 No reciben parámetros

Son métodos que no piden ningún dato u objeto para ejecutarse. Un método sin parámetros se identifica porque sus paréntesis finales están vacíos. Estos métodos no necesitan recibir información para ejecutarse.

4.6.2 Getters y setters

Son los métodos que ayudan a acceder a los atributos o variables por lo que también son llamados métodos de acceso, los métodos setters se encargan de asignarle un valor a un atributo, pero no tienen ningún tipo de retorno, es decir, siempre son de tipo void y solo permiten el acceso a determinados atributos los cuales el usuario podrá modificar, reciben un parámetro para establecer el valor del mismo. Los métodos getters ayudan a tener acceso ya sea recuperar y obtener el valor asignado a un atributo, por lo que su valor de retorno varía según el tipo de dato del atributo al que se desea acceder.

4.7 Constructores

Los constructores son elementos de una clase que se identifican por tener el mismo nombre de la case y que su objetivo es obligar y controlar el cómo se inicializan una instancia de una clase determinada, pues Java no permite que las variables miembros de una nueva instancia estén sin inicializar, por lo que son utilizados solo cuando se desea crear una nueva instancia. Se caracterizan por no tener un tipo de devolución explicito, por lo que se usa para asignar valores iniciales a las variables, Java proporciona un constructor por defecto donde las variables están inicializadas en null o bien en 0 esto dependerá del tipo de dato.

4.7.1 Reciben parámetros

Para poder declarar un constructor que si reciba parámetros se debe asignar el mismo identificador de la clase, pero sin especificar el tipo de valor de retorno. Se debe tomar en cuenta que ya creado el constructor con parámetros de esa clase ya no se podrá crear otro al menos que se use la sobrecarga la cual permite crear la declaración de más de un constructor con el mismo nombre, pero debe tener diferente número y tipo de parámetros. Se debe considerar que cuando se declaran este tipo de constructores ya no es posible usar el constructor por defecto sin parámetros.

4.7.2 No reciben parámetros

Este constructor es proporcionado por Java, donde no recibe ningún parámetro pero que al ejecutarse inicializa el valor de los atributos, los datos primitivos son inicializados en 0 en false mientras que los de tipo objeto se inicializan en null.

4.8 Herencia

Es una propiedad que permite que los objetos sean creados a partir de otros ya existentes, obteniendo características (métodos y atributos) similares a los ya existentes. Es la relación entre una clase general y otra clase más específica. Es un mecanismo que nos permite crear clases derivadas a partir de clase base, Nos permite compartir automáticamente métodos y datos entre clases subclases y objetos, lo cual nos facilita reutilizar el código, pero de una manera más elegante. Si una clase deriva de otra la palabra *extends* hereda todas sus variables y métodos, la clase derivada puede añadir nuevas variables y métodos o redefinir (overriden) las variables y métodos heredados. La herencia es transitiva. Esto quiere decir que, sean las clases A, B y C, si A hereda de B y B hereda de C entonces A hereda de C.

4.8.1 Clase base

Se conoce como clase base o clase padre, una clase que va a heredar sus propiedades (variables) y funcionalidades (métodos) a otras clases, y para que se puede realizar la herencia a la clase derivada a través de la palabra extends, porque sin esta palabra reservada no es posible la herencia, estaría sin derivar hacia las clases derivadas.

4.8.2 Clase derivada

La clase que hereda la otra clase se conoce como subclase (o una clase derivada, clase extendida o clase hija). La subclase puede agregar sus propios campos y métodos, además de los campos y métodos de la superclase. En las subclases podemos heredar los miembros tal como están, reemplazarlos, ocultarlos o complementarlos con nuevos miembros: Los campos heredados se pueden usar directamente, al igual que cualquier otro campo. Podemos declarar nuevos campos en la subclase que no están en la superclase. Los métodos heredados se pueden usar directamente tal como son.

4.9 Estructuras condicionales

Las estructuras condicionales comparan una variable contra otros valores para que, en base al resultado de esta comparación, se siga un curso de acción dentro del programa. Cabe mencionar que la comparación se puede hacer contra otra variable o contra una constante, según se necesite. Es importante recordar que la condición debe dar un resultado booleano, por lo que lo más normal es usar operadores relacionales y condicionales. Las estructuras de control son las siguientes:

4.9.1 If

Esta estructura permite ejecutar un conjunto de sentencias en función del valor que tenga la expresión de comparación (se ejecuta si la expresión de comparación tiene valor true). Las llaves {} sirven para agrupar en un bloque las sentencias que se han de ejecutar, y no son necesarias si sólo hay una sentencia dentro del if. [Imagen 15]

4.9.2 If-else:

Análoga a la anterior, de la cual es una ampliación. Las sentencias incluidas en el se ejecutan en el caso de no cumplirse la expresión de comparación (false), Permite introducir más de una expresión de comparación. Si la primera condición no se cumple, se compara la segunda y así sucesivamente. En el caso de que no se cumpla ninguna de las comparaciones se ejecutan las sentencias correspondientes. Se trata de una alternativa a la bifurcación if elseif else cuando se compara la misma expresión con distintos valores.

4.9.3 Switch:

Las características más relevantes de switch son las siguientes:

- Cada sentencia case se corresponde con un único valor de expresión. No se pueden establecer rangos o condiciones, sino que se debe comparar con valores concretos. El ejemplo del apartado 2.3.3.3 no se podría realizar utilizando switch.
- Los valores no comprendidos en ninguna sentencia casen se pueden gestionar en default, que es opcional.

- En ausencia de break, cuando se ejecuta una sentencia case se ejecutan también todas las case que van a continuación, hasta que se llega a un break o hasta que se termina el switch.

4.10 Estructuras repetitivas

Las estructuras repetitivas son las encargadas de repetir varias veces un conjunto de instrucciones de código de forma cíclica, es decir de manera repetitiva, en Java existen 3 tipos de ciclos.

4.10.1 For

El ciclo For es el que permite ejecutar iteraciones un número determinado de veces, aplicándose principalmente cuando ya se tiene el conocimiento de cuantas veces queremos que se ejecute la instrucción, siendo alimentada por e elementos indispensables para su funcionamiento la inicialización, la terminación y por último el incremento.

4.10.1 While

El ciclo While es aquel que ejecuta un conjunto de sentencias mientras se cumpla una determinada condición, en otras palabras, mientras la condición sea verdadera se seguirán ejecutando instrucciones de manera cíclica

4.10.1 Do-While

El ciclo Do-while trabaja muy similar al ciclo While, lo que lo diferencia de este es que en este ciclo la condición se evaluará hasta el final de este, permitiendo que las sentencias se ejecuten de manera cíclica por lo menos una vez, a diferencia del While que puede que no se ejecuten ni una sola vez.

4.11 Elementos del JFrame

Los elementos del JFrame son las clases cuyo nombre comienza con J que forman parte de la librería Swing.

Siendo estas unas clases abstractas representadas en cualquier componente con una representación gráfica mostrándose en las ventanas de un JFrame.

- **JLabel:** Un objeto JLabel es aquel objeto de control que permite dibujar en el formulario una etiqueta de texto, entendiéndose como etiqueta una expresión estática que se quiere colocar.
- **JTextField:** El control JTextField es un elemento que permite al operador del programa ingresar una cadena de caracteres por teclado.
- **JComboBox:** el control comboBox nos permite seleccionar un String que está dentro de una lista que contiene el JComboBox.
- **JTabbedPane:** El JTabbedPane es un componente que funciona al igual que un JPanel como un contenedor, con la diferencia que este tiene la capacidad de contener pestañas, resultando útil debido a que solo está visible lo que está en la pestaña actual y las demás quedan ocultas.
- **JPanel:** Un JPanel es un contenedor que tiene la capacidad de almacenar y organizar los componentes, su tarea principal es organizar los componentes dentro del mismo.
- **JTable:** El JTable es un componente visual que nos permite como tal dibujar una tabla, de forma que en cada fila/columna de la tabla podamos poner el dato que queramos.

4.12 Conexión de Java con PostgreSQL

Dentro de los elementos que delimitaron el funcionamiento de nuestra base de datos sobre el programa de Java, nos encontramos con la parte de la conexión que será importada por medio de variables predeterminadas por el gestor de base de datos empleado (postgresql), y definidas en la clase conexión de java, así como también hará uso de dependencias en función al maven, donde se colocarán drivers adecuados a las El número de conexiones a una base de datos que requiere un nodo de integración depende de las acciones de los flujos de mensajes que acceden a la base de datos. Para cada hebra de flujo de mensajes, un nodo de integración que accede a una base de datos realiza una conexión para cada nombre de origen de datos (DSN). Si un nodo diferente de la misma hebra utiliza el mismo DSN, se utiliza la misma conexión, a menos que se utilice una modalidad de transacción diferente, en cuyo caso se necesita otra conexión. Para obtener más información

sobre transacciones. #DNS: Domain name system #JDBC: Java Database Connectivity (conectividad de bases de datos Java). Para poder usar una base de datos en programas escritos en Java, es necesarios hacer uso de JDBC que es una API que nos ofrece las librerías para trabajar con base de datos, también de PgJDBC que es un controlador JDBC que permite que los programas Java se conecten a una base de datos PostgreSQL. Para poder usar PgJDBC tendremos que descargarlo desde el sitio oficial: PostgreSQL JDBC Driver Habiendo descargado el archivo con extensión .JAR, solo seria de añadir la librería externa al IDE de tu preferencia para poder compilar el programa Java, en este ejemplo utilizaremos la versión 42.2.24 de PgJDBC.

4.13 Sentencias para usar SQL en JAVA (con la librería java.sql.*)

Una vez cargado el driver apropiado para nuestro SGBD deberemos establecer la conexión con la BD. Para ello utilizaremos el siguiente método:

- **Connection con = DriverManager.getConnection(url, login, password);**

La conexión a la BD está encapsulada en un objeto Connection. Para su creación debemos proporcionar la url de la BD y, si la BD está protegida con contraseña, el login y password para acceder a ella. El formato de la url variará según el driver que utilicemos. Sin embargo, todas las url tendrán la siguiente forma general: jdbc:<subprotocolo>:<nombre>, con subprotocolo indicando el tipo de SGBD y con nombre indicando el nombre de la BD y aportando información adicional para la conexión.

Connection con = DriverManager.getConnection("jdbc:postgresql://localhost:5432/bd", "USWER", "PASSWORD");

Podemos depurar la conexión y determinar qué llamadas está realizando JDBC. Para ello haremos uso de un par de métodos que incorpora DriverManager. En el siguiente ejemplo se indica que las operaciones que realice JDBC se mostrarán por la salida estándar:

- **Statement stmt = con.createStatement();**

Podemos dividir las sentencias SQL en dos grupos: las que actualizan la BD y las que únicamente la consultan. En las siguientes secciones veremos cómo podemos realizar estas dos acciones.

- **El metodo createStatement()**

El método createStatement() se utiliza para crear un objeto que modela a una sentencia SQL. Es un objeto del tipo de una clase que implementa la interfaz Statement, y provee la infraestructura para ejecutar sentencias SQL sobre una conexión con una base de datos.

La forma de construir un objeto de este tipo es:

Statement stmtConsulta = laconexion.createStatement();

- **El método executeQuery()**

El método executeQuery() se utiliza para ejecutar una sentencia SQL y obtener el resultado correspondiente dentro de un objeto del tipo ResulSet. Este objeto representa un conjunto de resultados que se obtienen como consecuencia de ejecutar la sentencia SQL del tipo SELECT a través de la conexión.

La forma de generar un objeto de este tipo es:

ResulSet rs = stmConsulta.executeQuery(laConsulta);

Ø Sentencias de consulta

Para obtener datos almacenados en la BD podemos realizar una consulta SQL (query). Podemos ejecutar la consulta utilizando el objeto Statement, pero ahora haciendo uso del método executeQuery al que le pasaremos una cadena con la consulta SQL. Los datos resultantes nos los devolverá como un objeto ResultSet.

- **ResultSet result = stmt.executeQuery(query);**

La consulta SQL nos devolverá una tabla, que tendrá una serie de campos y un conjunto de registros, cada uno de los cuales consistirá en una tupla de valores correspondientes a los campos de la tabla.

Los campos que tenga la tabla resultante dependerán de la consulta que hagamos, de los datos que solicitemos que nos devuelva. Por ejemplo, podemos solicitar que una consulta nos devuelva los campos expediente y nombre de los alumnos o bien que nos devuelva todos los campos de la tabla alumnos.

El objeto ResultSet dispone de un cursor que estará situado en el registro que podemos consultar en cada momento. Este cursor en un principio estará situado en una posición anterior al primer registro de la tabla. Podemos mover el cursor al siguiente registro con el método next del ResultSet. La llamada a este método nos devolverá true mientras pueda pasar al siguiente registro, y false en el caso de que ya estuviéramos en el último registro de la tabla. Para la consulta de todos los registros obtenidos utilizaremos normalmente un bucle como el siguiente:

```
while(result.next()) {

        // Leer registro

}
```

Hemos de tener en cuenta que el tipo del campo en la tabla debe ser convertible al tipo de datos Java solicitado. Para especificar el campo que queremos leer podremos utilizar bien su nombre en forma de cadena, o bien su índice que dependerá de la ordenación de los campos que devuelve la consulta. También debemos tener en cuenta que no podemos acceder al mismo campo dos veces seguidas en el mismo registro. Si lo hacemos nos dará una excepción.

Los tipos principales que podemos obtener son los siguientes:

getInt	Datos enteros
getDouble	Datos reales
getBoolean	Campos booleanos (si/no)
getString	Campos de texto
getDate	Tipo fecha (Devuelve **Date**)
getTime	Tipo hora (Devuelve **Time**)

- *Restricciones y movimientos en el ResultSet*

Cuando realizamos llamadas a BD de gran tamaño el resultado de la consulta puede ser demasiado grande y no deseable en términos de eficiencia y memoria. JDBC permite restringir el número de filas que se devolverán en el ResultSet. La clase **Statement incorpora dos métodos, getMaxRows y setMaxRows**, que permiten obtener e imponer dicha restricción. Por defecto, el límite es cero, indicando que no se impone la restricción.

Statement createStatement (int resultSetType, int resultSetConcurrency)

next	Pasa a la siguiente fila
previous	Ídem fila anterior
last	Ídem última fila
first	Ídem primera fila
absolute(int fila)	Pasa a la fila número fila
relative(int fila)	Pasa a la fila número fila desde la actual
getRow	Devuelve el número de fila actual
isLast	Devuelve si la fila actual es la última
isFirst	Ídem la primera

Los posibles valores que puede tener resultSetType son: ResultSet.TYPE_FORWARD_ONLY, ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.TYPE_SCROLL_SENSITIVE. El primer valor es el funcionamiento por defecto: el ResultSet sólo se mueve hacia adelante. Los dos siguientes permiten que el resultado sea arrastable. Una característica importante en los resultados arrastables es que los cambios que se produzcan en la BD se reflejan en el resultado, aunque dichos cambios se hayan producido después de la consulta. Esto dependerá de si el driver y/o la BD soporta este tipo de comportamiento. En el caso de INSENSITIVE, el resultado no es sensible a dichos cambios y en el caso de SENSITIVE, sí. Los métodos que podemos utilizar para movernos por el ResultSet son:

El otro parámetro, resultSetConcurrency, puede ser uno de estos dos valores: **ResultSet.CONCUR_READ_ONLY y ResultSet.CONCUR_UPDATABLE.** El primero es el utilizado por defecto y no permite actualizar el resultado. El segundo permite que los cambios realizados en el ResultSet se actualicen en la base de datos. Si queremos modificar los datos obtenidos en una consulta y queremos reflejar esos cambios en la BD debemos crear una sentencia con TYPE_FORWARD_SENSITIVE y CONCUR_UPDATABLE.

Ø Actualización de datos

Para actualizar un campo disponemos de métodos updateXXXX, de la misma forma que teníamos métodos getXXXX. Estos métodos reciben dos parámetros: el primero indica el nombre del campo (o número de orden dentro del ResultSet); el segundo indica el nuevo valor que tomará el campo del registro actual. Para que los cambios tengan efecto en la BD debemos llamar al método updateRow. El siguiente código es un ejemplo de modificación de datos:

```
rs.updateString("nombre","manolito");

rs.updateRow();
```

Ø Sentencias de actualización

La clase statement dispone de un método llamado executeUpdate el cual recibe como parámetro la cadena de caracteres que contiene la sentencia SQL a ejecutar. Este método únicamente permite realizar sentencias de actualización de la BD: creación de tablas (CREATE), inserción (INSERT), actualización (UPDATE) y borrado de datos (DELETE). El método a utilizar es el siguiente:

stmt.executeUpdate(sentencia);

Ø Otras llamadas a la BD

En la interfaz Statement podemos observar un tercer método que podemos utilizar para la ejecución de sentencias SQL. Hasta ahora hemos visto como para la ejecución de sentencias que devuelven datos (consultas) debemos usar executeQuery, mientras que para las sentencias INSERT, DELETE, UPDATE e instrucciones DDL utilizamos executeUpdate. Sin embargo, puede haber ocasiones en las que no conozcamos de antemano el tipo de la sentencia que vamos a utilizar (por ejemplo si la sentencia la introduce el usuario). En este caso podemos usar el método execute.

boolean hay_result = stmt.execute(sentencia);

Podemos ver que el método devuelve un valor booleano. Este valor será true si la sentencia ha devuelto resultados (uno o varios objetos ResultSet), y false en el caso de que sólo haya devuelto el número de registros afectados. Tras haber ejecutado la sentencia con el método anterior, para obtener estos datos devueltos proporciona una serie de métodos:

int n = stmt.getUpdateCount();

El método getUpdateCount nos devuelve el número de registros a los que afecta la actualización, inserción o borrado, al igual que el resultado que devolvía executeUpdate.

ResultSet rs = stmt.getResultSet();

El método getResultSet nos devolverá el objeto ResultSet que haya devuelto en el caso de ser una consulta, al igual que hacía executeQuery. Sin embargo, de esta forma nos permitirá además tener múltiples objetos ResultSet como resultado de una llamada. Eso puede ser necesario, por ejemplo, en el caso de una llamada a un procedimiento, que nos puede devolver varios resultados como veremos más adelante. Para movernos al siguiente ResultSet utilizaremos el siguiente método:

boolean hay_mas_results = stmt.getMoreResults();

La llamada a este método nos moverá al siguiente ResultSet devuelto, devolviéndonos true en el caso de que exista, y false en el caso de que no haya más resultados. Si existe, una vez nos hayamos movido podremos consultar el nuevo ResultSet llamando nuevamente al método getResultSet.

Otra llamada disponible es el método executeBatch. Este método nos permite enviar varias sentencias SQL a la vez. No puede contener sentencias SELECT. Devuelve un array de enteros que indicará el número de registros afectados por las sentencias SQL. Para añadir sentencias haremos uso del método addBatch. Un ejemplo de ejecución es el siguiente:

int[ ] res = stmt.executeBatch();

Por último, vamos a comentar el método getGeneratedKeys, también del objeto Statement. En muchas ocasiones hacemos inserciones en tablas cuyo identificador es un autonumérico. Por lo tanto, este valor no lo especificaremos nosotros manualmente, sino que se asignará de forma automática en la inserción. Sin embargo, muchas veces nos puede interesar conocer cual ha sido dicho identificador, para así por ejemplo poder insertar a continuación un registro de otra tabla que haga referencia al primero. Esto lo podremos hacer con el método getGeneratedKeys, que nos devuelve un ResultSet que contiene la clave generada:

5.CONCLUSIÓN

Como conclusión al presente proyecto presentado, mostramos un programa resolutor el problema explicado, para el que se realizaron investigaciones referentes a la estructura tanto del programa como de la base de datos implementada para ayudar a mejorar la eficiencia del programa.

Estas investigaciones llevadas al trabajo práctico de la codificación en base de datos se hicieron para tener un mejor control de datos que se ingresaran en esta, esto a su vez se creó para la facilidad del usuario al emplear el programa usando los diferentes JFrame, donde el usuario tiene la posibilidad de guardar, modificar, eliminar y ver todos los registros de las diversas entidades que emplea.

Es relevante mencionar la importancia de entender cómo funcionan y se aplican los procesos y métodos en el programa, estos son específicamente un conjunto de instrucciones que ejecutan una tarea determinada y que son llamadas en las diferentes clases creadas que hemos encapsulado en un formato estándar para que nos sea muy sencillo de manipular y utilizar, ejemplificando y haciendo más fácil su uso en el código fuente.

La aplicación permitió un análisis más detallado de la influencia de cada variable y método en los resultados mostrados, permite almacenar resultados y representar las variaciones de manera gráfica y sencilla de utilizar para el usuario.

Por último, mencionar una característica importante del trabajo en equipo que se llevó a cabo, el cual es establecer espacios de creatividad e innovación, que permita la participación activa y dinámica de los participantes, implementando un ambiente laboral de escucha mutua, sin tener en cuenta los niveles de jerarquía, valorando sin distinción todas las opiniones, facilitando la estructuración de todo el proyecto mutuo provocando más rendimiento en nuestras actividades, en pro del aprendizaje colectivo y la generación de resultados.

6. BIBLIOGRAFÍA

Alarcón, J. M. (13 de noviembre de 2019). *campusmvp*. Obtenido de campusmvp: https://www.campusmvp.es/recursos/post/los-conceptos-fundamentales-sobre-programacion-orientada-objetos-explicados-de-manera-simple.aspx

Aponte, L. E. (19 de junio de 2009). *programandoenjava*. Obtenido de programandoenjava: http://programandoenjava.over-blog.es/article-32829724.html

Caules, C. Á. (26 de mayo de 2014). *arquitecturajava*. Obtenido de arquitecturajava: https://www.arquitecturajava.com/java-iterator-vs-foreach/

CovanTec. (22 de junio de 2018). *readthedocs*. Obtenido de readthedocs: https://entrenamiento-python-basico.readthedocs.io/es/latest/leccion9/poo.html

Eguiluz, J. (15 de abril de 2014). *arkaitzgarro*. Obtenido de arkaitzgarro: https://www.arkaitzgarro.com/java/capitulo-13.html

Gómez, F. U. (18 de septiembre de 2013). *discoduroderoer*. Obtenido de discoduroderoer: https://www.discoduroderoer.es/estructuras-condicionales-en-java/#:~:text=Las%20estructuras%20condicionales%20nos%20permiten,usar%20operadores%20relacionales%20y%20condicionales

Kinsta. (8 de octubre de 2020). kinsta. Obtenido de kinsta: https://kinsta.com/es/base-de-conocimiento/que-es-github/

Roldán, Á. (10 de 9 de 2019). ciberaula. Obtenido de ciberaula: https://www.ciberaula.com/cursos/java/metodos_clase.php

ANDREW1234. (Desconocido de Desconocido de 2022). acervolima.com. Obtenido de acervolima.com: https://es.acervolima.com/java-swing-jpanel-con-ejemplos/#:~:text=JPanel%2C%20una%20parte%20del%20paquete,tiene%20una%20barra%20de%20t%C3%ADlo.

Berzal, F. (Desconocido de Desconocido de Desconocido). https://elvex.ugr.es/. Obtenido de https://elvex.ugr.es/: https://elvex.ugr.es/decsai/java/pdf/D1-swing.pdf

Desconocido. (07 de febrero de 2007). chuidiang.org. Obtenido de chuidiang.org: http://www.chuidiang.org/java/tablas/tablamodelo/tablamodelo.php

Fernández, Ó. B. (Desconocido de Desconocido de 2017). http://www3.uji.es/. Obtenido de http://www3.uji.es/: http://www3.uji.es/~belfern/Docencia/Presentaciones/ProgramacionAvanzada/Tema3/swing.html#1

P. (2021, 9 abril). Las clases y librerías básicas de Java para bases de datos relacionales. Blog Bitix. Recuperado 15 de mayo de 2022, de https://picodotdev.github.io/blog-bitix/2021/04/las-clases-y-librerias-basicas-de-java-para-bases-de-datos-relacionales/

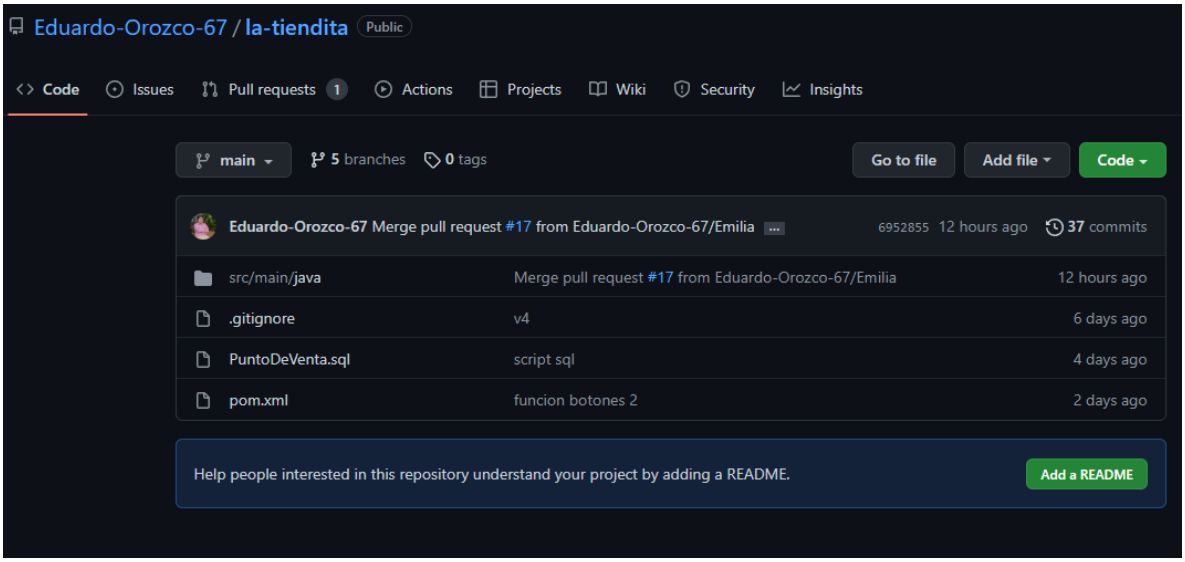
Clase Java DriverManager. (2019). © Copyright IBM Corp. 1998, 2014. Recuperado 15 de mayo de 2022, de https://www.ibm.com/docs/es/i/7.2?topic=connections-java-drivermanager-class

Gigena, M., Gigena, M., & Perfil, V. T. M. (2022, 16 mayo). JDBC: Conexión con Base de Datos. Todo JAVA. http://labojava.blogspot.com/2012/05/jdbc-conexion-con-base-de-datos.html

7. CRONOGRAMA DE ACTIVIDADES

ACTIVIDAD	RESPONSABLE (S)
Planeacion general del proyecto (ideas princiaples, soluciones...)	TODOS LOS INTEGRANTES
Creacion del Diagrama ER	José Eduardo Orozco Cárdenas
Script de la base de datos	Emilia Zuñiga Losada
Procedimientos almacenados	Emilia Zuñiga Losada
Diagrama de clases UML	José Eduardo Orozco Cárdenas
Diagrama de casos de uso	Pedro Octavio Culebro Prado
Conexion de Java y postgresql	José Eduardo Orozco Cárdenas
Setters y getters	Samuel Sánchez Guzman
Paquete para las clases de las acciones (modificar, eliminar...)	Emilia Zuñiga Losada
Paquete para las vistas (con todas las vistas del UML)	José Eduardo Orozco Cárdenas
Diseño de las ventanas	Pedro Octavio Culebro Prado
Consultas de SQL para las funciones	Samuel Sánchez Guzman
Funcionalidad de las ventanas para el cajero	Emilia Zuñiga Losada
Funcionalidad de las ventanas para el administrador	Pedro Octavio Culebro Prado José Eduardo Orozco Cárdenas Samuel Sánchez Guzman
Inventario terminado	Emilia Zuñiga Losada
Testeo	TODOS LOS INTEGRANTES

8. ANEXOS



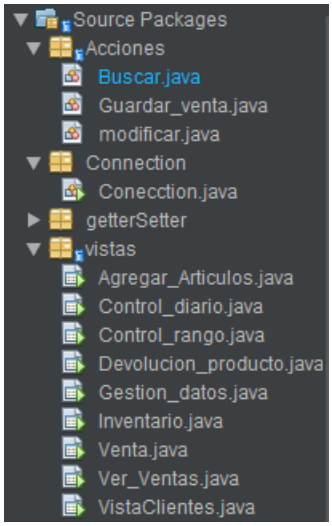
Captura del repositorio de GitHub

```
int idCombo
int barras
int cant =
```

Variables

```
Ver_Ventas verv = new Ver_Ventas();
Agregar_Articulos agre = new Agregar_Articulos();
Devolucion_producto devol = new Devolucion_producto();
ticket tick = new ticket();
```

Objetos



Clases por paquetes

```
public String[] controldiario(){
    String sql = "select SUM(com.monto_final) as Ventas_totales from compra_venta com WHERE com.fecha = CURRENT_DATE;";
    String []x=new String[3];
    try{
       st=conexion.createStatement();
       res=st.executeQuery(sql);
       while(res.next()){
           x[0]=res.getString("Ventas_totales");
       }
    }catch(Exception e){
       JOptionPane.showMessageDialog(null,e.getMessage() );
       System.out.println(e.getMessage());
    }finally{
        try {
            st.close();
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null,ex.getMessage() );
        }
    }
    return x;
}
```

Método que no recibe parámetros

```
public String[] controlrango(String fecha_inicio, String fecha_fin){
    String sql = "select SUM(com.monto_final) as Ventas_totales from compra_venta com WHERE com.fecha >= '"+fecha_inicio+"' AND com.fecha <= '"+fecha_fin+"'";
    String []x=new String[1];
    try{
       st=conexion.createStatement();
       res=st.executeQuery(sql);
       while(res.next()){
          x[0]=res.getString("Ventas_totales");
       }
    }catch(Exception e){
       JOptionPane.showMessageDialog(null,e.getMessage() );
       System.out.println(e.getMessage());
    }finally{
        try {
           st.close();
        } catch (SQLException ex) {
           JOptionPane.showMessageDialog(null,ex.getMessage() );
        }
    }
    return x;
}
```

Método que recibe parámetros

```
public static String getRfc() {
    return rfc;
}

public static void setRfc(String rfc) {
    getterSetter.rfc = rfc;
}
```

Getters y Setters

```
public Buscar(){
    Conecction c = new Conecction();
    conexion = c.conectar();
}
```

Constructores

```
public class Agregar_Articulos extends javax.swing.JFrame {
```

Herencia

```
if(pRes==0){
    JOptionPane.showMessageDialog
    limpiar_articulos();
}else if(pRes==1){
```

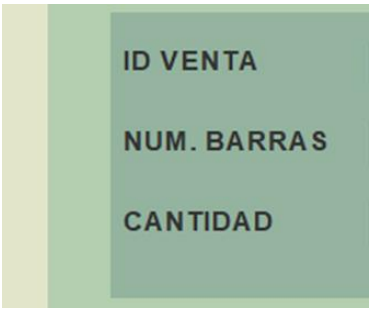
Condicional: If- else

```
for(String i:barras){
    comboventaAA.addItem(i);
}
```

Repetitivas: For

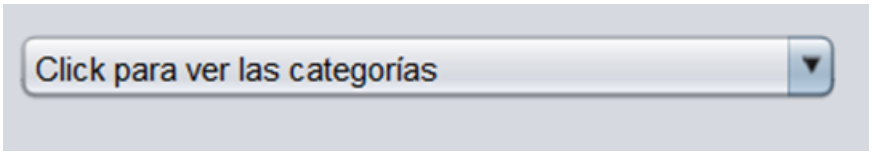
```
while(res.next()){
    x[filas]=res.getString("ID_cliente");
    filas++;
}
```

Repetitivas: While

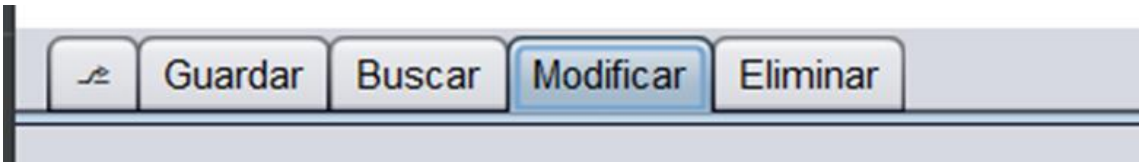


Ejemplo de JLabel

Ejemplo de JTextField

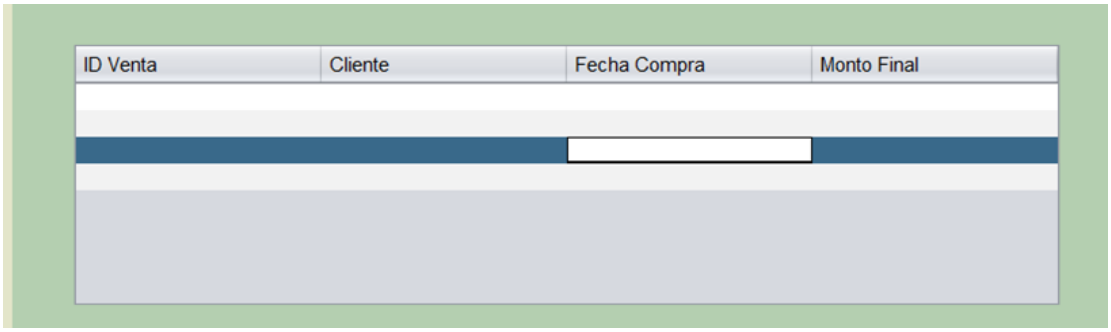


Ejemplo de JComboBox



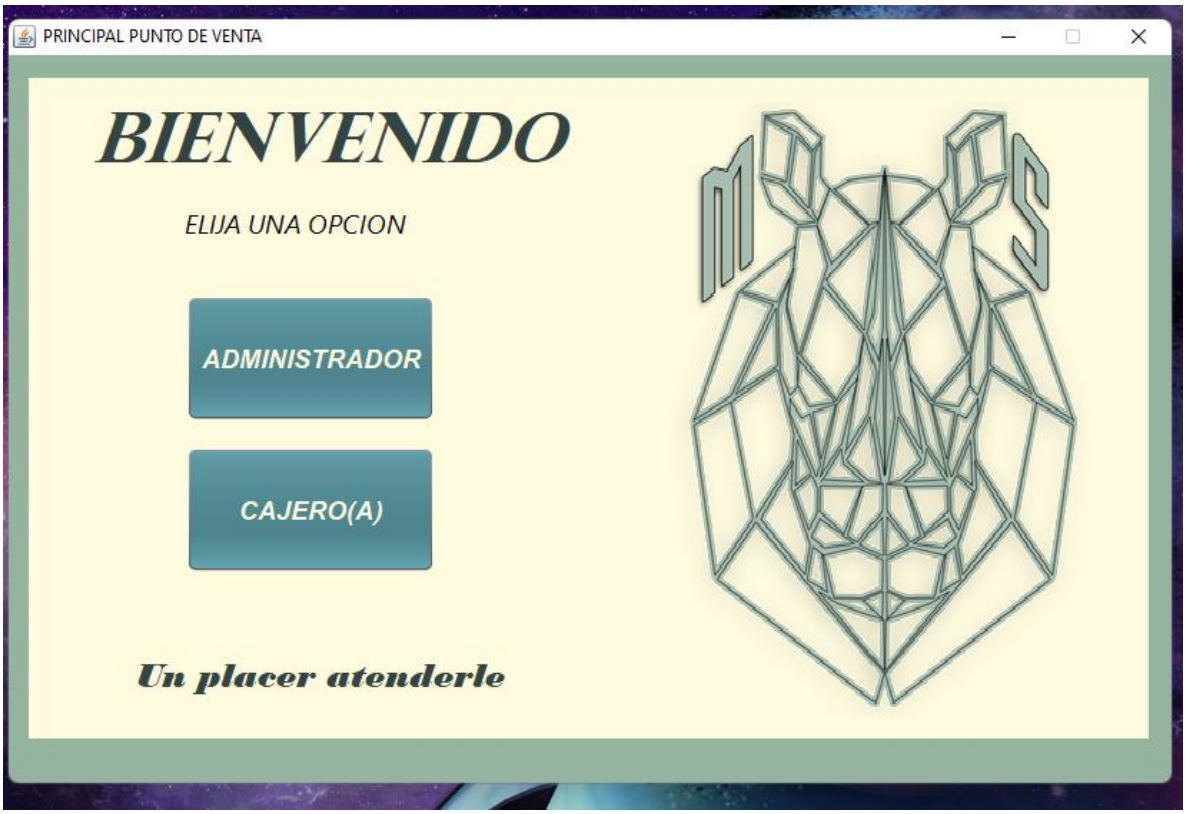
Ejemplo de JTabbedPane

Ejemplo de JPanel



Ejemplo de JTable

9. VENTANAS PRINCIPALES



Ventana para las ventas, ingresar productos y hacer devoluciones.

Acceder a la gestión de datos tanto de los clientes como el inventario

Ver el inventario de los productos y los reportes

Acceder a los inventarios y al control de ventas.